

NoSQL



BASIS DATA

Penulis

Dr. Tertia Avini, S.Kom., M.Kom | Yudi Herdiana, S.T., M.T

Agung Yuliyanto Nugroho, S.Kom., M.Kom., M.Par

Dr. Mohamad Ilyas Abas, S.Si, M.Kom | Tri Santoso, S.Kom, M.Kom

Ir. Hilmansyah Gani, S.Kom., M.Kom

Sulistianto Sutrisno Wanda, A.MI, S.Pd., M.M., M.Kom

Sutiyono, S.T., M.Kom | Widya Eka Pranata, S.Si., M.Si

Rizal Lamusu, S.Kom., M.T | Muhamad Ikmal Wiawan, M.Kom., IPP

Irawan Ibrahim, S.T., M.Kom | Syahrial, S.Kom., M.Kom



Editor : Weni Yuliani, S.Si., M.M., C.Ed

BASIS DATA

Penulis:

Dr. Tertia Avini, S.Kom., M.Kom; Yudi Herdiana, S.T., M.T;
Agung Yuliyanto Nugroho, S.Kom., M.Kom., M.Par;
Dr. Mohamad Ilyas Abas, S.Si, M.Kom; Tri Santoso, S.Kom, M.Kom;
Ir. Hilmansyah Gani, S.Kom., M.Kom;
Sulistianto Sutrisno Wanda, A.MI, S.Pd., M.M., M.Kom;
Sutiyono, S.T., M.Kom; Widya Eka Pranata, S.Si., M.Si;
Rizal Lamusu, S.Kom., M.T; Muhamad Ikmal Wiawan, M.Kom. IPP;
Irawan Ibrahim, S.T., M.Kom; Syahrial, S.Kom., M.Kom

Editor : Weni Yuliani, S.Si., M.M., C.Ed



BRAVO PRESS

CV BRAVO PRESS INDOENSIA

BASIS DATA

Penulis :

Dr. Tertia Avini, S.Kom., M.Kom; Yudi Herdiana, S.T., M.T;
Agung Yuliyanto Nugroho, S.Kom., M.Kom., M.Par;
Dr. Mohamad Ilyas Abas, S.Si, M.Kom;
Tri Santoso, S.Kom, M.Kom;
Ir. Hilmansyah Gani, S.Kom., M.Kom;
Sulistianto Sutrisno Wanda, A.MI, S.Pd, M.M, M.Kom;
Sutiyono, S.T., M.Kom; Widya Eka Pranata, S.Si., M.Si;
Rizal Lamusu., S.Kom., M.T;
Muhamad Ikmal Wiawan, M.Kom., IPP;
Irawan Ibrahim, S.T., M.Kom; Syahrial, S.Kom., M.Kom.

ISBN : 978-634-7109-84-2

Editor : Weni Yuliani, S.Si., M.M., C.Ed

Penyunting : Alda Anesa Vetdri, S.Pd

Desain Sampul dan Tata Letak : Septia Fakhira Risti S. Ds

Penerbit : CV BRAVO PRESS INDONESIA

Anggota IKAPI No.

Redaksi :

Perumahan Indah Harisanda blok f6 Jalan saudara RT 03/RW
06 Kel/Desa Tuah Madani, Kec. Tuah Madani, Kota Pekanbaru,
Riau

Website : www.bravopress.id

Email : bravopressindonesia@gmail.com

Cetakan pertama, Februari 2025

Hak cipta dilindungi undang-undang Dilarang memperbanyak
karya tulis ini dalam bentuk dan dengan cara apapun tanpa izin
tertulis dari penerbit.

KATA PENGANTAR

Dengan mengucapkan puji syukur kehadiran Allah SWT, atas limpahan rahmat dan hidayahNya, maka Penulisan Buku dengan judul basis data dapat diselesaikan. Buku tentang *basis data* biasanya mencakup berbagai topik yang berkaitan dengan desain, implementasi, manajemen, dan pengolahan data dalam sistem komputer.

Buku ini masih banyak kekurangan dalam penyusunannya. Oleh karena itu, kami sangat mengharapkan kritik dan saran demi perbaikan dan kesempurnaan buku ini selanjutnya. Kami mengucapkan terima kasih kepada berbagai pihak yang telah membantu dalam proses penyelesaian Buku ini. Semoga Buku ini dapat menjadi sumber referensi dan literatur yang mudah dipahami.

Tim Penulis

DAFTAR ISI

KATA PENGANTAR.....	i
DAFTAR ISI	ii
DAFTAR GAMBAR	vii
DAFTAR TABEL.....	x
BAB 1 KONSEP TEKNOLOGI BASIS DATA	1
Oleh : Dr. Tertia Avini, S.Kom., M.Kom	1
A. Pengantar Teknologi Basis Data	1
B. Tren Terkini dalam Teknologi Basis Data	7
C. Inovasi dalam Desain dan Pengelolaan Basis Data	17
D. Keamanan dan Privasi Data	24
E. Pengembangan Berkelanjutan dan DevOps untuk Basis Data	27
F. Tren Masa Depan Teknologi Basis Data.....	31
F. Tantangan dan Arah Masa Depan	36
G. Kesimpulan dari Konsep Teknologi Basis Data	40
DAFTAR PUSTAKA.....	44
BAB 2 ARSITEKTUR BASIS DATA DAN LINGKUNGAN SISTEM BASIS DATA.....	49
Oleh : Yudi Herdiana, S.T., M.T.	49
A. Pengantar.....	49

B. Arsitektur Basis Data	49
C. Lingkungan Sistem Basis Data	51
D. Contoh Kasus Sederhana.....	54
DAFTAR PUSTAKA.....	57
BAB 3 PEMODELAN DATA DENGAN ER DAN	
MENGGAMBARKANNYA DENGAN NOTASI BARKER....	63
Oleh : Agung Yuliyanto Nugroho S.Kom., M.Kom., M.Par	63
A. Konsep Dasar <i>Entity-Relationship</i> (ER)	63
B. Notasi Barker.....	77
DAFTAR PUSTAKA.....	82
BAB 4 RELASI LOGIKA <i>SUPERTYPE, SUBTYPE, INHERITANCE,</i>	
<i>GENERALIZATION, DAN SPECIALIZATION</i> DENGAN	
NOTASI BARKER.....	83
Oleh : Dr. Mohamad Ilyas Abas, S.Si, M.Kom	83
A. <i>Supertype</i> dan <i>Subtype</i>	84
B. <i>Inheritance</i>	98
C. <i>Generalization</i>	111
D. <i>Specialization</i>	119
E. Relasi Antar Konsep.....	126
F. Kesimpulan	133
DAFTAR PUSTAKA.....	139

**BAB 5 NORMALISASI DAN DENORMALISASI DATA PADA
SUATU TABEL DATA YANG TIDAK NORMAL..... 141**

Oleh : Tri Santoso, S.Kom, M.Kom 141

A. Definisi Normalisasi 141

B. Tahapan Normalisasi.....143

C. Definisi Denormalisasi 159

DAFTAR PUSTAKA..... 167

BAB 6 MENGHASILKAN *SCRIPT* UNTUK

MENGIMPLEMENTASIKAN DESAIN LOGIK KE DESAIN

FISIK BASIS DATA..... 169

Oleh : Ir. Hilmansyah Gani, S.Kom., M.Kom..... 169

A. Konsep Dasar Desain Logik dan Fisik Basis Data 170

B. Proses Transformasi Desain Logik ke Fisik.....173

C. Implementasi dengan SQL *Data Definition Language*
(DDL) 175

D. Optimisasi Desain Fisik 177

DAFTAR PUSTAKA..... 181

BAB 7 CARA MEMBUAT MODEL ER DENGAN METODE CHEN183

Oleh : Sulistianto Sutrisno Wanda, A.MI, S.Pd, M.M, M.Kom.. 183

A. Pengantar ke Model ER dan Metode Chen.....183

B. Komponen Dasar dalam Metode Chen 186

C. Normalisasi dengan Metode Chen.....197

D. Latihan dan Penilaian.....200

E. Kesimpulan	204
DAFTAR PUSTAKA.....	206
BAB 8 DATA DEFINITION LANGUAGE (DDL) DAN DATA MANIPULATION LANGUAGE (DML).....	207
Oleh : Sutiyono, S.T., M.Kom	207
A. Pengantar DDL dan DML	207
B. Konsep Dasar DDL (<i>Data Definition Language</i>).....	209
C. Konsep Dasar DML (<i>Data Manipulation Language</i>).....	221
D. Penutup.....	228
DAFTAR PUSTAKA.....	230
BAB 9 DATA QUERY LANGUAGE (DQL) DAN DATA CONTROL LANGUAGE (DCL).....	231
Oleh : Widya Eka Pranata, S.Si., M.Si.....	231
A. <i>Data Query Language</i> (DQL).....	231
B. <i>Data Control Language</i> (DCL)	239
DAFTAR PUSTAKA.....	246
BAB 10 TRANSACTION CONTROL LANGUAGE (TCL) DAN PROCEDURE AND FUNCTION.....	247
Oleh : Rizal Lamusu., S.Kom.,MT	247
A. Pengertian <i>Transaction Control Language</i> (TCL)	247
B. <i>Procedure</i> dan <i>Function</i>	251
C. Kesimpulan	259
DAFTAR PUSTAKA.....	260

BAB 11 INSTALASI SOFTWARE DBMS	261
Oleh : Muhamad Ikmal Wiawan, M.Kom., IPP	261
A. Pengantar SOFTWARE RDBMS.....	261
B. Instalasi <i>Relational Database Mysql Server</i> di Windows....	261
C. Instalasi <i>Relational Database Mysql Client</i> di Windows.....	273
DAFTAR PUSTAKA.....	281
BAB 12 PROSES <i>RECOVERY</i> DAN <i>DATABASE SECURITY</i> ...	283
Oleh : Irawan Ibrahim, S.T M.Kom	283
A. Proses <i>Recovery</i> dalam Database.....	284
B. Keamanan Database (<i>Database Security</i>).....	293
C. Hubungan Antara <i>Recovery</i> dan Keamanan Database	302
DAFTAR PUSTAKA.....	309
BAB 13 ARSITEKTUR BASIS DATA TERDISTRIBUSI	311
Oleh : Syahrial, S.Kom., M.Kom	311
A. Pengantar Basis Data Terdistribusi.....	311
B. Komponen Utama dalam Basis Data Terdistribusi	312
C. Jenis-Jenis Arsitektur Basis Data Terdistribusi	313
D. Desain Arsitektur Terdistribusi	316
E. Implementasi Arsitektur Terdistribusi	319
DAFTAR PUSTAKA.....	322
BIODATA PENULIS.....	323

DAFTAR GAMBAR

Gambar 3. 1 Gambar Entitas (Entity).....	64
Gambar 3. 2 Gambar Atribut	65
Gambar 3. 3 Gambar Relasi	66
Gambar 3. 4 Contoh ER Diagram.....	76
Gambar 3. 5 Contoh dengan Notasi Barker.....	80
Gambar 5. 1 Entitas dan Atribut.....	142
Gambar 5. 2 Field dan Record.....	143
Gambar 5. 3 Contoh Berkas.....	144
Gambar 7. 1 Entitas Independen.....	187
Gambar 7. 2 Entitas Lemah	188
Gambar 7. 3 Atribut Sederhana.....	188
Gambar 7. 4 Atribut Komposit.....	189
Gambar 7. 5 Atribut <i>Multivalued</i>	189
Gambar 7. 6 Atribut Derivatif.....	190
Gambar 7. 7 Atribut dengan <i>Primary Key</i>	190
Gambar 7. 8 Hubungan Unary.....	191
Gambar 7. 9 Hubungan Binary.....	191
Gambar 7. 10 Hubungan Ternary	192
Gambar 7. 11 Kardinalitas 1 to 1.....	192
Gambar 7. 12 Kardinalitas 1 to M	193

Gambar 7. 13 Kardinalitas M to N	193
Gambar 7. 14 Kardinalitas M to N + Atribut tambahan	194
Gambar 7. 15 Tampilan Lucidchart	195
Gambar 7. 16 Tampilan Draw.io.....	195
Gambar 7. 17 Tampilan Microsoft Visio	196
Gambar 7. 18 Tampilan Dbdiagram.io	196
Gambar 11. 1 Laman Awal situs <i>Downloader</i> Mysql	262
Gambar 11. 2 Laman Awal situs Downloader Mysql	263
Gambar 11. 3 Laman Awal situs <i>Downloader</i> Mysql	263
Gambar 11. 4 Laman Login / <i>Signup</i> (pendaftaran) akun oracle.	264
Gambar 11. 5 Link Download Mysql yang sudah tersedia	265
Gambar 11. 6 Pilihan Paket Instalasi Mysql.....	265
Gambar 11. 7 Pilihan Paket Instalasi Mysql.....	266
Gambar 11. 8 Pilihan Paket Instalasi Mysql.....	267
Gambar 11. 9 Metode Otentikasi Mysql Server.....	268
Gambar 11. 10 Laman Konfigurasi Akun dan Role.....	269
Gambar 11. 11 Laman Konfigurasi Akun dan Role.....	270
Gambar 11. 12 Konfigurasi <i>Server File Permissions</i>	271
Gambar 11. 13 Tampilan akhir konfigurasi dan instalasi mysql..	272
Gambar 11. 14 Pengujian hasil instalasi	272
Gambar 11. 15 Tampilan Mysql CLI.....	273
Gambar 11. 16 Pengujian hasil instalasi	275

Gambar 11. 17 <i>License Agreement</i> tahapan instalasi HeidiSQL	275
Gambar 11. 18 <i>Additional Tasks Instalation Process</i>	276
Gambar 11. 19 Halaman <i>Dialog Ready to Install</i>	276
Gambar 11. 20 Proses Instalasi yang sedang berlangsung.....	277
Gambar 11. 21 Hasil Akhir Proses Instalasi HeidiSQL.....	277
Gambar 11. 22 Tampilan Awal HeidiSQL	278
Gambar 11. 23 Tampilan Pembuatan Koneksi ke Mysql Server.	278
Gambar 11. 24 Konfigurasi Akhir Setup Koneksi DBMS heidiSQL.....	280
Gambar 11. 25 Tampilan Antarmuka HeidiSQL yang sudah terkoneksi ke Mysql Server	280

DAFTAR TABEL

Tabel 5. 1 Tabel Bentuk Tidak Normal	145
Tabel 5. 2 Tabel Bentuk Tidak Normal	146
Tabel 5. 3 Tabel Bentuk Normal Pertama	147
Tabel 5. 4 Ketergantungan Parsial	148
Tabel 5. 5 Tabel Penjualan	150
Tabel 5. 6 Tabel Detail Penjualan	150
Tabel 5. 7 Tabel Barang	150
Tabel 5. 8 Ketergantungan Transitif.....	152
Tabel 5. 9 Tabel Penjualan	153
Tabel 5. 10 Tabel Pelanggan.....	153
Tabel 5. 11 Tabel Detail Penjualan	153
Tabel 5. 12 Tabel Barang.....	154
Tabel 5. 13 Tabel Kategori.....	154
Tabel 5. 14 Tabel Mahasiswa	162
Tabel 5. 15 Tabel Anggota Perpustakaan	162
Tabel 5. 16 Tabel Data Mahasiswa.....	162
Tabel 5. 17 Tabel Mahasiswa	163
Tabel 5. 18 Tabel Peminjaman Buku	163
Tabel 5. 19 Tabel Peminjaman Buku	164
Tabel 5. 20 Tabel Peminjaman Buku	165

Tabel 5. 21 Tabel Detail Peminjaman	165
Tabel 5. 22 Tabel Peminjaman Buku.....	165
Tabel 7. 1 Tabel Data yang belum di Normalisasi.....	198
Tabel 7. 2 Normalisasi Tahap 1	199
Tabel 7. 3 Normalisasi Tahap 2	199
Tabel 7. 4 Normalisasi Tahap 3	200

BAB 1

KONSEP TEKNOLOGI BASIS DATA

Oleh : Dr. Tertia Avini, S.Kom., M.Kom

A. Pengantar Teknologi Basis Data

1. Definisi dan Ruang Lingkup Teknologi Basis Data

Basis data adalah kumpulan data yang terorganisasi secara sistematis sehingga dapat diakses, dikelola, dan diperbarui dengan efisien. Dalam pengertian sederhana, basis data berfungsi sebagai tempat penyimpanan data yang dirancang untuk memudahkan pengambilan dan manipulasi informasi secara terstruktur. Namun, konsep ini tidak hanya mencakup aspek penyimpanan, tetapi juga melibatkan proses pengelolaan data secara keseluruhan, termasuk integritas, keamanan, dan ketersediaan data.

Teknologi basis data mengacu pada alat, metode, dan praktik yang digunakan untuk menciptakan, memelihara, dan mengoptimalkan sistem basis data. Teknologi ini terus berkembang seiring dengan meningkatnya kebutuhan akan data yang cepat, akurat, dan dapat diandalkan dalam berbagai sektor, seperti bisnis, kesehatan, pendidikan, dan pemerintahan. Beberapa

elemen penting yang menjadi bagian dari teknologi basis data meliputi perangkat lunak manajemen basis data (DBMS), teknik desain basis data, arsitektur basis data, dan alat-alat untuk analisis data.

2. Definisi dan Ruang Lingkup Teknologi Basis Data

Ruang lingkup teknologi basis data sangat luas, mencakup berbagai aspek penting yang berperan dalam memastikan pengelolaan data yang optimal:

- a. **Desain Basis Data** Proses desain basis data melibatkan perencanaan struktur data yang akan disimpan, hubungan antar data, dan skema logis yang memastikan konsistensi dan efisiensi. Pendekatan seperti model Entitas-Relasi (ER) sering digunakan untuk mendesain basis data (Silberschatz et al., 2011).
- b. **Implementasi Basis Data** Implementasi mencakup proses pengaturan basis data di dalam sistem DBMS. Tahap ini melibatkan pembuatan tabel, hubungan antar tabel, serta definisi aturan dan batasan yang memastikan integritas data (Avini et al., 2024).
- c. **Pemeliharaan Basis Data** Pemeliharaan basis data mencakup aktivitas seperti pencadangan, pemulihan data, pembaruan perangkat lunak DBMS, dan pemantauan kinerja untuk

memastikan basis data tetap berfungsi secara optimal (Ashdown et al., 2021).

- d. Optimasi Basis Data Optimasi adalah proses meningkatkan kinerja basis data, termasuk kecepatan akses data, efisiensi penyimpanan, dan pengurangan latensi query. Teknik seperti normalisasi dan denormalisasi data sering diterapkan untuk meningkatkan efisiensi basis data (Silberschatz et al., 2011).
- e. Keamanan Basis Data Dalam era digital, keamanan data menjadi prioritas utama. Teknologi basis data mencakup mekanisme untuk melindungi data dari akses yang tidak sah, seperti enkripsi, kontrol akses berbasis peran (RBAC), dan audit log (VeraSafe, 2016) .
- f. Integrasi dan Interoperabilitas Basis data modern sering kali harus diintegrasikan dengan sistem lain, seperti aplikasi web, perangkat IoT, atau layanan cloud. Teknologi basis data mendukung integrasi ini melalui antarmuka API, middleware, dan protokol data seperti JSON atau XML (Silberschatz et al., 2011).
- g. Pengelolaan Big Data Dengan pertumbuhan data besar (big data), teknologi basis data mencakup alat dan teknik untuk mengelola data yang memiliki volume tinggi, kecepatan tinggi,

dan keragaman tipe data. Contoh alat yang sering digunakan adalah Apache Hadoop dan Apache Spark (Zikopoulos, 2012).

- h. Teknologi Basis Data Masa Depan Teknologi seperti basis data otonom dan basis data berbasis blockchain sedang menjadi tren karena menawarkan efisiensi tinggi dan keamanan yang lebih baik. Basis data otonom menggunakan kecerdasan buatan untuk mengelola dirinya sendiri, sementara blockchain menawarkan transparansi dan integritas data yang lebih tinggi (Ashdown et al., 2021; Lahoti & Singh, 2022).

Secara keseluruhan, teknologi basis data tidak hanya terbatas pada penyimpanan dan pengambilan data, tetapi juga mencakup semua aspek yang memungkinkan data digunakan secara maksimal untuk mendukung pengambilan keputusan, meningkatkan efisiensi operasional, dan memberikan nilai tambah bagi organisasi. Dengan perkembangan teknologi seperti cloud computing, big data, dan AI, ruang lingkup teknologi basis data terus berkembang, menjadikannya komponen vital dalam transformasi digital di berbagai sektor.

3. Peran Basis Data dalam Era Digital

Dalam era digital, basis data memainkan peran kunci sebagai fondasi untuk berbagai aplikasi teknologi modern, termasuk *e-commerce*, perbankan, kesehatan, pendidikan, dan media sosial.

Basis data memungkinkan organisasi untuk menyimpan, mengelola, dan memanfaatkan informasi dengan cara yang efektif untuk mendukung pengambilan keputusan berbasis data dan otomatisasi proses bisnis. Dengan adopsi teknologi big data, volume data yang dihasilkan oleh aktivitas manusia dan mesin telah meningkat secara eksponensial. Oleh karena itu, teknologi basis data modern dirancang untuk menangani data yang sangat besar, bervariasi, dan bergerak dengan kecepatan tinggi (*high velocity*) (Hartono et al., 2021).

Di sektor *e-commerce*, basis data digunakan untuk mengelola informasi pelanggan, inventaris, transaksi, dan preferensi belanja, yang memungkinkan pengalaman pengguna yang dipersonalisasi. Dalam perbankan, basis data membantu dalam menyimpan dan memproses transaksi keuangan secara real-time, mendeteksi penipuan, dan mematuhi peraturan keuangan. Di bidang kesehatan, basis data memungkinkan pengelolaan catatan medis elektronik (EMR) yang mendukung diagnosis berbasis data dan pengobatan yang lebih efisien.

Selain itu, basis data memainkan peran penting dalam analitik data, yang membantu organisasi memahami pola, tren, dan wawasan dari data mereka. Dengan alat seperti Apache Hadoop dan Spark, perusahaan dapat menganalisis data besar untuk

mendukung pengambilan keputusan strategis. Peran ini semakin diperkuat oleh integrasi teknologi kecerdasan buatan (AI) yang dapat mengoptimalkan kinerja basis data melalui prediksi pola penggunaan data dan pemeliharaan otomatis (Ashdown et al., 2021).

Keamanan juga menjadi aspek penting dalam peran basis data di era digital. Teknologi enkripsi data, kontrol akses berbasis peran, dan audit log memungkinkan organisasi untuk melindungi data sensitif dari ancaman keamanan siber. Standar seperti GDPR (*General Data Protection Regulation*) memastikan bahwa data pribadi dilindungi dengan kebijakan yang ketat, memengaruhi cara basis data dirancang dan dikelola (VeraSafe, 2016).

Dengan peran yang begitu luas dan vital, basis data tidak hanya menjadi alat teknis tetapi juga menjadi enabler utama dalam inovasi dan transformasi digital. Organisasi yang mampu memanfaatkan teknologi basis data secara efektif akan memiliki keunggulan kompetitif yang signifikan di pasar yang semakin didorong oleh data.

B. Tren Terkini dalam Teknologi Basis Data

1. Big Data dan Analitik

a. Pengertian Big Data

Big Data merujuk pada kumpulan data yang sangat besar, kompleks, dan beragam yang tidak dapat dikelola, diproses, atau dianalisis menggunakan alat atau teknik tradisional. Big Data umumnya memiliki tiga karakteristik utama, yang dikenal sebagai "3V": *Volume*, *Velocity*, dan *Variety*. *Volume* mengacu pada jumlah data yang sangat besar, yang bisa mencapai *petabytes* atau bahkan *exabytes*. *Velocity* mengacu pada kecepatan data dihasilkan dan perlu diproses, yang seringkali memerlukan pemrosesan secara *real-time* atau hampir *real-time*. *Variety* mencakup berbagai jenis data yang ada, baik terstruktur (misalnya, data dalam basis data relasional) maupun tidak terstruktur (misalnya, data teks, gambar, video, dan data sensor). Karena volume data yang sangat besar, Big Data memerlukan teknologi khusus seperti penyimpanan terdistribusi, pemrosesan paralel, dan teknik analisis lanjutan untuk mengungkap wawasan yang tersembunyi dalam data tersebut. Dalam dunia bisnis, pemanfaatan Big Data dapat memberikan keuntungan kompetitif dengan

meningkatkan pengambilan keputusan berbasis data, serta membantu organisasi dalam meramalkan tren dan memahami pola perilaku konsumen.

b. Alat dan Teknik untuk Analisis Big Data

Untuk menganalisis Big Data, teknologi dan alat khusus diperlukan untuk menangani volume, kecepatan, dan keragaman data yang ada. Salah satu alat yang paling terkenal adalah *Apache Hadoop*, sebuah kerangka kerja open-source yang memungkinkan penyimpanan dan pemrosesan data dalam skala besar secara terdistribusi. Hadoop menggunakan *Hadoop Distributed File System* (HDFS) untuk penyimpanan data secara terdistribusi dan MapReduce untuk pemrosesan data dalam bentuk batch secara paralel. Namun, Hadoop memerlukan waktu yang lebih lama dalam pemrosesan karena berbasis pada pemrosesan *batch*.

Di sisi lain, *Apache Spark* adalah alat lain yang populer untuk pemrosesan Big Data yang lebih cepat. *Spark* mendukung pemrosesan data in-memory dan pemrosesan secara *real-time*, yang memungkinkan analisis lebih cepat dan efisien. *Spark* juga kompatibel dengan Hadoop, dan sering digunakan untuk analisis yang membutuhkan latensi

rendah, seperti analisis *streaming* data dari sensor atau media sosial. Selain itu, *machine learning* dan *data mining* adalah teknik yang digunakan untuk menggali pola dan tren dari data yang besar. Algoritma *machine learning* seperti *regresi*, *clustering*, dan jaringan saraf digunakan untuk membuat prediksi, mengelompokkan data, atau mengenali pola tersembunyi (I. P. Putri et al., 2024).

c. Kasus Penggunaan dalam Bisnis

Big Data memberikan manfaat besar di berbagai sektor industri. Berikut adalah beberapa contoh penerapannya:

- 1) *E-commerce*: Big Data memungkinkan perusahaan e-commerce untuk menganalisis perilaku pelanggan dan memberikan rekomendasi produk yang lebih personal. Dengan menganalisis data transaksi, klik, dan riwayat pencarian pelanggan, perusahaan dapat menyesuaikan pengalaman belanja untuk meningkatkan konversi dan loyalitas pelanggan. Sistem rekomendasi berbasis Big Data telah terbukti meningkatkan pendapatan dengan menyarankan produk yang relevan kepada pengguna.
- 2) Kesehatan: Di sektor kesehatan, Big Data digunakan untuk menganalisis data pasien dan meningkatkan diagnosis serta perawatan. Dengan mengintegrasikan

data dari berbagai sumber seperti catatan medis elektronik, data sensor, dan gambar medis, tenaga medis dapat mendapatkan wawasan yang lebih mendalam tentang kondisi pasien. Misalnya, algoritma machine learning dapat digunakan untuk menganalisis gambar medis guna mendeteksi penyakit seperti kanker lebih awal dan lebih akurat.

- 3) Perbankan: Big Data digunakan untuk meningkatkan keamanan dan mendeteksi penipuan di sektor perbankan. Dengan menganalisis transaksi keuangan secara *real-time*, bank dapat mendeteksi pola transaksi yang tidak biasa atau mencurigakan. Ini membantu mencegah penipuan dan melindungi akun nasabah dengan cepat. Big Data juga digunakan untuk analisis kredit, di mana bank dapat menggunakan data alternatif untuk menilai kelayakan kredit nasabah. (Antoni et al., 2024).

2. Jenis-jenis NoSQL

NoSQL adalah jenis basis data yang tidak mengikuti model relasional tradisional dan dirancang untuk mengelola data yang sangat besar, terdistribusi, dan tidak terstruktur. Ada beberapa

jenis NoSQL, masing-masing dengan kegunaan yang berbeda berdasarkan jenis data dan kebutuhan aplikasi.

- a. Dokumen (*Document-Based*): Basis data jenis ini menyimpan data dalam format dokumen, yang umumnya menggunakan format JSON atau BSON (Binary JSON). MongoDB adalah contoh populer dari basis data dokumen. MongoDB memungkinkan penyimpanan data semi-terstruktur, seperti data dalam format JSON yang berisi pasangan *key-value*. Setiap dokumen dalam MongoDB bersifat *independen*, memungkinkan struktur yang fleksibel dan memungkinkan aplikasi untuk mengelola data dengan skema yang bervariasi. Basis data ini sangat cocok untuk aplikasi yang memerlukan fleksibilitas tinggi dalam pengelolaan data yang tidak selalu mengikuti format yang sama, seperti aplikasi web dan *mobile* yang membutuhkan kecepatan dalam pengambilan data yang terstruktur secara dinamis.
- b. Graf (*Graph-Based*): Basis data graf digunakan untuk menyimpan data yang berhubungan dengan entitas lain melalui hubungan yang kompleks. Neo4j adalah contoh dari basis data graf yang digunakan untuk aplikasi yang membutuhkan hubungan antar data yang intensif, seperti

sosial media, sistem rekomendasi, dan analisis jaringan. Dalam Neo4j, data disimpan dalam node yang terhubung oleh hubungan atau edges, memungkinkan aplikasi untuk melakukan pencarian dan traversal grafik dengan efisien. Basis data graf ideal digunakan untuk data yang melibatkan banyak hubungan kompleks antara entitas, seperti analisis jaringan sosial atau pengelolaan data biologi.

- c. *Key-Value (Key-Value Store)*: Basis data *key-value* menyimpan data sebagai pasangan *key-value*. Redis adalah contoh dari basis data jenis ini, dan sering digunakan untuk keperluan caching dan penyimpanan sementara. Redis sangat cepat dalam hal akses data, karena setiap elemen data dapat diakses dengan menggunakan *key* unik. *Key-value store* cocok untuk aplikasi yang memerlukan kecepatan tinggi dalam pengambilan data, seperti sistem real-time yang membutuhkan latensi rendah, misalnya dalam pengelolaan sesi pengguna atau pengaturan konfigurasi aplikasi.

1) Kapan Menggunakan NoSQL Dibandingkan SQL

Pemilihan antara NoSQL dan SQL tergantung pada kebutuhan aplikasi dan karakteristik data yang akan disimpan. Berikut adalah beberapa situasi di mana NoSQL lebih cocok dibandingkan SQL:

- a) **Data Tidak Memiliki Skema Tetap:** Jika data yang akan disimpan tidak memiliki skema yang tetap atau berubah-ubah, NoSQL adalah pilihan yang lebih baik. SQL mengharuskan data disimpan dalam tabel dengan struktur yang jelas dan skema yang tetap, yang tidak selalu sesuai dengan kebutuhan aplikasi modern yang bekerja dengan data semi-terstruktur atau tidak terstruktur. Basis data NoSQL seperti MongoDB memungkinkan penyimpanan data dalam format dokumen yang lebih fleksibel, sehingga perubahan skema atau penambahan kolom baru tidak memerlukan perubahan besar pada *database*.
- b) **Skalabilitas Horizontal:** NoSQL dirancang untuk skala horizontal, yang berarti data dapat didistribusikan di banyak server atau node secara efisien. Jika aplikasi memerlukan penyimpanan data yang sangat besar dan harus dapat menangani beban lalu lintas yang tinggi, NoSQL lebih cocok. Basis data NoSQL seperti Cassandra dan MongoDB memungkinkan data untuk dibagi ke beberapa server, meningkatkan kemampuan skalabilitas tanpa perlu memperbesar satu server saja. Hal ini berbeda dengan SQL yang lebih sering mengandalkan skalabilitas vertikal, yaitu menambah daya komputasi pada satu server.

- c) Kebutuhan Latensi Rendah: Aplikasi yang membutuhkan latensi rendah dan kecepatan tinggi dalam akses data, seperti aplikasi *real-time* atau sistem caching, akan lebih cocok menggunakan NoSQL. Misalnya, Redis, sebagai key-value store, dirancang untuk memberikan akses data yang sangat cepat, yang sangat penting dalam aplikasi yang membutuhkan respons cepat seperti analisis data streaming atau sistem rekomendasi produk. Basis data SQL, meskipun efisien dalam transaksi yang kompleks, cenderung memiliki overhead lebih tinggi dalam operasi yang memerlukan akses cepat.
- d) Pengolahan Data dengan Hubungan Kompleks: Jika aplikasi membutuhkan pengolahan data yang melibatkan banyak hubungan antar entitas, seperti aplikasi media sosial, analisis jaringan, atau rekomendasi berbasis hubungan, basis data graf seperti Neo4j lebih sesuai. Dalam situasi ini, SQL tidak seefisien dalam menangani data yang sangat terhubung dan memiliki relasi yang rumit.

3. Integrasi AI dalam Pengelolaan Basis Data

Integrasi Kecerdasan Buatan (AI) dalam pengelolaan basis data semakin penting dalam era digital yang menghasilkan volume data yang sangat besar. AI dapat digunakan untuk mengotomatiskan berbagai aspek pengelolaan basis data, mulai

dari pemeliharaan hingga optimasi performa sistem. Salah satu aplikasi utama AI dalam pengelolaan basis data adalah optimasi performa *query*. AI dapat menganalisis pola-pola dalam eksekusi *query* dan memberikan rekomendasi untuk meningkatkan waktu eksekusi atau mengurangi beban server. Teknologi *machine learning* dapat digunakan untuk mempelajari pola *query* yang sering digunakan dan melakukan penyesuaian otomatis, seperti indeksasi yang lebih efisien atau perubahan dalam struktur *database*.

Selain itu, AI juga berperan dalam pemantauan kinerja sistem basis data. Dengan menggunakan teknik pembelajaran mesin, sistem dapat mendeteksi anomali atau potensi masalah dalam performa *database* secara otomatis, seperti lonjakan penggunaan CPU, memori, atau penurunan kecepatan akses data. Pemantauan ini memungkinkan administrator basis data untuk melakukan perbaikan proaktif sebelum masalah berkembang menjadi isu besar. AI dapat juga digunakan untuk melakukan manajemen sumber daya otomatis dengan menyesuaikan alokasi sumber daya seperti penyimpanan dan komputasi berdasarkan kebutuhan data yang terdeteksi.

Contoh lainnya adalah penerapan AI dalam pembersihan data (*data cleaning*). AI dapat digunakan untuk mendeteksi data

duplikat, kesalahan format, atau inkonsistensi data yang bisa mengganggu kualitas basis data. Dengan algoritma pembelajaran mesin, sistem dapat memperbaiki data secara otomatis, meningkatkan kualitas dan akurasi data dalam basis data.

4. Contoh Aplikasi Pembelajaran Mesin untuk Optimasi *Query*

Pembelajaran mesin (*machine learning*) telah menunjukkan potensi yang besar dalam mengoptimalkan query dalam basis data, salah satunya adalah melalui penggunaan *Oracle Autonomous Database*. *Oracle Autonomous Database* adalah salah satu contoh aplikasi basis data yang memanfaatkan *machine learning* untuk deteksi dan perbaikan otomatis *query*. Sistem ini menggunakan pembelajaran mesin untuk menganalisis query-query yang sering dijalankan oleh pengguna, kemudian mengidentifikasi pola-pola yang dapat menyebabkan inefisiensi atau pemborosan sumber daya.

Sistem ini dapat melakukan beberapa perbaikan otomatis pada query, seperti indeksasi otomatis atau perubahan struktur *query* yang lebih efisien. Misalnya, jika ada *query* yang selalu mengakses tabel besar tanpa filter yang sesuai, *Oracle Autonomous Database* dapat membuat indeks otomatis atau merekomendasikan penggunaan filter untuk meningkatkan

performa. Dengan cara ini, sistem dapat memperbaiki masalah query tanpa campur tangan manual dari DBA (*Database Administrator*), yang dapat menghemat waktu dan biaya operasional. Selain itu, *Oracle Autonomous Database* dapat memantau kinerja query secara real-time menggunakan AI untuk mendeteksi adanya masalah seperti latensi tinggi atau *bottleneck* dalam eksekusi *query*. Begitu masalah terdeteksi, sistem akan memberikan saran atau bahkan memperbaiki *query* secara otomatis untuk meningkatkan efisiensi.

Dengan menggunakan pembelajaran mesin, *Oracle Autonomous Database* dapat melakukan *self-tuning* atau penyesuaian otomatis untuk memastikan bahwa *query* yang dijalankan selalu optimal tanpa perlu campur tangan manusia. Hal ini sangat menguntungkan untuk aplikasi dengan beban kerja yang tinggi dan kompleks, di mana optimasi manual mungkin tidak selalu cukup cepat atau efektif.

C. Inovasi dalam Desain dan Pengelolaan Basis Data

1. *Database Multitenant*

a. Konsep dan Manfaat untuk SaaS

Multitenancy adalah arsitektur basis data di mana beberapa pengguna, atau yang biasa disebut sebagai *tenant*, berbagi satu instance basis data secara bersamaan.

Meskipun menggunakan basis data yang sama, data masing-masing tenant tetap terisolasi dan aman. Model ini sangat penting dalam konteks *Software as a Service* (SaaS), di mana aplikasi di-host dan disediakan untuk banyak pengguna dengan kebutuhan yang berbeda-beda namun tetap menggunakan infrastruktur yang sama. Keuntungan utama dari model *multitenancy* adalah efisiensi sumber daya. Karena banyak tenant berbagi sumber daya yang sama, biaya operasional seperti penyimpanan dan komputasi dapat dibagi di antara banyak pengguna, yang mengurangi biaya infrastruktur. Selain itu, penyedia layanan dapat mengelola satu *instance* aplikasi dan basis data, yang menyederhanakan pemeliharaan dan pembaruan perangkat lunak. Dengan menggunakan model ini, penyedia SaaS dapat dengan mudah menangani banyak pelanggan tanpa perlu menyediakan sumber daya terpisah untuk setiap pengguna.

Multitenancy juga memberikan fleksibilitas yang sangat dibutuhkan oleh model SaaS. Pengguna atau tenant dapat memiliki pengaturan dan konfigurasi aplikasi mereka sendiri, meskipun mereka berbagi instance yang sama. Ini memungkinkan personalisasi tingkat tinggi tanpa mempengaruhi tenant lainnya. Misalnya, dalam aplikasi

SaaS berbasis CRM, setiap perusahaan (*tenant*) dapat mengonfigurasi dasbor dan pengaturan sesuai kebutuhan, sementara penyedia layanan tetap dapat mengelola satu sistem. Namun, meskipun *multitenancy* menawarkan efisiensi, pengelolaan isolasi data dan keamanan antar tenant menjadi tantangan yang memerlukan perhatian serius.

b. Tantangan dalam Manajemen *Multitenancy*

Manajemen *multitenancy* memiliki beberapa tantangan utama yang perlu diatasi untuk memastikan bahwa sistem dapat berjalan dengan lancar dan aman meskipun banyak tenant berbagi sumber daya yang sama. Tiga tantangan utama yang sering dihadapi dalam manajemen *multitenancy* adalah isolasi data antar tenant, pengelolaan keamanan, dan skala.

1) *Isolasi Data Antar Tenant*: Salah satu tantangan terbesar dalam *multitenancy* adalah isolasi data antar tenant. Meskipun tenant berbagi instance basis data yang sama, sangat penting untuk memastikan bahwa data setiap tenant terisolasi dengan baik. Tanpa isolasi yang tepat, ada potensi risiko di mana data dari satu tenant dapat diakses oleh tenant lain. Solusi untuk

masalah ini termasuk penggunaan teknik seperti partisi data atau kolom terpisah untuk setiap tenant, serta kontrol akses yang ketat untuk memastikan hanya pengguna yang berwenang yang dapat mengakses data mereka sendiri. Sistem kontrol akses berbasis peran (RBAC) dan enkripsi data juga sering digunakan untuk meningkatkan keamanan data antar tenant.

- 2) **Pengelolaan Keamanan:** Keamanan menjadi lebih kompleks dalam lingkungan *multitenancy* karena beberapa tenant berbagi sumber daya yang sama, baik dalam hal komputasi, penyimpanan, maupun jaringan. Pengelolaan autentikasi dan otorisasi yang baik sangat penting untuk melindungi data dari potensi akses tidak sah. Selain itu, adanya perbedaan tingkat keamanan antara tenant juga perlu diperhatikan, misalnya dalam aplikasi SaaS yang memiliki tenant dengan kebutuhan keamanan yang berbeda-beda. Oleh karena itu, penting untuk menggunakan enkripsi data, VPN untuk komunikasi aman, serta audit keamanan secara berkala untuk memastikan bahwa data dan aplikasi tetap aman.
- 3) **Skalabilitas:** Skalabilitas dalam model *multitenancy* bisa menjadi tantangan, terutama ketika jumlah tenant meningkat secara signifikan. Sistem harus dapat

menangani beban kerja yang tinggi tanpa mengorbankan kinerja. Skalabilitas horizontal, yang memungkinkan menambahkan lebih banyak server atau sumber daya untuk menangani lonjakan permintaan, sering diterapkan dalam model multitenancy. Namun, memastikan bahwa sumber daya dikelola dengan efisien untuk menghindari pemborosan adalah tantangan lain yang perlu diatasi.

2. Prinsip Dasar *Blockchain*

Blockchain adalah teknologi ledger terdistribusi yang menyimpan data dalam bentuk blok yang terhubung secara kriptografis. Setiap blok dalam *blockchain* mengandung data transaksi dan hash yang terhubung ke blok sebelumnya, menciptakan struktur berantai yang aman dan tidak dapat diubah. Prinsip dasar *blockchain* adalah desentralisasi, yang berarti data tidak disimpan pada satu server pusat, tetapi didistribusikan di seluruh jaringan komputer yang terhubung, yang dikenal sebagai node. Setiap node dalam jaringan memiliki salinan dari *blockchain* yang diperbarui secara otomatis setiap kali transaksi baru ditambahkan.

Blockchain menggunakan prinsip kriptografi untuk menjaga integritas dan keamanan data. Setiap blok dilengkapi dengan *hash* kriptografi, yang merupakan representasi digital dari data dalam

blok tersebut. Setiap transaksi yang dicatat dalam blockchain juga dienkripsi untuk memastikan bahwa data tersebut tidak dapat diubah tanpa mempengaruhi seluruh struktur blockchain. Hal ini membuat *blockchain* sangat aman terhadap manipulasi atau pemalsuan data.

Selain itu, blockchain beroperasi dengan konsensus antara node-node dalam jaringan. Setiap transaksi yang terjadi harus diverifikasi oleh mayoritas node, yang menjamin bahwa tidak ada satu pihak pun yang bisa mengubah data secara sepihak. Ini juga berarti bahwa blockchain bersifat transparan, karena setiap transaksi dapat dilihat oleh semua pihak yang terlibat, namun tetap aman dan terlindungi dari perubahan.

a. Aplikasi *Blockchain* untuk Keamanan Data

Blockchain memiliki aplikasi yang luas dalam keamanan data, terutama dalam audit trail dan pengelolaan identitas digital. Keunggulan utama blockchain dalam konteks ini adalah transparansi, keamanan, dan integritas data yang sangat tinggi.

- 1) Audit Trail: *Blockchain* memungkinkan pencatatan transaksi atau perubahan data secara transparan dan aman. Setiap transaksi yang terjadi dalam jaringan *blockchain* dicatat dalam blok yang tidak dapat diubah setelah ditambahkan, menciptakan audit trail yang permanen. Hal ini sangat

berguna dalam industri yang memerlukan pelacakan perubahan data secara detail, seperti dalam sektor keuangan atau pemerintahan. Setiap transaksi dapat diaudit untuk memastikan keaslian dan integritasnya, dan karena blockchain bersifat terbuka, audit dapat dilakukan oleh semua pihak yang berwenang tanpa memerlukan pihak ketiga yang dapat memanipulasi data.

- 2) Pengelolaan Identitas Digital: *Blockchain* juga digunakan untuk mengelola identitas digital dengan cara yang lebih aman dan terdesentralisasi. Dalam sistem tradisional, data identitas seperti nama, alamat, atau nomor identifikasi sering disimpan di pusat data yang dapat menjadi target serangan. *Blockchain* memungkinkan pengguna untuk memiliki kendali penuh atas data identitas mereka dan membagikannya hanya ketika diperlukan. Setiap perubahan atau pembaruan data identitas dapat dicatat dalam *blockchain*, memastikan bahwa informasi tersebut aman dan tidak dapat dimanipulasi oleh pihak yang tidak berwenang. Hal ini sangat penting dalam pengelolaan identitas di sektor yang memerlukan tingkat keamanan tinggi, seperti dalam layanan keuangan atau medis.

D. Keamanan dan Privasi Data

1. Teknik Keamanan Modern

Keamanan data adalah salah satu aspek yang sangat penting dalam dunia teknologi informasi saat ini, terutama dengan meningkatnya ancaman terhadap data sensitif dan privasi pengguna. Untuk melindungi data, berbagai teknik keamanan modern diterapkan, dua di antaranya yang paling umum adalah enkripsi dan RBAC (*Role-Based Access Control*).

- a. Enkripsi: Enkripsi adalah teknik untuk mengubah data asli menjadi format yang tidak dapat dibaca tanpa kunci dekripsi yang sah. Teknik ini memastikan bahwa bahkan jika data berhasil diakses oleh pihak yang tidak berwenang, mereka tidak dapat mengartikan atau menggunakan data tersebut. Ada berbagai jenis enkripsi yang digunakan, seperti enkripsi simetris di mana kunci yang sama digunakan untuk enkripsi dan dekripsi, dan enkripsi asimetris yang menggunakan pasangan kunci publik dan privat. Dalam konteks basis data, enkripsi dapat diterapkan pada data yang disimpan (enkripsi *data at rest*) dan data yang sedang ditransfer (enkripsi *data in transit*) untuk memastikan bahwa data tetap aman selama seluruh siklus hidupnya. Penggunaan algoritma enkripsi yang kuat, seperti

AES (*Advanced Encryption Standard*), adalah cara yang umum diterapkan untuk melindungi data dari akses tidak sah.

- b. RBAC (*Role-Based Access Control*): RBAC adalah model kontrol akses yang membatasi akses ke data berdasarkan peran yang dimiliki oleh pengguna dalam organisasi. Dalam RBAC, hak akses diberikan berdasarkan peran tertentu yang dimiliki oleh pengguna, seperti admin, manajer, atau karyawan biasa. Setiap peran memiliki tingkat akses yang berbeda ke data, sehingga pengguna hanya dapat mengakses data yang diperlukan untuk melaksanakan tugas mereka. Hal ini mengurangi risiko akses data yang tidak sah dan meminimalkan potensi kebocoran data. RBAC membantu organisasi untuk memastikan bahwa hanya orang yang berwenang yang dapat mengakses informasi sensitif, dan juga memudahkan dalam pengelolaan hak akses, karena perubahan dalam peran tidak memerlukan perubahan langsung pada pengaturan akses pengguna individual.

2. Perlindungan Privasi dalam Basis Data

Dengan semakin ketatnya regulasi yang mengatur privasi dan perlindungan data, seperti *General Data Protection Regulation* (GDPR) di Eropa, organisasi kini dituntut untuk tidak hanya melindungi data tetapi juga memastikan bahwa data yang mereka simpan atau kelola digunakan dengan cara yang transparan dan

aman. Perlindungan privasi dalam basis data mencakup beberapa teknik yang dapat membantu organisasi mematuhi regulasi yang ada, sekaligus menjaga data agar tetap aman dan terlindungi.

- a. **Anonimisasi Data:** Anonimisasi adalah teknik untuk menghapus informasi yang dapat mengidentifikasi individu dalam dataset, dengan tujuan agar data tersebut tidak dapat dikaitkan kembali dengan individu tertentu meskipun jatuh ke tangan yang salah. Dalam konteks basis data, ini bisa dilakukan dengan mengganti atau menghapus data yang mengidentifikasi orang, seperti nama atau nomor identifikasi, tanpa mengubah nilai informasi lainnya yang tetap berguna untuk analisis. Teknik ini sangat berguna dalam konteks penelitian atau pengolahan data besar, di mana organisasi tetap dapat memperoleh wawasan dari data tanpa melanggar privasi pengguna.
- b. **Penghapusan Data Secara Aman:** Regulasi seperti GDPR mewajibkan organisasi untuk menghapus data pribadi yang tidak lagi diperlukan untuk tujuan yang sah atau ketika diminta oleh individu yang bersangkutan. Oleh karena itu, teknik penghapusan data secara aman menjadi sangat penting. Penghapusan data yang aman melibatkan proses untuk memastikan bahwa data yang dihapus tidak dapat dipulihkan atau diakses kembali setelah dihapus. Ini termasuk teknik seperti *overwriting*, yang menulis data baru di atas data lama

untuk mencegah pemulihan, serta menggunakan perangkat lunak yang dirancang untuk menghancurkan data secara permanen.

- c. **Keamanan Data Sensitif:** Dalam desain basis data, perlindungan privasi juga berarti memastikan bahwa data sensitif, seperti informasi kartu kredit, nomor jaminan sosial, atau catatan medis, diperlakukan dengan hati-hati. Teknik ini termasuk penerapan enkripsi untuk data sensitif, penggunaan tokenisasi untuk menggantikan data sensitif dengan data yang tidak dapat digunakan kembali, serta penerapan audit trail untuk melacak siapa yang mengakses data dan kapan.

Regulasi yang terus berkembang, seperti GDPR, semakin mendesak perusahaan untuk tidak hanya melindungi data mereka dengan cara yang tradisional tetapi juga memastikan bahwa data diproses dan dikelola dengan cara yang menghormati hak privasi individu.

E. Pengembangan Berkelanjutan dan DevOps untuk Basis Data

1. Praktik CI/CD untuk Database

Integrasi Berkelanjutan (CI) dan Pengiriman Berkelanjutan (CD) adalah dua konsep penting dalam pengembangan perangkat lunak

modern yang juga diterapkan dalam manajemen basis data. Praktik ini memungkinkan pembaruan basis data dilakukan secara otomatis dan lebih efisien tanpa mengganggu operasional yang sedang berlangsung. Dengan menggunakan CI/CD, tim pengembang dapat secara otomatis menguji, membangun, dan mengirimkan perubahan kode ke basis data, memastikan bahwa pembaruan dilakukan secara konsisten dan tanpa kesalahan.

- a. Integrasi Berkelanjutan (CI): CI dalam konteks basis data mengacu pada proses otomatisasi pengujian dan penggabungan perubahan yang dilakukan pada skema basis data atau data itu sendiri ke dalam repositori utama. Setiap kali perubahan diterapkan pada basis data, baik itu perubahan struktur tabel, pengindeksan, atau prosedur yang disimpan, perubahan tersebut harus diuji dan diuji kembali untuk memastikan bahwa tidak ada masalah integritas atau kinerja. Alat seperti Jenkins dan GitLab CI dapat digunakan untuk membangun pipeline CI yang memungkinkan pengujian otomatis setiap kali ada perubahan pada skema atau data, mengurangi kemungkinan kesalahan manusia dan meningkatkan keandalan aplikasi yang menggunakan basis data tersebut.
- b. Pengiriman Berkelanjutan (CD): Pengiriman berkelanjutan untuk basis data melibatkan penerapan otomatis pembaruan skema atau data yang telah lulus tes ke lingkungan produksi.

Dengan CD, perubahan basis data yang telah diuji dan disetujui akan diterapkan secara otomatis ke server produksi tanpa intervensi manual. Ini mempercepat siklus pengembangan dan pengujian, mengurangi potensi *downtime*, serta meminimalkan risiko yang timbul dari penerapan pembaruan yang tidak tepat. Proses ini juga memungkinkan pengembang untuk melakukan pembaruan kecil dan sering, mengurangi risiko perubahan besar yang dapat merusak sistem produksi.

Dengan menerapkan CI/CD untuk basis data, pengembang dapat lebih cepat merilis pembaruan, memperbaiki bug, dan menambahkan fitur baru tanpa mengorbankan stabilitas aplikasi atau basis data.

2. Alat Otomatisasi dalam Manajemen Basis Data

Pengelolaan dan pemeliharaan basis data yang efisien membutuhkan otomatisasi, terutama saat bekerja dengan basis data dalam pengembangan perangkat lunak yang terus berkembang. Alat otomatisasi seperti *Flyway* dan *Liquibase* memberikan solusi untuk mengelola migrasi basis data secara otomatis, mengurangi waktu yang diperlukan untuk merilis pembaruan, serta menghindari kesalahan yang sering terjadi dalam prosedur manual.

- a. *Flyway*: *Flyway* adalah alat migrasi basis data yang dirancang untuk memfasilitasi penerapan perubahan skema database secara otomatis. *Flyway* bekerja dengan menyimpan perubahan skema dalam file skrip SQL terstruktur yang disebut migrasi, yang dapat dijalankan secara berurutan untuk memperbarui basis data. *Flyway* mendukung berbagai jenis basis data dan dapat digunakan dalam sistem CI/CD untuk mengotomatisasi pembaruan skema secara teratur. Setiap migrasi yang diterapkan disertai dengan informasi tentang versi, pengarang, dan deskripsi perubahan, yang memungkinkan tim untuk melacak perubahan yang dilakukan pada basis data. Keunggulan *Flyway* adalah kemampuannya untuk menangani perubahan basis data dengan cara yang sangat ringan dan mudah diintegrasikan dalam *pipeline* pengembangan.
- b. *Liquibase*: *Liquibase* adalah alat migrasi basis data yang lebih fleksibel dan kuat dibandingkan *Flyway*. Dengan *Liquibase*, perubahan basis data disimpan dalam file XML, YAML, atau JSON, yang mendeskripsikan perubahan yang perlu diterapkan pada skema. Keunggulan *Liquibase* terletak pada kemampuannya untuk mendukung lebih banyak fitur seperti *rollback*, yaitu kemampuan untuk membatalkan migrasi yang telah diterapkan jika diperlukan. Hal ini memberi fleksibilitas lebih dalam pengelolaan perubahan yang mungkin salah atau

menyebabkan masalah. Liquibase juga mendukung banyak jenis basis data dan memungkinkan pengembang untuk memverifikasi dan membandingkan versi basis data yang berbeda, yang sangat berguna dalam tim pengembangan besar.

Alat-alat ini meningkatkan efisiensi siklus pengembangan perangkat lunak dengan memastikan bahwa basis data selalu diperbarui dengan cara yang konsisten dan aman, sekaligus mengurangi potensi kesalahan manusia. Penggunaan otomatisasi dalam pengelolaan basis data memudahkan pembaruan basis data pada lingkungan produksi yang besar dan kompleks, mengurangi biaya operasional dan meminimalkan risiko *downtime*.

F. Tren Masa Depan Teknologi Basis Data

1. Pemrosesan Data *Real-Time*

Pemrosesan data *real-time* adalah kemampuan untuk menganalisis dan memproses data saat data tersebut dihasilkan, memungkinkan keputusan yang cepat dan berbasis data yang lebih akurat. Teknologi seperti Apache Kafka dan Apache Flink memungkinkan perusahaan untuk memanfaatkan aliran data yang terus-menerus dengan cara yang efisien dan tepat waktu.

- a. *Apache Kafka*: *Apache Kafka* adalah sistem streaming data yang memungkinkan pengumpulan dan pemrosesan data secara real-

time dalam jumlah besar. Kafka dirancang untuk menangani *streaming data* dalam volume tinggi dan dapat menangani data yang masuk dalam bentuk urutan pesan. Sebagai platform terdistribusi, Kafka digunakan untuk menghubungkan aplikasi dan sistem yang berbeda dengan menyediakan saluran data yang cepat dan handal. Kafka memungkinkan analisis data secara *real-time*, sehingga organisasi dapat mendapatkan wawasan yang lebih cepat untuk pengambilan keputusan yang lebih baik. Ini sangat penting dalam sektor-sektor seperti *e-commerce*, perbankan, dan telekomunikasi, di mana respons yang cepat terhadap perubahan kondisi pasar atau perilaku pengguna sangat penting.

- b. *Apache Flink*: *Apache Flink* adalah *platform* pemrosesan stream yang dapat menangani aliran data dalam waktu nyata dengan latensi yang rendah. *Flink* mendukung pemrosesan stateful dan memungkinkan pengolahan data dalam bentuk *windowing*, agregasi, dan pencocokan pola. Keunggulan *Flink* adalah kemampuannya untuk melakukan pemrosesan data berkelanjutan, yang memungkinkan organisasi untuk mengidentifikasi pola dalam data yang terus berubah. *Flink* juga mendukung pemrosesan data *batch*, memberikan fleksibilitas lebih dalam pengolahan data. Dengan *Flink*, organisasi dapat melakukan analisis *real-time* untuk mengambil keputusan lebih

cepat, seperti mendeteksi penipuan, memberikan rekomendasi produk yang lebih akurat, atau mengoptimalkan operasi.

Teknologi pemrosesan data *real-time* memungkinkan perusahaan untuk merespons situasi dengan lebih cepat, meningkatkan pengalaman pelanggan, dan meraih keunggulan kompetitif.

2. Basis Data Otonom

Basis data otonom merujuk pada sistem basis data yang menggunakan kecerdasan buatan (AI) untuk melakukan banyak tugas pengelolaan secara otomatis, mengurangi intervensi manusia dan meningkatkan efisiensi operasional. Salah satu contoh terkemuka dari teknologi ini adalah *Oracle Autonomous Database*.

- a. *Oracle Autonomous Database: Oracle Autonomous Database* adalah basis data berbasis AI yang dapat mengelola dirinya sendiri, termasuk dalam hal pencadangan, pemulihan, dan pengelolaan kinerja. Teknologi ini mengintegrasikan fitur seperti *self-healing*, yang memungkinkan *database* untuk secara otomatis memperbaiki kesalahan dan *self-tuning*, yang menyesuaikan konfigurasi untuk mengoptimalkan kinerja tanpa campur tangan manusia. Dengan menggunakan *machine learning*, basis data ini dapat memprediksi dan mencegah

masalah kinerja sebelum terjadi, memungkinkan ketersediaan dan keandalan yang lebih tinggi. Pencadangan otomatis dan pemulihan data adalah bagian dari proses ini, mengurangi waktu yang dibutuhkan untuk pemulihan sistem jika terjadi kegagalan (I. P. Putri et al., 2024).

Keunggulan utama dari *Oracle Autonomous Database* adalah kemampuannya untuk meningkatkan efisiensi operasional dan mengurangi biaya operasional dengan otomatisasi. Organisasi tidak perlu lagi mengalokasikan banyak waktu dan sumber daya untuk tugas-tugas administratif seperti tuning kinerja atau pemeliharaan rutin. Hal ini memungkinkan tim TI untuk fokus pada pengembangan aplikasi dan strategi bisnis, serta memanfaatkan teknologi untuk meningkatkan produktivitas secara keseluruhan.

3. Integrasi *Blockchain* dengan Basis Data

Blockchain adalah teknologi ledger terdistribusi yang mendukung pencatatan data secara aman dan transparan. Ketika diterapkan pada sistem basis data, blockchain dapat meningkatkan integritas data, memberikan transparansi, dan menjamin keamanan yang lebih tinggi.

- a. Prinsip Dasar *Blockchain* dalam Basis Data: *Blockchain* menyimpan data dalam blok yang terhubung dengan kriptografi, membentuk rantai yang tidak dapat diubah. Setiap

blok mengandung informasi transaksi dan referensi ke blok sebelumnya, memastikan urutan yang konsisten. Keunggulan utama dari teknologi *blockchain* dalam basis data adalah kemampuannya untuk menyediakan audit trail yang transparan dan tidak dapat diubah. Ini sangat penting dalam aplikasi yang membutuhkan verifikasi transaksi dan keamanan tingkat tinggi, seperti logistik, keuangan, dan perbankan. Dengan menggunakan *blockchain*, semua perubahan pada basis data tercatat dan dapat dilacak dengan mudah, mengurangi potensi manipulasi data atau penipuan.

- b. Aplikasi *Blockchain* dalam Keamanan Data: Integrasi *blockchain* dengan basis data menyediakan tingkat keamanan yang lebih tinggi, terutama dalam aplikasi yang membutuhkan integritas data yang tinggi. Sebagai contoh, dalam sektor perbankan, *blockchain* dapat digunakan untuk menyimpan catatan transaksi secara aman dan transparan. Begitu transaksi dicatat dalam *blockchain*, ia tidak bisa diubah atau dihapus, yang mencegah potensi penipuan. Di logistik, *blockchain* memungkinkan pelacakan barang secara real-time, memberikan transparansi yang lebih besar kepada semua pihak yang terlibat. Integrasi ini dapat mengurangi biaya administrasi dan meningkatkan efisiensi dengan menyediakan satu sumber data yang dapat dipercaya.

Teknologi *blockchain* memberikan solusi untuk masalah keamanan dan transparansi yang seringkali menjadi kendala dalam pengelolaan basis data tradisional, terutama dalam industri yang sangat bergantung pada verifikasi dan audit data yang tepat waktu.

F. Tantangan dan Arah Masa Depan

Pengelolaan basis data di era digital terus berkembang dengan teknologi yang semakin canggih. Meskipun teknologi basis data modern menawarkan banyak solusi, ada sejumlah tantangan yang harus dihadapi oleh organisasi saat mengelola data dalam jumlah besar, kompleksitas yang tinggi, serta kecepatan akses yang semakin cepat. Di sisi lain, kemajuan teknologi, seperti penggunaan kecerdasan buatan (AI) dan *blockchain*, membuka arah masa depan yang menjanjikan dalam dunia basis data.

1. Tantangan Pengelolaan Big Data

Pengelolaan *Big Data* adalah tantangan yang kompleks karena sifat data yang besar, beragam, dan cepat berkembang. Tantangan ini melibatkan berbagai aspek:

- a. Volume Data yang Meningkat: Data yang dihasilkan setiap detiknya semakin besar, terutama dengan semakin berkembangnya *Internet of Things* (IoT), transaksi digital, dan aplikasi berbasis *cloud*. Organisasi harus mengelola volume data

yang dapat mencapai *terabyte* hingga *petabyte* tanpa mengorbankan kinerja atau efisiensi biaya. Untuk itu, penggunaan *data lakes* atau sistem penyimpanan terdistribusi seperti Hadoop menjadi semakin penting. Namun, tantangan terbesar adalah memastikan bahwa penyimpanan data dilakukan dengan cara yang efisien dan tidak membebani sumber daya organisasi.

- b. Keragaman Format Data: Data tidak hanya terstruktur dalam tabel seperti yang digunakan dalam basis data relasional, tetapi juga hadir dalam bentuk tidak terstruktur, seperti teks, gambar, dan video. Oleh karena itu, teknologi NoSQL dan *Apache Kafka* diperlukan untuk mengelola berbagai jenis data, baik terstruktur maupun tidak terstruktur, yang dihasilkan oleh sistem yang berbeda.
- c. Kecepatan dan Pemrosesan *Real-Time*: Kecepatan data yang terus meningkat menjadi tantangan signifikan. Dengan adanya *streaming data* yang terus mengalir, perusahaan harus memiliki kemampuan untuk menganalisis dan bertindak atas data dalam waktu nyata. Teknologi seperti *Apache Kafka* dan *Apache Flink* memungkinkan pemrosesan dan analisis data secara real-time, yang sangat penting untuk aplikasi yang membutuhkan respons cepat, seperti dalam keuangan dan deteksi penipuan.

Pengelolaan *Big Data* membutuhkan pendekatan yang holistik, dengan solusi yang mampu menangani volume, variasi, dan kecepatan data yang terus berkembang.

2. Peningkatan Keamanan dan Privasi Data

Keamanan dan privasi data menjadi perhatian utama di era digital, terutama karena ancaman *cyber* yang semakin berkembang. Perusahaan harus mengimplementasikan sistem yang tidak hanya melindungi data dari ancaman eksternal, tetapi juga memastikan bahwa data pribadi individu dilindungi sesuai dengan regulasi yang berlaku.

- a. *Zero Trust Architecture* (ZTA): Konsep *Zero Trust* berfokus pada prinsip bahwa tidak ada entitas yang dapat dipercaya secara otomatis, baik itu pengguna internal maupun eksternal. Setiap permintaan untuk mengakses sistem atau data harus diverifikasi terlebih dahulu melalui berbagai metode autentikasi, seperti *multi-factor authentication* (MFA). ZTA memastikan bahwa hanya entitas yang terverifikasi yang dapat mengakses data, bahkan jika mereka berada di dalam jaringan perusahaan.
- b. Regulasi dan Kepatuhan: Organisasi diharuskan untuk mematuhi berbagai regulasi yang mengatur pengelolaan data pribadi, seperti GDPR di Eropa dan CCPA di California. Untuk itu, implementasi teknik seperti *data anonymization* dan *data*

erasure menjadi penting agar data pribadi dapat diproses dengan aman dan sesuai dengan aturan yang berlaku. Selain itu, teknologi seperti *blockchain* dapat menawarkan transparansi lebih dalam pengelolaan data pribadi, memungkinkan audit yang lebih mudah dan aman.

Sebagai respons terhadap ancaman keamanan yang semakin canggih, penggunaan model seperti *Zero Trust* dan implementasi kebijakan pengelolaan privasi data yang ketat akan menjadi bagian penting dari strategi organisasi dalam menjaga integritas dan kerahasiaan data.

3. Masa Depan Basis Data Otonom

Basis data otonom, yang didorong oleh kecerdasan buatan, akan semakin berkembang di masa depan. Sistem ini mampu mengelola dirinya sendiri tanpa campur tangan manusia, meningkatkan efisiensi dan mengurangi ketergantungan pada administrator basis data.

- a. AI dan Pembelajaran Mesin dalam Basis Data Otonom: Basis data otonom seperti *Oracle Autonomous Database* memanfaatkan AI untuk melakukan optimasi otomatis, pemeliharaan, dan pemulihan sistem. Dengan *machine learning*, basis data ini dapat mengidentifikasi pola penggunaan, memperbaiki masalah kinerja, dan menyesuaikan konfigurasi

untuk mencapai performa terbaik. Kemampuan ini mengurangi beban kerja tim TI dan memungkinkan perusahaan untuk fokus pada inovasi, bukan pada pemeliharaan rutin.

- b. Pengurangan Ketergantungan pada Administrasi Manual: Dengan otomatisasi pengelolaan basis data, perusahaan akan mengurangi ketergantungan pada administrator basis data yang mengelola dan memelihara infrastruktur basis data. Sistem yang otonom ini dapat secara otomatis mengatasi masalah yang biasanya memerlukan perhatian manusia, seperti optimasi kinerja, pencadangan, dan pemulihan. Penggunaan AI untuk mendeteksi masalah dan memperbaikinya secara otomatis memungkinkan operasi yang lebih efisien dan mengurangi risiko kesalahan manusia.

Masa depan basis data otonom menunjukkan potensi besar untuk meningkatkan efisiensi operasional, mengurangi biaya, dan memungkinkan pengelolaan data yang lebih canggih dengan sedikit intervensi manual.

G. Kesimpulan dari Konsep Teknologi Basis Data

Konsep Teknologi Basis Data, telah mengidentifikasi beberapa tren, inovasi, dan tantangan yang membentuk lanskap basis data saat ini dan di masa depan. Beberapa aspek kunci yang dapat disimpulkan adalah sebagai berikut:

1. **Evolusi Teknologi Basis Data:** Teknologi basis data terus berkembang untuk mengakomodasi kebutuhan pengelolaan data yang semakin kompleks, terutama dengan hadirnya *Big Data*, AI, dan *blockchain*. Konsep-konsep baru seperti basis data otonom yang didorong oleh kecerdasan buatan dan kemampuan pemrosesan *real-time* telah memungkinkan pengelolaan data yang lebih efisien dan otomatis, mengurangi ketergantungan pada pengelolaan manual dan meningkatkan kinerja sistem.
2. **Pengelolaan *Big Data*:** *Big Data* mengharuskan penggunaan teknologi yang mampu menangani volume besar, keragaman format, dan kecepatan tinggi data yang dihasilkan. Platform seperti *Apache Hadoop*, *Apache Kafka*, dan NoSQL databases (seperti MongoDB dan Redis) memungkinkan organisasi untuk menyimpan dan menganalisis data secara lebih efektif, baik dalam format terstruktur maupun tidak terstruktur.
3. **Keamanan dan Privasi Data:** Aspek keamanan dan privasi data menjadi prioritas utama dengan meningkatnya ancaman siber dan kebutuhan akan perlindungan data pribadi. Pendekatan *Zero Trust Architecture* dan regulasi seperti GDPR serta CCPA semakin mendorong organisasi untuk memastikan bahwa data diproses dengan aman dan transparan. Enkripsi, RBAC, dan

penerapan kebijakan penghapusan data juga menjadi bagian penting dari strategi perlindungan data.

4. Inovasi dalam Manajemen Basis Data: *Blockchain* dan basis data *multitenant* membuka jalan baru untuk transparansi, keamanan, dan efisiensi dalam pengelolaan data. Teknologi ini memungkinkan pengelolaan data yang lebih aman dan dapat diaudit, serta mendukung aplikasi yang membutuhkan data yang tidak dapat diubah atau tercatat secara permanen.
5. Masa Depan Teknologi Basis Data: Masa depan basis data akan semakin dipengaruhi oleh teknologi AI, *machine learning*, dan *blockchain*, dengan tujuan untuk mengoptimalkan manajemen data secara otomatis, mempercepat pemrosesan data secara real-time, dan meningkatkan keamanan serta transparansi. Penggunaan basis data otonom yang dapat mengelola dirinya sendiri akan mengurangi ketergantungan pada intervensi manusia, yang dapat meningkatkan efisiensi operasional dan mengurangi kesalahan dalam pengelolaan data.
6. Tantangan Pengelolaan *Big Data*: Tantangan terbesar dalam pengelolaan *Big Data* meliputi bagaimana menangani volume data yang semakin besar, keragaman format, serta kecepatan data yang terus berkembang. Selain itu, perusahaan harus memiliki solusi yang mampu mengelola dan memproses data

dengan cara yang efisien tanpa mengorbankan kinerja dan biaya.

Secara keseluruhan, konsep teknologi basis data yang terus berkembang menawarkan solusi yang lebih canggih, efisien, dan aman untuk memenuhi tuntutan dunia digital yang semakin kompleks. Dengan adopsi teknologi yang tepat, organisasi dapat lebih mudah mengelola data mereka, mengoptimalkan operasional, serta menjaga keamanan dan privasi data di tengah tantangan yang ada.

DAFTAR PUSTAKA

- Antoni, D., Avini, T., & Heryati, A. H. (2024). *Business process reengineering*.
- Ashdown, L., Keesling, D., & Kyte, T. (2021). *Oracle® Database database concepts 21c*. Oracle.
<https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/database-concepts.pdf>
- Avini, T., Alamin, Z., Maulani, G., & Perkasa, E. B. (2024). *Fundamental algoritma*.
- Elmasri, R., & Navathe, S. B. (2011). *Fundamentals of database systems* (M. Hirsch, Ed.; 7th ed.). Pearson Education, Inc.
<https://drive.google.com/file/d/0B633TIYUDLMZVHUzNnBaNTFDc2c/edit?resourcekey=0-KPyZ7loQltX36oCrmysSOQ>
- Hartatik, B., Titin, A., Nengsih, S., Si, A., Baradja, E. B., Harto, & Avini, T. (2023). *Data science for business*.
www.sonpedia.com
- Hartono, Y., Ermatita, Rini, D. P., & Terttiaavini, T. (2021). Building a weighted performance indicator concept utilized the respondent's opinion approach. *Proceeding - ICERA 2021: 2021 3rd International Conference on Electronics*

Representation and Algorithm.

<https://doi.org/10.1109/ICERA53111.2021.9538635>

Lahoti, S., & Singh, D. (2022). Blockchain technology based secure data sharing in cloud computing. *IEEE International Conference on Knowledge Engineering and Communication Systems (ICKES 2022)*.
<https://doi.org/10.1109/ICKECS56523.2022.10060616>

Marcelina, D., Kurnia, A., & Terttiaavini, T. (2023). Analisis kluster kinerja usaha kecil dan menengah menggunakan algoritma K-Means clustering. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, 3(2), 293–301.
<https://doi.org/10.57152/malcom.v3i2.952>

Putri, I. P., Terttiaavini, T., & Arminarahmah, N. (2024). Analisis perbandingan algoritma machine learning untuk prediksi stunting pada anak. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, 4(1).
<https://doi.org/10.57152/malcom.v4i1.1078>

Putri, S. S., Ani, Y. A., & Terttiaavini, T. (2023). Sistem pendukung keputusan penilaian kinerja pegawai dengan metode Simple Additive Weighting (SAW) (studi kasus Badan Narkotika Nasional Kabupaten Ogan Komering Ilir). *Jurnal Nasional Komputasi Dan Teknologi Informasi*, 6(3).

- Sahamony, N. F., Terttiaavini, T., & Rianto, H. (2024). Analisis perbandingan kinerja model machine learning untuk memprediksi risiko stunting pada pertumbuhan anak. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, 4(2).
<https://doi.org/10.57152/malcom.v4i2.1210>
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database system concepts* (7th ed., p. 1349). McGraw-Hill.
<https://www.db-book.com/>
- Terttiaavini, Hartono, Y., Ermatita, & Rini, D. P. (2023a). Comparison of Simple Additive Weighting method and Weighted Performance Indicator method for lecturer performance assessment. *International Journal of Modern Education and Computer Science*, 15(2).
<https://doi.org/10.5815/ijmeecs.2023.02.01>
- Terttiaavini, Hartono, Y., Ermatita, & Rini, D. P. (2023b). Development of a decision support system on employee performance assessment using Weighted Performance Indicators method. *International Journal of Information Engineering and Electronic Business*, 15(3).
<https://doi.org/10.5815/ijieeb.2023.03.01>
- Terttiaavini, & Saputra, T. S. (2020). *Data mining model dan implementasi di perguruan tinggi*.

- T., Zamzam, F., Ramadhan, M., Rosni, K. A., Setiawan Saputra, T., Heryati, A., & D. (2018). Clustering analysis of premier research fields. *International Journal of Engineering & Technology*, 7(4.44).
<https://doi.org/10.14419/ijet.v7i4.44.26860>
- VeraSafe. (2016). *General Data Protection Regulation*.
<https://gdpr.verasafe.com/recitals>
- Zikopoulos, P. (2012). *Understanding big data: Analytics for enterprise-class Hadoop and streaming data*.
https://books.google.com/books/about/Understanding_Big_Data_Analytics_for_Ent.html?hl=id&id=0sJqV1t4UVsC

BAB 2

ARSITEKTUR BASIS DATA DAN LINGKUNGAN SISTEM BASIS DATA

Oleh : Yudi Herdiana, S.T., M.T.

A. Pengantar

Basis data (*database*) adalah kumpulan data yang disimpan secara sistematis dan dapat diakses serta dikelola dengan menggunakan perangkat lunak tertentu. Arsitektur basis data dan lingkungan sistem basis data mencakup berbagai komponen dan proses yang mendukung operasi basis data, termasuk perangkat keras, perangkat lunak, data, prosedur, dan pengguna.

B. Arsitektur Basis Data

Arsitektur basis data merujuk pada desain dan struktur sistem basis data yang menentukan bagaimana data disimpan, diakses, dan dikelola. Arsitektur ini biasanya dibagi menjadi beberapa tingkatan atau lapisan, masing-masing dengan fungsi dan tanggung jawab tertentu.

1. Tingkatan Arsitektur Basis Data

a. Tingkat Internal (*Internal Level*):

- 1) Tingkatan ini berkaitan dengan bagaimana data disimpan secara fisik di dalam sistem basis data. Ini meliputi struktur penyimpanan, pengindeksan, pengelolaan ruang penyimpanan, dan mekanisme akses data fisik.
- 2) Fokus pada efisiensi penyimpanan dan akses data.

b. Tingkat Konseptual (*Conceptual Level*):

- 1) Tingkatan ini berhubungan dengan representasi logis dari data yang tidak tergantung pada cara penyimpanan fisiknya. Model data pada tingkatan ini mencakup entitas, atribut, dan hubungan antar entitas.
- 2) Menggambarkan struktur data yang digunakan oleh seluruh organisasi.

c. Tingkat Eksternal (*External Level*):

- 1) Tingkatan ini berkaitan dengan pandangan pengguna terhadap data. Setiap pengguna atau kelompok pengguna dapat memiliki pandangan yang berbeda terhadap data yang disesuaikan dengan kebutuhan mereka.
- 2) Memungkinkan berbagai pandangan dan keamanan data berdasarkan hak akses pengguna.

2. Model Basis Data

Beberapa model basis data yang umum digunakan meliputi:

- a. Model Relasional: Data diorganisasikan dalam tabel (relasi) yang terdiri dari baris dan kolom.
- b. Model Hierarkis: Data diatur dalam struktur pohon dengan hubungan induk-anak.
- c. Model Jaringan: Data diatur dalam grafik dengan hubungan yang lebih kompleks daripada model hierarkis.
- d. Model Berorientasi Objek: Data diatur sebagai objek, seperti dalam pemrograman berorientasi objek.
- e. Model NoSQL: Digunakan untuk data yang tidak terstruktur atau semi-terstruktur, meliputi basis data dokumen, basis data graf, dan basis data kunci-nilai.

C. Lingkungan Sistem Basis Data

Lingkungan sistem basis data mencakup semua komponen dan sumber daya yang mendukung operasi dan manajemen basis data.

1. Komponen Lingkungan Sistem Basis Data

- a. Perangkat Keras (*Hardware*):
Komputer server, penyimpanan, jaringan, dan perangkat lainnya yang diperlukan untuk mendukung sistem basis data.

b. Perangkat Lunak (*Software*):

- 1) Sistem Manajemen Basis Data (DBMS): Perangkat lunak yang digunakan untuk membuat, mengelola, dan memanipulasi basis data.
- 2) Aplikasi Pendukung: Perangkat lunak lain yang mendukung operasi basis data, seperti alat pemantauan, alat backup, dan alat keamanan.

c. Data: Kumpulan data yang disimpan dalam basis data, termasuk data mentah, metadata, dan data derivatif.

d. Prosedur: Panduan dan aturan yang mengatur bagaimana sistem basis data dioperasikan dan dikelola. Ini mencakup kebijakan keamanan, prosedur backup, dan rencana pemulihan bencana.

e. Pengguna:

- 1) Administrator Basis Data (DBA): Orang yang bertanggung jawab atas pengelolaan dan pemeliharaan basis data.
- 2) Pengembang: Orang yang membuat dan memelihara aplikasi basis data.
- 3) Pengguna Akhir: Orang yang menggunakan aplikasi basis data untuk mengakses dan memanipulasi data.

2. Fungsi dan Tanggung Jawab dalam Lingkungan Sistem Basis Data

a. Administrasi Basis Data:

- 1) Melakukan tugas pemeliharaan, backup, dan pemulihan basis data.
- 2) Mengoptimalkan kinerja basis data dan memastikan ketersediaannya.

b. Keamanan Basis Data:

- 1) Melindungi data dari akses yang tidak sah dan memastikan kerahasiaan, integritas, dan ketersediaan data.
- 2) Mengimplementasikan kontrol akses dan enkripsi data.

c. Desain Basis Data:

- 1) Merancang struktur basis data yang efisien dan efektif untuk memenuhi kebutuhan pengguna.
- 2) Menggunakan model data untuk memvisualisasikan struktur basis data.

d. Pemantauan dan Optimasi Kinerja:

- 1) Memantau kinerja basis data dan mengidentifikasi serta mengatasi masalah kinerja.

- 2) Mengoptimalkan kueri dan indeks untuk meningkatkan kinerja basis data.

e. Backup dan Pemulihan:

- 1) Melakukan backup rutin untuk memastikan data dapat dipulihkan dalam situasi darurat.
- 2) Merencanakan dan menguji prosedur pemulihan bencana untuk meminimalkan kehilangan data.

D. Contoh Kasus Sederhana

1. Kasus Sederhana: Toko Buku Online

Sebuah toko buku online menggunakan model basis data relasional untuk mengelola data buku, pelanggan, dan pesanan. Sistem ini terdiri dari beberapa tabel, termasuk tabel buku (*book*), pelanggan (*customer*), dan pesanan (*order*).

a. Tingkat Internal:

- 1) Data buku, pelanggan, dan pesanan disimpan dalam tabel yang diindeks untuk meningkatkan kecepatan pencarian.
- 2) Mekanisme penyimpanan yang efisien digunakan untuk mengurangi penggunaan ruang penyimpanan.

b. Tingkat Konseptual:

- 1) Model data menyertakan entitas Buku, Pelanggan, dan Pesanan dengan atribut masing-masing.

- 2) Relasi antara tabel diatur melalui kunci asing, seperti ID pelanggan yang menghubungkan tabel pelanggan dengan tabel pesanan.

c. Tingkat Eksternal:

- 1) Pengguna akhir, seperti pelanggan dan admin toko, memiliki pandangan berbeda terhadap data.
- 2) Pelanggan hanya dapat melihat data buku yang tersedia dan riwayat pesanan mereka, sementara admin dapat mengakses dan mengelola seluruh data buku, pelanggan, dan pesanan.

2. Kasus Sederhana: Sistem Pendaftaran Sekolah

Sebuah sekolah menggunakan sistem basis data untuk mengelola pendaftaran siswa. Basis data ini mencakup tabel siswa (*student*), kursus (*course*), dan pendaftaran (*enrollment*).

a. Tingkat Internal:

Data siswa, kursus, dan pendaftaran disimpan secara efisien dengan menggunakan indeks pada kolom yang sering digunakan dalam pencarian, seperti ID siswa dan ID kursus.

b. Tingkat Konseptual:

- 1) Model data menyertakan entitas Siswa, Kursus, dan Pendaftaran.

- 2) Setiap pendaftaran menghubungkan satu siswa dengan satu kursus melalui kunci asing.

c. Tingkat Eksternal:

- 1) Siswa dapat melihat daftar kursus yang tersedia dan riwayat pendaftaran mereka.
- 2) Administrator sekolah dapat mengakses data lengkap tentang siswa, kursus, dan pendaftaran untuk mengelola kurikulum dan pendaftaran.

Arsitektur basis data dan lingkungan sistem basis data adalah komponen penting dalam pengelolaan data yang efektif. Dengan memahami berbagai tingkatan arsitektur basis data dan komponen lingkungan sistem basis data, organisasi dapat merancang dan mengelola sistem basis data yang efisien, aman, dan handal. Contoh kasus sederhana yang disajikan menunjukkan penerapan prinsip-prinsip arsitektur dan lingkungan basis data dalam skenario praktis, menggambarkan bagaimana sistem basis data dapat disesuaikan untuk memenuhi kebutuhan spesifik setiap organisasi.

DAFTAR PUSTAKA

- Abadi, D. J., & Boncz, P. A. (2013). Column-store systems. *VLDB Journal*, 22(4), 469-490.
- Abdillah, L. A., Santosa, P. I., & Nur, M. (2019). A review of NoSQL implementation in Big Data Analytics using Google BigQuery and Apache Hive. *Journal of Physics: Conference Series*, 1235(1), 012024. <https://doi.org/10.1088/1742-6596/1235/1/012024>
- Arief, M. (2020). *Pemrograman Basis Data dengan Python*. Yogyakarta: Penerbit Andi.
- Casteel, J. (2013). *Oracle 12c: SQL* (3rd ed.). Cengage Learning.
- Connolly, T. M., & Begg, C. E. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson.
- Date, C. J. (2019). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.
- DeWitt, D. J., & Gray, J. (1992). Parallel Database Systems: The Future of High-Performance Database Systems. *Communications of the ACM*, 35(6), 85-98.
- Dignazio, F., & Heck, T. (2017). *Data Science with SQL Server 2017*. Packt Publishing.

- Fauzan, R., & Nugroho, A. S. (2020). Comparative study of MongoDB and MySQL performance on data insertion and retrieval. *Journal of Information Systems Engineering and Business Intelligence*, 6(1), 43-51. <https://doi.org/10.20473/jisebi.6.1.43-51>
- Hakim, L., & Purwanto, W. (2021). Design and implementation of a distributed database system for educational institutions using MongoDB. *International Journal of Emerging Trends in Engineering Research*, 9(7), 888-894. <https://doi.org/10.30534/ijeter/2021/38972021>
- Harrison, G. (2010). *Next Generation Databases: NoSQL and Big Data*. Apress.
- Hoffer, J. A., Ramesh, V., & Topi, H. (2016). *Modern Database Management* (12th ed.). Pearson.
- Kurniawan, B., & Putra, P. D. (2018). Performance analysis of relational and non-relational databases in storing big data: A case study of real-time monitoring system. *Journal of Information Technology and Computer Science*, 3(1), 23-30. <https://doi.org/10.25126/jitecs.2022.01.003>
- Kurniawan, R. (2018). *Pemrograman Basis Data dengan MySQL dan PHP*. Jakarta: PT Gramedia Pustaka Utama.
- Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media, Inc.

- Molinaro, A. (2005). *SQL Cookbook: Query Solutions and Techniques for Database Developers*. O'Reilly Media.
- Nugroho, H. T., & Wijaya, A. A. (2019). Implementation of database design in educational information systems. *International Journal of Computer Applications*, 182(16), 9-13. <https://doi.org/10.5120/ijca2019918501>
- O'Neil, P., & O'Neil, E. (2001). *Database: Principles, Programming, and Performance* (2nd ed.). Morgan Kaufmann.
- Patel, A., & Gupta, S. (2016). *Database Management Systems: A Practical Approach* (1st ed.). Firewall Media.
- Prasetyo, E., & Purba, J. T. (2020). Analysis of database performance using SQL and NoSQL: A study on the processing speed of queries. *Journal of Information Systems*, 12(2), 45-52. <https://doi.org/10.21067/jis.v12i2.5205>
- Pratama, A., & Suryadi, A. (2021). *Desain dan Implementasi Basis Data*. Bandung: Informatika.
- Ramadhani, N. F., & Setiawan, M. A. (2018). Evaluating the scalability of NoSQL databases in IoT applications. *Indonesian Journal of Electrical Engineering and Computer Science*, 10(1), 124-131. <https://doi.org/10.11591/ijeecs.v10.i1.pp124-131>
- Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems* (3rd ed.). McGraw-Hill.

- Redgate. (2015). *SQL Server Performance Tuning*. Redgate Books.
- Rahayu, S., & Lestari, D. (2019). *Pengantar Basis Data: Teori dan Aplikasi*. Jakarta: PT Elex Media Komputindo.
- Rob, P., & Coronel, C. (2008). *Database Systems: Design, Implementation, & Management* (9th ed.). Course Technology.
- Setiawan, H. (2020). *Manajemen Basis Data: Konsep dan Praktik*. Surabaya: Penerbit Graha Ilmu.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill Education.
- Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4), 10-11.
- Subekti, T. (2021). *Analisis dan Desain Basis Data*. Jakarta: Penerbit Erlangga.
- Sutanto, A. (2020). *Sistem Basis Data Distribusi: Teori dan Praktik*. Yogyakarta: Penerbit Graha Ilmu.
- Susanto, A. (2018). *Implementasi Sistem Basis Data untuk E-commerce*. Bandung: Penerbit Informatika.
- Susanto, H., & Wijaya, H. (2022). Comparison of query performance between SQL and NoSQL databases for e-commerce platforms. *Journal of Computer Science and Applications*, 14(3), 55-62.
<https://doi.org/10.35421/jcsa.v14i3.987>

- Tanenbaum, A. S., & Steen, M. V. (2007). *Distributed Systems: Principles and Paradigms* (2nd ed.). Prentice Hall.
- Wibowo, B. H. (2020). *Sistem Basis Data dan Implementasinya*. Yogyakarta: Penerbit Andi.
- Watson, M. (2010). *Data Modeling Made Simple: A Practical Guide for Business & IT Professionals*. Technics Publications.
- Wibowo, B. H. (2020). *Sistem Basis Data dan Implementasinya*. Yogyakarta: Penerbit Andi.

BAB 3

PEMODELAN DATA DENGAN ER DAN MENGGAMBARKANNYA DENGAN NOTASI BARKER

Oleh : Agung Yuliyanto Nugroho S.Kom., M.Kom., M.Par

Pemodelan data dengan ER (*Entity-Relationship*) adalah proses untuk menggambarkan struktur data dalam bentuk diagram yang menunjukkan hubungan antar entitas dalam sistem. Pemodelan ini digunakan untuk merancang basis data yang efisien dan memudahkan pemahaman hubungan antar data.

A. Konsep Dasar *Entity-Relationship* (ER)

Konsep Dasar *Entity-Relationship* (ER) adalah pendekatan yang digunakan untuk memodelkan data secara konseptual dalam sistem informasi. Model ER dikembangkan oleh Peter Chen pada tahun 1976 dan menjadi salah satu teknik paling populer dalam desain basis data. Berikut adalah konsep-konsep dasar dalam *Entity-Relationship* (ER):

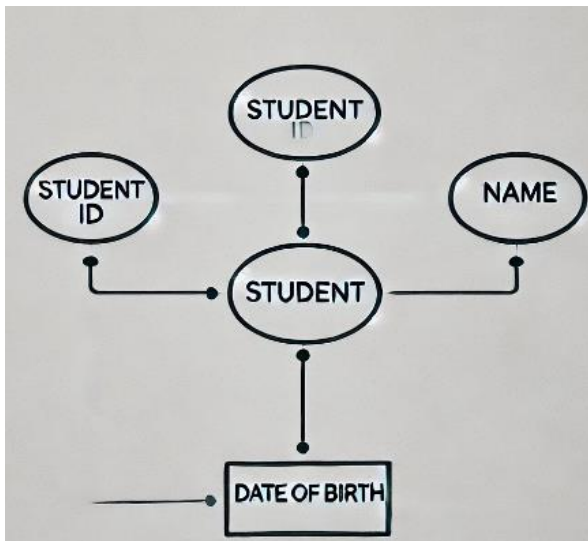
1. Entitas (*Entity*)

Entitas adalah objek nyata atau abstrak yang dapat diidentifikasi secara unik dalam domain sistem.

a. Jenis Entitas:

- 1) Entitas Kuat (*Strong Entity*): Entitas yang dapat berdiri sendiri dan memiliki atribut kunci (*Primary Key*) untuk identifikasi unik.
- 2) Entitas Lemah (*Weak Entity*): Entitas yang tidak dapat berdiri sendiri dan bergantung pada entitas lain. Memerlukan kunci utama dari entitas lain untuk identifikasinya.

b. Contoh: Mahasiswa, Dosen, Mata Kuliah.



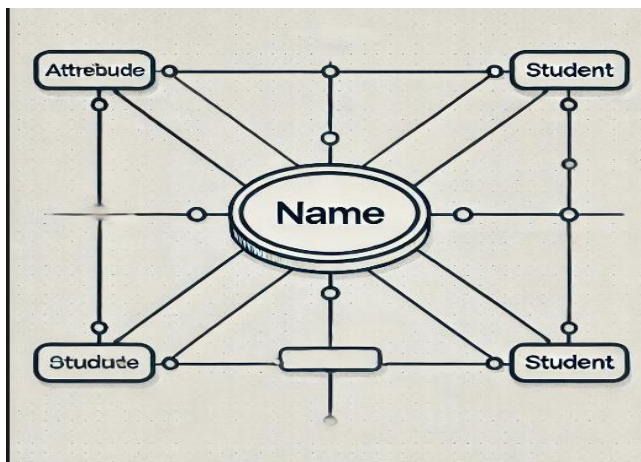
Gambar 3. 1 Gambar Entitas (*Entity*)

2. Atribut (*Attribute*)

Atribut adalah sifat atau karakteristik dari entitas atau hubungan yang memberikan informasi lebih lanjut.

a. Jenis Atribut:

- 1) Atribut Sederhana (*Simple*): Tidak dapat dipecah lagi (contoh: Nama).
- 2) Atribut Komposit (*Composite*): Terdiri dari beberapa komponen (contoh: Alamat, yang terdiri dari Jalan, Kota, dan Kode Pos).
- 3) Atribut *Multivalued*: Dapat memiliki lebih dari satu nilai (contoh: Nomor Telepon).
- 4) Atribut Derivatif: Nilainya dihitung dari atribut lain (contoh: Usia dihitung dari Tanggal Lahir).



Gambar 3.2 Gambar Atribut

3. Relasi (*Relationship*)

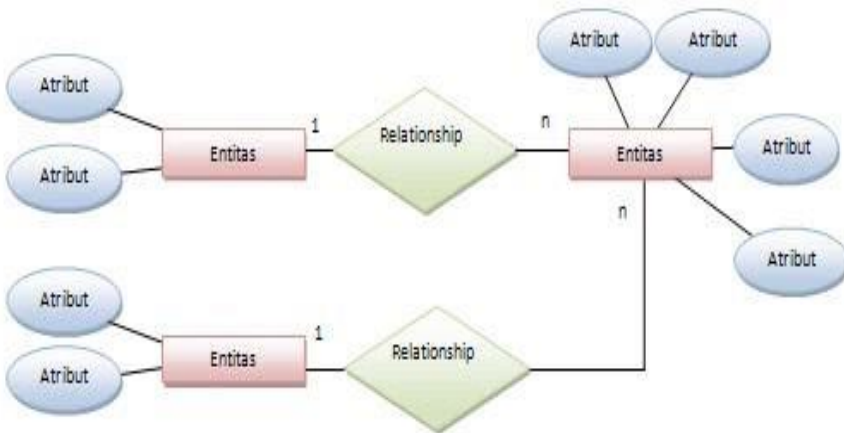
Relasi adalah hubungan antar entitas dalam model ER.

a. Derajat Relasi:

- 1) Unary (Relasi dengan diri sendiri).
- 2) Binary (Relasi antara dua entitas).
- 3) Ternary (Relasi antara tiga entitas atau lebih).

b. Jenis Relasi:

- 1) *One-to-One* (1:1): Satu entitas A hanya berhubungan dengan satu entitas B, dan sebaliknya.
- 2) *One-to-Many* (1:N): Satu entitas A dapat berhubungan dengan banyak entitas B, tetapi entitas B hanya berhubungan dengan satu entitas A.
- 3) *Many-to-Many* (M:N): Banyak entitas A dapat berhubungan dengan banyak entitas B.



Gambar 3. 3 Gambar Relasi

4. Kardinalitas (*Cardinality*)

Kardinalitas (*Cardinality*) dalam konteks basis data dan model entitas-relasi (ER) merujuk pada jumlah entitas yang dapat berhubungan dengan entitas lainnya dalam suatu relasi. Ada tiga jenis utama kardinalitas yang sering digunakan dalam desain basis data:

a. *One-to-One* (1:1):

- 1) Setiap entitas pada tabel pertama hanya dapat berhubungan dengan satu entitas pada tabel kedua, dan sebaliknya.
- 2) Contoh: Setiap pegawai memiliki satu nomor identitas pegawai (ID) yang unik, dan setiap ID hanya dimiliki oleh satu pegawai.

b. *One-to-Many* (1:N):

- 1) Setiap entitas pada tabel pertama dapat berhubungan dengan banyak entitas pada tabel kedua, tetapi setiap entitas pada tabel kedua hanya dapat berhubungan dengan satu entitas pada tabel pertama.
- 2) Contoh: Satu pelanggan dapat melakukan banyak pemesanan, tetapi setiap pemesanan hanya dilakukan oleh satu pelanggan.

c. *Many-to-Many* (M:N):

- 1) Banyak entitas pada tabel pertama dapat berhubungan dengan banyak entitas pada tabel kedua, dan sebaliknya.
- 2) Contoh: Siswa dapat mendaftar di banyak mata pelajaran, dan setiap mata pelajaran dapat memiliki banyak siswa yang terdaftar.
- 3) Contoh: Mahasiswa (N) dapat mengambil banyak Mata Kuliah (M) → *Many-to-Many*.

5. Kunci (*Key*)

Kunci adalah atribut atau kombinasi atribut yang digunakan untuk mengidentifikasi entitas secara unik.

a. Jenis Kunci:

- 1) *Primary Key*: Atribut unik untuk identifikasi entitas.
- 2) *Foreign Key*: Atribut yang menghubungkan entitas dengan entitas lain.
- 3) *Candidate Key*: Atribut yang dapat menjadi *primary key*.
- 4) *Composite Key*: Kombinasi dua atau lebih atribut untuk identifikasi unik.

6. Entitas Lemah dan Hubungan Pengidentifikasi (*Weak Entity and Identifying Relationship*)

Entitas Lemah (*Weak Entity*) dan Hubungan Pengidentifikasi (*Identifying Relationship*) adalah konsep dalam model entitas-

relasi (ER) yang digunakan untuk menggambarkan entitas yang bergantung pada entitas lain untuk dapat diidentifikasi dengan jelas. Berikut penjelasan lebih lanjut:

a. Entitas Lemah (*Weak Entity*)

- 1) Definisi: Entitas lemah adalah entitas yang tidak dapat berdiri sendiri tanpa adanya entitas lain yang dapat mengidentifikasinya. Entitas ini tidak memiliki atribut kunci yang cukup untuk membedakan setiap instansinya, sehingga memerlukan bantuan dari entitas lain (entitas yang lebih kuat) untuk membentuk identitasnya.
- 2) Ciri-ciri:
 - a) Tidak memiliki atribut kunci utama yang dapat membedakan entitas tersebut.
 - b) Bergantung pada entitas lain (yang disebut *entitas pengidentifikasi*) untuk membentuk kunci utama.
 - c) Biasanya, entitas lemah digambarkan dengan persegi panjang ganda dalam diagram ER.
- 3) Contoh:
 - a) Sebuah tanggungan dalam suatu keluarga, yang tidak dapat diidentifikasi hanya dengan nama atau informasi lainnya. Tanggungan memerlukan entitas keluarga

atau anggota keluarga yang lebih kuat sebagai referensi untuk identitasnya.

b) Entitas "*Invoice Line*" (baris faktur) yang tergantung pada entitas "*Invoice*" (faktur) karena baris faktur tidak dapat berdiri sendiri tanpa faktur yang menyertainya.

b. Hubungan Pengidentifikasi (*Identifying Relationship*)

1) Definisi: Hubungan pengidentifikasi adalah hubungan antara entitas lemah dan entitas yang lebih kuat, di mana entitas lemah tidak dapat ada atau diidentifikasi tanpa entitas pengidentifikasinya. Dalam hubungan ini, entitas pengidentifikasi menyediakan kunci utama bagi entitas lemah.

2) Ciri-ciri:

a) Hubungan ini digambarkan dengan garis ganda dalam diagram ER.

b) Entitas lemah bergantung pada entitas yang lebih kuat untuk membentuk kunci utama yang unik.

c) Hubungan ini memastikan bahwa entitas lemah dapat diidentifikasi dengan jelas berdasarkan entitas pengidentifikasinya.

3) Contoh:

- a) Dalam hubungan antara tanggungan dan keluarga, hubungan tersebut adalah pengidentifikasi karena tanggungan tidak dapat diidentifikasi tanpa keluarga yang mengidentifikasinya.
- b) Dalam kasus *Invoice Line* dan *Invoice*, baris faktur bergantung pada faktur yang ada, sehingga hubungan ini juga merupakan hubungan pengidentifikasi.

c. Contoh Diagram ER:

Dalam diagram ER, entitas lemah biasanya digambarkan dengan persegi panjang ganda, dan hubungan pengidentifikasi digambarkan dengan garis ganda yang menghubungkan entitas lemah dan entitas pengidentifikasi.

Contoh:

- 1) Entitas *Invoice* (Faktur) sebagai entitas kuat, dan *Invoice Line* (Baris Faktur) sebagai entitas lemah yang bergantung pada faktur untuk identifikasi.
- 2) Hubungan pengidentifikasi antara *Invoice* dan *Invoice Line* menggambarkan ketergantungan baris faktur pada faktur tersebut untuk identitasnya.

7. Diagram ER (ER Diagram)

Diagram Entitas-Relasi (ER Diagram) adalah representasi grafis dari struktur basis data yang menggambarkan entitas, atribut, dan hubungan antar entitas dalam suatu sistem. Diagram ini digunakan dalam desain basis data untuk menggambarkan bagaimana data terhubung dan diatur. ER Diagram membantu merancang dan memvisualisasikan struktur data sebelum implementasi dalam sistem basis data. Komponen Utama dalam ER Diagram:

a. Entitas (*Entity*):

- 1) Entitas adalah objek atau hal yang memiliki keberadaan dalam dunia nyata yang perlu disimpan dalam basis data. Setiap entitas biasanya memiliki atribut yang menggambarkan sifat-sifatnya.
- 2) Simbol: Persegi panjang.
- 3) Contoh: Pelanggan, Produk, Pegawai.

b. Atribut (*Attribute*):

- 1) Atribut adalah sifat atau karakteristik yang dimiliki oleh entitas.
- 2) Simbol: *Elips* (lingkaran).

3) Jenis Atribut:

- a) Atribut Kunci (*Primary Attribute*): Atribut yang digunakan untuk mengidentifikasi entitas secara unik (misalnya, Nomor ID Pegawai).
- b) Atribut *Multi-valued*: Atribut yang dapat memiliki lebih dari satu nilai (misalnya, Nomor Telepon untuk seorang pelanggan).
- c) Atribut Terhitung (*Derived Attribute*): Atribut yang nilainya dihitung berdasarkan atribut lain (misalnya, Usia yang dihitung berdasarkan tanggal lahir).

c. Hubungan (*Relationship*):

- 1) Hubungan menggambarkan cara entitas berinteraksi satu sama lain.
- 2) Simbol: Rombus (*lozenge*).
- 3) Contoh: Pelanggan melakukan Pemesanan, Pegawai bekerja di Departemen.

d. Kardinalitas (*Cardinality*):

- 1) Menunjukkan jumlah entitas yang dapat berhubungan dengan entitas lain dalam suatu hubungan.
- 2) Jenis Kardinalitas:
 - a) *One-to-One* (1:1): Satu entitas berhubungan dengan satu entitas lainnya.

- b) *One-to-Many* (1:N): Satu entitas berhubungan dengan banyak entitas lain.
- c) *Many-to-Many* (M:N): Banyak entitas berhubungan dengan banyak entitas lain.
- e. Entitas Lemah dan Hubungan Pengidentifikasi:
 - 1) Entitas Lemah (*Weak Entity*): Entitas yang bergantung pada entitas lain untuk diidentifikasi. Biasanya memiliki atribut kunci yang tidak lengkap.
 - 2) Hubungan Pengidentifikasi (*Identifying Relationship*): Hubungan antara entitas lemah dan entitas yang lebih kuat yang menyediakan kunci utama untuk entitas lemah.

Langkah-Langkah Membuat ER Diagram:

1. Identifikasi Entitas: Tentukan objek atau hal yang akan disimpan dalam basis data.
2. Tentukan Atribut: Tentukan atribut yang diperlukan untuk masing-masing entitas.
3. Tentukan Hubungan: Tentukan bagaimana entitas-entitas tersebut berhubungan satu sama lain.
4. Tentukan Kardinalitas: Tentukan jenis hubungan (1:1, 1:N, M:N) antara entitas.

5. Gambarkan Diagram: Gunakan simbol-simbol ER untuk menggambarkan entitas, atribut, hubungan, dan kardinalitas.

Contoh ER Diagram: Misalnya, sebuah sistem basis data untuk toko online:

1. Entitas:

- a. Pelanggan (ID_Pelanggan, Nama, Alamat)
- b. Produk (ID_Produk, Nama_Produk, Harga)
- c. Pemesanan (ID_Pemesanan, Tanggal_Pemesanan)

2. Hubungan:

- a. Melakukan: Pelanggan melakukan Pemesanan.
- b. Memesan: Pemesanan memuat Produk.

3. Kardinalitas:

- a. Seorang pelanggan dapat membuat banyak pemesanan (1:N).
- b. Sebuah pemesanan dapat berisi banyak produk, dan produk dapat ada di banyak pemesanan (M:N).



Gambar 3. 4 Contoh ER Diagram

- Pelanggan berhubungan dengan Pemesanan dalam hubungan *One-to-Many* (1:N), yang berarti seorang pelanggan dapat melakukan banyak pemesanan.
- Pemesanan berhubungan dengan Produk dalam hubungan *Many-to-Many* (M:N), yang berarti sebuah pemesanan dapat berisi banyak produk, dan produk dapat ada di banyak pemesanan.

ER Diagram adalah alat yang sangat berguna untuk mendesain dan merencanakan struktur basis data secara visual, sehingga lebih mudah dipahami dan diterapkan dalam sistem basis data yang sebenarnya.

B. Notasi Barker

Notasi Barker adalah salah satu teknik untuk menggambarkan model ER. Notasi ini dikenal dengan kesederhanaannya dan digunakan secara luas dalam pemodelan basis data relasional.

Notasi Barker adalah salah satu jenis notasi yang digunakan dalam Diagram Entitas-Relasi (ER Diagram) untuk menggambarkan hubungan antara entitas dalam sebuah sistem basis data. Notasi ini dikembangkan oleh Richard Barker dan digunakan dalam banyak metode desain basis data untuk memberikan representasi yang jelas dan konsisten mengenai entitas, atribut, dan hubungan antar entitas.

1. Komponen Utama dalam Notasi Barker

Notasi Barker memiliki cara khas untuk menggambarkan elemen-elemen ER Diagram, seperti entitas, atribut, dan hubungan. Berikut adalah penjelasan tentang komponen-komponen utama dalam notasi ini:

a. Entitas (*Entity*):

- 1) Entitas digambarkan dengan persegi panjang.
- 2) Nama entitas ditulis di dalam persegi panjang.
- 3) Entitas bisa menjadi entitas kuat (yang memiliki identitas yang dapat berdiri sendiri) atau entitas lemah.

b. Atribut (*Attribute*):

- 1) Atribut digambarkan dengan *elips* atau lingkaran yang terhubung dengan entitas.
- 2) Atribut dapat dibagi menjadi beberapa jenis, seperti:
 - a) Atribut Kunci (*Primary Attribute*): Atribut yang digunakan untuk mengidentifikasi entitas secara unik, biasanya digarisbawahi.
 - b) Atribut Terhitung (*Derived Attribute*): Atribut yang nilainya dihitung berdasarkan atribut lain.
 - c) Atribut *Multi-valued*: Atribut yang dapat memiliki lebih dari satu nilai untuk satu entitas (digambarkan dengan *elips ganda*).

c. Hubungan (*Relationship*):

- 1) Hubungan antara entitas digambarkan dengan rombus (*lozenge*).
- 2) Nama hubungan ditulis di dalam rombus.
- 3) Hubungan bisa memiliki kardinalitas yang menggambarkan berapa banyak entitas yang terlibat dalam hubungan tersebut. Kardinalitas biasanya ditunjukkan dengan angka di dekat garis hubungan:
 - a) 1 (*One*) menunjukkan hubungan satu ke satu (*One-to-One*).

- b) N (*Many*) menunjukkan hubungan banyak ke banyak (*Many-to-Many*).
 - c) 1..N menunjukkan hubungan satu ke banyak (*One-to-Many*).
 - d. Kardinalitas (*Cardinality*):
 - 1) Kardinalitas dalam notasi Barker sering kali ditunjukkan dengan angka-angka di sepanjang garis yang menghubungkan entitas dengan hubungan.
 - 2) 1 di satu sisi dan N di sisi lain menunjukkan hubungan satu ke banyak (1:N).
 - 3) 1 di kedua sisi menunjukkan hubungan satu ke satu (1:1).
 - 4) N di kedua sisi menunjukkan hubungan banyak ke banyak (M:N).
 - e. Entitas Lemah (*Weak Entity*) dan Hubungan Pengidentifikasi (*Identifying Relationship*):
 - 1) Entitas Lemah digambarkan dengan persegi panjang ganda.
 - 2) Hubungan Pengidentifikasi digambarkan dengan rombus ganda.
- 2. Contoh Notasi Barker:** Misalnya, kita ingin menggambarkan hubungan antara entitas Pelanggan, Pemesanan, dan Produk dalam sebuah toko online.

a. Entitas:

- 1) Pelanggan: ID_Pelanggan, Nama, Alamat
- 2) Pemesanan: ID_Pemesanan, Tanggal_Pemesanan
- 3) Produk: ID_Produk, Nama_Produk, Harga

b. Hubungan:

- 1) Melakukan: Hubungan antara Pelanggan dan Pemesanan.
- 2) Memuat: Hubungan antara Pemesanan dan Produk.



Gambar 3. 5 Contoh dengan Notasi Barker

Penjelasan:

- a. Pelanggan dan Pemesanan terhubung dalam hubungan Melakukan dengan kardinalitas 1:N, yang berarti satu pelanggan dapat melakukan banyak pemesanan.
- b. Pemesanan dan Produk terhubung dalam hubungan Memuat dengan kardinalitas M:N, yang berarti banyak produk dapat dimuat dalam banyak pemesanan, dan sebaliknya.

Kelebihan Notasi Barker:

- a. Sederhana dan Mudah Dibaca: Notasi Barker lebih sederhana dan mudah dipahami, terutama untuk orang yang baru belajar tentang diagram entitas-relasi.
- b. Mendukung Kardinalitas dengan Jelas: Notasi ini mendukung representasi kardinalitas yang jelas, yang memudahkan pemahaman hubungan antar entitas.
- c. Cocok untuk Desain Basis Data: Notasi Barker banyak digunakan dalam perancangan sistem basis data untuk menggambarkan struktur data dengan lebih efisien.

DAFTAR PUSTAKA

- Barker, R. (1990). *Entity-relationship modeling: A guide for database designers*. Addison-Wesley Longman Publishing Co., Inc.
- Chen, P. P. (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9–36.
- Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of database systems* (7th ed.). Pearson.
- Kroenke, D. M. (2013). *Database processing: Fundamentals, design, and implementation* (13th ed.). Pearson.
- Rob, P., & Coronel, C. (2007). *Database systems: Design, implementation, and management* (8th ed.). Cengage Learning.
- Teorey, T. J., Yang, J., & Fry, J. P. (2011). *Database modeling and design: Logical design* (5th ed.). Morgan Kaufmann.

BAB 4

RELASI LOGIKA *SUPERTYPE*, *SUBTYPE*, *INHERITANCE*, *GENERALIZATION*, DAN *SPECIALIZATION* DENGAN NOTASI BARKER

Oleh : Dr. Mohamad Ilyas Abas, S.SI, M.Kom

Relasi logika seperti *supertype*, *subtype*, *inheritance*, *generalization*, dan *specialization* merupakan fondasi penting dalam desain basis data modern. Pemahaman mendalam mengenai konsep-konsep ini membantu menciptakan model data yang lebih efisien, terstruktur, dan mudah dipahami. Bab ini akan mengeksplorasi setiap konsep tersebut secara rinci, dilengkapi dengan contoh implementasi menggunakan notasi Barker sebagai alat representasi visual yang efisien.

A. Supertype dan Subtype

1. Definisi

Supertype adalah entitas generik yang mencakup atribut atau karakteristik umum yang dimiliki oleh beberapa entitas lain yang lebih spesifik. *Supertype* adalah entitas yang mewakili konsep yang lebih umum atau abstrak, yang memiliki atribut atau karakteristik yang dimiliki bersama oleh beberapa entitas lain yang lebih spesifik, yang dikenal sebagai *subtype*. Dalam pemodelan basis data, supertype berfungsi untuk menyatukan atribut-atribut umum yang relevan untuk beberapa entitas yang memiliki kesamaan, tetapi juga memungkinkan spesifikasi lebih lanjut untuk entitas-entitas yang lebih khusus melalui *subtype*. *Supertype* tidak selalu mengandung data yang lengkap, tetapi lebih berfokus pada aspek-aspek umum yang relevan untuk kelompok entitas yang lebih luas. Misalnya, dalam sebuah sistem basis data untuk rumah sakit, kita dapat memiliki entitas supertype yang bernama *Person*. Entitas *Person* ini akan mencakup atribut umum seperti *Name* (Nama), *Date of Birth* (Tanggal Lahir), dan *Gender* (Jenis Kelamin), yang relevan untuk semua individu yang terlibat dalam sistem, baik itu seorang *Doctor*, *Nurse*, *Patient*, atau *Staff*. Semua entitas tersebut memiliki beberapa kesamaan, seperti nama, tanggal lahir, dan jenis kelamin, sehingga informasi ini cukup untuk disimpan di *supertype Person*.

Namun, dalam hal ini, *Doctor*, *Nurse*, dan *Patient* adalah contoh *subtype* dari *Person*, di mana setiap *subtype* memiliki atribut tambahan yang lebih spesifik sebagai contoh:

- a. *Doctor* bisa memiliki atribut tambahan seperti *Specialization* (Spesialisasi), *License Number* (Nomor Lisensi), atau *Years of Experience* (Tahun Pengalaman).
- b. *Patient* bisa memiliki atribut tambahan seperti *Medical History* (Riwayat Medis) atau *Insurance Number* (Nomor Asuransi).
- c. *Nurse* bisa memiliki atribut seperti *Shift Timing* (Jadwal Shift) atau *Ward Assigned* (Ruangan Tugas).

Dengan cara ini, *supertype* mengurangi redundansi data, karena atribut umum hanya perlu disimpan satu kali di entitas *Person*, sementara entitas *subtype* menyimpan data yang lebih spesifik terkait masing-masing peran dalam sistem. Selain itu, *supertype* memungkinkan fleksibilitas dalam pengelolaan data, karena entitas-entitas yang lebih spesifik (*subtypes*) dapat berbagi struktur umum dari *supertype*, namun tetap mempertahankan karakteristik dan kebutuhan uniknya masing-masing. Jika ada perubahan dalam atribut umum, hanya *supertype* yang perlu diperbarui, dan semua *subtype* yang mewarisi atribut tersebut akan langsung tercakup oleh perubahan tersebut, yang meningkatkan efisiensi dan konsistensi dalam basis data. Secara

keseluruhan, *supertype* membantu merancang basis data yang lebih terstruktur, efisien, dan dapat diskalakan dengan baik, mengurangi duplikasi data dan memastikan hubungan antar entitas lebih mudah dikelola.

Subtype adalah entitas yang lebih spesifik, mewarisi atribut dari *supertype*, tetapi juga memiliki atribut unik yang membedakannya dari subtype lainnya. *Subtype* adalah entitas yang lebih spesifik dalam sebuah hirarki entitas, yang mewarisi atribut dari *supertype*, tetapi juga memiliki atribut tambahan atau unik yang membedakannya dari subtype lainnya. Dalam konteks basis data, *subtype* digunakan untuk merepresentasikan perbedaan yang lebih mendetail dalam entitas yang lebih luas, yang sudah digambarkan oleh *supertype*. Meskipun semua subtype berbagi atribut umum dari *supertype*, masing-masing subtype dapat menambahkan informasi yang lebih spesifik yang hanya relevan untuk mereka, tanpa mempengaruhi entitas lain yang ada dalam hirarki. Untuk menjelaskan lebih lanjut, mari kita lihat contoh penggunaan *supertype* dan *subtype* dalam konteks sistem rumah sakit.

Contoh Kasus: Sistem Rumah Sakit

Misalkan kita memiliki *supertype* bernama Person, yang berfungsi untuk menyimpan atribut umum yang dimiliki oleh semua orang

yang terlibat dalam sistem rumah sakit, seperti *Name* (Nama), *Date of Birth* (Tanggal Lahir), dan *Gender* (Jenis Kelamin). Atribut-atribut ini berlaku untuk semua individu dalam rumah sakit, tidak peduli apakah mereka adalah dokter, perawat, atau pasien. Namun, sistem rumah sakit ini juga membutuhkan informasi lebih spesifik tentang jenis orang yang terlibat. Inilah alasan mengapa kita menggunakan *subtype* untuk membedakan entitas berdasarkan peran mereka, dengan menambahkan atribut yang lebih relevan.

Contoh Subtype 1: *Doctor*

Doctor adalah *subtype* dari *Person* yang mewarisi atribut umum seperti *Name*, *Date of Birth*, dan *Gender*, tetapi juga memiliki atribut yang lebih khusus, seperti:

- a. *Specialization* (Spesialisasi), misalnya dokter umum, spesialis bedah, spesialis jantung.
- b. *License Number* (Nomor Lisensi), sebagai identifikasi profesional.
- c. *Years of Experience* (Tahun Pengalaman), untuk menunjukkan pengalaman dalam praktik medis.

Meskipun *Doctor* mewarisi atribut umum dari *Person*, atribut-atribut seperti *Specialization* dan *License Number* hanya relevan

untuk dokter, dan tidak akan digunakan oleh entitas lain seperti perawat atau pasien.

Contoh *Subtype 2: Patient*

Patient juga merupakan *subtype* dari *Person*, tetapi dengan atribut yang berbeda, seperti:

- a. *Medical History* (Riwayat Medis), yang berisi informasi tentang kondisi medis pasien sebelumnya.
- b. *Insurance Number* (Nomor Asuransi), untuk administrasi klaim asuransi.
- c. *Admission Date* (Tanggal Masuk), untuk mencatat kapan pasien masuk rumah sakit.

Atribut-atribut ini sangat relevan untuk pasien, tetapi tidak diperlukan untuk entitas lain seperti *Doctor* atau *Nurse*.

Contoh *Subtype 3: Nurse*

Nurse adalah *subtype* lain dari *Person*, yang memiliki atribut spesifik seperti:

- a. *Shift Timing* (Jadwal *Shift*), yang menunjukkan waktu atau periode kerja perawat.
- b. *Ward Assigned* (Ruangan Tugas), yang mencatat ruangan atau unit rumah sakit tempat perawat bekerja.

Atribut-atribut ini penting untuk perawat, tetapi tidak relevan untuk *Doctor* atau *Patient*.

Karakteristik Subtype

- a. Pewarisan Atribut dari *Supertype*: Setiap *subtype* mewarisi atribut dari *supertype*. Dalam contoh di atas, *Doctor*, *Patient*, dan *Nurse* semuanya mewarisi atribut *Name*, *Date of Birth*, dan *Gender* dari *Person*. Ini menghindari redundansi data, karena tidak perlu menyimpan informasi umum tersebut di setiap *subtype* secara terpisah.
- b. Atribut Unik untuk Setiap *Subtype*: Setiap *subtype* menambahkan atribut yang hanya relevan untuk mereka. *Doctor* menambahkan *Specialization* dan *License Number*, sementara *Patient* menambahkan *Medical History* dan *Insurance Number*. Atribut-atribut unik ini membedakan setiap *subtype* dan memungkinkan mereka menyimpan data yang lebih spesifik sesuai dengan peran mereka.
- c. Pengelompokan yang Lebih Terstruktur: Dengan menggunakan *subtype*, data yang lebih spesifik dikelompokkan dengan lebih baik, yang meningkatkan keterbacaan dan pemeliharaan basis data. Atribut yang relevan hanya ada di entitas yang tepat, dan pengelolaan data lebih efisien.

- d. Keterkaitan Antar *Subtype*: Meskipun setiap *subtype* memiliki atribut yang berbeda, mereka tetap terkait melalui supertype yang lebih umum. Hal ini memungkinkan pemrogram atau perancang basis data untuk menangani data lebih terstruktur dan tidak terpisah-pisah, karena ada relasi yang jelas antara *supertype* dan *subtype*.

Manfaat Menggunakan *Subtype*

- a. Pengurangan Redundansi: *Subtype* menghindari duplikasi data dengan mewarisi atribut dari supertype yang sama, sehingga menghemat ruang penyimpanan.
- b. Fleksibilitas dalam Penambahan Atribut: Setiap *subtype* dapat dengan mudah dikembangkan untuk memasukkan atribut baru yang relevan dengan perubahan dalam kebutuhan sistem tanpa mengubah struktur dasar *supertype*.
- c. Peningkatan Kejelasan dan Organisasi: Data lebih terorganisir dengan jelas, karena setiap entitas memiliki atribut yang relevan dan dapat dipisahkan berdasarkan kategori yang lebih spesifik.

Secara keseluruhan, *subtype* memungkinkan desain basis data yang lebih fleksibel, efisien, dan mudah dikelola, dengan memberikan struktur yang jelas untuk data yang bervariasi.

2. Karakteristik *Supertype* dan *Subtype*

- a. Atribut Bersama: Semua atribut yang umum diletakkan di *supertype*.
- b. Atribut Khusus: Setiap subtype memiliki atribut tambahan yang hanya relevan untuk entitas tersebut.
- c. Diskriminator: Elemen yang menentukan jenis subtype dari sebuah supertype.

Konsep *supertype* dan *subtype* dalam pemodelan basis data berfungsi untuk menyusun entitas berdasarkan kesamaan dan perbedaan. *Supertype* menyatukan atribut yang bersifat umum, sedangkan *subtype* menangani atribut yang lebih spesifik dan unik. Berikut adalah penjelasan lebih detail mengenai karakteristik utama dari *supertype* dan *subtype*:

a. Atribut Bersama (*Shared Attributes*)

Atribut bersama merujuk pada atribut yang bersifat umum dan dimiliki oleh beberapa entitas yang lebih spesifik dalam model *supertype-subtype*. Atribut-atribut ini disimpan di *supertype*, karena atribut tersebut relevan untuk semua subtype yang ada. Hal ini menghindari redundansi data dan memastikan konsistensi dalam struktur basis data. Contoh: Misalnya, dalam sistem rumah sakit, kita dapat memiliki *supertype* yang disebut *Person*, yang memiliki atribut umum seperti:

- 1) *Name* (Nama)
- 2) *Date of Birth* (Tanggal Lahir)
- 3) *Gender* (Jenis Kelamin)
- 4) *Address* (Alamat)
- 5) *Phone Number* (Nomor Telepon)

Atribut-atribut ini dimiliki oleh semua entitas yang lebih spesifik dalam sistem rumah sakit, baik itu dokter, perawat, pasien, atau staf rumah sakit lainnya. Dengan menempatkan atribut-atribut ini di *supertype* Person, kita menghindari pengulangan data yang sama di setiap entitas spesifik.

b. Atribut Khusus (*Specific Attributes*)

Setiap *subtype* memiliki atribut tambahan yang hanya relevan untuk jenis entitas tertentu. Atribut-atribut ini tidak dimiliki oleh *supertype* atau *subtype* lainnya, karena mereka menggambarkan informasi yang unik bagi entitas tersebut. Atribut khusus ini memberikan karakteristik yang membedakan antara satu *subtype* dengan yang lainnya. Contoh: Melanjutkan contoh sebelumnya, kita memiliki beberapa *subtype* dari Person, seperti *Doctor*, *Nurse*, dan *Patient*, dan masing-masing memiliki atribut yang hanya relevan untuk mereka.

- 1) *Doctor* (Dokter) mungkin memiliki atribut khusus seperti:
 - a) *Specialization* (Spesialisasi) – Misalnya, dokter bedah atau dokter umum.
 - b) *License Number* (Nomor Lisensi) – Nomor lisensi profesional dokter.
 - c) *Years of Experience* (Tahun Pengalaman) – Pengalaman kerja dalam bidang medis.
- 2) *Nurse* (Perawat) mungkin memiliki atribut seperti:
 - a) *Ward Assigned* (Ruangan Tugas)–Ruangan rumah sakit tempat perawat bertugas.
 - b) *Shift Timing* (Jadwal Shift)–Waktu kerja perawat, misalnya shift pagi atau malam.
- 3) *Patient* (Pasien) mungkin memiliki atribut seperti:
 - a) *Medical History* (Riwayat Medis)–Riwayat kesehatan pasien.
 - b) *Admission Date* (Tanggal Masuk)–Tanggal pasien dirawat di rumah sakit.

Atribut-atribut ini bersifat spesifik dan hanya relevan untuk subtype masing-masing. Dengan demikian, data untuk *Doctor*, *Nurse*, dan *Patient* tetap terstruktur secara terpisah dan tidak tercampur, meskipun mereka semua merupakan turunan dari *supertype* Person.

c. Diskriminator (*Discriminator*)

Diskriminator adalah elemen penting dalam model *supertype-subtype*, yang berfungsi untuk menentukan atau membedakan jenis *subtype* yang tepat dari sebuah *supertype*. Dalam kata lain, discriminator membantu sistem untuk mengetahui entitas mana yang harus dipilih berdasarkan nilai yang ada pada *discriminator*. Diskriminator seringkali berupa atribut atau kolom yang ada di dalam *supertype* yang menunjukkan tipe entitas spesifik yang ada. Nilai dari *discriminator* ini digunakan untuk memutuskan *subtype* mana yang akan diterapkan pada suatu instance data. Contoh: Misalnya, dalam sistem rumah sakit, kita memiliki supertype Person dengan atribut *Person Type* sebagai *discriminator*, yang menentukan apakah entitas tersebut adalah *Doctor*, *Nurse*, atau *Patient*. Atribut *Person Type* ini mungkin memiliki nilai-nilai seperti:

- 1) *Doctor* untuk entitas yang merupakan seorang dokter.
- 2) *Nurse* untuk entitas yang merupakan seorang perawat.
- 3) *Patient* untuk entitas yang merupakan seorang pasien.

Dengan menggunakan nilai dari *Person Type* (misalnya, "*Doctor*"), sistem dapat mengetahui bahwa entitas tersebut harus dihubungkan dengan *Doctor* sebagai *subtype* dan akan memiliki atribut tambahan yang relevan untuk dokter, seperti *Specialization* dan *License Number*.

Ringkasan Karakteristik:

- a. Atribut Bersama (*Shared Attributes*) – Atribut yang dimiliki bersama oleh semua subtype, dan diletakkan di supertype untuk menghindari redundansi data. Contoh: Nama, Tanggal Lahir, Jenis Kelamin.
- b. Atribut Khusus (*Specific Attributes*) – Atribut tambahan yang hanya relevan untuk subtype tertentu dan tidak diwarisi oleh subtype lain. Contoh: Spesialisasi (untuk *Doctor*), Riwayat Medis (untuk *Patient*).
- c. Diskriminator–Elemen yang menentukan jenis subtype dari sebuah *supertype*. Biasanya berupa atribut dalam supertype yang menyatakan tipe dari entitas tersebut, seperti *Person Type* yang membedakan antara *Doctor*, *Nurse*, dan *Patient*.

Manfaat Penggunaan Karakteristik *Supertype* dan *Subtype*

- a. Efisiensi Penyimpanan: Mengurangi redundansi data dengan menyimpan atribut bersama di *supertype* dan hanya menambahkan atribut yang relevan di *subtype*.
- b. Fleksibilitas: Memungkinkan penambahan atribut baru yang spesifik untuk subtype tertentu tanpa mengubah struktur data yang ada di *supertype*.
- c. Pengelolaan Data yang Lebih Terstruktur: Memberikan organisasi yang lebih baik dalam basis data dengan

memisahkan data yang bersifat umum dan data yang spesifik, sekaligus menjaga hubungan antar entitas dengan jelas.

3. Contoh Kasus

Bayangkan sebuah sistem yang mencatat data tentang orang ("*Person*") yang bisa menjadi "*Student*" atau "*Teacher*".

a. *Supertype*: Person

Atribut: *PersonID*, *Name*, *Address*

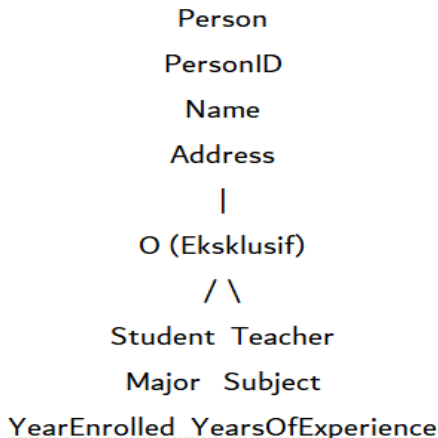
b. *Subtype*:

1) *Student*: memiliki atribut tambahan *Major* dan *Year Enrolled*.

2) *Teacher*: memiliki atribut tambahan *Subject* dan *Years Of Experience*.

Representasi dengan Notasi Barker

Dalam notasi Barker, hubungan antara *supertype* dan *subtype* divisualisasikan dengan hierarki sederhana:



Penjelasan: Diagram menunjukkan bahwa setiap "*Person*" hanya dapat menjadi salah satu dari "*Student*" atau "*Teacher*". Hubungan eksklusif ini dijelaskan melalui simbol lingkaran diskriminator.

Contoh Kasus Tambahan

1. Sistem Manajemen Perusahaan:

a. *Supertype: Employee*

Atribut: *EmployeeID*, *Name*, *HireDate*

b. *Subtypes:*

a. *Manager*: atribut tambahan *DepartmentManaged*

b. *Technician*: atribut tambahan *SkillSet*

c. *Clerk*: atribut tambahan *OfficeLocation*

Representasi:

Employee

EmployeeID

Name

HireDate

/

O (Eksklusif)

// \

Manager Technician Clerk

Department SkillSet Office Location

2. Sistem Transportasi:

- a. *Supertype*: Transport
 - a. Atribut: *TransportID*, *Capacity*
- b. *Subtypes*:
 - a. Car: atribut tambahan *CarType*
 - b. *Train*: atribut tambahan *NumberOfCarriages*
 - c. *Plane*: atribut tambahan *WingSpan*

Representasi:

Transport

TransportID

Capacity

|

O (Eksklusif)

/ | \

Car Train Plane

CarType NumberOfCarriages WingSpan

B. Inheritance

1. Definisi *Inheritance*

Inheritance adalah mekanisme pewarisan atribut dan hubungan dari *supertype* ke *subtype*. Tujuan utama *inheritance* adalah untuk mendukung prinsip pengelompokan ulang atribut yang digunakan berulang kali.

Inheritance adalah mekanisme penting dalam model basis data yang memungkinkan entitas *subtype* mewarisi atribut dan hubungan dari entitas *supertype*. Konsep ini membantu mengelola data yang serupa namun dengan variasi spesifik, dan memungkinkan pengelompokan ulang atribut yang digunakan berulang kali. Dengan cara ini, data yang umum hanya perlu disimpan di satu tempat, sedangkan entitas yang lebih spesifik dapat menambahkan informasi tambahan sesuai kebutuhan. Dalam pemodelan basis data, *supertype* adalah entitas yang lebih umum atau abstrak, sementara *subtype* adalah entitas yang lebih spesifik. Melalui *inheritance*, *subtype* mewarisi atribut dan hubungan yang dimiliki oleh *supertype*. Setiap *subtype* dapat menambahkan atribut tambahan atau hubungan yang unik sesuai dengan karakteristik spesifik mereka. Contoh: Misalkan kita memiliki sebuah model basis data yang menangani sistem rumah sakit, dan kita memiliki *supertype Person* yang mencakup atribut-atribut seperti:

- a. *Name* (Nama)
- b. *Date of Birth* (Tanggal Lahir)
- c. *Gender* (Jenis Kelamin)

Kemudian kita memiliki beberapa subtype, seperti *Doctor*, *Nurse*, dan *Patient*, yang masing-masing mewarisi atribut dasar

dari supertype Person, tetapi juga memiliki atribut tambahan yang spesifik:

- a. *Doctor* bisa memiliki atribut tambahan seperti *Specialization* (Spesialisasi), *License Number* (Nomor Lisensi), dan *Years of Experience* (Tahun Pengalaman).
- b. *Nurse* bisa memiliki atribut seperti *Ward Assigned* (Ruangan Tugas), *Shift Timing* (Jadwal Shift), dan lain-lain.
- c. *Patient* bisa memiliki atribut seperti *Medical History* (Riwayat Medis), *Insurance Number* (Nomor Asuransi), dan lainnya.

Dengan *inheritance*, setiap subtype mewarisi atribut-atribut umum dari Person, yang kemudian memungkinkan mereka untuk menambahkan atribut tambahan sesuai dengan peran spesifik mereka. Tujuan *Inheritance* : Tujuan utama dari *inheritance* dalam basis data adalah untuk mendukung prinsip pengelompokan ulang atribut yang digunakan berulang kali. Hal ini berkontribusi pada beberapa hal penting, yaitu:

a. Pengurangan Redundansi Data

Dengan mewarisi atribut-atribut umum yang berlaku untuk semua subtype dari supertype, kita menghindari duplikasi data. Misalnya, atribut *Name*, *Date of Birth*, dan *Gender* hanya perlu disimpan sekali di entitas supertype Person, dan kemudian diwarisi oleh semua *subtype* (*Doctor*, *Nurse*, *Patient*). Ini

mengurangi jumlah penyimpanan yang diperlukan, serta memastikan data konsisten dan terhindar dari inkonsistensi.

b. Fleksibilitas dan Skalabilitas

Inheritance memberikan fleksibilitas dalam pengelolaan entitas yang serupa namun memiliki variasi spesifik. Ketika atribut baru perlu ditambahkan, cukup dilakukan di satu tempat (*subtype* atau *supertype*), dan semua entitas yang mewarisinya akan menerima perubahan tersebut secara otomatis. Ini mempermudah pemeliharaan basis data dan meningkatkan skalabilitas sistem.

c. Konsistensi Data

Dengan mewarisi atribut dan hubungan yang sama, data tetap konsisten di seluruh entitas yang serupa. Misalnya, jika perubahan dibuat pada atribut umum seperti *Gender*, perubahan ini akan berlaku untuk semua entitas dalam subkategori tersebut. Konsistensi seperti ini sangat penting terutama dalam sistem yang membutuhkan integrasi dan akurasi data.

d. Peningkatan Relasi dan Integrasi Data

Inheritance memungkinkan hubungan yang lebih terstruktur antar entitas. Misalnya, setiap *subtype* akan terkait langsung dengan *supertype*-nya, yang memudahkan proses pengelolaan

data dan akses informasi terkait. Hal ini sangat berguna dalam berbagai sistem seperti sistem rumah sakit, sistem pendidikan, sistem keuangan, dan lain-lain.

Manfaat *Inheritance* : secara keseluruhan, *inheritance* memberikan keuntungan besar dalam perancangan basis data yang lebih terstruktur, efisien, dan dapat disesuaikan dengan kebutuhan yang terus berubah. Beberapa manfaat utama dari penggunaan *inheritance* dalam basis data meliputi:

- a. Reduksi redundansi data, sehingga meningkatkan efisiensi penyimpanan.
- b. Fleksibilitas dalam menambahkan atribut dan perubahan, tanpa perlu merevisi seluruh struktur basis data.
- c. Konsistensi dan integrasi data, yang memungkinkan pengelolaan data yang lebih terorganisir.
- d. Peningkatan keterbacaan dan pemeliharaan basis data, dengan struktur yang lebih terorganisir dan pengelompokan data yang logis.

Dengan menerapkan *inheritance* dengan benar, basis data yang dirancang akan lebih tangguh, lebih mudah dikelola, dan dapat diskalakan dengan kebutuhan bisnis yang berkembang.

2. Proses *Inheritance*

- a. Identifikasi Atribut Umum: Semua atribut yang digunakan bersama diletakkan di *supertype*.
- b. Turunan Atribut: *Subtype* mewarisi atribut dari *supertype* dan dapat menambahkan atribut khususnya sendiri.

Proses *inheritance* dalam desain basis data melibatkan dua langkah utama: identifikasi atribut umum dan turunan atribut. Proses ini bertujuan untuk menyusun entitas dalam hierarki yang memungkinkan pengelolaan data yang lebih efisien dengan mengelompokkan atribut-atribut yang digunakan berulang kali, dan memungkinkan penambahan atribut-atribut khusus yang relevan untuk masing-masing entitas.

1) Identifikasi Atribut Umum

Langkah pertama dalam proses *inheritance* adalah identifikasi atribut umum yang digunakan bersama oleh beberapa entitas dalam sistem. Atribut umum ini merujuk pada informasi yang berlaku untuk seluruh entitas dalam *supertype*, yang kemudian akan diwariskan ke semua *subtype* yang ada. Dengan menempatkan atribut yang digunakan bersama ini dalam *supertype*, kita dapat menghindari redundansi data dan mengurangi inkonsistensi yang mungkin terjadi jika atribut yang sama disalin di berbagai tempat.

Langkah-Langkah Identifikasi Atribut Umum:

- a. Analisis kebutuhan data: Pertama, tentukan data apa yang digunakan oleh banyak entitas yang memiliki kesamaan. Atribut yang dibutuhkan oleh semua entitas dalam hirarki *supertype-subtype* akan ditempatkan di *supertype*. Misalnya, jika kita memiliki entitas Person, atribut umum seperti *Name*, *Date of Birth*, *Gender*, dan *Address* adalah atribut yang relevan untuk semua entitas yang terlibat, apakah itu *Doctor*, *Patient*, atau *Nurse*.
- b. Menentukan atribut yang berlaku umum: Atribut yang berlaku umum biasanya berhubungan dengan karakteristik dasar atau identitas entitas. Contoh dalam dunia rumah sakit, atribut *Name*, *Address*, *Phone Number*, dan *Gender* adalah elemen yang sama di semua entitas yang terlibat dalam sistem rumah sakit.
- c. Menghindari redundansi: Dengan menempatkan atribut umum ini di *supertype*, kita memastikan bahwa informasi yang digunakan berulang kali hanya perlu disimpan satu kali. Misalnya, atribut *Name* hanya disimpan di Person dan tidak perlu diulang di entitas *Doctor*, *Patient*, atau Nurse.
Contoh: Misalnya, dalam basis data rumah sakit, kita memiliki supertype Person dengan atribut-atribut umum seperti:

- 1) *Name* (Nama)
- 2) *Date of Birth* (Tanggal Lahir)
- 3) *Gender* (Jenis Kelamin)
- 4) *Address* (Alamat)
- 5) *Phone Number* (Nomor Telepon)

Atribut-atribut ini akan diwariskan oleh semua *subtype* dari *Person*.

2) Turunan Atribut (*Inheritance* dari *Supertype* ke *Subtype*)

Setelah atribut umum diidentifikasi dan diletakkan di *supertype*, langkah berikutnya adalah turunan atribut, yang berarti bahwa *subtype* mewarisi atribut-atribut dari *supertype* dan dapat menambahkan atribut tambahan yang hanya relevan untuk entitas tersebut. Masing-masing *subtype* memiliki karakteristik uniknya, dan di sinilah *inheritance* memungkinkan *subtype* untuk mengembangkan dan menyesuaikan diri dengan kebutuhan spesifik mereka.

Langkah-Langkah Turunan Atribut:

- a. Pewarisan Atribut: Atribut yang sudah ada di *supertype* secara otomatis diwariskan oleh *subtype*. Sebagai contoh, dalam sistem rumah sakit, semua *Doctor*, *Patient*, dan *Nurse* akan mewarisi atribut *Name*, *Date of Birth*, dan *Gender* dari *Person*,

karena atribut-atribut tersebut relevan untuk setiap individu di rumah sakit, terlepas dari peran atau fungsinya.

- b. Penambahan Atribut Khusus: Setiap subtype dapat menambahkan atribut tambahan yang relevan hanya untuk entitas tersebut. Misalnya, *Doctor* bisa memiliki atribut seperti *Specialization* (Spesialisasi), *License Number* (Nomor Lisensi), dan *Years of Experience* (Tahun Pengalaman). Atribut-atribut ini tidak relevan untuk *Patient* atau *Nurse*, jadi hanya ada pada *Doctor*.
- c. Hubungan dengan *Supertype*: Selain atribut, *subtype* juga dapat mewarisi hubungan dari *supertype*. Ini memungkinkan entitas yang lebih spesifik untuk mempertahankan hubungan yang relevan dengan entitas lainnya. Misalnya, *Person* bisa memiliki hubungan dengan entitas lain seperti *Hospital* atau *Appointment*, dan hubungan ini akan diwariskan ke semua *subtype*. Dengan cara ini, kita dapat menjaga integritas data dan relasi antar entitas dalam basis data. Contoh:
 - 1) *Doctor* adalah *subtype* dari *Person* yang mewarisi atribut *Name*, *Date of Birth*, dan *Gender*. Namun, *Doctor* juga memiliki atribut spesifik, seperti *Specialization*, *License Number*, dan *Years of Experience*.

- 2) *Patient* adalah *subtype* dari *Person* yang mewarisi atribut *Name*, *Date of Birth*, dan *Gender*, tetapi memiliki atribut spesifik seperti *Medical History*, *Insurance Number*, dan *Admission Date*.
- 3) *Nurse* adalah *subtype* dari *Person* yang mewarisi atribut *Name*, *Date of Birth*, dan *Gender*, dengan atribut tambahan seperti *Ward Assigned* (Ruangan Tugas) dan *Shift Timing* (Jadwal *Shift*).

Proses Lengkap *Inheritance*.

- a. Identifikasi Atribut Umum: Atribut yang berlaku untuk semua entitas dalam hirarki *supertype-subtype* ditempatkan di *supertype*, mengurangi redundansi dan menjaga konsistensi data.
- b. Turunan Atribut: Setiap *subtype* mewarisi atribut umum dari *supertype* dan menambahkan atribut khusus yang relevan hanya untuk entitas tersebut, memungkinkan pengelolaan data yang lebih fleksibel dan spesifik.

Manfaat *Inheritance*.

- 1) Efisiensi Penyimpanan dan Pengelolaan Data: Dengan menempatkan atribut umum di *supertype*, kita menghindari pengulangan data di setiap *subtype*. Selain itu, perubahan pada

atribut umum hanya perlu dilakukan di *supertype* dan secara otomatis tercermin pada semua *subtype*.

- 2) **Fleksibilitas:** Setiap *subtype* dapat menambahkan atribut dan hubungan tambahan yang unik, yang memungkinkan sistem untuk berkembang dan disesuaikan dengan kebutuhan spesifik.
- 3) **Konsistensi dan Integrasi:** Mewarisi atribut dan hubungan dari *supertype* memungkinkan data tetap konsisten, dan hubungan antar entitas tetap terjaga secara integratif.

Secara keseluruhan, proses *inheritance* memungkinkan desain basis data yang lebih terstruktur, modular, dan mudah dipelihara, dengan memberikan fleksibilitas dalam menambahkan atribut dan hubungan yang relevan bagi entitas yang lebih spesifik tanpa mengorbankan konsistensi dan integritas data.

3. Contoh Kasus

Sebuah sistem yang mencatat kendaraan dapat memiliki:

a. *Supertype: Vehicle*

Atribut: *VehicleID*, *LicensePlate*

b. *Subtype:*

1) *Car*: memiliki atribut tambahan *CarType*

2) *Truck*: memiliki atribut tambahan *TruckCapacity*

Representasi dengan Notasi Barker

Vehicle

VehicleID

LicensePlate

|

O (Inklusif)

/ \

Car Truck

CarType TruckCapacity

Penjelasan: Semua kendaraan memiliki *VehicleID* dan *LicensePlate*.
Subtype Car dan *Truck* mewarisi atribut ini, tetapi juga memiliki atribut unik masing-masing.

4. Contoh Kasus Tambahan

a. Sistem Inventaris Barang:

1) *Supertype: Product*

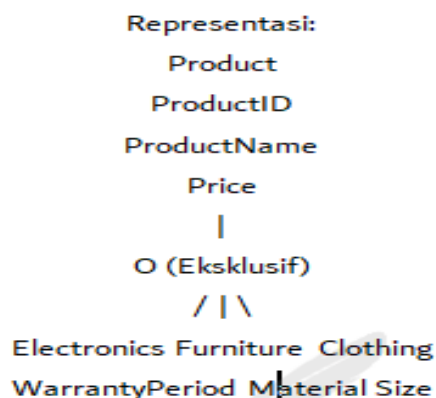
Atribut: *ProductID*, *ProductName*, *Price*

2) *Subtypes:*

a) *Electronics*: atribut tambahan *WarrantyPeriod*

b) *Furniture*: atribut tambahan *Material*

c) *Clothing*: atribut tambahan *Size*



b. Sistem Pemesanan Tiket:

1) *Supertype: Ticket*

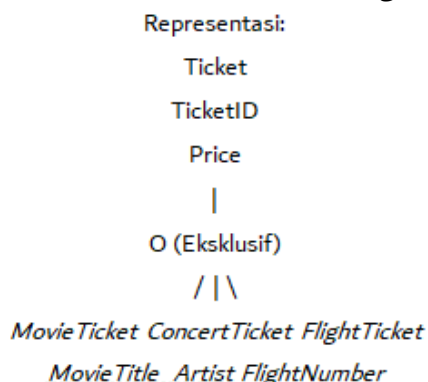
a) Atribut: *TicketID, Price*

2) *Subtypes:*

a) *MovieTicket*: atribut tambahan *MovieTitle*

b) *ConcertTicket*: atribut tambahan *Artist*

c) *FlightTicket*: atribut tambahan *FlightNumber*



C. Generalization

1. Definisi Generalization

Generalization adalah proses penggabungan dua atau lebih entitas yang memiliki atribut serupa menjadi satu entitas induk (*supertype*). *Generalization* bertujuan untuk mengurangi duplikasi data. *Generalization* adalah proses dalam pemodelan basis data yang bertujuan untuk menggabungkan dua atau lebih entitas yang memiliki atribut yang serupa atau umum menjadi satu entitas induk yang lebih umum, yang disebut *supertype*. Proses ini dilakukan untuk mengurangi duplikasi data, meningkatkan efisiensi dalam penyimpanan data, serta mempermudah pemeliharaan dan pengelolaan data dalam sistem.

Langkah-Langkah dalam *Generalization*:

- a. Identifikasi Atribut yang Sama: Langkah pertama dalam *generalization* adalah mengidentifikasi entitas-entitas yang memiliki atribut yang serupa. Misalnya, dua entitas yang berbeda, *Employee* dan *Manager*, mungkin memiliki atribut yang sangat mirip seperti *Name*, *Employee ID*, *Phone Number*, dan *Address*. Dalam hal ini, atribut-atribut yang serupa ini dapat digabungkan ke dalam satu entitas induk yang lebih umum, seperti *Person*.

- b. **Penyatuan Entitas:** Entitas yang memiliki atribut serupa akan digabungkan menjadi satu *supertype*. Misalnya, entitas *Employee* dan *Manager* bisa digabungkan menjadi entitas induk *Person*, yang akan menyimpan atribut umum seperti *Name*, *Employee ID*, *Phone Number*, dan *Address*.
- c. **Penyederhanaan Model:** Dengan *generalization*, model basis data menjadi lebih sederhana dan lebih mudah dikelola. Alih-alih mengelola dua entitas yang memiliki data yang sama, Anda hanya perlu mengelola satu entitas induk dengan atribut bersama, sementara entitas yang lebih spesifik akan mewarisi atribut ini.

Tujuan dan Manfaat *Generalization*

Proses *generalization* bertujuan untuk menyederhanakan dan merampingkan model basis data dengan mengurangi redundansi data dan mempermudah pemeliharaan. Berikut adalah beberapa tujuan dan manfaat utama dari *generalization*.

- a. **Mengurangi Duplikasi Data**

Generalization membantu dalam mengurangi pengulangan atribut yang sama dalam entitas yang berbeda. Misalnya, jika ada dua entitas yang memiliki atribut yang sama, kita dapat memindahkan atribut-atribut tersebut ke dalam entitas *supertype*, sehingga hanya ada satu tempat penyimpanan data

tersebut. Ini mengurangi kebutuhan penyimpanan dan menjaga konsistensi data. Contoh: Jika dalam sistem rumah sakit ada entitas *Doctor* dan *Patient*, kedua entitas ini mungkin memiliki atribut yang sama, seperti *Name*, *Date of Birth*, dan *Address*. Alih-alih menyimpan atribut-atribut tersebut dua kali di kedua entitas, kita dapat menggabungkan keduanya menjadi satu entitas *Person* yang memiliki atribut-atribut tersebut, dan kemudian *Doctor* dan *Patient* akan menjadi subtype yang mewarisi atribut-atribut tersebut.

b. Simplifikasi dan Modularisasi Data

Dengan menggabungkan entitas yang memiliki atribut serupa ke dalam satu entitas *supertype*, model basis data menjadi lebih terstruktur dan mudah dipahami. Setiap entitas spesifik yang dibangun di atas *supertype* akan lebih mudah dikelola dan diperbarui. Selain itu, perubahan pada atribut yang bersifat umum hanya perlu dilakukan di satu tempat (*supertype*), dan perubahan ini akan tercermin di seluruh sistem.

c. Pemeliharaan yang Lebih Mudah

Generalization membuat pemeliharaan data lebih mudah karena kita mengelola satu entitas induk (*supertype*) daripada entitas-entitas yang lebih spesifik. Jika ada perubahan yang perlu

dilakukan pada atribut umum, kita hanya perlu melakukannya di *supertype*, tanpa perlu memperbarui banyak entitas.

d. Fleksibilitas dan Ekstensibilitas

Dengan *generalization*, kita dapat dengan mudah menambahkan entitas baru yang berbagi atribut yang sama tanpa perlu menyesuaikan setiap entitas secara individu. Misalnya, setelah *generalization*, kita dapat menambahkan entitas *Staff* sebagai subtype dari *Person* tanpa perlu mendefinisikan ulang atribut *Name* atau *Date of Birth*.

Contoh *Generalization*

Misalnya, dalam sistem rumah sakit, kita memiliki entitas *Doctor* dan *Patient* yang masing-masing memiliki beberapa atribut yang serupa, seperti *Name*, *Date of Birth*, *Gender*, dan *Address*. Sebelumnya, kita mungkin akan mendefinisikan dua tabel berbeda: satu untuk *Doctor* dan satu untuk *Patient*, dengan atribut yang serupa disalin di kedua tabel. Dengan menggunakan *generalization*, kita dapat menggabungkan *Doctor* dan *Patient* menjadi entitas induk yang lebih umum, yaitu *Person*. *Person* akan berisi atribut-atribut umum seperti:

- a. *Name*
- b. *Date of Birth*
- c. *Gender*

d. *Address*

Kemudian, *Doctor* dan *Patient* akan menjadi subtype dari *Person* dan hanya menyimpan atribut yang relevan untuk masing-masing. Misalnya:

- a. *Doctor* dapat memiliki atribut tambahan seperti *Specialization*, *License Number*, dan *Years of Experience*.
- b. *Patient* dapat memiliki atribut seperti *Medical History*, *Insurance Number*, dan *Admission Date*.

Dengan cara ini, *Doctor* dan *Patient* mewarisi atribut yang sama dari *Person*, sementara atribut tambahan yang spesifik untuk masing-masing entitas tetap terpisah.

Proses *Generalization* dalam Praktik

- a. Identifikasi Entitas dengan Atribut Serupa: Tentukan entitas yang memiliki banyak atribut yang sama atau serupa. Misalnya, *Employee* dan *Manager*.
- b. Buat *Supertype*: Gabungkan atribut umum tersebut dalam satu entitas induk yang lebih umum, yang disebut *supertype* (misalnya, *Person*).
- c. Pemisahan Subtype: Tentukan entitas-entitas yang lebih spesifik yang akan menjadi *subtype* dari *supertype*, dan

tentukan atribut tambahan atau hubungan unik yang dimiliki oleh masing-masing subtype (misalnya, *Doctor* dan *Patient*).

- d. Pengelolaan Redundansi: Pastikan bahwa atribut yang sama tidak disalin ke dalam entitas yang berbeda, dan hanya disimpan di *supertype*. Pastikan bahwa perubahan pada atribut yang umum dilakukan di *supertype*, dan secara otomatis diterapkan pada semua *subtype*.

Generalization adalah teknik untuk menggabungkan dua atau lebih entitas yang memiliki atribut yang serupa menjadi satu entitas induk yang lebih umum. Tujuannya adalah untuk mengurangi duplikasi data, menyederhanakan model basis data, dan membuat pengelolaan data lebih efisien. Proses ini meningkatkan konsistensi, mempermudah pemeliharaan, dan memungkinkan pengembangan sistem yang lebih fleksibel dan terstruktur.

2. Contoh Kasus

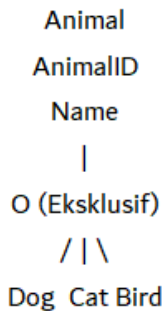
Misalkan kita memiliki entitas "*Dog*", "*Cat*", dan "*Bird*" yang semuanya memiliki atribut *Animal ID* dan *Name*. Atribut ini dapat digeneralisasi ke dalam entitas *Animal*.

- a. *Supertype*: *Animal*

Atribut: *AnimalID*, *Name*

b. *Subtype*:

- a. *Dog*: memiliki atribut tambahan *Breed*
- b. *Cat*: memiliki atribut tambahan *FurLength*
- c. *Bird*: memiliki atribut tambahan *WingSpan*

Representasi dengan Notasi Barker

Penjelasan: Dengan generalisasi, atribut umum hanya didefinisikan sekali pada entitas *supertype*, mengurangi redundansi.

Contoh Kasus Tambahan

1. Sistem Pendidikan:

a. *Supertype: Institution*

Atribut: *InstitutionID*, *Name*

b. *Subtypes*:

- a. *School*: atribut tambahan *SchoolLevel*
- b. *University*: atribut tambahan *NumberOfFaculties*

Representasi:

Institution

InstitutionID

Name

|

O (Eksklusif)

/ \

School University

SchoolLevel NumberOfFaculties

2. Sistem Pengelolaan Proyek:

a. *Supertype*: Project

Atribut: *ProjectID*, *Budget*

b. *Subtypes*:

a. *Construction*: atribut tambahan *Location*

b. *IT*: atribut tambahan *TechnologyStack*

Representasi:

Project

ProjectID

Budget

|

O (Eksklusif)

/ \

Construction IT

Location TechnologyStack

D. *Specialization*

1. Definisi *Specialization*

Specialization adalah kebalikan dari *generalization*, di mana entitas *supertype* dipecah menjadi beberapa entitas *subtype* berdasarkan karakteristik unik. Pendekatan ini digunakan untuk menangani perbedaan spesifik antar-entitas.

Specialization adalah kebalikan dari *generalization* dalam pemodelan basis data, yang digunakan untuk membagi sebuah entitas *supertype* yang lebih umum menjadi beberapa entitas *subtype* yang lebih spesifik berdasarkan karakteristik atau atribut unik yang membedakan masing-masing entitas. *Specialization* membantu dalam menangani perbedaan antar-entitas yang memiliki karakteristik yang berbeda namun tetap berbagi beberapa atribut atau sifat yang umum.

Specialization adalah proses pemecahan entitas *supertype* yang lebih umum menjadi beberapa entitas *subtype* yang lebih khusus, dengan menambahkan atribut atau karakteristik yang membedakan *subtype* satu sama lain. Setiap *subtype* mewakili grup entitas yang memiliki atribut atau perilaku yang lebih spesifik. Dengan pendekatan ini, kita dapat menangani perbedaan-perbedaan khusus antar-entitas sambil tetap menjaga elemen umum yang dimiliki oleh semua entitas dalam *supertype*.

Langkah-Langkah dalam *Specialization*:

- a. Identifikasi Perbedaan Antar Entitas: Langkah pertama dalam *specialization* adalah mengidentifikasi perbedaan yang ada antara entitas yang berbagi beberapa karakteristik yang sama, tetapi memiliki kebutuhan atau atribut yang berbeda.
- b. Pemisahan Atribut Khusus: Berdasarkan perbedaan tersebut, kita membagi *supertype* menjadi beberapa *subtype*. Setiap subtype akan memiliki atribut yang hanya relevan untuk entitas tersebut, selain atribut yang diwarisi dari supertype.
- c. Menentukan Diskriminator: Sebagai bagian dari proses *specialization*, kita perlu menentukan suatu atribut yang disebut discriminator, yang digunakan untuk membedakan entitas mana yang termasuk dalam *subtype* tertentu. Diskriminator ini sering kali berupa nilai atau kategori yang dapat mengidentifikasi tipe entitas yang lebih spesifik.

2. Tujuan dan Manfaat Specialization

Specialization memungkinkan untuk menangani perbedaan dalam data yang perlu dipecah menjadi kelompok-kelompok yang lebih spesifik sambil tetap mempertahankan hubungan hierarkis dengan supertype. Berikut adalah beberapa tujuan dan manfaat utama dari *specialization*:

a. Menangani Perbedaan Spesifik Antar Entitas

Salah satu alasan utama untuk menggunakan *specialization* adalah untuk menangani perbedaan yang ada antar-entitas yang memiliki beberapa kesamaan, tetapi juga memiliki atribut atau karakteristik unik. Misalnya, dalam sistem rumah sakit, entitas *Person* (*supertype*) dapat dibagi menjadi *subtype* seperti *Doctor* dan *Patient*, yang memiliki karakteristik yang sangat berbeda meskipun keduanya berbagi atribut dasar yang sama (seperti *Name* dan *Date of Birth*).

b. Mengurangi Ambiguitas

Dengan *specialization*, kita dapat menghindari kebingungan atau ambiguitas yang mungkin terjadi jika kita mencoba menampung informasi yang berbeda dalam satu entitas yang lebih umum. Setiap *subtype* yang diciptakan memiliki atribut yang jelas dan unik, yang membedakan mereka satu sama lain, sehingga memudahkan dalam pengelolaan dan pemrosesan data.

c. Pengelolaan Data yang Lebih Fleksibel

Specialization memungkinkan pengelolaan data yang lebih fleksibel dengan mengelompokkan data berdasarkan karakteristik khususnya. Hal ini memberi keuntungan dalam hal query dan pemeliharaan, karena kita dapat mengakses data

yang lebih spesifik dengan lebih mudah tanpa perlu menavigasi atribut yang tidak relevan.

d. Peningkatan Keamanan Data

Dengan membagi data ke dalam beberapa *subtype*, kita juga bisa mengelompokkan data berdasarkan tingkat akses atau keamanan. Misalnya, informasi yang lebih sensitif seperti *Medical History* pada entitas Patient bisa lebih dilindungi dengan mengelompokkan data tersebut dalam subtype yang memiliki hak akses tertentu.

3. Contoh *Specialization*

Dalam sistem manajemen rumah sakit, kita mungkin memiliki supertype Person yang mencakup atribut umum seperti *Name*, *Date of Birth*, *Gender*, dan *Address*. Berdasarkan karakteristik unik dari entitas-entitas dalam sistem rumah sakit, kita kemudian dapat membagi Person menjadi beberapa *subtype*:

- a. *Doctor*: *Subtype* ini mewarisi atribut umum dari Person dan menambahkan atribut yang spesifik seperti *Specialization*, *License Number*, dan *Years of Experience*.
- b. *Patient*: *Subtype* ini juga mewarisi atribut umum dari Person, namun memiliki atribut spesifik seperti *Medical History*, *Admission Date*, dan *Insurance Number*.

- c. *Nurse: Subtype* ini mewarisi atribut yang sama dari *Person*, tetapi memiliki atribut seperti *Ward Assigned*, *Shift Timing*, dan *Qualifications*.

Pada titik ini, setiap entitas *Doctor*, *Patient*, dan *Nurse* dapat dikelola dan diproses sesuai dengan karakteristiknya, sementara tetap berbagi data umum dari *Person*.

4. Proses *Specialization* dalam Praktik

- a. Identifikasi Entitas *Supertype*: Tentukan entitas yang memiliki atribut atau karakteristik umum yang digunakan oleh beberapa entitas lainnya. Misalnya, *Person* adalah entitas *supertype* yang memiliki atribut umum.
- b. Pemisahan Entitas *Supertype* Menjadi *Subtype*: Tentukan perbedaan utama antara entitas yang berbagi beberapa atribut yang sama dan tentukan *subtype* yang lebih spesifik. Misalnya, memisahkan *Person* menjadi *Doctor*, *Patient*, dan *Nurse* berdasarkan karakteristik unik masing-masing.
- c. Menentukan Atribut Khusus untuk Setiap *Subtype*: Setiap *subtype* akan memiliki atribut tambahan yang spesifik. Misalnya, *Doctor* akan memiliki atribut *Specialization*, sementara *Patient* akan memiliki atribut *Medical History*.

- d. Diskriminator: Tentukan atribut diskriminator yang digunakan untuk mengidentifikasi subtype dari supertype. Misalnya, diskriminator bisa berupa status atau jenis entitas yang bisa dipilih berdasarkan kriteria tertentu, seperti *Person Type* yang dapat bernilai *Doctor*, *Patient*, atau *Nurse*.

5. Manfaat *Specialization*

Peningkatan Akurasi dan Organisasi Data: Dengan membagi *supertype* menjadi beberapa *subtype*, kita dapat memastikan bahwa setiap entitas lebih terorganisir dan hanya menyimpan informasi yang relevan untuk tipe tersebut.

- a. Keterbacaan dan Pemeliharaan yang Lebih Baik: Data lebih mudah dipahami dan dikelola ketika ada pembagian yang jelas antara entitas yang lebih umum dan entitas yang lebih spesifik.
- b. Pengelolaan Atribut Unik: Atribut yang hanya relevan untuk tipe tertentu dapat disimpan dengan rapi tanpa mencampurkannya dengan atribut yang tidak terkait.

Specialization adalah proses di mana sebuah entitas *supertype* dibagi menjadi beberapa entitas *subtype* yang lebih spesifik berdasarkan perbedaan karakteristik atau atribut yang membedakan masing-masing entitas. Dengan *specialization*, kita bisa menangani perbedaan antar-entitas secara lebih terstruktur

dan efisien, mengurangi ambiguitas, dan mempermudah pengelolaan serta pemeliharaan data yang lebih kompleks. *Specialization* membantu memenuhi kebutuhan data yang lebih kompleks dengan menangani variabilitas dalam atribut.

Contoh Kasus

Dari entitas *Employee*, kita dapat membuat subtypes berdasarkan peran:

a. *Supertype: Employee*

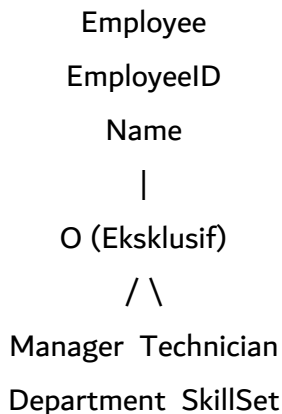
Atribut: *EmployeeID, Name*

b. Subtypes:

a. Manager: Department

b. Technician: SkillSet

Representasi dengan Notasi Barker



Penjelasan: Setiap Employee bisa menjadi Manager atau Technician, tetapi tidak keduanya karena hubungan bersifat eksklusif.

Studi Kasus Lanjutan

Dalam organisasi besar, kita dapat memperluas model dengan menambahkan subtype seperti "*Contractor*" atau "*Intern*". Hal ini memastikan bahwa data setiap kategori karyawan tercatat dengan baik.

E. Relasi Antar Konsep

Hubungan dan Implementasi

a. *Supertype/Subtype*:

- 1) Membentuk hierarki data.
- 2) *Subtype* mewarisi semua atribut dari *supertype*.

b. *Inheritance*:

- 1) Mekanisme pewarisan atribut dalam *supertype-subtype*.
- 2) Memastikan bahwa semua atribut umum tidak perlu didefinisikan ulang di setiap *subtype*.

c. *Generalization*:

- 1) Digunakan untuk menggabungkan entitas dengan atribut yang serupa menjadi *supertype*.
- 2) Mengurangi redundansi dan meningkatkan efisiensi.

d. *Specialization*:

- 1) Digunakan untuk membagi entitas umum menjadi beberapa entitas spesifik (subtype).
- 2) Menangani kebutuhan data yang lebih spesifik.

e. Eksklusif vs Inklusif:

- 1) Eksklusif: Hanya satu subtype yang valid untuk supertype tertentu.
- 2) Inklusif: Bisa memiliki lebih dari satu subtype.

Relasi antar-konsep seperti *Supertype/Subtype*, *Inheritance*, *Generalization*, *Specialization*, serta Eksklusif vs Inklusif adalah bagian penting dari pemodelan data yang membantu dalam menyusun dan mengelola data dengan cara yang lebih terstruktur, efisien, dan fleksibel. Konsep-konsep ini bekerja bersama untuk menciptakan hierarki data yang memungkinkan pemeliharaan dan pengelolaan data menjadi lebih mudah, serta meminimalkan redundansi. Berikut adalah penjelasan yang lebih detail mengenai hubungan dan implementasi dari konsep-konsep ini:

1. *Supertype/Subtype*

Supertype/Subtype adalah hubungan antara entitas yang lebih umum (*supertype*) dan entitas yang lebih spesifik (*subtype*). *Supertype* berisi atribut yang berlaku untuk seluruh kelompok entitas yang lebih spesifik, sementara *subtype* hanya

menyimpan atribut yang relevan untuk entitas tersebut. Hubungan ini membentuk hierarki data, di mana *supertype* berada di tingkat yang lebih tinggi, dan *subtype* berada di bawahnya.

Implementasi:

- a. Mewarisi Atribut: Semua atribut yang terdapat dalam *supertype* diwariskan ke semua *subtype*, yang berarti tidak perlu mendefinisikan atribut yang sama berulang kali di setiap *subtype*. Hal ini mengurangi redundansi dan meningkatkan efisiensi.
- b. Struktur Hierarki: *Supertype/subtype* membentuk struktur hirarkis dalam model data. *Supertype* bisa memiliki beberapa *subtype* yang masing-masing memiliki karakteristik yang unik.

Contoh: Dalam sistem rumah sakit, entitas Person bisa menjadi *supertype* yang memiliki atribut umum seperti *Name*, *Date of Birth*, *Address*, yang kemudian diwariskan ke subtype seperti *Doctor* dan *Patient*, yang menambahkan atribut khusus untuk masing-masing entitas.

2. Inheritance

Inheritance adalah mekanisme yang memungkinkan *subtype* mewarisi atribut dan hubungan yang ada pada *supertype*. Ini

memastikan bahwa data yang umum hanya perlu didefinisikan sekali di *supertype*, tanpa perlu menduplikasi definisi atribut di setiap *subtype*.

Implementasi:

- a. Pewarisan Atribut: Semua atribut yang ada di *supertype* secara otomatis diwariskan ke setiap *subtype*. *Subtype* kemudian dapat menambahkan atribut unik yang hanya relevan untuk tipe entitas tersebut.
- b. Konsistensi Data: Dengan *inheritance*, perubahan pada atribut yang ada di *supertype* akan diterapkan ke semua *subtype* tanpa perlu memperbarui tiap-tiap *subtype* secara manual.

Contoh: Jika ada perubahan dalam atribut *Address* pada *Person*, perubahan tersebut langsung tercermin pada *Doctor* dan *Patient* tanpa perlu dilakukan perubahan terpisah di kedua entitas tersebut.

3. *Generalization*

Generalization adalah proses penggabungan dua atau lebih entitas yang memiliki atribut serupa menjadi satu entitas induk yang lebih umum, yang disebut *supertype*. Tujuan dari *generalization* adalah untuk mengurangi duplikasi data dan meningkatkan efisiensi model basis data.

Implementasi:

- a. Penggabungan Atribut Serupa: *Generalization* dilakukan dengan mengidentifikasi entitas-entitas yang memiliki atribut yang serupa dan menyatukannya dalam satu *supertype*. Hal ini mengurangi redundansi dan memudahkan pemeliharaan data.
- b. Menyederhanakan Model: Dengan menggabungkan entitas yang mirip, model data menjadi lebih sederhana dan lebih mudah dipahami serta dikelola.

Contoh: Dalam sistem rumah sakit, entitas *Employee* dan *Manager* bisa digabungkan menjadi entitas *Person* yang lebih umum, karena keduanya memiliki atribut serupa seperti *Name*, *Date of Birth*, dan *Address*.

4. *Specialization*

Specialization adalah kebalikan dari *generalization*. Di sini, sebuah entitas *supertype* yang lebih umum dibagi menjadi beberapa entitas *subtype* yang lebih spesifik, masing-masing dengan atribut unik yang hanya berlaku untuk subtype tersebut. *Specialization* digunakan untuk menangani kebutuhan data yang lebih spesifik.

Implementasi:

- a. Pemisahan Entitas Umum: *Specialization* dilakukan dengan membagi entitas *supertype* menjadi beberapa *subtype*, yang masing-masing memiliki atribut atau karakteristik yang membedakan mereka dari yang lain.
- b. Menangani Perbedaan Spesifik: Setiap *subtype* dapat memiliki atribut yang relevan hanya untuk tipe entitas tersebut, sehingga memungkinkan penanganan data yang lebih terstruktur.

Contoh: Dalam sistem rumah sakit, entitas *Person* bisa dibagi menjadi *subtype* seperti *Doctor*, *Nurse*, dan *Patient*. Masing-masing *subtype* memiliki atribut yang spesifik seperti *Specialization* untuk *Doctor*, *Ward Assigned* untuk *Nurse*, dan *Medical History* untuk *Patient*.

5. Eksklusif vs Inklusif

Hubungan antara *supertype* dan *subtype* bisa bersifat eksklusif atau inklusif, yang mengatur apakah lebih dari satu *subtype* dapat diterapkan pada suatu *supertype*.

- a. Eksklusif: Dalam hubungan eksklusif, hanya satu *subtype* yang dapat berlaku untuk *supertype* pada waktu tertentu. Ini berarti entitas yang termasuk dalam *supertype* hanya dapat menjadi satu *subtype*, bukan lebih dari satu.

- b. Inklusif: Dalam hubungan inklusif, lebih dari satu *subtype* dapat berlaku untuk *supertype* pada waktu yang sama. Dengan kata lain, entitas yang termasuk dalam *supertype* bisa tergabung dalam lebih dari satu *subtype*.

Implementasi:

- a. Eksklusif: Biasanya digunakan ketika tipe entitas yang lebih spesifik saling bertentangan atau saling eksklusif, seperti dalam kasus *Employee* yang bisa menjadi *Full-time* atau *Part-time*, tetapi tidak bisa keduanya pada saat yang sama.
- b. Inklusif: Digunakan ketika entitas bisa tergolong dalam lebih dari satu tipe yang lebih spesifik, seperti seorang *Student* yang juga dapat menjadi *Employee* pada waktu yang sama.

Contoh:

- a. Eksklusif: Dalam sistem manajemen kepegawaian, seorang *Employee* bisa menjadi *Full-time* atau *Part-time*, tetapi tidak keduanya pada saat yang sama.
- b. Inklusif: Dalam sistem rumah sakit, seorang *Person* bisa menjadi *Doctor* dan *Teacher* pada saat yang sama, karena atribut atau pekerjaan mereka tidak saling bertentangan.

Ringkasan Relasi Antar-Konsep:

- a. *Supertype/Subtype* membentuk struktur hierarki data, di mana *subtype* mewarisi atribut dari *supertype*.
- b. *Inheritance* memastikan bahwa atribut yang diwariskan dari *supertype* tidak perlu didefinisikan ulang di setiap *subtype*.
- c. *Generalization* menggabungkan entitas dengan atribut serupa menjadi satu *supertype* untuk mengurangi redundansi.
- d. *Specialization* membagi entitas *supertype* menjadi beberapa entitas *subtype* yang lebih spesifik berdasarkan atribut unik.
- e. Eksklusif vs Inklusif menentukan apakah hanya satu *subtype* yang dapat berlaku untuk *supertype* pada waktu tertentu (eksklusif) atau lebih dari satu *subtype* dapat berlaku pada saat yang sama (inklusif).

Dengan memahami hubungan ini, kita dapat merancang model basis data yang lebih efisien, terstruktur dengan baik, dan mudah dipelihara, yang memungkinkan pengelolaan data lebih efektif dan fleksibel.

F. Kesimpulan

Penggunaan relasi logika *supertype*, *subtype*, *inheritance*, *generalization*, dan *specialization* sangat penting untuk menciptakan model basis data yang terstruktur dan efisien.

Dengan menggunakan notasi Barker, hubungan ini dapat divisualisasikan dengan jelas, memudahkan pemahaman dan implementasi dalam desain basis data. Pendekatan ini membantu dalam mengelola data yang kompleks dengan cara yang logis, memungkinkan perancang untuk fokus pada kebutuhan spesifik tanpa kehilangan esensi data yang lebih umum.

Pendekatan relasi logika seperti supertype, subtype, inheritance, generalization, dan specialization adalah bagian integral dalam perancangan basis data yang berorientasi objek atau menggunakan model entitas-relasi yang lebih maju. Dengan notasi Barker, hubungan-hubungan ini dapat divisualisasikan untuk memudahkan pemahaman dan implementasi. Berikut penjelasan tentang masing-masing konsep dalam konteks basis data:

1. *Supertype* dan *Subtype*:

Supertype adalah entitas yang mewakili konsep yang lebih umum, sementara *subtype* adalah entitas yang lebih spesifik dan merupakan turunan dari *supertype*. Sebagai contoh, dalam sistem manajemen rumah sakit, entitas "*Person*" (*supertype*) bisa memiliki *subtype* seperti "*Doctor*", "*Nurse*", atau "*Patient*". Relasi ini memungkinkan pengelompokan data berdasarkan kesamaan dan perbedaan karakteristik yang lebih spesifik.

2. *Inheritance* (Pewarisan):

Konsep *inheritance* memungkinkan entitas *subtype* mewarisi atribut dan hubungan dari supertype-nya. Sebagai contoh, entitas "*Doctor*" akan mewarisi atribut umum dari *supertype* "*Person*" seperti "*Name*" dan "*DOB*" (Tanggal Lahir), tetapi juga bisa menambah atribut spesifik seperti "*Specialization*". Ini memudahkan pengelolaan data yang berulang dan meningkatkan efisiensi dalam basis data.

3. *Generalization* dan *Specialization*:

Generalization adalah proses penggabungan entitas-entitas yang memiliki kesamaan atribut menjadi supertype yang lebih umum. *Specialization* adalah proses pembagian entitas yang lebih umum menjadi subtype yang lebih spesifik. Kedua konsep ini berfungsi untuk mempermudah organisasi data, membantu mengurangi redundansi dan meningkatkan fleksibilitas desain.

Dengan menggunakan notasi Barker, relasi ini dapat digambarkan dengan jelas dalam diagram entitas-relasi (ER), yang membantu tim desain dalam memahami dan membangun struktur basis data yang lebih terorganisir dan mudah dikelola. Pendekatan ini juga penting dalam mendukung kebutuhan pengguna akhir, memastikan data yang relevan tetap terjaga, dan memungkinkan pengolahan data yang lebih efisien.

Penggunaan relasi logika seperti *supertype*, *subtype*, *inheritance*, *generalization*, dan *specialization* sangat penting dalam perancangan basis data yang efisien dan terstruktur. Konsep-konsep ini membantu dalam mendefinisikan hubungan antar entitas dan memungkinkan perancang basis data untuk mengelola data yang kompleks dengan cara yang lebih logis dan terorganisir. Dengan memanfaatkan notasi Barker, hubungan ini dapat divisualisasikan dengan jelas, mempermudah pemahaman serta implementasi dalam desain basis data. Pendekatan ini sangat berguna dalam merancang sistem basis data yang mampu menangani kebutuhan yang bervariasi tanpa mengorbankan integritas dan efisiensi struktur data.

Supertype dan *subtype* adalah konsep dasar yang memungkinkan pemodelan entitas yang lebih umum (*supertype*) dan entitas yang lebih spesifik (*subtype*). Sebagai contoh, dalam sistem manajemen rumah sakit, entitas "*Person*" bisa menjadi *supertype* yang memiliki *subtype* seperti "*Doctor*", "*Nurse*", dan "*Patient*". Masing-masing *subtype* ini akan memiliki karakteristik yang lebih spesifik namun tetap mempertahankan hubungan dengan *supertype*. Dengan menggunakan pendekatan ini, data yang serupa dapat dikelompokkan dalam *supertype*, sementara perbedaan yang lebih spesifik dapat ditangani dalam *subtype*. Hal

ini sangat membantu dalam meminimalisir redundansi data dan mempermudah pengelolaan informasi.

Inheritance atau pewarisan adalah mekanisme di mana entitas *subtype* mewarisi atribut dan relasi dari *supertype*. Misalnya, entitas "*Doctor*" sebagai *subtype* dari "*Person*" dapat mewarisi atribut umum seperti "*Name*", "*Date of Birth*", dan "*Gender*", sementara juga dapat memiliki atribut tambahan yang spesifik, seperti "*Specialization*" atau "*License Number*". Konsep ini memungkinkan perancang basis data untuk menghindari duplikasi data, mengurangi kompleksitas desain, serta mempermudah pemeliharaan dan pembaruan struktur basis data.

Generalization dan *specialization* adalah dua konsep yang saling berkaitan dalam hal pengelompokkan dan pemecahan entitas. *Generalization* adalah proses menyatukan beberapa entitas yang memiliki kesamaan atribut menjadi satu entitas yang lebih umum (*supertype*), sementara *specialization* adalah kebalikan dari *generalization*, yaitu memecah entitas umum menjadi beberapa entitas yang lebih spesifik (*subtype*). Kedua konsep ini memungkinkan perancang untuk lebih fleksibel dalam mendefinisikan data yang ada, serta menjaga agar data tetap terstruktur dengan baik dan mudah diakses sesuai dengan kebutuhan spesifik.

Dengan menggabungkan semua konsep ini dalam perancangan basis data, sistem yang dibangun akan lebih efisien, mudah dikelola, dan dapat beradaptasi dengan kebutuhan yang terus berkembang. Notasi Barker, yang menggambarkan hubungan ini secara visual, mempermudah komunikasi antara perancang basis data dan pemangku kepentingan lainnya, serta memperjelas struktur dan hubungan antar entitas. Pendekatan ini sangat membantu dalam menciptakan model basis data yang mampu menangani data yang besar dan kompleks, sambil tetap menjaga kinerja dan fleksibilitas yang tinggi.

DAFTAR PUSTAKA

- Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of database systems* (7th ed.). Pearson.
- Kroenke, D. M. (2013). *Database processing: Fundamentals, design, and implementation* (13th ed.). Pearson.
- Rob, P., & Coronel, C. (2007). *Database systems: Design, implementation, and management* (8th ed.). Cengage Learning.
- Teorey, T. J., Yang, J., & Fry, J. P. (2011). *Database modeling and design: Logical design* (5th ed.). Morgan Kaufmann.

BAB 5

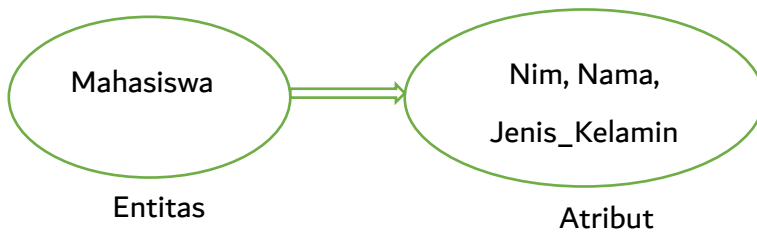
NORMALISASI DAN DENORMALISASI DATA PADA SUATU TABEL DATA YANG TIDAK NORMAL

Oleh : Tri Santoso, S.Kom, M.Kom

A. Definisi Normalisasi

Normalisasi memiliki banyak pengertian. Ada yang menyebutkan bahwa normalisasi merupakan proses pengelompokan atau pengorganisasian data menjadi tabel (Qalbi, 2014). Ada pula yang mengartikan jika normalisasi merupakan proses dekomposisi untuk menghilangkan anomali relasional (Jamaluddin et al., 2022). Selain untuk menghilangkan anomali, normalisasi dilakukan untuk menghilangkan ambiguitas dan menghindari redudansi serta mencegah atau mengurangi masalah pengolahan data (Alia et al., 2023). Dengan demikian, bisa ditarik kesimpulan bahwa normalisasi adalah proses dekomposisi dalam upaya untuk mengelompokkan data menjadi organisasi-organisasi berbeda yang nantinya organisasi-organisasi tersebut menjadi tabel-tabel yang

bisa saling berelasi untuk mencegah timbulnya ambiguitas, redudansi, anomaly dan masalah-masalah lain dalam pengolahan data. Sebelum melangkah lebih jauh, perlu untuk mengetahui istilah dan konsep yang akan sering digunakan pada proses normalisasi ini.



Gambar 5. 1 Entitas dan Atribut

Dari gambar diatas terdapat dua istilah, yaitu entitas dan atribut. Entitas merupakan sebuah objek yang bisa berupa benda, orang, atau badan. Sedangkan atribut merupakan karakteristik dari entitas. Sebagai contoh pada gambar, entitas adalah “Mahasiswa”. Dimana mahasiswa masuk dalam kategori orang. Objek (entitas) mahasiswa memiliki karakteristik (atribut) “nim”, “nama” dan “jenis_kelamin”. Jumlah atribut bisa disesuaikan dengan keperluan yang ada. Jika menginginkan data yang detail dan spesifik, maka memerlukan atribut yang banyak pula.



nim	nama	jenis_kelamin
0809121	Andi	Laki-laki
0809122	Ani	Perempuan

Gambar 5. 2 Field dan Record

Dari gambar diatas terdapat dua istilah, yaitu *field* dan *record*. *Field* sebenarnya sama dengan atribut, hanya saja *field* biasa merupakan penyebutan atribut yang sudah berbentuk kolom pada tabel. Sedangkan *record* merupakan gabungan dari field-field yang ada yang membentuk 1 baris data yang saling terkait. Sebagai contoh pada gambar diatas, field adalah “nim” sama seperti atribut yang dibahas sebelumnya. Dengan begitu “nama” dan “jenis_kelamin” juga merupakan field. Sedangkan *record* adalah gabungan isi dari “0809122”, “Ani” dan “Perempuan” yang membentuk 1 baris data. Dengan begitu “0809121”, “Andi” dan “Laki-laki” juga merupakan 1 *record*.

B. Tahapan Normalisasi

Dalam prosesnya normalisasi memiliki beberapa tahapan yang dilalui. Dimana tiap tahapan dibuat untuk menghilangkan sebuah ketergantungan pada data untuk mencegah munculnya masalah saat pengolahan data. Tahapan tersebut diantaranya:

Pada tahap ini, semua data yang diperlukan dikumpulkan dan dimasukkan ke dalam satu tabel. Dalam proses ini, tidak ada ketentuan khusus yang harus diikuti. Sehingga bisa saja terdapat data yang tidak lengkap, terduplikasi dan memiliki anomali. Sebagai contoh penerapannya, anggaplah terdapat berkas yaitu :

Gambar 5. 3 Contoh Berkas

Dari contoh berkas diatas, terdapat beberapa data yang bisa diambil. Data-data tersebut bisa dimasukkan ke dalam tabel seperti dibawah ini:

Tabel 5. 1 Tabel Bentuk Tidak Normal

<u>Id</u> <u>Nota</u>	<u>Tgl</u> <u>Nota</u>	<u>Id</u> <u>Barang</u>	<u>Nama</u> <u>Barang</u>	<u>Id</u> <u>Kategori</u>	<u>Kategori</u>	<u>Harga</u>	<u>Qty</u>	<u>Jumlah</u>	<u>Total</u>	<u>Id</u> <u>Pelanggan</u>	<u>Nama</u> <u>Pelanggan</u>
N001	10-	B001,	Pulpen,	K001,	Alat	5.000,	3, 2	15.000,	29.000	P001	Andi
	11-	B002	Buku	K002	Tulis,	7.000		14.000			
	2012		Tulis		Buku						
N002	11-	B003,	Kertas	K003,	Kertas,	500,	10,	10.000,	22.000	P002	Ani
	11-	B004	A4,	K001	Alat	3.000	4	12.000			
	2012		Pensil		Tulis						

Dari hasil pengumpulan data yang ada, terbentuklah tabel seperti diatas yang merupakan contoh dari bentuk tidak normal dalam normalisasi. Terlihat dalam tabel tersebut banyak atribut multi value atau bernilai banyak. Hal ini dikarenakan dalam satu kolom terdapat lebih dari satu nilai. Kondisi ini akan diatasi pada bentuk normal pertama dipoin selanjutnya.

2. Normal Satu (1NF/ *1st Normal Form*)

Tahap ini biasa disebut dengan bentuk normal pertama dan merupakan pengembangan dari tahap tidak normal pada poin sebelumnya. Tahap ini dibuat untuk menghilangkan perulangan pada kelompok data tertentu. Pada tahap ini data mulai dikelompokkan berdasarkan jenisnya. Kemudian atribut yang ada

dibuat agar hanya memiliki satu nilai atau value yang bersifat atomik. Oleh karena itu hanya akan ada satu nilai saja disetiap pertemuan baris dan kolom atau atribut. Kondisi ini biasa disebut dengan istilah *single value attribute*. Secara otomatis kondisi normal pertama ini menghilangkan atribut yang memiliki multi atau banyak nilai atau biasa disebut multi *value attribute*. Pada tahap ini pula terdapat penentuan *candidate key*.

Candidate key sendiri merupakan satu atau beberapa kolom yang nilainya bisa mewakili 1 *record*. Kolom yang menjadi *candidate key* akan menjadi calon *primary key* pada tahap selanjutnya. Sebagai contoh penerapannya, akan diambil contoh tabel bentuk tidak normal dari poin sebelumnya. Selanjutnya akan diubah menjadi bentuk normal pertama. Pertama-tama, bentuk tidak normal dari poin sebelumnya diubah seperti dibawah ini:

Tabel 5.2 Tabel Bentuk Tidak Normal

<u>Id</u> <u>Nota</u>	<u>Tgl</u> <u>Nota</u>	<u>Id</u> <u>Barang</u>	<u>Nama</u> <u>Barang</u>	<u>Id</u> <u>Kategori</u>	<u>Kategori</u>	<u>Harga</u>	<u>Qty</u>	<u>Jumlah</u>	<u>Total</u>	<u>Id</u> <u>Pelanggan</u>	<u>Nama</u> <u>Pelanggan</u>
N001	10- 11- 2012	B001	Pulpen	K001	Alat Tulis	5.000	3	15.000	29.000	P001	Andi
		B002	Buku Tulis	K002	Buku	7.000	2	14.000			
N002	11- 11- 2012	B003	Kertas A4	K003	Kertas	500	10	10.000	22.000	P002	Ani
		B004	Pensil	K001	Alat Tulis	3.000	4	12.000			

Hal ini dilakukan untuk membuat data dari atribut yang memiliki multi value menjadi *single value*. Setelah itu, melengkapi kolom-kolom kosong yang ada seperti dibawah ini:

Tabel 5.3 Tabel Bentuk Normal Pertama

<u>Id</u> Nota	<u>Tgl</u> Nota	<u>Id</u> <u>Barang</u>	<u>Nama</u> <u>Barang</u>	<u>Id</u> <u>Kategori</u>	<u>Kategori</u>	<u>Harga</u>	<u>Qty</u>	<u>Jumlah</u>	Total	<u>Id</u> <u>Pelanggan</u>	<u>Nama</u> <u>Pelanggan</u>
N001	10-11-2012	B001	Pulpen	K001	Alat Tulis	5.000	3	15.000	29.000	P001	Andi
N001	10-11-2012	B002	Buku Tulis	K002	Buku	7.000	2	14.000	29.000	P001	Andi
N002	11-11-2012	B003	Kertas A4	K003	Kertas	500	10	10.000	22.000	P002	Ani
N002	11-11-2012	B004	Pensil	K001	Alat Tulis	3.000	4	12.000	22.000	P002	Ani

Setelah selesai mengisi kolom yang kosong, maka makna dari kondisi tabel diatas akan sama dengan kondisi tabel 5.1 hanya saja dibuat menjadi *single value* ditiap atributnya. Dengan demikian jadilah bentuk normal pertama dari proses normalisasi. Hal ini ditandai dimana sudah tidak ada atribut multi value pada tabel tersebut.

Selanjutnya, jika dilihat dengan seksama atribut-atribut pada tabel 5.3 maka atribut tersebut dapat dikelompokkan menjadi beberapa jenis data diantaranya: data penjualan, data detail

penjualan dan data data barang. Data penjualan melingkupi atribut: Id Nota, Tgl Nota, Total, Id Pelanggan dan Nama Pelanggan. Data detail penjualan melingkupi atribut: Id Nota, Id Barang, Qty dan Jumlah. Data barang melingkupi atribut: Id Barang, Nama Barang, Id Kategori, Kategori dan Harga. Dari kelompok yang ada, dapat dilihat bahwa atribut Id Nota dan Id Barang lebih dominan dari atribut yang lain. Atribut tersebut juga dapat mempengaruhi atribut yang lain. Dengan demikian atribut Id Nota dan Id Barang dapat dijadikan sebagai *candidate key*.

3. Normal Dua (2NF/ 2nd Normal Form)

Tahap ini biasa disebut dengan bentuk normal kedua. Tahap ini hanya bisa dicapai jika telah memenuhi kondisi normal pertama. Tahap ini dibuat untuk menghilangkan ketergantungan parsial atau sebagian. Ketergantungan parsial sendiri merupakan kondisi dimana terdapat atribut yang hanya bergantung pada salah satu key saja.

Tabel 5.4 Ketergantungan Parsial

Kolom1	Kolom2	Kolom3
A	B	C

Sebagai contoh, semisal terdapat sebuah tabel dengan 3 kolom seperti gambar diatas. Dimana kondisinya adalah atribut A dan B merupakan atribut key yang dapat mempengaruhi atribut C (A, B

→C). Namun ternyata atribut C hanya dipengaruhi oleh atribut B saja ($B \rightarrow C$). Kondisi tersebutlah yang dimaksud ketergantungan parsial. Untuk mengatasinya, atribut key harus dipecah menjadi tabel-tabel tersendiri.

Dalam tahap ini, data yang sudah dikelompokkan di tahap normal pertama kemudian di pecah menjadi tabel-tabel tersendiri dan candidate key yang ada akan dijadikan primary key. Jika terdapat lebih dari 1 *candidate key* dalam 1 tabel, maka akan di pilih 1 candiidate key yang benar-benar dapat mewakili 1 record lalu mengubahnya menjadi *primary key*. Namun bisa saja dalam 1 tabel terdapat lebih dari 1 *primary key*. Hal ini terjadi jika key-key tersebut saling melengkapi untuk membuat 1 record tersebut menjadi unik dan akan menimbulkan redudansi jika hanya salah satu key saja yang dijadikan *primary key*.

Primary key sendiri merupakan *candidate key* yang terpilih karena mampu mewakili 1 *record data* dengan baik. Sedangkan jika ada *candidate key* yang tidak terpilih menjadi *primary key* maka akan disebut dengan istilah *alternate key*. Sebagai contoh penerapannya, akan diambil contoh tabel bentuk normal pertama dari poin sebelumnya. Selanjutnya akan diubah menjadi bentuk normal kedua. Dari bentuk normal sebelumnya, maka akan

terbentuk bentuk normal kedua dengan memecah tabel menjadi seperti dibawah ini:

Tabel 5. 5 Tabel Penjualan

Id Nota	Tgl Nota	Total	Id Pelanggan	Nama Pelanggan
N001	10-11-2012	29.000	P001	Andi
N001	10-11-2012	29.000	P001	Andi
N002	11-11-2012	22.000	P002	Ani
N002	11-11-2012	22.000	P002	Ani

Tabel 5. 6 Tabel Detail Penjualan

Id Nota	Id Barang	Qty	Jumlah
N001	B001	3	15.000
N001	B002	2	14.000
N002	B003	10	10.000
N002	B004	4	12.000

Tabel 5. 7 Tabel Barang

Id Barang	Nama Barang	Id Kategori	Kategori	Harga
B001	Pulpen	K001	Alat Tulis	5.000
B002	Buku Tulis	K002	Buku	7.000
B003	Kertas A4	K003	Kertas	500
B004	Pensil	K001	Alat Tulis	3.000

Tabel-tabel 5.5, 5.6, 5.7 terbentuk dengan menghilangkan ketergantungan parsial dari tabel bentuk normal pertama pada poin sebelumnya. Dari tabel-tabel diatas, terdapat beberapa hal yang dapat dibahas. Dimulai dari ketergantungan parsial yang dihilangkan dan penentuan *primary key* dari *candidate key* yang ada.

Ketergantungan parsial yang dihilangkan adalah ketergantungan parsial dari atribut Id Nota dan Id barang. Dimana terdapat beberapa atribut yang hanya bergantung pada atribut Id Nota saja seperti Tgl Nota, Total, Id Pelanggan, dan Nama Pelanggan. Ada pula beberapa atribut lain yang hanya bergantung pada Id Barang saja seperti Nama Barang, Id Kategori, Kategori, dan harga. Sedangkan atribut sisanya bergantung pada Id Nota dan Id Barang sekaligus seperti Qty dan Jumlah.

Untuk penentuan *primary key*, dalam tahap sebelumnya sudah ditetapkan bahwa atribut Id Nota dan Id Barang menjadi *candidate key*. Dengan melihat tabel 5.5 dimana atribut Id Nota menjadi acuan dan tempat atribut lain bergantung, maka atribut Id Nota pantas menjadi *primary key* pada tabel tersebut. Begitu pula pada table 5.7 dimana atribut Id Barang mejadi acuan dan tempat aribut lain bergantung, maka atribut Id Barang pantas menjadi *primary key* pada tabel tersebut. Sedangkan pada tabel 5.6 dimana

atribut Id Nota dan Id Barang sama-sama menjadi acuan dan tempat atribut lain bergantung, maka atribut Id Nota dan Id Barang sama-sama dijadikan primary key pada tabel tersebut.

4. Normal Tiga (3NF / 3rd Normal Form)

Tahap ini biasa disebut dengan bentuk normal ketiga. Tahap ini hanya bisa dicapai jika telah memenuhi kondisi normal kedua. Tahap ini dibuat untuk menghilangkan ketergantungan transitif.

Ketergantungan transitif sendiri merupakan kondisi dimana terdapat atribut yang bergantung pada atribut lain, namun atribut lain tersebut juga bergantung dengan atribut yang lainnya juga.

Tabel 5.8 Ketergantungan Transitif

Kolom1	Kolom2	Kolom3
A	B	C

Sebagai contoh, semisal terdapat sebuah tabel dengan 3 kolom seperti gambar diatas. Dimana kondisinya adalah atribut B bergantung pada atribut A ($A \rightarrow B$), lalu atribut C bergantung pada atribut B ($B \rightarrow C$). Dari Kondisi tersebut, maka bisa disimpulkan bahwa atribut C secara tidak langsung bergantung pada atribut A ($A \rightarrow C$). Untuk mengatasinya, maka tabel tersebut harus dipecah menjadi tabel-tabel yang lebih detail lagi. Sebagai contoh penerapannya, akan diambil contoh tabel bentuk normal kedua dari poin sebelumnya. Selanjutnya akan diubah menjadi bentuk

normal ketiga. Dari bentuk normal sebelumnya, maka akan terbentuk bentuk normal ketiga dengan memecah tabel menjadi seperti dibawah ini

Tabel 5.9 Tabel Penjualan

Id Nota	Tgl Nota	Total	Id Pelanggan
N001	10-11-2012	29.000	P001
N001	10-11-2012	29.000	P001
N002	11-11-2012	22.000	P002
N002	11-11-2012	22.000	P002

Tabel 5.10 Tabel Pelanggan

Id Pelanggan	Nama Pelanggan
P001	Andi
P001	Andi
P002	Ani
P002	Ani

Tabel 5.11 Tabel Detail Penjualan

Id Nota	Id Barang	Qty	Jumlah
N001	B001	3	15.000
N001	B002	2	14.000
N002	B003	10	10.000
N002	B004	4	12.000

Tabel 5.12 Tabel Barang

Id Barang	Nama Barang	Harga	Id Kategori
B001	Pulpen	5.000	K001
B002	Buku Tulis	7.000	K002
B003	Kertas A4	500	K003
B004	Pensil	3.000	K001

Tabel 5.13 Tabel Kategori

Id Kategori	Kategori
K001	Alat Tulis
K002	Buku
K003	Kertas
K001	Alat Tulis

Tabel-tabel diatas terbentuk dengan menghilangkan ketergantungan transitif dari tabel bentuk normal kedua pada poin sebelumnya. Dari tabel-tabel diatas, terdapat beberapa hal yang bisa kita bahas. Hal tersebut adalah ketergantungan transitif yang dihilangkan dan penentuan *primary key* pada tabel yang baru terbentuk.

Pembahasan pertama difokuskan pada ketergantungan transitif yang dihilangkan. Pada tabel 5.5 dipoin sebelumnya terdapat 5 atribut diantaranya: Id Nota, Tgl Nota, Total, Id

Pelanggan dan Nama Pelanggan. Atribut Id Nota menjadi tempat bergantung dari atribut yang lain. Dengan kata lain, jika atribut Id Nota diubah maka atribut lain juga akan berubah. Namun jika melihat konsep ketergantungan transitif yang telah dibahas sebelumnya, terdapat atribut yang tidak secara langsung bergantung pada atribut Id Nota. Atribut tersebut adalah Nama Pelanggan. Hal ini dikarenakan atribut Nama Pelanggan sebenarnya bergantung pada atribut Id Pelanggan. Namun karena atribut Id Pelanggan juga bergantung pada atribut Id Nota, jadi secara tidak langsung atribut Nama Pelanggan juga bergantung pada atribut Id Nota. Sehingga jika atribut Id Nota diubah, maka atribut Id Pelanggan juga akan berubah. Dikarenakan atribut Id Pelanggan berubah, maka atribut Nama Pelanggan juga ikut berubah. Kondisi inilah yang disebut ketergantungan transitif.

Untuk menghilangkan kondisi seperti ini, maka diperlukan pemecahan tabel menjadi lebih terperinci. Oleh karena itu, maka terbentuklah tabel 5.9 dan tabel 5.10. Pada tabel 5.9 terdapat atribut Id Nota, Tgl Nota, Total dan Id Pelanggan. Dimana atribut Tgl Nota, Total, dan Id Pelanggan hanya bergantung pada atribut Id Nota saja. Sedangkan pada tabel 5.10 terdapat atribut Id Pelanggan dan atribut Nama Pelanggan. Dimana atribut Nama Pelanggan hanya bergantung pada atribut Id Pelanggan saja.

Hal yang serupa juga terjadi pada tabel 5.7. Dimana pada tabel tersebut terdapat atribut Id Barang, Nama Barang Id Kategori, Kategori, dan Harga. Atribut Id Barang menjadi tempat bergantung dari atribut yang lain. Dengan begitu, jika atribut Id Barang diubah maka atribut lain juga akan berubah. Namun terdapat atribut yang tidak secara langsung bergantung pada atribut Id Barang. Atribut tersebut adalah Kategori. Hal ini dikarenakan atribut Kategori sebenarnya bergantung pada atribut Id Kategori. Namun karena atribut Id Kategori juga bergantung pada atribut Id Barang, jadi secara tidak langsung atribut Kategori juga bergantung pada atribut Id Barang. Sehingga jika atribut Id Barang diubah, maka atribut Id Kategori juga akan berubah. Dikarenakan atribut Id Kategori berubah, maka atribut Kategori juga ikut berubah. Kondisi ini juga merupakan ketergantungan transitif.

Serupa dengan penanganan sebelumnya, untuk menghilangkan ketergantungan transitif ini maka diperlukan pemecahan tabel menjadi lebih terperinci. Oleh karena itu, maka terbentuklah tabel 5.12 dan tabel 5.13. Pada tabel 5.12 terdapat atribut Id Barang, Nama Barang, Harga, dan Id Kategori. Dimana atribut Nama Barang, Harga, dan Id Kategori hanya bergantung pada atribut Id Barang saja. Sedangkan pada tabel 5.13 terdapat atribut Id Kategori dan atribut Kategori. Dimana atribut Kategori

hanya bergantung pada atribut Id Kategori saja. Pembahasan berikutnya mengenai penentuan *primary key* pada tabel yang baru terbentuk akibat proses penghilangan ketergantungan transitif.

Tabel 5.9 dan tabel 5.10 merupakan pecahan dari tabel 5.5. Pada tabel 5.5, atribut Id Nota merupakan *primary key* pada tabel tersebut. Hal ini dikarenakan atribut lain bergantung pada atribut Id Nota. Pada tabel 5.9, atribut Id Nota juga menjadi *primary key* dikarenakan hal yang sama yaitu atribut lain bergantung pada atribut Id Nota. Sedangkan pada tabel 5.10, atribut Id Pelanggan menjadi *primary key* pada tabel tersebut dengan alasan atribut Nama Pelanggan bergantung pada atribut Id Pelanggan.

Selanjutnya beralih menuju Tabel 5.12 dan tabel 5.13 yang merupakan pecahan dari tabel 5.7. Pada tabel 5.7, atribut Id Barang merupakan *primary key* pada tabel tersebut. Hal ini dikarenakan atribut lain bergantung pada atribut Id Barang. Pada tabel 5.12, atribut Id Barang juga menjadi *primary key* dikarenakan hal yang sama yaitu atribut lain bergantung pada atribut Id Barang. Sedangkan pada tabel 5.13, atribut Id Kategori menjadi *primary key* pada tabel tersebut dengan alasan atribut Kategori bergantung pada atribut Id Kategori.

Khusus untuk tabel 5.6 dan tabel 5.11 tidak perlu diberikan pembahasan lagi. Hal ini dikarenakan kedua tabel tersebut

merupakan tabel yang sama dan tidak mengalami perubahan dikarenakan tidak terdapat ketergantungan transitif didalam tabel tersebut. Untuk pembahasan tambahan, jika diperhatikan dengan lebih detail terdapat atribut yang sama pada tabel 5.9 dan tabel 5.10 yang merupakan pecahan dari tabel 5.5. Atribut tersebut adalah Id Pelanggan. Kondisi ini merupakan contoh dari *foreign key*.

Dimana selain *primary key* dan *alternate key* yang telah dibahas sebelumnya, ada pula yang disebut dengan *foreign key*. *Foreign key* merupakan kunci tamu sebagai hasil dari relasi antar tabel. Terdapatnya *foreign key* ini dimaksudkan agar data antar 2 tabel bisa saling terhubung. Atribut *foreign key* ini biasanya merupakan atribut *primary key* pada tabel asalnya. Sebagai contoh seperti diatas, terdapat tabel 5.5 yang dipecah menjadi tabel 5.9 dan tabel 5.10 sebagai hasil dari penghilangan ketergantungan transitif. Agar data pada tabel 5.9 dan tabel 5.10 tetap terhubung maka dibuat 2 atribut kembar yaitu Id Pelanggan sebagai jembatan penghubungnya. Dimana atribut Id Pelanggan pada tabel 5.10 merupakan *primary key* sedangkan atribut Id Pelanggan pada tabel 5.9 merupakan *foreign key*.

Hal yang sama juga terjadi pada tabel 5.7 yang dipecah menjadi tabel 5.12 dan tabel 5.13 sebagai hasil dari penghilangan ketergantungan transitif. Agar data pada tabel 5.2 dan tabel 5.13

tetap terhubung maka dibuat 2 atribut kembar yaitu Id Kategori sebagai jembatan penghubungnya. Dimana atribut Id Kategori pada tabel 5.13 merupakan primary key sedangkan atribut Id Kategori pada tabel 5.12 merupakan *foreign key*.

5. Tahapan Selanjutnya

Setelah bentuk normal ketiga ini, sebenarnya masih ada beberapa tahap normalisasi lanjutan seperti normal *boyce-codd* (BCNF), normal empat (4NF/*4th normal form*) dan normal lima (5NF/*5th normal form*). Namun banyak yang berhenti pada normal ketiga karena sudah menghilangkan anomali-anomali yang ada.

C. Definisi Denormalisasi

Jika berkaca pada apa yang dilakukan dalam proses normalisasi, proses tersebut dilakukan untuk menghilangkan anomali dan redundansi data yang ada. Namun terkadang terdapat kondisi-kondisi yang membuat hasil dari normalisasi tersebut menjadi bentuk yang tidak normal kembali. Hal ini bisa terjadi karena menginginkan kecepatan akses data dan kemudahan dalam pemanggilan data. Jadi bisa disimpulkan bahwa denormalisasi merupakan kebalikan dari proses normalisasi, yaitu membuat struktur tabel yang sudah normal menjadi tidak normal dimana denormalisasi ini dilakukan untuk kebutuhan kecepatan akses dan kemudahan pemanggilan data. Sebenarnya tidak ada keharusan

untuk melakukan denormalisasi ini. Semuanya kembali pada kebutuhan dan keinginan pemakai dan pembuat database. Tentunya terdapat sisi positif dan negatif dari proses denormalisasi yang dilakukan. Terdapat beberapa kondisi yang memungkinkan untuk munculnya denormalisasi seperti dibawah ini:

1. Atribut Turunan

Atribut turunan merupakan atribut yang bisa dihilangkan karena berasal dari hasil pengolahan dari atribut yang ada. Sebagai contoh, atribut “umur” bisa dihilangkan karena umur merupakan hasil pengolahan dari pengurangan tanggal sekarang dan tanggal lahir. Namun ada kondisi dimana umur ini tetap dimasukan sebagai atribut untuk mengindari proses perhitungan agar proses pemanggilan data menjadi lebih cepat.

2. Tabel Rekapitulasi

Tabel rekapitulasi ini biasanya muncul karena kabutuhan untuk pembuatan laporan. Sedangkan laporan itu sendiri tidak jarang yang membutuhkan gabungan dari beberapa tabel yang dijadikan satu sehingga tercipta sebuah laporan yang diinginkan. Jika menggunakan kondisi tabel yang sudah normal, maka akan membutuhkan query yang rumit untuk memanggil setiap data yang diinginkan dari masing-masing tabel. Tidak hanya itu, program juga membutuhkan waktu untuk memilah dan memilih

data yang diinginkan dari tiap tabelnya lalu menggabungkannya menjadi satu dalam suatu laporan. Jika datanya sangat banyak dan penggunaan *database* saat itu sedang tinggi, maka waktu yang diperlukan untuk membuat dan menampilkan laporan akan sangat lama.

Karena hal itu, dibuat tabel rekapitulasi dengan tujuan untuk memudahkan pembuatan laporan. Dengan adanya tabel rekapitulasi ini, *query* yang dibutuhkan untuk memanggil data menjadi lebih sederhana dan waktu yang dibutuhkan program untuk membuat dan menampilkan laporan menjadi singkat karena bisa langsung mengakses tabel rekapitulasi tanpa harus menggabungkan data dari berbagai tabel. Namun bentuk dari tabel rekapitulasi ini tentu saja tidak berdasarkan proses normalisasi yang ada. Isi dari tabel rekapitulasi akan lebih seperti bentuk tidak normal pada normalisasi karena merupakan gabungan dari beberapa tabel yang terpisah.

3. Menggabungkan 2 Tabel

Kondisi ini biasanya terjadi pada 2 tabel dengan relasi *one to one* (1:1). Dimana hal ini dilakukan untuk memudahkan dalam penulisan *query* dan mempercepat pengaksesan data. Sebagai contoh, jika terdapat 2 tabel seperti dibawah ini:

Tabel 5. 14 Tabel Mahasiswa

Id Mahasiswa	Nama Mahasiswa	Jenis Kelamin
M001	Ani	Perempuan
M002	Budi	Laki-laki
M003	Anto	Laki-laki

Tabel 5. 15 Tabel Anggota Perpustakaan

Id Anggota	Id Mahasiswa	Tanggal Daftar	Masa Berlaku
P001	M001	01-01-2010	01-01-2013
P002	M002	15-01-2010	15-01-2013
P003	M003	22-02-2012	22-02-2015

Kemudian 2 tabel diatas digabung menjadi satu seperti dibawah ini:

Tabel 5. 16 Tabel Data Mahasiswa

<u>Id Mahasiswa</u>	<u>Nama Mahasiswa</u>	<u>Jenis Kelamin</u>	<u>Id Anggota</u>	<u>Tanggal Daftar</u>	<u>Masa Berlaku</u>
M001	Ani	Perempuan	P001	01-01-2010	01-01-2013
M002	Budi	Laki-laki	P002	15-01-2010	15-01-2013
M003	Anto	Laki-laki	P003	22-02-2012	22-02-2015

Dengan kondisi seperti diatas, tentu akan lebih mudah dalam penulisan query dan pengaksesan data. Namun akan terjadi ketergantungan fungsional, dimana kondisi tersebut seperti mundur dari normal ketiga ke bentuk normal pertama atau kedua.

Selain itu, terdapat beberapa resiko dengan kondisi seperti ini. Resiko pertama, akan terdapat beberapa data kosong jika mahasiswa tersebut tidak mendaftar menjadi anggota perpustakaan. Resiko kedua, data mahasiswa akan terhapus jika menghapus data anggota perpustakaan, begitu pula sebaliknya.

4. Menduplikat Atribut

Kondisi ini biasanya terjadi pada 2 tabel dengan relasi *one to many* (1:M). Penyebab denormalisasi ini tidak jauh berbeda dengan poin sebelumnya, dimana hal ini dilakukan untuk memudahkan dalam penulisan query dan mempercepat pengaksesan data. Semua kembali kepada kebutuhan yang diinginkan. Sebagai contoh, jika terdapat 2 tabel seperti dibawah ini:

Tabel 5. 17 Tabel Mahasiswa

Id Mahasiswa	Nama Mahasiswa	Jenis Kelamin
M001	Ani	Perempuan
M002	Budi	Laki-laki
M003	Anto	Laki-laki

Tabel 5. 18 Tabel Peminjaman Buku

<u>Id Peminjaman</u>	<u>Id Mahasiswa</u>	<u>Id Buku</u>	<u>Tanggal Pinjam</u>	<u>Tanggal Kembali</u>
P001	M001	B001	01-01-2010	05-01-2010
P002	M002	B002	15-01-2010	20-01-2010
P003	M003	B003	22-02-2012	27-02-2012

Kemudian atribut “Nama Mahasiswa” pada tabel mahasiswa diduplikat ke tabel peminjaman buku. Sehingga berbentuk seperti dibawah ini:

Tabel 5. 19 Tabel Peminjaman Buku

<u>Id Peminjaman</u>	<u>Id Mahasiswa</u>	<u>Nama Mahasiswa</u>	<u>Id Buku</u>	<u>Tanggal Pinjam</u>	<u>Tanggal Kembali</u>
P001	M001	Ani	B001	01-01-2010	05-01-2010
P002	M002	Budi	B002	15-01-2010	20-01-2010
P003	M003	Anto	B003	22-02-2012	27-02-2012

Dengan kondisi seperti diatas, tentu akan lebih mudah dalam penulisan query dan pengaksesan data. Hal ini dikarenakan atribut “Nama Mahasiswa” sudah ada pada tabel peminjaman buku, sehingga tidak perlu lagi mengakses tabel mahasiswa untuk mendapatkan data nama mahasiswa. Atribut yang diduplikat tidak terbatas hanya pada atribut non kunci saja, namun bisa terjadi pula pada atribut kunci dengan tujuan dan penyebab yang sama.

Sisi negatif dari kondisi ini pun hampir serupa dengan poin sebelumnya, dimana akan terjadi ketergantungan fungsional yang menyebabkan kondisi tersebut seperti mundur dari normal ketiga ke bentuk normal pertama atau kedua.

5. Pembatasan Atribut

Kondisi ini biasanya terjadi pada 2 tabel dengan relasi *one to many* (1:M). Penyebab denormalisasi ini tidak jauh berbeda juga dengan

poin sebelumnya, dimana hal ini dilakukan untuk memudahkan dalam penulisan query dan mempercepat pengaksesan data.

Sebagai contoh, jika terdapat 2 tabel seperti dibawah ini:

Tabel 5. 20 Tabel Peminjaman Buku

Id Peminjaman	Id Mahasiswa	Tanggal Pinjam	Tanggal Kembali
P001	M001	01-01-2010	05-01-2010
P002	M002	15-01-2010	20-01-2010
P003	M003	22-02-2012	27-02-2012

Tabel 5. 21 Tabel Detail Peminjaman

Id Peminjaman	Id Buku
P001	B001
P002	B002
P003	B003

Kemudian 2 tabel diatas digabung menjadi satu seperti dibawah ini:

Tabel 5. 22 Tabel Peminjaman Buku

Id Peminjaman	Id Mahasiswa	Id Buku 1	Id Buku 2	Id Buku 3	Tanggal Pinjam	Tanggal Kembali
P001	M001	B001	B002	B003	01-01-2010	05-01-2010
P002	M002	B003	B001	B002	15-01-2010	20-01-2010
P003	M003	B002	B003	B001	22-02-2012	27-02-2012

Dengan kondisi seperti diatas, tentu akan lebih mudah dalam penulisan query dan pengaksesan data. Hal ini dikarenakan dengan hanya 1 tabel saja sudah terdapat data peminjaman yang lengkap beserta buku apa saja yang dipinjam.

Namun akan terjadi pembatasan atribut, dimana buku yang bisa dipinjam hanya sebanyak 3 buku saja dalam 1 kali peminjaman. Jika suatu saat terdapat peraturan baru dimana anggota perpustakaan dapat meminjam 5 buku dalam 1 kali peminjaman, maka tabel peminjaman harus menambah kolom baru untuk menampung data buku ke 4 dan ke 5 dalam peminjaman tersebut. Jika kolom sudah ditambahkan, maka data peminjaman buku yang lama akan memiliki kolom kosong pada data buku ke 4 dan ke 5 dikarenakan tidak ada data yang terinput pada data tersebut. Hal ini akan mempengaruhi kinerja dari aplikasi yang dipakai.

Selain kondisi-kondisi yang sudah dijelaskan, masih banyak lagi kondisi lain yang memungkinkan untuk terjadinya denormalisasi. Baik normalisasi maupun denormalisasi memiliki sisi positif dan negatif masing-masing. Semuanya kembali lagi pada kebutuhan dan keinginan masing-masing.

DAFTAR PUSTAKA

- Alia, P. A., Prayogo, J. S., Kartiko, E. Y., Prasetyo, D., Khairunusi, Y., Na'am, J., Wijaya, A., Setyadi, A. T., Remawati, D., Mair, Z. R., Febriana, R. W., Setyadinsa, R., Maspupah, A., & Cahyono, W. A. (2023). Sistem basis data. *Sustainability (Switzerland)*, 11(1). PT Penamuda Media.
- Harianto, K. (1994). *Konsep & perancangan database*. ANDI.
- Jamaluddin, S., Samosir, K., S, W., Devia, E., Santoso, L. W., Yuniansyah, Juanaidi, Nursari, S. R. C., Azizah, N., & Saputra, M. H. (2022). *BUKU (Book chapter)-Sistem basis data (Elmi Devia)*. PT Global Eksekutif Teknologi.
- Putri, R. A. (2022). *Buku ajar basis data edisi kedua* (2nd ed.). CV. Media Sains Indonesia.
- Qalbi, I. A. (2014). *Konsep dasar basis data*. Universitas Mulawarman.
- Soyusiawaty, D. (2020). *Buku ajar basis data informatika* (p. 104). Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Ahmad Dahlan

BAB 6

MENGHASILKAN *SCRIPT* UNTUK MENGIMPLEMENTASIKAN DESAIN LOGIK KE DESAIN FISIK BASIS DATA

Oleh : Ir. Hilmansyah Gani, S.Kom., M.Kom

Dalam pengembangan sistem informasi modern, basis data memiliki peran yang sangat vital. Basis data tidak hanya menjadi tempat penyimpanan data, tetapi juga menjadi dasar untuk pengambilan keputusan strategis dalam organisasi. Oleh karena itu, proses perancangan basis data harus dilakukan dengan cermat dan terstruktur. Salah satu tahapan penting dalam perancangan basis data adalah mengonversi desain logik menjadi desain fisik yang dapat diimplementasikan pada sebuah sistem manajemen basis data (DBMS).

Desain logik basis data merepresentasikan kebutuhan data dalam bentuk model konseptual yang tidak bergantung pada platform teknologi tertentu. Model ini biasanya berupa diagram *Entity-Relationship* (ER) yang menggambarkan entitas, atribut, dan hubungan antar-entitas. Desain logik memberikan pandangan

abstrak tentang bagaimana data harus disusun untuk memenuhi kebutuhan bisnis.

Namun, desain logik tidak dapat langsung digunakan dalam DBMS. Untuk itu, diperlukan langkah-langkah untuk mengonversinya menjadi desain fisik. Desain fisik adalah bentuk implementasi nyata dari desain logik dalam DBMS tertentu, mencakup pemilihan tipe data, indeks, partisi, dan berbagai aturan integritas yang harus diterapkan. Proses ini melibatkan pembuatan script *Structured Query Language* (SQL) yang akan digunakan untuk membuat, mengelola, dan memanipulasi basis data.

A. Konsep Dasar Desain Logik dan Fisik Basis Data

Desain Logik Basis Data

Desain logik adalah representasi abstrak dari kebutuhan data yang mencerminkan struktur dan hubungan antar entitas dalam basis data. Tahapan ini penting untuk memastikan bahwa semua kebutuhan data organisasi teridentifikasi dan dimodelkan dengan baik tanpa memperhatikan aspek teknis dari implementasi.

Elemen Utama Desain Logik:

- a. Entitas: Objek utama yang memiliki atribut dan dapat diidentifikasi secara unik, seperti "Pelanggan" atau "Produk".
- b. Atribut: Karakteristik atau properti dari entitas, misalnya "Nama" dan "Alamat" untuk entitas Pelanggan.

- c. Relasi: Hubungan antar entitas, misalnya "Pelanggan melakukan Pemesanan".
- d. *Primary Key*: Atribut atau kombinasi atribut yang secara unik mengidentifikasi setiap entitas.
- e. *Foreign Key*: Atribut yang menghubungkan satu entitas dengan entitas lainnya.

Diagram ER: Desain logik sering divisualisasikan menggunakan Diagram *Entity-Relationship* (ER) yang membantu memahami struktur data secara keseluruhan.

Normalisasi: Proses normalisasi digunakan untuk menghilangkan redundansi data dan memastikan integritas data. Langkah-langkah normalisasi meliputi:

- a. 1NF (*First Normal Form*): Menghilangkan atribut yang mengandung grup berulang.
- b. 2NF (*Second Normal Form*): Menghilangkan ketergantungan parsial pada *primary key*.
- c. 3NF (*Third Normal Form*): Menghilangkan ketergantungan transitif.

Desain Fisik Basis Data: Desain fisik adalah implementasi teknis dari desain logik pada platform DBMS tertentu. Tahapan ini

bertujuan untuk memastikan bahwa basis data dirancang untuk kinerja yang optimal dan sesuai dengan kebutuhan sistem.

Elemen Utama Desain Fisik:

- a. Tipe Data: Memilih tipe data yang tepat untuk setiap atribut, seperti INTEGER, VARCHAR, DATE, dll.
- b. Indeks: Meningkatkan kecepatan akses data dengan menambahkan indeks pada kolom yang sering digunakan dalam pencarian.
- c. Partisi: Membagi tabel besar menjadi bagian-bagian yang lebih kecil untuk meningkatkan kinerja.
- d. *Constraints*: Menambahkan aturan integritas seperti *primary key*, *foreign key*, *unique*, dan *check constraints*.

Contoh Konversi: Misalnya, entitas "Pelanggan" dengan atribut "ID Pelanggan" (*Primary Key*), "Nama", dan "Alamat" dapat dikonversi menjadi script SQL:

```
CREATE TABLE Pelanggan (  
    ID_Pelanggan INT PRIMARY KEY,  
    Nama VARCHAR(100),  
    Alamat TEXT  
);
```

Optimasi Desain Fisik: Optimasi dilakukan untuk memastikan bahwa basis data dapat menangani beban kerja yang tinggi dengan efisien. Beberapa teknik optimasi meliputi:

- a. Penggunaan indeks pada kolom yang sering digunakan dalam WHERE clause.
- b. Denormalisasi pada tabel tertentu untuk mengurangi jumlah join dalam query.
- c. Pengaturan konfigurasi spesifik DBMS untuk mendukung kinerja.

Desain logik dan fisik merupakan dua tahap yang saling melengkapi. Desain logik memastikan bahwa data direpresentasikan secara akurat sesuai kebutuhan bisnis, sedangkan desain fisik memastikan bahwa basis data dapat berfungsi secara optimal dalam lingkungan teknis yang spesifik.

B. Proses Transformasi Desain Logik ke Fisik

Transformasi dari desain logik ke desain fisik adalah langkah krusial dalam proses implementasi basis data. Proses ini melibatkan penerjemahan elemen-elemen desain logik ke dalam bentuk yang dapat diimplementasikan pada DBMS tertentu.

1. Identifikasi Entitas dan Atribut:

- a. Setiap entitas dalam desain logik diterjemahkan menjadi tabel.
- b. Atribut entitas menjadi kolom dalam tabel.
- c. Tentukan tipe data untuk setiap atribut berdasarkan sifat data yang akan disimpan.

2. Penentuan Kunci Utama (*Primary Key*):

- a. Tetapkan kolom atau kombinasi kolom yang dapat digunakan sebagai kunci utama pada setiap tabel.

3. Definisi Relasi Antar-Tabel:

- a. Relasi antar-entitas diterjemahkan menjadi *foreign key* di tabel terkait.
- b. Jika hubungan adalah one-to-many, tambahkan *foreign key* di tabel sisi "*many*".
- c. Jika hubungan adalah many-to-many, buat tabel baru untuk merepresentasikan hubungan tersebut.

4. Penerapan Constraints:

- a. Terapkan constraints seperti *NOT NULL*, *UNIQUE*, *CHECK*, dan *DEFAULT* untuk memastikan integritas data.

5. Penambahan Indeks:

- a. Identifikasi kolom yang sering digunakan dalam operasi pencarian atau join, lalu tambahkan indeks pada kolom tersebut.

6. Partisi dan Pengaturan Fisik:

- a. Jika tabel besar, pertimbangkan untuk membagi tabel menjadi partisi-partisi untuk meningkatkan kinerja.

Contoh Proses Transformasi:

Diagram ER:

- a. Entitas: Pelanggan(ID_Pelanggan, Nama, Alamat)
- b. Entitas: Pesanan(ID_Pesanan, Tanggal, ID_Pelanggan)
- c. Relasi: Pelanggan melakukan Pesanan (*One-to-Many*)

Transformasi ke SQL:

```
CREATE TABLE Pelanggan (  
    ID_Pelanggan INT PRIMARY KEY,  
    Nama VARCHAR(100) NOT NULL,  
    Alamat TEXT  
);  
  
CREATE TABLE Pesanan (  
    ID_Pesanan INT PRIMARY KEY,  
    Tanggal DATE NOT NULL,  
    ID_Pelanggan INT,  
    FOREIGN KEY (ID_Pelanggan) REFERENCES Pelanggan(ID_Pelanggan)  
);
```

Dengan proses transformasi yang terstruktur, desain fisik dapat diimplementasikan secara efisien, memastikan bahwa basis data mampu mendukung kebutuhan operasional dan analitik organisasi.

c. Implementasi dengan SQL *Data Definition Language* (DDL)

SQL DDL (*Data Definition Language*) adalah sekumpulan perintah SQL yang digunakan untuk mendefinisikan struktur basis data.

Dengan DDL, kita dapat membuat, mengubah, dan menghapus objek basis data seperti tabel, indeks, dan skema.

Perintah Utama dalam DDL:

CREATE: Digunakan untuk membuat objek basis data baru.

Contoh:

```
CREATE TABLE Produk (  
    ID_Produk INT PRIMARY KEY,  
    Nama_Produk VARCHAR(100),  
    Harga DECIMAL(10, 2),  
    Stok INT  
);
```

1. ALTER: Digunakan untuk mengubah struktur tabel yang sudah ada.

```
ALTER TABLE Produk ADD Kategori VARCHAR(50);
```

2. DROP: Digunakan untuk menghapus objek basis data.

```
DROP TABLE Produk;
```

3. TRUNCATE: Menghapus semua data dalam tabel tanpa menghapus struktur tabel.

```
TRUNCATE TABLE Produk;
```

Contoh Studi Kasus Implementasi:

Diagram ER:

- a. Entitas: Karyawan(ID_Karyawan, Nama, Jabatan, Gaji)

- b. Entitas: Proyek (ID_Projek, Nama_Projek, Tanggal_Mulai, ID_Karyawan)
- c. Relasi: Karyawan mengelola Proyek (*One-to-Many*)

Implementasi SQL DDL:

```
CREATE TABLE Karyawan (  
    ID_Karyawan INT PRIMARY KEY,  
    Nama VARCHAR(100) NOT NULL,  
    Jabatan VARCHAR(50),  
    Gaji DECIMAL(10, 2)
```

D. Optimisasi Desain Fisik

Optimisasi desain fisik adalah langkah penting untuk memastikan bahwa basis data tidak hanya memenuhi kebutuhan fungsional, tetapi juga dapat bekerja secara efisien dalam skenario beban kerja nyata. Langkah ini melibatkan berbagai teknik untuk meningkatkan kinerja akses data, penggunaan sumber daya, dan skalabilitas.

Teknik-Teknik Optimisasi Desain Fisik:

1. Penggunaan Indeks yang Efisien
 - a. Indeks meningkatkan kecepatan akses data, tetapi juga memerlukan ruang penyimpanan tambahan dan memperlambat operasi tulis seperti *INSERT*, *UPDATE*, dan *DELETE*.

- b. Gunakan indeks hanya pada kolom yang sering digunakan dalam operasi pencarian atau pengurutan.

```
CREATE INDEX idx_nama_produk ON Produk(Nama_Produk);
```

2. Partisi Tabel

- a. Membagi tabel besar menjadi beberapa partisi berdasarkan kriteria tertentu, seperti rentang tanggal atau nilai geografis.
- b. Hal ini meningkatkan kinerja query dengan memproses hanya partisi yang relevan.

```
CREATE TABLE Transaksi (  
    ID_Transaksi INT,  
    Tanggal DATE,  
    Jumlah DECIMAL(10, 2)  
) PARTITION BY RANGE (Tanggal) (  
    PARTITION p1 VALUES LESS THAN ('2024-01-01'),  
    PARTITION p2 VALUES LESS THAN ('2025-01-01')  
);
```

3. Denormalisasi

Denormalisasi dilakukan dengan menambahkan redundansi data secara terkendali untuk mengurangi jumlah join yang diperlukan dalam query. Contoh: Tambahkan kolom "Nama_Karyawan" langsung ke tabel Proyek untuk menghindari join dengan tabel Karyawan.

4. *Materialized Views*

- a. Gunakan materialized views untuk menyimpan hasil query kompleks yang sering diakses.

```
CREATE MATERIALIZED VIEW Total_Penjualan AS
SELECT ID_Produk, SUM(Jumlah) AS Total
FROM Penjualan
GROUP BY ID_Produk;
```

5. Konfigurasi Parameter DBMS

- a. Sesuaikan parameter seperti ukuran cache, jumlah koneksi maksimum, dan strategi eksekusi query sesuai kebutuhan aplikasi.

6. Pemantauan dan Analisis Kinerja

- a. Gunakan alat pemantauan bawaan DBMS untuk mengidentifikasi query yang lambat atau tabel yang sering diakses.
- b. Lakukan analisis rutin untuk mengoptimalkan struktur basis data.

Studi Kasus Optimisasi:

Masalah: Query penjualan produk tertentu lambat karena tabel transaksi berisi jutaan baris.

Solusi:

1. Tambahkan indeks pada kolom "ID_Produk" di tabel Transaksi:

```
CREATE INDEX idx_id_produk ON Transaksi(ID_Produk);
```

2. Gunakan partisi tabel berdasarkan tahun transaksi.
3. Buat materialized view untuk total penjualan per produk:

```
CREATE MATERIALIZED VIEW Total_Penjualan AS  
SELECT ID_Produk, SUM(Jumlah) AS Total  
FROM Transaksi  
GROUP BY ID_Produk;
```

Dengan menggabungkan berbagai teknik optimisasi, basis data dapat menangani beban kerja yang tinggi dengan kinerja yang optimal. Optimisasi yang tepat juga membantu meminimalkan penggunaan sumber daya dan meningkatkan pengalaman pengguna secara keseluruhan.

DAFTAR PUSTAKA

- Connolly, T., & Begg, C. (2015). *Database systems: A practical approach to design, implementation, and management* (6th ed.). Pearson Education.
- Date, C. J. (2006). *An introduction to database systems* (8th ed.). Pearson Education.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson Education.
- Kroenke, D. M., & Auer, D. J. (2013). *Database concepts* (7th ed.). Pearson Education.
- Microsoft. (2021). *SQL Server documentation*.
- Oracle Corporation. (2021). *Oracle Database SQL Language Reference*. Official Oracle documentation providing details on SQL DDL and its implementation.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill Education.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database system concepts* (7th ed.). McGraw-Hill Education.
- Titman, S., Keown, A. J., & Martin, J. D. (2021). *Financial management: Principles and applications* (latest ed.). Pearson.

Ullman, J. D., & Widom, J. (2008). *A first course in database systems* (3rd ed.). Pearson Education.

MySQL Documentation. (2021). *MySQL 8.0 Reference Manual*.

BAB 7

CARA MEMBUAT MODEL ER DENGAN METODE CHEN

Oleh : Sulistianto Sutrisno Wanda, A.MI, S.Pd, M.M, M.Kom

Membuat model *Entity-Relationship* (ER) dengan metode Chen adalah proses untuk memvisualisasikan struktur data dan hubungan antara entitas dalam sebuah sistem. Model ini sering digunakan dalam perancangan basis data. Berikut adalah kerangka konsep pembelajaran untuk membuat model ER dengan metode Chen:

A. Pengantar ke Model ER dan Metode Chen

1. Definisi Model *Entity-Relationship* (ER)

Model *Entity-Relationship* (ER) adalah representasi grafis dari struktur data yang mencerminkan entitas dalam suatu sistem dan hubungan antara entitas-entitas tersebut. Model ini digunakan untuk merancang dan mengatur basis data, memastikan data terstruktur dengan cara yang logis dan efisien. Model ER adalah alat visual yang membantu pengembang dan desainer sistem

untuk memahami dan mendokumentasikan persyaratan data dengan cara yang mudah dipahami.

2. Tujuan dan Manfaat Model ER

Visualisasi Struktur Data yaitu memudahkan pemahaman tentang bagaimana data terkait dalam suatu sistem. Komunikasi yang Jelas: Memfasilitasi komunikasi antara analis sistem, pengembang, dan pengguna bisnis. Perencanaan Basis Data yaitu membantu dalam merancang skema basis data yang efektif dan efisien. Analisis dan Validasi yaitu membantu dalam mengidentifikasi kebutuhan data dan validasi persyaratan bisnis.

3. Sejarah dan Pengembangan Metode Chen

Metode ER pertama kali diperkenalkan oleh Peter Chen pada tahun 1976 dalam makalahnya yang berjudul "*The Entity-Relationship Model - Toward a Unified View of Data.*" Metode ini menjadi salah satu model yang paling populer untuk merancang basis data konseptual karena kesederhanaan dan fleksibilitasnya. Chen mengusulkan penggunaan diagram ER sebagai alat untuk menyatukan berbagai pandangan data dari pengguna dan pengembang, serta untuk menyediakan basis yang konsisten untuk desain basis data logis dan fisik.

4. Konsep Dasar dalam Metode Chen

Entitas yaitu Sesuatu yang memiliki keberadaan yang dapat diidentifikasi dalam konteks sistem, misalnya, orang, tempat, atau objek. Dalam diagram, entitas digambarkan dengan persegi panjang. Atribut yaitu karakteristik atau properti dari entitas. Atribut bisa berupa kunci utama (*primary key*) yang unik atau atribut biasa. Dalam diagram, atribut digambarkan dengan elips yang terhubung ke entitasnya. Hubungan (*Relationship*): Menggambarkan bagaimana entitas-entitas berinteraksi satu sama lain. Hubungan digambarkan dengan berlian, dan garis menghubungkan berlian ke entitas-entitas yang terlibat. Kardinalitas: Menyatakan jumlah minimum dan maksimum dari keterlibatan entitas dalam suatu hubungan. Kardinalitas biasanya ditunjukkan dalam bentuk (1:1), (1), atau (M).

5. Perbedaan dengan Metode ER Lain

Metode Chen menonjol dengan notasi dan fokusnya yang detail pada atribut dan hubungan. Berbeda dengan model ER lainnya, seperti notasi Crow's Foot, yang lebih fokus pada penggunaan simbol yang lebih sederhana dan penggambaran hubungan tanpa terlalu banyak detail pada atribut. Metode Chen lebih tepat digunakan dalam tahap awal analisis dan desain konseptual karena lebih ekspresif dan detail.

6. Kelebihan dan Kekurangan Metode Chen

Kelebihan:

Detail dan jelas, membantu dalam pengumpulan dan dokumentasi persyaratan. Memudahkan identifikasi atribut dan hubungan yang mungkin diabaikan. Berguna dalam perencanaan dan desain awal basis data.

Kekurangan:

Bisa menjadi kompleks dan sulit dipahami jika ada terlalu banyak entitas dan atribut. Tidak seefisien notasi lain dalam representasi yang lebih ringkas. Kerangka pengantar ini memberikan dasar yang kuat bagi para pembelajar untuk memahami konsep dasar dan pentingnya model ER, khususnya metode Chen, dalam perancangan sistem informasi dan basis data.

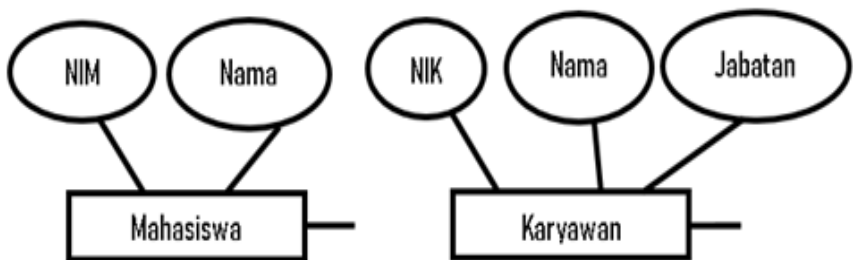
B. Komponen Dasar dalam Metode Chen

Metode Chen dalam pemodelan *Entity-Relationship* (ER) memiliki beberapa komponen dasar yang digunakan untuk merepresentasikan data dalam sebuah sistem. Komponen-komponen ini membantu dalam menggambarkan entitas, atribut, dan hubungan yang ada. Berikut adalah penjelasan detail dari masing-masing komponen:

1. Entitas

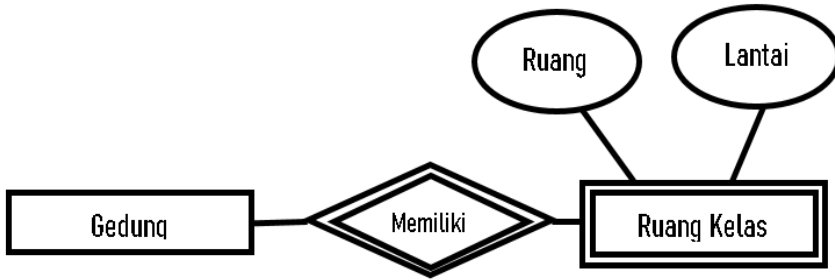
Entitas adalah objek atau konsep yang memiliki keberadaan yang dapat diidentifikasi dalam sistem. Entitas dapat berupa benda nyata seperti manusia, produk, atau tempat, maupun konsep abstrak seperti transaksi atau proyek. Penggambaran dalam Diagram: Dalam diagram ER, entitas digambarkan dengan bentuk persegi panjang. Nama entitas biasanya ditulis di dalam persegi panjang ini. Jenis Entitas:

- a. Entitas Independen (*Strong Entity*): Entitas yang keberadaannya tidak bergantung pada entitas lain. Contoh: "Mahasiswa", "Karyawan".



Gambar 7.1 Entitas Independen

- b. Entitas Tergantung (*Weak Entity*): Entitas yang keberadaannya tergantung pada entitas lain, biasanya tidak memiliki kunci utama sendiri. Contoh: "Ruang Kelas" (tergantung pada "Gedung").

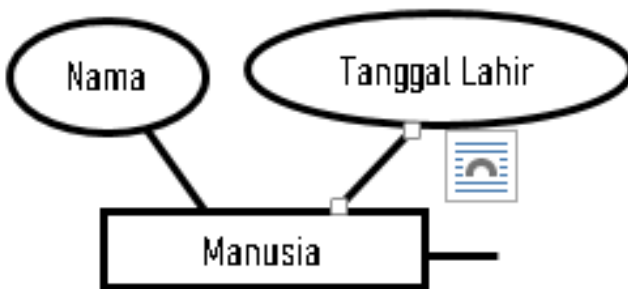


Gambar 7. 2 Entitas Lemah

2. Atribut

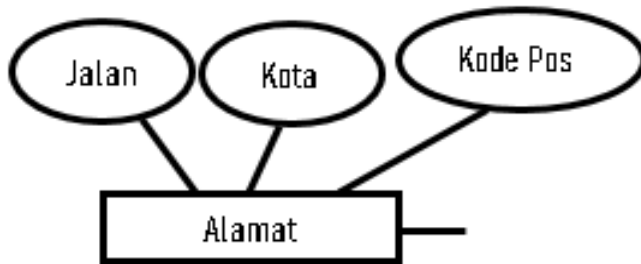
Atribut adalah karakteristik atau properti yang dimiliki oleh entitas. Atribut memberikan informasi tambahan tentang entitas dan membantu membedakan satu entitas dari entitas lainnya. Penggambaran dalam Diagram: Atribut digambarkan dengan *elips* yang terhubung ke persegi panjang entitas. Jenis Atribut:

- Atribut Sederhana: Atribut yang tidak dapat dipecah menjadi bagian yang lebih kecil. Contoh: "Nama", "Tanggal Lahir".



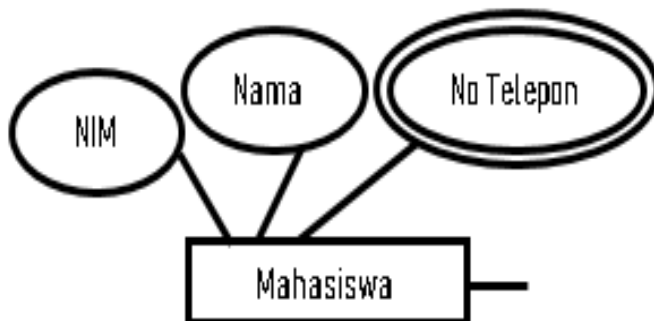
Gambar 7. 3 Atribut Sederhana

- b. Atribut Komposit: Atribut yang terdiri dari beberapa komponen.
Contoh: "Alamat" bisa terdiri dari "Jalan", "Kota", "Kode Pos".



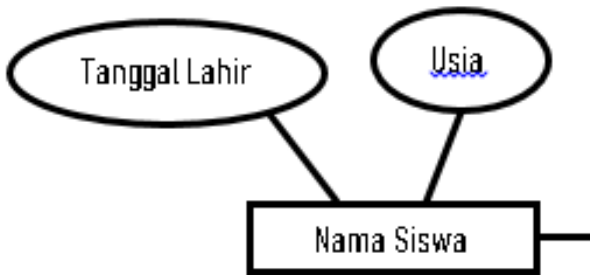
Gambar 7.4 Atribut Komposit

- c. Atribut Multivalued: Atribut yang bisa memiliki lebih dari satu nilai. Contoh: "Nomor Telepon" untuk seseorang.



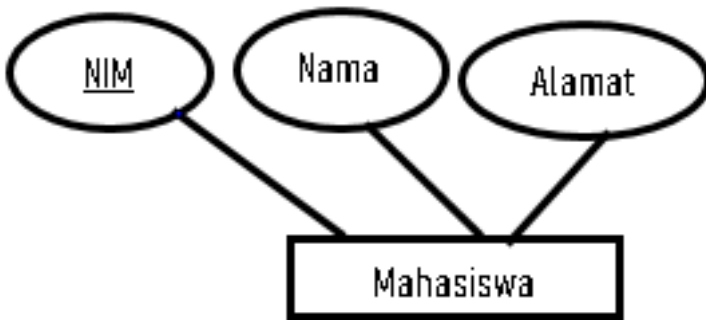
Gambar 7.5 Atribut *Multivalued*

- d. Atribut Derivatif: Atribut yang nilainya dapat diturunkan dari atribut lain. Contoh: "Usia" bisa dihitung dari "Tanggal Lahir"



Gambar 7. 6 Atribut Derivatif

- e. Kunci Utama (*Primary Key*): Atribut atau kombinasi atribut yang secara unik mengidentifikasi sebuah entitas dalam kumpulan entitas. Digarisbawahi dalam diagram untuk menandai bahwa itu adalah kunci utama.



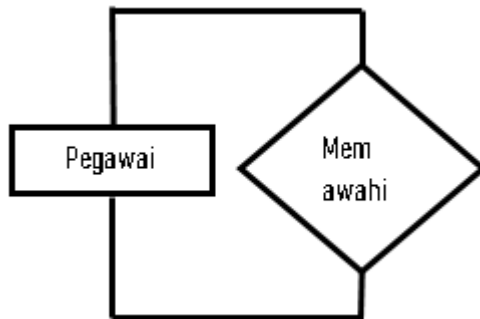
Gambar 7. 7 Atribut dengan *Primary Key*

3. Hubungan (*Relationship*)

Hubungan menunjukkan bagaimana dua atau lebih entitas berinteraksi satu sama lain dalam sistem. Penggambaran dalam Diagram: Hubungan digambarkan dengan berlian, dan garis

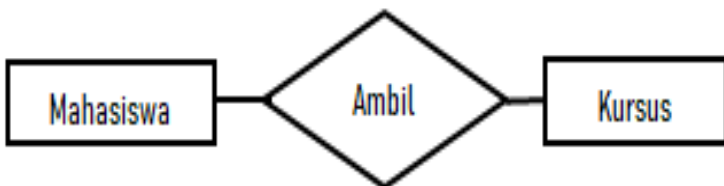
menghubungkan berlian ke entitas-entitas yang terlibat dalam hubungan tersebut. Jenis Hubungan:

- a. Hubungan *Unary* (Rekursif): Hubungan di mana entitas berhubungan dengan dirinya sendiri. Contoh: "Karyawan" yang membawahi "Karyawan" lainnya.



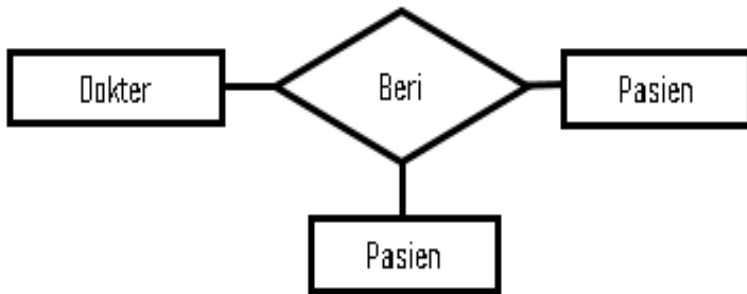
Gambar 7. 8 Hubungan Unary

- b. Hubungan *Binary*: Hubungan antara dua entitas. Contoh: "Mahasiswa" mengambil "Kursus".



Gambar 7. 9 Hubungan Binary

- c. Hubungan Ternary: Hubungan yang melibatkan tiga entitas. Contoh: "Dokter" memberikan "Resep" untuk "Pasien".



Gambar 7.10 Hubungan Ternary

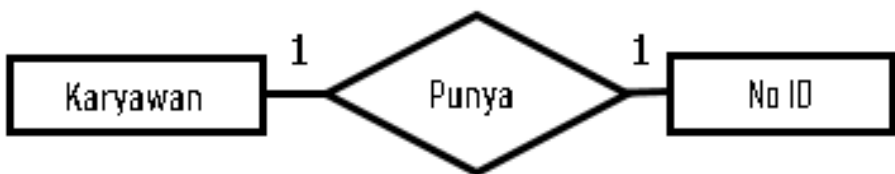
4. Kardinalitas

Kardinalitas mendefinisikan jumlah minimum dan maksimum dari instansi entitas yang dapat berpartisipasi dalam suatu hubungan.

Jenis Kardinalitas:

a. *One-to-One* (1:1)

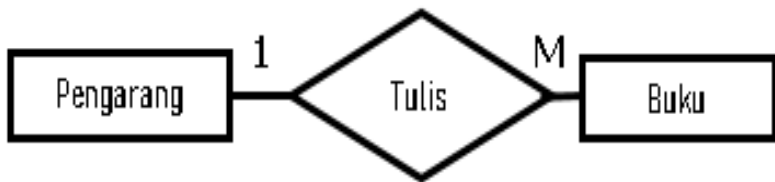
Setiap instansi dari satu entitas berhubungan dengan satu instansi dari entitas lain. Contoh: Setiap karyawan memiliki satu nomor ID.



Gambar 7.11 Kardinalitas 1 to 1

b. *One-to-Many* (M)

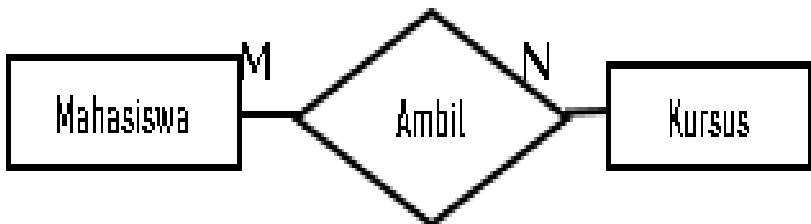
Satu instansi dari satu entitas dapat berhubungan dengan banyak instansi dari entitas lain. Contoh: Satu pengarang bisa menulis banyak buku.



Gambar 7. 12 Kardinalitas 1 to M

c. *Many-to-Many* (N)

Banyak instansi dari satu entitas dapat berhubungan dengan banyak instansi dari entitas lain. Contoh: Banyak mahasiswa dapat mengambil banyak kursus



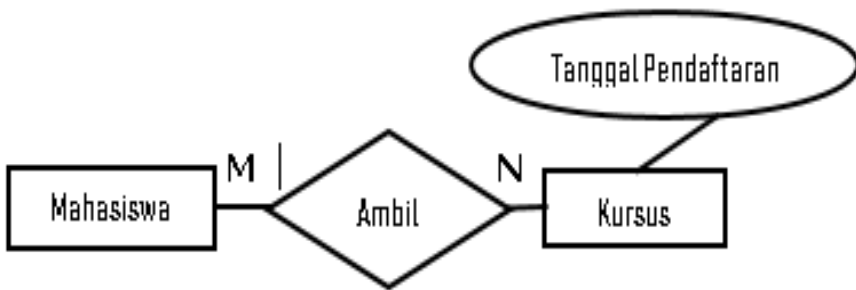
Gambar 7. 13 Kardinalitas M to N

4. Atribut pada Hubungan

Kadang-kadang, hubungan juga memiliki atribut, terutama dalam kasus hubungan banyak ke banyak (M). Atribut pada hubungan

menyimpan informasi tambahan yang tidak dapat ditempatkan pada entitas yang terlibat. Contoh: Dalam hubungan "Mahasiswa" mendaftar untuk "Kursus", atribut "Tanggal Pendaftaran" dapat ditambahkan untuk mencatat kapan pendaftaran terjadi.

Dengan memahami komponen dasar ini, seseorang dapat merancang model ER yang efektif dan efisien menggunakan metode Chen, menggambarkan semua entitas, atribut, dan hubungan yang relevan dengan cara yang terstruktur dan mudah dipahami.



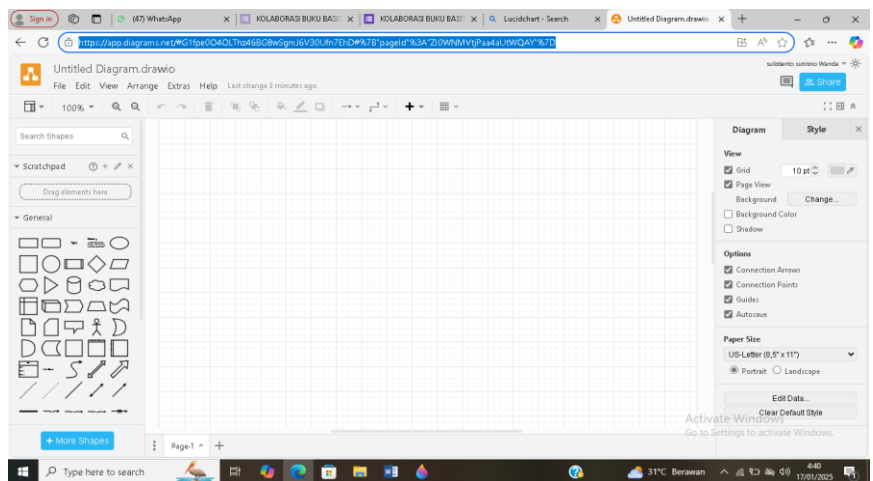
Gambar 7. 14 Kardinalitas M to N + Atribut tambahan

5. Atribut pada Hubungan

Untuk membuat diagram ER yang profesional dan mudah dimodifikasi, gunakan alat desain diagram seperti:

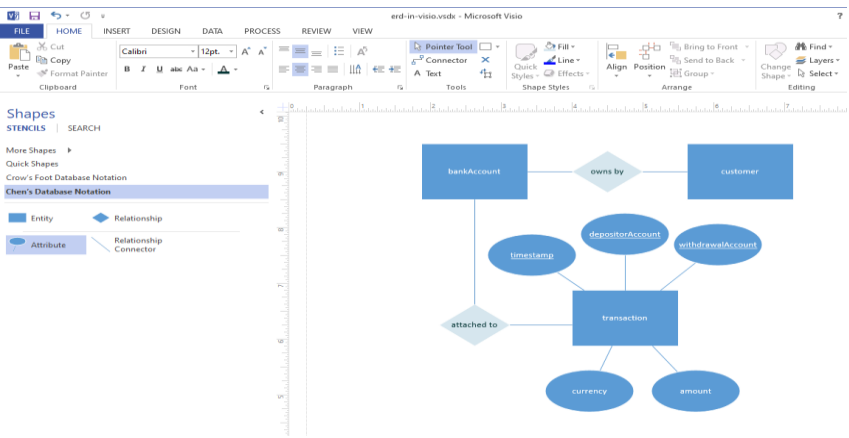
- a. Lucidchart: Alat berbasis web yang mudah digunakan untuk membuat berbagai jenis diagram, termasuk diagram ER.

b. Draw.io: Alat gratis berbasis web dengan berbagai template dan simbol.



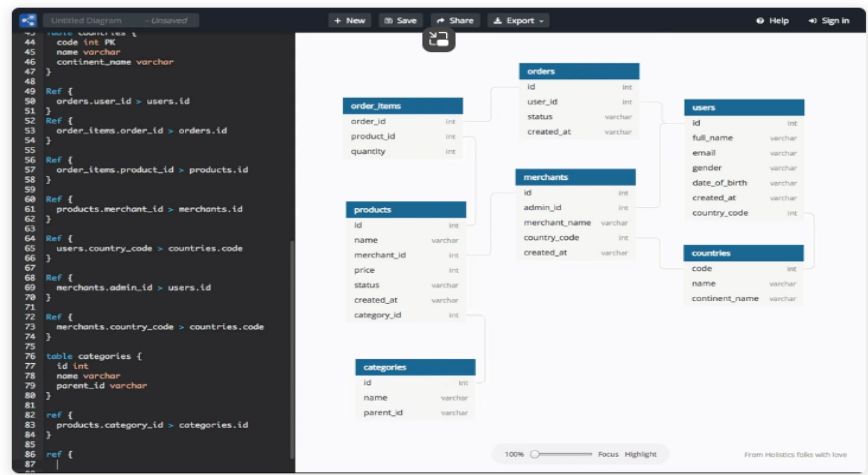
195

c. Microsoft Visio: Alat berbayar yang menyediakan fitur-fitur lengkap untuk pembuatan diagram.



Gambar 7. 17 Tampilan Microsoft Visio

d. Dbdiagram.io: Alat *open-source* untuk membuat diagram ER dan jenis diagram lainnya.



Gambar 7. 18 Tampilan Dbdiagram.io

C. Normalisasi dengan Metode Chen

1. Normalisasi Data

Normalisasi adalah proses pengorganisasian atribut dan entitas dalam basis data untuk mengurangi redundansi dan meningkatkan integritas data. Normalisasi hingga Bentuk Normal Ketiga (3NF): adalah normalisasi data hingga bentuk normal ketiga, di mana semua atribut yang bukan kunci bergantung sepenuhnya pada kunci utama. Pengecualian pada Normalisasi: Pada beberapa kasus, denormalisasi mungkin diperlukan untuk meningkatkan performa akses data, namun ini harus dilakukan dengan hati-hati dan dengan alasan yang kuat.

Tujuan Normalisasi Database adalah:

- a. Menghilangkan dan mengurangi redundansi data.
- b. Memastikan dependensi data (data berada pada tabel yang tepat).

Jika sebuah database belum dinormalisasi, kemungkinan terburuk yang akan merugikan sistem adalah:

- a. *INSERT* Anomali

Situasi dimana tidak memungkinkan untuk memasukkan beberapa jenis data secara langsung di *database*.

b. *UPDATE* Anomali

Situasi dimana nilai yang diubah mengakibatkan ketidakkonsistenan database, artinya data yang diubah tidak sesuai dengan yang diinginkan.

c. *DELETE* Anomali

Penghapusan data tidak sesuai yang diharapkan, artinya data yang seharusnya tidak terhapus malah terhapus.

Hanya tipe “*relational database*” yang bisa dinormalisasi. Untuk melakukan normalisasi database kita harus mengidentifikasi data seperti apa yang akan disimpan. Sebagai contoh saya punya data dari struk penjualan. Saya ambil contoh data struk penjualan karena data ini yang paling umum di sekitar kita.

Tabel 7.1 Tabel Data yang belum di Normalisasi

kode_faktur	tanggal	kode_barang	nama_barang	harga	qty
KD_001	13/07/2020	BRG_001	Indomie Goreng	2500	5
		BRG_002	Indomie Goreng Jumbo	3000	8
		BRG_003	Mie Sedaap Goreng	2500	9
		BRG_004	Mie Sedaap Soto	2300	3
KD_002	13/07/2020	BRG_005	Indomie Goreng	1500	6
		BRG_006	Indomie Soto	1500	6
KD_003	14/07/2020	BRG_007	Pop Mie Ayam	4500	3

1NF : Setiap atribut dari data tersebut hanya memiliki nilai tunggal dalam satu baris. Jadi, tabel yang belum dinormalisasi perlu diubah, sehingga bentuk 1NF menjadi seperti ini:

Tabel 7. 2 Normalisasi Tahap 1

1NF					
kode_faktur	tanggal	kode_barang	nama_barang	harga	qty
KD_001	13/07/2020	BRG_001	Indomie Goreng	2500	5
KD_001	13/07/2020	BRG_002	Indomie Goreng Jumbo	3000	8
KD_001	13/07/2020	BRG_003	Mie Sedaap Goreng	2500	9
KD_001	13/07/2020	BRG_004	Mie Sedaap Soto	2300	3
KD_002	13/07/2020	BRG_005	Intermie Goreng	1500	6
KD_002	13/07/2020	BRG_006	Intermie Soto	1500	6
KD_003	14/07/2020	BRG_007	Pop Mie Ayam	4500	3

Tidak boleh ada *grouping data* ataupun duplikasi data.

2NF : Tidak diperbolehkan adanya partial “*functional dependency*” kepada primary key dalam sebuah tabel. *Functional dependency* adalah setiap atribut yang bukan kunci (*non key*) bergantung secara fungsional terhadap primary key. Normalisasi 2NF pada tabel tersebut harus dipecah berdasarkan *primary key*. Sehingga bentuk normalisasi 2NF dari tabel tersebut adalah sebagai berikut :

Tabel 7. 3 Normalisasi Tahap 2

2NF							
Tabel Barang			Tabel Transaksi				
kode_barang	nama_barang	harga	kode_faktur	tanggal	kode_barang	qty	
BRG_001	Indomie Goreng	2500	KD_001	13/07/2020	BRG_001	5	
BRG_002	Indomie Goreng Jumbo	3000	KD_001	13/07/2020	BRG_002	8	
BRG_003	Mie Sedaap Goreng	2500	KD_001	13/07/2020	BRG_003	9	
BRG_004	Mie Sedaap Soto	2300	KD_001	13/07/2020	BRG_004	3	
BRG_005	Intermie Goreng	1500	KD_002	13/07/2020	BRG_005	6	
BRG_006	Intermie Soto	1500	KD_002	13/07/2020	BRG_006	6	
BRG_007	Pop Mie Ayam	4500	KD_003	14/07/2020	BRG_007	3	

3NF : Tidak diperbolehkan adanya partial “*transitive dependency*” pada sebuah tabel. Biasa terjadi pada tabel hasil relasi atau kondisi dimana terdapat tiga atribut A, B, C. Kondisinya adalah $A \Rightarrow B$ dan $B \Rightarrow C$. Maka C dikatakan sebagai transitive dependency terhadap A melalui B. Jika terdapat suatu atribut yang tidak bergantung pada primary key tapi bergantung pada field yang lain maka atribut-atribut tersebut perlu dipisah ke tabel baru. Terlihat pada atribut qty, kolom tersebut tidak bergantung langsung pada *primary key* kode_faktur melainkan bergantung pada kolom kode_barang. akan menghasilkan tabel berikut:

Tabel 7. 4 Normalisasi Tahap 3

3NF					
Tabel Barang				Tabel Transaksi	
kode_barang	nama_barang	harga		kode_faktur	tanggal
BRG_001	Indomie Goreng	2500		KD_001	13/07/2020
BRG_002	Indomie Goreng Jumbo	3000		KD_002	13/07/2020
BRG_003	Mie Sedaap Goreng	2500		KD_003	14/07/2020
BRG_004	Mie Sedaap Soto	2300			
BRG_005	Intermie Goreng	1500			
BRG_006	Intermie Soto	1500			
BRG_007	Pop Mie Ayam	4500			
Tabel Detail Barang					
kode_faktur	kode_barang	qty	harga		
KD_001	BRG_001	5	2500		
KD_001	BRG_002	8	2500		
KD_001	BRG_003	9	2500		
KD_001	BRG_004	3	2500		
KD_002	BRG_005	6	3000		
KD_002	BRG_006	6	3000		
KD_003	BRG_007	3	2500		

D. Latihan dan Penilaian

Untuk memperkuat pemahaman dan keterampilan dalam membuat diagram *Entity-Relationship* (ER) menggunakan metode

Chen, latihan praktis dan penilaian adalah bagian penting dari proses pembelajaran. Berikut adalah pendekatan untuk merancang latihan dan penilaian yang efektif:

1. Latihan Praktis

Latihan praktis berfokus pada penerapan teori ke dalam praktik nyata. Latihan ini dapat mencakup berbagai skenario dan studi kasus yang dirancang untuk melatih keterampilan dalam mengenali dan menggambarkan entitas, atribut, dan hubungan.

Identifikasi Entitas dan Atribut:

- a. Latihan 1: Berikan deskripsi sistem bisnis sederhana, seperti sistem pemesanan restoran, dan minta peserta untuk mengidentifikasi entitas yang relevan (misalnya, Pelanggan, Pesanan, Menu) dan atribut utama untuk setiap entitas.
- b. Latihan 2: Dengan skenario yang lebih kompleks, seperti sistem manajemen rumah sakit, minta peserta untuk mengidentifikasi entitas yang lebih mendetail (misalnya, Pasien, Dokter, Jadwal) dan atribut terkait.

Penentuan Hubungan dan Kardinalitas:

- a. Latihan 3: Dengan menggunakan contoh sistem e-commerce, minta peserta untuk menentukan hubungan antara entitas seperti Pelanggan, Produk, dan Pesanan. Tentukan juga kardinalitas dari hubungan tersebut (misalnya, satu pelanggan

dapat memiliki banyak pesanan, satu pesanan dapat mencakup banyak produk).

- b. Latihan 4: Minta peserta untuk menggambarkan hubungan yang melibatkan entitas dengan hubungan yang lebih kompleks, seperti hubungan banyak-ke-banyak, dan mengatasi hubungan tersebut dengan entitas penghubung.

Membuat Diagram ER Lengkap:

- a. Latihan 5: Dengan memberikan studi kasus lengkap, seperti sistem perpustakaan, minta peserta untuk membuat diagram ER yang lengkap. Ini termasuk penggambaran semua entitas, atribut, hubungan, dan kardinalitas.
- b. Latihan 6: Berikan skenario perubahan bisnis, seperti penambahan fitur baru dalam sistem yang sudah ada, dan minta peserta untuk memperbarui diagram ER yang sesuai.

2. Penilaian

Penilaian membantu untuk mengukur seberapa baik peserta memahami konsep dan keterampilan yang telah dipelajari. Berikut adalah beberapa metode untuk menilai keterampilan dalam membuat diagram ER:

- a. Penilaian Berbasis Rubrik:

Kembangkan rubrik penilaian yang jelas dan terstruktur, yang mencakup kriteria seperti ketepatan identifikasi entitas,

kelengkapan atribut, kejelasan hubungan, keakuratan kardinalitas, dan presentasi visual. Misalnya, kriteria bisa meliputi: Identifikasi Entitas (0-5 poin): Seberapa lengkap dan akurat entitas diidentifikasi? Kelengkapan Atribut (0-5 poin): Apakah semua atribut yang relevan disertakan?, Hubungan dan Kardinalitas (0-5 poin): Apakah hubungan dan kardinalitas diidentifikasi dengan benar dan jelas?, Presentasi Visual (0-5 poin): Seberapa baik diagram diatur dan disajikan?

b. Ujian Teori dan Praktek:

Adakan ujian tertulis atau *online* yang mencakup pertanyaan teoritis tentang metode Chen dan diagram ER. Sertakan juga bagian praktek, di mana peserta harus membuat diagram ER berdasarkan studi kasus yang diberikan.

c. Proyek Akhir:

Sebagai bagian dari penilaian akhir, minta peserta untuk mengerjakan proyek yang mencakup desain diagram ER untuk sistem bisnis tertentu. Proyek ini bisa berupa tugas individu atau kelompok. Penilaian dapat mencakup komponen presentasi, di mana peserta harus menjelaskan keputusan desain mereka dan bagaimana diagram mereka mencerminkan kebutuhan bisnis.

Studi Kasus dan Skenario Alternatif

Selain latihan rutin, gunakan studi kasus dan skenario alternatif untuk mengeksplorasi berbagai situasi dan tantangan yang mungkin muncul dalam desain basis data.

1) Studi Kasus 1: Sistem Perbankan

Jelaskan bagaimana mengelola informasi tentang nasabah, rekening, transaksi, dan layanan bank. Fokus pada penanganan data sensitif dan kebijakan akses data.

2) Studi Kasus 2: *Platform* Media Sosial

Identifikasi entitas seperti Pengguna, Postingan, Komentar, dan Like. Jelaskan cara menangani hubungan antar pengguna dan interaksi dengan konten.

3) Studi Kasus 3: Sistem Penjualan Tiket *Online*

Diskusikan entitas seperti Acara, Tiket, Pembeli, dan Penjual. Fokus pada manajemen stok tiket dan pemrosesan pembayaran.

E. Kesimpulan

Metode Chen dalam pembuatan diagram ER adalah alat yang sangat berguna untuk merancang basis data yang efisien dan efektif. Dengan menyediakan representasi visual yang jelas, diagram ER membantu dalam perencanaan dan implementasi basis data yang terstruktur dengan baik, meminimalkan kesalahan

desain, dan memastikan bahwa semua aspek penting dari data dan hubungannya diidentifikasi dan diatur dengan benar.

Memahami dan menerapkan metode Chen dengan benar memungkinkan pengembang dan desainer basis data untuk merancang sistem yang memenuhi kebutuhan bisnis dengan lebih baik, sambil memastikan integritas dan keakuratan data. Melalui latihan praktis dan penggunaan alat yang tepat, individu dapat mengembangkan keterampilan yang diperlukan untuk mengelola proyek desain basis data dengan sukses.

Dalam membangun basis data yang efisien dan efektif, diagram *Entity-Relationship* (ER) memainkan peran penting sebagai alat desain yang membantu dalam visualisasi struktur data dan hubungan antar entitas. Metode Chen, salah satu pendekatan untuk menggambar diagram ER, menawarkan notasi yang jelas dan sederhana untuk menggambarkan entitas, atribut, dan hubungan mereka.

DAFTAR PUSTAKA

- Brealey, R.A., Myers, S.C. & Allen, F. (2022) *Principles of Corporate Finance*. Edisi terbaru. McGraw-Hill Education.
- Brigham, E.F. & Houston, J.F. (2021) *Fundamentals of Financial Management*. Edisi terbaru. Cengage Learning.
- Chang, W.J. (2021) 'Experiential marketing, brand image and brand loyalty: a case study of Starbucks', *British Food Journal*, 123(1), pp. 209–223.
<https://doi.org/10.1108/BFJ-01-2020-0014>.
- Titman, S., Keown, A.J. & Martin, J.D. (2021) *Financial Management: Principles and Applications*. Edisi terbaru. Pearson.

BAB 8

DATA DEFINITION LANGUAGE (DDL) DAN DATA MANIPULATION LANGUAGE (DML)

Oleh : Sutiyono, S.T., M.Kom

A. Pengantar DDL dan DML

1. Pentingnya Pemahaman DDL dan DML dalam Sistem Basis Data

Sistem basis data telah menjadi fondasi dari hampir setiap aplikasi teknologi informasi modern. Dalam dunia yang semakin terhubung, pengelolaan data yang efisien, terstruktur, dan dapat diakses dengan mudah menjadi kebutuhan mendesak bagi organisasi, bisnis, dan individu. Di tengah kompleksitas pengelolaan data ini, pemahaman tentang *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML) memainkan peran yang sangat penting.

2. DDL dan DML: Pilar Pengelolaan Data

DDL dan DML adalah dua komponen utama dari *Structured Query Language* (SQL) yang digunakan untuk berinteraksi dengan basis data. Meskipun memiliki tujuan yang berbeda, keduanya saling melengkapi dalam memastikan sistem basis data dapat memenuhi kebutuhan pengguna. DDL digunakan untuk mendefinisikan struktur basis data, sementara DML memungkinkan manipulasi data yang ada di dalamnya. Tanpa kedua elemen ini, sistem basis data akan kehilangan fungsionalitas esensialnya.

3. Penerapan di Dunia Nyata

Dalam kehidupan sehari-hari, DDL dan DML digunakan dalam berbagai konteks, mulai dari pembuatan sistem manajemen inventaris hingga aplikasi perbankan *online*. Misalnya, dalam sistem *e-commerce*, DDL diperlukan untuk mendefinisikan tabel produk, pelanggan, dan pesanan, sementara DML digunakan untuk memperbarui status pesanan atau menghapus produk yang sudah tidak tersedia. Pemahaman mendalam tentang kedua aspek ini memungkinkan pengembang untuk membangun solusi yang andal dan *scalable*.

4. Manfaat Belajar DDL dan DML

Dengan menguasai DDL, pengguna dapat memastikan struktur basis data sesuai dengan kebutuhan bisnis. Sementara itu, keahlian dalam DML membantu memastikan bahwa data dapat dimanipulasi secara akurat, menjaga konsistensi dan keandalan sistem. Bersama-sama, keduanya membantu menciptakan basis data yang kuat, efisien, dan mendukung pengambilan keputusan yang lebih baik.

B. Konsep Dasar DDL (*Data Definition Language*)

1. Pengertian DDL

Data Definition Language (DDL) adalah bagian dari *Structured Query Language* (SQL) yang digunakan untuk mendefinisikan dan mengelola struktur data dalam basis data. DDL memungkinkan pengguna untuk membuat, mengubah, dan menghapus objek-objek penting dalam basis data, seperti tabel, indeks, dan skema. Pemahaman mendalam tentang DDL penting bagi pengembang dan administrator basis data karena merupakan langkah awal dalam membangun sistem basis data yang andal.

2. Fungsi dan Kegunaan DDL

DDL memiliki fungsi utama untuk mendefinisikan struktur basis data dan memastikan integritas data. Berikut adalah kegunaan utama DDL:

a. Membuat Struktur Data

Perintah DDL seperti *CREATE* digunakan untuk mendefinisikan struktur tabel, tipe data, dan hubungan antar tabel. Misalnya, dalam sebuah aplikasi penggajian, tabel "karyawan" dapat dibuat dengan kolom-kolom seperti nama, jabatan, dan gaji.

b. Memodifikasi Struktur Tabel

ALTER memungkinkan pengguna untuk melakukan perubahan pada struktur tabel atau objek lainnya tanpa kehilangan data. Ini penting dalam situasi di mana kebutuhan bisnis berubah, seperti menambahkan kolom baru untuk menyimpan informasi tambahan.

c. Menghapus Objek yang Tidak Dibutuhkan

DROP digunakan untuk menghapus objek yang sudah tidak relevan, memastikan basis data tetap bersih dan efisien.

d. Menjamin Integritas Data

Dengan menambahkan *constraint* seperti *primary key*, *foreign key*, dan *unique constraint*, DDL memastikan bahwa data yang dimasukkan ke dalam tabel memenuhi aturan yang telah ditetapkan.

3. Perintah DDL

Perintah-perintah DDL dirancang untuk bekerja di level struktural, memastikan bahwa basis data memiliki kerangka kerja yang

mendukung pengelolaan data secara efektif. Berikut adalah yang termasuk perintah-perintah DDL:

- a. *CREATE*: Perintah ini digunakan untuk membuat tabel, skema, atau objek lain dalam basis data. Dengan *CREATE*, pengguna dapat menentukan nama tabel, kolom, tipe data, serta *constraint* yang dibutuhkan.
- b. *ALTER*: Perintah *ALTER* digunakan untuk mengubah struktur objek yang sudah ada. Misalnya, menambahkan kolom baru ke dalam tabel, mengubah tipe data kolom, atau menambahkan *constraint*.
- c. *DROP*: Perintah *DROP* digunakan untuk menghapus objek, seperti tabel atau indeks, dari basis data. Penggunaan *DROP* harus dilakukan dengan hati-hati karena penghapusan bersifat permanen.

4. Contoh Implementasi DDL

- a. Pembuatan *Database* Sederhana

Sebagai contoh, membuat database sederhana untuk mengelola data karyawan di sebuah perusahaan.

```
-- Membuat database
CREATE DATABASE Perusahaan;

-- Menggunakan database
USE Perusahaan;

-- Membuat tabel karyawan
CREATE TABLE Karyawan (
    ID_Karyawan INT PRIMARY KEY,
    Nama VARCHAR(100) NOT NULL,
    Jabatan VARCHAR(50),
    Gaji DECIMAL(10, 2)
);
```

b. Skrip SQL ALTER

Berikut adalah beberapa contoh variasi perintah Alter.

- a. Menambahkan kolom baru "Email" ke tabel "Karyawan":
`ALTER TABLE Karyawan ADD Email VARCHAR(100);`
- b. Merubah tipe data kolom Jabatan dari "Varchar" menjadi "Char":
`ALTER TABLE Karyawan MODIFY COLUMN Jabatan CHAR(50);`
- c. Mengganti nama kolom "Gaji" menjadi "Honor":
`ALTER TABLE Karyawan CHANGE COLUMN Gaji Honor DECIMAL(10,2);`

c. Contoh Skrip SQL DROP

Berikut adalah beberapa contoh variasi perintah Drop.

- a. Menghapus Database "Perusahaan":
`DROP Perusahaan;`
- b. Menghapus tabel "Karyawan":
`DROP TABLE Karyawan;`
- c. Menghapus kolom "Gaji" pada tabel "Karyawan":
`ALTER TABLE Karyawan DROP COLUMN Gaji;`

Dengan memahami perintah *CREATE*, *ALTER*, dan *DROP*, pengguna dapat mengelola struktur basis data secara efisien dan sesuai dengan kebutuhan.

5. *Constraint*

Dalam pengelolaan basis data relasional, *constraint* (batasan) memainkan peran yang sangat penting. *Constraint* digunakan untuk memastikan integritas dan validitas data dalam tabel, serta untuk menjaga konsistensi antar tabel dalam basis data. Tanpa *constraint* yang tepat, data yang tidak valid atau tidak konsisten dapat masuk ke dalam sistem, yang dapat menyebabkan kesalahan dalam pengolahan data dan pengambilan keputusan.

Constraint adalah aturan yang diterapkan pada kolom atau tabel dalam basis data untuk memastikan bahwa data yang dimasukkan memenuhi kriteria tertentu. *Constraint* digunakan untuk:

- a. Memastikan bahwa data valid.
- b. Menjaga hubungan antar data dalam tabel.
- c. Mencegah kesalahan atau data yang tidak sesuai dengan aturan bisnis.

Secara umum, *constraint* dapat diterapkan pada dua level:

- a. Kolom: Misalnya, untuk memastikan nilai yang dimasukkan dalam kolom tertentu mengikuti aturan tertentu (misalnya, tidak boleh kosong, atau harus unik).
- b. Tabel: Misalnya, untuk memastikan integritas referensial antar tabel (misalnya, *foreign key*).

Jenis-Jenis Constraint dalam Basis Data

1) *Primary Key Constraint*

Primary Key (PK) adalah sebuah constraint yang digunakan untuk mengidentifikasi setiap baris dalam sebuah tabel secara unik. Setiap tabel dalam basis data relasional harus memiliki primary key.

- a. Fungsi: Memastikan bahwa setiap entri dalam tabel adalah unik dan tidak boleh ada dua baris yang memiliki nilai primary key yang sama.
- b. Aturan: Kolom yang dipilih sebagai primary key tidak boleh bernilai NULL, Setiap nilai dalam kolom primary key harus unik. Contoh:

```
CREATE TABLE Mahasiswa (  
    id_mahasiswa INT PRIMARY KEY,  
    nama VARCHAR(100),  
    umur INT  
);
```

2) *Foreign Key Constraint*

Foreign Key (FK) adalah sebuah *constraint* yang digunakan untuk menghubungkan satu tabel dengan tabel lainnya. *Foreign key* memastikan bahwa nilai di kolom tertentu di tabel anak (*child table*) hanya bisa ada jika nilai yang sesuai ada di kolom *primary key* tabel induk (*parent table*).

- a. Fungsi: Menjaga integritas referensial antar tabel.
- b. Aturan: Nilai foreign key harus ada di tabel referensi (tabel induk) atau bisa NULL.

Contoh:

```
CREATE TABLE Transaksi (  
    id_transaksi INT PRIMARY KEY,  
    id_mahasiswa INT,  
    jumlah INT,  
    FOREIGN KEY (id_mahasiswa) REFERENCES  
    Mahasiswa(id_mahasiswa)  
);
```

3) *Unique Constraint*

Unique Constraint digunakan untuk memastikan bahwa semua nilai dalam kolom yang diatur dengan *constraint* ini adalah unik, yaitu tidak ada dua baris yang memiliki nilai yang sama pada kolom tersebut.

- a. Fungsi: Memastikan bahwa tidak ada duplikasi data dalam kolom yang diatur dengan *unique constraint*.
- b. Aturan: Tidak ada dua nilai yang identik dalam kolom yang dikenakan *unique constraint*. Kolom ini bisa berisi nilai NULL, tetapi hanya satu nilai NULL yang diizinkan.

Contoh:

```
CREATE TABLE Produk (  
    id_produk INT PRIMARY KEY,  
    kode_produk VARCHAR(50) UNIQUE,  
    nama_produk VARCHAR(100)  
);
```

4) *Check Constraint*

Check Constraint digunakan untuk memastikan bahwa data yang dimasukkan ke dalam kolom memenuhi kondisi atau aturan tertentu yang ditentukan oleh pengguna.

- a. Fungsi: Memastikan bahwa data yang dimasukkan ke dalam kolom memenuhi kondisi yang telah ditentukan, misalnya nilai dalam kolom umur harus lebih besar dari 0.
- b. Aturan: *Check constraint* dapat diterapkan untuk memastikan nilai dalam kolom berada dalam rentang yang diinginkan atau mengikuti kondisi tertentu.

Contoh:

```
CREATE TABLE Karyawan (  
    id_karyawan INT PRIMARY KEY,  
    nama VARCHAR(100),  
    umur INT CHECK (umur >= 18)  
);
```

5) *Not Null Constraint*

Not Null Constraint digunakan untuk memastikan bahwa kolom dalam tabel tidak boleh berisi nilai NULL. Ini berarti bahwa setiap entri dalam kolom tersebut harus memiliki nilai.

- a. Fungsi: Menjamin bahwa kolom tidak akan memiliki nilai NULL.
- b. Aturan: Kolom yang memiliki not null *constraint* harus selalu diisi dengan data yang valid.

Contoh:

```
CREATE TABLE Pelanggan (  
    id_pelanggan INT PRIMARY KEY,  
    nama_pelanggan VARCHAR(100) NOT NULL,  
    alamat VARCHAR(255)  
);
```

6) *Default Constraint*

Default Constraint digunakan untuk memberikan nilai *default* pada kolom jika pengguna tidak memberikan nilai pada saat memasukkan data.

- a. Fungsi: Menetapkan nilai default untuk kolom jika tidak ada nilai yang dimasukkan.
- b. Aturan: Nilai default hanya berlaku ketika kolom tersebut tidak diisi secara eksplisit.

Contoh:

```
CREATE TABLE Produk (  
    id_produk INT PRIMARY KEY,  
    nama_produk VARCHAR(100),  
    stok INT DEFAULT 0  
);
```

Integritas Data dan Penggunaan Constraint

a. Integritas Entitas

Integritas entitas berkaitan dengan memastikan bahwa setiap baris dalam tabel dapat diidentifikasi secara unik. Primary key adalah cara utama untuk menjaga integritas entitas, karena setiap baris dalam tabel harus memiliki nilai unik pada primary key-nya.

b. Integritas Referensial

Integritas referensial memastikan bahwa hubungan antar tabel dijaga. *Foreign key* adalah mekanisme utama untuk menjaga integritas referensial antara dua tabel. Dengan menggunakan *foreign key*, kita dapat memastikan bahwa setiap entri di tabel anak memiliki hubungan yang valid dengan tabel induk.

c. Integritas Domain

Integritas domain berkaitan dengan memastikan bahwa nilai dalam kolom tabel sesuai dengan tipe data yang ditentukan dan

memenuhi aturan tertentu. *Constraint* seperti *Check* dan *Not Null* digunakan untuk memastikan bahwa nilai yang dimasukkan sesuai dengan domain yang diizinkan.

Penggunaan Constraint dalam Desain Database

1) Menyusun Struktur Tabel

Dalam desain basis data, penting untuk merancang tabel dan menentukan *constraint* yang diperlukan untuk menjaga kualitas data. Sebagai contoh, jika kita merancang tabel untuk menyimpan informasi karyawan, kita akan menggunakan primary key untuk identifikasi unik, dan foreign key untuk menghubungkan tabel karyawan dengan tabel departemen.

2) Menangani Pelanggaran Constraint

Kadang-kadang, pelanggaran terhadap *constraint* dapat terjadi, misalnya ketika kita mencoba memasukkan nilai yang melanggar aturan unique atau foreign key. Dalam kasus ini, RDBMS akan menghasilkan error dan menolak transaksi tersebut. Penting untuk menangani error ini dengan baik agar aplikasi tetap berjalan dengan lancar.

3) Menguji dan Memvalidasi Data

Constraint tidak hanya digunakan saat memasukkan data, tetapi juga sangat berguna dalam menguji dan memvalidasi data. Dengan menggunakan *constraint*, kita dapat memastikan

bahwa data yang dimasukkan ke dalam basis data sudah memenuhi aturan yang ditentukan, sehingga mengurangi kemungkinan kesalahan di kemudian hari.

Contoh Constraint Dalam Query

Berikut adalah beberapa queri yang menunjukkan penerapan *constraint* pada struktur tabelnya.

a. Membuat tabel Jabatan

```
CREATE TABLE Jabatan (  
    ID_Jabatan INT PRIMARY KEY AUTO_INCREMENT,  
        -- Primary Key Constraint  
    Nama_Jabatan VARCHAR(100) UNIQUE NOT NULL  
        -- Unique dan Not Null Constraint  
);
```

b. Membuat tabel Karyawan

```
CREATE TABLE Karyawan (  
    ID_Karyawan INT PRIMARY KEY AUTO_INCREMENT,  
        -- Primary Key Constraint  
    Nama VARCHAR (100) NOT NULL,  
        -- Not Null Constraint  
    ID_Jabatan INT, Gaji DECIMAL (10, 2) CHECK (Gaji >= 0),  
        -- Check Constraint  
    Email VARCHAR (100) UNIQUE,  
        -- Unique Constraint
```

```
Tanggal_Lahir    DATE,    FOREIGN    KEY    (ID_Jabatan)
REFERENCES Jabatan (ID_Jabatan)
    -- Foreign Key Constraint
);
```

C. Konsep Dasar DML (*Data Manipulation Language*)

1. Pengertian DML

Data Manipulation Language (DML) adalah bagian dari *Structured Query Language* (SQL) yang digunakan untuk mengelola data dalam tabel basis data. DML memungkinkan pengguna untuk melakukan manipulasi data, seperti menambah, memperbarui, atau menghapus data. Berbeda dengan DDL yang berfokus pada struktur basis data, DML bekerja langsung pada konten atau isi tabel.

2. Fungsi dan Kegunaan DML

DML memiliki beberapa fungsi utama yang menjadikannya bagian penting dalam pengelolaan basis data:

a. Menambahkan Data Baru

Dengan menggunakan perintah *INSERT*, pengguna dapat menambahkan data baru ke tabel yang telah didefinisikan sebelumnya.

b. Memperbarui Data yang Sudah Ada

Perintah *UPDATE* memungkinkan pengguna untuk mengubah data yang telah ada dalam tabel, sesuai dengan kebutuhan atau perubahan kondisi.

c. Menghapus Data yang Tidak Relevan

Perintah *DELETE* digunakan untuk menghapus data tertentu dari tabel, menjaga agar data yang tersimpan tetap relevan dan terkini.

d. Mendukung Proses Bisnis

Dengan kemampuan manipulasi data, DML memungkinkan pengembang untuk mendukung berbagai proses bisnis yang dinamis, seperti pengelolaan inventaris, transaksi pelanggan, dan analisis data.

3. Perintah DML

Perintah-perintah dalam query yang termasuk kepada kelompok perintah DML adalah sebagai berikut:

a. *INSERT*: Digunakan untuk menambahkan data baru ke dalam tabel. Perintah ini memungkinkan pengguna untuk menyisipkan satu atau beberapa baris data dalam satu eksekusi.

b. *UPDATE*: Digunakan untuk memperbarui data yang telah ada dalam tabel. Perintah ini memungkinkan perubahan spesifik pada satu atau lebih kolom berdasarkan kondisi tertentu.

- c. *DELETE*: Digunakan untuk menghapus data dari tabel. Perintah ini sering digunakan ketika data sudah tidak relevan atau diperlukan.

4. Contoh Penggunaan Perintah DML

Sebagaimana kita ketahui, pada sub bahasan sebelumnya kita memiliki sebuah database “Perusahaan” dengan dua tabel, diantaranya tabel “Karyawan” dan tabel “Jabatan”. Setelah kita memiliki struktur basis data beserta tabelnya, langkah selanjutnya akan melibahkan kelompok perintah DML. Berikut adalah contoh-contoh penggunaan perintah DML:

- a. Menambahkan data pada tabel

-- Menambahkan data ke tabel Jabatan

```
INSERT INTO Jabatan (Nama_Jabatan) VALUES  
( 'Manager'), ( 'Staff'), ( 'Intern');
```

-- Menambahkan data ke tabel Karyawan

```
INSERT INTO Karyawan  
(Nama, ID_Jabatan, Gaji, Email, Tanggal_Lahir)  
VALUES  
( 'Andi', 1, 15000000.00, 'andi@company.com', '1985-05-  
15'),  
( 'Budi', 2, 8000000.00, 'budi@company.com', '1990-08-  
22'),
```

```
('Citra', 3, 4000000.00, 'citra@company.com', '1995-11-10');
```

b. Memperbarui Data Pegawai

-- Mengubah gaji untuk karyawan dengan

```
UPDATE Karyawan SET Gaji = 15000000 WHERE  
ID_Karyawan = 1;
```

SET: Menentukan kolom yang akan diubah dan nilai barunya.

WHERE: Memberikan kriteria untuk memilih data yang akan diperbarui.

c. Menghapus Data Pegawai

-- Menghapus karyawan dengan ID_Karyawan = 1

```
DELETE FROM Karyawan WHERE ID_Karyawan = 1;
```

DELETE FROM: Perintah untuk menghapus data dari tabel.

WHERE: Kriteria untuk memilih data yang akan dihapus.

Hati-hati: Jika WHERE tidak disertakan, semua data dalam tabel akan dihapus.

Pada bab ini memberikan panduan langkah demi langkah dalam menerapkan konsep DDL dan DML pada skenario dunia nyata. Pendekatan ini diharapkan dapat membantu pembaca memahami bagaimana teori SQL diterapkan dalam praktik untuk membangun sistem basis data yang efisien dan fungsional.

5. Kesalahan Umum dan Tips Optimasi

Kesalahan Umum dalam Penggunaan DDL dan DML: Pengguna sering melakukan kesalahan saat menggunakan DDL dan DML yang dapat menyebabkan masalah pada sistem basis data. Beberapa kesalahan umum meliputi:

a. Tidak Menentukan Constraint dengan Jelas

Misalnya, lupa menambahkan primary key pada tabel, sehingga data tidak dapat diidentifikasi secara unik.

b. Menghapus Data tanpa Kondisi yang Tepat

Penggunaan perintah *DELETE* tanpa *WHERE* dapat menghapus seluruh data dalam table.

Untuk menghindari kesalahan umum dan meningkatkan kualitas basis data, berikut beberapa praktik terbaik yang dapat diterapkan dalam penulisan SQL:

a. Gunakan Nama Kolom yang Jelas dan Deskriptif

Hindari penggunaan nama kolom yang ambigu atau terlalu umum, seperti *Data1* atau *FieldX*. Sebaliknya, gunakan nama yang menjelaskan fungsi atau isi kolom, seperti *Nama_Pelanggan* atau *Tanggal_Transaksi*.

b. Tambahkan Constraint Secara Konsisten

Selalu definisikan *primary key*, *foreign key*, dan *constraint* lainnya sesuai kebutuhan. *Constraint* ini tidak hanya menjaga

integritas data tetapi juga membantu mencegah input data yang tidak valid.

c. Gunakan Indeks dengan Bijak

Tambahkan indeks pada kolom yang sering digunakan untuk pencarian atau pengurutan. Namun, hindari penggunaan indeks yang berlebihan karena dapat memperlambat operasi DML seperti *INSERT*, *UPDATE*, dan *DELETE*.

d. Cek dan Uji Perintah Sebelum Eksekusi

Sebelum menjalankan perintah DDL atau DML, terutama perintah yang bersifat destruktif seperti *DROP* atau *DELETE*, pastikan untuk memeriksa kembali query agar tidak menyebabkan kerugian data.

e. Gunakan Transaksi untuk Operasi yang Kompleks

Pada operasi yang melibatkan banyak perintah DML, gunakan mekanisme transaksi untuk memastikan bahwa semua perintah berhasil dijalankan sebelum data di-commit ke basis data.

f. Dokumentasikan Perubahan yang Dilakukan

Catat setiap perintah DDL atau perubahan signifikan pada basis data. Dokumentasi ini akan sangat membantu dalam pelacakan dan pemeliharaan di masa depan.

Tips Optimasi *Query* untuk Manipulasi Data: *Query* yang tidak dioptimalkan dapat memengaruhi kinerja sistem basis data secara

signifikan. Berikut beberapa tips untuk mengoptimalkan query DML:

a. Pilih Kolom yang Spesifik

Hindari menggunakan *SELECT* dalam query *SELECT*, kecuali benar-benar diperlukan. Sebaiknya, hanya pilih kolom yang dibutuhkan untuk mengurangi *overhead* data.

b. Gunakan Kondisi Filter yang Efisien

Saat menggunakan perintah seperti *UPDATE* atau *DELETE*, tambahkan klausa *WHERE* yang spesifik untuk membatasi cakupan perubahan data. Ini akan mengurangi risiko memodifikasi atau menghapus data yang tidak diinginkan.

c. Buat Indeks pada Kolom yang Sering Dicari

Indeks dapat meningkatkan kecepatan pencarian data, terutama pada tabel dengan jumlah baris yang besar. Namun, hindari membuat indeks pada kolom yang jarang digunakan untuk pencarian.

d. Pantau dan Analisis Kinerja Query

Gunakan alat analisis query seperti *EXPLAIN* (untuk MySQL) atau *EXPLAIN ANALYZE* (untuk PostgreSQL) untuk memahami bagaimana query dijalankan dan menemukan area yang perlu dioptimalkan.

e. Pertimbangkan Partisi Tabel untuk Data yang Besar

Pada tabel dengan data yang sangat besar, gunakan partisi untuk membagi tabel menjadi beberapa bagian yang lebih kecil. Ini akan membantu mempercepat akses data pada subset tertentu.

D. Penutup

Pada bab ini, kita telah membahas mengenai dua kategori utama perintah SQL, yaitu *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML). DDL digunakan untuk mendefinisikan struktur database seperti pembuatan tabel, penambahan kolom, serta penghapusan tabel. Di sisi lain, DML digunakan untuk mengelola data yang ada di dalam tabel, termasuk memasukkan data (*INSERT*), memperbarui data (*UPDATE*), dan menghapus data (*DELETE*).

Melalui latihan dan studi kasus, kita telah mempraktikkan penggunaan perintah-perintah tersebut dalam berbagai situasi yang umum dijumpai dalam pengelolaan *database*. Memahami DDL dan DML adalah dasar yang sangat penting bagi siapa saja yang bekerja dengan *database*.

Relevansi dan Aplikasi Praktis DDL dan DML : Perintah DDL dan DML memiliki peran yang sangat penting dalam pengelolaan database sehari-hari. DDL membantu dalam merancang dan

memodifikasi struktur *database* sesuai dengan kebutuhan aplikasi atau sistem, sementara DML memungkinkan pengguna untuk memanipulasi data sesuai dengan perubahan yang terjadi dalam dunia nyata.

Aplikasi praktis dari perintah-perintah ini sangat luas, termasuk dalam pembuatan aplikasi sistem informasi, aplikasi *e-commerce*, manajemen inventaris, aplikasi keuangan, dan banyak lagi. Pemahaman yang mendalam tentang DDL dan DML akan mempermudah dalam membangun dan mengelola sistem yang berbasis database secara efisien dan efektif.

DAFTAR PUSTAKA

- Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Pearson Education.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.
- Kroenke, D. M., & Auer, D. (2013). *Database Processing: Fundamentals, Design, and Implementation* (13th ed.). Pearson.
- Kurniawan, B. (2020). *Belajar SQL untuk Pemula*. Jakarta: PT Elex Media Komputindo.
- Nugroho, H. (2021). *Pengantar Sistem Basis Data*. Malang: UB Press.
- Rob, P., & Coronel, C. (2020). *Database Systems: Design, Implementation, & Management* (13th ed.). Cengage Learning.
- Siregar, A. F. (2019). *Database Management Systems*. Yogyakarta: Andi Publisher.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill.

BAB 9

DATA QUERY LANGUAGE (DQL) DAN DATA CONTROL LANGUAGE (DCL)

Oleh : Widya Eka Pranata, S.Si., M.Si

A. Data Query Language (DQL)

Data Query Language (DQL) adalah salah satu subset dari *Structured Query Language* (SQL) yang dirancang untuk mengambil data dari basis data. DQL memungkinkan pengguna untuk memanipulasi dan menampilkan data tanpa mengubah struktur atau isi basis data itu sendiri. Fokus utama DQL adalah mengambil informasi untuk mendukung analisis, pelaporan, atau aplikasi lainnya.

Menurut Connolly dan Begg (2015), DQL merupakan elemen utama dalam interaksi antara pengguna dan basis data. Perintah utama DQL adalah SELECT, yang digunakan untuk mengekstraksi data dengan fleksibilitas yang tinggi berdasarkan kebutuhan spesifik pengguna. DQL memungkinkan pengguna untuk mengakses data dengan cara yang efisien dan fleksibel, yang

merupakan fondasi dalam pengelolaan informasi di era modern. DQL memainkan peran utama dalam analitik, pelaporan, dan pengambilan keputusan berbasis data.

Fungsi Utama DQL :

- a. Mengambil Data dari Basis Data
- b. DQL membantu mengambil informasi yang relevan dari tabel atau kombinasi tabel dalam basis data.

- c. Memfilter Data

Dengan menggunakan perintah seperti *WHERE*, DQL memungkinkan pengguna untuk mengambil data berdasarkan kondisi tertentu.

- d. Mengatur Tampilan Data

DQL memungkinkan pengguna untuk mengelompokkan data (*GROUP BY*), mengurutkan data (*ORDER BY*), atau membatasi jumlah data yang ditampilkan (*LIMIT*).

- e. Menyediakan Informasi untuk Keputusan

Hasil dari perintah DQL sering digunakan untuk mendukung pengambilan keputusan berbasis data dalam berbagai aplikasi.

- f. Mendukung Analisis Data

Dalam konteks analitik, DQL memungkinkan penggalian data secara mendalam untuk mendapatkan wawasan penting.

1. Statement SELECT

Statement *SELECT* adalah inti dari DQL. Perintah ini memungkinkan pengguna untuk menentukan kolom, tabel, kondisi, dan bahkan cara data diurutkan atau dikelompokkan

Statement SELECT
<pre>SELECT column1, column2 FROM table_name WHERE condition;</pre>

Contoh : Perintah *SELECT* digunakan untuk mengambil data dari satu atau beberapa tabel. Mengambil semua kolom, Menampilkan seluruh data dari tabel *employees*

Statement SELECT
<pre>SELECT * FROM employees;</pre>

Mengambil kolom tertentu

Statement SELECT
<pre>SELECT name, salary FROM employees;</pre>

Menampilkan hanya kolom *name* dan *salary* dari tabel *employees*
Select dapat digunakan untuk operasi sederhana hingga kompleks, seperti penggabungan tabel (*joins*) atau *subqueries*

2. Filter dan Kondisi

Filter dan kondisi digunakan untuk membatasi data yang diambil sesuai dengan kebutuhan. Misalnya:

- a. *WHERE* membatasi data berdasarkan kriteria tertentu.
- b. *LIKE* digunakan untuk pencarian pola, seperti menemukan nama yang dimulai dengan huruf tertentu.
- c. *IN* dan *BETWEEN* mempermudah pencarian data dalam daftar atau rentang tertentu.

Contoh:

Statement <i>WHERE</i> , <i>LIKE</i> , <i>IN</i> , <i>BETWEEN</i>
<pre>SELECT * FROM employees WHERE salary > 5000 AND department IN ('HR', 'IT');</pre>

3. Fungsi Agregasi

Fungsi agregasi sering digunakan untuk menghitung ringkasan data, seperti jumlah, rata-rata, maksimum, atau total. Elmasri dan Navathe (2016) menyoroti fungsi-fungsi ini sebagai alat penting dalam analisis data berbasis SQL.

- a. *COUNT* menghitung jumlah baris.
- b. *SUM* menghitung total nilai.
- c. *AVG* menghitung rata-rata nilai.
- d. *Max* /*min* menemukan nilai terbesar /terkecil dalam kolom

4. Pengelompokan dan Pengurutan

- a. GROUP BY: Mengelompokkan data berdasarkan kolom tertentu untuk analisis.

Contoh:

Statement GROUP BY
<pre>SELECT department, COUNT(*) FROM employees GROUP BY department;</pre>

- b. ORDER BY : Mengurutkan hasil query berdasarkan kolom tertentu, baik secara *ascending* (ASC) atau *descending* (DESC)

Contoh:

Statement ORDER BY
<pre>SELECT name, salary FROM employees ORDER BY salary DESC;</pre>

5. Penggabungan Tabel (Joins)

Dalam sistem basis data relasional, penggabungan tabel menjadi sangat penting. Joins memungkinkan pengambilan data dari dua atau lebih tabel berdasarkan relasi di antara tabel tersebut.

- a. INNER JOIN: Mengembalikan data yang cocok di kedua tabel.

- b. LEFT JOIN: Mengembalikan semua data dari tabel kiri dan data yang cocok dari tabel kanan.
- c. RIGHT JOIN: Sebaliknya dari LEFT JOIN.

6. Subqueries (Kueri Bersarang)

Subquery adalah query yang digunakan di dalam query lain. *Subquery* biasanya digunakan untuk membandingkan nilai dengan hasil dari query lain

Contoh:

Statement
<pre>SELECT name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);</pre>

7. Union dan Operasi Set

Menggabungkan hasil dari beberapa query dengan *UNION*, *INTERSECT*, atau *EXCEPT*, *UNION* menggabungkan hasil tanpa duplikasi, contoh :

Statement UNION, INTERSECT, EXCEPT
<pre>SELECT city FROM customers UNION SELECT city FROM suppliers;</pre>

Cara Kerja DQL :

1. Proses Query

- 1) Pengguna mengirimkan perintah SELECT melalui antarmuka SQL.
- 2) DBMS menerima perintah dan memecahnya menjadi langkah-langkah yang lebih kecil untuk diproses.
- 3) *Query optimizer* dalam DBMS menentukan cara paling efisien untuk mengeksekusi query.

2. Eksekusi Query

DBMS membaca data dari tabel atau kombinasi tabel berdasarkan logika query, memfilter data, dan memformat hasil sesuai dengan instruksi.

3. Pengembalian Hasil

Setelah query dijalankan, DBMS mengembalikan hasil dalam bentuk tabel atau data terstruktur lainnya.

Pentingnya DQL dalam Sistem Manajemen Basis Data

a. Mempermudah Akses Data

Dengan DQL, pengguna tidak perlu memahami detail teknis internal basis data untuk mengambil informasi.

b. Mendukung Analisis Data

DQL adalah fondasi dari banyak alat analitik modern yang

digunakan dalam data science, business intelligence, dan visualisasi data.

c. Meningkatkan Produktivitas

Penggunaan perintah DQL memungkinkan pengambilan informasi yang cepat dan efisien tanpa harus memodifikasi struktur data.

d. Mengintegrasikan Sistem

Dalam aplikasi modern, DQL sering digunakan untuk menarik data yang akan digunakan dalam API, dasbor, atau aplikasi lain.

Aplikasi DQL dalam Dunia Nyata

a. Industri Perbankan

DQL digunakan untuk menghasilkan laporan transaksi nasabah berdasarkan parameter seperti waktu, jenis transaksi, atau lokasi.

b. *E-commerce*

Platform belanja online menggunakan DQL untuk menampilkan produk berdasarkan filter yang diberikan pengguna, seperti kategori, harga, atau popularitas.

c. Sistem Informasi Akademik

DQL membantu mengambil data seperti nilai mahasiswa, daftar hadir, atau daftar mata kuliah yang sedang berlangsung.

d. Big Data dan Analitik

Dalam pengelolaan big data, DQL sering menjadi langkah awal dalam pipeline analitik untuk mengakses subset data yang relevan.

Data Query Language (DQL) adalah alat yang sangat penting dalam dunia basis data, memungkinkan pengguna untuk mengambil, memfilter, dan menganalisis data dengan mudah. Dengan dukungan fitur seperti *SELECT*, *WHERE*, *GROUP BY*, dan lainnya, DQL memainkan peran krusial dalam memastikan data yang relevan dapat diakses dan digunakan secara efektif.

B. *Data Control Language* (DCL)

Data Control Language (DCL) adalah salah satu komponen dari *Structured Query Language* (SQL) yang berfungsi untuk mengelola kontrol akses dan hak istimewa pengguna dalam basis data. Fokus utama DCL adalah memastikan keamanan dan integritas data dengan mengatur siapa yang diizinkan untuk mengakses atau memodifikasi data tertentu dalam *database*.

Menurut Date (2019), DCL adalah elemen kunci dari sistem manajemen basis data yang aman, membantu organisasi menjaga kerahasiaan data, mematuhi regulasi, dan melindungi data dari akses atau modifikasi yang tidak sah. Tujuan Utama *Data Control Language* (DCL) :

a. Meningkatkan Keamanan Data

DCL dirancang untuk membatasi akses data hanya kepada pengguna yang diberi izin, memastikan data sensitif tetap aman.

b. Mendukung Kebijakan dan Regulasi

Dalam banyak sektor, seperti keuangan atau kesehatan, regulasi seperti GDPR atau HIPAA mengharuskan data hanya dapat diakses oleh pihak yang berwenang.

c. Mencegah Penyalahgunaan Data

Dengan DCL, administrator dapat memastikan bahwa pengguna hanya dapat melakukan tindakan yang sesuai dengan peran mereka di organisasi.

Fungsi Utama *Data Control Language* (DCL)

1) Memberikan Izin Akses dengan *GRANT*: Statement GRANT digunakan untuk memberikan hak atau izin kepada pengguna untuk melakukan tindakan tertentu pada objek basis data (seperti tabel, kolom, atau prosedur).

Komponen GRANT

a. Hak Istimewa

Menentukan tindakan spesifik yang diizinkan, seperti *SELECT*, *INSERT*, *UPDATE*, atau *DELETE*.

b. Objek

Menentukan objek basis data yang terkait, seperti tabel atau kolom.

c. Penerima

Menentukan pengguna atau peran yang diberi izin. Contoh: Memberikan izin untuk membaca data ini untuk, Pengguna manager diizinkan membaca data dari tabel *employees*.

Statement GRANT

```
GRANT SELECT ON employees TO 'manager'@'localhost';
```

Memberikan izin untuk mengubah data, Ini untuk, Pengguna editor diizinkan mengubah data di tabel *employees*.

Statement GRANT

```
GRANT UPDATE ON employees TO 'editor'@'localhost';
```

Memberikan semua izin, Pengguna admin diizinkan melakukan semua tindakan pada tabel *employees*

Statement GRANT

```
GRANT ALL PRIVILEGES ON employees TO  
'admin'@'localhost';
```

2) Mencabut Izin dengan REVOKE

Statement REVOKE digunakan untuk mencabut izin atau hak istimewa yang sebelumnya diberikan kepada pengguna. Ini penting untuk mengelola akses pengguna ketika peran atau tanggung jawab mereka berubah.

Komponen REVOKE

- Hak Istimewa, hak atau izin spesifik yang akan dicabut.
- Objek, Objek basis data yang terkait.
- Penerima, Pengguna atau peran yang akan dicabut izinnya.

Contoh:

- Mencabut izin untuk memasukkan data, Pengguna editor tidak lagi diizinkan memasukkan data ke tabel *employees*

Statement REVOKE
REVOKE INSERT ON <i>employees</i> FROM 'editor'@'localhost';

- Mencabut semua izin, Semua izin pengguna admin pada tabel *employees* dicabut

Statement REVOKE
REVOKE INSERT ON <i>employees</i> FROM 'editor'@'localhost';

3) Manajemen Pengguna

DCL juga mencakup perintah untuk mengelola pengguna dalam basis data, seperti membuat akun baru, menetapkan kata sandi, atau menghapus akun

- a. *CREATE USER*: Membuat pengguna baru.

Statement
CREATE USER 'new_user'@'localhost' IDENTIFIED BY 'password123';

- b. *DROP USER*: Menghapus akun pengguna.

Statement
DROP USER 'new_user'@'localhost';

- c. *ALTER USER*: Mengubah informasi akun pengguna.

Statement
ALTER USER 'new_user'@'localhost' IDENTIFIED BY 'newpassword456';

- d. *SHOW GRANTS*: Menampilkan izin yang dimiliki oleh pengguna.

Statement
SHOW GRANTS FOR 'manager'@'localhost';

4) Hak Lanjutan dan Fitur Keamanan

Pemberian hak dengan opsi ini memungkinkan pengguna untuk memberikan izin yang sama kepada pengguna lain.

Contoh:

Statement
<code>GRANT SELECT ON employees TO 'manager'@'localhost' WITH GRANT OPTION;</code>

Pengguna manager dapat memberikan izin SELECT kepada pengguna lain.

Replikasi Data

DCL adalah alat penting untuk memastikan bahwa data dikelola secara aman dan sesuai dengan kebijakan organisasi. Dengan adanya kontrol akses yang tepat, organisasi dapat:

- Melindungi Data Sensitif: Mengurangi risiko pencurian atau kebocoran data.
- Memenuhi Regulasi: Mematuhi standar industri seperti GDPR atau HIPAA.
- Meningkatkan Kepercayaan: Menjamin klien bahwa data mereka aman.

DCL adalah elemen penting dalam pengelolaan basis data yang fokus pada keamanan dan kontrol akses. Dengan memanfaatkan perintah seperti GRANT, REVOKE, dan manajemen pengguna, DCL

membantu organisasi memastikan bahwa data mereka tetap aman, hanya diakses oleh pihak yang berwenang, dan dikelola secara efisien.

DAFTAR PUSTAKA

- Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems*. Pearson.
- Date, C. J. (2019). *An Introduction to Database Systems*. Addison-Wesley.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts*. McGraw-Hill

BAB 10

TRANSACTION CONTROL LANGUAGE (TCL) DAN PROCEDURE AND FUNCTION

Oleh : Rizal Lamusu., S.Kom.,MT

A. Pengertian *Transaction Control Language* (TCL)

Transaction Control Language (TCL) adalah subset dari SQL yang digunakan untuk mengelola transaksi dalam basis data. Transaksi merupakan kumpulan perintah SQL yang dieksekusi bersama sebagai unit kerja tunggal. Dengan TCL, pengguna dapat memastikan integritas data melalui mekanisme pengendalian transaksi seperti commit, *rollback*, dan savepoint.

Definisi *Transaction Control Language* (TCL) menurut para pakar:

- a. Connolly dan Begg (2005) : TCL merupakan kumpulan perintah dalam SQL yang memungkinkan pengguna untuk mengontrol proses transaksi dalam sistem basis data, seperti memulai, mengakhiri, atau membatalkan transaksi guna menjaga integritas data.
- b. Silberschatz, Korth, dan Sudarshan (2011) : TCL mencakup perintah-perintah yang berfungsi untuk mengelola proses

transaksi pada basis data, terutama untuk mengatur commit (penyimpanan permanen) dan *rollback* (pembatalan) dari operasi yang dilakukan dalam suatu transaksi.

- c. Elmasri dan Navathe (2015) : *Transaction Control Language* digunakan untuk menangani proses commit dan rollback dalam transaksi basis data, dengan tujuan memastikan bahwa semua operasi dalam transaksi dilakukan secara lengkap atau tidak sama sekali (*atomicity*).

Transaction Control Language (TCL) Terdapat beberapa Komponen Utama :

Komponen Utama TCL

1. COMMIT

Digunakan untuk menyimpan semua perubahan yang dilakukan selama transaksi ke dalam basis data secara permanen.

- Sintaks:

```
```sql
COMMIT;
```
```

- Contoh:

```
```sql
INSERT INTO students (id, name) VALUES (1, 'John');
COMMIT;
```
```

2. **ROLLBACK**

Membatalkan semua perubahan yang belum disimpan, mengembalikan basis data ke kondisi sebelum transaksi dimulai.

- Sintaks:

```
```sql
ROLLBACK;
```
```

- Contoh:

```
```sql
DELETE FROM students WHERE id = 1;
ROLLBACK;
```
```

3. **SAVEPOINT**

Digunakan untuk membuat titik pemulihan dalam transaksi sehingga sebagian transaksi dapat dibatalkan tanpa memengaruhi seluruh transaksi.

- Sintaks:

```
```sql
SAVEPOINT savepoint_name;
```
```

- Contoh:

```
```sql
INSERT INTO students (id, name) VALUES (2, 'Alice');
```

```
SAVEPOINT sp1;
DELETE FROM students WHERE id = 2;
ROLLBACK TO sp1;
...
```

### Manfaat TCL

- a. Konsistensi Data: Menjamin integritas data selama eksekusi transaksi.
- b. Kontrol Transaksi: Memberikan kemampuan untuk menentukan kapan perubahan disimpan atau dibatalkan.
- c. Rekam Jejak Perubahan: Memungkinkan pengguna untuk menelusuri perubahan data dengan cara terstruktur.

### Konsep ACID dalam Transaksi

Prinsip ACID memastikan bahwa setiap transaksi dieksekusi dengan cara yang aman dan andal:

- a. *Atomicity* : Transaksi dilakukan secara penuh atau tidak sama sekali.
- b. *Consistency* : Menjaga konsistensi data sebelum dan sesudah transaksi.
- c. *Isolation*: Transaksi berjalan terisolasi satu sama lain.
- d. *Durability* : Data yang sudah di-commit akan tetap tersimpan meskipun terjadi kegagalan sistem.
- e. Pengertian *Procedure* dan *Function*

## **B. *Procedure* dan *Function***

*Procedure* dan *Function* adalah objek basis data yang digunakan untuk mengeksekusi logika bisnis yang kompleks secara efisien. *Procedure* merupakan serangkaian perintah SQL yang dapat dijalankan berulang kali untuk melaksanakan tugas tertentu. Sementara *Function* adalah blok kode yang mengembalikan nilai tunggal berdasarkan input parameter tertentu. Keduanya berperan penting dalam mengotomatisasi proses, meningkatkan efisiensi sistem, dan mengurangi beban pada aplikasi klien.

Dafinisi *Procedure* dan *Function* menurut para pakar:

- a. Silberschatz et al., (2011) *function* adalah bagian kode terprogram dalam basis data yang bertujuan untuk menyederhanakan operasi tertentu, seperti perhitungan, transformasi data, atau validasi. Function lebih cocok untuk operasi yang membutuhkan pengembalian hasil sebagai bagian dari query SQL.
- b. Coronel et al., (2016) dalam *Database Systems: Design, Implementation, and Management*, function dijelaskan sebagai metode terenkapsulasi yang bertugas mengolah data secara spesifik untuk menghasilkan output yang digunakan oleh pengguna atau aplikasi. Function biasanya dipanggil dalam

query SQL seperti pernyataan SELECT untuk memanipulasi atau memformat data.

### 1. *Procedure*

*Procedure* adalah blok kode SQL yang tersimpan dalam basis data dan dapat dipanggil untuk melaksanakan tugas tertentu. *Procedure* digunakan untuk mengotomatisasi operasi yang kompleks dan sering kali berulang, seperti validasi data atau pembaruan massal.

#### **Membuat Procedure**

Sintaks Dasar :

```
```sql
CREATE PROCEDURE procedure_name
AS
BEGIN
    -- Perintah SQL
END;
```
```

Contoh Implementasi :

```
```sql
CREATE PROCEDURE InsertStudent
AS
BEGIN
    INSERT INTO students (id, name) VALUES (3, 'David');
```

```
END;
```

```
***
```

Memanggil Procedure

Sintaks :

```
```sql
```

```
EXEC procedure_name;
```

```

```

Contoh :

```
```sql
```

```
EXEC InsertStudent;
```

```
***
```

Parameter dalam *Procedure*

Procedure dapat menerima parameter untuk meningkatkan fleksibilitasnya. Parameter dapat berupa:

Parameter : Mengirimkan nilai ke procedure.

OUT Parameter : Mengembalikan nilai dari procedure.

INOUT Parameter : Kombinasi IN dan OUT.

Contoh :

```
```sql
```

```
CREATE PROCEDURE UpdateStudentName (@id INT, @name
VARCHAR(100))
```

```
AS
```

```
BEGIN
```

```
UPDATE students SET name = @name WHERE id = @id;
END;
EXEC UpdateStudentName 3, 'Daniel';
...
```

### **Manfaat Procedure**

Efisiensi Kode : Mengurangi redundansi dengan menggunakan kembali logika yang sama.

Keamanan : Membatasi akses langsung ke tabel.

Performa : Mengurangi beban aplikasi klien dengan memproses logika di sisi server.

## **2. Function**

*Function* adalah blok kode dalam basis data yang menerima parameter, menjalankan logika tertentu, dan mengembalikan nilai tunggal. *Function* biasanya digunakan untuk perhitungan atau manipulasi data yang spesifik.

### **Membuat Function**

Sintaks Dasar :

```
```sql  
CREATE FUNCTION function_name (parameter_name  
data_type)  
RETURNS return_data_type  
AS  
BEGIN
```



```
-- Perintah SQL  
RETURN value;  
END;  
...
```

Contoh Implementasi :

```
```sql  
CREATE FUNCTION GetStudentName (@id INT)
RETURNS VARCHAR(100)
AS
BEGIN
 DECLARE @name VARCHAR(100);
 SELECT @name = name FROM students WHERE id = @id;
 RETURN @name;
END;
...
```

Memanggil Function

Sintaks :

```
```sql  
SELECT function_name(parameter_value);  
...
```

Contoh :

```
```sql  
SELECT GetStudentName(1);
```

...

### Manfaat Function

Kemudahan Penggunaan : Mempermudah pengolahan data yang kompleks.

Efisiensi Query : Mengurangi overhead dengan mengeksekusi logika secara lokal.

Konsistensi Logika : Menjamin bahwa logika bisnis diterapkan dengan cara yang sama di seluruh sistem.

### Perbedaan *Function* dan *Procedure*

Fungsi Utama : Function selalu mengembalikan nilai, sedangkan procedure tidak.

Penggunaan : Function dapat digunakan dalam query SQL seperti SELECT, sedangkan procedure tidak.

Kompleksitas : Procedure cocok untuk operasi yang kompleks, sedangkan function biasanya untuk logika sederhana.

### Studi Kasus

#### Implementasi TCL

Skenario: Menambahkan data mahasiswa baru dan memastikan data disimpan dengan aman.

```
```sql
```

```
BEGIN TRANSACTION;
```

```
INSERT INTO students (id, name) VALUES (4, 'Emma');
```

```
SAVEPOINT sp1;
```

```
UPDATE students SET name = 'Emily' WHERE id = 4;
ROLLBACK TO sp1;
COMMIT;
---
```

Implementasi Procedure dan Function

Skenario: Membuat procedure untuk menambahkan data mahasiswa dan function untuk mengambil nama mahasiswa berdasarkan ID.

```
```sql
CREATE PROCEDURE AddStudent (@id INT, @name
VARCHAR(100))
AS
BEGIN
 INSERT INTO students (id, name) VALUES (@id, @name);
END;

CREATE FUNCTION GetStudentName (@id INT)
RETURNS VARCHAR(100)
AS
BEGIN
 DECLARE @name VARCHAR(100);
 SELECT @name = name FROM students WHERE id = @id;
 RETURN @name;

```

END;

EXEC AddStudent 5, 'Liam';

SELECT GetStudentName(5);

...

Penggunaan ACID pada Transaksi

Skenario: Memastikan data konsisten saat transaksi gagal atau berhasil.

```sql

BEGIN TRANSACTION;

INSERT INTO orders (order_id, customer_name, amount)

VALUES (1, 'Alice', 100);

SAVEPOINT sp1;

INSERT INTO payments (payment_id, order_id, amount) VALUES

(1, 1, 100);

ROLLBACK TO sp1;

COMMIT;

...

Latihan

1. Buat transaksi dengan TCL untuk mengelola data pelanggan dan pesanan.
2. Tulis Procedure untuk mencatat aktivitas pengguna.
3. Tulis Function untuk menghitung harga barang setelah diskon 15%.

C. Kesimpulan

Transaction Control Language, Procedure, dan Function adalah elemen penting dalam pengelolaan dan pemrosesan data di sistem basis data. TCL memastikan integritas data dengan mengelola transaksi. *Procedure* dan *Function* memungkinkan eksekusi logika bisnis secara efisien, yang berkontribusi pada performa sistem basis data secara keseluruhan. Penguasaan ketiga konsep ini memungkinkan pengembang dan administrator basis data untuk merancang sistem yang lebih andal, konsisten, dan mudah dikelola.

DAFTAR PUSTAKA

Date, C. J. (2019). *An Introduction to Database Systems*. Addison-Wesley.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems*. Pearson.

MySQL Documentation. Retrieved from <https://dev.mysql.com/doc/>

BAB 11

INSTALASI SOFTWARE DBMS

Oleh : Muhamad Ikmal Wiawan, M.Kom., IPP

A. Pengantar SOFTWARE RDBMS

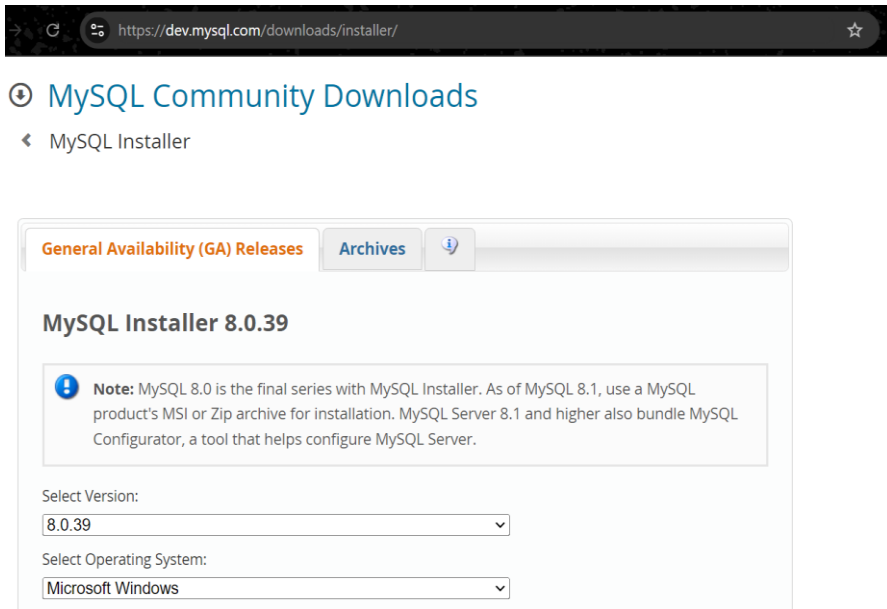
DBMS (*Database Management System*) adalah perangkat lunak yang digunakan untuk mengelola, mengorganisir, dan mengontrol data dalam sebuah basis data. DBMS memungkinkan pengguna untuk membuat, memodifikasi, mengakses, dan mengelola data secara efisien dan aman. Sistem ini juga bertanggung jawab atas integritas data, keamanan, pengelolaan transaksi, dan manajemen akses data. Dengan menggunakan DBMS, organisasi dapat menyimpan data dengan cara yang terstruktur dan mudah diambil kembali, mendukung berbagai aplikasi dan pengguna.

B. Instalasi *Relational Database Mysql Server* di Windows

MySQL adalah salah satu *Relational Database Management System* (RDBMS) yang paling populer dan banyak digunakan di seluruh dunia. MySQL menggunakan bahasa SQL (*Structured*

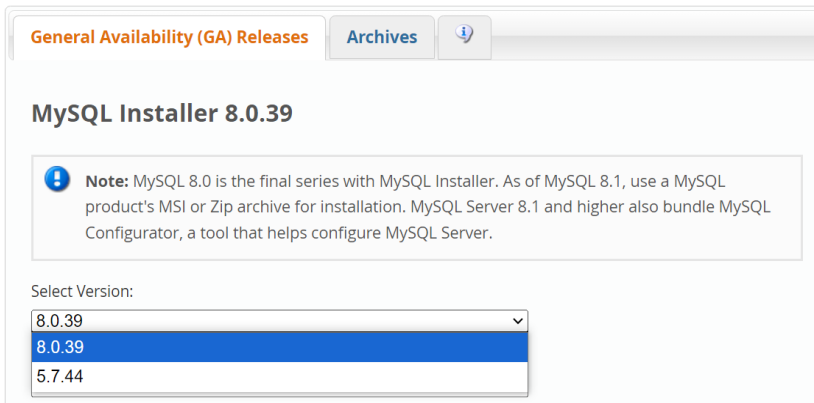
Query Language) untuk mengelola dan memanipulasi data yang disimpan dalam tabel. MySQL dikenal karena kecepatan, skalabilitas, dan keandalannya, serta dukungannya terhadap berbagai platform seperti Windows, Linux, dan Mac OS. MySQL biasanya digunakan dalam aplikasi berbasis web seperti WordPress, Joomla, dan Drupal. Berikut ini adalah Langkah Langkah instalasi Mysql / Mariadb Di Sistem Operasi Windows :

1. Download Installer Mysql dari Sumber terpercaya yaitu dari laman resmi sesuai dengan link berikut :
<https://dev.mysql.com/downloads/installer/>.



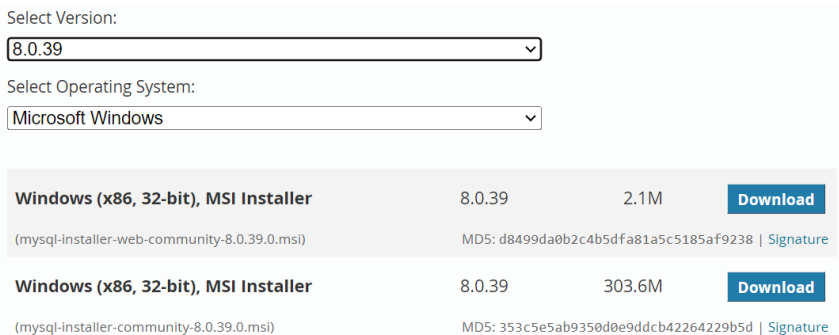
Gambar 11. 1 Laman Awal situs *Downloader* Mysql

2. Pilih Versi Mysql yang hendak digunakan. Versi terbaru saat ini yaitu Mysql dengan Versi



Gambar 11. 2 Laman Awal situs Downloader Mysql

Saat ini terdiri dari dua Versi yaitu versi 8.0.39 dan versi 5.7.44. tidak disarankan Menggunakan versi di bawah 8.x.x karena versi lama biasanya memiliki keterbatasan fitur dan Terdapat kerentanan keamanan. Terdapat dua jenis Installer Mysql yaitu online dan Offline Installer.



Gambar 11. 3 Laman Awal situs *Downloader* Mysql

Online Installer merupakan source instalasi yang tidak menyimpan installer pada saat download melainkan akan secara otomatis mengunduhnya pada saat proses set up. Sedangkan *offline installer* merupakan *source* instalasi yang secara lengkap tersedia di dalam file unduhan, biasanya ukurannya lebih besar dari *Online Installer*. Penulis merekomendasikan *offline installer* supaya dapat digunakan kembali untuk kebutuhan instalasi di tempat lain.

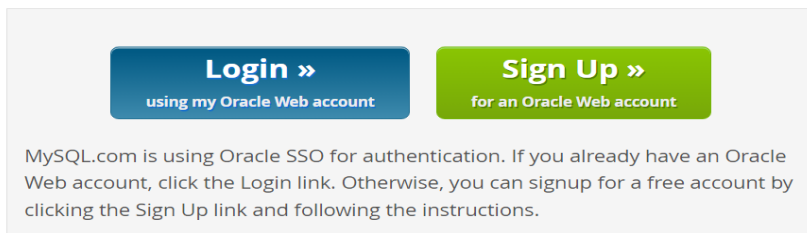
3. Setelah menentukan jenis *installer* yang akan diunduh, selanjutnya pembaca akan meng-klik tautan link tersebut dan diarahkan ke halaman Login sebagai berikut :

📥 [MySQL Community Downloads](#)

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system



Gambar 11. 4 Laman Login / *Signup* (pendaftaran) akun oracle.

Proses Pendaftaran akun Oracle ini gratis dan mudah namun menjadikan syarat wajib dalam Langkah akhir pengunduhan

installer. Setelah selesai login atau signup maka akan muncul tampilan sebagai berikut :

MySQL Community Downloads

To begin your download, please click the Download Now button below.



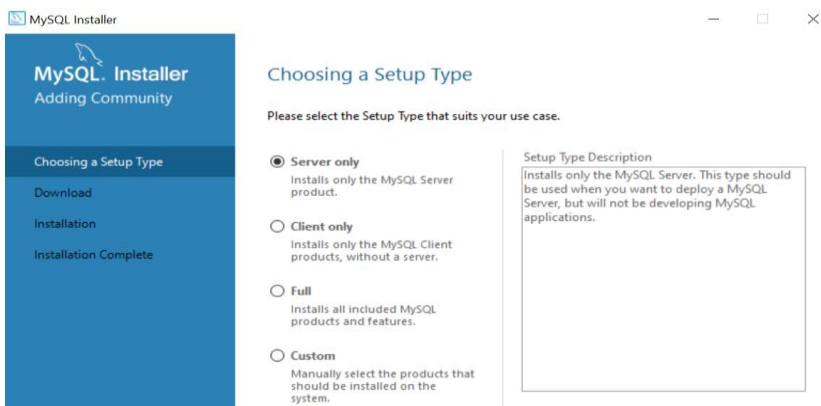
MD5: 353c5e5ab9350d0e9ddcb42264229b5d

Size: 303.6M

[Signature](#)

Gambar 11. 5 Link Download Mysql yang sudah tersedia

4. Setelah Selesai Download, selanjutnya mengeksekusi file unduhan tersebut
5. Halaman awal akan menampilkan dialog Pilihan paket Instalasi mysql yang ditawarkan



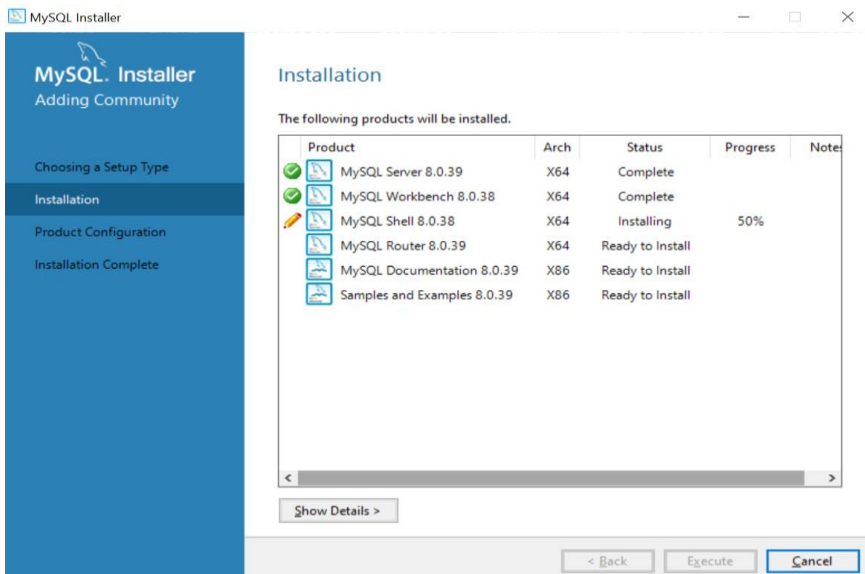
Gambar 11. 6 Pilihan Paket Instalasi Mysql

- a. *Server Only*: Instalasi hanya Mysql Servernya saja
- b. *Client Only*: Instalasi hanya mysql Clientnya saja
- c. *Full* : Instalasi Mysql *Client*, *Server*, *Tools* Pendukung dan dokumentasi
- d. *Custom*: Instalasi yang dapat disesuaikan dengan kebutuhan

Penulis menyarankan untuk memilih Pilihan “*Full*” *Installation*, agar pemula dapat memastikan semua kebutuhan tools dan Dokumentasi tersedia di awal instalasi.

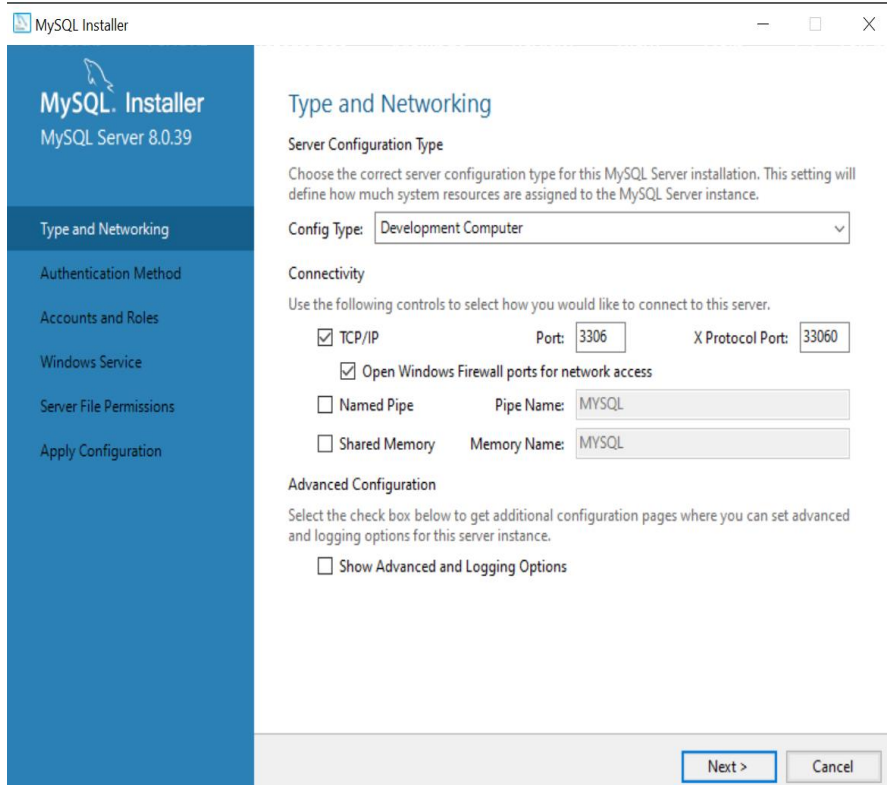
6. Klik “*Execute*”, untuk memulai proses instalasi

Berikut tampilan pada proses Instalasi yang sedang berjalan



Gambar 11. 7 Pilihan Paket Instalasi Mysql

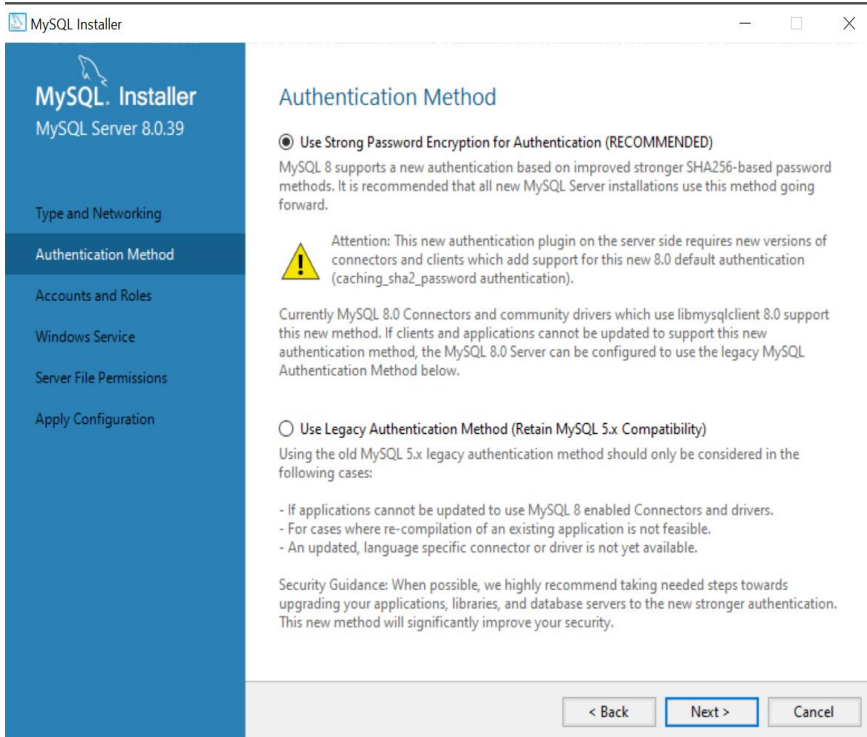
7. Setelah selesai Instalasi, berikutnya adalah konfigurasi “*Type and Networking*” sebagai berikut :



Gambar 11. 8 Pilihan Paket Instalasi Mysql

Pada dasarnya konfigurasi di atas sudah merupakan kondisi “*Default*”, yang secara umum digunakan. Jika pembaca hendak mengubah Portnya maka silahkan mengganti port 3306 menjadi port yang diinginkan. Setelah itu klik “*Next*”

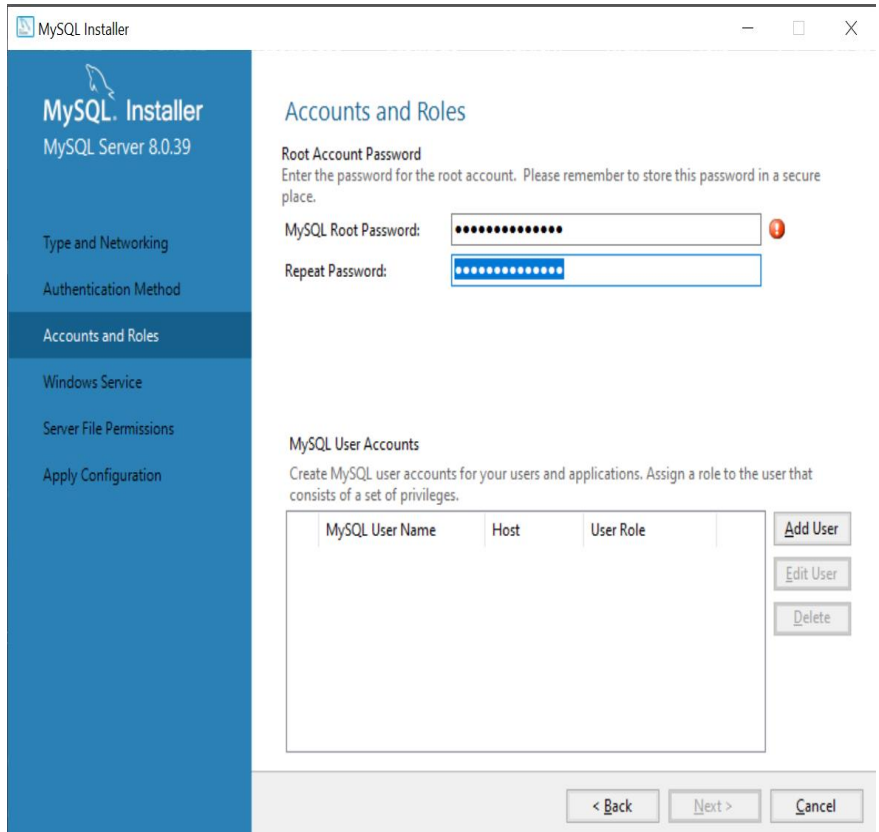
8. Langkah berikutnya adalah menentukan Jenis Otentikasi pada *Mysql Server* yang digunakan.



Gambar 11. 9 Metode Otentikasi Mysql Server.

“Use Strong Password Encryption for Authentication”, merupakan otentikasi yang Menggunakan Hash SHA-256 sehingga aman dan lebih direkomendasikan. Sedangkan *“ Use Legacy Authentication Method”*, merupakan otentikasi yang Menggunakan hash Md5 yang kompatibel dengan mysql server versi 5.x. selanjutnya Klik Next.

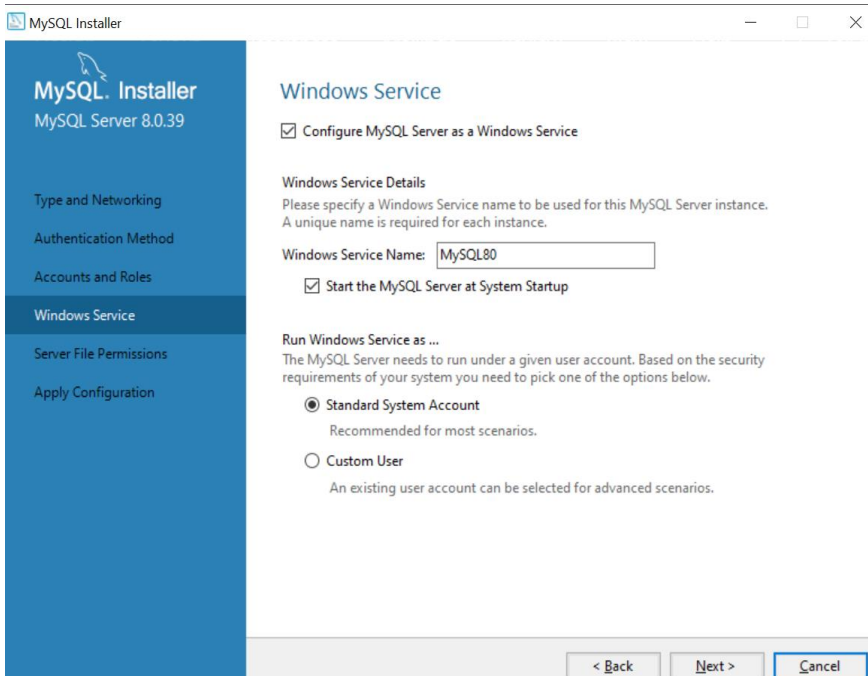
9. Pada tahapan ini pembaca akan Menampilkan laman konfigurasi User dan *Password Mysql*



Gambar 11. 10 Laman Konfigurasi Akun dan Role.

Jika pembaca hendak melakukan konfigurasi role user secara manual maka pembaca harus mengklik “*Add User*”, pada bagian bawah dialog box lalu kemudian menentukan hak aksesnya. Jika tidak maka tinggal meng-klik “*next*”.

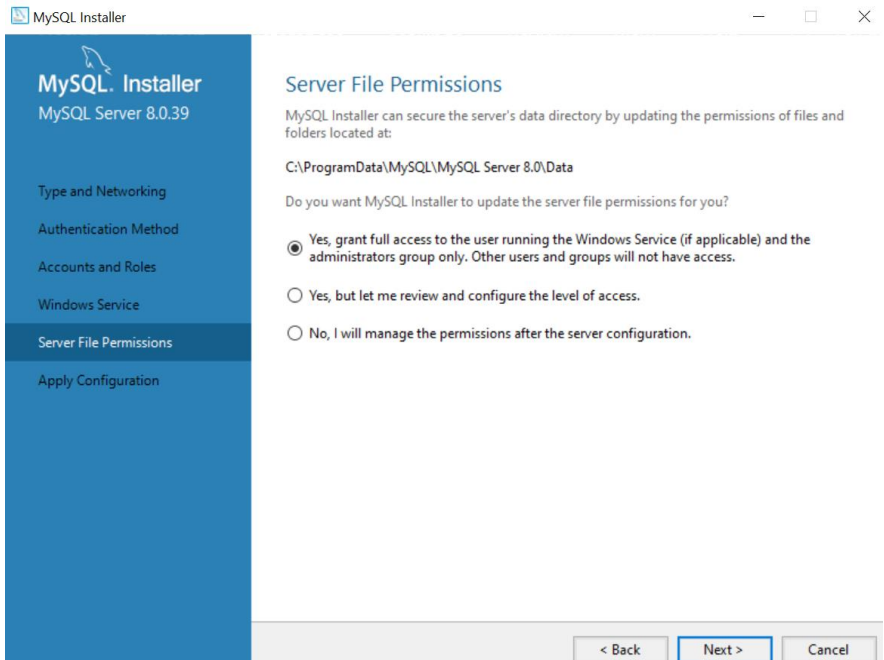
10. Berikutnya adalah tahapan konfigurasi Service di Sistem Operasi Windows.



Gambar 11. 11 Laman Konfigurasi Akun dan Role.

“Start The Mysql Server at System Startup”, adalah konfigurasi untuk menentukan apakah service mysql server akan berjalan otomatis startup atau tidak. Kemudian klik *next*.

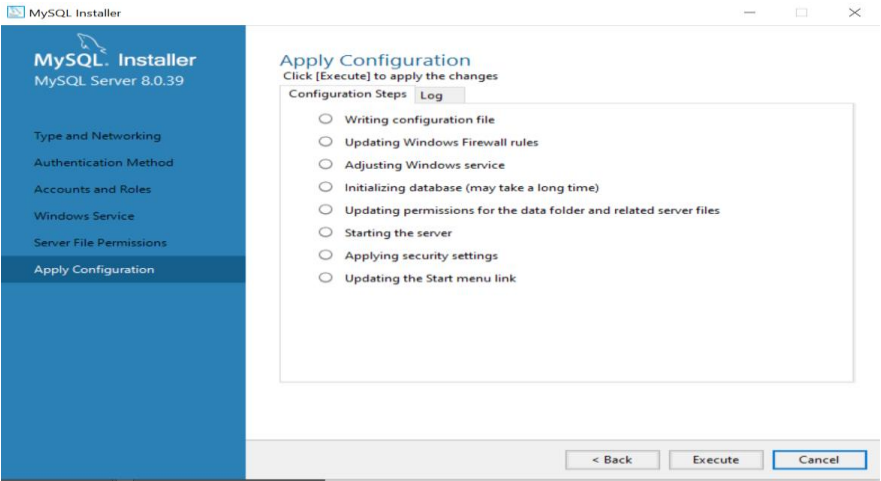
11. Langkah Berikutnya adalah Konfigurasi “*Server File Permission*”.



Gambar 11. 12 Konfigurasi *Server File Permissions*

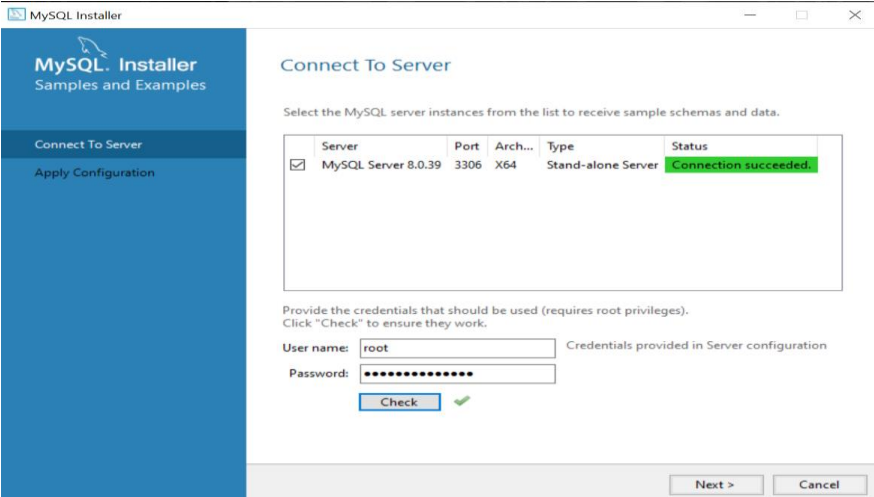
Opsi Pertama dimaksudkan untuk Memberikan akses penuh mysql terhadap folder “C:\ProgramData\MySql\MySql Server 8.0\Data”. Opsi kedua adalah Opsi pemberian akses namun kita menentukan sendiri hak akses dan sub foldernya. Opsi ketiga adalah tidak memberikan hak akses sama sekali kepada folder tersebut melainkan akan dikonfigurasi setelah instalasi.

12. Tampilan akhir konfigurasi instalasi Mysql , kemudian klik eksekusi



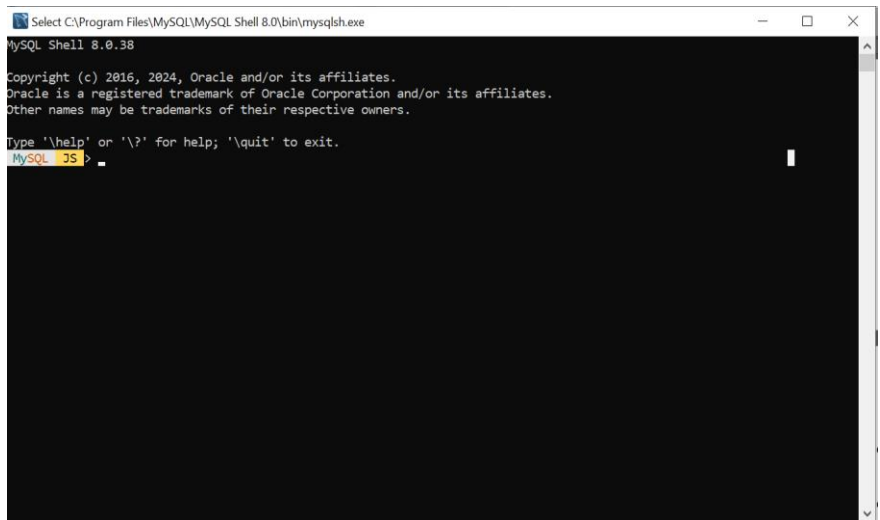
Gambar 11. 13 Tampilan akhir konfigurasi dan instalasi mysql

13. Langkah terakhir adalah Pengujian hasil konfigurasi



Gambar 11. 14 Pengujian hasil instalasi

Tampilan Awal Mysql CLI setelah Proses Instalasi berhasil



Gambar 11. 15 Tampilan Mysql CLI

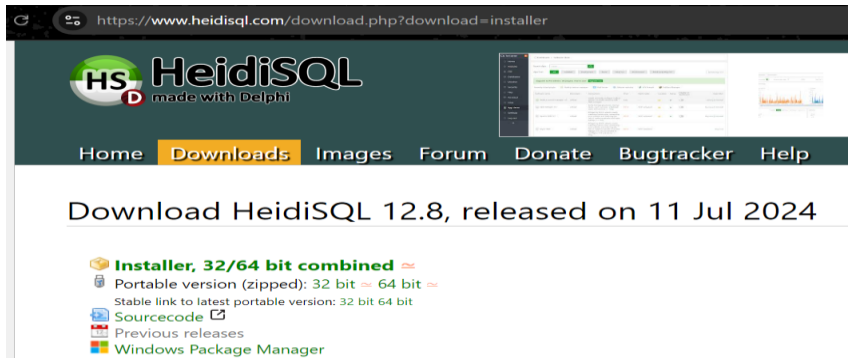
C. Instalasi *Relational Database Mysql Client* di Windows

Database Management System (DBMS) terdiri dari dua komponen utama yaitu *client* dan *Server*. *Database* server bertugas sebagai komponen utama yang mengelola, menyimpan, mengendalikan dan menyediakan akses kepada client. Sedangkan *Database* client adalah *Software* yang menjembatani Pengguna saat berinteraksi dengan *Database server*, atau dalam kata lain sebagai antarmuka pengguna dalam melakukan DML, DDL, dan DCL. Banyak sekali di pasaran vendor yang menyediakan *tools Database client* baik yang *opensource*, *freeware* maupun berbayar sebagai berikut :

- a. MySQL Workbench
- b. pgAdmin
- c. DBeaver
- d. HeidiSQL
- e. Navicat
- f. SQL Server Management Studio (SSMS)
- g. Toad for SQL Server
- h. Squirrel SQL
- i. DataGrip
- j. DbVisualizer
- k. Aqua Data Studio
- l. RazorSQL
- m. SQLyog
- n. Mysql CLI

Dari sekian banyak daftar *software / tools database Client*, penulis merekomendasikan heidiSQL sebagai bahan pembahasan karena *software* ini sangat mudah, selalu diperbarui dan *Opensource*. Berikut ini adalah Langkah Langkah Instalasi dan konfigurasi HeidiSQL :

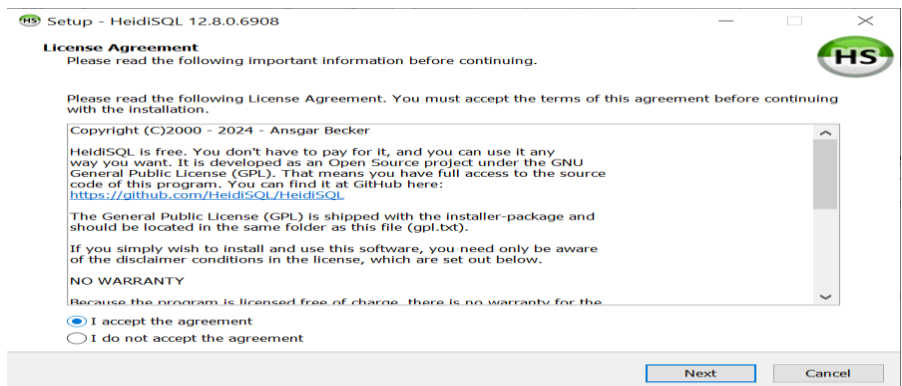
1. Download Installer heidiSQL dari website resminya :
<https://www.heidisql.com/download.php?download=installer>



Gambar 11. 16 Pengujian hasil instalasi

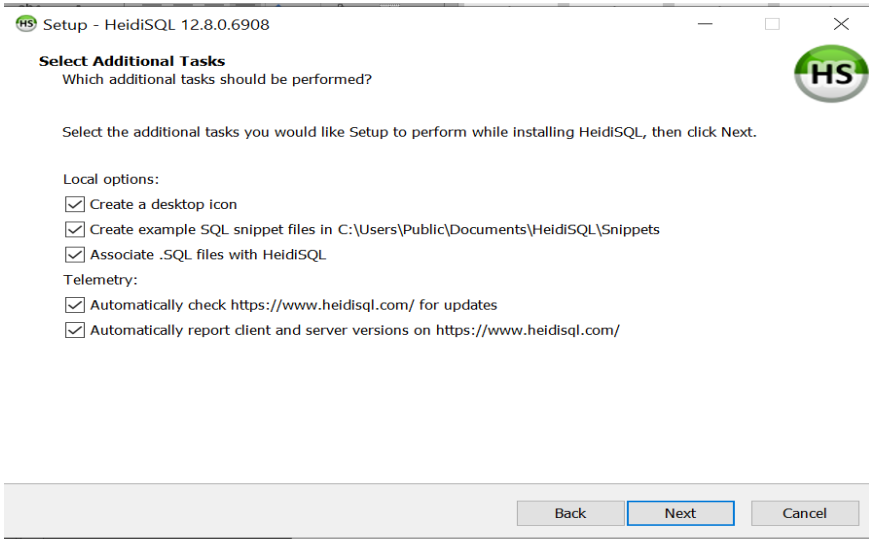
Tersedia dua pilihan jenis file uduhan yaitu *Installer* dan *Portable Version*. *Installer* merupakan media instalasi yang umum dipilih untuk menjadi sumber proses instalasi sedangkan *Portable* merupakan *software* yang dapat dibuka secara langsung tanpa melalui proses instalasi terlebih dahulu.

2. Halaman Awal Instalasi berisi persetujuan lisensi, kemudian klik **Next**



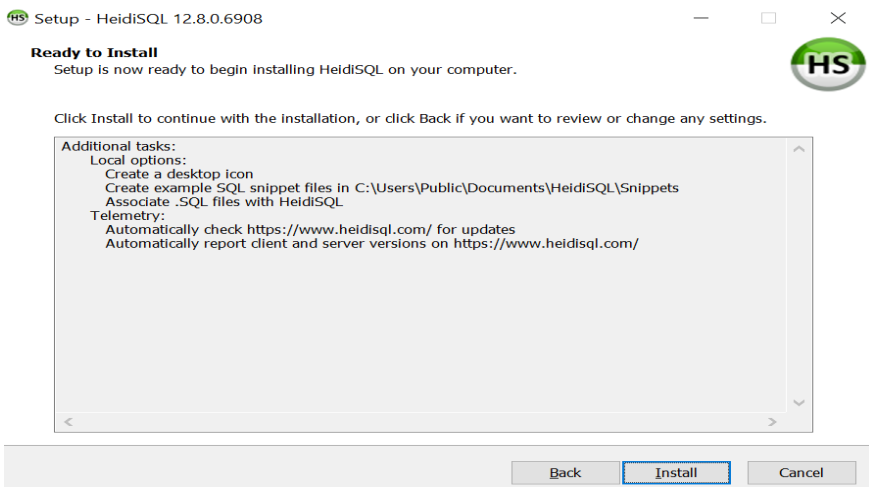
Gambar 11. 17 *License Agreement* tahapan instalasi HeidiSQL

3. Pemilihan Tasks Tambahan proses Instalasi HeidiSQL

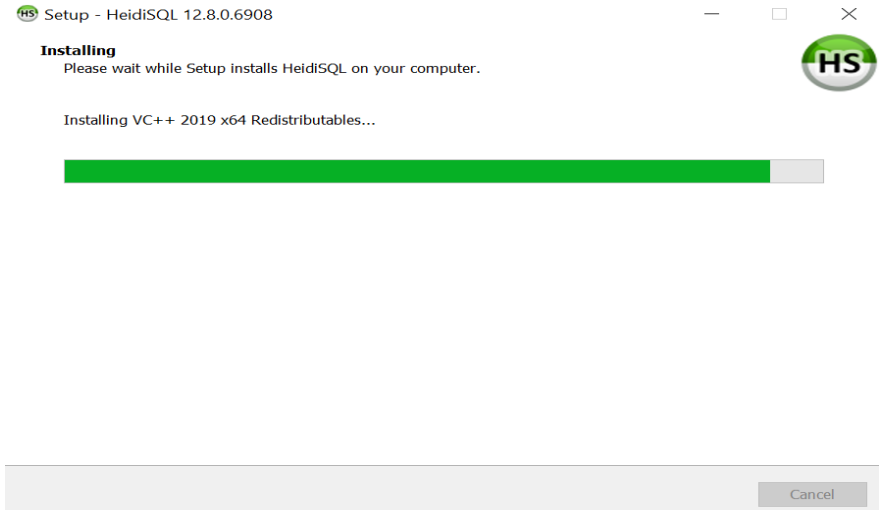


Gambar 11. 18 *Aditonal Tasks Instalation Process*

4. Halaman Persiapan Instalasi heidiSQL

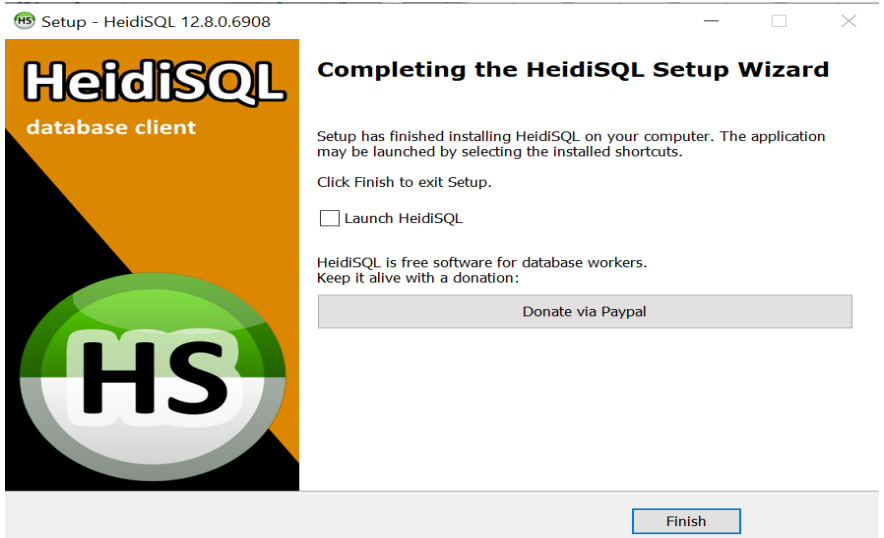


Gambar 11. 19 *Halaman Dialog Ready to Install*



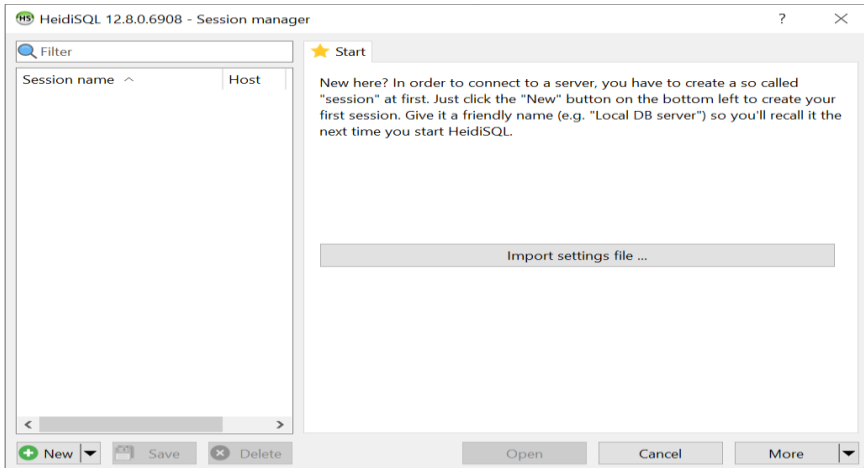
Gambar 11. 20 Proses Instalasi yang sedang berlangsung.

5. Hasil Akhir Proses Instalasi



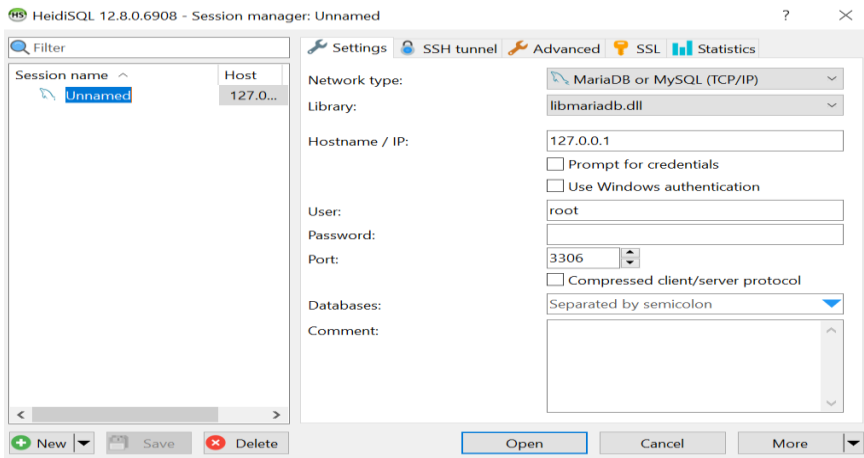
Gambar 11. 21 Hasil Akhir Proses Instalasi HeidiSQL

6. Tampilan Awal / *Default* HeidiSQL



Gambar 11. 22 Tampilan Awal HeidiSQL

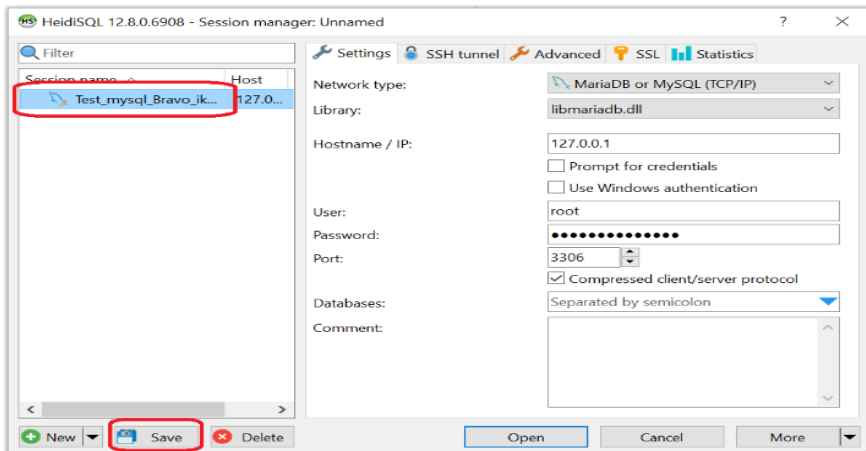
7. Tahapan Selanjutnya adalah kita mengkoneksikan *Database* server yang telah dibuat di Point 11.2 di atas Menggunakan antarmuka HeidiSQL dengan cara meng-klik “New”



Gambar 11. 23 Tampilan Pembuatan Koneksi ke Mysql Server

- a. *Network Type* : “MariaDB or Mysql (TCP/IP)”
- b. *Library* : “Libmariadb.dll”.
- c. *Hostname* : “127.0.0.1”, atau sesuaikan dengan alamat IP server yang digunakan pada tahapan instalasi Mysql Server sebelumnya.
- d. *User* : “root”, atau sesuaikan Dengan Konfigurasi User yang digunakan pada tahapan instalasi Mysql Server sebelumnya.
- e. *Password* : sesuaikan Dengan Konfigurasi User password yang digunakan pada tahapan instalasi Mysql Server sebelumnya.
- f. *Port* : “3306”, atau sesuaikan dengan alamat *Port server* yang digunakan pada tahapan instalasi Mysql Server sebelumnya.
- g. *Database* : Optional, abaikan jika tidak ingin spesifik menentukan *Database* yang digunakan setelah login ke mysql Server.

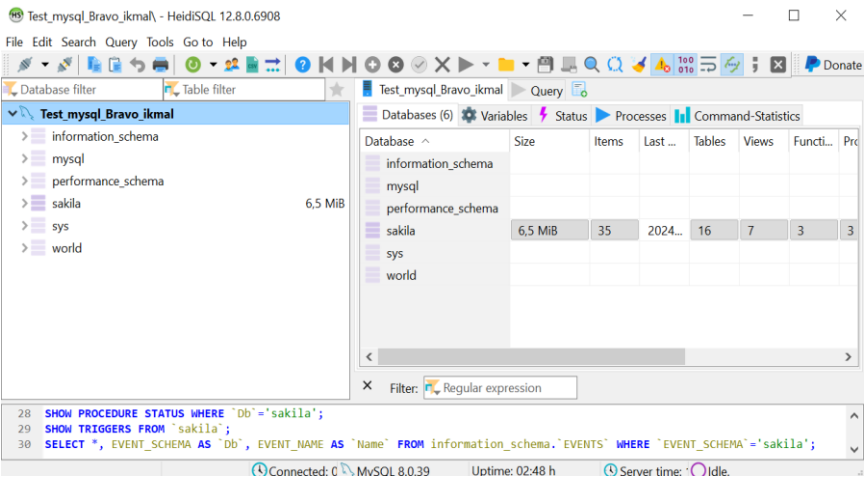
Setelah Selesai Melakukan Konfigurasi, maka Klik Simpan terlebih dahulu, kemudian klik kanan rename sesuaikan nama koneksi database dengan yang diinginkan, lalu simpan lagi.



Gambar 11. 24 Konfigurasi Akhir Setup Koneksi DBMS heidiSQL

Setelah selesai maka klik “Open”.

8. Berikut ini adalah Tampilan yang membuktikan keberhasilan koneksi konfigurasi HeidiSQL dengan Mysql Server



Gambar 11. 25 Tampilan Antarmuka HeidiSQL yang sudah terkoneksi ke Mysql Server

DAFTAR PUSTAKA

Connolly, T., & Begg, C. (2014). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). Pearson Education.

BAB 12

PROSES *RECOVERY* DAN *DATABASE SECURITY*

Oleh : Irawan Ibrahim, S.T M.Kom

Dalam era digital yang semakin berkembang pesat, informasi menjadi aset yang sangat berharga bagi organisasi dan individu. Dalam konteks ini, database memainkan peran yang sangat penting dalam penyimpanan, pengelolaan, dan pengambilan data. Tanpa adanya sistem database yang efisien dan andal, keberhasilan suatu organisasi dalam mengelola informasi akan terhambat. Namun, seperti halnya teknologi lainnya, sistem database juga tidak terhindar dari risiko kegagalan. Baik itu karena kesalahan manusia, kerusakan perangkat keras, serangan siber, atau bencana alam, data yang ada dalam database dapat hilang atau menjadi tidak konsisten.

Oleh karena itu, dua aspek yang sangat penting dalam pengelolaan database adalah proses *recovery* dan keamanan database. Keduanya tidak hanya penting untuk memastikan

keandalan dan ketersediaan data, tetapi juga untuk menjaga integritas dan kerahasiaan informasi yang disimpan. Tanpa adanya mekanisme *recovery* yang tepat, organisasi berisiko kehilangan data yang vital, sementara tanpa pengamanan yang memadai, data tersebut dapat jatuh ke tangan yang salah atau rusak akibat serangan.

A. Proses *Recovery* dalam Database

Proses *recovery* merujuk pada serangkaian teknik yang digunakan untuk mengembalikan data ke keadaan yang benar dan konsisten setelah mengalami kegagalan. Kegagalan ini bisa disebabkan oleh banyak faktor, seperti *crash* sistem, kesalahan pengguna, atau kerusakan perangkat keras. Dalam konteks ini, *recovery* bertujuan untuk memastikan bahwa data tetap tersedia, transaksi yang sah tetap tersimpan, dan transaksi yang tidak lengkap atau tidak sah dapat dibatalkan dengan aman.

Mekanisme *recovery* ini juga memainkan peran penting dalam menjaga prinsip ACID (*Atomicity, Consistency, Isolation, Durability*) yang harus dijaga dalam setiap sistem database. Jika satu bagian dari transaksi gagal, maka sistem *recovery* harus memastikan bahwa transaksi lainnya tetap terjaga konsistensinya dan data tidak dalam keadaan korup atau hilang. Oleh karena itu, pemahaman yang mendalam mengenai berbagai metode *recovery*,

seperti *log-based recovery*, *checkpointing*, dan *backup & restore*, sangat diperlukan untuk menjaga kelangsungan operasional database dalam menghadapi kegagalan.

1. Pengertian

Recovery dalam database adalah serangkaian teknik dan prosedur yang digunakan untuk mengembalikan sistem database ke keadaan yang konsisten setelah terjadinya suatu kegagalan. Kegagalan yang terjadi bisa berupa kesalahan pengguna, kerusakan perangkat keras, kesalahan sistem, atau bahkan bencana alam yang mengganggu operasional database. Tanpa adanya mekanisme *recovery* yang tepat, data yang hilang atau rusak dapat menyebabkan gangguan serius pada aplikasi dan layanan yang bergantung pada database tersebut.

Secara umum, tujuan utama dari *recovery* database adalah untuk memastikan bahwa database kembali ke dalam kondisi yang benar, dengan cara:

- 1) Mencegah kehilangan data: Data yang telah dimodifikasi atau ditambahkan oleh transaksi yang telah diselesaikan harus tetap aman.
- 2) Menjaga konsistensi data: Data dalam database harus tetap memenuhi aturan-aturan integritas yang ditentukan, meskipun terjadi kegagalan.

- 3) Memulihkan transaksi yang terganggu: Transaksi yang tidak selesai atau terputus akibat kegagalan harus dapat dibatalkan (*rollback*) atau dilanjutkan kembali (*redo*), tergantung pada statusnya.
- 4) Proses *recovery* ini juga harus memastikan bahwa database kembali ke dalam kondisi yang memenuhi prinsip ACID (*Atomicity, Consistency, Isolation, Durability*), yang menjadi dasar dari sistem transaksi dalam database. Dengan kata lain, *recovery* harus menjamin bahwa:
 - a) *Atomicity*: Transaksi yang gagal sepenuhnya dibatalkan, dan transaksi yang berhasil tetap dipertahankan.
 - b) *Consistency*: Database harus kembali ke dalam keadaan yang konsisten setelah *recovery*.
 - c) *Isolation*: Transaksi yang terjadi setelah *recovery* tidak boleh dipengaruhi oleh transaksi yang belum selesai.
 - d) *Durability*: Data yang telah berhasil diproses dan disimpan tidak akan hilang setelah kegagalan.

2. Jenis-Jenis Kegagalan dalam Sistem Database

Proses *recovery* hanya diperlukan jika terjadi kegagalan sistem, yang bisa dikategorikan dalam beberapa jenis berdasarkan penyebab dan dampaknya. Masing-masing jenis kegagalan

memiliki implikasi yang berbeda terhadap data, dan karena itu memerlukan pendekatan *recovery* yang berbeda pula.

1) Kegagalan Transaksi (*Transaction Failure*)

Kegagalan transaksi terjadi ketika suatu transaksi yang sedang diproses gagal karena berbagai alasan, seperti:

- a) Kesalahan input data: Misalnya, pelanggaran aturan integritas (seperti nilai duplikat pada kolom dengan kunci unik).
- b) Deadlock: Terjadi ketika dua atau lebih transaksi saling menunggu satu sama lain untuk mendapatkan sumber daya yang terkunci, dan tidak dapat diselesaikan.
- c) Kesalahan aplikasi atau bug: Misalnya, aplikasi yang menyebabkan transaksi tidak selesai dengan benar.
- d) Dalam kasus kegagalan transaksi, rollback adalah teknik yang digunakan untuk membatalkan perubahan yang dilakukan oleh transaksi yang gagal, sehingga database kembali ke keadaan sebelum transaksi dimulai.

2) Kegagalan Sistem (*System Failure*)

Kegagalan sistem terjadi ketika sistem operasi atau perangkat lunak yang menjalankan database mengalami *crash* atau gangguan, seperti kerusakan sistem operasi yang

menyebabkan server database berhenti bekerja. Kegagalan perangkat keras yang menyebabkan database tidak bisa diakses atau dimodifikasi. Kegagalan sistem biasanya mengakibatkan hilangnya data yang belum tertulis ke disk. Dalam hal ini, recovery dilakukan dengan menggunakan log transaksi dan checkpoint untuk memastikan bahwa perubahan yang belum tersimpan tidak mengganggu integritas database.

3) Kegagalan Perangkat Keras (*Hardware Failure*)

Kegagalan perangkat keras dapat terjadi akibat kerusakan pada perangkat penyimpanan (seperti hard disk atau SSD), memori, atau bahkan gangguan pada RAID (*Redundant Array of Independent Disks*). Kegagalan ini dapat menyebabkan database menjadi tidak dapat diakses atau data hilang. Jika kegagalan perangkat keras terjadi, backup yang disimpan di lokasi terpisah atau di media redundan (seperti *cloud storage*) akan sangat membantu untuk memulihkan data. Salah satu teknik yang digunakan adalah RAID, yang menawarkan redundansi dan melindungi data dari kerusakan perangkat keras.

4) Kegagalan Bencana (*Catastrophic Failure*)

Kegagalan bencana seperti kebakaran, banjir, atau serangan siber (misalnya *ransomware*) dapat menghancurkan seluruh sistem database. Dalam situasi ini, teknik *recovery* bergantung

pada *backup* eksternal yang disimpan di lokasi terpisah dan aman. Untuk mengatasi bencana besar, penyimpanan cadangan di lokasi geografis yang berbeda dan penggunaan cloud-based *backup* menjadi solusi yang lebih aman dan efisien.

3. Metode *Recovery* dalam Database

Terdapat berbagai metode yang dapat digunakan untuk memulihkan database setelah terjadi kegagalan. Pemilihan metode yang tepat bergantung pada jenis kegagalan yang terjadi dan tingkat kerusakan yang dialami oleh sistem database.

1) *Log-Based Recovery*

Log-based recovery adalah salah satu metode *recovery* yang paling umum digunakan dalam sistem database modern. Metode ini mengandalkan pencatatan log transaksi yang mencatat setiap perubahan yang dilakukan pada database. Log transaksi ini berfungsi untuk memastikan bahwa database tetap konsisten meskipun terjadi kegagalan. Redo (*roll-forward*) adalah proses menerapkan kembali perubahan yang tercatat dalam log transaksi untuk transaksi yang telah berhasil diselesaikan, tetapi belum disimpan secara permanen ke disk. Undo (*roll-back*) adalah proses membatalkan transaksi yang tidak selesai atau gagal, dengan cara mengembalikan data ke

keadaan sebelum transaksi dimulai. Log transaksi memiliki dua jenis utama:

- a) *Write-Ahead Logging (WAL)*: Teknik ini memastikan bahwa perubahan data selalu dicatat dalam log sebelum ditulis ke disk.
- b) *Immediate Update*: Teknik ini memungkinkan perubahan data dilakukan langsung ke disk, sementara log digunakan untuk melacak perubahan tersebut.

2) *Checkpointing*

Checkpointing adalah proses di mana database menyimpan titik pemulihan (*checkpoint*) pada suatu waktu tertentu. *Checkpoint* ini mencatat keadaan database, termasuk data dan transaksi yang sedang berlangsung. Ketika terjadi kegagalan, proses *recovery* dapat dimulai dari checkpoint terakhir, yang mengurangi waktu yang diperlukan untuk pemulihan dibandingkan jika harus memulihkan seluruh log transaksi.

Checkpoint memiliki dua keuntungan utama:

- a) Meningkatkan Efisiensi *Recovery*: Dengan *checkpoint*, hanya perubahan setelah checkpoint yang perlu diproses ulang.
- b) Mengurangi Ukuran Log yang Dibutuhkan: Data yang telah dicatat dalam *checkpoint* tidak perlu dimasukkan lagi dalam log selama proses *recovery*.

3) *Shadow Paging*

Shadow paging adalah metode *recovery* yang tidak bergantung pada penggunaan log transaksi. Metode ini menyimpan dua salinan halaman data: satu salinan aktif yang digunakan oleh sistem, dan satu salinan bayangan (*shadow*) yang merupakan salinan stabil dari data sebelum perubahan dilakukan. Jika terjadi kegagalan, sistem cukup mengganti salinan halaman aktif dengan salinan bayangan, sehingga database kembali ke kondisi sebelumnya tanpa perlu melakukan rollback atau redo transaksi. Meskipun memiliki keuntungan dalam hal kecepatan dan pengurangan overhead, teknik ini kurang fleksibel dibandingkan *log-based recovery* karena tidak mencatat transaksi secara rinci.

4) *Backup dan Restore*

Metode *recovery* lainnya adalah menggunakan backup data. Backup adalah salinan dari database yang disimpan secara terpisah dari database utama. Proses pemulihan dari backup dilakukan dengan restore, yaitu memulihkan data dari salinan cadangan yang telah disimpan. Jenis backup yang digunakan dapat berupa:

- a) *Full Backup*: Salinan lengkap dari seluruh database.
- b) *Incremental Backup*: Menyimpan perubahan data sejak backup terakhir.

- c) *Differential Backup*: Menyimpan perubahan data sejak backup penuh terakhir.
- d) *Backup* yang teratur dan dikelola dengan baik adalah salah satu cara paling efektif untuk melindungi database dari kegagalan besar, seperti bencana alam atau kerusakan perangkat keras.

4. Langkah-Langkah dalam Proses *Recovery*

Setelah kegagalan terjadi, langkah-langkah *recovery* yang tepat harus diambil untuk memastikan bahwa database dapat dipulihkan ke keadaan yang konsisten dan dapat diakses kembali. Langkah-langkah tersebut meliputi:

- a) **Identifikasi Kegagalan:** Langkah pertama adalah mengidentifikasi jenis kegagalan yang terjadi, apakah itu kegagalan transaksi, sistem, perangkat keras, atau bencana.
- b) **Pemulihan dengan Log:** Jika menggunakan log-based recovery, lakukan redo dan undo untuk transaksi yang gagal atau belum selesai.
- c) **Pemulihan dengan *Checkpoint* atau *Backup*:** Jika menggunakan teknik *checkpointing* atau *backup, restore* data yang hilang dan terapkan perubahan yang relevan.

- d) Verifikasi Keamanan dan Integritas Data: Setelah pemulihan, lakukan verifikasi untuk memastikan bahwa data yang dipulihkan tidak terkontaminasi atau rusak.
- e) Pemantauan dan Pemulihan Sistem Lanjutan: Setelah database pulih, sistem harus terus dipantau untuk memastikan tidak ada masalah tambahan yang muncul dan untuk memperbaiki potensi celah keamanan.

5. Studi Kasus dan Simulasi *Recovery*

Sebagai ilustrasi, kita akan melihat beberapa studi kasus di mana metode *recovery* ini diterapkan, termasuk cara organisasi menangani pemulihan setelah kegagalan, serta bagaimana *recovery* dilakukan dengan menggunakan *log*, *checkpoint*, dan *backup*.

B. Keamanan Database (*Database Security*)

1. Keamanan Database

Keamanan database adalah upaya untuk melindungi database dari berbagai ancaman yang dapat merusak, mengakses secara tidak sah, atau mengubah data yang ada di dalamnya. Keamanan ini mencakup berbagai aspek, seperti kerahasiaan (*confidentiality*), integritas (*integrity*), dan ketersediaan (*availability*) data yang disimpan dalam sistem database. Sistem database yang aman

tidak hanya mencegah akses tidak sah, tetapi juga memastikan bahwa data tetap akurat dan tidak dapat dimanipulasi oleh pihak yang tidak berwenang.

Keamanan database sangat penting mengingat bahwa database modern menyimpan informasi yang sangat bernilai, seperti data pribadi, finansial, kesehatan, dan bahkan data sensitif terkait strategi bisnis. Jika informasi ini jatuh ke tangan yang salah, bisa menimbulkan kerugian yang sangat besar bagi individu maupun organisasi. Oleh karena itu, pengelolaan keamanan database yang efektif menjadi salah satu prioritas utama dalam manajemen sistem informasi. Keamanan database tidak hanya berfokus pada perlindungan data, tetapi juga mencakup proses pemulihan (*recovery*) data yang aman setelah terjadinya serangan atau kegagalan sistem, serta memastikan bahwa data tetap tersedia dan dapat diakses oleh pengguna yang berwenang kapan saja.

2. Ancaman terhadap Keamanan Database

Ancaman terhadap keamanan database dapat berasal dari berbagai sumber, baik itu internal maupun eksternal. Ancaman ini bisa sangat merusak, baik dalam skala kecil maupun besar, dan dapat mempengaruhi seluruh integritas dan keberlanjutan operasional database.

a. Ancaman Internal

Ancaman internal biasanya datang dari pengguna atau administrator sistem yang memiliki akses sah ke database, tetapi menyalahgunakan akses tersebut. Ini termasuk: Kesalahan manusia: Pengguna yang tidak sengaja menghapus data, melakukan perubahan yang salah, atau salah konfigurasi sistem. Penyalahgunaan akses: Seorang administrator yang mencoba mengakses data yang tidak berhak atau melakukan modifikasi data untuk kepentingan pribadi.

Penggunaan akses yang tidak terkendali: Pengguna atau aplikasi yang memiliki akses terlalu luas tanpa pembatasan yang cukup, memungkinkan mereka untuk memanipulasi atau merusak data. Untuk mengurangi ancaman internal, kontrol akses yang ketat dan kebijakan penggunaan yang jelas sangat penting. Penggunaan peran pengguna (*user roles*) dan pemisahan tugas (*segregation of duties*) dapat membantu mencegah penyalahgunaan akses.

b. Ancaman Eksternal

Ancaman eksternal adalah serangan yang berasal dari luar organisasi atau sistem database. Beberapa jenis ancaman eksternal yang paling umum adalah: SQL Injection: Salah satu metode serangan yang paling populer di mana penyerang

menyisipkan perintah SQL berbahaya melalui aplikasi untuk mengeksploitasi celah keamanan di aplikasi web yang terhubung dengan database. Tujuan dari SQL injection adalah untuk memperoleh akses tidak sah ke data atau bahkan merusak database.

Malware dan Ransomware: Malware, termasuk ransomware, adalah perangkat lunak berbahaya yang dirancang untuk merusak atau mengunci data pada database. Ransomware, khususnya, mengenkripsi data dan meminta tebusan untuk membuka kunci data tersebut. *Serangan Denial of Service (DoS):* Serangan ini bertujuan untuk menghentikan ketersediaan database dengan membanjiri server database dengan trafik yang sangat tinggi, membuatnya tidak dapat diakses oleh pengguna sah. Untuk melindungi database dari ancaman eksternal, pengamanan jaringan seperti firewall, sistem deteksi intrusi (IDS), dan pembaruan rutin terhadap patch keamanan sangat diperlukan.

c. Ancaman Fisik

Ancaman fisik mengacu pada risiko yang dapat merusak perangkat keras tempat database disimpan, seperti:

- a) Pencurian perangkat keras: Seperti server atau disk yang berisi data sensitif.

- b) Bencana alam: Seperti kebakaran, banjir, atau gempa bumi yang dapat merusak server fisik tempat data disimpan.
- c) Kerusakan perangkat keras: Seperti kegagalan pada hard drive atau kerusakan komponen lain yang menyebabkan hilangnya data.
- d) Untuk mitigasi ancaman fisik, penting untuk memiliki backup data yang aman di lokasi terpisah atau menggunakan cloud storage yang dapat menyediakan perlindungan terhadap bencana fisik.

3. Prinsip-prinsip Keamanan dalam Database

Keamanan database dapat dibangun dengan mengikuti prinsip-prinsip dasar yang dikenal dengan singkatan CIA, yaitu *Confidentiality*, *Integrity*, dan *Availability*. Ketiga prinsip ini merupakan fondasi utama dari kebijakan keamanan database yang efektif.

1) Kerahasiaan (*Confidentiality*)

Kerahasiaan merujuk pada upaya untuk menjaga data agar hanya dapat diakses oleh pihak yang sah, yaitu mereka yang memiliki izin untuk melihat atau mengubah data tersebut. Beberapa langkah yang diterapkan untuk menjaga kerahasiaan data adalah:

- a) Enkripsi Data: Menggunakan algoritma enkripsi untuk memastikan bahwa data yang disimpan dalam database tidak dapat dibaca oleh pihak yang tidak berwenang, baik saat data berada dalam penyimpanan (*data-at-rest*) maupun saat data dikirim melalui jaringan (*data-in-transit*).
- b) Kontrol Akses: Pembatasan akses berdasarkan hak akses pengguna (*user privileges*). Setiap pengguna hanya diberikan akses ke data yang diperlukan sesuai dengan peran atau tugas mereka, menggunakan model kontrol akses berbasis peran (RBAC – *Role-Based Access Control*).
- c) Autentikasi dan Otorisasi: Menetapkan mekanisme autentikasi yang kuat, seperti *password* yang kuat, otentikasi dua faktor (2FA), dan token keamanan untuk memastikan hanya pengguna yang terautentikasi yang dapat mengakses database.

2) Integritas (*Integrity*)

Integritas mengacu pada pemeliharaan akurasi dan konsistensi data dalam database. Data harus bebas dari modifikasi yang tidak sah, baik oleh pengguna internal maupun eksternal. Langkah-langkah untuk menjaga integritas data antara lain:

- a) Penggunaan Checksum dan Hashing: Metode ini digunakan untuk memverifikasi bahwa data tidak telah dimodifikasi tanpa izin.

- b) Kontrol Transaksi: Menggunakan sistem transaksi yang menjamin konsistensi data, seperti prinsip ACID yang diterapkan dalam sistem basis data untuk memastikan bahwa transaksi berlangsung dengan benar dan data tetap konsisten setelah setiap transaksi.
- c) Audit Log: Mencatat dan memantau semua perubahan yang terjadi pada data di dalam database, sehingga jika ada perubahan yang tidak sah, dapat dengan mudah ditelusuri dan diidentifikasi.

3) Ketersediaan (*Availability*)

Ketersediaan memastikan bahwa data yang disimpan dalam database selalu dapat diakses oleh pengguna yang sah kapan pun dibutuhkan. Upaya untuk menjaga ketersediaan data antara lain:

- a) Redundansi dan Backup: Menyimpan salinan cadangan data di lokasi yang aman dan menggunakan RAID (*Redundant Array of Independent Disks*) untuk memastikan data tetap tersedia meskipun terjadi kegagalan perangkat keras.
- b) Pemulihan Bencana: Merencanakan dan mengimplementasikan solusi pemulihan bencana untuk memastikan database dapat dipulihkan dengan cepat setelah

terjadi kegagalan besar, seperti bencana alam atau serangan ransomware.

- c) Distribusi Beban dan Skalabilitas: Menggunakan sistem distribusi beban dan mekanisme skalabilitas untuk memastikan bahwa database tetap dapat melayani permintaan tinggi dari pengguna tanpa menurunkan kinerja.

4) Teknik dan Kebijakan Keamanan Database

Untuk melindungi database dari ancaman yang telah dijelaskan, berbagai teknik dan kebijakan dapat diterapkan, di antaranya:

a. Kontrol Akses

Kontrol akses adalah salah satu elemen kunci dalam keamanan database. Dengan kontrol akses yang baik, organisasi dapat memastikan bahwa hanya pengguna yang berwenang yang dapat mengakses data atau melakukan perubahan pada database. Beberapa teknik kontrol akses yang umum digunakan adalah:

a) *Role-Based Access Control* (RBAC)

Mengelompokkan pengguna berdasarkan peran mereka di dalam organisasi dan memberikan hak akses sesuai dengan kebutuhan mereka.

b) *Mandatory Access Control* (MAC)

Mengontrol akses berdasarkan kebijakan yang ditentukan oleh administrator, yang tidak dapat diubah oleh pengguna biasa.

c) *Discretionary Access Control* (DAC)

Memberikan kontrol akses kepada pemilik data atau objek, yang dapat menetapkan izin akses untuk pengguna lain.

b. Enkripsi Data

Enkripsi adalah salah satu teknik utama dalam menjaga kerahasiaan data. Data yang dienkripsi hanya dapat diakses oleh pihak yang memiliki kunci enkripsi yang sah. Enkripsi dapat diterapkan pada dua level utama:

a) Enkripsi data-at-rest: Melindungi data yang disimpan dalam database, memastikan bahwa data tetap aman meskipun perangkat keras dicuri atau diakses secara fisik.

b) Enkripsi data-in-transit: Melindungi data yang dikirim melalui jaringan, sehingga mencegah penyadapan atau modifikasi oleh pihak yang tidak berwenang.

c. Audit dan Monitoring

Audit dan monitoring adalah praktik untuk memantau aktivitas di dalam database guna mendeteksi ancaman atau perilaku mencurigakan. Aktivitas yang diawasi antara lain:

- a) *Login* dan *logout* pengguna: Mencatat siapa yang mengakses database, kapan, dan apa yang mereka lakukan.
- b) Perubahan pada data sensitif: Memantau perubahan yang dilakukan pada data kritis, seperti perubahan status transaksi atau informasi pelanggan.
- c) Peringatan keamanan: Mengatur sistem untuk memberikan peringatan ketika terdeteksi aktivitas yang mencurigakan atau tidak sah.

C. Hubungan Antara *Recovery* dan Keamanan Database

Keamanan database dan proses *recovery* adalah dua aspek penting dalam pengelolaan sistem database yang andal dan aman. Meskipun keduanya memiliki fokus yang berbeda keamanan berfokus pada perlindungan data dari ancaman eksternal dan internal, sedangkan *recovery* berfokus pada pemulihan data setelah terjadi kegagalan kedua hal ini sangat saling terkait dan

bekerja bersama untuk memastikan kelangsungan operasional database dan mencegah kerugian besar akibat hilangnya data atau pelanggaran keamanan. Sebagai contoh, ketika terjadi suatu kegagalan dalam database, seperti kerusakan perangkat keras atau serangan ransomware, tidak hanya data yang perlu dipulihkan, tetapi juga penting untuk memastikan bahwa data yang dipulihkan tetap aman dan tidak terpapar risiko. Dalam hal ini, mekanisme *recovery* yang efektif harus mempertimbangkan aspek keamanan agar tidak membuka celah yang dapat dimanfaatkan oleh pihak yang tidak berwenang.

Di sisi lain, kebijakan dan teknik keamanan yang diterapkan pada database juga memainkan peran penting dalam proses *recovery*. Jika data atau konfigurasi yang digunakan dalam pemulihan tidak terlindungi dengan baik, maka pemulihan bisa gagal atau bahkan memperburuk kondisi database. Oleh karena itu, pemahaman tentang bagaimana kedua aspek ini saling mendukung adalah krusial dalam menciptakan sistem database yang aman dan dapat diandalkan.

1. Menggabungkan Proses Pemulihan dengan Keamanan

Proses *recovery* yang melibatkan pemulihan data harus dilakukan dengan cara yang aman untuk memastikan bahwa data yang dipulihkan tidak terpapar risiko keamanan. Dalam hal ini, *recovery*

tidak hanya mencakup pemulihan data yang hilang atau rusak, tetapi juga memastikan bahwa data yang dipulihkan tidak terkontaminasi atau dimanipulasi oleh pihak yang tidak sah. Beberapa contoh hubungan antara *recovery* dan keamanan dalam konteks ini adalah:

a) Pemulihan Data dengan Enkripsi

Enkripsi adalah salah satu teknik utama yang digunakan dalam menjaga kerahasiaan data, baik saat data berada di disk (*data-at-rest*) maupun saat data sedang ditransfer melalui jaringan (*data-in-transit*). Namun, selama proses *recovery*, ada risiko bahwa data yang dipulihkan dari backup atau log transaksi dapat diakses oleh pihak yang tidak berwenang jika tidak dilindungi dengan baik. Untuk itu, ketika melakukan *recovery* menggunakan backup data, penting untuk memastikan bahwa data yang dipulihkan tetap terenskripsi dengan baik. Jika *recovery* dilakukan dengan menggunakan backup terenskripsi, maka selama pemulihan data, data tersebut tetap aman meskipun berada di lingkungan yang berisiko, seperti ketika menggunakan media yang tidak aman atau terhubung ke jaringan publik.

b) Penggunaan Log dan Audit dalam *Recovery*

Salah satu teknik yang digunakan dalam *recovery* adalah *log-based recovery*, yang mengandalkan log transaksi untuk

memulihkan perubahan data yang belum disimpan atau untuk membatalkan transaksi yang gagal. Namun, penggunaan log transaksi juga bisa menjadi celah keamanan jika log tersebut tidak dilindungi dengan baik. Log transaksi menyimpan informasi penting tentang setiap operasi yang dilakukan pada database, termasuk informasi yang sangat sensitif. Oleh karena itu, saat menggunakan log untuk recovery, penting untuk memastikan bahwa log tersebut dilindungi dengan enkripsi dan hanya dapat diakses oleh pengguna yang berwenang. Implementasi audit trail yang tepat dapat memberikan jejak rekam yang memungkinkan administrator untuk memantau penggunaan log transaksi dan mendeteksi upaya penyalahgunaan.

c) *Pengelolaan Akses Selama Recovery*

Selama proses recovery, akses ke database harus dibatasi dengan ketat untuk menghindari potensi manipulasi atau pencurian data. Kontrol akses yang ketat dan pemisahan tugas (*segregation of duties*) harus diterapkan selama proses pemulihan. Misalnya, hanya administrator atau pengguna dengan peran tertentu yang diperbolehkan untuk mengakses log transaksi atau backup yang digunakan dalam pemulihan. Selain itu, selama *recovery* berlangsung, ada kemungkinan sistem database akan dalam keadaan sementara tidak stabil

atau tidak sepenuhnya terlindungi, sehingga memberikan peluang bagi pihak yang tidak sah untuk mengeksploitasi celah keamanan. Oleh karena itu, kontrol akses yang kuat perlu dipastikan agar hanya pihak yang berwenang yang dapat melakukan pemulihan atau modifikasi terhadap data yang sedang dipulihkan.

2. Keamanan Data yang Diperoleh dari Proses *Recovery*

Selama proses *recovery*, bukan hanya data yang perlu dipulihkan, tetapi juga penting untuk memastikan bahwa data yang dipulihkan tetap aman dan tidak mudah diakses oleh pihak yang tidak berwenang. Salah satu tantangan dalam *recovery* adalah memastikan bahwa data yang dipulihkan tidak mengandung celah keamanan yang dapat dimanfaatkan oleh penyerang. Misalnya, dalam situasi serangan ransomware, penyerang mungkin telah menambahkan atau memodifikasi data dalam database. Oleh karena itu, dalam pemulihan data, kita harus memastikan bahwa data yang dipulihkan tidak mengandung perangkat lunak berbahaya atau perubahan yang tidak sah. Beberapa langkah untuk memastikan keamanan data selama *recovery* antara lain:

- a) Verifikasi keaslian *backup*: Sebelum memulihkan data dari backup, pastikan bahwa *backup* tersebut berasal dari sumber yang terpercaya dan tidak terinfeksi oleh perangkat lunak berbahaya.

- b) Pemulihan dengan *rollback*: Dalam beberapa kasus, teknik rollback dapat digunakan untuk mengembalikan database ke titik tertentu sebelum terjadi gangguan atau serangan. Ini penting untuk memastikan bahwa perubahan yang dilakukan oleh penyerang tidak dipulihkan.
- c) Pemeriksaan keamanan setelah *recovery*: Setelah proses *recovery* selesai, lakukan pemeriksaan keamanan yang menyeluruh untuk memastikan bahwa database tidak terpengaruh oleh serangan atau pelanggaran yang mungkin terjadi selama kegagalan.

3. Perlindungan Data Selama Proses Pemulihan

Proses recovery sering kali melibatkan penggunaan salinan backup atau log transaksi untuk mengembalikan data yang hilang atau rusak. Namun, salinan data tersebut sering kali berada di luar sistem database utama, yang berarti bahwa data tersebut berpotensi terpapar ancaman selama proses pemulihan. Untuk melindungi data selama proses pemulihan, beberapa langkah berikut perlu diperhatikan:

- a) Penyimpanan backup yang aman: Pastikan bahwa backup data disimpan di lokasi yang aman dan terpisah dari sistem utama, serta terenkripsi untuk melindungi data dari ancaman.

- b) Keamanan komunikasi data: Jika data harus dipulihkan melalui jaringan, pastikan bahwa data tersebut dienkripsi selama transfer untuk menghindari pencurian data atau penyadapan.
- c) Validasi data setelah *recovery*: Setelah data dipulihkan, lakukan validasi untuk memastikan bahwa data yang dipulihkan tidak hanya akurat, tetapi juga aman. Validasi ini dapat mencakup pengecekan terhadap konsistensi data dan pengujian terhadap potensi keberadaan malware atau data yang telah dimodifikasi.

DAFTAR PUSTAKA

- Himanshi. (2019). *Database recovery techniques in DBMS*. GeeksforGeeks. <https://www.geeksforgeeks.org/database-recovery-techniques-in-dbms/>
- Thakur, S. (2019, June 15). *Database recovery in DBMS and its techniques*. What is DBMS. <https://whatisdbms.com/database-recovery-in-dbms-and-its-techniques/>

BAB 13

ARSITEKTUR BASIS DATA

TERDISTRIBUSI

Oleh : Syahrial, S.Kom., M.Kom

A. Pengantar Basis Data Terdistribusi

Arsitektur basis data terdistribusi mengacu pada desain sistem basisdata di mana data tidak disimpan pada satu lokasi atau server, melainkan tersebar di beberapa lokasi atau node yang terhubung dalam jaringan. Meskipun data tersebar di berbagai lokasi fisik, sistem basis data ini berfungsi untuk memberikan transparansi, yaitu memungkinkan pengguna untuk mengakses data secara seragam tanpa memperhatikan lokasi fisik data tersebut. Tujuan utama dari arsitektur ini adalah untuk meningkatkan ketersediaan, skalabilitas, dan ketahanan sistem, sambil memastikan bahwa data tetap dapat diakses dengan cara yang transparan dan efisien, meskipun data tersebar di banyak lokasi. Sistem ini memungkinkan organisasi untuk menyebarkan data mereka secara geografis, sehingga meningkatkan

ketersediaan, skalabilitas, dan ketahanan terhadap kegagalan. Namun, hal ini juga menghadirkan tantangan tersendiri dalam hal koordinasi antar node, konsistensi data, dan manajemen transaksi.

Arsitektur ini juga menghadirkan tantangan terkait komunikasi antar node, pengelolaan konsistensi data, dan pengelolaan transaksi antar node yang terdistribusi. Oleh karena itu, desain arsitektur harus mempertimbangkan banyak faktor teknis, termasuk cara data dipartisi, disalin, dan dikelola di seluruh sistem.

B. Komponen Utama dalam Basis Data Terdistribusi

Sistem basis data terdistribusi terdiri dari beberapa komponen penting (Özsu & Valduriez, 2011):

- 1) **Nodes:** Komputer atau server yang menyimpan data dan menjalankan sistem manajemen basis data (DBMS). Setiap node memiliki salinan data atau bagian dari data yang terpisah, tergantung pada metode distribusi data yang diterapkan.
- 2) **Jaringan Komunikasi:** Menghubungkan node-node dalam sistem basis data terdistribusi. Jaringan ini memungkinkan pertukaran data dan koordinasi antar node untuk menjalankan transaksi dan query secara efisien.

- 3) Manajer Basis Data (DBMS): Setiap node dalam sistem terdistribusi memiliki manajer basis data sendiri yang mengatur bagaimana data disimpan, dikelola, dan diakses. Sistem ini memungkinkan pengelolaan basis data di banyak lokasi secara transparan kepada pengguna.

C. Jenis-Jenis Arsitektur Basis Data Terdistribusi

Arsitektur basis data terdistribusi dapat dibedakan menjadi beberapa jenis utama berdasarkan cara data dikelola dan diproses. Jenis-jenis arsitektur ini mempengaruhi bagaimana node saling berkomunikasi dan bagaimana data disebar. Arsitektur basis data terdistribusi bisa bervariasi, tergantung pada desain sistem dan kebutuhannya. Beberapa arsitektur yang umum digunakan antara lain (Ramakrishnan & Gehrke, 2003):

1. Arsitektur Homogen vs. Heterogen

Arsitektur Homogen: Semua node dalam sistem basis data terdistribusi menggunakan perangkat keras, perangkat lunak, dan sistem operasi yang serupa atau identik. Hal ini mempermudah integrasi dan pengelolaan karena tidak ada perbedaan teknis yang signifikan antara node.

- a) Keuntungan: Manajemen dan pengelolaan lebih mudah karena konsistensi perangkat keras dan perangkat lunak.

- b) Contoh: Sistem berbasis UNIX yang diimplementasikan dengan sistem manajemen basis data (DBMS) yang serupa.

Arsitektur Heterogen: Node dalam sistem ini menggunakan berbagai jenis perangkat keras, sistem operasi, dan DBMS yang berbeda. Sistem harus menangani perbedaan-perbedaan ini dengan cara yang transparan bagi pengguna.

- a) Keuntungan: Lebih fleksibel dan memungkinkan penggunaan perangkat keras dan perangkat lunak yang lebih beragam.
- b) Contoh: Implementasi basis data terdistribusi di berbagai pusat data yang menggunakan sistem operasi dan perangkat keras yang berbeda, tetapi tetap saling terhubung.

2. Arsitektur *Client-Server*

Dalam model ini, ada satu server pusat yang mengelola dan menyimpan data, sementara klien (*client*) mengirimkan permintaan untuk data tersebut. Server bertanggung jawab atas penyimpanan data, pemrosesan transaksi, dan pengelolaan basis data, sementara klien berperan sebagai pengakses data.

- a) Keuntungan: Mudah untuk dikelola, dan ada pemisahan yang jelas antara klien dan server.
- b) Kekurangan: Jika server pusat mengalami kegagalan, seluruh sistem bisa terhenti.

3. Arsitektur *Peer-to-Peer* (P2P)

Arsitektur ini mengandalkan interaksi langsung antar node tanpa adanya server pusat. Setiap node memiliki kemampuan untuk berfungsi sebagai klien dan server. Data tidak hanya disalin ke node lain tetapi juga diproses dan dikelola secara merata di seluruh sistem.

- a) Keuntungan: Lebih tahan terhadap kegagalan karena tidak ada titik pusat kegagalan.
- b) Kekurangan: Manajemen sistem lebih kompleks dan membutuhkan protokol koordinasi yang canggih.

4. Arsitektur Replikasi dan Fragmentasi Data

Replikasi Data: Data yang sama disalin ke beberapa node untuk meningkatkan ketersediaan dan keandalan. Replikasi bisa dilakukan secara sinkron (langsung disalin) atau asinkron (salinan terjadi setelah transaksi selesai).

- a) Keuntungan: Meningkatkan ketersediaan data meskipun ada kegagalan di salah satu node.
- b) Contoh: Sistem seperti Cassandra yang menggunakan replikasi untuk menjamin ketersediaan data tinggi.

Fragmentasi Data: Data dibagi menjadi fragmen-fragmen kecil dan didistribusikan di berbagai node. Ada dua jenis fragmentasi utama:

- a) Fragmentasi Horizontal: Memecah data menjadi baris-baris dan mendistribusikannya ke node yang berbeda.
- b) Fragmentasi Vertikal: Memecah data menjadi kolom-kolom dan mendistribusikannya ke node yang berbeda.

Keuntungan: Mengurangi beban pada satu node dan meningkatkan skalabilitas.

D. Desain Arsitektur Terdistribusi

1. Arsitektur Homogen vs Heterogen

Pada desain arsitektur homogen semua node dalam arsitektur basis data terdistribusi menggunakan perangkat keras dan perangkat lunak yang serupa. Hal ini memudahkan manajemen dan pemeliharaan karena standar teknisnya konsisten. Desain arsitektur heterogen: Setiap node menggunakan perangkat keras dan perangkat lunak yang berbeda, yang memerlukan lapisan perangkat lunak tambahan untuk menjamin komunikasi antar node yang berbeda. Pendekatan ini lebih fleksibel, tetapi juga lebih kompleks untuk dikelola (Özsu & Valduriez, 2011).

2. Arsitektur *Client-Server* vs *Peer-to-Peer*

Pada arsitektur *Client-Server* dibangun dengan membentuk server pusat yang mengelola basis data dan klien yang mengaksesnya. Server menjadi pusat untuk penyimpanan data

dan klien berinteraksi dengan data melalui server tersebut. Sedangkan pada arsitektur Peer-to-Peer dibangun setiap node dalam sistem berfungsi sebagai klien dan server sekaligus, sehingga menghilangkan ketergantungan pada satu server pusat. Model ini lebih skalabel dan memungkinkan distribusi beban yang lebih merata (Ramakrishnan & Gehrke, 2003).

3. Replikasi dan Fragmentasi Data

Desain arsitektur Replikasi struktur dibangun untuk memungkinkan data yang sama dapat disalin ke beberapa node untuk meningkatkan ketersediaan dan keandalan sistem. Replikasi ini bisa dilakukan secara sinkron (data diperbarui secara bersamaan di semua node) atau asinkron (data diperbarui di node lain setelah transaksi selesai di node utama). Pada arsitektur Fragmentasi desain dibentuk sehingga data dapat dibagi menjadi fragmen-fragmen dan disebar ke beberapa node. Fragmentasi dapat dilakukan secara horizontal (memecah data berdasarkan baris) atau vertikal (memecah data berdasarkan kolom).

4. Manajemen Transaksi Terdistribusi

Pada sistem basis data terdistribusi, transaksi sering kali melibatkan beberapa node. Berdasarkan hal ini, protokol manajemen transaksi seperti *Two-Phase Commit* (2PC) atau

Three-Phase Commit (3PC) diperlukan untuk memastikan bahwa transaksi dijalankan secara atomik di seluruh node (Kleppmann, 2017).

- a) *Two-Phase Commit* (2PC): Protokol ini memastikan bahwa transaksi akan dilaksanakan di seluruh sistem atau dibatalkan sepenuhnya jika salah satu node gagal mengonfirmasi.
- b) *Three-Phase Commit* (3PC): Protokol ini adalah pengembangan dari 2PC, dirancang untuk mengurangi kemungkinan terjadinya kegagalan yang tidak dapat dipulihkan.

Fragmentasi data harus dipilih berdasarkan pola akses data yang sering digunakan. Misalnya, jika ada query yang sering mengakses data berdasarkan atribut tertentu, fragmentasi vertikal pada kolom tersebut mungkin lebih efisien. Replikasi penting untuk menjaga ketersediaan data, terutama dalam sistem yang melayani aplikasi global yang membutuhkan akses cepat ke data dari berbagai lokasi. Namun, replikasi yang berlebihan dapat menyebabkan *overhead* dan masalah konsistensi.

Desain arsitektur basis data terdistribusi juga harus mempertimbangkan aspek keamanan dan manajemen. Keamanan dalam sistem terdistribusi lebih kompleks karena data berada di

banyak lokasi. Penggunaan enkripsi dan kontrol akses berbasis peran (RBAC) harus diterapkan di setiap node. Manajemen sistem juga lebih menantang karena harus mengawasi kinerja, melakukan pemeliharaan, dan menangani kegagalan sistem di berbagai node.

E. Implementasi Arsitektur Terdistribusi

Dalam implementasi arsitektur basis data terdistribusi, penting untuk memilih model replikasi yang sesuai. Misalnya, dalam sistem seperti Amazon DynamoDB, replikasi dilakukan secara asinkron untuk memberikan ketersediaan yang lebih tinggi, meskipun ini bisa menurunkan konsistensi dalam beberapa kasus. Sebaliknya, sistem seperti Google Spanner menggunakan replikasi sinkron untuk memastikan konsistensi yang tinggi di seluruh dunia, meskipun ini mempengaruhi latensi dan performa (Kleppmann, 2017).

1. Fragmentasi Data

Pengelolaan fragmentasi data yang tepat sangat penting dalam desain arsitektur basis data terdistribusi. Fragmentasi horizontal cocok untuk aplikasi yang sering mengakses data berdasarkan kriteria tertentu (misalnya, baris data), sementara fragmentasi vertikal lebih cocok untuk aplikasi yang membutuhkan akses cepat ke subset kolom data. Google Spanner, misalnya, menggabungkan

pendekatan fragmentasi horizontal dan vertikal untuk menyebarkan data secara efisien di seluruh dunia (Özsu & Valduriez, 2011).

2. Penanganan Kegagalan dan Ketahanan

Dalam implementasi sistem basis data terdistribusi, penanganan kegagalan adalah aspek yang sangat penting. Dengan menggunakan teknik replikasi dan redundansi, kegagalan satu atau beberapa node tidak akan mempengaruhi ketersediaan sistem secara keseluruhan. Selain itu, penggunaan teknik seperti *checkpointing* dan *log-based recovery* memungkinkan sistem untuk memulihkan data setelah kegagalan atau crash (Kleppmann, 2017). Studi kasus dalam Implementasi arsitektur terdistribusi dalam *Google Spanner* dan Amazon DynamoDB. *Google Spanner* adalah contoh sistem basis data terdistribusi yang sangat efisien, mendukung transaksi terdistribusi dengan konsistensi yang tinggi dan latensi rendah. *Google Spanner* menggunakan kombinasi replikasi sinkron dan fragmentasi data untuk mendukung aplikasi skala besar dan kebutuhan konsistensi global (Corbett et al., 2013).

Implementasi arsitektur terdistribusi pada Amazon DynamoDB merupakan basis data terdistribusi yang memanfaatkan replikasi asinkron dan fragmentasi otomatis.

DynamoDB lebih fokus pada ketersediaan dan skalabilitas, dengan sedikit penekanan pada konsistensi yang kuat. Hal ini membuatnya cocok untuk aplikasi web yang memerlukan latensi rendah dan ketersediaan tinggi (DeCandia et al., 2007).

DAFTAR PUSTAKA

- Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., ... Woodford, D. (2013). Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3).
<https://doi.org/10.1145/2491245>
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *Operating Systems Review (ACM)*, 205–220.
<https://doi.org/10.1145/1294261.1294281>
- Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media, Inc.
- Özsu, M. T., & Valduriez, P. (2011). *Principles of Distributed Database Systems* (3rd ed.). Springer.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems* (3rd ed.). McGraw-Hill Higher Education.

BIODATA PENULIS



Dr. Tertia Avini, S.Kom., M.Kom

Dr. Tertia Avini, S.Kom., M.Kom., lahir di Palembang, Sumatera Selatan, adalah seorang dosen, peneliti, dan penulis yang berdedikasi di bidang Teknik Informatika. Ia menyelesaikan pendidikan S1 pada tahun 1998 di Universitas Gunadarma, S2 pada tahun 2012 di Universitas Binadarma, dan S3 pada tahun 2023 di Universitas Sriwijaya. Fokus utama keilmuannya, yaitu *Decision Support System (DSS)*, *Data Science*, *Data Mining*, dan *Machine Learning*, *Business intelligence*

Sebagai dosen di Universitas Indo Global Mandiri, Dr. Tertia Avini telah mengajar sejak tahun 1999 hingga sekarang. Dalam kapasitasnya sebagai akademisi, Dr. Tertia Avini aktif dalam kegiatan penelitian dan pengabdian kepada masyarakat. Ia telah mempublikasikan berbagai artikel ilmiah di jurnal nasional maupun internasional. Salah satu karya penelitiannya yang paling diakui

adalah *Designing a Decision Support System Using the Weighted Performance Indicator Method*, yang membahas tentang sistem penilaian kinerja dan memberikan kontribusi signifikan dalam bidang *Decision Support System* (DSS). Selain itu, ia sering menjadi pembicara dalam seminar, lokakarya, dan konferensi tingkat nasional maupun internasional untuk berbagi wawasan dan hasil penelitian dengan komunitas akademik serta praktisi.

Selain mengajar dan meneliti, Dr. Tertia Avini juga memiliki minat besar dalam dunia penulisan. Ia telah menerbitkan tujuh buku sejak tahun 2017, yang tidak hanya digunakan sebagai bahan ajar, tetapi juga menjadi referensi bagi para peneliti dan praktisi di bidang terkait.

Di luar aktivitas akademik, Dr. Tertia Avini juga aktif dalam berbagai kegiatan pengabdian kepada masyarakat. Ia terlibat dalam program Pengabdian kepada Masyarakat (PKM) yang didanai oleh Hibah Ristekdikti dan Universitas Indo Global Mandiri. Selain itu, ia berperan sebagai pendamping bagi pelaku UMKM di Sumatera Selatan, khususnya dalam membantu pengembangan bisnis mereka melalui penerapan strategi Digital Marketing. Baginya, seorang akademisi memiliki tanggung jawab sosial untuk memberikan kontribusi nyata yang dapat memberdayakan dan meningkatkan kesejahteraan masyarakat.

Dengan semangat dan dedikasi yang tinggi, Dr. Tertia Avini terus berkomitmen untuk mencetak generasi penerus yang cerdas, kritis, dan berkontribusi bagi kemajuan masyarakat. Baginya, menjadi dosen dan penulis adalah panggilan hidup yang memberikan kesempatan untuk berbagi ilmu sekaligus menciptakan dampak bermakna bagi masa depan.

---000---

BIODATA PENULIS

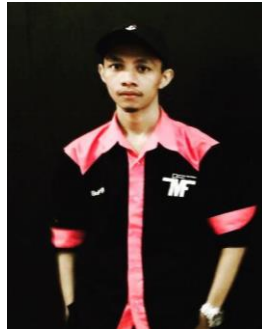


Yudi Herdiana, S.T., M.T.

Penulis adalah seorang Dosen Program Studi Teknik Informatika Universitas Bale Bandung Kabupaten Bandung Provinsi Jawa Barat Indonesia. Penulis lahir di Bandung tanggal 28 Februari 1975. Penulis adalah dosen tetap pada Program Studi Teknik Informatika Fakultas Teknologi Informasi Universitas Bale Bandung. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Jurusan Teknik Informatika. Penulis dapat dihubungi melalui e-mail: yudiherdiana@unibba.ac.id

---000---

BIODATA PENULIS



Agung Yuliyanto Nugroho S.Kom., M.Kom., M.Par

Penulis memiliki kepakaran dibidang Web *Technology*, Data *Science* dan Kepariwisata. Dan untuk mewujudkan karir sebagai dosen profesional, penulis pun aktif sebagai peneliti dibidang kepakarannya tersebut. Beberapa penelitian yang telah dilakukan didanai oleh internal perguruan tinggi dan juga Kemenristek DIKTI. Selain peneliti, penulis juga aktif menulis buku dengan harapan dapat memberikan kontribusi positif bagi bangsa dan negara yang sangat tercinta ini. Atas dedikasi dan kerja keras dalam membuat suatu karya, Republik Indonesia Kementerian Hukum Dan Hak Asasi Manusia sudah mencatat ada kurang lebih 100 karya yang sudah tercatat di surat pencatatan ciptaan sebagai salah satu kontribusi dalam melindungi hak kekayaan intelektual. Email Penulis agungboiler11@gmail.com

---000---

BIODATA PENULIS



Dr. Mohamad Ilyas Abas, S.SI, M.Kom

Penulis lahir di Gorontalo, 26 Agustus 1991. Penulis adalah dosen tetap pada Program Studi Ilmu Komputer Universitas Muhammadiyah Gorontalo. Menyelesaikan pendidikan S1 pada Prodi Sistem Informasi Universitas Negeri Gorontalo, S2 pada Jurusan Teknik Informatika Universitas Dian Nuswantoro dan alhamdulillah telah menyelesaikan S3 di Teknik Elektro konsentrasi Informatika di Universitas Hasanuddin. Penulis juga cukup aktif sebagai periset beberapa penelitian berhubungan data mining sistem informasi dan pengolahan citra. Penulis dapat dihubungi melalui email: ilyasabas26@gmail.com

---000---

BIODATA PENULIS



Tri Santoso, S.Kom, M.Kom

Dosen Program Studi Sistem Informasi Fakultas Teknologi Informasi Universitas Nusa Mandiri. Penulis lahir di Jakarta tanggal 21 April 1990. Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Teknologi Informasi Universitas Nusa Mandiri. Menyelesaikan pendidikan S1 pada Sistem Informasi dan melanjutkan S2 pada Jurusan Ilmu Komputer.

Penulis dapat dihubungi melalui e-mail: tri.tos@nusamandiri.com

---000---

BIODATA PENULIS



Ir. Hilmansyah Gani, S.Kom., M.Kom

Dosen Program Studi Sistem Informasi Fakultas Sains dan Ilmu Komputer Universitas Muhammadiyah Gorontalo. Penulis lahir di Gorontalo tanggal 11 Mei 1985. Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Sains dan Ilmu Komputer Universitas Muhammadiyah Gorontalo. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Jurusan Sistem Informasi menekuni Analisis Desain Sistem Informasi dan Basis Data. Penulis dapat dihubungi melalui e-mail: hilmansyahgani@umgo.ac.id

---000---

BIODATA PENULIS



Sulistianto Sutrisno Wanda, A.MI, S.Pd, M.M, M.Kom.,

Penulis adalah seorang mantan karyawan pabrik yang kini berkiprah sebagai pendidik. Beliau memiliki latar belakang pendidikan di bidang teknik dan komputer, dengan gelar DIV/S1 di Teknik Manajemen Industri dan Pendidikan Teknik Sipil, serta S2 di Manajemen Sistem Informasi dan Ilmu Komputer. Selain itu, beliau telah mengikuti berbagai pelatihan di dalam dan luar negeri, termasuk di bidang Elektronika, Teknisi Komputer, Pemrograman, Manajemen, Training dan Implementasi Sistem, Chef Kuliner, serta Cloud Computing.

Pengalaman kerja yang dimiliki sangat beragam, mencakup berbagai sektor industri. Dalam industri manufaktur, beliau pernah bekerja di bidang perencanaan, pergudangan, logistik, pengadaan, dan pembelian dengan tanggung jawab yang meliputi administrasi

data, perencanaan produksi, manajemen suplai material, serta implementasi sistem ERP (SAP system). Selain itu, beliau juga memiliki pengalaman dalam sektor perdagangan, khususnya dalam penyediaan kebutuhan pengolahan limbah industri dan pengembangan bisnis di tingkat nasional maupun internasional.

Di sektor konstruksi, beliau terlibat dalam pengawasan proyek, administrasi, dan pelaksanaan konstruksi. Pengalaman di pemerintahan meliputi pendampingan wirausaha, administrasi, serta perdagangan. Saat ini, beliau aktif sebagai dosen di Universitas Nusa Mandiri (*homebase*), Universitas BSI, dan Fakultas Teknik Universitas Krisnadwipayana, dengan tugas sebagai penasihat akademik, pembimbing, serta penguji praktik kerja lapangan dan skripsi.

Selain mengajar, beliau juga menjadi tutor dalam bimbingan belajar bagi calon ASN dan PPPK bidang Pranata Komputer. Beliau telah berkontribusi sebagai penulis dalam beberapa buku ajar dan buku referensi secara kolaboratif, serta aktif dalam konsultasi di bidang kewirausahaan dan industri. Penulis dapat dihubungi melalui e-mail: sulistianto68@gmail.com.

---000---

BIODATA PENULIS



Sutiyono, S.T., M.Kom

Sutiyono, S.T., M.Kom., lahir di Banyumas pada 27 Maret 1982, adalah seorang akademisi dan praktisi teknologi informasi yang berdomisili di Baleendah. Beliau menyelesaikan pendidikan sarjana di bidang Teknik Industri dari Universitas Islam Bandung, kemudian melanjutkan studi magister pada bidang Sistem Informasi di Universitas Komputer Indonesia. Saat ini, Sutiyono menjabat sebagai dosen tetap di Universitas Bale Bandung, sekaligus memegang posisi strategis sebagai Kepala Laboratorium Fakultas Teknologi Informasi di universitas yang sama. Dengan keahlian di bidang Basis Data, Jaringan Komputer, dan Data Science, beliau aktif berkontribusi dalam pengembangan akademik dan teknologi di institusinya.

---000---

BIODATA PENULIS



Widya Eka Pranata, S.Si., M.Si

Dosen Program Studi Sarjana Ilmu Komputer Universitas Muhammadiyah Gorontalo. Aktif mengajar pada mata kuliah basis data, data science dan pembelajaran mesin. Penulis lahir di Gorontalo tanggal 26 November 1998. Penulis adalah dosen tetap pada Program Studi Sarjana Ilmu Komputer Fakultas Sains dan Ilmu Komputer Universitas Muhammadiyah Gorontalo. Menyelesaikan pendidikan S1 pada Program Studi Matematika di Universitas Negeri Gorontalo dan melanjutkan S2 pada Program Studi Sains Komputasi menekuni bidang analisis data dan pembelajaran mesin. Penulis dapat dihubungi melalui e-mail: widyapranata@umgo.ac.id

---000---

BIODATA PENULIS



Rizal Lamusu., S.Kom.,MT

Penulis seorang Dosen Program Studi Ilmu Komputer Universitas Muhammadiyah Gorontalo. Penulis lahir di Gorontalo tanggal 29 April 1981. Penulis adalah dosen tetap pada Program Studi Ilmu Komputer Fakultas Sains Dan Ilmu Komputer, Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Program Studi teknik Elektro, Bidang Ilmu Teknik Informatika. Penulis dapat dihubungi melalui e-mail : rizal_lamusu@umgo.ac.id

---000---

BIODATA PENULIS



Muhamad Ikmal Wiawan, M.Kom. IPP

Penulis adalah seorang dosen tetap di Program Studi Teknik Informatika Universitas Islam Nusantara (UNINUS), yang mulai mengajar sejak tahun 2020 hingga saat ini. Beliau lahir di Bandung pada 29 Agustus 1945 dan memiliki latar belakang pendidikan yang kuat di bidang teknik informatika. Penulis menyelesaikan pendidikan D4 di Politeknik Piksi Ganesha Bandung pada tahun 2015, kemudian melanjutkan studi ke jenjang Magister Teknik Informatika di Universitas Langlangbuana dan meraih gelar Magister Komputer pada tahun 2017. Pada tahun 2022, beliau mendapatkan sertifikat Insinyur Profesional (IPP), yang semakin memperkuat kompetensinya di bidangnya.

Selain sebagai akademisi, Muhamad Ikmal Wiawan juga memiliki berbagai sertifikasi profesional yang menunjukkan keahlian dan kredibilitasnya dalam bidang teknologi informasi dan

keamanan siber. Beberapa sertifikat yang telah diraihnyanya antara lain *Certified Ethical Hacker* dari EC Council, Keamanan Siber dari BNSP, ISO 27001 *Lead Auditor*, serta sejumlah sertifikasi lainnya seperti *Network Administrator*, *Oracle Database*, *SQL Database*, *Network Security*, dan SCRUM. Sertifikasi ini menjadikannya seorang profesional yang diakui baik secara nasional maupun internasional.

Di luar dunia akademik, penulis juga berperan aktif dalam berbagai proyek strategis nasional. Ia turut berkontribusi dalam pengembangan Sistem Informasi Pemerintahan Daerah (SIPD) yang dikelola oleh Kementerian Dalam Negeri dan terlibat dalam kajian tentang Government Internet Exchange yang sedang dikaji bersama Kementerian Komunikasi dan Informatika. Dengan kombinasi pengalaman sebagai pengajar dan profesional di bidang teknologi informasi, Muhamad Ikmal Wiawan berkomitmen untuk memberikan kontribusi signifikan bagi kemajuan teknologi informasi dan pengembangan produk akademik di Indonesia. Penulis dapat dihubungi melalui e-mail di: ikmal.wiawan@gmail.com.

---000---

BIODATA PENULIS



Irawan Ibrahim, S.T., M.Kom

Penulis seorang dosen program studi Ilmu Komputer Universitas Muhammadiyah Gorontalo. Aktif mengajar pada mata kuliah Sistem Digital dan Sistem Tertanam dan Robotika.

Penulis lahir di Kabila tanggal 11 April 1983. Penulis adalah dosen tetap pada Program Studi Ilmu Komputer Fakultas Sains dan Ilmu Komputer Universitas Muhammadiyah Gorontalo. Menyelesaikan pendidikan S1 pada Jurusan Elektro dan melanjutkan S2 pada Jurusan Teknik Informatika menekuni bidang Ilmu Komputer dan Internet of Things. Penulis dapat dihubungi melalui e-mail: irawan_ibrahim@umgo.ac.id

---000---

BIODATA PENULIS



Syahrial, S.Kom., M.Kom

Penulis adalah seorang Dosen Program Studi Ilmu Komputer Universitas Muhammadiyah Gorontalo. Aktif mengajar pada mata kuliah basis data dan pembelajaran mesin. Penulis lahir di Barru tanggal 18 Agustus 1981. Penulis adalah dosen tetap pada Program Studi Ilmu Komputer Fakultas Sains dan Ilmu Komputer Universitas Muhammadiyah Gorontalo. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Jurusan Teknik Informatika menekuni bidang analisis data dan pembelajaran mesin. Penulis dapat dihubungi melalui e-mail: syahrialpaola@gmail.com

---000---

BASIS DATA

Di dunia yang semakin terdigitalisasi, pengelolaan data yang efisien dan aman menjadi kunci utama kesuksesan bisnis dan inovasi teknologi. "Basis Data" adalah buku panduan lengkap yang akan membekali Anda dengan pemahaman mendalam tentang bagaimana data disusun, diolah, dan dimanfaatkan secara optimal.

Buku ini tidak hanya menjelaskan konsep dasar basis data, tetapi juga mengupas strategi pemodelan, teknik optimasi, serta implementasi sistem basis data modern. Anda akan diajak memahami berbagai jenis database, bahasa SQL, serta teknologi terkini seperti kecerdasan buatan (AI), blockchain, dan big data dalam dunia basis data.

Ditulis oleh para pakar di bidangnya, buku ini menyajikan teori yang kuat dengan pendekatan praktis, menjadikannya referensi ideal bagi mahasiswa, profesional IT, dan siapa saja yang ingin menguasai dunia data. Jika Anda ingin meningkatkan keterampilan dalam manajemen basis data dan siap menghadapi revolusi digital, buku ini adalah langkah awal yang tepat!

