SISTEM BASIS DATA

MISWAR PAPUANGAN
MUDAR SAFI
YANI SUGIYANI
ARIE QUR'ANIA
SUMANTO

CV. Keranjang Teknologi Media

Sistem Basis Data

DITULIS OLEH

Miswar Papuangan
Mudar Safi
Yani Sugiyani
Arie Qurania
Sumanto

2023

CV. KERANJANG TEKNOLOGI MEDIA

SISTEM BASIS DATA

Penulis:

Miswar Papuangan
Mudar Safi
Yani Sugiyani
Arie Qurania
Sumanto

Editor:
Dyah Ayu Megawaty

Cetakan Pertama:
Bandarlampung, November 2023

ISBN: 978-623-09-6519-7 (PDF)

Penerbit:

CV. KERANJANG TEKNOLOGI MEDIA

PRAKATA

Puji Syukur kehadirat Tuhan Yang Maha Esa, atas limpahan rahmat-Nya sehingga Sistem Basis Data dapat terselesaikan dengan baik. Buku ini disusun berdasarkan pada proses pembelajaran yang lebih menempatkan mahasiswa sebagai pusat kegiatan belajar (*Student- Centered Learning*).

Kami menyadari bahwa dalam penyusunan Buku ini masih banyak kekurangan. Oleh karena itu, kami sangat mengharapkan kritik dan saran dari para pembaca demi perbaikan dan kesempurnaan Buku ini. Tak lupa, kami mengucapkan terima kasih kepada berbagai pihak yang telah membantu proses penyelesaian Buku ini. Semoga Sistem Basis Data ini dapat bermanfaat bagi kita semua, khususnya bagi dunia pendidikan.

Bandarlampung, November 2023

Penulis

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENGENALAN SISTEM BASIS DATA

Sistem basis data adalah komponen integral dalam dunia teknologi informasi yang memainkan peran kunci dalam pengelolaan, penyimpanan, dan akses data dalam berbagai konteks. Sebagai fondasi bagi beragam aplikasi, mulai dari sistem perbankan hingga media sosial, sistem basis data memungkinkan organisasi dan individu untuk mengorganisir, menyimpan, dan memanfaatkan data dengan efisien. Pengenalan sistem basis data mencakup pemahaman tentang konsep dasar seperti entitas, relasi, dan bahasa *querying*, serta teknologi yang berkembang seperti basis data relasional, basis data *NoSQL*, dan berbagai model penyimpanan data. Pemahaman mendalam tentang sistem basis data adalah kunci dalam mengoptimalkan penggunaan data dalam berbagai aspek kehidupan kita, dan ini merupakan landasan penting dalam dunia digital yang terus berkembang. Dalam era di mana data menjadi aset berharga, pemahaman tentang sistem basis data juga memiliki implikasi bisnis yang signifikan. Perusahaan dapat menggunakannya untuk mendukung pengambilan keputusan yang berdasar pada data, meningkatkan efisiensi operasional, dan memberikan pengalaman pelanggan yang lebih personal. Selain itu, dalam konteks teknologi yang terus berkembang, seperti kecerdasan buatan (AI) dan analitik data, sistem basis data menjadi semakin penting untuk mendukung inovasi dan perkembangan teknologi. Oleh karena itu, pengenalan sistem basis data adalah langkah awal yang penting dalam memahami fondasi teknologi informasi yang mendasari dunia digital saat ini, dan menjadi pengetahuan yang sangat berharga untuk siapa saja yang terlibat dalam bidang ini.

1.1. PENGANTAR SISTEM BASIS DATA

Sistem Basis Data adalah fondasi penting dalam dunia komputasi modern, memainkan peran kunci dalam mengelola dan menyimpan informasi secara efisien. Dalam konteks teknologi informasi, sistem basis data memungkinkan pengguna untuk menyimpan, mengambil, dan mengelola data dengan cara yang terstruktur, aman, dan mudah diakses. Sistem ini membantu organisasi dalam berbagai aspek, mulai dari manajemen inventaris hingga pelacakan informasi pelanggan, dan memainkan peran sentral dalam mendukung aplikasi berbasis data seperti situs web, aplikasi bisnis, dan banyak lagi. Pemahaman yang baik tentang konsep-konsep dasar sistem basis data, seperti tabel, kunci, relasi, dan bahasa query, sangat penting dalam mengelola informasi dengan efektif dalam lingkungan digital saat ini(Hellerstein et al., 2007). Sistem Basis Data juga memiliki peran vital dalam memastikan integritas data, konsistensi, dan keamanan, dengan mengimplementasikan mekanisme otentikasi dan otorisasi. Manajemen basis data, termasuk perencanaan, desain, implementasi, dan pemeliharaan, menjadi aspek penting dalam siklus hidup data. Selain itu, perkembangan teknologi dalam bidang basis data seperti basis data berdistribusi, basis data bergerak, dan komputasi awan semakin memperluas cakupan dan kompleksitas sistem basis data. Dengan demikian, pemahaman yang mendalam tentang sistem basis data menjadi kunci dalam dunia teknologi informasi modern, memungkinkan organisasi untuk mengoptimalkan penggunaan data mereka, mengambil keputusan yang lebih baik, dan merespons perubahan pasar dengan lebih cepat.

Sistem Basis Data merupakan fondasi integral dalam dunia komputasi modern, berfungsi sebagai struktur penyimpanan yang terstruktur untuk mengelola data dengan

efisien. Dalam era informasi yang semakin berkembang, sistem ini memainkan peran sentral dalam mengorganisir, menyimpan, dan mengakses informasi dengan cara yang aman dan terstruktur. Sistem Basis Data memungkinkan organisasi dan individu untuk mengelola volume data yang besar dengan lebih efisien dan memberikan manfaat yang signifikan dalam pengambilan keputusan, analisis, dan pelacakan informasi. Dalam pengantar ini, kita akan menjelajahi konsep dasar sistem basis data, komponen utama, bahasa query, serta peran pentingnya dalam lingkungan teknologi informasi saat ini. Sistem basis data mendukung aspek-aspek krusial dalam manajemen informasi, seperti perencanaan basis data, desain yang efisien, implementasi, dan pemeliharaan. Selain itu, kita akan menjelajahi pentingnya keamanan dan integritas data dalam sistem basis data, termasuk perlindungan data, pengendalian otorisasi, serta manajemen konflik dan kunci. Dalam era terbaru, kita akan membahas peran sistem basis data dalam mengatasi tantangan data besar (Big Data), Internet of Things (IoT), dan bagaimana teknologi terkini seperti kecerdasan buatan (AI) semakin mengubah cara kita berinteraksi dengan basis data. Pemahaman tentang sistem basis data merupakan komponen kunci dalam menghadapi era digital yang terus berubah. Dengan evolusi teknologi yang cepat, sistem basis data tidak hanya bertanggung jawab untuk menyimpan dan mengelola data, tetapi juga untuk memungkinkan inovasi, kecerdasan buatan, analitik data yang mendalam, dan lebih banyak lagi. Melalui pemahaman yang kokoh tentang sistem basis data, kita dapat mengoptimalkan cara kita mengumpulkan, memanfaatkan, dan mengurai data yang menjadi aset berharga dalam dunia yang semakin terhubung. Dalam pengantar ini, kita akan membahas dasar-dasar dan konsep utama, menciptakan fondasi yang kuat untuk menjelajahi peran sistem basis data

dalam revolusi digital yang sedang berlangsung dan memahami potensinya dalam mengubah dunia.

Tujuan utama dari Sistem Basis Data adalah menyediakan sebuah pendekatan yang terstruktur dan efisien untuk mengelola, menyimpan, mengambil, dan memelihara data. Sistem basis data dirancang untuk memastikan integritas data, konsistensi, dan keamanan, sehingga memungkinkan pengguna untuk mengakses informasi dengan mudah dan akurat. Tujuan lainnya adalah untuk meminimalkan redundansi data, memfasilitasi akses bersama dan berbagi data, serta mendukung pengambilan keputusan yang lebih baik melalui pemrosesan data yang lebih cepat dan analisis yang lebih mendalam. Selain itu, sistem basis data juga bertujuan untuk menyediakan tingkat abstraksi yang memungkinkan pengguna untuk berinteraksi dengan data tanpa harus terlibat dalam detail teknis penyimpanan dan pengolahan data yang kompleks. Dengan demikian, sistem basis data memainkan peran sentral dalam mendukung operasi bisnis, penelitian, dan berbagai aplikasi di berbagai sektor. Selain itu Sistem basis data juga bertujuan untuk memungkinkan skalabilitas, artinya sistem dapat tumbuh sejalan dengan kebutuhan organisasi atau aplikasi tanpa harus mengalami perubahan fundamental. Selain itu, mereka memfasilitasi manajemen data yang efisien, sehingga pengguna dapat dengan mudah mencari, memperbarui, dan menghapus data sesuai kebutuhan.

Dengan tujuan ini, sistem basis data membantu mengurangi kerumitan dalam manajemen informasi dan memungkinkan organisasi untuk merespons perubahan pasar, kebutuhan pelanggan, atau tujuan bisnis dengan lebih fleksibel dan efektif. Secara keseluruhan, sistem basis data bertujuan untuk memastikan bahwa data menjadi sumber daya yang dapat diandalkan dan berharga bagi organisasi, dan memberikan kerangka kerja yang kokoh untuk

mengelola aset informasi yang krusial dalam lingkungan teknologi modern. Salah satu tujuan penting sistem basis data adalah meningkatkan efisiensi dan produktivitas. Dengan menyediakan mekanisme otomatis untuk mengelola data seperti pencarian, penyimpanan, dan penyortiran, sistem basis data membantu menghemat waktu dan sumber daya. Ini memungkinkan pengguna untuk mengakses dan memanfaatkan data dengan lebih cepat, yang pada gilirannya dapat meningkatkan produktivitas dan daya saing organisasi. Sistem basis data juga memungkinkan pengembangan aplikasi lebih mudah dan cepat, karena mereka menyediakan lingkungan yang siap pakai untuk berinteraksi dengan data. Dengan tujuan ini, sistem basis data tidak hanya berperan sebagai penyimpan data, tetapi juga sebagai alat untuk mempercepat proses bisnis dan inovasi teknologi, serta menghadirkan nilai tambah bagi organisasi di era digital. Sistem basis data juga memiliki tujuan dalam mengelola data dengan akurat dan konsisten. Mereka membantu dalam mencegah kesalahan-kesalahan yang dapat terjadi saat data dikelola secara manual atau menggunakan sistem file tradisional. Dengan konsistensi dan integritas yang dijaga dengan ketat, sistem basis data membantu meminimalkan risiko kesalahan manusia dan menjaga keandalan data. Selain itu, mereka juga dapat memberikan tingkat keamanan yang tinggi untuk melindungi data sensitif dari akses yang tidak sah. Secara keseluruhan, tujuan sistem basis data adalah menyediakan alat yang dapat dipercaya, efisien, dan aman untuk mengelola data, yang merupakan aset paling berharga dalam lingkungan informasi modern.

Ruang lingkup Sistem Basis Data mencakup berbagai aspek terkait dengan pengelolaan dan penggunaan data dalam konteks teknologi informasi. Ini meliputi desain dan struktur basis data, implementasi sistem manajemen basis

data (DBMS), perencanaan dan pemeliharaan basis data, serta pengembangan aplikasi yang memanfaatkan data tersebut. Ruang lingkup juga mencakup bahasa query dan akses data, termasuk penggunaan SQL untuk mengambil, memperbarui, dan mengolah informasi. Aspek keamanan dan integritas data menjadi fokus penting dalam ruang lingkup ini, termasuk perlindungan data sensitif dan manajemen hak akses. Selain itu, sistem basis data juga mempertimbangkan isu-isu terkait dengan skala dan kinerja, termasuk basis data terdistribusi dan implementasi di lingkungan komputasi awan. Dengan demikian, ruang lingkup sistem basis data sangat luas dan penting dalam menyediakan fondasi yang kokoh untuk pengelolaan data yang efisien dan andal dalam berbagai aplikasi dan organisasi. Ruang lingkup Sistem Basis Data juga mencakup konsep dasar seperti entitas, atribut, hubungan, kunci primer, dan kunci asing yang digunakan untuk merancang struktur data yang tepat. Normalisasi basis data dan model data seperti model relasional juga termasuk dalam ruang lingkup ini untuk memastikan bahwa data tersimpan dengan cara yang efisien dan tidak redundan. Selain itu, pengelolaan transaksi dan kontrol konkurensi adalah bagian penting dari ruang lingkup ini, yang memungkinkan data tetap konsisten saat digunakan oleh banyak pengguna secara bersamaan. Dalam dunia yang semakin terhubung dan bergerak, ruang lingkup Sistem Basis Data juga memperluas diri ke basis data bergerak dan kemampuan akses data dari berbagai perangkat. Kesimpulannya, ruang lingkup Sistem Basis Data sangat luas dan mencakup banyak aspek yang diperlukan untuk mengelola dan memanfaatkan data secara efisien dalam berbagai konteks.

1.2. PENTINGNYA SISTEM BASIS DATA

Pentingnya Sistem Basis Data dalam dunia modern tidak dapat dilebih-lebihkan. Sistem Basis Data memainkan peran sentral dalam mengelola, menyimpan, dan mengakses data secara efisien dan terstruktur, yang menjadi aset yang sangat berharga dalam berbagai organisasi dan aplikasi. Mereka memastikan integritas, konsistensi, dan keamanan data, dan memfasilitasi pengambilan keputusan yang lebih baik melalui analisis data yang mendalam. Dengan penggunaan sistem basis data yang tepat, organisasi dapat meningkatkan produktivitas, mengurangi redundansi data, dan merespons perubahan pasar dengan lebih cepat. Dalam era informasi yang semakin berkembang, pemahaman dan penerapan sistem basis data menjadi kunci untuk mencapai efisiensi, inovasi, dan daya saing yang tinggi. Sistem Basis Data juga mendukung pengembangan aplikasi yang lebih kompleks dan fleksibel, memungkinkan organisasi untuk mengotomatiskan proses bisnis, menghasilkan laporan yang akurat, dan menyediakan layanan berbasis data yang unggul. Mereka juga memberikan fondasi yang kokoh untuk teknologi terkini seperti kecerdasan buatan (AI), analitik data, dan internet of things (IoT), yang semuanya bergantung pada akses yang efisien dan andal ke data. Selain itu, dalam konteks privasi dan keamanan, sistem basis data memainkan peran kunci dalam melindungi data sensitif dan memastikan bahwa hanya pengguna yang berwenang yang dapat mengaksesnya. Dengan demikian, pentingnya sistem basis data membantu organisasi untuk mengelola dan memanfaatkan data mereka dengan cara yang lebih efektif, meningkatkan daya saing, dan meraih kesuksesan dalam dunia yang semakin terhubung dan kompetitif.

1.3. KOMPONEN UTAMA SISTEM BASIS DATA

Dalam dunia teknologi informasi yang semakin berkembang, pemahaman tentang komponen utama dalam Sistem Basis Data adalah kunci untuk mengelola dan memanfaatkan data dengan efisien. Sistem Basis Data, sebagai fondasi penyimpanan data yang terstruktur, memerlukan komponen inti untuk menjalankan berbagai tugas, mulai dari manajemen dan penyimpanan data hingga penyediaan antarmuka untuk aplikasi dan pengguna akhir. Komponen utama, seperti Sistem Manajemen Basis Data (DBMS), klien aplikasi, bahasa *query*, dan data *dictionary*, bekerja bersama untuk menciptakan infrastruktur yang kokoh dan andal bagi data, yang menjadi aset penting dalam berbagai organisasi dan aplikasi(Huang et al., 2009). Dalam panduan ini, kita akan menjelajahi dengan lebih mendalam masing-masing komponen ini dan bagaimana mereka berperan dalam mengoptimalkan pengelolaan data dalam lingkungan sistem basis data yang kompleks. Ketika kita menjelajahi masing-masing komponen utama dalam Sistem Basis Data, kita akan menemukan bagaimana Sistem Manajemen Basis Data (DBMS) bertanggung jawab atas manajemen, penyimpanan, dan akses data. DBMS menyediakan infrastruktur yang diperlukan untuk menjalankan operasi basis data, seperti menyimpan, mengambil, memperbarui, dan mengamankan data. Klien aplikasi, sebagai pengguna akhir atau aplikasi bisnis, berinteraksi dengan DBMS melalui antarmuka yang disediakan, yang memungkinkan mereka untuk mengakses data dengan cara yang terstruktur. Bahasa *query*, seperti SQL, digunakan untuk membuat permintaan data dan mengelola informasi dalam basis data. Data *dictionary* mencatat metadata yang mendokumentasikan struktur basis data, membantu dalam administrasi dan pengembangan

sistem. Dengan pemahaman mendalam tentang komponen utama ini, kita dapat merancang dan mengelola sistem basis data dengan lebih efektif, mengoptimalkan penggunaan data, dan mendukung operasi bisnis serta inovasi teknologi. Komponen utama dalam Sistem Basis Data mencakup:

A. Sistem Manajemen Basis Data (DBMS): Ini adalah perangkat lunak yang bertanggung jawab atas manajemen dan pengelolaan basis data. DBMS digunakan untuk membuat, mengakses, mengelola, dan memelihara data dalam basis data. Beberapa contoh DBMS yang terkenal adalah *MySQL*, *Oracle*, *SQL Server*, dan *PostgreSQL*.
B. Klien Aplikasi: Klien aplikasi adalah perangkat lunak atau aplikasi yang digunakan oleh pengguna akhir atau aplikasi bisnis untuk berinteraksi dengan basis data. Mereka mengirimkan permintaan data ke DBMS dan menerima hasilnya. Klien aplikasi dapat berupa aplikasi desktop, aplikasi web, atau perangkat lunak lain yang membutuhkan akses ke data.
C. Bahasa *Query*: Bahasa *query*, seperti SQL (*Structured Query Language*), digunakan untuk mengakses, mengubah, dan mengelola data dalam basis data. Pengguna dan aplikasi menggunakan bahasa query untuk membuat pertanyaan atau instruksi yang akan dieksekusi oleh DBMS.
D. Data *Dictionary*: Data *dictionary* adalah kumpulan *metadata* yang berisi informasi tentang struktur basis data, seperti tabel, kolom, indeks, kunci, dan hak akses. Ini membantu dalam dokumentasi basis data dan menyediakan panduan tentang bagaimana data disimpan dan diakses.
E. Tabel: Tabel adalah entitas dasar dalam basis data yang digunakan untuk mengorganisasi data. Mereka terdiri dari baris dan kolom yang berisi informasi terkait.

Setiap tabel memiliki atribut yang mendefinisikan jenis data yang dapat disimpan dalam tabel, serta kunci yang digunakan untuk mengidentifikasi setiap baris secara unik.

F. Indeks: Indeks adalah struktur yang digunakan untuk meningkatkan kecepatan pencarian data dalam basis data. Mereka membantu mempercepat operasi pencarian dengan memberikan akses langsung ke data yang diperlukan.

G. Relasi: Relasi adalah hubungan antara tabel dalam basis data. Relasi ini digunakan dalam basis data relasional, di mana data disimpan dalam beberapa tabel yang terkait satu sama lain melalui kunci asing.

H. Normalisasi: Normalisasi adalah proses desain basis data yang mengurangi redundansi data dan memastikan data disimpan dalam bentuk yang paling efisien dan terstruktur. Ini membantu menjaga konsistensi data dan mencegah anomali data.

I. Hak Akses dan Keamanan: Komponen ini mencakup manajemen hak akses ke data, pengamanan basis data, dan kontrol akses pengguna. Ini penting untuk melindungi data sensitif dan mencegah akses yang tidak sah.

J. Pemulihan Data: Pemulihan data melibatkan strategi dan mekanisme untuk mengamankan data dalam hal kegagalan sistem atau kerusakan data. Ini mencakup pencadangan data dan pemulihan dari kerusakan.

Dengan bekerja bersama, komponen-komponen ini memungkinkan pengguna untuk menyimpan, mengelola, dan mengakses data dengan cara yang terstruktur dan efisien, yang sangat penting dalam berbagai konteks, mulai dari bisnis hingga penelitian dan aplikasi teknologi.

1.4. JENIS-JENIS SISTEM BASIS DATA

Jenis-jenis Sistem Basis Data adalah penting dalam konteks teknologi informasi yang semakin berkembang. Jenis-jenis ini memungkinkan organisasi dan pengembang untuk memilih platform yang paling sesuai dengan kebutuhan mereka. Mulai dari Sistem Basis Data Relasional yang umum digunakan hingga Sistem Basis Data NoSQL yang fleksibel, masing-masing jenis menawarkan pendekatan unik untuk penyimpanan, pengambilan, dan manajemen data. Selain itu, jenis-jenis tersebut juga termasuk dalam Sistem Basis Data Terdistribusi, Kolom, Dokumen, Grafik, dan banyak lagi, masing-masing dengan karakteristik dan keunggulan tertentu. Dalam panduan ini, kita akan menjelajahi lebih mendalam setiap jenis Sistem Basis Data, membantu dalam memahami cara mereka beroperasi dan kapan mereka cocok untuk berbagai kebutuhan bisnis dan teknis. Ketika kita menjelajahi berbagai jenis Sistem Basis Data, kita akan menemukan bahwa setiap jenis memiliki keunggulan dan keterbatasan masing-masing, serta aplikasi yang paling sesuai dengan lingkungan tertentu. Misalnya, Sistem Basis Data Relasional cocok untuk aplikasi yang membutuhkan struktur data yang ketat dan konsistensi, sementara Sistem Basis Data NoSQL lebih cocok untuk situasi yang memerlukan fleksibilitas dan skalabilitas yang tinggi. Sistem Basis Data Terdistribusi memungkinkan organisasi untuk mengelola data mereka secara efisien di seluruh lokasi geografis, sementara Sistem Basis Data Grafik adalah pilihan ideal untuk aplikasi yang berkaitan dengan hubungan dan jaringan data kompleks. Dengan pemahaman yang mendalam tentang jenis-jenis Sistem Basis Data, organisasi dapat membuat keputusan yang cerdas dalam memilih teknologi yang sesuai dengan tujuan dan tantangan

mereka, menjadikan data sebagai aset yang kuat dalam era digital yang terus berkembang.

Ada beberapa jenis Sistem Basis Data yang berbeda, masing-masing dirancang untuk mengatasi kebutuhan dan tantangan khusus dalam pengelolaan data. Berikut adalah beberapa jenis utama Sistem Basis Data:

A. Sistem Basis Data Relasional (RDBMS): Jenis ini adalah yang paling umum dan banyak digunakan. RDBMS menggunakan tabel untuk menyimpan data dan memanfaatkan relasi antara tabel untuk mengaitkan informasi. SQL (*Structured Query Language*) adalah bahasa yang paling sering digunakan untuk mengelola basis data relasional.
B. Sistem Basis Data *NoSQL*: Sistem Basis Data *NoSQL* adalah kelompok beragam basis data yang dirancang untuk mengatasi masalah skala, fleksibilitas, dan model data yang beragam. Mereka termasuk basis data berbasis dokumen, basis data kolom, basis data grafik, dan lain-lain.
C. Sistem Basis Data Terdistribusi: Jenis ini dirancang untuk mengelola data di sejumlah server yang terdistribusi geografis. Mereka mendukung replikasi data, partisi data, dan skalabilitas horizontal untuk mendukung lingkungan bisnis yang besar dan tersebar.
D. Sistem Basis Data Kolom: Basis data kolom mengkhususkan diri dalam menyimpan data dalam format kolom daripada baris. Mereka efisien untuk analisis dan pemrosesan data yang melibatkan banyak kolom dalam satu *query*.
E. Sistem Basis Data Dokumen: Basis data dokumen mengelola data dalam format dokumen, seperti JSON atau XML. Mereka cocok untuk aplikasi yang membutuhkan fleksibilitas dalam struktur data.

F. Sistem Basis Data Grafik: Jenis ini memodelkan data dalam bentuk grafik dan digunakan untuk mengelola data yang memiliki banyak hubungan dan dependensi.
G. Sistem Basis Data Waktu Nyata (*Real-Time Database*): Sistem ini memungkinkan pengelolaan data dengan tingkat respons yang sangat tinggi, yang sering digunakan dalam aplikasi waktu nyata seperti sensor *IoT* dan permainan daring.
H. Sistem Basis Data *In-Memory*: Basis data *in-memory* menyimpan data dalam memori komputer daripada disk, yang memungkinkan akses data yang sangat cepat.
I. Sistem Basis Data Khusus: Ada juga Sistem Basis Data khusus yang dirancang untuk kasus penggunaan tertentu, seperti basis data geografis, basis data tempat penyimpanan, dan lain-lain.

Setiap jenis Sistem Basis Data memiliki karakteristik dan keunggulan yang berbeda, sehingga pemilihan tergantung pada kebutuhan spesifik suatu aplikasi atau organisasi.

1.5. SOAL LATIHAN SISTEM BASIS DATA

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) Mengapa Sistem Basis Data sangat penting dalam dunia teknologi informasi? Jelaskan dalam beberapa kalimat.
2) Apa manfaat utama yang diberikan oleh Sistem Basis Data dalam pengelolaan data bagi organisasi?
3) Apa yang dimaksud dengan bahasa *query* dalam Sistem Basis Data, dan mengapa itu digunakan?
4) Bagaimana data *dictionary* mendukung administrasi dan pengembangan basis data?
5) Apa perbedaan utama antara Sistem Basis Data Relasional (RDBMS) dan Sistem Basis Data *NoSQL*, dan kapan Anda akan memilih salah satunya?

6) Apa perbedaan antara Sistem Basis Data Waktu Nyata (*Real-Time Database*) dan Sistem Basis Data konvensional, dan di mana situasi penggunaan yang paling sesuai untuk masing-masing?

BAB II
NORMALISASI DALAM SISTEM BASIS DATA

Normalisasi dalam sistem basis data adalah suatu proses esensial yang dirancang untuk mengorganisasi dan mengoptimalkan struktur basis data agar memenuhi prinsip-prinsip desain yang tepat. Praktik normalisasi ini bertujuan untuk mengurangi redundansi data, meminimalkan kesalahan penyimpanan, dan meningkatkan efisiensi operasional dalam sistem basis data. Dengan mengatur data ke dalam tabel yang terstruktur dengan baik dan memastikan setiap data hanya tersimpan sekali, normalisasi membantu dalam pemeliharaan data yang konsisten dan akurat, sehingga memudahkan proses manipulasi, pengambilan, dan pembaruan informasi. Dengan pemahaman yang mendalam tentang konsep normalisasi, pengelola basis data dan pengembang aplikasi dapat mencapai desain basis data yang optimal untuk memenuhi kebutuhan bisnis dan aplikasi dengan lebih efektif. Selain efisiensi dalam penyimpanan data, normalisasi juga berkontribusi pada integritas referensial, yang merupakan aspek penting dalam pemeliharaan konsistensi data. Dengan memastikan data yang saling berhubungan dipisahkan dengan benar dalam tabel yang berbeda, normalisasi meminimalkan potensi kesalahan dan inkonsistensi data. Konsep normalisasi juga memungkinkan untuk perubahan dan perluasan struktur basis data dengan lebih sedikit gangguan terhadap aplikasi yang bergantung pada basis data tersebut. Oleh karena itu, pemahaman yang kuat tentang normalisasi adalah landasan penting dalam merancang, mengelola, dan memelihara basis data yang andal dan efisien. Dalam dunia yang terus berubah dan semakin terhubung ini, normalisasi menjadi elemen kunci dalam

memastikan data tetap dapat diakses, dikelola, dan dianalisis dengan baik untuk mendukung keputusan bisnis dan inovasi teknologi.

2.1. PENTINGNYA NORMALISASI

Normalisasi adalah proses kritis yang membantu memastikan basis data tetap efisien, konsisten, dan mudah dikelola. Dengan mengurangi redundansi data, normalisasi membantu menghemat ruang penyimpanan, mengurangi risiko kesalahan data, dan meningkatkan integritas data. Hal ini juga mendukung kinerja basis data dengan mengoptimalkan pencarian dan pemrosesan data. Normalisasi memungkinkan perubahan dan perluasan struktur basis data dengan lebih mudah, memungkinkan fleksibilitas dalam menghadapi perubahan kebutuhan bisnis. Selain itu, data yang diorganisasi dengan baik dalam basis data yang dinormalisasi memfasilitasi integrasi dengan berbagai aplikasi dan mendukung pengambilan keputusan yang lebih baik. Dalam dunia yang semakin terhubung dan data-driven, normalisasi adalah langkah penting dalam memastikan data menjadi aset yang kuat dan berharga bagi organisasi. Sangat penting untuk diingat bahwa normalisasi juga berkontribusi pada efisiensi dalam pengembangan perangkat lunak dan pemeliharaan sistem. Saat struktur basis data telah dinormalisasi dengan baik, perubahan dalam aplikasi atau kebutuhan bisnis dapat diterapkan dengan lebih lancar. Ini karena perubahan hanya perlu dilakukan pada satu tempat, bukan di beberapa tempat dalam basis data yang saling terkait. Dengan begitu, normalisasi tidak hanya membantu meminimalkan risiko kesalahan data, tetapi juga menyederhanakan pengembangan dan pemeliharaan sistem secara keseluruhan. Dalam lingkungan bisnis yang berubah dengan cepat, normalisasi adalah alat

yang memungkinkan adaptasi dan inovasi yang lebih mudah dan efisien dalam pengelolaan data dan sistem informasi.

Normalisasi juga mendukung standarisasi dan dokumentasi yang lebih baik dalam pengelolaan data. Dengan struktur data yang telah dinormalisasi, konsep dan hubungan antar data menjadi lebih jelas dan mudah untuk didokumentasikan. Ini membantu dalam komunikasi dan pemahaman bersama antara tim pengembangan, administrator basis data, dan pengguna akhir. Standarisasi ini juga memudahkan pelaporan dan analisis data yang konsisten, yang pada gilirannya mendukung pengambilan keputusan yang lebih baik. Terakhir, penting untuk mencatat bahwa sementara normalisasi sangat penting, tidak selalu menjadi pilihan yang benar untuk setiap situasi. Dalam beberapa kasus, normalisasi yang berlebihan dapat mengakibatkan kompleksitas yang tidak perlu atau mengorbankan kinerja. Oleh karena itu, perlu mempertimbangkan kebutuhan khusus dan karakteristik aplikasi ketika merancang basis data, dan ada situasi di mana denormalisasi (mengurangi tingkat normalisasi) dapat menjadi pilihan yang lebih bijak. Dalam rangka mencapai manfaat yang paling optimal, perancangan basis data dan tingkat normalisasi harus disesuaikan dengan konteks dan kebutuhan spesifik dari proyek atau aplikasi tertentu.

Normalisasi adalah proses penting dalam desain dan pengelolaan Sistem Basis Data. Berikut adalah beberapa alasan mengapa normalisasi sangat penting:

A. Pengurangan Redundansi Data: Normalisasi membantu mengurangi duplikasi dan redundansi data dalam basis data. Ini mengurangi kebutuhan untuk menyimpan data yang sama berulang kali, yang pada gilirannya menghemat ruang penyimpanan dan meminimalkan risiko kesalahan dalam konsistensi data.

B. Integritas Data: Normalisasi memastikan integritas data dengan membatasi data yang disimpan dalam tabel ke informasi yang relevan. Dengan menghindari penambahan data yang tidak perlu, basis data tetap konsisten dan akurat.
C. Penghematan Ruang Penyimpanan: Dengan menghilangkan duplikasi data dan hanya menyimpan informasi yang benar-benar diperlukan, normalisasi dapat menghemat ruang penyimpanan. Ini berarti basis data lebih efisien dan menghemat biaya penyimpanan fisik.
D. Perbaikan Kinerja: Normalisasi dapat meningkatkan kinerja pengambilan data. Dengan meminimalkan ukuran tabel, pencarian dan pemrosesan data dapat menjadi lebih cepat dan efisien.
E. Perawatan yang Lebih Mudah: Normalisasi membuat pemeliharaan basis data menjadi lebih mudah. Ketika perubahan struktur diperlukan, perubahan hanya perlu dilakukan pada satu tempat daripada beberapa tabel yang saling terkait.
F. Ketelitian Data: Dengan menyimpan data hanya dalam satu tempat, normalisasi membantu memastikan bahwa data tidak tersimpan secara inkonsisten atau kontradiktif dalam basis data.
G. Kemudahan dalam Perubahan dan Perluasan: Normalisasi mempermudah penyesuaian struktur basis data dengan perubahan kebutuhan bisnis dan perluasan sistem. Basis data yang telah dinormalisasi cenderung lebih fleksibel.
H. Integrasi dengan Aplikasi: Normalisasi memudahkan integrasi basis data dengan berbagai aplikasi. Data yang diorganisasi dengan baik dalam basis data yang dinormalisasi lebih mudah diakses oleh aplikasi yang berbeda.

Dengan kata lain, normalisasi adalah praktik desain yang esensial dalam memastikan bahwa data dalam basis data tetap konsisten, akurat, dan efisien digunakan. Ini membantu dalam menghindari masalah yang mungkin muncul ketika data disimpan secara tidak terstruktur atau redundan, serta mendukung fleksibilitas dan kemudahan pemeliharaan dalam jangka panjang.

2.2. ATURAN NORMALISASI

Aturan normalisasi adalah pedoman kunci dalam merancang struktur basis data yang efisien dan konsisten. Prinsip-prinsip normalisasi mengarah pada pengurangan redundansi data, meningkatkan integritas data, dan meminimalkan anomali yang mungkin muncul dalam pengelolaan data. Aturan-aturan ini melibatkan pemisahan data ke dalam tabel yang logis, identifikasi kunci utama yang unik, dan menjaga ketergantungan fungsional yang jelas antara atribut. Melalui normalisasi, basis data menjadi lebih mudah dikelola, pemeliharaan data menjadi lebih terstruktur, dan integritas data terjaga dengan baik. Meskipun normalisasi penting dalam desain basis data, perancang harus tetap mempertimbangkan kebutuhan khusus dan tujuan aplikasi untuk mencapai keseimbangan yang tepat antara normalisasi dan kinerja sistem. Selain itu, aturan normalisasi juga mencakup konsep pemisahan data ke dalam tabel terkait yang memungkinkan pengorganisasian data dengan lebih baik. Dengan demikian, normalisasi tidak hanya meminimalkan redundansi, tetapi juga memungkinkan perancang basis data untuk menghindari anomali data seperti perubahan yang inkonsisten, penghapusan yang tidak diinginkan, atau ketidaksesuaian data. Penerapan aturan normalisasi dengan tepat dapat meningkatkan kualitas data dan memudahkan penggunaan data dalam aplikasi yang beragam. Pemahaman

yang kuat tentang aturan-aturan normalisasi merupakan landasan penting dalam merancang basis data yang efisien dan andal yang mendukung kebutuhan bisnis dan teknis suatu organisasi.

Aturan normalisasi mengacu pada prinsip-prinsip dan pedoman yang digunakan dalam proses merancang struktur basis data yang dinormalisasi. Normalisasi bertujuan untuk mengurangi redundansi data, meningkatkan integritas data, dan meminimalkan anomali data. Ada beberapa bentuk normalisasi yang didefinisikan dalam bentuk "normalisasi tingkat pertama," "normalisasi tingkat kedua," dan seterusnya, yang semuanya mengacu pada aturan tertentu. Berikut adalah aturan normalisasi utama yang diterapkan pada setiap tingkatan normalisasi:

A. Eliminasi Redundansi: Aturan utama dalam normalisasi adalah menghilangkan redundansi data. Ini berarti setiap fakta atau informasi hanya disimpan sekali dalam basis data.
B. Ketergantungan Fungsional: Setiap kolom atau atribut dalam tabel harus tergantung sepenuhnya pada kunci utama. Dalam normalisasi, ini dikenal sebagai "ketergantungan fungsional penuh."
C. Identifikasi Kunci: Setiap tabel harus memiliki satu atau lebih kunci utama yang unik. Kunci utama digunakan untuk mengidentifikasi secara unik setiap baris dalam tabel.
D. Pemisahan Data: Data yang berkaitan tetapi tidak sepenuhnya tergantung satu sama lain harus dipisahkan ke dalam tabel terpisah. Ini membantu dalam menghindari anomali data.
E. Tabel Terkait: Untuk setiap ketergantungan fungsional yang ada di antara tabel, hubungan antar tabel harus didefinisikan melalui penggunaan kunci asing.

F. Penamaan yang Konsisten: Menggunakan penamaan yang konsisten untuk tabel, kolom, dan indeks mempermudah pemahaman dan pemeliharaan basis data.
G. Normalisasi Hingga Tingkat yang Diperlukan: Normalisasi tidak selalu harus mencapai tingkat yang paling tinggi (normalisasi tingkat kelima). Sebagai perancang basis data, Anda perlu mempertimbangkan keseimbangan antara normalisasi dan kinerja. Terkadang, sedikit denormalisasi dapat diterapkan untuk meningkatkan kinerja.
H. Pemeliharaan Data: Basis data yang dinormalisasi memerlukan perhatian yang cermat terhadap pemeliharaan integritas data dan perubahan struktur yang mungkin diperlukan seiring waktu.

Aturan-aturan ini membantu memastikan bahwa basis data dapat menjaga integritas data, mengurangi risiko kesalahan, dan mengoptimalkan kinerja. Meskipun normalisasi merupakan prinsip penting dalam desain basis data, perlu diingat bahwa setiap proyek atau situasi dapat memiliki kebutuhan khusus yang mempengaruhi sejauh mana normalisasi harus diterapkan.

Aturan normalisasi juga memiliki dampak penting dalam kaitannya dengan pemeliharaan dan fleksibilitas dalam pengelolaan basis data. Dengan basis data yang dinormalisasi, pemeliharaan dan perubahan struktur basis data menjadi lebih terstruktur dan mudah dikelola. Hal ini dapat mengurangi risiko kesalahan dan memungkinkan perubahan yang lebih lancar dalam respons terhadap perubahan kebutuhan bisnis atau teknologi. Selain itu, normalisasi mendukung fleksibilitas dalam pengelolaan data. Dengan struktur yang terorganisir dengan baik, basis data dapat digunakan dengan efisien dalam berbagai aplikasi yang berbeda. Ini juga memungkinkan data untuk

diintegrasikan dengan mudah dalam konteks yang beragam, yang mendukung pengambilan keputusan yang lebih baik.

2.3. TINGKATAN NORMALISASI

Tingkatan normalisasi merujuk pada hierarki desain basis data yang dirancang untuk mengoptimalkan cara data disimpan dan diorganisasi dalam tabel. Normalisasi bertujuan untuk mengurangi redundansi data, meningkatkan integritas data, dan menghindari anomali yang mungkin terjadi dalam pengelolaan data(McHugh et al., 1997). Tingkatan normalisasi mulai dari tingkat pertama (1NF) hingga tingkat yang lebih tinggi seperti BCNF dan 5NF, masing-masing dengan persyaratan yang semakin ketat. Tingkatan normalisasi yang lebih tinggi mengharuskan konsiderasi yang lebih dalam terhadap ketergantungan dan hubungan antar kolom dalam basis data. Pemilihan tingkat normalisasi yang tepat harus didasarkan pada kebutuhan dan karakteristik aplikasi tertentu, dengan mempertimbangkan keseimbangan antara normalisasi dan kinerja sistem. Tingkatan normalisasi adalah alat penting dalam merancang basis data yang efisien dan konsisten. Normalisasi membantu dalam menghindari masalah seperti duplikasi data, ketergantungan yang tidak perlu, dan anomali yang dapat merusak integritas data. Meskipun tingkatan normalisasi yang lebih tinggi sering dianggap sebagai tujuan yang baik dalam desain basis data, penting juga untuk memahami bahwa setiap proyek atau aplikasi mungkin memiliki kebutuhan yang berbeda. Terlalu banyak normalisasi dapat mengurangi kinerja dalam beberapa kasus, sedangkan tingkat normalisasi yang lebih rendah dapat memungkinkan fleksibilitas yang dibutuhkan. Oleh karena itu, perancang basis data perlu mempertimbangkan dengan cermat persyaratan unik setiap proyek dan

mengambil keputusan yang bijaksana dalam memilih tingkat normalisasi yang paling sesuai dengan kebutuhan tersebut.

Selain memilih tingkat normalisasi yang sesuai, perancang basis data juga perlu memantau kinerja basis data dalam jangka panjang. Terlalu banyak normalisasi dapat mengakibatkan kueri menjadi lambat karena seringkali memerlukan penggabungan data dari beberapa tabel. Dalam beberapa situasi, sedikit denormalisasi (mengurangi tingkat normalisasi) dapat digunakan untuk meningkatkan kinerja, asalkan dengan bijak dan sesuai dengan kebutuhan aplikasi. Penting juga untuk diingat bahwa kebutuhan bisnis dan teknis dapat berubah seiring waktu. Oleh karena itu, reevaluasi dan modifikasi terhadap struktur basis data dapat menjadi perlu, dan perancang basis data harus memiliki pemahaman yang kuat tentang tingkatan normalisasi untuk membuat keputusan yang tepat. Pemahaman yang mendalam tentang tingkatan normalisasi adalah kunci dalam merancang basis data yang memenuhi persyaratan aplikasi, menghindari masalah data, dan memastikan kinerja yang optimal dalam jangka panjang.

Normalisasi adalah proses desain basis data yang bertujuan mengorganisasi data dalam tabel dengan cara yang mengurangi redundansi dan meningkatkan integritas data. Normalisasi menghasilkan beberapa tingkatan, biasanya disebut "normalisasi tingkat pertama," "normalisasi tingkat kedua," dan seterusnya, yang memiliki tingkat normalisasi yang semakin tinggi. Berikut adalah tingkatan normalisasi yang umumnya diakui:

A. Normalisasi Tingkat Pertama (1NF - *First Normal Form*): Pada tingkat ini, data dalam tabel diorganisasi sedemikian rupa sehingga setiap kolom berisi nilai atomik, tidak ada kelompok nilai atau berbagai nilai yang disimpan dalam satu kolom. Ini menghilangkan duplikasi data pada baris yang sama.

B. Normalisasi Tingkat Kedua (2NF - *Second Normal Form*): Pada tingkat ini, selain memenuhi syarat Normalisasi Tingkat Pertama, setiap kolom yang bukan merupakan kunci utama harus sepenuhnya bergantung pada kunci utama. Ini menghilangkan ketergantungan parsial.
C. Normalisasi Tingkat Ketiga (3NF - *Third Normal Form*): Selain memenuhi syarat Normalisasi Tingkat Kedua, setiap kolom yang bukan kunci utama tidak boleh memiliki ketergantungan transitif pada kunci utama. Ini meminimalkan ketergantungan antara kolom non-kunci.
D. Normalisasi Tingkat Keempat (4NF - *Fourth Normal Form*): Pada tingkat ini, kita mengatasi masalah ketergantungan multivalued, yaitu ketika satu atau lebih kolom memiliki lebih dari satu nilai yang tidak ada kaitannya dengan nilai-nilai kolom lain.
E. Normalisasi Tingkat Kelima (5NF - *Fifth Normal Form*): Normalisasi tingkat ini terkait dengan masalah ketergantungan join. Ini mengatasi hubungan kompleks antara tabel yang terkait.

Selain tingkatan normalisasi ini, ada juga tingkatan yang lebih tinggi seperti tingkat normalisasi BCNF (*Boyce-Codd Normal Form*) dan tingkat normalisasi 6NF. Tingkat normalisasi yang lebih tinggi mengharuskan pemenuhan kriteria yang semakin ketat, namun juga memerlukan analisis yang lebih cermat dalam perancangan basis data. Pemilihan tingkat normalisasi yang tepat harus selalu mempertimbangkan kebutuhan aplikasi, kinerja, dan kompleksitas. Terlalu banyak normalisasi dapat mengurangi kinerja, sementara terlalu sedikit dapat mengakibatkan redundansi data. Oleh karena itu, perancang basis data harus memutuskan tingkat normalisasi yang paling sesuai dengan tujuan dan persyaratan aplikasi.

2.4. PROSES NORMALISASI

Proses normalisasi adalah langkah penting dalam merancang struktur basis data yang efisien dan konsisten. Ini melibatkan serangkaian tahap yang bertujuan mengorganisasi data dalam tabel agar mengikuti prinsip-prinsip normalisasi. Proses dimulai dengan identifikasi entitas dan atribut, penentuan kunci utama, dan memastikan bahwa data dalam tabel memenuhi kriteria Normalisasi Tingkat Pertama (1NF), seperti menghindari kelompok nilai dan duplikasi data. Kemudian, langkah-langkah berlanjut untuk memastikan ketergantungan data yang sesuai, mengidentifikasi kunci asing untuk menghubungkan tabel, dan mengoptimalkan struktur basis data. Selain itu, dokumentasi yang cermat dan pemeliharaan yang terus-menerus penting dalam memastikan integritas data dalam jangka panjang. Proses normalisasi memberikan dasar kuat untuk penyimpanan data yang efisien, konsisten, dan dapat dikelola, sesuai dengan kebutuhan dan tujuan aplikasi.

Proses normalisasi adalah serangkaian langkah yang dilakukan dalam desain basis data untuk mengorganisasi data sehingga mengikuti prinsip normalisasi. Normalisasi bertujuan mengurangi redundansi data, meningkatkan integritas data, dan menghindari anomali data. Proses normalisasi biasanya terdiri dari beberapa tingkatan yang dikenal sebagai "normalisasi tingkat pertama" (1NF), "normalisasi tingkat kedua" (2NF), dan seterusnya. Berikut adalah langkah-langkah umum dalam proses normalisasi:

A. Identifikasi Entitas dan Atribut: Identifikasi entitas utama (tabel) dalam basis data dan atribut (kolom) yang terkait dengan entitas tersebut.
B. Identifikasi Kunci Utama: Tentukan kunci utama untuk setiap entitas. Kunci utama adalah kolom atau gabungan

kolom yang unik mengidentifikasi setiap baris dalam tabel.

C. Memastikan Normalisasi Tingkat Pertama (1NF): Pastikan bahwa setiap kolom dalam tabel hanya berisi nilai atomik (tidak ada kelompok nilai atau larik) dan bahwa tidak ada duplikasi data dalam baris yang sama.
D. Normalisasi Tingkat Kedua (2NF): Pastikan setiap kolom yang bukan kunci utama sepenuhnya bergantung pada kunci utama. Jika ada ketergantungan parsial, pisahkan data ke dalam tabel terkait.
E. Normalisasi Tingkat Ketiga (3NF): Pastikan bahwa tidak ada ketergantungan transitif pada kunci utama. Jika ada, pisahkan kolom non-kunci yang terkait.
F. Normalisasi Tingkat Selanjutnya (Opsional): Untuk mencapai tingkat normalisasi yang lebih tinggi seperti BCNF atau 5NF, identifikasi dan atasi masalah seperti ketergantungan multivalued atau ketergantungan join.
G. Hubungkan Tabel: Tentukan hubungan antara tabel dengan menggunakan kunci asing. Ini memungkinkan pengambilan data yang berkaitan melalui operasi JOIN.
H. Optimasi Struktur: Periksa struktur basis data yang telah dinormalisasi untuk memastikan bahwa kinerja masih memadai. Dalam beberapa kasus, pertimbangan denormalisasi mungkin diperlukan untuk meningkatkan kinerja.
I. Dokumentasi: Dokumentasikan struktur basis data dengan baik, termasuk kunci utama, hubungan, dan ketergantungan antar tabel.
J. Pemeliharaan: Selalu pertimbangkan pemeliharaan data dalam jjangka panjang dan reevaluasi struktur basis data ketika perubahan kebutuhan bisnis muncul.

Proses normalisasi adalah proses berkelanjutan yang melibatkan analisis dan perancangan struktur basis data yang cerdas, dengan tujuan untuk mencapai keseimbangan

antara normalisasi yang optimal dan kinerja sistem yang efisien. Keseluruhan proses ini memerlukan pemahaman mendalam tentang aturan normalisasi dan persyaratan unik dari aplikasi atau proyek tertentu.

Ilustrasi tahapan normalisasi dengan contoh sederhana tentang sebuah basis data untuk merekam informasi mahasiswa dan mata kuliah yang diambil. Dalam contoh ini, kita akan melalui tahapan normalisasi dari awal hingga mencapai tahap 3NF. Tabel berikut ini adalah tabel awal sebelum dilakukan normalisasi.

Tabel 2.1. Tabel Awal

NIM	Nama Mahasiswa	Mata Kuliah	Dosen	Nilai
111	Ahmad	Web	Prof. Handoko	A
112	Budi	SBD	Dr. Rio	B
113	Joko	Algoritma	Prof. Wawan	C

Selanjutnya kita akan membuat tahapan normalisasi 1NF dengan membuat tabel mahasiswa.

Tabel 2.2. Tabel Mahasiswa

NIM	Nama Mahasiswa	Mata Kuliah	Dosen	Nilai
111	Ahmad	Web	Prof. Handoko	A
112	Budi	SBD	Dr. Rio	B
113	Joko	Algoritma	Prof. Wawan	C

Selanjutnya kita akan membuat tahapan normalisasi 2NF dengan membuat tabel mahasiswa dan tabel mata kuliah.

Tabel 2.3. Tabel Mahasiswa

NIM	Nama Mahasiswa	Nilai
111	Ahmad	A
112	Budi	B
113	Joko	C

Tabel 2.4. Tabel Matakuliah

Mata Kuliah	Dosen
Web	Prof. Handoko
SBD	Dr. Rio
Algoritma	Prof. Wawan

Selanjutnya kita akan membuat tahapan normalisasi 3NF dengan membuat tabel mahasiswa, tabel mata kuliah dan tabel dosen.

Tabel 2.5. Tabel Mahasiswa

NIM	Nama Mahasiswa	Nilai
111	Ahmad	A
112	Budi	B
113	Joko	C

Tabel 2.6. Tabel Matakuliah

Mata Kuliah	Kode Dosen
Web	D-101
SBD	D-102
Algoritma	D-103

Tabel 2.7. Tabel Dosen

Kode Dosen	Nama Dosen
D-101	Prof. Handoko
D-102	Dr. Rio
D-103	Prof. Wawan

Penjelasan tahap normalisasi diatas yaitu:

a) Tahap 1NF: Tabel awal sudah memenuhi 1NF karena setiap sel berisi satu nilai saja dan tidak ada nilai berulang dalam satu kolom.
b) Tahap 2NF: Kita mengidentifikasi bahwa ada ketergantungan parsial antara atribut "Mata Kuliah" dan

"Dosen" terhadap "NIM". Sebagai hasilnya, kita memecah tabel menjadi "Mahasiswa" dan "Mata Kuliah".

c) Tahap 3NF: Dalam tabel "Mata Kuliah", kita mengidentifikasi ketergantungan transitif antara atribut "Mata Kuliah" dan "Dosen" terhadap "Kode Dosen". Oleh karena itu, kita memecah "Mata Kuliah" menjadi "Mata Kuliah" dan "Dosen".

Setelah tahap 3NF, basis data telah dinormalisasi dengan baik dan memiliki struktur yang efisien dan bebas dari redundansi data. Hubungan antar tabel juga lebih jelas, dan tabel-tabel yang terpisah memungkinkan penyimpanan dan manipulasi data yang lebih efisien.

2.5. SOAL LATIHAN NORMALISASI

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) Diberikan sebuah tabel dengan atribut berikut: (A, B, C, D, E). Identifikasi kunci utama dari tabel ini dan pastikan bahwa tabel ini memenuhi Normalisasi Tingkat Pertama (1NF).
2) Anda memiliki tabel dengan atribut: (Nomor_Pegawai, Nama_Pegawai, Alamat_Kantor, Nomor_Telepon). Identifikasi kunci utama tabel ini dan normalisasikan tabel ke Normalisasi Tingkat Kedua (2NF).
3) Dalam basis data perpustakaan, ada tabel dengan atribut: (ID_Buku, Judul_Buku, Penulis, ID_Pustaka, Nama_Pustaka). Identifikasi kunci utama dan pastikan bahwa tabel memenuhi Normalisasi Tingkat Ketiga (3NF).
4) Anda bekerja pada sistem penjualan online yang memiliki tabel dengan atribut: (ID_Transaksi, Nama_Pelanggan, Alamat_Pengiriman, Nama_Produk, Harga_Produk). Identifikasi kunci utama dan pastikan

bahwa tabel ini memenuhi Normalisasi Tingkat Pertama (1NF).

5) Dalam sebuah sistem manajemen proyek, ada tabel yang menyimpan informasi tentang tugas-tugas proyek. Atribut tabel ini adalah (ID_Tugas, Nama_Tugas, Deskripsi, ID_Proyek, Nama_Proyek, ID_Karyawan, Nama_Karyawan). Tentukan kunci utama dan normalisasikan tabel ke tingkat yang sesuai.

BAB III
STRUCTURED QUERY LANGUAGE

Structured Query Language (SQL) adalah bahasa pemrograman khusus yang digunakan dalam manajemen sistem basis data relasional. SQL memiliki peran sentral dalam interaksi dengan basis data, memungkinkan pengguna untuk mengakses, memanipulasi, dan mengambil data dengan cara yang terstruktur dan efisien. Keunggulan SQL terletak pada kemampuannya yang universal, sehingga dapat digunakan di berbagai sistem basis data, seperti *MySQL*, *Oracle*, dan *Microsoft* SQL *Server*. Pemahaman yang kuat tentang SQL menjadi penting bagi pengelola basis data, pengembang aplikasi, dan analis data, karena ini adalah kunci untuk mengoptimalkan akses dan pengelolaan data, serta menjalankan tugas-tugas penting dalam pengembangan dan analisis informasi. SQL juga memiliki peran vital dalam mendukung bisnis dan pengambilan keputusan berdasarkan data. Dengan menggunakan SQL, organisasi dapat menggali wawasan berharga dari data mereka, mengidentifikasi tren, menganalisis performa bisnis, dan menyajikan laporan yang relevan. SQL juga memfasilitasi interaksi dengan basis data dalam aplikasi bisnis, seperti sistem manajemen inventaris, sistem keuangan, dan sistem manajemen pelanggan. Selain itu, SQL terus berkembang dengan penambahan fitur-fitur baru dalam setiap versi, seperti analitik data, pemrosesan transaksi, dan keamanan data yang semakin canggih. Oleh karena itu, pemahaman yang mendalam tentang SQL adalah keterampilan yang sangat berharga dalam dunia teknologi informasi dan bisnis saat ini, yang semakin bergantung pada data untuk mendukung keputusan dan inovasi.

3.1. PENGANTAR *STRUCTURED QUERY LANGUAGE*

Structured Query Language (SQL) adalah bahasa pemrograman yang digunakan untuk mengelola dan mengakses basis data relasional. SQL adalah salah satu bahasa yang paling umum digunakan di dunia teknologi informasi, digunakan untuk membuat, mengubah, menghapus, dan mengambil data dari *database*(Sumathi & Esakkirajan, 2007). Dengan SQL, pengguna dapat menggabungkan, mengorganisir, dan mengambil informasi dari tabel-tabel yang berhubungan dalam database, serta menjalankan berbagai operasi seperti pengurutan, penyaringan, dan penghitungan data. Kemampuan SQL untuk mengelola data secara terstruktur dan efisien menjadikannya alat yang vital dalam pengembangan aplikasi berbasis *database*, analisis data, dan pengambilan keputusan berdasarkan informasi yang tersimpan dalam basis data relasional. Selain itu, SQL memiliki standar yang kuat, yang memungkinkan para pengembang dan administrator database untuk bekerja dengan basis data yang berbeda tanpa harus belajar bahasa yang berbeda-beda. SQL juga memberikan fleksibilitas dalam menentukan aturan integritas data, seperti kunci asing, kunci utama, dan pembatasan lainnya, yang membantu dalam menjaga kualitas dan konsistensi data. Dengan demikian, pemahaman yang baik tentang SQL sangat penting bagi siapa saja yang terlibat dalam pengelolaan data atau pengembangan aplikasi berbasis *database*. SQL terus berkembang dan diperbarui, sehingga para profesional TI perlu terus memperbarui pengetahuan mereka untuk mengikuti perkembangan terbaru dalam dunia basis data relasional.

Penting untuk dicatat bahwa SQL memiliki beberapa varian atau dialek yang digunakan oleh berbagai sistem *database*, seperti *MySQL*, *PostgreSQL*, *Microsoft* SQL *Server*,

Oracle, dan banyak lainnya. Meskipun prinsip dasar SQL tetap sama, sintaksisnya dapat bervariasi sedikit antara *platform-platform* ini. Oleh karena itu, pengguna SQL perlu memahami perbedaan-perbedaan tersebut ketika bekerja dengan sistem database yang berbeda. Selain itu, SQL juga terus berkembang dengan munculnya standar baru dan fitur-fitur tambahan, seperti pemrosesan big data dan analisis lanjutan. Dalam perkembangan teknologi informasi yang terus berlanjut, SQL tetap menjadi alat yang kuat dan relevan dalam pengelolaan data dan analisis, dan pengetahuan yang mendalam tentang SQL menjadi aset berharga bagi siapa saja yang terlibat dalam dunia database dan informasi.

Dalam konteks dunia modern, SQL juga telah menjadi bagian integral dari berbagai aplikasi dan layanan web. Banyak situs web dan *platform* layanan *cloud* menggunakan SQL sebagai cara untuk menyimpan dan mengakses data pengguna, seperti informasi akun, pesanan, dan sebagainya. SQL juga digunakan dalam aplikasi *mobile* dan banyak sistem perangkat lunak berbasis *server*. Dengan demikian, pemahaman SQL menjadi esensial bagi pengembang perangkat lunak, analis data, dan administrator sistem yang berurusan dengan beragam teknologi. Terlepas dari peran kunci SQL dalam mengelola data relasional, ada juga perkembangan teknologi *NoSQL* (*Not Only SQL*) yang menghadirkan alternatif bagi kasus penggunaan yang berbeda, seperti data semi-struktural atau tidak terstruktur. Meskipun demikian, SQL tetap menjadi fondasi penting dalam berbagai domain, dan pemahaman yang kuat tentang bahasa ini membantu profesional teknologi informasi dalam mengatasi berbagai tantangan yang terkait dengan basis data relasional dan analisis data. Selain itu, terus berkembangnya SQL memungkinkan adopsi lebih luas di berbagai kasus penggunaan, termasuk pemrosesan data

real-time dan skala besar. Oleh karena itu, SQL tetap menjadi landasan penting dalam dunia teknologi informasi yang terus berubah.

Terus berkembangnya SQL juga berdampak pada peningkatan keamanan data. SQL injection adalah salah satu contoh serangan siber yang dapat memanfaatkan celah dalam kode SQL untuk mengakses atau merusak data. Oleh karena itu, para profesional TI perlu memahami praktik-praktik terbaik dalam mencegah serangan semacam ini dan mengimplementasikannya dalam kode SQL mereka. Pengetahuan yang kuat tentang SQL juga penting untuk memahami dan mengoptimalkan kinerja kueri database, sehingga aplikasi dapat berjalan lebih efisien dan responsif. Terakhir, SQL juga memiliki peran dalam analisis data dan pengambilan keputusan. Dengan kemampuannya untuk menggabungkan, menyaring, dan mengolah data, SQL memungkinkan para analis data untuk menghasilkan wawasan yang berharga dari informasi yang tersimpan dalam basis data. Ini berarti SQL tidak hanya menjadi alat bagi para pengembang dan administrator database, tetapi juga bagi para analis data yang memanfaatkannya untuk menjawab pertanyaan bisnis, mengidentifikasi tren, dan membuat rekomendasi yang didukung oleh data. Dengan demikian, SQL tetap menjadi komponen utama dalam ekosistem teknologi informasi yang melibatkan data, aplikasi, dan analisis. Penting untuk diingat bahwa SQL juga memiliki peran penting dalam dunia bisnis dan keuangan. Data bisnis yang disimpan dalam basis data relasional seringkali memuat informasi kunci tentang kinerja perusahaan, pelanggan, stok, dan lainnya. Dengan bantuan SQL, para profesional keuangan dapat menyusun laporan keuangan, mengikuti tren, dan melakukan analisis yang membantu dalam pengambilan keputusan bisnis. Hal ini menjadikan SQL sebagai alat yang sangat berharga dalam

mendukung pengelolaan bisnis yang efektif dan pengembangan strategi bisnis yang berdasarkan data.

3.2. DASAR-DASAR SQL

Structured Query Language, adalah fondasi utama bagi siapa saja yang berkecimpung dalam dunia basis data relasional. SQL merupakan bahasa pemrograman yang digunakan untuk mengelola, mengambil, memasukkan, mengubah, dan menghapus data dalam database. Dalam pengembangan perangkat lunak, analisis data, administrasi database, dan banyak aspek teknologi informasi lainnya, pemahaman dasar SQL adalah suatu keharusan. Prinsip-prinsip kunci dalam SQL, termasuk perintah dasar, penggunaan tabel, kunci utama, kunci asing, serta berbagai konsep penting lainnya yang membantu memahami cara berinteraksi dengan dan memanfaatkan *database* relasional. Selain itu, pemahaman yang mendalam tentang SQL juga memungkinkan profesional IT untuk mengoptimalkan kinerja *query*, merancang basis data yang efisien, dan menjalankan analisis data yang mendalam. Kemampuan ini tidak hanya penting dalam konteks teknis, tetapi juga memberikan manfaat bisnis yang signifikan dengan memungkinkan pengambilan keputusan yang didukung oleh data yang akurat dan relevan. Dengan demikian, pengetahuan tentang dasar-dasar SQL adalah investasi berharga untuk siapa saja yang ingin sukses dalam dunia teknologi informasi dan manajemen data.

Selain itu, pemahaman yang mendalam tentang SQL juga memungkinkan para profesional IT untuk mengelola data dengan lebih efisien, memastikan integritas data, serta menjawab pertanyaan bisnis kunci. SQL tidak hanya menjadi landasan untuk mengambil dan menyimpan data, tetapi juga menjadi alat yang kuat dalam menggali wawasan bisnis dari data yang ada. Dengan SQL, kita dapat menyusun kueri yang

rumit untuk mengidentifikasi tren, mencari pola, dan memahami perilaku pengguna, yang semuanya dapat mendukung pengambilan keputusan yang lebih baik. Oleh karena itu, pemahaman mendalam tentang dasar-dasar SQL menjadi aset yang sangat berharga dalam dunia yang semakin terhubung dan didorong oleh data ini. Berikut adalah beberapa dasar-dasar SQL:

A. Menggunakan *Database*: SQL digunakan untuk bekerja dengan *database* relasional. Sebuah database adalah kumpulan tabel yang menyimpan data dalam bentuk terstruktur.
B. Tabel: Tabel adalah entitas dasar dalam *database*. Setiap tabel terdiri dari kolom (*field*) dan baris (*record*). Kolom mendefinisikan jenis data yang dapat disimpan dalam tabel, sedangkan baris berisi data aktual.
C. Perintah SQL: Ada berbagai perintah SQL yang digunakan untuk berinteraksi dengan *database*, seperti *SELECT* untuk mengambil data, *INSERT* untuk menambahkan data, *UPDATE* untuk mengubah data, dan *DELETE* untuk menghapus data.
D. Kunci Utama (*Primary Key*): Setiap tabel biasanya memiliki satu atau beberapa kolom yang digunakan sebagai kunci utama. Kunci utama adalah nilai unik yang digunakan untuk mengidentifikasi setiap baris dalam tabel.
E. Kunci Asing (*Foreign Key*): Kunci asing adalah kolom dalam satu tabel yang mengacu ke kunci utama dalam tabel lain. Ini digunakan untuk menghubungkan data antara tabel dan menciptakan hubungan antar tabel (relasi).
F. Pernyataan *SELECT*: Pernyataan *SELECT* digunakan untuk mengambil data dari satu atau lebih tabel. Anda dapat menentukan kolom mana yang ingin Anda ambil, filter hasil dengan kondisi, dan mengurutkannya.

G. Pernyataan *INSERT*: Pernyataan *INSERT* digunakan untuk menambahkan data baru ke dalam tabel. Anda harus menentukan nilai yang ingin Anda masukkan ke dalam kolom yang sesuai.
H. Pernyataan *UPDATE*: Pernyataan *UPDATE* digunakan untuk memperbarui data yang sudah ada dalam tabel. Anda harus menentukan kolom yang ingin diubah dan nilai baru yang ingin diberikan.
I. Pernyataan *DELETE*: Pernyataan *DELETE* digunakan untuk menghapus data dari tabel. Anda harus memberikan kondisi yang menentukan data mana yang akan dihapus.
J. Kondisi: Kondisi digunakan untuk memfilter data dalam pernyataan *SELECT*, *UPDATE*, dan *DELETE*. Misalnya, Anda dapat menggunakan kondisi *WHERE* untuk membatasi data yang akan diambil atau diubah.
K. Aggregasi: SQL mendukung fungsi agregasi seperti SUM, *COUNT*, AVG, MIN, dan MAX yang digunakan untuk menghitung statistik data, seperti jumlah atau rata-rata.
L. Gabungan Tabel (*JOIN*): SQL memungkinkan Anda untuk menggabungkan data dari beberapa tabel menggunakan pernyataan *JOIN*. Ini berguna untuk mengambil data yang berhubungan dari tabel yang berbeda.
M. Pembuatan Tabel: Anda dapat membuat tabel baru dengan pernyataan *CREATE TABLE*. Ini melibatkan menentukan nama tabel, kolom, tipe data kolom, dan kunci utama.
N. Indeks: Indeks digunakan untuk mempercepat pencarian data dalam tabel. Anda dapat membuat indeks pada satu atau beberapa kolom tertentu untuk meningkatkan kinerja.
O. Transaksi: SQL mendukung transaksi, yang memungkinkan Anda untuk menjalankan beberapa

perintah sebagai satu kesatuan yang konsisten, sehingga entri data tetap konsisten bahkan dalam situasi yang tidak terduga.

P. Fungsi: SQL menyediakan berbagai fungsi bawaan, seperti fungsi tanggal dan waktu (seperti *NOW*() atau *DATE_FORMAT*()), fungsi matematika (seperti *ROUND*() atau ABS()), dan fungsi teks (seperti *CONCAT*() atau *UPPER*()), yang memungkinkan Anda untuk melakukan operasi dan manipulasi data lebih lanjut.

Q. *Subquery*: SQL memungkinkan Anda untuk menyertakan *subquery*, yang adalah pernyataan SQL yang ada di dalam pernyataan SQL lainnya. *Subquery* digunakan untuk mengambil data dari tabel berdasarkan hasil dari pernyataan SQL lainnya.

R. Grup dan Pengelompokan (*GROUP BY* dan *HAVING*): Anda dapat menggunakan pernyataan *GROUP BY* untuk mengelompokkan data berdasarkan nilai dalam kolom tertentu. Pernyataan *HAVING* digunakan untuk menerapkan kondisi ke hasil pengelompokan.

S. Prosedur dan Fungsi: SQL mendukung pembuatan prosedur dan fungsi yang dapat Anda gunakan untuk mengotomatisasi tugas-tugas yang sering Anda lakukan dalam database.

T. Tingkat Isolasi Transaksi: SQL mendukung berbagai tingkat isolasi transaksi, seperti *READ COMMITTED* dan *SERIALIZABLE*, yang memengaruhi cara transaksi bersamaan berinteraksi dan mengakses data.

U. Penyortiran dan Pengindeksan: Anda dapat menggunakan pernyataan *ORDER BY* untuk menyortir data berdasarkan kolom tertentu. Pengindeksan adalah teknik yang digunakan untuk meningkatkan kinerja pencarian data dalam tabel.

V. Perintah Pembatasan (*LIMIT* dan *OFFSET*): Perintah *LIMIT* digunakan untuk membatasi jumlah baris yang

dikembalikan dalam hasil kueri. Perintah OFFSET digunakan untuk memulai hasil kueri dari baris tertentu.

W. *Triggers*: *Trigger* SQL memungkinkan Anda untuk menjalankan perintah otomatis ketika peristiwa tertentu terjadi dalam database, seperti penambahan atau penghapusan data.

X. Kontrol Akses: SQL juga memiliki mekanisme pengaturan hak akses (*permissions*) yang memungkinkan Anda untuk mengontrol siapa yang memiliki izin untuk menjalankan perintah tertentu atau mengakses tabel dan kolom tertentu dalam *database*.

Y. Ekstensi SQL: Selain SQL standar, berbagai sistem database mungkin memiliki ekstensi atau perubahan sintaksis yang spesifik untuk *platform* mereka. Pengguna SQL perlu memahami perbedaan ini saat berinteraksi dengan berbagai sistem *database*.

Ini adalah beberapa dasar-dasar SQL yang membantu Anda memahami bagaimana berinteraksi dengan *database* relasional dan melakukan berbagai tugas terkait data.

3.3. JENIS-JENIS SQL

Jenis-jenis SQL mencakup beragam variasi bahasa pemrograman ini yang dikustomisasi untuk berbagai keperluan. SQL standar adalah dasar bagi hampir semua jenis SQL dan mencakup perintah-perintah fundamental seperti *SELECT*, *INSERT*, *UPDATE*, dan *DELETE*. Selain itu, terdapat T-SQL yang dikembangkan oleh Microsoft untuk produk seperti SQL *Server*, PL/SQL yang digunakan dalam *database Oracle*, serta varian-varian khusus untuk sistem *database open-source* seperti *MySQL*, *PostgreSQL*, dan *SQLite*. Selain itu, ada juga SQL yang dioptimalkan untuk tugas khusus, seperti SQL geospasial untuk data geografis, SQL berorientasi grafik untuk basis data grafik, SQL *timeseries*

untuk data berkaitan waktu, dan SQL dalam konteks data *warehouse*, bisnis intelijen, analisis data, dan banyak lagi. Memahami berbagai jenis SQL ini penting untuk mengelola data dengan efisien dalam berbagai konteks aplikasi dan analisis. Setiap jenis SQL tersebut memiliki karakteristik dan fitur khusus yang sesuai dengan tujuannya. SQL geospasial mendukung operasi terkait peta dan lokasi, sementara SQL berorientasi grafik memungkinkan manipulasi data dalam bentuk grafik. SQL *timeseries* cocok untuk data berdasarkan waktu, yang penting dalam analisis data yang berkaitan dengan perubahan seiring waktu. Selain itu, SQL dalam konteks data *warehouse* dan business intelligence mampu mengatasi permintaan kueri yang kompleks dan skalabilitas data besar. Dalam dunia yang semakin terkoneksi dan diperkaya dengan data, pemahaman tentang jenis-jenis SQL yang berbeda membantu para profesional dalam mengelola, mengakses, dan menganalisis data dengan efisien sesuai dengan kebutuhan dan konteks spesifiknya.

Terdapat beberapa jenis SQL berdasarkan penggunaan dan spesifikasinya:

A. SQL Standar: Ini adalah bentuk SQL yang paling umum dan diakui secara luas. Ini mengikuti standar ANSI (*American National Standards Institute*) dan diterima oleh banyak sistem database relasional. SQL standar mencakup perintah dasar seperti *SELECT*, *INSERT*, *UPDATE*, dan *DELETE*.
B. T-SQL (*Transact*-SQL): T-SQL adalah varian SQL yang dikembangkan oleh Microsoft untuk digunakan dalam produk mereka, seperti *Microsoft* SQL *Server*. Ini mencakup ekstensi yang memungkinkan Anda untuk mengembangkan prosedur tersimpan, fungsi, dan mengelola transaksi dengan lebih efektif.
C. PL/SQL (*Procedural Language*/SQL): PL/SQL adalah ekstensi SQL yang dikembangkan oleh *Oracle*

Corporation untuk digunakan dalam sistem database *Oracle*. Ini memungkinkan Anda untuk menggabungkan kode SQL dengan kode pemrograman prosedural, seperti blok *IF-ELSE* dan *LOOP*, dalam prosedur tersimpan dan fungsi.

D. MySQL SQL: MySQL adalah sistem *database open-source* yang memiliki dialek SQL khususnya. Ini mencakup beberapa perintah tambahan dan ekstensi, serta dukungan untuk penyimpanan prosedur tersimpan dan fungsi.

E. *PostgreSQL* SQL: *PostgreSQL* adalah sistem *database open-source* lainnya yang memiliki dialek SQL yang spesifik. Ini terkenal karena dukungannya terhadap tipe data yang kaya dan fungsionalitas yang kuat, termasuk berbagai jenis indeks dan penyimpanan prosedur tersimpan.

F. *SQLite* SQL: SQLite adalah *database* ringan yang sering digunakan dalam pengembangan aplikasi seluler dan embedded. Ini mendukung sebagian besar perintah SQL standar, meskipun memiliki beberapa perbedaan kecil dalam dukungan fitur tertentu.

G. *NoSQL Query Language*: Meskipun NoSQL (*Not Only SQL*) adalah konsep yang berbeda, beberapa sistem NoSQL memiliki bahasa kueri mereka sendiri, seperti *MongoDB Query Language* (MQL) untuk MongoDB dan *Cypher Query Language* untuk database berorientasi grafik seperti Neo4j.

H. Ada juga berbagai ekstensi SQL yang ditambahkan oleh sistem database tertentu, seperti *Oracle* SQL *Developer*, *Microsoft* SQL *Server Management Studio*, atau MySQL *Workbench*. Ini adalah alat manajemen database yang memungkinkan Anda untuk berinteraksi dengan database menggunakan antarmuka visual dan perintah SQL.

Setiap jenis SQL ini memiliki sintaksis dan fungsionalitas yang sedikit berbeda, tergantung pada sistem database yang digunakan. Namun, dasar-dasar SQL, seperti *SELECT*, *INSERT*, *UPDATE*, dan *DELETE*, umumnya konsisten di seluruh jenis dan sistem *database* yang berbeda.

3.4. PENGGUNAAN SQL DALAM PRAKTIK

Penggunaan SQL dalam praktik adalah landasan utama bagi pengelolaan dan analisis data dalam berbagai sektor dan profesi. SQL, atau *Structured Query Language*, adalah bahasa pemrograman yang digunakan untuk mengelola database relasional. Dalam dunia nyata, SQL memainkan peran penting dalam manajemen data, pengambilan informasi, analisis data, pengembangan perangkat lunak, dan banyak lagi. Profesional di berbagai bidang, mulai dari pengembang perangkat lunak hingga analis data, mengandalkan SQL untuk menjalankan berbagai tugas yang berkaitan dengan data, mulai dari pencarian dan pengambilan data hingga analisis mendalam yang mendukung pengambilan keputusan bisnis yang tepat. Penggunaan SQL dalam praktik juga mencerminkan pentingnya pemahaman tentang bahasa ini dalam menghadapi tantangan sehari-hari, seperti manajemen data yang semakin kompleks, kebutuhan akan keamanan data yang ketat, dan permintaan akan analisis data yang lebih mendalam. SQL membantu memungkinkan organisasi untuk mengorganisir, menyimpan, mengambil, dan menganalisis data dengan cara yang efisien, sehingga memfasilitasi pengambilan keputusan yang didukung oleh fakta dan analisis yang kuat. Dengan demikian, pemahaman praktis tentang penggunaan SQL adalah salah satu kunci sukses dalam menghadapi era informasi yang semakin terhubung dan penuh dengan data.

Tidak hanya dalam bidang bisnis, SQL juga memiliki peran penting di berbagai sektor lainnya. Dalam ilmu pengetahuan, SQL digunakan untuk mengelola data penelitian dan eksperimen, sedangkan dalam sektor kesehatan, SQL membantu dalam pengelolaan data pasien dan analisis data klinis. Pemerintah menggunakan SQL untuk menyimpan dan mengakses data yang penting untuk kebijakan publik dan manajemen sumber daya. Di dunia pendidikan, SQL digunakan untuk mengelola catatan akademis dan melakukan analisis data siswa. Dengan demikian, pemahaman praktis tentang penggunaan SQL membantu individu dan organisasi untuk efektif dan efisien mengelola data, memahami tren, dan mengambil keputusan yang lebih baik dalam berbagai konteks. Penggunaan SQL dalam praktik juga memberikan kemungkinan integrasi data dari berbagai sumber yang berbeda. Hal ini memungkinkan organisasi untuk menggabungkan data dari sistem yang berbeda, seperti aplikasi perusahaan, sistem manajemen pelanggan, dan penyedia layanan pihak ketiga, untuk mendapatkan pandangan data yang lebih holistik dan berharga. SQL juga memberikan fleksibilitas dalam mengakses dan mengolah data secara real-time, yang sangat penting dalam bisnis yang beroperasi dengan kecepatan tinggi dan ketepatan waktu.

Dalam dunia teknologi yang terus berkembang, pemahaman SQL merupakan elemen utama dalam peran seperti pengembang perangkat lunak, administrator database, dan analis data. SQL juga menjadi elemen penting dalam bisnis yang mengandalkan data sebagai aset strategis. Seiring dengan perkembangan teknologi data yang semakin maju, pengetahuan dan penggunaan SQL akan terus menjadi aset berharga dalam menghadapi tantangan dan peluang yang ditawarkan oleh dunia data yang terus berkembang.

Artikel ini akan membahas lebih dalam tentang penggunaan SQL dalam berbagai aplikasi praktis.

Penggunaan SQL dalam praktik sangat luas dan mendalam, berikut adalah beberapa cara umum di mana SQL digunakan dalam dunia nyata:

A. Manajemen Data: SQL digunakan untuk membuat, mengelola, dan menjaga data dalam database. Ini mencakup pembuatan tabel, penambahan, perubahan, dan penghapusan data, serta pengelolaan indeks, kunci utama, dan kunci asing.
B. Pengambilan Data: SQL digunakan untuk mengambil data dari database. Pengguna dapat menentukan kriteria pencarian, menggabungkan data dari beberapa tabel, dan mengurutkan data sesuai kebutuhan mereka.
C. Analisis Data: SQL memungkinkan para analis data untuk menyusun kueri yang kompleks untuk mengidentifikasi tren, mencari pola, dan menghasilkan laporan yang memberikan wawasan bisnis yang berharga.
D. Pengembangan Aplikasi: SQL digunakan dalam pengembangan perangkat lunak untuk memungkinkan aplikasi berinteraksi dengan basis data. Ini mencakup operasi dasar seperti menambahkan, mengubah, dan menghapus data.
E. Pemrosesan Transaksi: SQL digunakan untuk menjalankan dan mengelola transaksi dalam sistem database, yang penting dalam aplikasi keuangan dan perdagangan yang melibatkan transaksi.
F. Manajemen Keamanan: SQL digunakan untuk mengelola izin dan hak akses pengguna ke tabel, kolom, atau database. Ini memastikan data terlindungi dan hanya dapat diakses oleh mereka yang memiliki otorisasi.
G. Manajemen Kinerja: SQL digunakan untuk mengoptimalkan kinerja database dengan menciptakan

indeks yang sesuai, merancang skema basis data yang efisien, dan menulis kueri yang efisien.

H. Pelaporan Bisnis: SQL digunakan untuk mengambil data dan menghasilkan laporan bisnis yang dapat digunakan oleh manajemen dalam pengambilan keputusan.

I. Integrasi Sistem: SQL digunakan untuk mengintegrasikan berbagai sistem dan aplikasi yang memerlukan akses ke data yang disimpan dalam database.

J. Pemeliharaan Database: SQL digunakan untuk melakukan tugas pemeliharaan, seperti backup, pemulihan data, dan peningkatan versi database.

K. Eksplorasi Data: SQL digunakan dalam eksplorasi dan analisis data mentah, termasuk penggalian data besar dan analisis statistik.

Penggunaan SQL dalam praktik sangat relevan di berbagai sektor, termasuk bisnis, ilmu pengetahuan, pemerintahan, kesehatan, pendidikan, dan banyak lagi. Profesional teknologi informasi, analis data, administrator database, dan pengembang perangkat lunak mengandalkan SQL untuk mengelola dan menganalisis data dalam berbagai konteks. Dengan teknologi semakin berkembang, SQL tetap menjadi alat yang sangat penting dalam pengelolaan dan analisis data.

3.5. SOAL LATIHAN SQL

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) Apa yang dimaksud dengan SQL? Jelaskan peran utama SQL dalam pengelolaan *database*.
2) Jelaskan perbedaan antara SQL standar dan T-SQL. Di sistem *database* mana T-SQL biasanya digunakan?

3) Sebutkan beberapa sistem database yang memiliki varian SQL khusus dan jelaskan bagaimana SQL mereka berbeda dari SQL standar.
4) Bagaimana SQL digunakan dalam manajemen data sehari-hari? Berikan contoh tugas yang dapat diselesaikan dengan SQL.
5) Mengapa pemahaman SQL penting dalam pengembangan aplikasi? Berikan contoh cara SQL digunakan dalam pengembangan perangkat lunak.

BAB IV
IMPLEMENTASI DATA DEFINITION LANGUAGE (DDL)

Implementasi Data Definition Language (DDL) adalah tahap kunci dalam pengembangan dan pemeliharaan sistem basis data. DDL adalah bagian dari bahasa SQL yang digunakan untuk mendefinisikan struktur basis data, termasuk tabel, indeks, kunci asing, dan konstrain. Proses implementasi DDL mencakup pembuatan, pengubahan, dan penghapusan objek-objek basis data sesuai dengan kebutuhan aplikasi dan bisnis. DDL memungkinkan pengelola basis data dan pengembang untuk merancang struktur basis data yang memenuhi persyaratan fungsional dan performa, serta memastikan integritas data. Pemahaman yang mendalam tentang implementasi DDL adalah esensial dalam membangun sistem basis data yang efisien, andal, dan sesuai dengan tujuan aplikasi, dan merupakan komponen penting dalam manajemen data yang sukses. Implementasi Data Definition Language (DDL) juga berperan penting dalam menjaga konsistensi dan keamanan data. Dengan DDL, pengguna dapat menetapkan batasan dan aturan yang diperlukan untuk data yang disimpan dalam basis data, seperti konstrain integritas referensial dan peraturan validasi. DDL juga memungkinkan pengelola basis data untuk mendefinisikan peran dan hak akses pengguna ke basis data, sehingga menjaga privasi dan keamanan data yang sensitif. Selain itu, perubahan struktur basis data yang dilakukan melalui DDL harus dilakukan dengan hati-hati, karena dapat berdampak signifikan pada aplikasi yang bergantung pada basis data tersebut. Oleh karena itu, pemahaman yang baik tentang implementasi DDL adalah kunci dalam memastikan bahwa basis data beroperasi sesuai dengan kebutuhan bisnis dan keamanan, serta dapat

menghadapi perubahan dan pertumbuhan yang mungkin terjadi di masa depan.

4.1. PENGANTAR DDL

DDL (*Data Definition Language*) adalah komponen utama dalam SQL yang digunakan untuk mendefinisikan, mengelola, dan mengatur struktur basis data. Perintah DDL mencakup perintah seperti *CREATE*, *ALTER*, dan *DROP* yang memungkinkan pengguna untuk membuat tabel, indeks, dan berbagai objek basis data, mengubah struktur objek yang ada, dan menghapus objek tersebut jika diperlukan(Pamungkas, 2017). DDL juga digunakan untuk mengatur batasan integritas data, seperti kunci unik, kunci utama, dan kunci asing. Perintah DDL sangat penting dalam perancangan dan pemeliharaan basis data, karena mereka mengatur kerangka kerja yang digunakan untuk menyimpan dan mengakses data. DDL memungkinkan administrator basis data untuk merancang dan mengubah struktur basis data sesuai dengan kebutuhan aplikasi dan aturan bisnis, sehingga memberikan kontrol yang kuat dalam manajemen data. Selain peran utamanya dalam mendefinisikan struktur basis data, DDL juga memainkan peran penting dalam menjaga integritas data. DDL digunakan untuk menentukan batasan dan peraturan yang harus diikuti oleh data yang dimasukkan ke dalam tabel, memastikan konsistensi dan validitas data. Selain itu, DDL memungkinkan pengguna untuk melakukan perubahan struktural, seperti menambahkan kolom baru atau mengubah tipe data, sehingga dapat menyesuaikan basis data dengan perkembangan kebutuhan aplikasi. Namun, perubahan DDL perlu dielaborasi dengan hati-hati, karena dapat memengaruhi data yang sudah ada dan harus dipertimbangkan dengan cermat dalam proses pemeliharaan dan pengembangan sistem basis data. DDL

adalah salah satu elemen kunci dalam SQL yang membantu dalam mengelola, mengatur, dan memastikan integritas data dalam basis data.

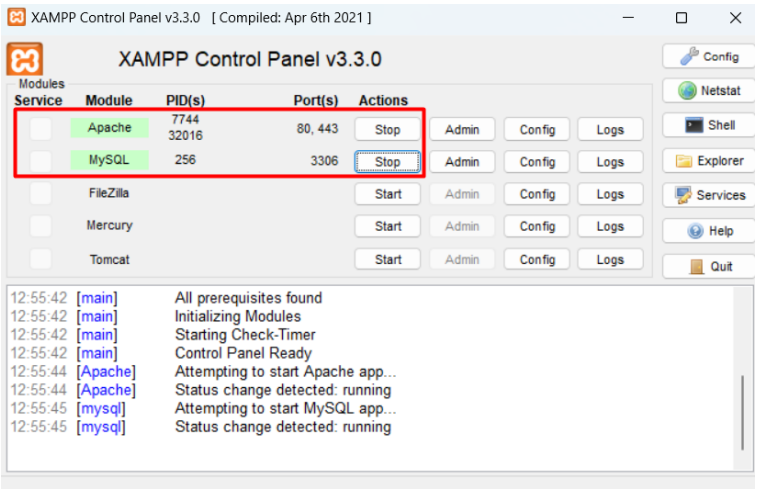
Selain itu, DDL juga terkait dengan keamanan basis data. Perintah DDL digunakan untuk mengendalikan izin akses, memberikan hak istimewa kepada pengguna atau peran tertentu, serta mengatur objek basis data seperti tabel, *view*, dan prosedur. Ini memungkinkan administrator basis data untuk mengatur siapa yang dapat melakukan perubahan struktural pada basis data dan mengontrol akses ke data dengan lebih cermat. Penting untuk diingat bahwa perintah DDL memengaruhi struktur basis data secara langsung, dan kesalahan dalam perintah DDL dapat berdampak besar pada integritas dan performa basis data. Oleh karena itu, perubahan DDL perlu diuji secara teliti dan dielaborasi dalam lingkungan pengembangan atau pengujian sebelum diaplikasikan ke basis data produksi. DDL adalah elemen kunci dalam manajemen basis data yang memungkinkan untuk mendefinisikan, mengelola, dan mengamankan struktur basis data sesuai dengan kebutuhan bisnis dan aplikasi yang berkembang.

4.2. PEMBUATAN DATABASE

Pembuatan *database* adalah tahap awal yang krusial dalam pengembangan sistem informasi yang berkaitan dengan penyimpanan dan pengelolaan data. Proses ini melibatkan perancangan struktur database, yang mencakup pembuatan tabel, kolom, dan relasi antar tabel. Selain itu, dalam pembuatan *database*, pengguna juga harus menentukan jenis data yang akan disimpan, aturan integritas, serta indeks yang diperlukan untuk mempercepat akses data. Setelah struktur database ditentukan, langkah selanjutnya adalah mengimplementasikan database tersebut menggunakan perintah SQL atau alat pengelolaan database

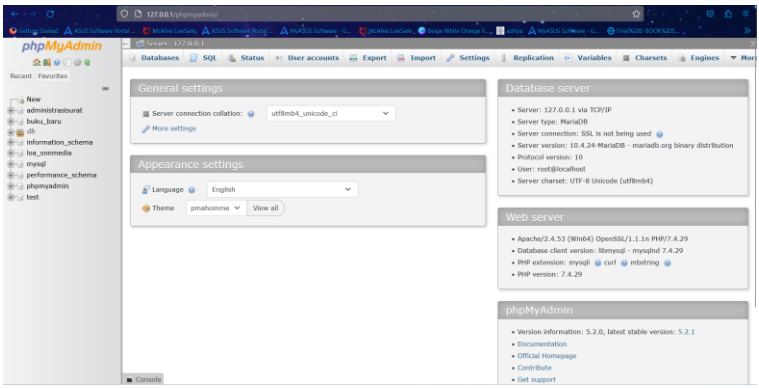
yang sesuai. *Database* yang baik dirancang untuk memenuhi kebutuhan bisnis, menyediakan pengelolaan data yang efisien, serta memungkinkan akses data yang mudah dan akurat. Proses pembuatan *database* memainkan peran kunci dalam memastikan keberhasilan dan kinerja sistem informasi yang bergantung pada data.

Impementasi pembuatan *database* kita akan membuat database dengan menggunakan MySql. Sebelumnya kita harus melakukan instalasi XAMPP. XAMPP merupakan sebuah paket perangkat lunak sumber terbuka yang menggabungkan komponen-komponen ini ke dalam satu paket untuk memungkinkan pengembang atau administrator sistem untuk dengan mudah membuat lingkungan pengembangan web lokal atau server web untuk menguji, mengembangkan, dan menjalankan aplikasi web. Untuk *install* XAMPP daapt di-unduh pada URL https://www.apachefriends.org/download.html. Setelah selesai diunduh dan instalasi kita akan mengaktifkan XAMPP tersebut. Silahkan dibuka XAMPP, maka akan muncul tampilan berikut ini.



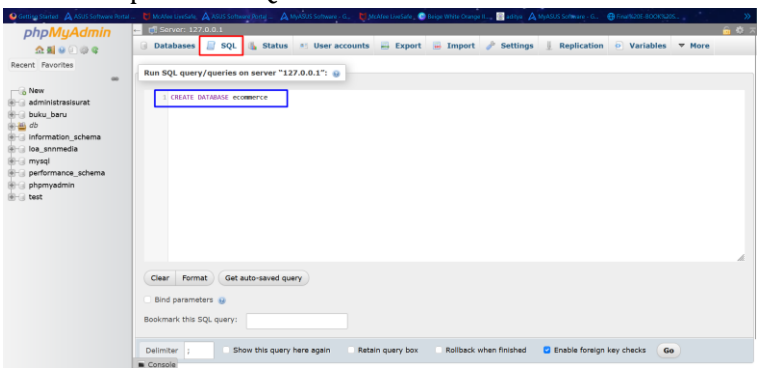
Gambar 4.1. Tampilan XAMPP

Selanjutnya aktifkan atau *start* bagian apache dan mysql sehingga seperti gambar diatas. Selanjutnya kita akan membuat *database MySql* menggunakan web browser ketik alamat URL berikut http://127.0.0.1/phpmyadmin/ maka akan muncul gambar berikut.



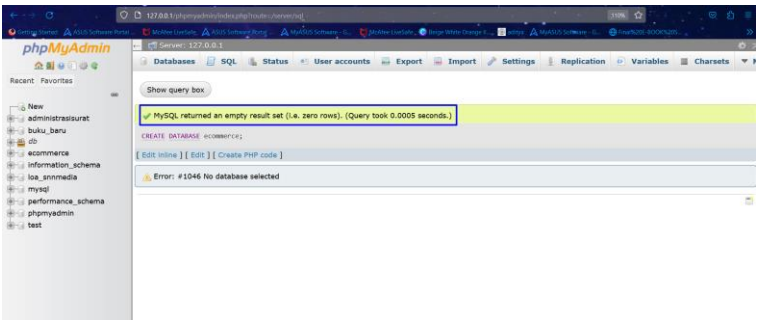
Gambar 4.2. Membuka *Database* MySql

Selanjutnya pilih toolbar SQL, disana kita akan membuat perintah SQL.



Gambar 4.3. Perintah Membuat *Database* MySql

Buat perintah SQL yaitu “CREATE DATABASE ecommece” selanjutnya pilih tombol “GO” maka akan muncul hasil seperti gambar berikut ini.



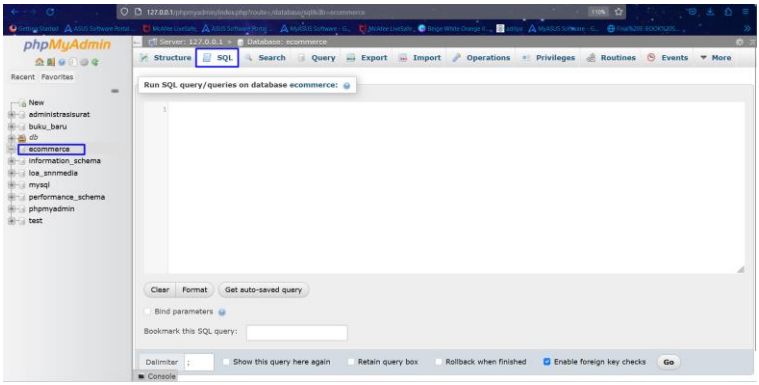
Gambar 4.4. Hasil *Execute* Perintah Membuat *Database* MySql

Hasil *execute* perintah diatas menunjukan kalua database yang kita buat dengan nama "***ecommerce***" tsudah berhasil dibuat. Sekarang kita telah mempunyai *database* untuk kita pergunakan dalam tahapan selanjutnya.

4.3. PEMBUATAN TABEL

Pembuatan tabel adalah tahap penting dalam perancangan basis data yang melibatkan pengaturan kerangka kerja untuk menyimpan dan mengatur data. Dalam proses ini, pengguna harus menentukan kolom-kolom yang akan ada dalam tabel, menetapkan tipe data untuk setiap kolom, dan menentukan batasan-batasan seperti kunci utama (*primary key*) atau kunci asing (*foreign key*) yang memungkinkan relasi antara tabel. Desain tabel yang baik harus mempertimbangkan struktur data yang efisien dan logika bisnis yang terkait dengan data tersebut. Setelah desain tabel selesai, langkah selanjutnya adalah mengimplementasikan tabel tersebut dengan perintah SQL atau menggunakan alat manajemen basis data. Tabel yang dibuat harus memenuhi kebutuhan aplikasi atau sistem dan memastikan konsistensi, integritas, serta akses data yang efisien.

Impementasi pembuatan tabel kita akan melanjutkan pada *database* yang telah kita buat sebelumnya. Pada menu sebelah kiri kita pilih dahulu nama *database* kita dan selanjutnya pilih *toolbar SQL*, seperti gambar berikut ini.



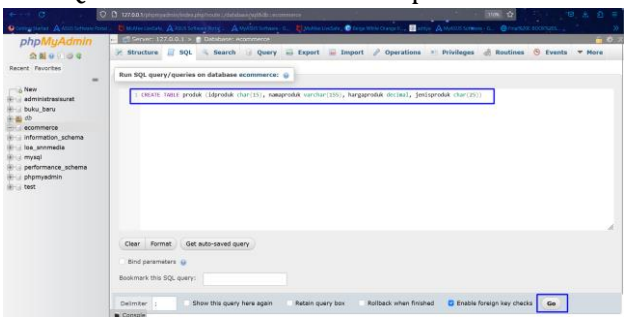
Gambar 4.5. Tampilan Memilih *Database Ecommerce*

Proses selanjutnya kita akan membuat tabel dengan deskripsi sebagai berikut.

Tabel 4.1. Tabel Produk

Nama Field	Tipe Data	Length
idproduk	*Char*	10
namaproduk	*Varchar*	155
hargaproduk	*Decimal*	
jenisproduk	*Char*	25

Perintah SQL untuk membuat tabel seperti berikut ini.



Gambar 4.6. Perintah Membuat Tabel Produk

Hasil *execute* perintah diatas seperti berikut ini.

Gambar 4.7. Hasil *Execute* Perintah Membuat Tabel

Dari hasil *execute* gambar diatas kita berhasil membuat tabel dengan nama produk, selanjutnya kita akan membuat 3 tabel Kembali dengan deskripsi sebagai berikut.

Tabel 4.2. Tabel Pelanggan

Nama Field	Tipe Data	Length
idpelanggan	*Char*	10
namapelanggan	*Varchar*	155
alamatpelanggan	*Text*	
nomorhppelanggan	*Char*	17

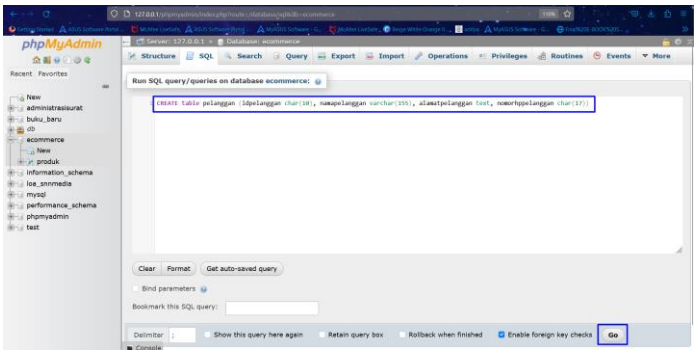
Tabel 4.3. Tabel Pemasok

Nama Field	Tipe Data	Length
idpemasok	*Char*	10
namapemasok	*Varchar*	155
alamatpemasok	*Text*	
nomorhppemasok	*Char*	17

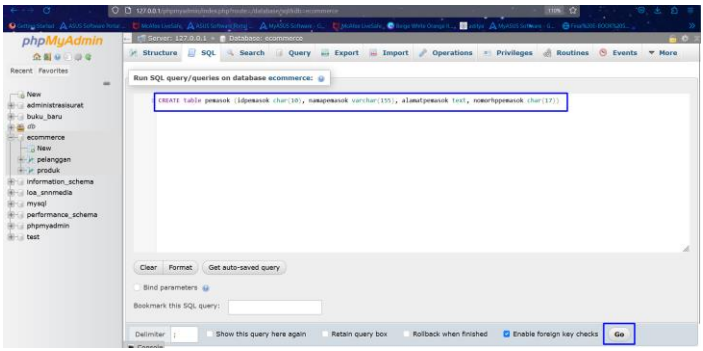
Tabel 4.4. Tabel Barang

Nama Field	Tipe Data	Length
idbarang	*Char*	10
namabarang	*Varchar*	155

Perintah SQL untuk membuat tabel pelanggan dan pemasok seperti gambar berikut ini.



Gambar 4.8. Perintah Membuat Tabel Pelanggan



Gambar 4.9. Perintah Membuat Tabel Pemasok

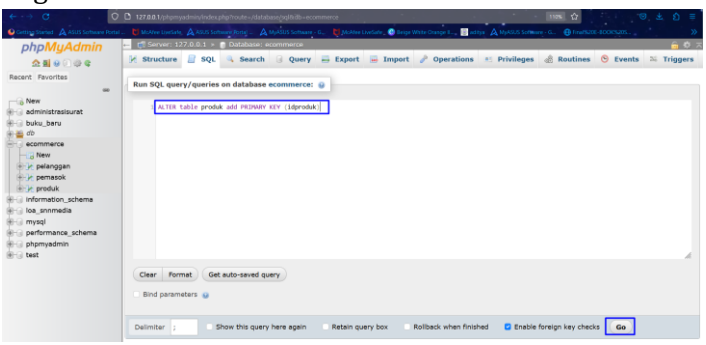
Hasil *execute* setelah menambahkan 2 tabel seperti berikut ini.

Gambar 4.10. Hasil *Execute* Menambah 2 Tabel

4.4. MENGUBAH TABEL

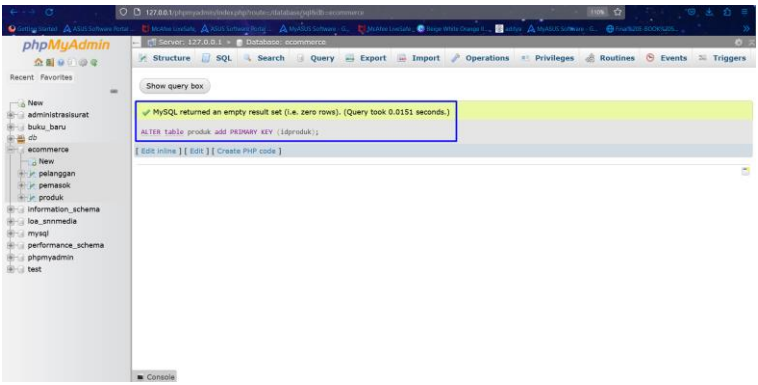
Mengubah tabel adalah proses penting dalam pengelolaan basis data yang memungkinkan pengguna untuk mengadaptasi struktur tabel sesuai dengan kebutuhan bisnis dan perkembangan aplikasi. Perubahan tabel dapat mencakup penambahan, penghapusan, atau perubahan kolom, pengubahan tipe data, serta penambahan atau penghapusan indeks. Selain itu, pengguna juga dapat mengubah batasan integritas data atau menentukan kunci unik dan kunci asing yang lebih sesuai. Dalam konteks pengembangan dan pemeliharaan sistem, kemampuan untuk mengubah tabel dengan aman dan efisien adalah kunci dalam memastikan konsistensi dan fleksibilitas dalam basis data. Proses perubahan tabel harus dielaborasi dengan hati-hati dan diuji secara teliti untuk meminimalkan dampak negatif pada data yang sudah ada, sehingga memastikan bahwa struktur tabel tetap sesuai dengan kebutuhan dan perkembangan bisnis.

Impementasi mengubah tabel kita akan melanjutkan pada *database* yang telah kita buat sebelumnya. Kita akan merubah tabel produk dimana tabel produk yang kita buat belum ada *primary key*, kita akan menambahkan primary key dalam tabel produk pada field idproduk, perintah SQL sebagai berikut.



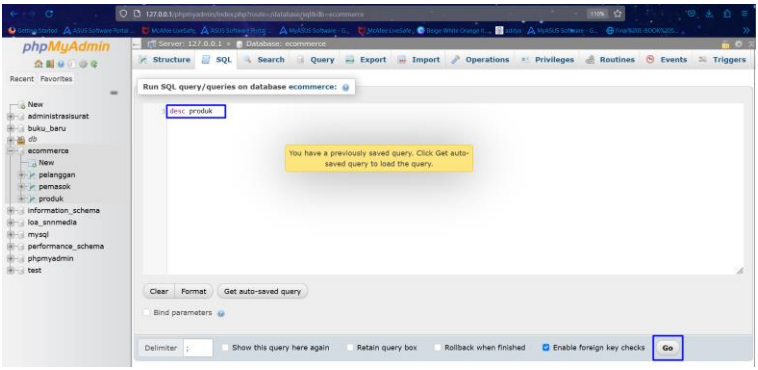
Gambar 4.10. Perintah Menambah Primary Key

Setelah itu kita *execute* dan hasilnya seperti berikut ini.



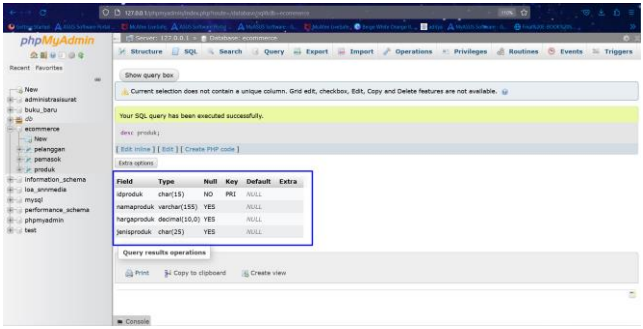
Gambar 4.11. Hasil *Execute* Menambah Primary Key

Selanjutnya tambahkan perintah SQL berikut ini



Gambar 4.12. Perintah SQL Melihat Deskripsi Tabel

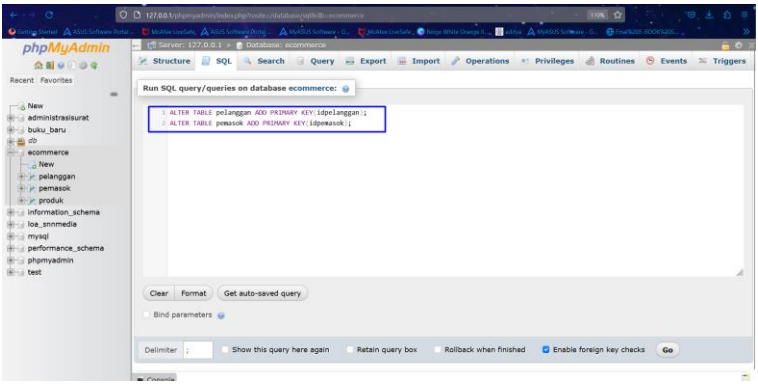
Setelah di *execute* hasilnya sebagai berikut.



Gambar 4.13. Hasil *execute* Melihat Deskripsi Tabel

Gambar diatas kita bisa melihat deskripsi tabel produk yang telah ubah, dimana ada kata **PRI** pada kolom key untuk idproduk.

Selanjutnya silahkan tambahkan primary key untuk tabel pelanggan dengan field idpelanggan, dan tabel pemasok dengan field idpemasok. Perintah SQL nya sebagai berikut.



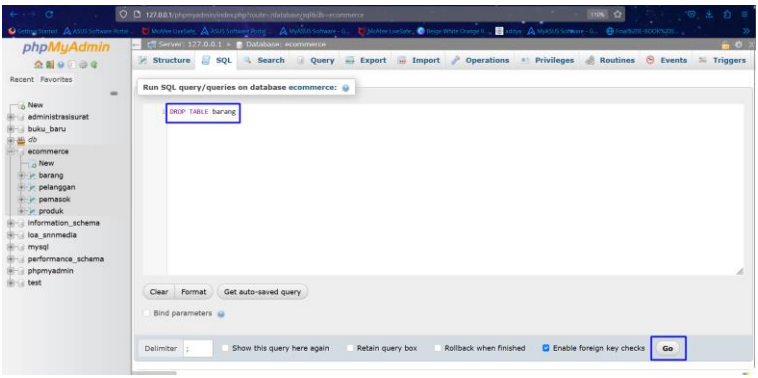
Gambar 4.14. Hasil *Execute* Menambah Primary Key 2 Tabel

4.5. MENGHAPUS TABEL

Menghapus tabel adalah tindakan yang perlu dilakukan dengan sangat hati-hati dalam pengelolaan basis data. Penghapusan tabel berarti kehilangan data yang terkandung di dalamnya, dan oleh karena itu, tindakan ini harus dipertimbangkan secara serius. Sebelum menghapus tabel, penting untuk memastikan bahwa data yang ada telah dicadangkan (*backup*) untuk menghindari kehilangan data yang tidak dapat dipulihkan. Selain itu, perlu memastikan bahwa tidak ada referensi atau relasi dari tabel lain yang bergantung pada tabel yang akan dihapus. Setelah yakin, perintah SQL seperti *DROP TABLE* dapat digunakan untuk menghapus tabel. Menghapus tabel dapat dilakukan jika tabel tersebut sudah tidak diperlukan, tidak relevan, atau

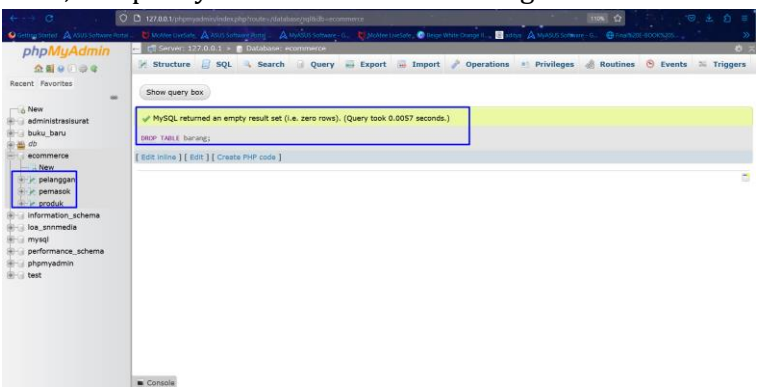
untuk tujuan pemulihan ruang penyimpanan. Namun, kebijakan penghapusan tabel harus diatur secara hati-hati untuk menghindari kehilangan data yang penting atau kesalahan yang tidak dapat diperbaiki.

Proses selanjutnya kita akan menghapus tabel dengan nama barang, perintah SQL nya sebagai berikut.



Gambar 4.15. Perintah SQL Menghapus Tabel

Selanjutnya *execute* dengan memilih tombol **GO**, akan muncul sebuah pesan apakah kita yakin akan menghapus tabel, kita pilih yes dan muncul hasil sebagai berikut.



Gambar 4.17. Hasil *Execute* Perintah Menghapus Tabel

4.6. SOAL LATIHAN DDL

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) Buat tabel berikut ini.

Tabel 4.5. Tabel Pegawai

Nama Field	Tipe Data	Length
idpegawai	*Char*	10
namapegawai	*Varchar*	155
alamatpegawai	*Text*	
emailpegawai	*Text*	
Nomorhp	*Char*	17

2) Selanjutnya buat idpegawai menjadi *primary key*.

BAB V
IMPLEMENTASI DATA *MANIPULATION LANGUAGE* (DML)

Implementasi Data *Manipulation Language* (DML) adalah tahap krusial dalam mengelola dan memanipulasi data dalam sistem basis data. DML adalah bagian dari bahasa SQL yang digunakan untuk menambah, mengubah, dan mengambil data dalam basis data. Ini berarti DML memungkinkan pengguna untuk melakukan operasi seperti penambahan, penghapusan, pembaruan, dan penarikan data sesuai dengan kebutuhan aplikasi. DML memainkan peran penting dalam menjaga data yang konsisten dan akurat, serta dalam mendukung berbagai tugas seperti analisis data, laporan, dan pembaruan informasi. Pemahaman yang mendalam tentang implementasi DML adalah esensial dalam pengembangan aplikasi yang efisien dan berkinerja tinggi, serta dalam memenuhi kebutuhan bisnis yang terus berubah terkait dengan data. Selain itu, implementasi Data *Manipulation Language* (DML) juga memiliki dampak yang signifikan pada produktivitas dan fleksibilitas dalam menggunakan sistem basis data. Dengan DML, pengguna memiliki kemampuan untuk mengakses dan memanipulasi data dengan cara yang sesuai dengan kebutuhan mereka, memungkinkan analis data dan pengembang aplikasi untuk menggali wawasan berharga dari informasi yang tersedia. DML juga berperan dalam menjaga integritas referensial dan konsistensi data dengan menerapkan aturan dan konstrain yang sesuai. Oleh karena itu, pemahaman yang baik tentang implementasi DML adalah kunci dalam memastikan bahwa data dapat dikelola, dianalisis, dan dimanfaatkan dengan efektif, serta mendukung tujuan bisnis yang beragam di berbagai sektor. Dalam era di mana data memiliki peran

sentral dalam pengambilan keputusan, implementasi DML menjadi elemen penting dalam mengoptimalkan penggunaan data dan memenuhi tantangan dalam dunia teknologi dan bisnis yang terus berkembang.

5.1. PENGANTAR DML

Data *Manipulation Language* (DML) adalah bagian dari bahasa SQL (*Structured Query Language*) yang digunakan untuk mengelola dan memanipulasi data dalam sebuah basis data. DML memungkinkan pengguna untuk melakukan operasi-operasi seperti penyisipan, pembaruan, penghapusan, dan pencarian data dalam tabel(Rosa & Salahudin, 2014). Melalui perintah-perintah DML, pengguna dapat mengubah konten basis data, menjalankan kueri untuk mengambil informasi yang mereka butuhkan, dan memastikan konsistensi data. DML memainkan peran kunci dalam aplikasi dan sistem database, memungkinkan interaksi yang efisien dengan data yang disimpan dalam basis data. Selain itu, DML juga memungkinkan transaksi dalam *database*. Transaksi adalah serangkaian perintah DML yang harus dieksekusi sebagai satu kesatuan. Ini berarti bahwa jika ada kesalahan dalam salah satu perintah dalam transaksi, maka seluruh transaksi akan dibatalkan (*rollback*), sehingga tidak ada perubahan yang diterapkan pada data. Sebaliknya, jika semua perintah dalam transaksi dieksekusi dengan sukses, maka perubahan data akan dikonfirmasi (*commit*). Ini adalah fitur penting untuk menjaga konsistensi data dan keandalan dalam lingkungan *database*.

Selain itu, DML juga memungkinkan pengguna untuk membuat, mengubah, dan menghapus indeks yang telah disebutkan sebelumnya dalam konteks *database*. Indeks membantu meningkatkan kinerja operasi pencarian data, dan penggunaan DML dalam hal ini memberikan pengguna

kontrol penuh atas bagaimana indeks dibangun dan dikelola. Dalam esensi, DML adalah alat yang sangat penting dalam manajemen data dan pengembangan aplikasi yang bergantung pada basis data. Penggunaannya memungkinkan pengguna untuk berinteraksi dengan data dengan cara yang fleksibel dan kuat, dengan potensi untuk memaksimalkan kinerja operasi dan menyediakan akses yang cepat dan efisien ke informasi yang diperlukan. DML merupakan bagian kunci dari SQL, yang digunakan dalam berbagai sistem *database* relasional dan *non*-relasional.

5.2. MENYIMPAN DATA

Untuk menyimpan data menggunakan SQL, kita dapat menggunakan perintah SQL *INSERT*. Perintah *INSERT* memungkinkan Anda untuk memasukkan data baru ke dalam sebuah tabel dalam basis data. Anda harus menentukan tabel yang akan diisi dan kolom-kolom yang akan diisi dengan nilai-nilai yang sesuai. Setelah menjalankan perintah *INSERT*, data yang baru saja dimasukkan akan menjadi bagian dari tabel dalam basis data. Perintah *INSERT* dapat digunakan untuk menyimpan satu baris data sekaligus, atau Anda dapat menggunakan perintah *INSERT INTO* dengan pernyataan *VALUES* untuk menyimpan beberapa baris data dalam satu operasi. Juga, jika Anda memiliki data dalam bentuk hasil kueri lainnya, Anda dapat menggunakan perintah *INSERT INTO* dengan *SELECT* untuk mengambil data dari kueri tersebut dan menyimpannya ke dalam tabel yang sesuai. Penting untuk memastikan bahwa data yang dimasukkan sesuai dengan aturan dan batasan yang telah ditetapkan dalam skema basis data Anda, seperti tipe data, kunci asing, dan batasan unik. Dengan menggunakan perintah *INSERT*, Anda dapat dengan mudah mengisi dan menyimpan data dalam database sesuai

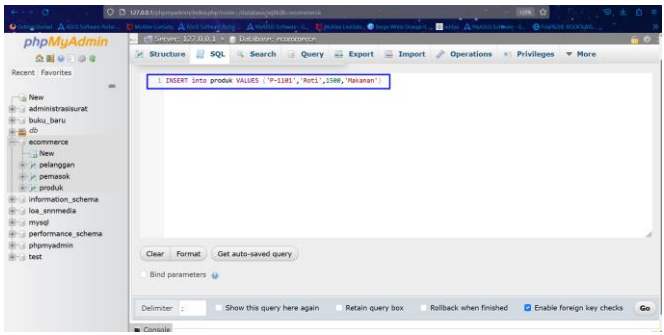
dengan kebutuhan aplikasi atau sistem yang Anda kembangkan.

Kita akan mencoba memasukan data dalam tabel produk dengan rincian data sebagai berikut.

Tabel 5.1. Data Produk

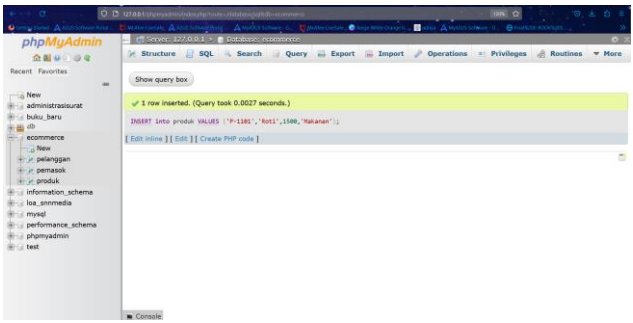
ID Produk	Nama Produk	Harga Produk	Jenis Produk
P-1101	Roti	1500	Makanan
P-1102	Air Mineral	3000	Minuman
P-1103	Sabun	11500	Lainnya
P-1104	Gula	35000	Sembako
P-1105	Beras	125000	Sembako

Dari data diatas kita akan mencoba memasukan data pertama dengan id produk P-1101, perintah SQL sebagai berikut.



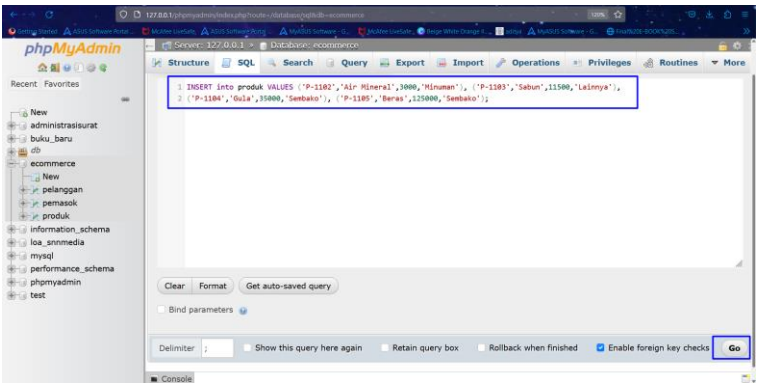
Gambar 5.1. Perintah SQL Menyimpan Data

Selanjutnya kita *execute* dan hasil *execute* sebagai berikut.



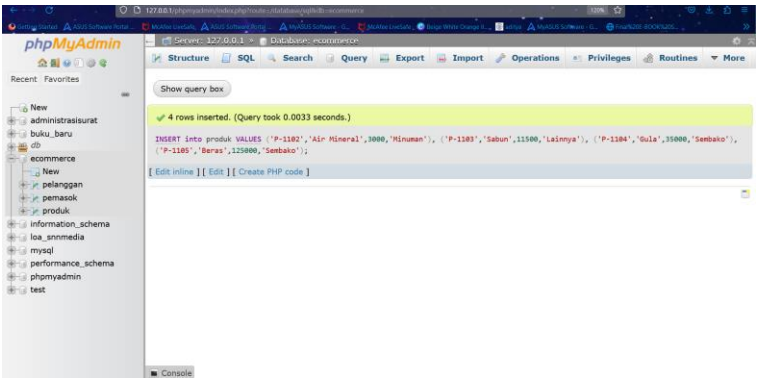
Gambar 5.2. Hasil *Execute* Menyimpan Data

Hasil *execute* perintah diatas kita berhasil menyimpan data dalam tabel produk. Selanjutnya kita akan menyimpan 4 data sekaligus dengan menggunakan 1 perintah SQL, perintah nya sebagai berikut.



Gambar 5.3. Perintah SQL Menyimpan Beberapa Data

Selanjutnya kita *execute* dan hasil *execute* sebagai berikut.



Gambar 5.4. Hasil *Execute* Menyimpan Beberapa Data

Selanjutnya silahkan tambahkan data untuk tabel pelanggan dan pemasok seperti berikut ini.

Tabel 5.2. Data Pelanggan

ID Pelanggan	Nama Pelanggan	Alamat Pelanggan	Nomor HP Pelanggan
PL001	Budi	Jakarta	0812123456
PL002	Joko	Bandung	0812123457
PL003	Halim	Cirebon	0812123458
PL004	Ahmad	Manado	0812123459
PL005	Yuda	Palembang	0812123450

Tabel 5.3. Data Pemasok

ID Pemasok	Nama Pemasok	Alamat Pemasok	Nomor HP Pemasok
PM1110	CV. Jaya	Jambi	0813567891
PM1111	PT. Mandiri	Lampung	0813567892
PM1112	CV. Maju	Tegal	0813567893
PM1113	PT. Berlian	Yogyakarta	0813567894
PM1114	CV. Sarana	Riau	0813567895

5.3. MENGUBAH DATA

Untuk mengubah data dalam database menggunakan SQL, Anda dapat menggunakan perintah SQL *UPDATE*. Perintah *UPDATE* memungkinkan Anda untuk memperbarui nilai-nilai yang ada dalam tabel dengan nilai-nilai baru yang Anda tentukan. Anda harus menentukan tabel yang akan diperbarui, kolom mana yang akan diubah, dan nilai-nilai baru yang akan diterapkan. Selain itu, Anda perlu menentukan kondisi (misalnya, *WHERE clause*) yang akan menentukan baris mana yang akan diperbarui. Setelah perintah *UPDATE* dijalankan, data dalam tabel akan diperbarui sesuai dengan perubahan yang Anda tetapkan. Ini memungkinkan Anda untuk mengubah informasi dalam basis data sesuai dengan perkembangan, perbaikan, atau perubahan yang terjadi dalam aplikasi atau sistem yang Anda kelola. Pastikan untuk berhati-hati saat menggunakan perintah *UPDATE* untuk menghindari perubahan data yang tidak diinginkan atau potensi kerusakan pada integritas data. Setelah menjalankan perintah *UPDATE*, perubahan yang Anda terapkan pada data akan langsung mencerminkan hasilnya dalam tabel dalam database. Perintah *UPDATE* dapat digunakan untuk mengubah satu atau lebih baris data dalam satu operasi, tergantung pada kondisi yang Anda tetapkan. Kondisi yang ditentukan dalam klausa *WHERE* dapat membatasi baris-baris mana yang akan diperbarui, sehingga Anda dapat memilih dengan tepat data mana yang ingin Anda ubah. Ini memungkinkan Anda untuk melakukan tugas-tugas seperti pembaruan informasi pelanggan, pengaturan status pesanan, atau perbaikan kesalahan data dalam tabel. Pastikan untuk membuat perubahan yang sesuai dan memeriksa data yang akan diubah sebelum menjalankan perintah *UPDATE*, dan jangan lupa untuk membuat cadangan data jika diperlukan untuk menghindari

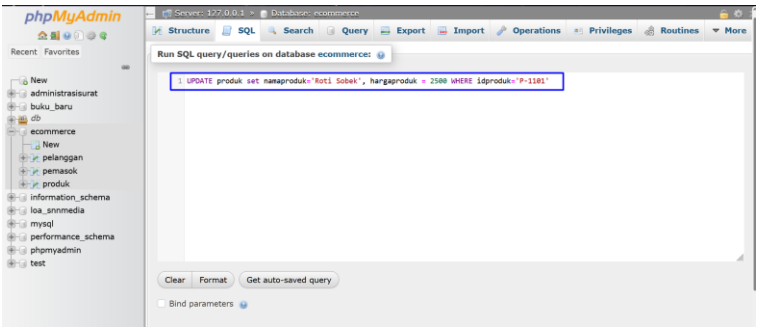
kehilangan informasi yang penting. Dengan perintah *UPDATE*, Anda memiliki alat yang kuat untuk memperbarui data dalam *database* sesuai dengan kebutuhan bisnis atau aplikasi Anda.

Kita akan mencoba mengubah data dalam tabel produk dengan rincian data sebagai berikut.

Tabel 5.4. Mengubah Data Produk

ID Produk	Nama Produk	Harga Produk	Jenis Produk
Data Awal			
P-1101	Roti	1500	Makanan
Data Baru			
P-1101	Roti Sobek	2500	Makanan

Dari data diatas kita akan mencoba mengubah data id produk P-1101, data yang berubah pada bagian nama produk dan harga produk. Perintah SQL sebagai berikut.



Gambar 5.5. Perintah SQL Mengubah Data

Selanjutnya kita *execute* dan hasil *execute* sebagai berikut.

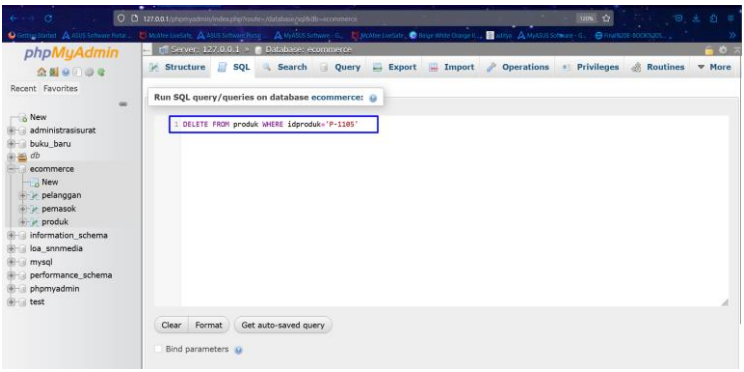
Gambar 5.6. Hasil *Execute* Mengubah Data

5.4. MENGHAPUS DATA

Untuk menghapus data dari *database* menggunakan SQL, Anda dapat menggunakan perintah SQL *DELETE*. Perintah *DELETE* memungkinkan Anda untuk menghapus satu atau lebih baris data dari tabel yang sesuai dengan kondisi yang Anda tentukan. Anda harus menentukan tabel yang akan dihapus dan klausa *WHERE* yang akan menentukan baris-baris mana yang akan dihapus. Setelah perintah *DELETE* dijalankan, data yang cocok dengan kondisi yang Anda tentukan akan dihapus dari tabel. Ini memungkinkan Anda untuk menghapus data yang sudah tidak diperlukan atau tidak relevan dalam basis data, seperti menghapus catatan pelanggan yang sudah tidak aktif atau mengelola data yang usang. Penting untuk berhati-hati saat menggunakan perintah *DELETE* untuk menghindari penghapusan data yang tidak diinginkan atau kehilangan informasi yang penting, oleh karena itu, pastikan untuk memverifikasi kondisi dengan teliti sebelum menjalankan perintah *DELETE*. Setelah menjalankan perintah *DELETE*, data yang sesuai dengan kondisi yang Anda tetapkan akan dihapus dari tabel dalam *database*. Perintah *DELETE* dapat digunakan untuk menghapus satu baris data sekaligus atau

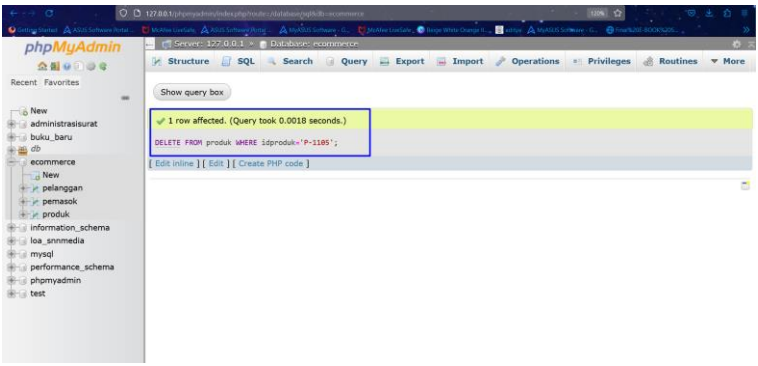
beberapa baris data dalam satu operasi, tergantung pada kondisi yang Anda tentukan. Kondisi yang Anda tentukan dalam klausa *WHERE* memungkinkan Anda untuk menentukan dengan tepat baris-baris mana yang akan dihapus. Perintah *DELETE* adalah alat penting dalam pemeliharaan dan pengelolaan *database*, dan perlu dilakukan dengan hati-hati untuk memastikan bahwa penghapusan data tidak menyebabkan kerusakan atau kehilangan informasi yang penting dalam basis data. Sebagai praktik terbaik, selalu pastikan untuk mencadangkan data yang akan dihapus jika diperlukan.

Kita akan mencoba menghapus data dalam tabel produk dengan id produk **"P-1105"**, perintah SQL sebagai berikut.



Gambar 5.7. Perintah SQL Menghapus Data

Selanjutnya kita *execute* dan hasil *execute* sebagai berikut.



Gambar 5.8. Hasil *Execute* Menghapus Data

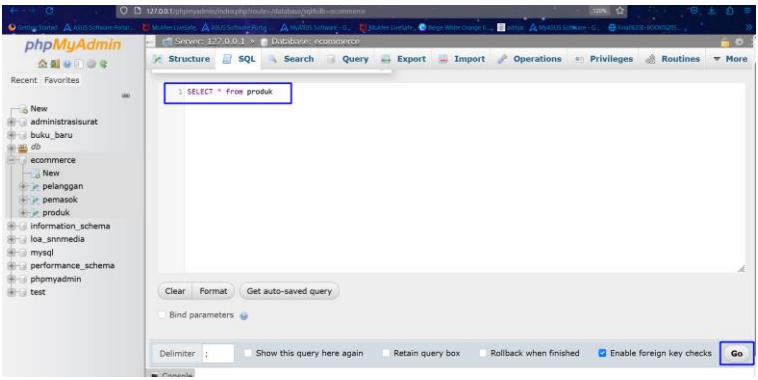
5.5. MENAMPILKAN DATA

Untuk menampilkan data dari *database* menggunakan SQL, Anda dapat menggunakan perintah SQL *SELECT*. Perintah *SELECT* memungkinkan Anda untuk mengambil data dari satu atau lebih tabel dalam basis data sesuai dengan kriteria tertentu. Anda dapat menentukan kolom-kolom yang ingin Anda tampilkan, tabel yang ingin Anda gunakan, dan kondisi yang akan memfilter data. Setelah menjalankan perintah *SELECT*, hasilnya akan berupa tabel hasil yang berisi data yang memenuhi kriteria yang Anda tetapkan. Ini memungkinkan Anda untuk mengakses dan menganalisis data dalam basis data dengan cara yang sesuai dengan kebutuhan aplikasi atau sistem yang Anda kembangkan. Anda juga dapat menggunakan klausa *JOIN* untuk menggabungkan data dari beberapa tabel, klausa *GROUP BY* untuk mengelompokkan data, serta klausa *ORDER BY* untuk mengurutkan hasil pencarian. Perintah *SELECT* adalah dasar dalam bekerja dengan data dalam database dan memberikan cara efektif untuk mengambil informasi yang Anda butuhkan.

Perintah *SELECT* dalam SQL juga dapat digunakan untuk mengambil data dari satu atau lebih tabel dengan fleksibilitas tinggi. Anda dapat menerapkan berbagai fungsi

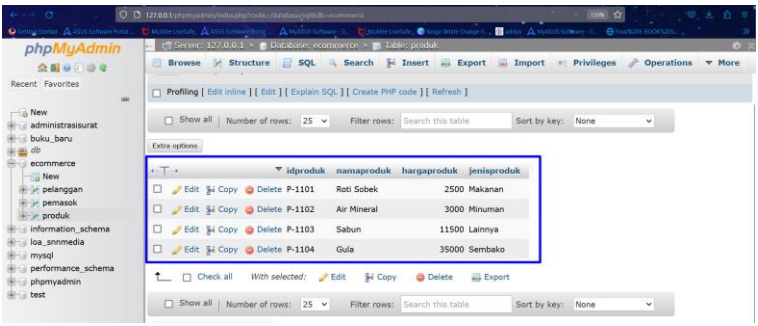
agregasi seperti *SUM*, *COUNT*, *AVG*, dan lainnya untuk menganalisis data dalam kelompok. Selain itu, Anda dapat menggunakan klausa *WHERE* untuk menyaring data berdasarkan kondisi tertentu, sehingga Anda hanya mendapatkan data yang sesuai dengan kriteria yang Anda tetapkan. Perintah *SELECT* juga dapat digunakan untuk mengambil data dari tabel yang tergabung (*subquery*), sehingga Anda dapat melakukan pencarian yang lebih kompleks atau menggabungkan hasil dari beberapa kueri untuk mendapatkan informasi yang lebih mendalam. Dengan SQL, Anda dapat mengambil, menyaring, mengelompokkan, dan menganalisis data dalam berbagai cara sesuai dengan kebutuhan Anda, dan hasilnya ditampilkan dalam bentuk tabel yang dapat diakses dan dimanipulasi sesuai kebutuhan. Perintah *SELECT* adalah alat yang sangat kuat dalam analisis data dan pengambilan informasi dari *database*.

Kita akan mencoba menampilkan seluruh data dalam tabel produk dengan perintah SQL sebagai berikut.



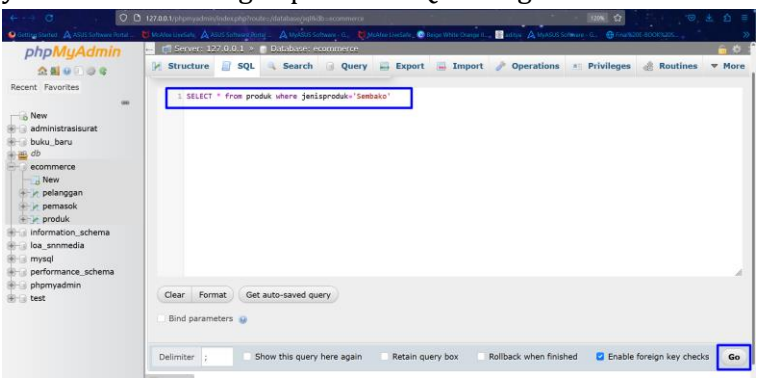
Gambar 5.9. Perintah SQL Menampilkan Seluruh Data

Selanjutnya kita *execute* dan hasil *execute* sebagai berikut.



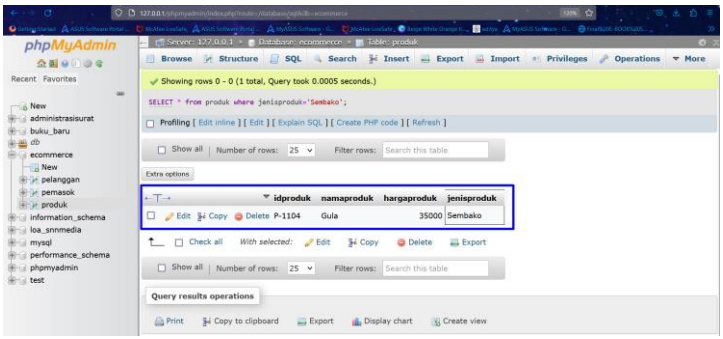
Gambar 5.10. Hasil *Execute* Menampilkan Seluruh Data

Selanjutnya kita akan mencoba menampilkan seluruh data dalam tabel produk dengan ketentuan jenis produk yaitu sembako dengan perintah SQL sebagai berikut.



Gambar 5.11. Perintah SQL Menampilkan Data Tertentu

Selanjutnya kita *execute* dan hasil *execute* sebagai berikut.



Gambar 5.12. Hasil *Execute* Menampilkan Data Tertentu

3.6. SOAL LATIHAN DML

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) Tambahkan data pada tabel pelanggan seperti tabel berikut ini.

Tabel 5.5. Data Pelanggan

ID Pelanggan	Nama Pelanggan	Alamat Pelanggan	Nomor HP Pelanggan
PL006	Yudi	Bogor	08781234212
PL007	Rido	Bandung	08781234214
PL008	Sanusi	Cirebon	08781234216

2) Selanjutnya ubah data pelanggan yaitu alamat awal bogor menjadi samarinda dengan id pelanggan PL006

BAB VI
IMPLEMENTASI DATA *CONTROL LANGUAGE* (DCL)

Implementasi Data *Control Language* (DCL) adalah aspek penting dalam manajemen keamanan dan hak akses dalam sistem basis data. DCL adalah komponen dari bahasa SQL yang digunakan untuk mengendalikan dan mengatur hak akses pengguna ke basis data, serta mengelola izin untuk operasi tertentu seperti pembacaan, penulisan, dan penghapusan data. DCL memungkinkan pengelola basis data untuk menentukan siapa yang memiliki hak untuk melakukan tindakan tertentu dalam basis data, serta untuk memastikan kebijakan keamanan yang ketat dalam melindungi data yang sensitif. Pemahaman yang mendalam tentang implementasi DCL menjadi sangat penting dalam menjaga privasi data, menghindari akses yang tidak sah, dan memenuhi peraturan yang berlaku, seperti GDPR. Dengan DCL, organisasi dapat mengontrol dengan cermat siapa yang memiliki akses ke data, mengelola hak pengguna, dan menjaga integritas serta keamanan data, yang merupakan aset berharga dalam lingkungan yang semakin terhubung dan rentan terhadap ancaman keamanan. Selain itu, implementasi Data *Control Language* (DCL) juga memberikan fleksibilitas dalam mengatur akses ke data. Ini memungkinkan organisasi untuk mengadopsi model hak akses yang sesuai dengan struktur organisasi mereka dan mengakomodasi kebutuhan bisnis yang beragam. DCL juga memfasilitasi audit dan pemantauan aktivitas pengguna dalam basis data, memungkinkan deteksi dini terhadap potensi ancaman keamanan atau penggunaan yang tidak sah. Lebih dari itu, DCL juga berperan dalam mengamankan data dan menjaga kepatuhan dengan regulasi, seperti HIPAA untuk data kesehatan atau PCI DSS untuk data pembayaran.

Dengan pemahaman yang baik tentang implementasi DCL, pengelola basis data dan profesional keamanan dapat memastikan bahwa data tetap aman, privasi terjaga, dan risiko keamanan data dikelola dengan efektif. Dalam dunia di mana keamanan data merupakan tantangan kunci, DCL adalah alat yang sangat penting dalam melindungi dan mengelola data secara tepat dan aman.

6.1. PENGANTAR DCL

Data *Control Language* (DCL) merupakan bagian dari bahasa pemrograman komputer yang berfungsi untuk mengatur hak akses, perizinan, dan izin dalam sistem pengelolaan basis data (DBMS). DCL digunakan untuk mengontrol bagaimana pengguna dan peran dapat mengakses dan memanipulasi data dalam basis data. Tujuan utama dari DCL adalah untuk menjaga keamanan, kerahasiaan, integritas, dan konsistensi data.

DCL merupakan salah satu komponen penting dalam sistem manajemen basis data karena memungkinkan administrator basis data untuk mengontrol tingkat akses dan perizinan pengguna. Dengan menggunakan DCL, administrator dapat memastikan bahwa data sensitif hanya dapat diakses oleh pihak yang berwenang, mencegah manipulasi data yang tidak sah, dan menjaga integritas serta konsistensi basis data. Hal ini sangat penting dalam lingkungan yang mengandung data rahasia atau penting yang perlu dijaga keamanannya.

Data Control Language (DCL) adalah komponen penting dalam bahasa SQL yang digunakan untuk mengendalikan hak akses dan keamanan data dalam sistem basis data. DCL mencakup pernyataan *GRANT*, yang memungkinkan administrator untuk memberikan hak akses tertentu kepada pengguna atau peran, serta pernyataan *REVOKE* yang digunakan untuk mencabut izin yang telah

diberikan sebelumnya. Melalui DCL, administrator basis data memiliki kontrol penuh atas siapa yang memiliki akses ke data, jenis operasi yang diizinkan, dan kebijakan keamanan yang diterapkan dalam basis data. Ini membantu menjaga integritas data dan memastikan bahwa data sensitif tetap dilindungi dari akses yang tidak sah, menjadikan DCL alat yang penting dalam pengelolaan basis data yang aman dan andal. Selain itu, DCL juga memberikan fleksibilitas dalam pengaturan tingkat akses untuk berbagai pengguna atau peran yang terlibat dalam sistem basis data. Ini berarti bahwa administrator dapat menentukan siapa yang memiliki izin untuk membaca, menulis, atau mengubah data tertentu, seiring dengan kemampuan untuk mencabut izin tersebut jika situasinya berubah atau jika akses yang diberikan tidak lagi diperlukan. DCL juga merupakan alat yang penting dalam menjalankan kebijakan keamanan data yang sesuai dengan peraturan dan persyaratan perusahaan. Dengan demikian, DCL adalah komponen kunci dalam memastikan bahwa data dalam basis data tetap terlindungi dan bahwa pengguna memiliki akses yang sesuai dengan peran dan tanggung jawab mereka.

6.2. PERINTAH DCL UTAMA

Perintah DCL tidak hanya mengenai memberikan dan mencabut izin akses, tetapi juga tentang memantau dan melacak aktivitas pengguna dalam basis data. Administrator dapat menggunakan DCL untuk mencatat siapa yang memiliki akses ke data, kapan mereka mengaksesnya, dan apa yang mereka lakukan. Ini adalah langkah penting dalam menjaga keamanan dan integritas data, serta dalam audit kepatuhan dengan peraturan seperti GDPR, HIPAA, atau regulasi keamanan data lainnya. Dengan DCL, organisasi dapat menjalankan pengelolaan keamanan data dengan

lebih baik, menjadikan pengelolaan basis data lebih aman dan andal.

Perintah DCL merupakan perintah-perintah dalam bahasa SQL yang digunakan untuk mengatur hak akses, perizinan, dan izin dalam sistem pengelolaan basis data (DBMS). DCL memungkinkan administrator basis data untuk memberikan atau mencabut hak akses tertentu kepada pengguna atau peran dalam basis data. Berikut adalah dua perintah DCL yang paling umum digunakan:

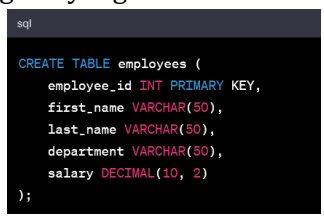
a) *GRANT*

Perintah *GRANT* digunakan untuk memberikan hak akses atau izin tertentu kepada pengguna atau peran dalam basis data. Hak akses yang diberikan dapat meliputi hak untuk membaca data (*SELECT*), menyisipkan data baru (*INSERT*), memperbarui data yang sudah ada (*UPDATE*), menghapus data (*DELETE*), dan lainnya.

b) *REVOKE*

Perintah *REVOKE* digunakan untuk mencabut hak akses atau izin yang telah diberikan sebelumnya kepada pengguna atau peran dalam basis data.

Implementasi Perintah DCL melibatkan penggunaan perintah *GRANT* dan *REVOKE* dalam bahasa SQL untuk memberikan atau mencabut hak akses serta perizinan kepada pengguna atau peran dalam basis data. Sebagai contoh kita buat tabel "employees" yang berisi informasi tentang karyawan, dan kita ingin memberikan izin tertentu kepada dua pengguna yang berbeda.



Gambar 6.1. Perintah Membuat Tabel

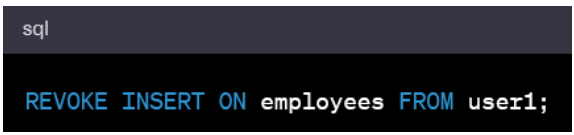
Dalam perintah *query* ini, kita membuat tabel "employees" dengan kolom "employee_id" sebagai kunci utama (*primary key*), dan kolom "first_name" dengan tipe data *varchar* dan *length* 50, "last_name" dengan tipe data *varchar* dan *length* 50, "department" dengan tipe data *varchar* dan *length* 50, dan "salary" dengan tipe data *decimal* dan *length* 10 dan mempunyai 2 nilai decimal setelah koma yang akan ditampilkan.

Selanjutnya kita akan menggunakan perintah *GRANT* seperti berikut ini.

Gambar 6.2. Perintah *Grant*

Dalam perintah *query* ini, kita berikan izin *SELECT* dan *INSERT* kepada pengguna "user1" agar dia dapat melihat dan menyisipkan data ke dalam tabel "employees".

Selanjutnya kita akan menggunakan perintah *REVOKE* seperti berikut ini.



Gambar 6.3. Perintah *Revoke*

Dalam perintah *query* ini, kita mencabut izin *INSERT* dari pengguna "user1" agar tidak dapat menyisipkan data lagi, tetapi dia masih dapat melihat data yang ada (hak akses *SELECT* tetap ada).

6.3. PENGGUNA DAN PERAN DALAM DCL

Dalam *Data Control Language* (DCL), pengguna dan peran memiliki peran penting dalam mengelola hak akses dan keamanan data dalam basis data. Pengguna merujuk

kepada individu yang memiliki akses ke basis data, sementara peran mengacu pada kelompok pengguna yang memiliki hak akses yang serupa. Dengan DCL, administrator dapat menentukan peran yang memiliki izin tertentu, dan kemudian mengaitkan pengguna dengan peran tersebut. Ini memungkinkan pengelolaan yang lebih efisien dan efektif, terutama dalam lingkungan dengan banyak pengguna. Pengguna dan peran dapat memiliki izin untuk membaca, menulis, mengubah, atau menghapus data tertentu dalam basis data sesuai dengan peran dan tanggung jawab mereka dalam organisasi. Dengan cara ini, DCL memungkinkan pengelolaan yang lebih tepat dan terstruktur atas hak akses, memastikan bahwa pengguna hanya memiliki akses yang sesuai dengan kebutuhan mereka dan meminimalkan risiko akses yang tidak sah ke data sensitif. Selain itu, pengguna dan peran dalam DCL juga memungkinkan untuk mengelola hak akses dengan lebih mudah dalam lingkungan yang berubah-ubah. Ketika peran atau hak akses individu berubah seiring waktu atau ketika ada kebutuhan untuk menyesuaikan tingkat akses, administrator dapat melakukan perubahan pada peran tersebut tanpa perlu mengubah izin pengguna satu per satu. Ini mempermudah tugas administrasi dan meminimalkan kesalahan manusiawi dalam mengelola hak akses. Pengguna dan peran juga membantu dalam pemahaman yang lebih baik tentang struktur basis data dan mengelola kompleksitas dalam hak akses dengan lebih terstruktur.

Pengelolaan pengguna dan peran dalam DCL adalah bagian penting dalam merancang basis data yang aman dan andal, serta memastikan bahwa hak akses data diberikan dan dikelola secara efektif sesuai dengan peraturan dan kebijakan yang berlaku dalam organisasi. Peran dalam DCL merujuk pada sekumpulan hak akses dan izin yang diberikan kepada pengguna atau kelompok pengguna tertentu dalam

basis data. Peran dalam DCL adalah alat yang penting dalam mengatur dan mengelola hak akses dan keamanan dalam basis data. Berikut adalah beberapa poin penting tentang peran dalam DCL:

A. Pengelolaan Hak Akses: Peran memungkinkan administrator basis data untuk mengelompokkan pengguna dengan hak akses yang serupa ke dalam satu entitas. Ini memudahkan pengelolaan dan pemeliharaan hak akses, terutama dalam lingkungan dengan banyak pengguna.
B. Simpel dan Terstruktur: Dengan menggunakan peran, administrator dapat mengatur hak akses dengan lebih terstruktur. Sebagai contoh, administrator dapat membuat peran khusus untuk pengguna yang memerlukan akses baca saja dan peran lain untuk pengguna yang memerlukan hak akses penuh. Hal ini meminimalkan risiko kesalahan dan memastikan bahwa pengguna hanya memiliki akses yang sesuai dengan peran mereka.
C. Perubahan Fleksibel: Peran memungkinkan perubahan hak akses dengan lebih fleksibel seiring waktu. Administrator dapat menyesuaikan peran dengan lebih mudah ketika peran atau hak akses individu berubah, tanpa perlu mengubah izin pengguna satu per satu.
D. Auditing dan Pemantauan: Penggunaan peran juga mendukung proses auditing dan pemantauan aktivitas pengguna. Dengan peran, lebih mudah untuk melacak dan memahami siapa yang melakukan apa dalam basis data.
E. Kepatuhan Regulasi: Peran juga dapat membantu organisasi dalam menjalankan kepatuhan dengan regulasi data dan peraturan tertentu seperti GDPR, HIPAA, atau regulasi keamanan data lainnya. Ini karena

peran memungkinkan pengaturan kebijakan keamanan yang sesuai dengan peraturan tersebut.

Penggunaan peran dalam DCL adalah langkah penting dalam merancang basis data yang aman, efisien, dan mudah dikelola. Ini membantu administrator dalam mengatur dan memantau hak akses dengan lebih baik dan memastikan bahwa data sensitif tetap terlindungi dan hanya dapat diakses oleh pihak yang diizinkan.

6.4. SOAL LATIHAN DCL

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) *GRANT Permission*: Dalam suatu basis data, Anda memiliki tabel "Pegawai" dan ingin memberikan hak akses *SELECT* ke pengguna "JohnDoe". Tuliskan pernyataan SQL yang benar untuk memberikan izin tersebut.
2) *REVOKE Permission*: Anda telah memberikan hak akses *INSERT* kepada peran "Manajer" dalam basis data Anda, tetapi sekarang Anda ingin mencabut izin tersebut. Tuliskan pernyataan SQL yang benar untuk mencabut izin *INSERT* dari peran "Manajer".
3) Pengguna dengan Beberapa Peran: Pengguna "Alice" adalah bagian dari dua peran, "Penulis" dan "Editor". Anda ingin memberikan hak akses *DELETE* hanya kepada peran "Editor" tanpa mempengaruhi hak akses pengguna "Alice" sebagai "Penulis". Bagaimana Anda akan melakukannya dengan pernyataan SQL?
4) Audit Log: Anda ingin membuat log aktivitas pengguna yang mencakup semua perubahan yang dilakukan dalam basis data. Apa jenis pernyataan SQL yang harus Anda gunakan untuk mencapai ini?
5) Kebijakan Kepatuhan: Dalam lingkungan yang harus mematuhi regulasi privasi data, seperti GDPR, Anda

harus memastikan bahwa data pribadi hanya dapat diakses oleh pengguna yang berwenang. Bagaimana Anda akan mengatur peran dan izin untuk mematuhi regulasi tersebut?

BAB VII
IMPLEMENTASI *VIEW*, *FUNCTION*, DAN *TRIGGER*

Implementasi *View*, *Function*, dan *Trigger* adalah komponen integral dalam pengelolaan dan manipulasi data dalam sistem basis data. *View* merupakan cara untuk membuat tampilan data yang abstrak dari tabel-tabel yang ada, yang memungkinkan pengguna untuk melihat dan mengakses data dengan cara yang terstruktur tanpa mengubah struktur basis data asli. *Function* adalah program kecil yang dapat menerima *input*, melakukan operasi tertentu, dan mengembalikan hasil yang dapat digunakan dalam kueri SQL. Sementara itu, *Trigger* adalah aturan yang digunakan untuk merespons peristiwa tertentu, seperti penambahan, pembaruan, atau penghapusan data, dan dapat digunakan untuk menjalankan tindakan otomatis seperti pembaruan data, audit, atau notifikasi. Implementasi *View*, *Function*, dan *Trigger* memiliki peran penting dalam mengoptimalkan akses, analisis, dan pemeliharaan data dalam sistem basis data, serta meningkatkan efisiensi dan fleksibilitas dalam mengelola informasi dalam konteks aplikasi dan bisnis yang beragam. Selain mendukung tampilan data yang lebih mudah dimengerti dan operasi data yang lebih efisien, implementasi *View*, *Function*, dan *Trigger* juga memainkan peran dalam menjaga konsistensi dan integritas data. Dengan *View*, pengguna dapat melihat data dengan cara yang telah diproses sebelumnya, sehingga meminimalkan peluang kesalahan. *Function* dapat digunakan untuk menerapkan aturan bisnis yang konsisten, seperti perhitungan pajak atau validasi data. *Trigger*, di sisi lain, memungkinkan tindakan otomatis untuk memastikan kepatuhan dengan aturan bisnis dan keamanan data. Oleh karena itu, pemahaman yang mendalam tentang

implementasi *View*, *Function*, dan *Trigger* adalah kunci dalam pengelolaan data yang andal, akurat, dan terstruktur dalam sistem basis data, serta dalam menghadapi tantangan yang berkaitan dengan pengembangan aplikasi dan analisis data di berbagai bidang. Dalam dunia yang semakin bergantung pada data untuk pengambilan keputusan, implementasi ini menjadi elemen penting dalam mengoptimalkan penggunaan data secara keseluruhan.

7.1. PENGANTAR *VIEW*, *TRIGGER*, DAN *FUNCTION*

Dalam SQL *View* adalah sebuah objek yang digunakan untuk menghasilkan tampilan virtual dari satu atau lebih tabel dalam basis data. *View* adalah sebuah cara untuk menyederhanakan kompleksitas permintaan data, memungkinkan pengguna untuk mengakses dan menganalisis data dari beberapa tabel seolah-olah mereka adalah satu tabel tunggal. *View* berperan penting dalam meningkatkan keamanan dan efisiensi, karena pengguna dapat diberikan akses terbatas hanya ke data yang relevan melalui *view*, sambil menjaga integritas data dalam tabel aslinya. *View* juga dapat digunakan untuk mengabstraksi logika bisnis kompleks, memungkinkan pengembang untuk mengisolasi detail implementasi dari pengguna akhir. Selain itu, *view* dalam SQL juga memungkinkan perubahan struktur data tanpa mempengaruhi aplikasi yang bergantung padanya. Dengan kata lain, jika Anda perlu menambahkan, menghapus, atau mengubah kolom dalam tabel, Anda dapat memperbarui *view* yang ada tanpa harus mengubah semua permintaan SQL di aplikasi. *View* juga dapat digunakan untuk menggabungkan data dari berbagai tabel dengan cara yang kohesif, sehingga memudahkan analisis dan pelaporan. Meskipun view tidak mengandung data aktual, mereka menyediakan visibilitas yang konsisten ke data tersebut, menjadikannya alat yang kuat dalam SQL untuk mengatasi

kebutuhan pengambilan data yang kompleks dan pengembangan aplikasi yang lebih fleksibel.

Manfaat utama dari penggunaan view dalam SQL adalah kemampuannya untuk menyediakan abstraksi data yang berguna, meningkatkan keamanan, dan meningkatkan efisiensi pengaksesan data. Dengan *view*, pengguna dapat melihat data dari berbagai tabel seolah-olah mereka adalah satu entitas tunggal, menyembunyikan kompleksitas struktur basis data yang sebenarnya. Hal ini juga memungkinkan administrator untuk memberikan akses terkontrol ke data, menjaga privasi dan integritas data dengan lebih baik. Selain itu, *view* dapat digunakan untuk mengurangi jumlah permintaan SQL yang perlu ditulis, mempromosikan pemeliharaan yang lebih mudah, dan mengizinkan pengguna untuk mengakses hanya data yang relevan, yang dapat meningkatkan kinerja dan efisiensi sistem. Selain itu, view juga memungkinkan pengguna untuk membuat lapisan logika bisnis yang kompleks, mengisolasi perubahan dalam struktur tabel, dan mempermudah pengembangan aplikasi yang lebih fleksibel. Dalam konteks analisis data dan pelaporan, *view* memainkan peran penting dalam menggabungkan data dari berbagai tabel dengan cara yang mudah dimengerti, sehingga memudahkan proses pengambilan keputusan. Mereka juga memungkinkan tim analisis untuk bekerja dengan tampilan data yang sudah difilter atau diolah sesuai dengan kebutuhan mereka, tanpa perlu mengganggu data asli atau struktur basis data yang mendasarinya. Selain itu, *view* juga mempermudah pengelolaan perubahan struktur data, karena perubahan hanya perlu diterapkan pada *view* yang relevan tanpa mengganggu aplikasi yang menggunakan view tersebut. Dengan manfaat-manfaat ini, pengguna SQL dapat lebih efektif dalam mengelola dan menganalisis data, mempercepat proses pengembangan, dan meningkatkan

fleksibilitas dalam memenuhi kebutuhan bisnis yang berkembang.

Manfaat utama dari penggunaan *view* dalam SQL adalah sebagai berikut:

A. Abstraksi Data: *View* memungkinkan pengguna untuk melihat data dari beberapa tabel seolah-olah mereka adalah satu tabel tunggal. Hal ini membantu dalam menyembunyikan kompleksitas struktur basis data yang sebenarnya, sehingga pengguna tidak perlu khawatir tentang rincian internal tabel.
B. Keamanan Data: Dengan *view*, administrator basis data dapat memberikan akses terkontrol ke data. Ini memungkinkan untuk menjaga privasi dan integritas data dengan lebih baik, karena pengguna hanya dapat mengakses data yang relevan bagi mereka melalui *view*, sambil menjaga data asli yang terlindungi.
C. Efisiensi Permintaan Data: *View* dapat membantu mengurangi jumlah permintaan SQL yang perlu ditulis. Ini mempromosikan pemeliharaan yang lebih mudah dan meningkatkan efisiensi sistem, terutama dalam lingkungan yang memerlukan pengambilan data yang kompleks.
D. Isolasi Logika Bisnis: *View* memungkinkan pembuatan lapisan logika bisnis yang kompleks, yang memungkinkan pengembang untuk mengisolasi perubahan dalam struktur tabel. Ini mempermudah pengembangan aplikasi yang lebih fleksibel.
E. Analisis Data: *View* sangat bermanfaat dalam analisis data dan pelaporan. Mereka memungkinkan penggabungan data dari berbagai tabel dengan cara yang mudah dimengerti, mempermudah proses pengambilan keputusan, dan memungkinkan tim analisis untuk bekerja dengan data yang sudah difilter atau diolah sesuai kebutuhan mereka.

F. Manajemen Perubahan Struktur Data: Perubahan dalam struktur data hanya perlu diterapkan pada *view* yang relevan tanpa mengganggu aplikasi yang menggunakan *view* tersebut, sehingga memudahkan pengelolaan perubahan.

Dengan manfaat-manfaat ini, *view* menjadi alat yang kuat dalam SQL untuk mengelola, menganalisis, dan mengakses data secara efektif, meningkatkan fleksibilitas, dan memenuhi kebutuhan bisnis yang berkembang.

Dalam SQL *Function* adalah objek yang memungkinkan pengguna untuk membuat fungsi kustom yang dapat menerima satu atau beberapa argumen, melakukan operasi tertentu, dan mengembalikan hasil. Fungsi-fungsi ini dapat digunakan dalam pernyataan SQL untuk memproses data, melakukan perhitungan matematis, mengambil nilai dari tabel, atau menjalankan logika bisnis yang spesifik. Penggunaan fungsi dapat memungkinkan kode SQL menjadi lebih modular dan dapat digunakan kembali, mengurangi duplikasi kode, serta meningkatkan keterbacaan dan pemeliharaan pernyataan SQL. Selain itu, SQL memiliki sejumlah fungsi bawaan yang dapat digunakan untuk tugas-tugas umum, seperti fungsi matematika, manipulasi string, atau agregasi data. Fungsi-fungsi ini memainkan peran penting dalam menganalisis data, pelaporan, dan pengelolaan data dalam SQL. Selain manfaat yang telah disebutkan, fungsi dalam SQL juga memungkinkan pengguna untuk mengenkapsulasi logika bisnis yang kompleks, menjaga integritas data, dan menghindari perubahan langsung pada data asli. Dengan menggabungkan fungsi dengan pernyataan SQL, pengguna dapat melakukan perhitungan dan manipulasi data dengan cara yang lebih efisien. Selain itu, fungsi juga dapat digunakan dalam pernyataan *SELECT* untuk mengambil hasil yang relevan atau dihitung dari basis data, memudahkan pengambilan

data yang disesuaikan dengan kebutuhan analisis. Namun, perlu diingat bahwa efisiensi dan kinerja harus diperhatikan saat merancang dan mengimplementasikan fungsi, karena penggunaan fungsi yang tidak tepat atau kompleksitas yang berlebihan dapat memengaruhi kinerja basis data. Dengan perencanaan dan pemahaman yang tepat, fungsi SQL dapat menjadi alat yang kuat dalam pengelolaan dan analisis data.

Selain itu, fungsi dalam SQL juga mendukung penggunaan data yang lebih bersih dan lebih konsisten. Dengan definisi fungsi yang tepat, Anda dapat memastikan bahwa operasi tertentu dilakukan dengan benar setiap kali fungsi tersebut dipanggil, tanpa risiko kesalahan manusia atau inkonsistensi dalam perhitungan atau manipulasi data. Fungsi juga dapat digunakan dalam berbagai pernyataan SQL, seperti WHERE, JOIN, atau GROUP BY, yang memberikan fleksibilitas dalam pemrosesan dan analisis data. Selain itu, fungsi dapat diakses dari berbagai bahasa pemrograman, sehingga memungkinkan integrasi yang lebih mudah dengan aplikasi yang menggunakan basis data SQL. Dengan memahami penggunaan dan perancangan yang benar, fungsi SQL dapat menjadi alat yang sangat berguna dalam pengelolaan dan analisis data dalam basis data.

Manfaat utama dari penggunaan fungsi dalam SQL adalah:

A. Modularitas: Fungsi memungkinkan pemisahan logika bisnis atau perhitungan tertentu dari pernyataan SQL utama, menciptakan kode yang lebih mudah dikelola, dibaca, dan dipahami. Hal ini juga mempromosikan praktik pengembangan yang lebih modular.
B. Penggunaan Kembali (*Reusability*): Anda dapat menggunakan fungsi yang sama berulang-ulang dalam pernyataan SQL yang berbeda atau dalam beberapa bagian dari aplikasi, mengurangi duplikasi kode dan meningkatkan efisiensi pengembangan.

C. Perhitungan Data: Fungsi memungkinkan perhitungan matematis, manipulasi string, atau transformasi data lainnya, memudahkan dalam analisis dan manipulasi data.
D. Integritas Data: Fungsi dapat digunakan untuk menerapkan aturan bisnis dan validasi data, memastikan bahwa data yang dimasukkan atau diperbarui sesuai dengan aturan yang telah ditentukan.
E. Penyaringan Data: Fungsi dapat digunakan dalam klausa *WHERE* untuk menyaring data berdasarkan kriteria yang lebih kompleks, memungkinkan pengambilan data yang lebih spesifik.
F. Aggregasi Data: Anda dapat menggunakan fungsi agregasi seperti *SUM*, *COUNT*, *AVG*, *MAX*, dan *MIN* untuk merangkum data dalam pernyataan *GROUP BY*, memungkinkan analisis data yang lebih mendalam.
G. Fleksibilitas: Fungsi dapat digunakan dalam berbagai jenis pernyataan SQL, termasuk *SELECT*, *INSERT*, *UPDATE*, dan *DELETE*, memberikan fleksibilitas dalam pemrosesan data.
H. Performa *Query*: Dengan menggunakan fungsi secara bijaksana dan mengoptimalkan kode fungsi, Anda dapat meningkatkan performa *query* dan mengurangi beban *server* basis data.
I. Kemudahan Integrasi: Fungsi SQL dapat diakses dari berbagai bahasa pemrograman, memfasilitasi integrasi dengan aplikasi lain yang menggunakan basis data SQL.

Penggunaan fungsi dalam SQL membantu menjaga kode SQL yang lebih bersih, lebih efisien, dan lebih terstruktur, sehingga memudahkan dalam pengembangan, pemeliharaan, dan analisis data dalam lingkungan basis data.

Dalam SQL *trigger* adalah objek yang digunakan untuk memantau perubahan data dalam tabel dan mengaktifkan tindakan otomatis ketika peristiwa tertentu terjadi. Trigger

memungkinkan pengguna untuk mendefinisikan aturan bisnis, validasi, atau logika khusus yang akan dieksekusi secara otomatis ketika operasi seperti penyisipan (*INSERT*), pembaruan (*UPDATE*), atau penghapusan (*DELETE*) data terjadi. Dengan trigger, perubahan data dapat dipantau dan diproses dengan konsisten, memastikan kepatuhan terhadap aturan bisnis dan menjaga integritas data. Mereka juga berguna dalam pencatatan perubahan, pemantauan aktivitas, atau menggabungkan data dengan cara tertentu. Namun, penggunaan *trigger* perlu dilakukan dengan hati-hati, karena jika tidak dielaborasi dengan benar, mereka dapat memengaruhi performa basis data dan memperumit pemeliharaan. *Trigger* dalam SQL juga dapat digunakan untuk menjalankan tugas otomatis yang dapat membantu dalam pengelolaan basis data, seperti pembaruan log waktu, pencatatan perubahan, atau tindakan pembersihan data. Mereka memungkinkan aplikasi bisnis untuk berfungsi lebih efisien dan konsisten, mengurangi kebutuhan untuk tindakan manual yang dapat rentan terhadap kesalahan. Trigger juga membantu dalam penanganan perubahan data berbasis peristiwa, yang sangat penting dalam banyak aplikasi yang bergantung pada pembaruan data secara *real-time*. Namun, penggunaan *trigger* harus bijaksana, karena terlalu banyak *trigger* atau logika yang kompleks dalam trigger dapat menyulitkan pemeliharaan dan pemahaman atas perilaku sistem. Oleh karena itu, desain dan implementasi trigger perlu dikerjakan dengan teliti untuk mencapai manfaat terbaik sambil meminimalkan dampak negatifnya.

Manfaat utama dari *trigger* dalam SQL adalah:

A. Penjagaan Integritas Data: *Trigger* memungkinkan pengguna untuk menerapkan aturan bisnis dan validasi ketat dalam basis data. Dengan ini, Anda dapat memastikan bahwa data yang dimasukkan atau

diperbarui memenuhi kriteria tertentu sebelum disimpan, sehingga menjaga integritas data.

B. Pemantauan Perubahan Data: *Trigger* dapat digunakan untuk memantau perubahan data dalam waktu nyata. Ini bermanfaat dalam aplikasi yang memerlukan pemantauan perubahan atau pencatatan aktivitas, seperti audit log.
C. Automatisasi Tugas: *Trigger* memungkinkan Anda untuk mengotomatisasi tugas-tugas rutin atau berulang. Misalnya, Anda dapat menggunakan trigger untuk memperbarui tanggal modifikasi atau mengirim pemberitahuan ketika suatu peristiwa penting terjadi.
D. Replikasi Data: *Trigger* dapat digunakan dalam sistem replikasi data, memungkinkan data yang dimasukkan atau diperbarui dalam satu basis data untuk secara otomatis diteruskan ke basis data lainnya.
E. Implementasi Bisnis Khusus: *Trigger* memungkinkan Anda untuk menerapkan logika bisnis yang sangat spesifik dan rumit, bahkan ketika perubahan data melibatkan beberapa tabel atau entitas yang terkait.
F. Keamanan Data: *Trigger* dapat digunakan untuk mengimplementasikan aturan keamanan, seperti pembatasan akses ke data berdasarkan peran pengguna atau atribut lainnya.

Penggunaan *trigger* harus diatur dan dikelola dengan hati-hati. Terlalu banyak *trigger* atau logika yang rumit dalam *trigger* dapat mempengaruhi kinerja basis data, dan pemahaman atas perilaku sistem menjadi lebih kompleks. Oleh karena itu, desain trigger harus dipikirkan dengan matang, dan pemantauan kinerja harus dilakukan untuk memastikan bahwa mereka berfungsi secara efisien.

7.2. IMPLEMENTASI *VIEW*

Implementasi *view* dalam SQL melibatkan beberapa tahapan kunci. Pertama, pengguna harus merancang view dengan hati-hati, menentukan tabel yang akan diakses, kolom yang akan ditampilkan, serta filter atau transformasi yang diperlukan. Selanjutnya, view harus dibuat dalam basis data dengan pernyataan *CREATE VIEW*, dan izin akses harus dikelola untuk memastikan hanya pengguna yang berwenang yang dapat mengakses view tersebut. View dapat digunakan dalam pernyataan SQL seperti *SELECT*, *JOIN*, atau *subquery* untuk mengakses data yang didefinisikan oleh *view*. Penting untuk memahami bahwa view hanyalah representasi *virtual* data dan tidak menyimpan data aktual. Oleh karena itu, setiap kali *view* diakses, *query* yang sesuai akan dijalankan untuk mengambil data dari tabel yang mendasarinya. Implementasi *view* yang baik dapat meningkatkan keamanan, efisiensi, dan pemeliharaan sistem basis data, serta memungkinkan pengguna untuk mengakses dan menganalisis data dengan cara yang lebih terstruktur dan nyaman. Selain itu, dalam proses implementasi *view*, penting juga untuk mempertimbangkan kinerja. Terlalu banyak *view* atau *view* yang kompleks dapat memengaruhi performa basis data. Oleh karena itu, desain *view* sebaiknya memperhatikan perhitungan dan pemrosesan yang efisien. Pengoptimalan query yang digunakan dalam *view* juga dapat membantu meningkatkan kinerja. Selain itu, *view* juga memerlukan pemantauan dan pemeliharaan yang teratur. Dalam lingkungan yang berkembang, struktur data dalam tabel mungkin berubah, dan *view* perlu diperbarui sesuai kebutuhan. Dengan perencanaan yang matang dan pemahaman yang baik tentang peran *view* dalam sistem basis data, implementasi *view* dapat menjadi alat yang kuat

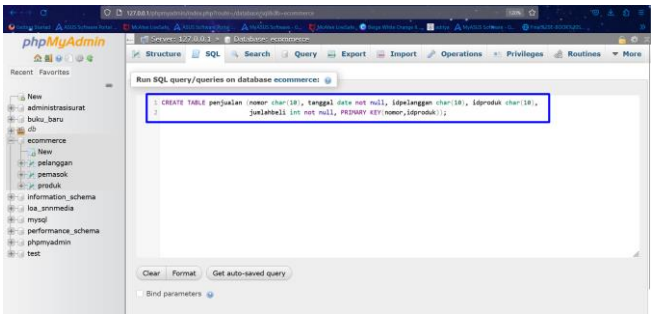
dalam mengelola dan mengakses data dalam lingkungan SQL.

Sebelum membuat *view* kita akan membuat 1 tabel terlebih dahulu dengan nama tabel penjualan, dan deskripsi tabel sebagai berikut.

Tabel 7.1. Tabel Penjualan

Nama Field	Tipe Data	Length	Keterangan
Nomor	*Char*	10	*Primary Key*
Tanggal	*Date*		*Not Null*
idpelanggan	*Char*	10	*Not Null*
idproduk	*Char*	10	*Primary Key*
jumlahbeli	*Int*		*Not Null*

Berdasarkan tabel diatas perintah SQL untuk membuat tabel yaitu.

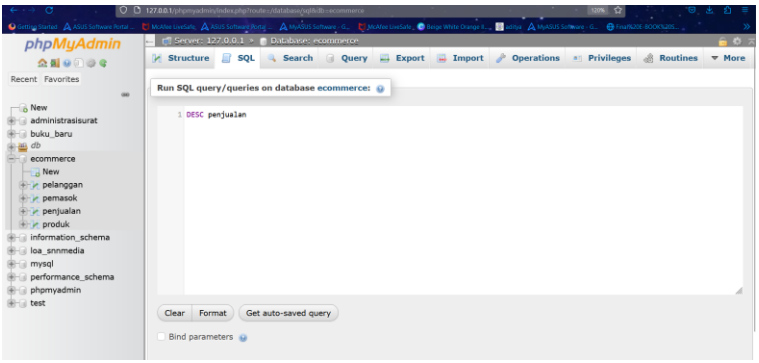


Gambar 7.1. Perintah SQL Membuat Tabel Penjualan

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* sebagai berikut.

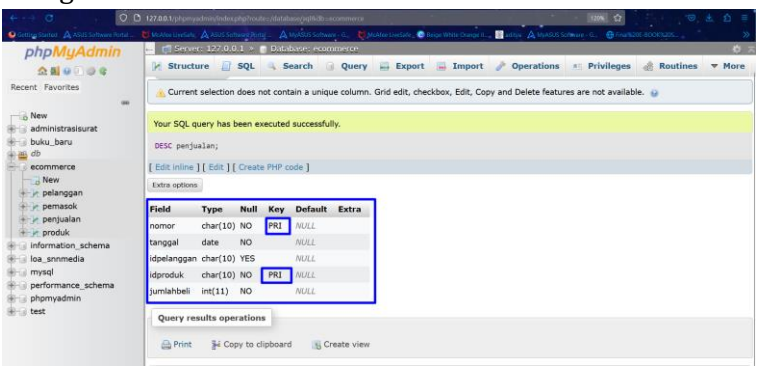
Gambar 7.2. Hasil *Execute* Membuat Tabel Penjualan

Selanjutnya kita akan melihat deskripsi dari tabel penjualan yang kita buat, perintah SQL nya sebagai berikut.



Gambar 7.3. Perintah SQL Melihat Deskripsi Tabel

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* sebagai berikut.



Gambar 7.4. Hasil *Execute* Melihat Deskripsi Tabel

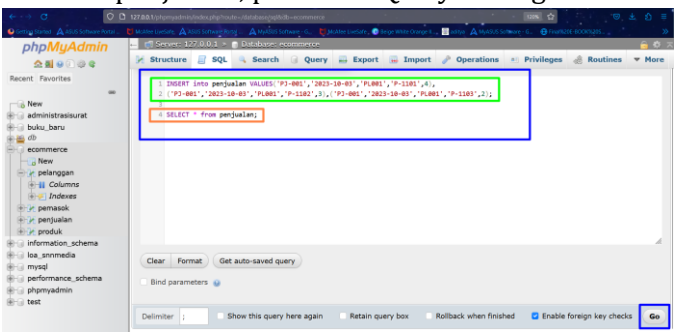
Dari hasil diatas dapat kita lihat untuk *primary key* dalam tabel penjualan yaitu nomor dan idproduk, nantinya ketika ada transaksi penjualan pelanggan tidak akan bisa membeli barang yang sama untuk 1 nomor transaksi.

Selanjutnya kita akan membuat memasukan data penjualan terlebih dahulu, data penjualan yang akan dimasukkan sebagai berikut.

Tabel 7.2. Data Penjualan

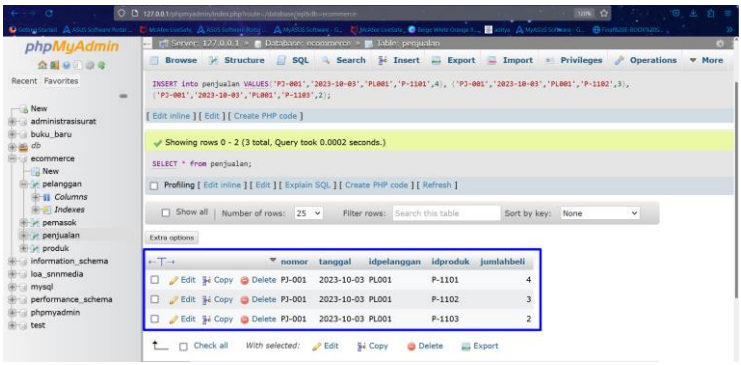
Nomor	Tanggal	ID Pelanggan	ID Produk	Jumlah Beli
PJ-001	2023-10-03	PL001	P-1101	4
PJ-001	2023-10-03	PL001	P-1102	3
PJ-001	2023-10-03	PL001	P-1103	2

Dari data diatas kita akan memasukan data tersebut dalam tabel penjualan, perintah SQL nya sebagai berikut.



Gambar 7.5. Perintah SQL Memasukan Data Penjualan

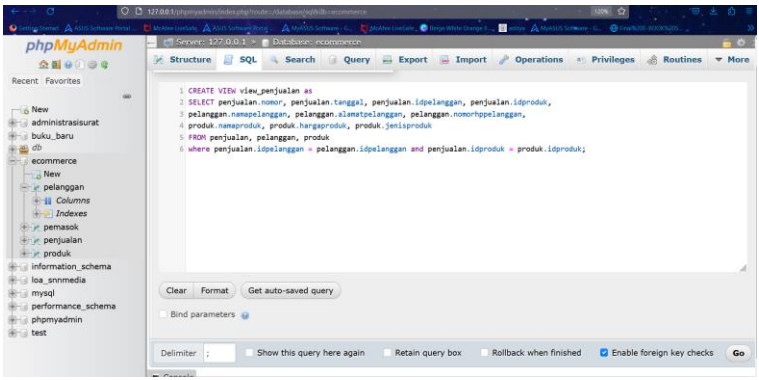
Perintah diatas terdapat 2 perintah DML dalam satu script, perintah pertama untuk menyimpan data penjualan, dan perintah ke dua untuk menampilkan data. Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



Gambar 7.6. Hasil *Execute* Memasukan Data Penjualan

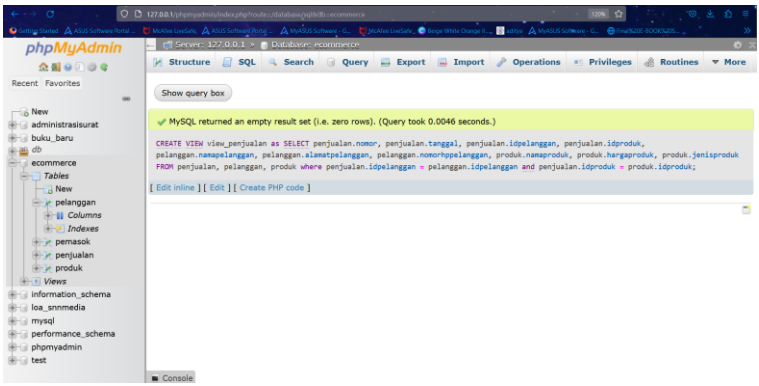
Dalam tabel penjualan yang telah kita buat tabel tersebut mempunyai relasi dengan tabel pelanggan dan tabel produk, dimana idproduk dari tabel produk menjadi penghubung sebagai *foreign key* dalam tabel penjualan, dan idpelanggan dari tabel pelanggan menjadi penghubung sebagai *foreign key* dalam tabel penjualan.

Selanjutnya kita akan membuat *view* yang menghubungkan 3 tabel yaitu penjualan, produk, dan pelanggan. Penghubung antara 3 tabel tersebut yaitu *foreign key* pada tabel penjualan. Perintah SQL nya sebagai berikut.



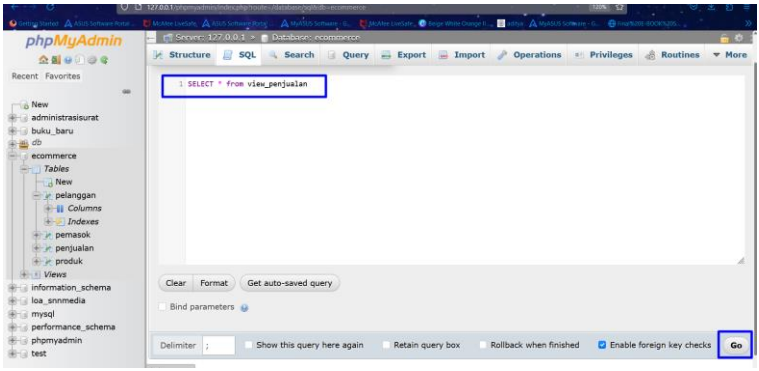
Gambar 7.7. Perintah SQL Membuat *View*

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



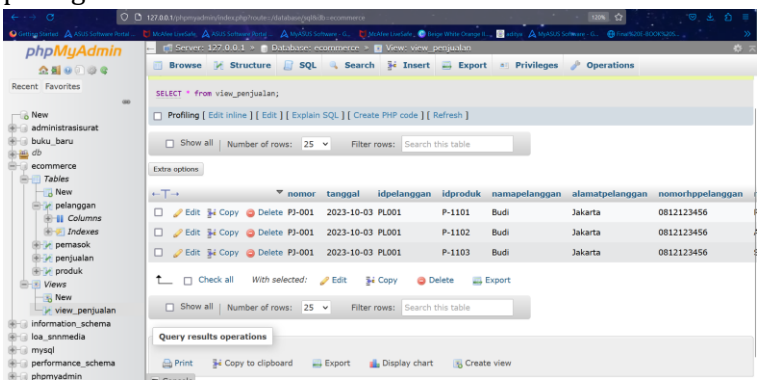
Gambar 7.8. Hasil *Execute* Membuat *View*

Proses selanjutnya kita akan melihat view yang telah kita buat dengan menggunakan perintah *SELECT*, perintah SQL sebagai berikut.



Gambar 7.9. Perintah SQL Menampilkan *View*

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



Gambar 7.10. Hasil *Execute* Menampilkan *View*

View yang kita buat berhasil menampilkan seluruh data dari 3 tabel yang berelasi yaitu penjualan, barang, dan produk.

7.3. IMPLEMENTASI *TRIGGER*

Implementasi *trigger* dalam SQL melibatkan beberapa langkah penting. Pertama, pengguna perlu merancang trigger dengan cermat, menentukan peristiwa yang akan

memicu *trigger*, seperti *INSERT*, *UPDATE*, atau *DELETE*, serta logika bisnis atau tindakan yang akan dijalankan saat peristiwa tersebut terjadi. Selanjutnya, *trigger* harus dibuat dalam basis data dengan pernyataan *CREATE TRIGGER*, dan izin akses harus diatur sesuai kebutuhan. Setelah trigger aktif, mereka akan secara otomatis dieksekusi ketika peristiwa yang sesuai terjadi. Penting untuk memahami dampak kinerja *trigger* pada basis data, dan pengguna harus berhati-hati dalam merancang *trigger* yang efisien. Pemantauan dan pemeliharaan *trigger* juga merupakan bagian penting dari implementasi, karena mereka dapat mempengaruhi integritas data dan kinerja sistem secara keseluruhan. Dengan implementasi *trigger* yang benar, pengguna dapat menjaga konsistensi data, menjalankan tindakan otomatis, serta memantau dan merespons perubahan data dalam basis data SQL. Dalam implementasi *trigger*, perlu diingat bahwa *trigger* dapat memiliki efek jaringan, terutama jika ada trigger yang memicu trigger lainnya (*nested trigger*). Oleh karena itu, perlu diperhatikan desain dan peringatan dalam mengatur *trigger* untuk menghindari perubahan berulang yang tidak diinginkan pada data. Juga, pengoptimalan kode *trigger* dan pemantauan kinerja sangat penting, terutama dalam basis data yang padat. Selain itu, perlu diingat bahwa penggunaan *trigger* harus sesuai dengan aturan bisnis dan persyaratan keamanan yang berlaku. Implementasi *trigger* yang tepat dapat meningkatkan otomatisasi, menjaga integritas data, dan membantu dalam pemantauan serta perubahan data dalam lingkungan SQL, dengan asumsi implementasi dan pemeliharaan yang baik.

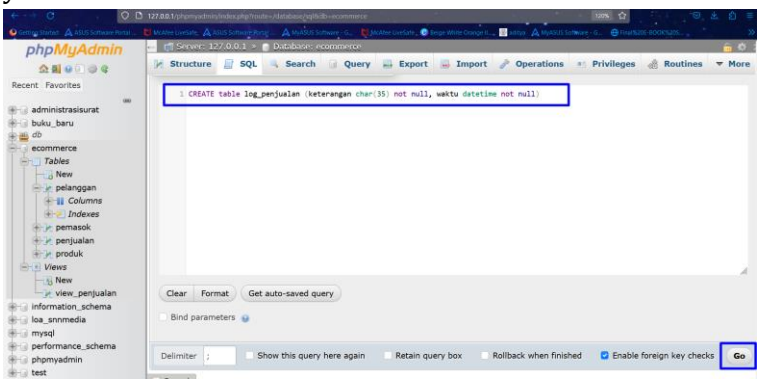
Kita akan membuat *trigger* log transaksi penjualan, yang akan menyimpan data history kita data penjualan disimpan, kita akan membuat 1 tabel terlebih dahulu dengan

nama tabel log penjualan, dan deskripsi tabel sebagai berikut.

Tabel 7.3. Tabel Log Penjualan

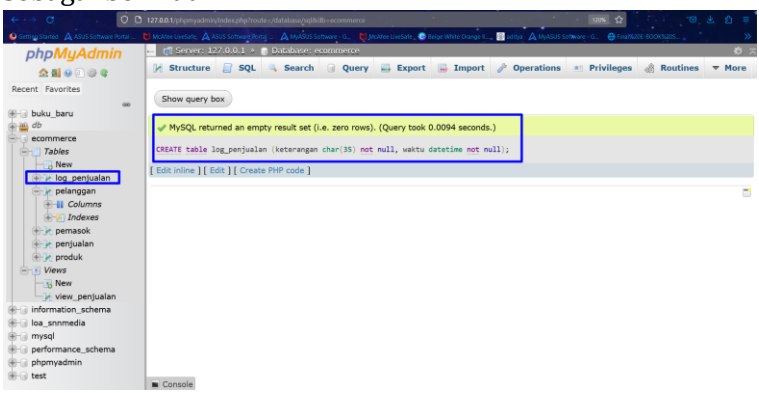
Nama Field	Tipe Data	Length	Keterangan
keterangan	*Char*	35	*Not Null*
waktu	*Datetime*		*Not Null*

Berdasarkan tabel diatas perintah SQL untuk membuat tabel yaitu.



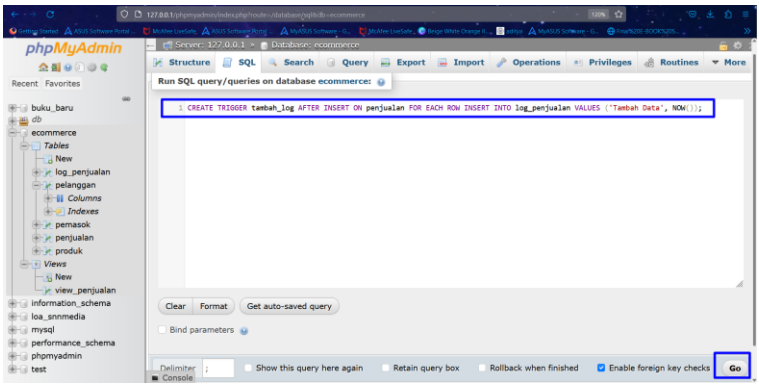
Gambar 7.11. Perintah SQL Membuat Log Penjualan

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* sebagai berikut.



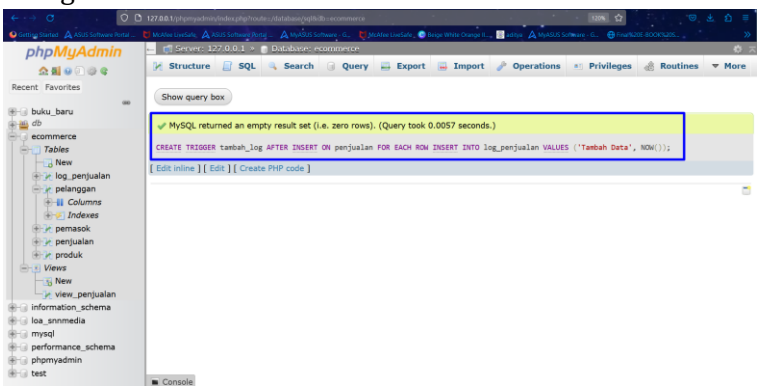
Gambar 7.12. Hasil *Execute* Membuat Log Penjualan

Proses selanjutnya kita akan membuat *trigger* dari log penjualan, perintah SQL membuat *trigger* sebagai berikut.



Gambar 7.13. Perintah SQL Membuat *Trigger*

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* sebagai berikut.



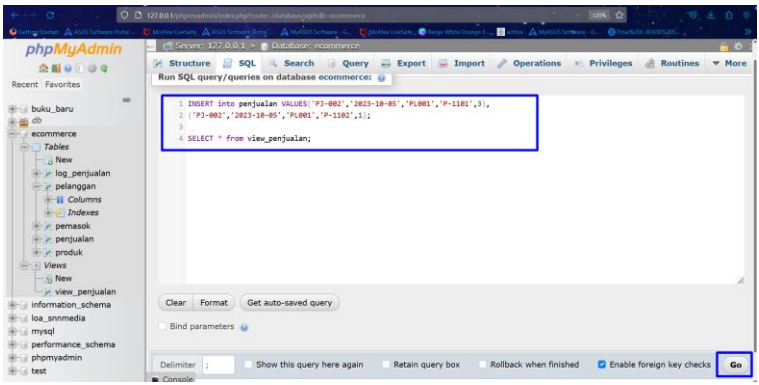
Gambar 7.14. Hasil *Execute* Membuat *Trigger*

Selanjutnya kita akan menguji trigger yang telah kita buat dengan memasukan data penjualan sebagai berikut.

Tabel 7.3. Tambah Data Penjualan

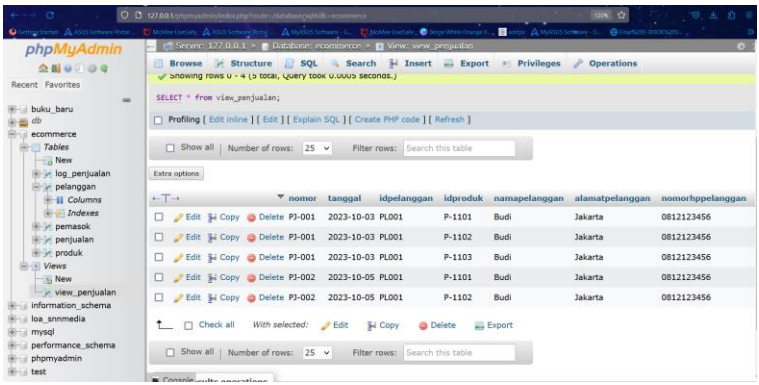
Nomor	Tanggal	ID Pelanggan	ID Produk	Jumlah Beli
PJ-002	2023-10-05	PL001	P-1101	3
PJ-002	2023-10-05	PL001	P-1102	1

Dari data diatas kita akan memasukan data tersebut dalam tabel penjualan, perintah SQL nya sebagai berikut.



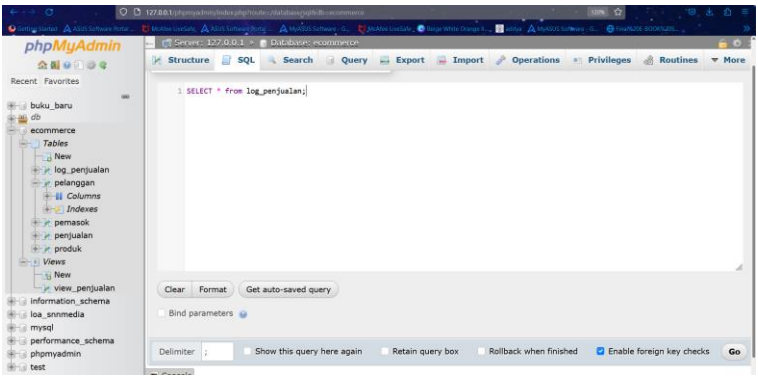
Gambar 7.15. Perintah SQL Tambah Data Penjualan

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



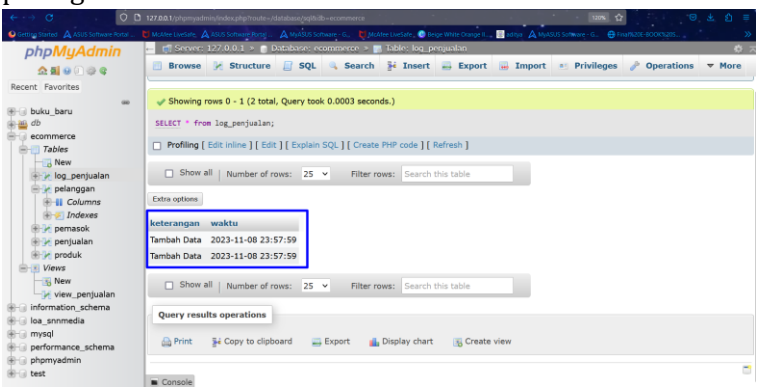
Gambar 7.16. Hasil *Execute* Tambah Data Penjualan

Selanjutnya kita akan lihat trigger yang telah kita buat berhasil menyimpan log penjualan, perintah SQL nya sebagai berikut.



Gambar 7.17. Perintah SQL Menampilkan Log

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



Gambar 7.18. Hasil *Execute* Menampilkan Log

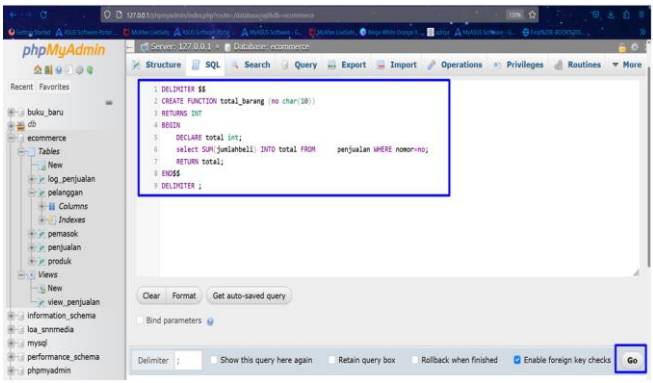
Dari gambar diatas dapat dilihat log yang kita buat berhasil menyimpan history untuk data penjualan.

7.4. IMPLEMENTASI *FUNCTION*

Implementasi fungsi dalam SQL melibatkan beberapa langkah penting. Pertama, pengguna harus merancang fungsi, mendefinisikan parameter dan logika bisnis yang ingin dijalankan oleh fungsi tersebut. Kemudian, fungsi harus dibuat dalam basis data dengan pernyataan *CREATE*

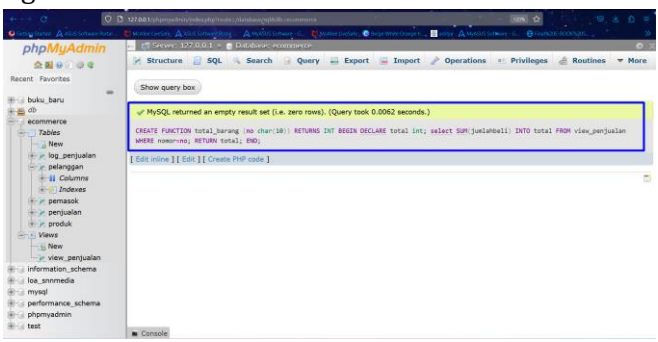
FUNCTION. Setelah dibuat, fungsi dapat digunakan dalam berbagai pernyataan SQL, seperti *SELECT*, *UPDATE*, atau *INSERT*, untuk melakukan perhitungan, manipulasi string, atau logika bisnis lainnya pada data. Fungsi juga dapat diintegrasikan dalam aplikasi yang menggunakan basis data SQL. Penting untuk mengoptimalkan kinerja fungsi, terutama jika fungsi tersebut akan digunakan dalam pernyataan yang kompleks atau dalam volume data besar. Pemantauan dan pemeliharaan fungsi juga penting, terutama jika logika bisnis atau aturan berubah seiring waktu. Dengan implementasi fungsi yang baik, pengguna dapat meningkatkan modularitas, penggunaan ulang, dan efisiensi kode SQL, serta menyederhanakan pernyataan dan pemeliharaan data dalam basis data SQL. Dalam proses implementasi fungsi, juga penting untuk memperhatikan pengelolaan hak akses. Pengguna perlu memastikan bahwa izin akses yang tepat diberikan kepada pengguna atau peran yang akan menggunakan fungsi tersebut, sesuai dengan kebijakan keamanan basis data. Selain itu, fungsi juga dapat memerlukan pemantauan dan pemeliharaan yang berkala, terutama jika ada perubahan dalam struktur data atau aturan bisnis. Implementasi fungsi yang baik dapat memberikan manfaat dalam modularitas dan kinerja pernyataan SQL, memungkinkan pengguna untuk mengotomatisasi perhitungan atau logika bisnis, dan mengoptimalkan akses dan penggunaan data dalam sistem basis data SQL. Dengan pemahaman dan perencanaan yang baik, implementasi fungsi SQL dapat membantu dalam manajemen dan analisis data secara efisien.

Function yang akan kita buat untuk menghitung jumlah barang untuk setiap nomor transaksi penjualan, perintah SQL membuat *function* sebagai berikut.



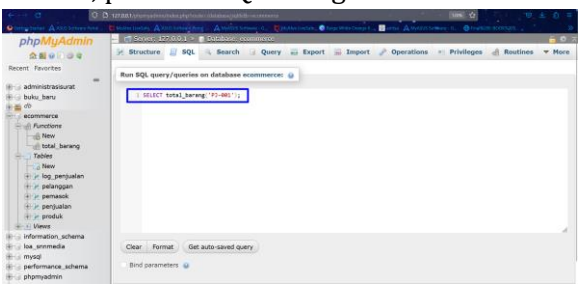
Gambar 7.19. Perintah SQL Membuat *Function*

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



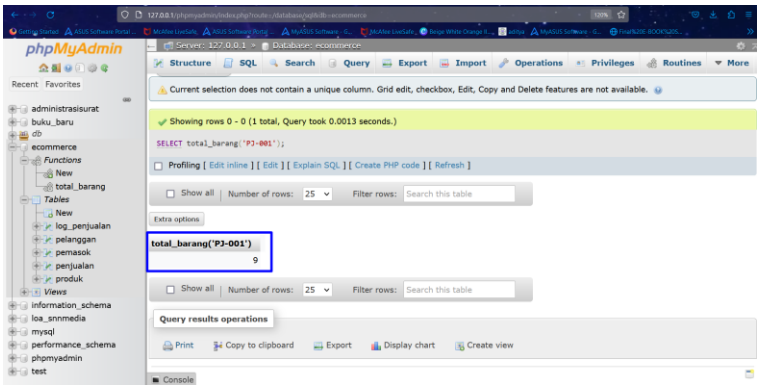
Gambar 7.20. Hasil *Execute* Membuat *Function*

Dari gambar diatas dapat dilihat *function* yang kita buat berhasil. Selanjutnya kita akan menampilkan *function* yang kita buat, perintah SQL sebagai berikut.



Gambar 7.21. Perintah SQL Menampilkan *Function*

Selanjutnya *execute* perintah SQL tersebut, hasil *execute* pada gambar dibawah ini.



Gambar 7.22. Hasil *Execute* Menampilkan *Function*

Dari gambar diatas jumlah barang yang dibeli dengan nomor penjualan **PJ-001** yaitu ada 9.

7.5. SOAL LATIHAN *VIEW*, *TRIGGER*, DAN *FUNCTION*

Bacalah semua soal sebelum memulai mengerjakan. Pastikan Anda memahami aturan dan persyaratan yang ada.

1) Buatlah sebuah *view* dengan nama view_penjualan_pelanggan yang menghubungkan antara tabel penjualan dan tabel pelanggan.
2) Buatlah sebuah *view* dengan nama view_penjualan_produk yang menghubungkan antara tabel penjualan dan tabel produk.

BAB VIII
IMPLEMENTASI PERANCANGAN BASIS DATA (STUDI KASUS)

Implementasi perancangan basis data dalam konteks studi kasus adalah langkah penting dalam mengaplikasikan prinsip-prinsip desain basis data ke dalam situasi dunia nyata. Perancangan basis data melibatkan pembuatan struktur, tabel, relasi, dan aturan yang sesuai untuk menyimpan dan mengelola data dengan efisien. Dalam studi kasus, pendekatan ini menerapkan konsep desain basis data ke dalam kasus konkret kita akan menerapkan perancangan sistem manajemen inventaris barang. Implementasi perancangan basis data dalam studi kasus memungkinkan pengguna untuk memahami bagaimana desain basis data dapat memenuhi kebutuhan bisnis spesifik, mendukung pengembangan aplikasi, dan memfasilitasi analisis data yang relevan. Studi kasus ini memberikan kesempatan untuk menerapkan teori desain basis data ke dalam situasi yang nyata, dan ini merupakan langkah penting dalam mencapai struktur basis data yang efisien, andal, dan sesuai dengan tujuan aplikasi. Selain itu, implementasi perancangan basis data dalam studi kasus juga memungkinkan identifikasi tantangan dan masalah yang mungkin muncul dalam penerapan desain tersebut. Hal ini memungkinkan pengguna untuk merespons dan menyesuaikan desain basis data sesuai dengan kebutuhan yang berkembang dan perubahan situasional. Selama proses ini, pengguna dapat belajar dari pengalaman langsung dan mendapatkan wawasan berharga tentang bagaimana desain basis data berdampak pada efisiensi operasional, analisis data, dan keseluruhan produktivitas. Oleh karena itu, implementasi perancangan basis data dalam studi kasus adalah alat yang efektif dalam

mendukung pengembangan aplikasi yang sukses dan manajemen data yang optimal dalam berbagai konteks bisnis dan teknologi. Dalam dunia yang terus berkembang dengan kebutuhan data yang semakin meningkat, implementasi ini menjadi kunci dalam memastikan bahwa struktur basis data tetap relevan dan mendukung kebutuhan yang berkembang.

Implementasi basis data adalah proses pembuatan dan penggunaan sistem basis data yang melibatkan rancangan, pembuatan, dan pengelolaan basis data untuk menyimpan, mengatur, dan mengakses informasi secara efisien. Implementasi basis data adalah proses yang kompleks dan memerlukan perencanaan yang cermat. Jika dilakukan dengan benar, dapat memberikan manfaat besar dalam menyimpan dan mengelola data secara efisien, memungkinkan organisasi untuk mengambil keputusan berdasarkan data yang akurat dan tepat waktu.

8.1. PEMBUATAN *DATABASE*

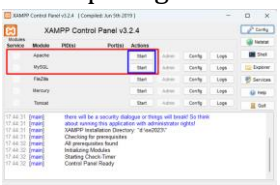
Dalam pembuatan basis data kita akan menggunakan *MySql* sebagai *database* yang akan menyimpan data yang akan kita input kedalam *database*. Tahapan pembuatan *database* menggunakan *mysql* yaitu.

a) Install XAMPP

Langkah pertama kita install terlebih dahulu XAMPP seperti yang telah dibahas dalam bab 4 sebelumnya.

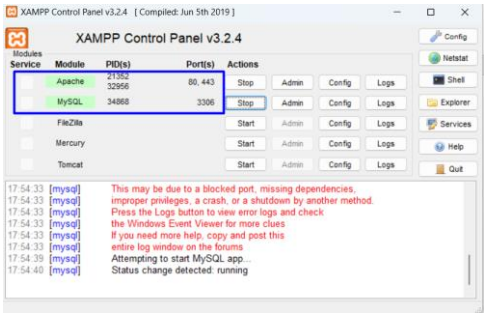
b) Buka XAMPP *Control Panel*

Langkah selanjutnya setelah instalasi selesai buka XAMPP *control panel* seperti gambar berikut ini.



Gambar 8.1. Membuka XAMPP *Control Panel*

Setelah berhasil dibuka kita aktifkan *apache* dan *mysql* dengan memilih tombol *start* sehingga setelah diaktifkan tampil seperti berikut.

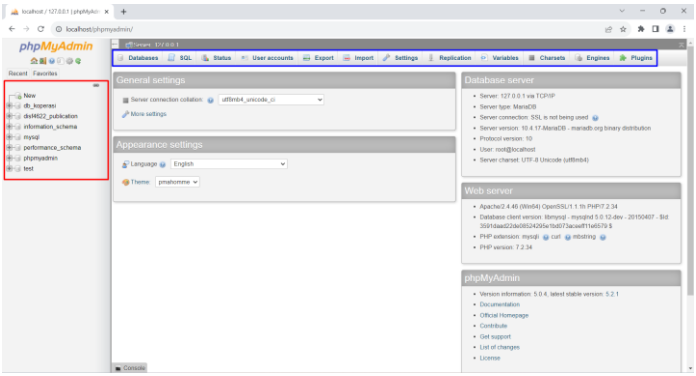


Gambar 8.2. XAMPP *Control Panel* Aktif

Jika *apache* dan *mysql* telah berwarna hijau dan PID serta Port dari *apache* dan *mysql* telah berisi angka maka hal tersebut menandakan *apache* dan *mysql* telah aktif dan siap untuk digunakan.

c) Membuka phpMyAdmin

Tahapan selanjutnya kita akan membuat database mysql dengan menggunakan phpMyAdmin dengan menggunakan web browser, pada URL web browser ketikan localhost/phpMyAdmin dan akhiri dengan enter. Maka tampilan akan seperti berikut ini.

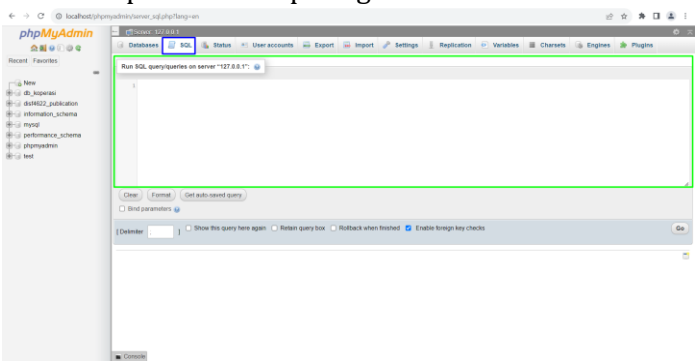


Gambar 8.3. Tampilan *Database MySql*

Tampilan diatas kita berhasil masuk dalam *database MySql*, kotak berwarna merah merupakan *database* yang telah dibuat dalam *MySql*, sedangkan kotak warna biru merupakan menu dalam *database MySqL*.

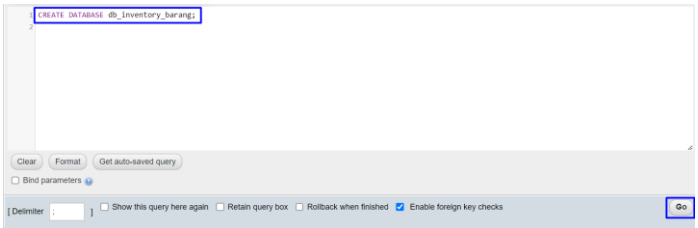
d) Membuat *Database*

Selanjutnya kita akan membuat *database* dengan nama **db_inventory_barang**. Pada menu *database MySql* yang pada gambar 9.3 berwarna biru kita pilih menu *SQL*, maka tampilan akan seperti gambar berikut ini.



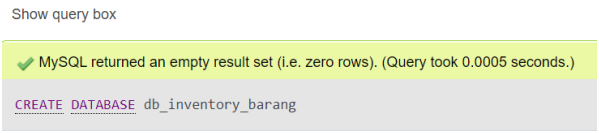
Gambar 8.4. Tampilan Menu SQL

Tampilan diatas kita telah memilih menu *SQL,* dan kotak yang berwarna hijau tempat kita menuliskan atau mengetik perintah SQL. Kita akan membuat database dengan nama **db_inventory_barang**, pada *query sql* kita ketikan perintah DDL yaitu `CREATE DATABASE db_inventory_barang;` untuk membuat *database* sebagai berikut.



Gambar 8.5. Perintah *SQL* Membuat *Database*

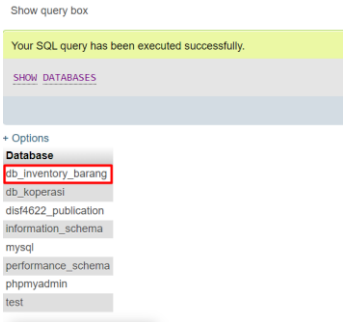
Setelah perintah tersebut diketik selanjutnya pilih tombol ***go*** untuk melakukan *execute* perintah *SQL* yang telah kita buat. Setelah perintah di *execute* maka akan muncul tampilan seperti gambar berikut.



Gambar 8.6. Info Berhasil Membuat *Database*

e) Menampilkan *Database*

Setelah database berhasil dibuat kita akan melihat database yang telah kita buat dalam MySql dengan menggunakan perintah `SHOW DATABASES` dan kembali ketik pada bagian menu SQL dan lakukan *execute*, sehingga tampilan seperti gambar berikut.



Gambar 8.7. Menampilkan *Database*

Database dengan nama **db_inventory_barang** telah berhasil kita buat.

8.2. PEMBUATAN TABEL

Setelah *database* berhasil kita buat selanjutnya kita akan membuat tabel yaitu tabel barang dan tabel pelanggan. Deskripsi tabel barang dan pelanggan yang akan dibuat seperti pada tabel 8.1. dan 8.2. berikut ini.

Tabel 8.1. Deskripsi Tabel Barang

Nama Tabel		tbl_barang	
Nama *Field*	**Tipe Data**	***Length***	**Keterangan**
id_barang	Char	10	*Primary Key*
nama_barang	Varchar	155	*Not Null*
harga_barang	Decimal		*Not Null*
satuan_barang	Char	35	*Not Null*
Stok_barang	Int		*Not Null*

Tabel 8.2. Deskripsi Tabel Pelanggan

Nama Tabel		tbl_pelanggan	
Nama *Field*	**Tipe Data**	***Length***	**Keterangan**
id_pelanggan	Char	10	*Primary Key*
nama_ pelanggan	Varchar	155	*Not Null*
Nomor_hp	Char	16	*Not Null*

Tahapan pembuatan tabel menggunakan *mysql* yaitu.

a) Membuat Tabel Barang

Pilih menu SQL pada menu bar *database MySql* dan ketik perintah seperti gambar berikut ini.

Gambar 8.8. Perintah *SQL* Membuat Tabel Barang

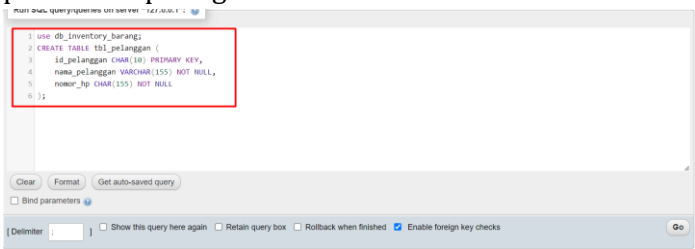
Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.

Gambar 8.9. Hasil *Execute SQL* Membuat Tabel

Hasil tersebut memberikan informasi tabel yang kita buat dengan nama **tbl_barang** telah berhasil dibuat.

b) Membuat Tabel Pelanggan

Pilih menu SQL pada menu bar *database MySql* dan ketik perintah seperti gambar berikut ini.



Gambar 8.10. Perintah *SQL* Membuat Tabel Pelanggan

c) Menampilkan Tabel Yang Telah Dibuat

Setelah tabel barang dan pelanggan berhasil kita buat, selanjutnya kita akan menampilkan seluruh tabel yang telah kita buat dari database yang kita miliki. Perintah menampilkan tabel seperti gambar berikut ini.

Gambar 8.11. Perintah *SQL* Menampilkan Tabel

Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.

Gambar 8.12. Hasil *Execute SQL* Menampilkan Tabel

Hasil tersebut memberikan informasi tabel yang kita buat dengan nama **tbl_barang** dan **tbl_pelanggan** telah berhasil ditampilkan.

8.3. MENAMBAHKAN *RECORD*

Untuk menambahkan *record* atau baris baru ke dalam tabel menggunakan *MySQL*, Anda perlu menggunakan perintah *SQL* ***INSERT INTO***. Untuk data barang yang akan ditambahkan kita akan lihat dalam tabel berikut ini.

Tabel 8.3. Data Tabel Barang

Nama *Field*	Data 1	Data 2	Data 3
id_barang	A-01	A-02	A-03
nama_barang	Pena	Pensil	Buku
harga_barang	2000	1500	3000
satuan_barang	Pcs	Pcs	Pcs
Stok_barang	25	15	35

Perintah menambahkan data barang seperti gambar berikut ini.

```
1 use db_inventory_barang;
2 INSERT INTO tbl_barang (id_barang, nama_barang,harga_barang,satuan_barang,stok_barang)
3 VALUES ('A-01','Pena',2000,'Pcs',25);
```

Gambar 8.13. Perintah *SQL* Menambah *Record* Tabel Barang

Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.

Gambar 8.14. Hasil *Execute SQL* Menambah *Record* Barang

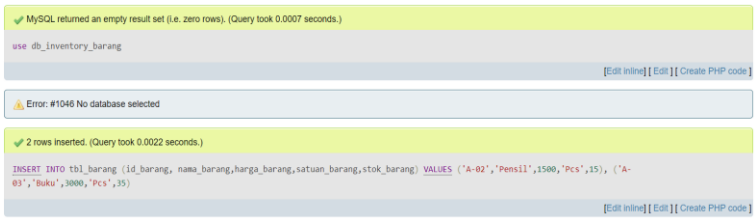
Hasil tersebut memberikan informasi yang kita masukan kedalam **tbl_barang** berhasil disimpan.

Selanjutnya kita akan menambahkan 2 data barang sekaligus menggunakan 1 perintah ***INSERT INTO***, perintah menambahkan 2 data sekaligus seperti gambar berikut ini

```
use db_inventory_barang;
INSERT INTO tbl_barang (id_barang, nama_barang,harga_barang,satuan_barang,stok_barang)
VALUES ('A-02','Pensil',1500,'Pcs',15),
('A-03','Buku',3000,'Pcs',35);
```

Gambar 8.15. Perintah *SQL* Menambah 2 *Record* Tabel Barang

Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.



Gambar 8.16. Hasil *Execute SQL* Menambah 2 *Record* Barang

Hasil tersebut memberikan informasi data yang kita masukan kedalam **tbl_barang** berhasil disimpan.

Selanjutnya silahkan tambahkan data pelanggan seperti pada tabel berikut ini.

Tabel 8.4. Data Tabel Pelanggan

Nama *Field*	Data 1	Data 2	Data 3
id_pelanggan	P-1	P-2	P-3
nama_ pelanggan	Aan	Budi	Rini
Nomor_hp	0812	0813	0852

8.4. MENAMPILKAN *RECORD*

Untuk menampilkan *record* atau data dari tabel di *MySQL*, Anda perlu menggunakan perintah *SQL* ***SELECT***. Berikut ini adalah contoh perintah *SQL* untuk menampilkan data.

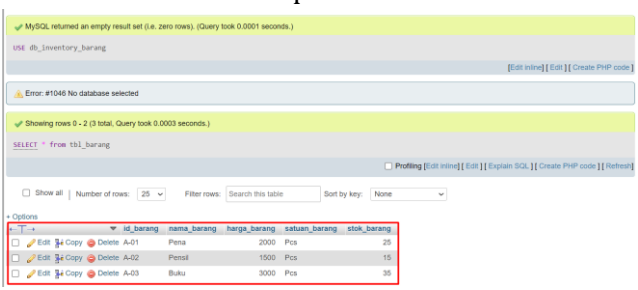
a) Menampilkan Seluruh *Record* Tabel Barang

Perintah untuk menampilkan seluruh data barang seperti gambar berikut ini.

```
1 USE db_inventory_barang;
2 SELECT * from tbl_barang;
```

Gambar 8.17. Perintah *SQL* Menampilkan Seluruh *Record* Tabel Barang

Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.



Gambar 8.18. Hasil *Execute SQL* Menampilkan Seluruh *Record* Tabel Barang

Hasil tersebut memberikan informasi seluruh data dalam **tbl_barang** berhasil ditampilkan.

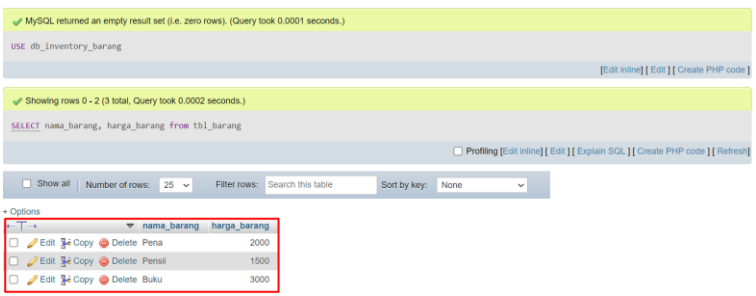
b) Menampilkan Hanya Sebagian *Record* Tabel Barang

Perintah untuk menampilkan sebagian data barang seperti gambar berikut ini.

```
1 USE db_inventory_barang;
2 SELECT nama_barang, harga_barang from tbl_barang;
3
```

Gambar 8.19. Perintah *SQL* Menampilkan Sebagian *Record* Tabel Barang

Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.



Gambar 8.20. Hasil *Execute SQL* Menampilkan Sebagian *Record* Tabel Barang

Hasil tersebut memberikan informasi sebagian data dalam **tbl_barang** berhasil ditampilkan yaitu nama_barang dan harga_barang.

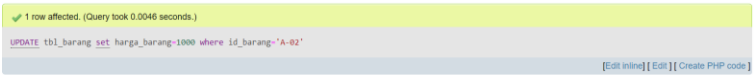
8.5. MENGUBAH *RECORD*

Untuk mengubah atau memperbarui data dalam tabel menggunakan *MySQL*, Anda perlu menggunakan perintah *SQL* ***UPDATE***. Perintah mengubah data barang seperti gambar berikut ini.

```
1 UPDATE tbl_barang set harga_barang=1000
2 where id_barang='A-02';
```

Gambar 8.21. Perintah *SQL* Ubah *Record* Tabel Barang

Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.



Gambar 8.22. Hasil *Execute SQL* Ubah *Record* Barang

Hasil tersebut memberikan informasi yang kita masukan kedalam **tbl_barang** berhasil diubah.

8.6. MENGHAPUS *RECORD*

Untuk menghapus *record* atau data dari tabel menggunakan *MySQL*, Anda perlu menggunakan perintah *SQL* ***DELETE***. Perintah menghapus data barang seperti gambar berikut ini.

```
1 DELETE from tbl_barang where id_barang='A-01';
2 SELECT * FROM tbl_barang;
3
```

Gambar 8.23. Perintah *SQL* Hapus *Record* Tabel Barang

Terdapat 2 perintah yaitu *DELETE* dan *SELECT* hal tersebut dimaksudkan setelah menghapus kita langsung menampilkan data barang. Setelah perintah tersebut diketik selanjutnya pilih ***Go*** maka akan muncul tampilan berikut ini.

Gambar 8.24. Hasil *Execute SQL* Hapus *Record* Barang

Hasil tersebut memberikan informasi yang kita masukan kedalam **tbl_barang** berhasil hapus.

Hellerstein, J. M., Stonebraker, M., & Hamilton, J. (2007). Architecture of a database system. *Foundations and Trends® in Databases*, *1*(2), 141–259.

Huang, J., Antova, L., Koch, C., & Olteanu, D. (2009). MayBMS: a probabilistic database management system. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 1071–1074.

McHugh, J., Abiteboul, S., Goldman, R., Quass, D., & Widom, J. (1997). Lore: A database management system for semistructured data. *ACM Sigmod Record*, *26*(3), 54–66.

Pamungkas, C. A. (2017). *Pengantar dan Implementasi Basis Data*. Deepublish.

Rosa, A. ., & Salahudin, M. (2014). *Rekayasa Perangkat Lunak : Terstruktur dan berorientasi objek*. Informatika.

Sumathi, S., & Esakkirajan, S. (2007). *Fundamentals of relational database management systems* (Vol. 47). Springer.

Miswar Papuangan, S.Pd, M.Cs.

Penulis lahir di Capalulu pada tahun 1988, merupakan Dosen Program Studi Teknik Informatika Fakultas Teknik, Universitas Pasifik Morotai. Minat bidang riset adalah Sistem Cerdas, Sistem Informasi, dan *Decision Support System*. Penulis pernah memperoleh Hibah Penelitian Kompetitif Nasional sebanyak 2 kali dan Hibah Pengabdian 1 kali dari Kemenristekdikti.

Mudar Safi, S.T, M.Eng.

Penulis lahir di Ternate, dan menyelesaikan Pendidikan S1 di Universitas Khairun Ternate Program Studi Teknik Elektro pada tahun 2008, selanjutnya menyelesaikan S2 nya di Universitas Gadjah Mada Yogyakarta Program Studi Teknik Elektro (Teknologi Informasi) tahun 2016. Saat ini menjadi Dosen Tetap di Akademi Ilmu Komputer (AIKOM) Ternate dan mengampuh beberapa Matakuliah antara lain, Sistem Basis Data, Teknik Digital, Sistem Operasi, Pemograman Web.

Yani Sugiyani,MM., M.Kom

Lahir di Bandung pada tahun 1978 dan menyelesaikan studinya di bidang Sistem Informasi, memulai Karir pada tahun 1998 sebagai Programmer dan Mulai Mengajar Mata Kuliah Struktur Data, Basis Data mulai tahun 2003, Tahun 2022 Menjadi Ketua Program Studi Teknik Informatika di Universitas Muhammadiyah Tangerang. Memperoleh Hibah Penelitian dosen Pemula pada tahun 2015 dibidang E-Learning, Menulis Buku dengan judul "Kewirausahaan" pada tahun 2022 dan aktif di organisasi INDOCEISS provinsi Banten Periode 2021 – 2025 sebagai Ketua divisi Humas.

Arie Qur'ania, S.Kom., M.Kom.

Penulis lahir di Bogor pada tahun 1976, merupakan Dosen program studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Pakuan. Minat bidang pendidikan penulis adalah *Artificial Intelligence*, Data Mining, Data *Warehouse*, *Image Processing*, dan *Computer Vision*. Penulis pernah memperoleh hibah penelitian Kompetitif Nasional dari Kemenristekdikti sebanyak 4 kali, serta hibah pengabdian sebanyak 2 kali dan aktif dalam penelitian dan pengabdian internal kampus. Penulis telah menerbitkan hasil penelitiannya di jurnal Internasional bereputasi dan jurnal Nasional Terakreditasi serta mendapatkan HKI dari luaran penelitian. Buku yang telah diterbitkan oleh penulis adalah Kewirausahaan dan Etika Profesi Kiat menjadi seorang Pengusaha pada tahun 2020, serta buku ajar Pengolahan Citra Penerapan pada *Smart Farming* pada tahun 2019.

Sumanto, S.Kom, M.Kom.

Lahir di Bojonegoro pada tahun 1984. Merupakan Dosen Fakultas Teknik dan Informatika Universitas Bina Sarana Informatika. Minat bidang pendidikan penulis adalah *Signal Processing*, *Image Processing*, Data *Mining*, dan *Artificial Intelligence*. Penulis telah menerbitkan hasil penelitiannya di Jurnal Internasional Bereputasi dan Jurnal Nasional Terakreditasi. Buku yang telah diterbitkan oleh penulis adalah Pemotongan dan penghasilan pajak perbaikan pada tahun 2021, Mudah Memahami Teknologi Informasi dan Komunikasi Pada Tahun 2021, Dasar-dasar pembelajaran mesin: *foundations of machine learning* Pada Tahun 2023.

Basis Data (*Database*)	Kumpulan data yang terorganisir dan tersimpan dalam satu lokasi yang dapat diakses dan dikelola.
DBMS (*Database Management System*)	Perangkat lunak yang digunakan untuk mengelola, menyimpan, dan mengakses data dalam basis data.
Table	Struktur dasar dalam basis data yang mengorganisasi data dalam baris dan kolom.
Primary Key	Sebuah kolom atau kumpulan kolom yang unik dan digunakan untuk mengidentifikasi setiap baris dalam tabel.
Structured Query Language	Bahasa pemrograman yang digunakan untuk mengakses dan memanipulasi data dalam basis data.
Normalization	Proses merancang basis data untuk menghindari redundansi data dan memastikan konsistensi.
Query	Permintaan yang diajukan ke basis data untuk mengambil atau memanipulasi data.

Buku Teks

SISTEM BASIS DATA

Sistem basis data adalah infrastruktur perangkat lunak yang dirancang untuk mengelola, menyimpan, dan mengakses data dengan efisien. Ini berperan penting dalam pengorganisasian dan pengelolaan data dalam suatu organisasi atau aplikasi. Sistem basis data memungkinkan data untuk diatur dalam struktur yang terstruktur, memfasilitasi pencarian dan pengambilan data yang cepat, serta memungkinkan pengguna untuk melakukan berbagai operasi seperti penambahan, penghapusan, atau pembaruan data. Pentingnya sistem basis data terletak pada kemampuannya untuk menjaga integritas dan konsistensi data, meningkatkan keamanan, dan mendukung pengambilan keputusan yang lebih baik dalam berbagai konteks, mulai dari bisnis hingga ilmu pengetahuan.

Sistem basis data dapat beroperasi dalam berbagai lingkungan, termasuk basis data relasional yang populer, basis data NoSQL yang fleksibel, dan berbagai sistem khusus lainnya. Selain itu, konsep normalisasi dan desain basis data yang baik merupakan komponen kunci dalam memastikan bahwa data disimpan dengan efisien dan dapat diakses secara efektif. Dengan teknologi yang terus berkembang, sistem basis data terus mengalami perbaikan dan peningkatan untuk mengatasi tuntutan data yang semakin kompleks dan jumlah data yang semakin besar dalam dunia modern. Sistem basis data adalah dasar penting dalam pengelolaan data yang efisien dan menjadi elemen utama dalam aplikasi dan sistem informasi masa kini.

Jalan Gatot Subroto No 47 LK I, Desa/Kelurahan Tanjunggading, Kec. Kedamaian, Kota Bandar Lampung, Provinsi Lampung
0821-8031-8941
cvkeranjangteknologimedia@gmail.com
https://ebook.kertekmedia.com

ISBN 978-623-09-6519-7 (PDF)