

MODUL PRAKTIKUM

SISTEM BASIS DATA



Disusun oleh :

Fatah Yasin Al Irsyadi, S.T., M.T.

Yusuf Sulistyo Nugroho, S.T., M.Eng.

Endang Wahyu Pamungkas, S.Kom., M.Kom.

PROGRAM STUDI INFORMATIKA
FAKULTAS KOMUNIKASI DAN INFORMATIKA
UNIVERSITAS MUHAMMADIYAH SURAKARTA

DAFTAR ISI

Modul 1 Pengenalan dan Instalasi MySQL	3
A. Tujuan	3
B. Landasan Teori.....	3
C. Alat dan Bahan.....	7
D. Langkah-langkah Praktikum	8
E. Tugas.....	12
Modul 2 Perancangan Basis Data	13
A. Tujuan	13
B. Landasan Teori.....	13
C. Alat dan Bahan.....	18
D. Langkah-langkah Praktikum	18
E. Tugas.....	20
Modul 3 Perancangan E-R Diagram dengan DBDesigner.....	21
A. Tujuan	21
B. Landasan Teori.....	21
C. Alat dan Bahan.....	21
D. Langkah-langkah Praktikum	21
E. Tugas.....	22
Modul 4 Data Definition Language (DDL)	23
A. Tujuan	23
B. Landasan Teori.....	23
C. Alat dan Bahan.....	26
D. Langkah-langkah Praktikum	27
E. Tugas.....	29
Modul 5 Data Manipulation Language (DML).....	30
A. Tujuan	30
B. Landasan Teori.....	30
C. Alat dan Bahan	31
D. Langkah-langkah praktikum	31
E. Tugas.....	35

Modul 6 Query Standart dan Variasinya.....	36
A. Tujuan	36
B. Landasan Teori.....	36
C. Alat dan Bahan.....	41
D. Langkah-langkah Praktikum	41
E. Tugas.....	42
Modul 7 Agregasi	43
A. Tujuan	43
B. Landasan Teori.....	43
C. Alat dan Bahan.....	47
D. Langkah-langkah Praktikum	47
E. Tugas.....	48
Modul 8 Data Control Language	49
A. Tujuan	49
B. Landasan Teori.....	49
C. Alat dan Bahan.....	52
D. Langkah-langkah Praktikum	52
E. Tugas.....	53
Modul 9 SubQuery.....	54
A. Tujuan	54
B. Landasan Teori.....	54
C. Alat dan Bahan.....	58
D. Langkah Praktikum.....	58
E. Tugas.....	59
Modul 10 MySQL dan Python.....	60
A. Tujuan	60
B. Landasan Teori.....	60
C. Alat dan Bahan.....	63
D. Langkah-Langkah Praktikum.....	63
E. Tugas.....	64

Modul 1

Pengenalan dan Instalasi MySQL

A. Tujuan

1. Mahasiswa memahami cara instalasi MySQL.
2. Mahasiswa memahami cara konfigurasi MySQL

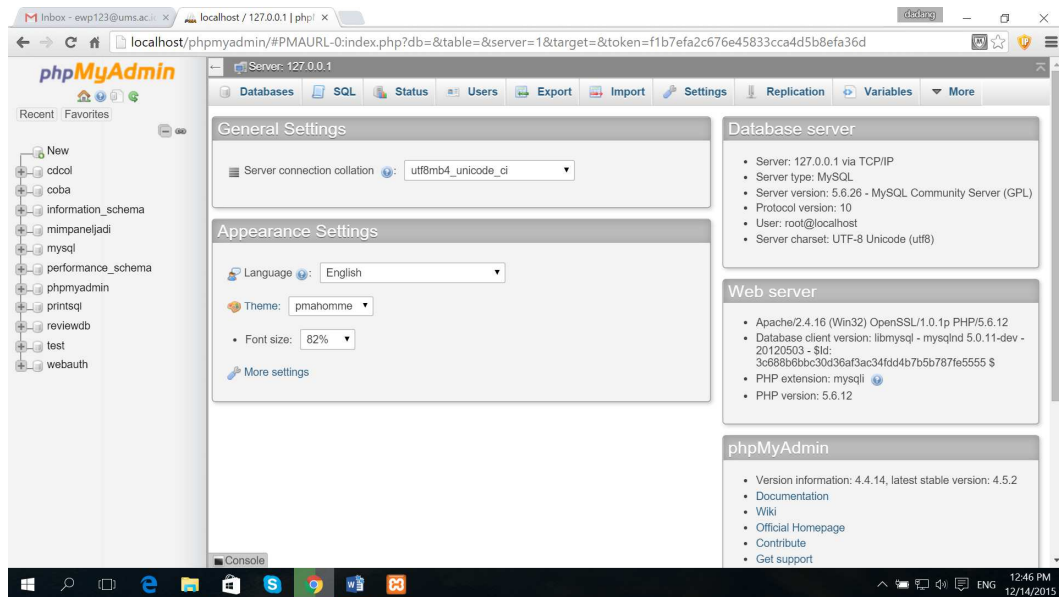
B. Landasan Teori

1. MySQL

MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL atau DBMS yang multi-thread, multi-user, dengan sekitar 6 juta instalasi seluruh dunia. MySQL AB membuat MySQL tersedia sebagai perangkat lunak gratis di bawah lisensi GNU General Public License (GPL), tetapi mereka juga menjual di bawah lisensi komersial untuk kasus-kasus di mana penggunaannya tidak cocok dengan penggunaan GPL.

MySQL adalah sebuah implementasi dari sistem manajemen basis data relational (RDBMS) yang didistribusikan secara gratis. Setiap pengguna dapat secara bebas menggunakan MySQL, namun dengan batasan bahwa perangkat lunak tersebut tidak boleh dijadikan produk turunan yang bersifat komersial. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam basis data yang telah ada sebelumnya. SQL (Structured Query Language) merupakan sebuah konsep pengoperasian basis data, terutama untuk pemilihan atau seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis.

Saat ini versi open source MySQL yang dapat didownload dan digunakan secara gratis adalah MySQL Community Server 5.529. Versi ini merupakan versi MySQL yang sering digunakan dalam pengembangan website. Biasanya dalam paket apache server XAMPP di Windows terdapat MySQL. Pada XAMPP, MySQL dikendalikan secara administratif menggunakan Bahasa PHP yang kemudian user interface nya dalam bentuk phpMyAdmin.



Gambar 1.1 Tampilan Antarmuka PHPMYAdmin

2. Instalasi MySQL pada XAMPP

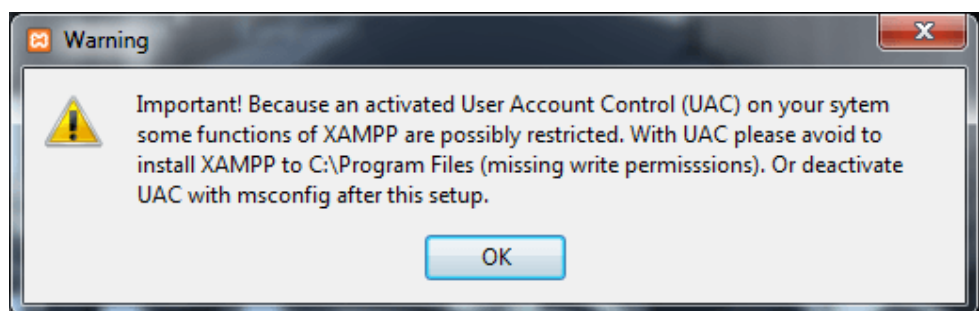
Ikuti langkah-langkah berikut ini untuk melakukan instalasi XAMPP. Untuk mendapatkan installer XAMPP anda dapat mengakses ke halaman <https://www.apachefriends.org/download.html>.

➤ Langkah 1.

Non-aktifkan anti-virus karena dapat menyebabkan beberapa komponen XAMPP yang tidak bias di install dengan lancer.

➤ Langkah 2.

Untuk pengguna Windows 7 atau Windows 8 anda akan melihat jendela pop up, peringatan tentang User Account Control (UAC) yang aktif pada sistem. Klik “OK” untuk melanjutkan instalasi.



Gambar 1.2. User Account Control Warning

➤ Langkah 3.

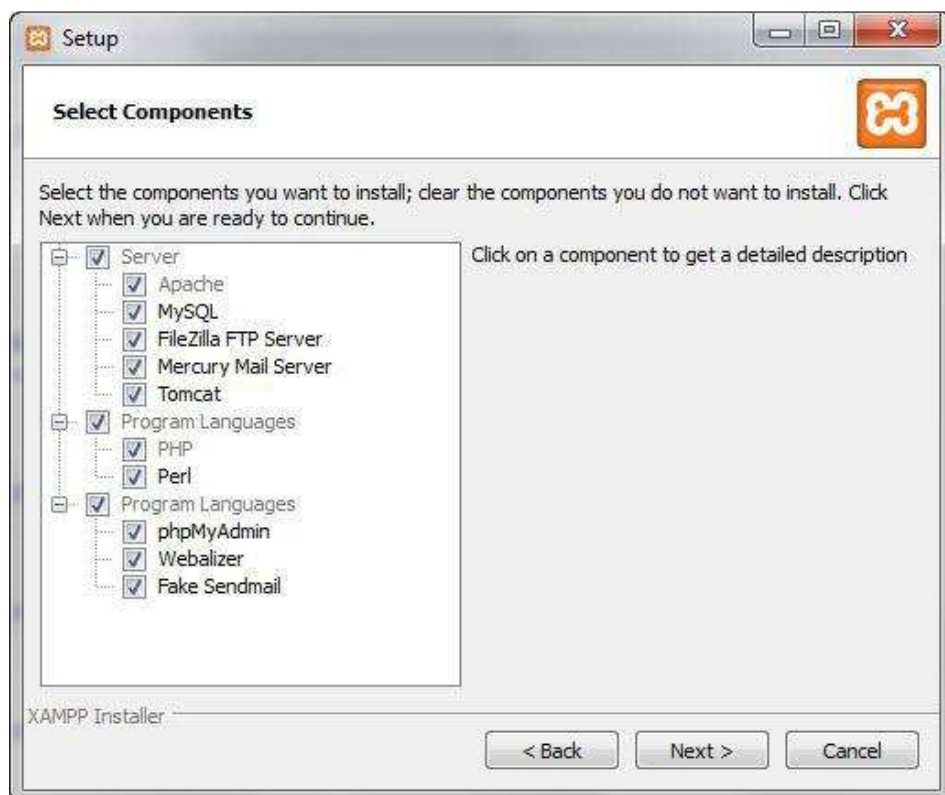
Mulai proses instalasi dengan klik dua kali pada installer XAMPP. Klik “Next” setelah splash screen.



Gambar 1.3. Splash Screen XAMPP

➤ Langkah 4.

Di sini, kita dapat memilih komponen apa saja yang ingin kita install. Pilih pilihan default dan pilih “Next”.



Gambar 1.4. Pilihan Komponen yang Akan diinstall

➤ Langkah 5.

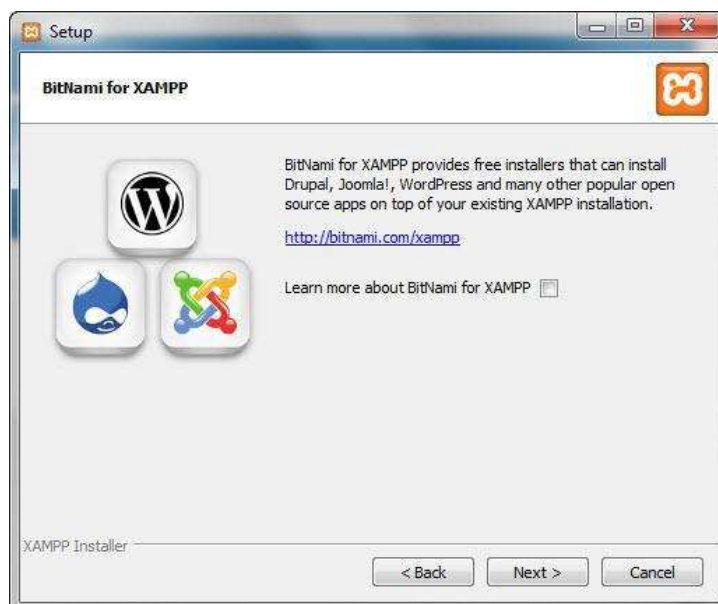
Pilih folder sebagai tempat XAMPP akan diinstal, di folder ini kita akan menyimpan semua file aplikasi web kita, jadi pastikan untuk memilih drive yang masih memiliki banyak ruang (space).



Gambar 1.5. Pemilihan Folder Instalasi

➤ Langkah 6.

Layar berikutnya adalah promo untuk Bitnami, sebuah tool aplikasi untuk server perangkat lunak. Hapus centang pada kotak “Learn more about Bitnami for XAMPP”.



Gambar 1.6 Promo Bitnami

➤ Langkah 7.

Sekarang Setup sudah siap untuk menginstall XAMPP. Klik Next dan tunggu instaler untuk membongkar paket-nya dan memasang komponen yang dipilih. Mungkin memakan waktu beberapa menit. Nanti mungkin kita akan diminta untuk menyetujui akses Firewall untuk komponen tertentu (seperti Apache) selama proses instalasi.

➤ Langkah 8.

Proses Install sudah selesai! Pilih Kotak centang 'Do you want to start the Control Panel now?' untuk membuka panel kontrol XAMPP.



Gambar 1.7 Finish Installation

C. Alat dan Bahan

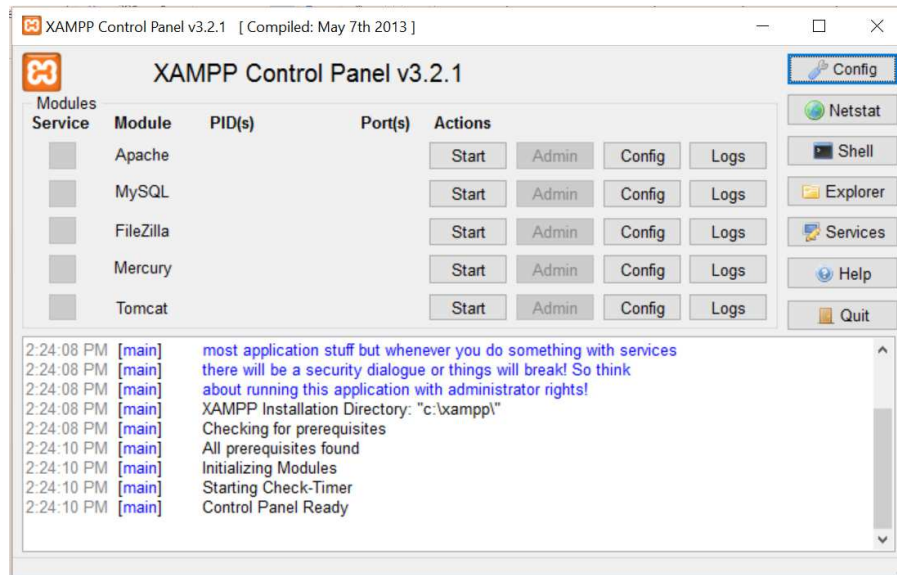
1. Program aplikasi XAMPP yang telah terinstall.
2. Web Browser.
3. Modul Praktikum Sistem Basis Data.

D. Langkah-langkah Praktikum

1. Mengakses PhpMyAdmin

➤ Langkah 1.

Buka XAMPP Control Panel.

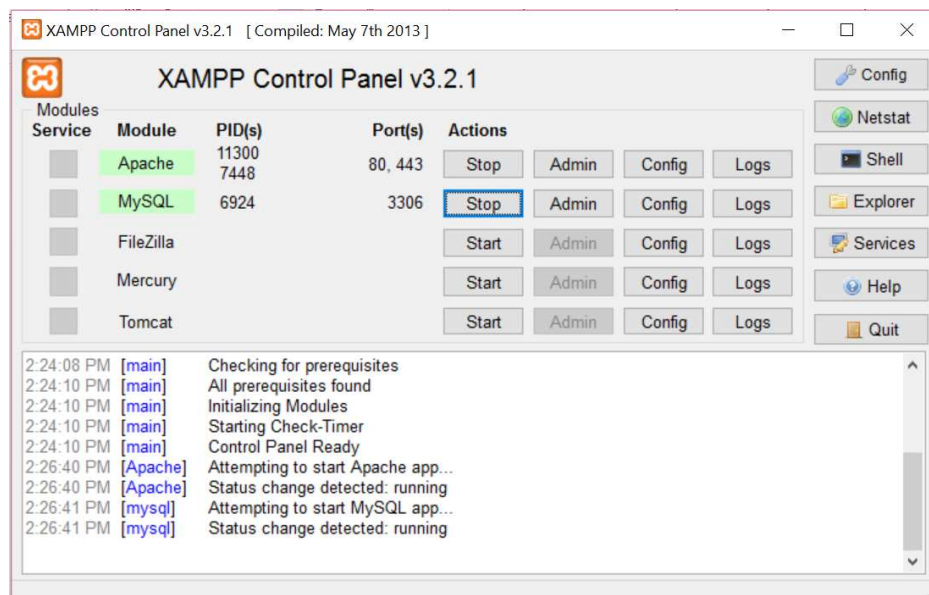


Gambar 1.8 XAMPP Control Panel

➤ Langkah 2.

Jalankan Apache Server dan MySQL Server dengan menekan tombol “Start”.

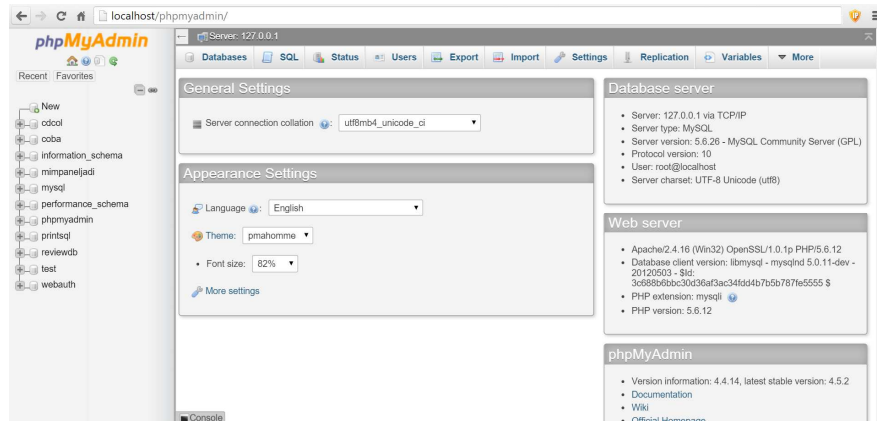
Tunggu hingga muncul warna hijau pada nama Module.



Gambar 1.9. Service telah berjalan

➤ Langkah 3.

Buka web browser anda dan ketikkan <http://localhost/phpmyadmin/>.



Gambar 1.10. Tampilan halaman depan phpMyAdmin

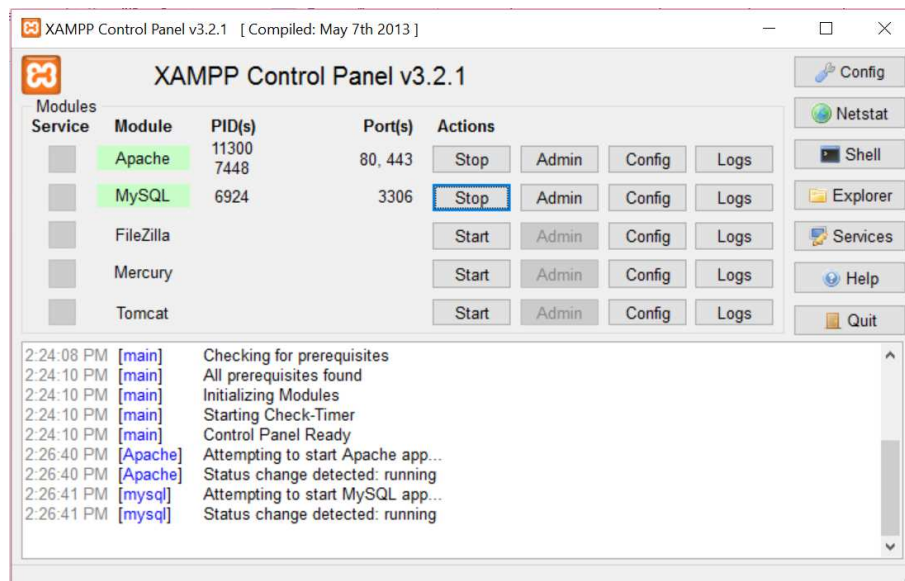
➤ Langkah 4.

Pada halaman PhpMyAdmin ini kita dapat melakukan pembangunan basis data dan juga melakukan manipulasi isi basis data dengan MySQL. PhpMyAdmin merupakan halaman GUI administratif MySQL server yang saat ini paling banyak dipakai dalam pengembangan aplikasi berbasis web.

2. Mengakses MySQL lewat command prompt

➤ Langkah 1.

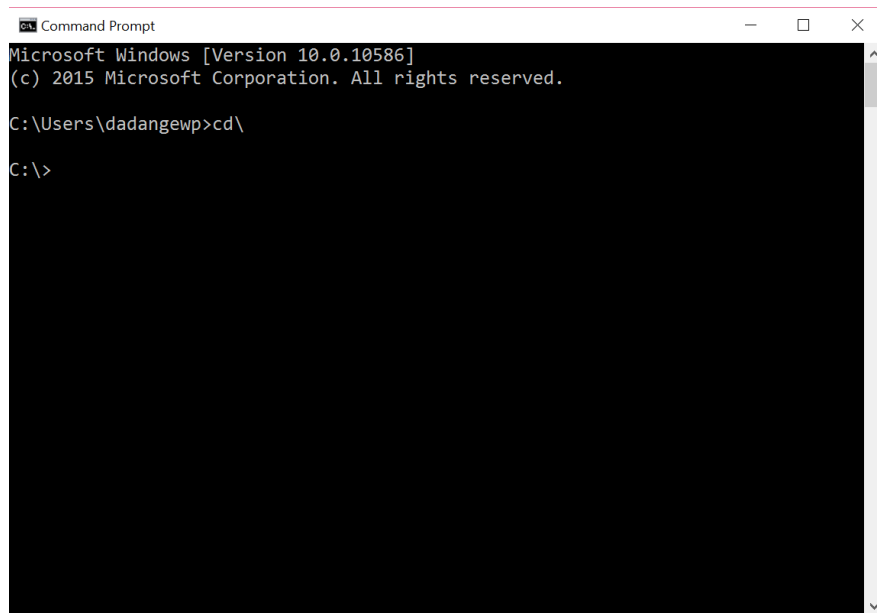
Pastikan bahwa server MySQL telah berjalan.



Gambar 1.11. Service yang sedang berjalan

➤ Langkah 2.

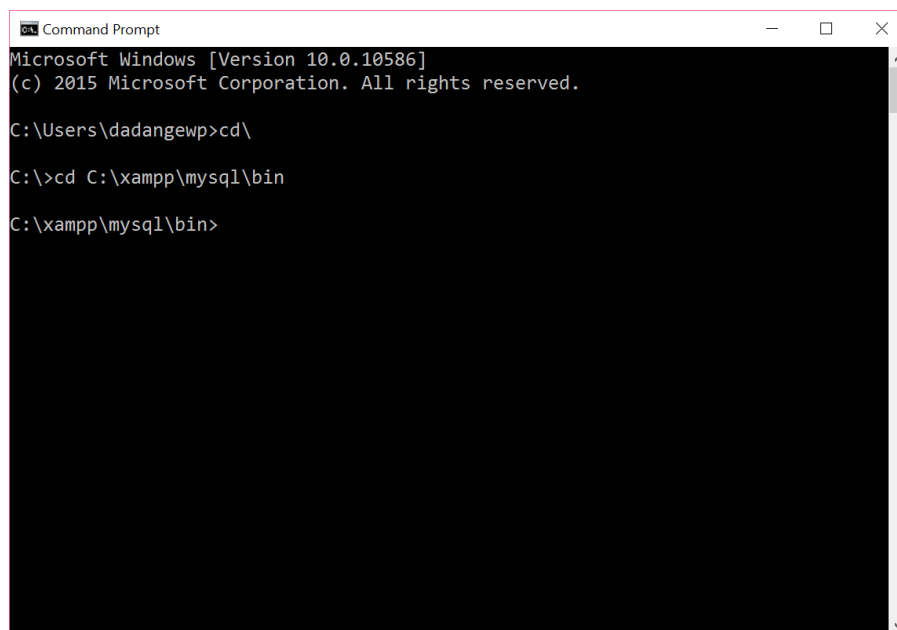
Buka command prompt dan ketik 'cd\' dan tekan "Enter". Sehingga anda akan berada di direktori (C:\).



Gambar 1.12. Tampilan Window Command Prompt

➤ Langkah 3.

setelah itu arahkan ke folder C:\xampp\mysql\bin, caranya ketik 'cd C:\xampp\mysql\bin' (tanpa ' ') kemudian tekan tombol Enter.



Gambar 1.13. Arahkan ke directory C:\xampp\mysql\bin

➤ Langkah 4.

setelah berada di dalam folder C:\xampp\mysql\bin, baru anda dapat mengakses mysql. ketik : 'mysql -u root -p' (tanpa ' ') kemudian tekan tombol Enter. Masukkan password (jika ada) kemudian klik tombol Enter lagi (seara default tidak ada password untuk root).

```
Command Prompt - mysql -u root -p
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\dadangewp>cd\

C:\>cd C:\xampp\mysql\bin

C:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Gambar 1.15. Login ke MySQL

➤ Langkah 5.

Setelah tampilan seperti di Gambar 1.15 berarti anda telah berhasil masuk ke MySQL sebagai root user. Untuk melihat database yang ada pada server anda dapat mengetikan 'show databases;'. Jangan lupa untuk selalu mengakhiri command dengan ';'.

```
Command Prompt - mysql -u root -p
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| cdcol |
| coba |
| mimpaneljadi |
| mysql |
| performance_schema |
| phpmyadmin |
| printsql |
| reviewdb |
| test |
| webauth |
+-----+
11 rows in set (0.00 sec)

mysql>
```

Gambar 1.16. Menampilkan Database yang ada di Server

E. Tugas

1. Jelaskan mengapa dibutuhkan data!
2. Jelaskan manfaat database dan contohnya!
3. Untuk menentukan jenis database yang digunakan, apa yang menjadi acuan dalam pemilihan database tersebut?
4. Jelaskan istilah atau terminology yang digunakan dalam Database (database, table, field, record)
5. Bandingkan perbedaan pengolahan data secara manual dengan menggunakan system database.
6. Mengapa dibutuhkan DBMS?
7. Pada percobaan diatas ada beberapa field yang tipe data dan ukurannya berbeda. Jelaskan!

Modul 2

Perancangan Basis Data

A. Tujuan

1. Mahasiswa memahami makna entitas, atribut, dan relasi.
2. Mahasiswa memahami tahap-tahap normalisasi.
3. Mahasiswa mampu merancang basis data melalui tahap-tahap perancangannya.

B. Landasan Teori

1. Entitas

Entity adalah suatu obyek yang dapat dikenali dari obyek yang lain. Contoh : seseorang yang khusus, perusahaan, peristiwa, tanaman dan lain-lain. *Entity* mempunyai atribut, contoh : seseorang mempunyai nama dan alamat. Kumpulan *entity* adalah suatu kumpulan *entity* dengan tipe yang sama yang berbagi properti yang sama pula. Contoh : kumpulan orang, perusahaan, tanaman, tempat liburan dan lain-lain.

2. Atribut

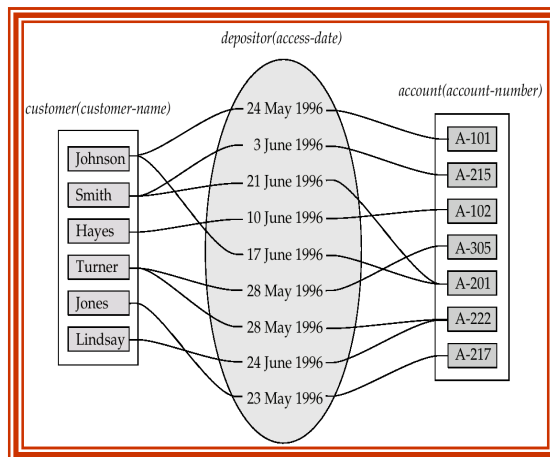
Entity ditampilkan oleh sekumpulan *attribute*, yang mana properti deskriptifnya dikuasai oleh seluruh anggota dalam kumpulan *entity*.

Tipe *attribute* :

1. *Simple* (sederhana) dan *composite* (gabungan) *attributes*.
2. *Single-valued* (satu-fungsi) dan *multi-valued* (multi-fungsi) *attributes*. Contoh : atribut multi-fungsi : nomor telepon
3. *Derived* (turunan) *attributes* : dapat diperhitungkan dari atribut lain Contoh : umur, tanggal kelahiran.

3. Relasi

Relationship adalah kesesuaian antar beberapa *entity*. *Relationship set* adalah hubungan matematika antara *entity* $n > 2$, tiap bagiannya diambil dari satuan *entity* $\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$ dimana (e_1, e_2, \dots, e_n) adalah *relationship*. *Attribute* dapat merupakan properti dari *relationship set*.



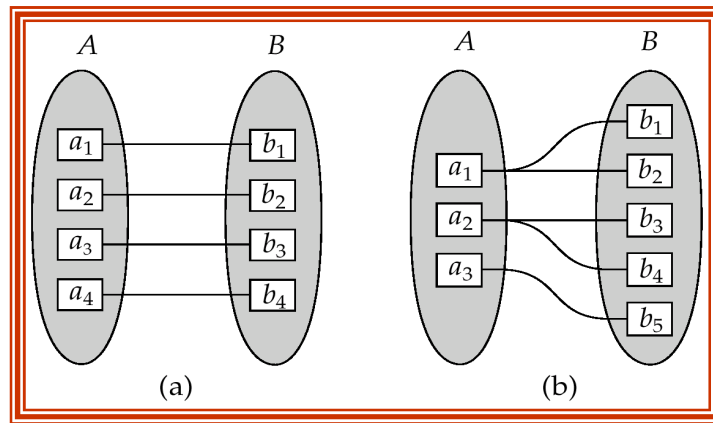
Sebagai contoh: *depositor* merupakan *relationship set* antara *entity sets* *customer* dan *account* mungkin beratribut *access-date*. Mengacu pada jumlah *entity sets* yang terlibat dalam *relationship set*. *Relationship sets* yang melibatkan dua *entity sets* adalah *binary* (atau tingkat dua). Umumnya, hampir semua *relationship sets* dalam sistem database adalah *binary*. *Relationship sets* mungkin melibatkan lebih dari dua *entity sets*.

Contoh : misal seorang pegawai bank bekerja (bertanggung jawab) dalam beberapa cabang, dengan tugas yang berlainan dalam setiap cabang yang berlainan pula. Maka disini terdapat *relationship set ternary* antara *entity sets* pegawai (*employee*), tugas (*job*) dan cabang (*branch*). *Relationships* antara lebih dari dua *entity sets* sangat jarang.

4. Mapping Cardinalities

Mengungkap jumlah entitas ke entitas yang lain bisa dihubungkan melalui *relationship set*. Cardinalitas pemetaan paling banyak digunakan dalam menggambarkan *relationship sets* biner. Untuk *relationship set* biner cardinalitas pemetaan harus merupakan salah satu dari tipe berikut:

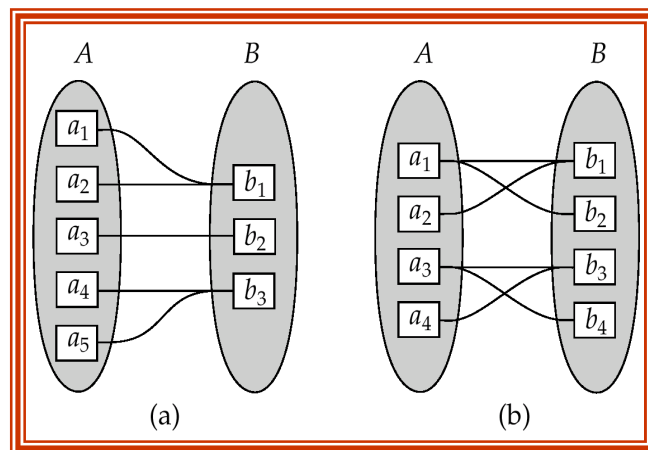
1. *One to one* (satu ke satu)
2. *One to many* (satu ke banyak)
3. *Many to one* (banyak ke satu)
4. *Many to many* (banyak ke banyak)



One to one

One to many

Catatan : Beberapa elemen di A dan B mungkin tidak dipetakan ke elemen manapun di himpunan lain.



Many to one

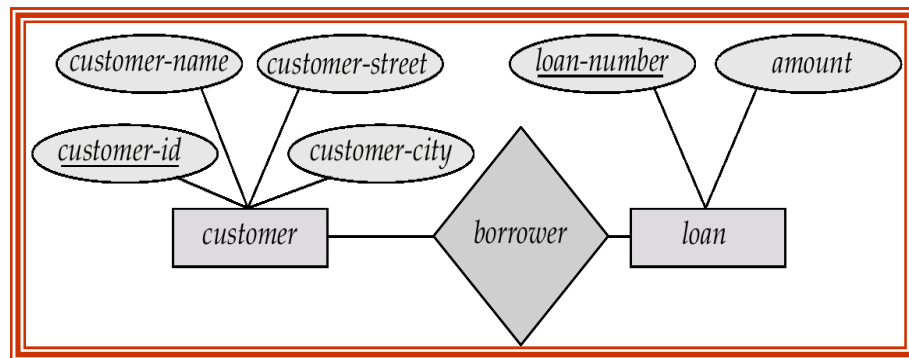
Many to many

Catatan : Beberapa elemen di A dan B mungkin tidak dipetakan pada elemen mana pun di himpunan lain. *Mapping Cardinalities* mempengaruhi *ER Design*

5. Entity Relationship Diagram

Mencerminkan model database: hubungan antara entities (tabel-tabel) dan relationships (hubungan-hubungan) di antara entities tersebut.

Contoh ERD:



- **Rectangles** melambangkan set-set entitas .
- **Diamonds** melambangkan set-set hubungan (*relationship*) .
- **Lines** menghubungkan atribut dengan set-set entitas serta set-set entitas dengan set-set hubungan (*relationship*).
- **Ellipses** mewakili *attributes*
 - **Double ellipses** mewakili atribut *multivalued* .
 - **Dashed ellipses** menunjukkan sifat yang didapat..
 - **Underline** menunjukkan sifat kunci pokok (*Primary key*)

6. Aturan untuk Rancangan Database

1. Tiap baris harus berdiri sendiri (independent): tidak tergantung baris-baris yang lain, dan urutan baris tidak mempengaruhi model database.
2. Tiap baris harus unik: tidak boleh ada 2 atau lebih baris yang sama persis.
3. Kolom harus berdiri sendiri (independent): tidak tergantung kolom-kolom yang lain, dan urutan kolom tidak mempengaruhi model database.
4. Nilai tiap kolom harus berupa satu kesatuan: tidak berupa sebuah daftar.

7. Tahap Pembuatan Database

a. Tahap 1 : Tentukan Entitas

Tentukan entities (object-object dasar) yang perlu ada di *database*.

Sifat-sifat *entity*:

- Signifikan: memang perlu disimpan di *database*
- Umum: tidak menunjuk pada sesuatu yang khusus
- Fundamental: dapat berdiri sendiri sebagai *entity* yang dasar dan independent
- Unitary: merupakan satu kesatuan yang tidak dapat dipecah lagi

b. Tahap 2 : Tentukan Atribut

Tentukan attributes (sifat-sifat) masing-masing *entity* sesuai kebutuhan database :

- Tentukan sifat-sifat (*fields* atau kolom) yang dimiliki tiap *entity*, serta tipe datanya.
- *Attribute* yang sesuai harus:
 1. Signifikan: memang penting dan perlu dicatat di dalam database
 2. Bersifat langsung (*direct*), bukan *derived*. Contoh *attribute direct*: tanggal_lahir. Contoh *attribute derived*: umur
- Tentukan *attribute* yang menjadi *Primary Key* untuk *entity* yang bersangkutan.
- Jika satu *attribute* tidak cukup, gabungan beberapa *attribute* bisa menjadi *Composite Primary Key*.
- Jika *Composite Primary Key* banyak (lebih dari 3 *attribute*), sebaiknya menambahkan *attribute* buatan yang menjadi *Primary Key* yang tunggal.

c. Tahap 3 : Tentukan Relasi

Tentukan relationships (hubungan-hubungan) di antara entities tersebut :

- Tentukan jenis hubungan di antara *entity* yang satu dengan entities yang lain.
- Macam hubungan ada 3:
 1. *One-to-one* (1:1)
 2. *One-to-many* (1:n)
 3. *Many-to-many* (m:n)
- Dalam membentuk hubungan di antara 2 entities, tentukan *attribute* mana yang digunakan untuk menghubungkan kedua entities tersebut.
- Tentukan *entity* mana yang menjadi tabel utama, dan *entity* mana yang menjadi tabel kedua.
- *Attribute* (dari tabel utama) yang menghubungkannya dengan tabel kedua menjadi *Foreign Key* di tabel kedua.

d. Tahap 4 : Pembuatan ERD

- Buat *Entity Relationship Diagram* (ERD) berdasarkan hasil dari Tahap 1- 3.
- Ada berbagai macam notasi untuk pembuatan ERD.
- Anda bisa menggunakan *software* khusus untuk menggambar ERD.

e. Tahap 5 : Normalisasi Basis Data

f. Tahap 6 : Implementasi Basis Data

C. Alat dan Bahan

1. Perangkat komputer dengan OS Windows 7.
2. Modul Praktikum Sistem Basis Data

D. Langkah-langkah Praktikum

Rancanglah basis data untuk menyelesaikan permasalahan berikut ini.

Suatu perusahaan software diminta membuat basis data yang akan menangani data-data perbankan. Data-data yang akan ditanganinya adalah: data pribadi mengenai nasabah, data account deposit yang dimiliki oleh nasabah, cabang bank di mana nasabah membuka depositnya, dan data transaksi yang dilakukan nasabah. Nasabah boleh mempunyai lebih dari satu account deposit, dan satu account deposit boleh dimiliki oleh lebih dari satu nasabah sekaligus (*joint account*).

Langkah-langkah perancangan database perbankan :

1. Menentukan entities (object-object dasar) yang perlu ada di database.
 - **nasabah:** menyimpan semua data pribadi semua nasabah
 - **rekening:** menyimpan informasi semua rekening yang telah dibuka
 - **cabang_bank:** menyimpan informasi tentang semua cabang bank
 - **transaksi:** menyimpan informasi tentang semua transaksi yang telah terjadi
2. Menentukan attributes (sifat-sifat) masing-masing *entity* sesuai kebutuhan database
 - **nasabah:**
 - id_nasabah: nomor id untuk nasabah (integer) PK
 - nama_nasabah: nama lengkap nasabah (varchar(45))
 - alamat_nasabah: alamat lengkap nasabah (varchar(255))
 - **rekening:**
 - no_rekening: nomor rekening (integer) PK
 - pin : personal identification number (varchar(10))
 - saldo: jumlah saldo rekening dalam Rp (integer)
 - **cabang_bank:**
 - kode_cabang: kode untuk cabang bank (varchar(10)) PK
 - nama_cabang: nama lengkap cabang bank (varchar(20))
 - alamat_cabang: alamat lengkap cabang bank (varchar(255))
 - **transaksi:**
 - no_transaksi: nomor transaksi (integer) PK
 - jenis_transaksi: kredit atau debit (varchar(10))
 - tanggal: tanggal terjadinya transaksi (date)
 - jumlah: besarnya transaksi dalam Rp (integer)

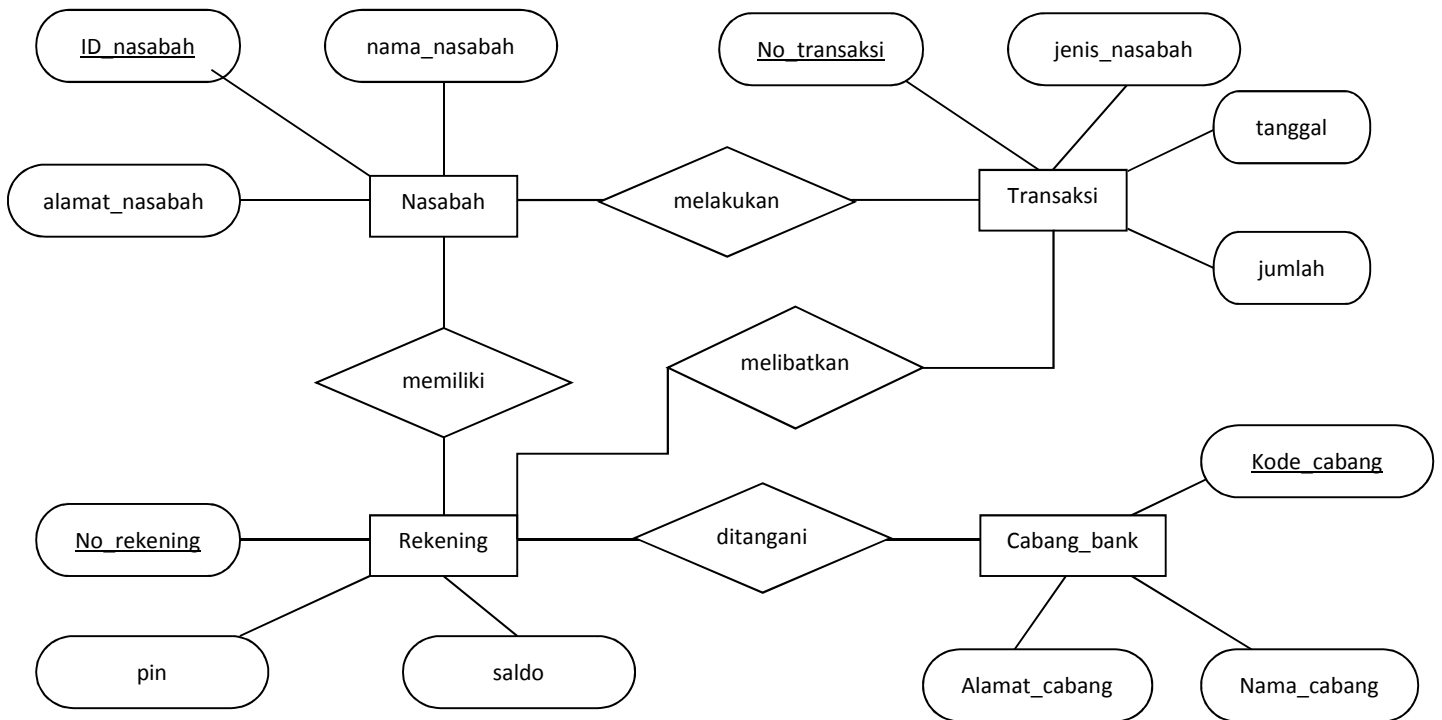
3. Menentukan relationship (hubungan) antar entitas

	nasabah	rekening	cabang_bank	transaksi
nasabah	-	m:n	-	1:n
rekening		-	n:1	1:n
cabang_bank			-	-
transaksi				-

Hubungan

- **nasabah** memiliki **rekening**:
 - Tabel utama: **nasabah, rekening**
 - Tabel kedua: **nasabah_has_rekening**
 - Relationship: Many-to-many (m:n)
 - Attribute penghubung: **id_nasabah, no_rekening** (FK **id_nasabah, no_rekening** di **nasabah_has_rekening**)
- **nasabah** melakukan **transaksi**:
 - Tabel utama: **nasabah**
 - Tabel kedua: **transaksi**
 - Relationship: One-to-many (1:n)
 - Attribute penghubung: **id_nasabah** (FK **id_nasabah** di **transaksi**)
- **cabang_bank** menangani **rekening**:
 - Tabel utama: **cabang_bank**
 - Tabel kedua: **rekening**
 - Relationship: One-to-many (1:n)
 - Attribute penghubung: **kode_cabang** (FK **kode_cabang** di **rekening**)
- **rekening** terlibat dalam **transaksi**:
 - Tabel utama: **rekening**
 - Tabel kedua: **transaksi**
 - Relationship: One-to-many (1:n)
 - Attribute penghubung: **no_rekening** (FK **no_rekening** di **transaksi**)

4. Menggambar ERD Diagram



E. Tugas

1. Buatlah rancangan sebuah database untuk menangani data-data kuliah. Data-data yang akan ditanganinya adalah: data pribadi mengenai mahasiswa, data pribadi mengenai dosen, data mata kuliah dan data ruang kelas. Mahasiswa boleh mengambil lebih dari satu mata kuliah, dan satu mata kuliah boleh diambil oleh lebih dari satu mahasiswa sekaligus (*joint account*). Buatlah ER Diagram manual untuk kasus tersebut dari tahap 1 sampai tahap 4!
2. Ambil contoh sembarang database (harus berbeda untuk setiap mahasiswa). Buatlah rancangan ER Diagram manual database tersebut dari tahap 1 sampai tahap 4, dengan ketentuan database minimal mengandung 4 buah entitas.

Modul 3

Perancangan E-R Diagram dengan DBDesigner

A. Tujuan

1. Mahasiswa mampu merancang basis data melalui tahap-tahap perancangannya.
2. Mahasiswa mampu mewujudkan hasil perancangan basis data ke dalam diagram E-R menggunakan DBDesigner.

B. Landasan Teori

1. Landasan teori pada modul Bab 3.


C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows XP
2. Program aplikasi DBDesigner
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah-langkah Praktikum



1. Menggambar ER Diagram dengan menggunakan DB Designer :

- a. Jalankan program aplikasi DB Designer




- b. Klik button *new table*  kemudian klik pada area kerja sehingga akan menghasilkan tabel baru.

- c. Double klik pada tabel baru untuk membuka tabel editor, ganti nama pada table name dengan nama *nasabah*, kemudian isikan atribut tabel dengan data seperti pada langkah nomor 2 yaitu sebagai berikut :

Coloumn Name	Data Type
id_nasabah	Integer
nama_nasabah	Varchar(45)
alamat_nasabah	Varchar(255)




- d. Klik  pada *coloumn name* id_nasabah untuk mengeset id_nasabah menjadi *primary key* sehingga berubah menjadi .

- e. Klik  untuk menutup table editor sehingga tabel nasabah menjadi :

nasabah
 id_nasabah: INTEGER
 nama_nasabah: VARCHAR(45)
 alamat_nasabah: VARCHAR(255)

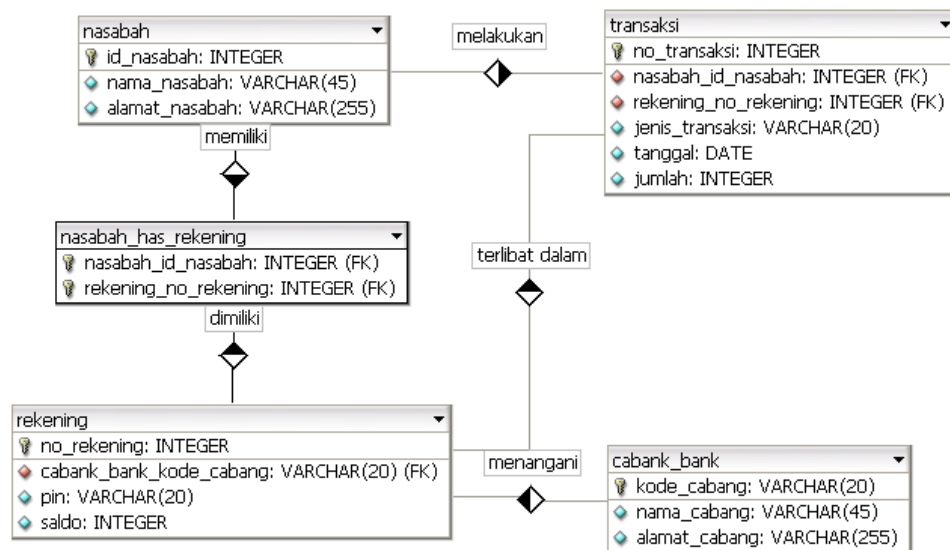
- f. Lakukan langkah b sampai e untuk membuat tabel *rekening*, *cabang_bank* dan *transaksi*.

- g. Setelah semua tabel dibuat, hubungkan setiap tabel dengan tabel lain dengan button sebagai berikut :

Button	Fungsi Relationship
	1:n (<i>one to many</i>)
	1:1 (<i>one to one</i>)
	n:m (<i>many to many</i>)

Keterangan : klik salah satu button kemudian klik tabel yang akan dihubungkan.

- h. Ubah nama relationship dengan membuka relationship editor, sehingga setelah selesai hasil akhir menjadi seperti berikut :



E. Tugas

Dari tugas pada modul 3 soal nomer 1 dan 2, buatlah rancangan basis data dari tahap 1 sampai tahap 4. Gunakan program DB Designer untuk membuat ER Diagram.

Modul 4

Data Definition Language (DDL)

A. Tujuan

1. Mahasiswa mampu membuat struktur tabel-tabel basis data berdasarkan perancangan di modul 3 dan mengimplementasikan tabel-tabel tersebut di basis data.

B. Landasan Teori

Basis data dapat diimplementasikan berdasarkan E-R diagram yang telah dibuat. Implementasi database bisa:

1. Secara manual (dengan perintah SQL 'CREATE TABLE')
 2. Secara semi-manual dengan bantuan client berbasis GUI (MySQL Front, PgAccess, phpPgAdmin, dst.)
 3. Secara otomatis dengan CASE Tools (DBDesigner)
- Entity dan Relationships
Macam *Relationships* :
 1. *One-to-One* : menggunakan notasi 1:1
 - a. Satu *record* di *entity* pertama berhubungan dengan hanya satu *record* di *entity* kedua, dan demikian pula sebaliknya.
 - b. Entity mana saja bisa menjadi tabel utama, sesuai dengan situasi/kebutuhan/analisis.
 - c. *Primary Key* (PK) dari tabel utama menjadi *Foreign Key* (FK) di tabel kedua.
 2. *One-to-Many* (atau *Many-to-One*) : menggunakan notasi 1:n
 - a. Jika A terhadap B mempunyai *one-to-many relationship* (atau B terhadap A mempunyai *many-to-one relationship*): satu *record* di A bisa berhubungan dengan banyak *record* di B, tetapi satu *record* di B berhubungan dengan hanya satu *record* di A.
 - b. *Entity* di sisi *One* dalam hubungan *One-to-Many* menjadi tabel utama, dan *entity* di sisi *Many* menjadi tabel kedua.
 - c. PK dari tabel utama menjadi FK di tabel kedua.
 - d. Di contoh sebelumnya, A menjadi tabel utama dan B menjadi tabel kedua. Maka, ada FK di B yang merupakan PK di A.
 3. *Many-to-Many* : menggunakan notasi m:n
 - a. Jika A terhadap B mempunyai *many-to-many relationship*: satu *record* di A bisa berhubungan dengan banyak *record* di B, dan demikian pula sebaliknya.

- b. Dalam implementasi database, harus ada sebuah tabel perantara di antara A dan B. A dan B menjadi tabel utama, dan tabel perantara menjadi tabel kedua.
 - c. PK dari A menjadi FK di tabel perantara (tabel kedua), dan PK dari B juga menjadi FK di tabel kedua. Gabungan semua FK di tabel kedua menjadi *Composite* PK untuk tabel kedua.
 - d. A terhadap tabel perantara mempunyai 1:n *relationship*.
 - e. B terhadap tabel perantara juga mempunyai 1:n *relationship*.
- Tahap Pembuatan Tabel
 1. Membuat semua tabel yang paling utama (yang tidak memiliki FK).
 2. Membuat semua tabel yang berhubungan langsung (atau memiliki *relationship*) dengan tabel yang dibuat di tahap sebelumnya. Mulailah secara urut dari tabel dengan jumlah FK yang paling sedikit ke yang paling banyak.
 3. Ulangi tahap 2 sampai semua tabel selesai dibuat.

- Implementasi Manual

Contoh untuk menentukan *Primary Key* (PK):

```
CREATE TABLE dosen (nip INTEGER PRIMARY KEY, nama_dosen
VARCHAR(45), alamat_dosen VARCHAR(255));
```

```
CREATE TABLE ruang (kode_ruang VARCHAR(20) PRIMARY KEY,
lokasi_ruang VARCHAR(255), kapasitas_ruang INTEGER);
```

Contoh untuk menentukan *Foreign Key* (FK):

```
CREATE TABLE mahasiswa(nim INTEGER PRIMARY KEY, nip INTEGER
REFERENCES dosen (nip), nama_mhs VARCHAR(45), alamat_mhs
VARCHAR(255))
```

Contoh untuk banyak FK:

```
CREATE TABLE mata_kuliah(kode_mk INTEGER PRIMARY KEY, nip
INTEGER REFERENCES dosen (nip), kode_ruang VARCHAR(20)
REFERENCES ruang (kode_ruang), nama_mk VARCHAR(45), deskripsi_mk
VARCHAR(255));
```

Contoh untuk menentukan *composite* PK:

```
CREATE TABLE mhs_ambil_mk(nim INTEGER REFERENCES mahasiswa
(nim), kode_mk INTEGER REFERENCES mata_kuliah(kode_mk), PRIMARY
KEY(nim, kode_mk));
```

- Referential Integrity
 1. Integritas database mengacu pada hubungan antar tabel melalui *Foreign Key* yang bersangkutan.
 2. Pada *insert*, *record* harus dimasukkan di tabel utama dahulu, kemudian baru di tabel kedua.
 3. Pada *delete*, *record* harus dihapus di tabel kedua dahulu, kemudian baru di tabel utama.
 4. Secara *default*, kebanyakan DBMS yang ada akan menolak *insert* atau *delete* yang melanggar integritas database.
- Advance Create Table Option
 1. Default

Untuk menentukan nilai default kolom jika tidak ada data yang di-insert untuk kolom itu:

```
CREATE TABLE mahasiswa(nim integer PRIMARY KEY, nama_mhs varchar(45), fakultas varchar(5) DEFAULT 'FKI');
```
 2. Not Null
 - Untuk membatasi agar nilai kolom tidak boleh NULL:


```
CREATE TABLE ruang(kode_ruang varchar(20) PRIMARY KEY, lokasi_ruang varchar(255) NOT NULL, kapasitas_ruang integer NOT NULL);
```
 - Jika kolom ditentukan NOT NULL, maka *insert* harus memasukkan nilai untuk kolom tersebut.
 - Bisa menggunakan DEFAULT sehingga nilai kolom ditambahkan secara otomatis.
 3. Unique
 - Untuk memastikan bahwa nilai kolom unik:


```
CREATE TABLE mata_kuliah(kode_mk integer PRIMARY KEY, nama_mk varchar(45) UNIQUE);
```
 - Untuk multikolom yang unik:


```
CREATE TABLE dosen(nip integer PRIMARY KEY, nama_dosen varchar(45), alamat_dosen varchar(255), UNIQUE (nama_dosen, alamat_dosen));
```
 4. Check
 - Untuk membatasi nilai kolom, misalnya:


```
CREATE TABLE produk(kode_produk integer PRIMARY KEY, nama_produk varchar(45), harga integer, CHECK (harga <= 100000 AND kode_produk > 100));
```

- *Check* di atas membatasi bahwa harga harus maksimal Rp 100000, dan kode_produk harus di atas 100.
5. Penentuan Referential Integrity
- Contoh:

```
CREATE TABLE pemasok(kode_pemasok integer PRIMARY KEY,
nama_pemasok varchar(45), kode_produk integer REFERENCES produk ON
DELETE CASCADE ON UPDATE CASCADE);
```
 - Untuk contoh di atas, jika ada *update* atau *delete* di tabel utama, maka tabel kedua secara otomatis disesuaikan.
6. Action
- NO ACTION atau RESTRICT: *update* atau *delete* tidak dilakukan. Ini merupakan pilihan *default*.
 - CASCADE: nilai kolom di tabel kedua disesuaikan dengan nilai kolom di tabel utama
 - SET NULL: nilai kolom di tabel kedua dijadikan NULL
 - SET DEFAULT: nilai kolom di tabel kedua dijadikan nilai DEFAULT (nilai DEFAULT harus ditentukan pada waktu pembuatan tabel).
7. Autoincrement
- Untuk fitur *autoincrement*, gunakan “serial”:

```
CREATE TABLE nasabah(id_nasabah serial PRIMARY KEY, nama_nasabah
varchar(45));
```
 - Untuk contoh di atas, id_nasabah tidak perlu di-*insert*, karena database secara otomatis akan menambahkannya secara urut.
 - “serial” hanya bisa dari 1 sampai 232.
 - Jika tidak cukup, gunakan “bigserial” yang bisa dari 1 sampai 264.
 - Jika kolom menggunakan “serial” atau “bigserial”, jangan sekali-kali memasukkan nilai kolom tersebut secara manual! Biarkan database menambahkannya sendiri secara urut!
 - Penghapusan *record* tidak akan mempengaruhi urutan untuk serial dan *bigserial*. Nilai untuk kolom yang menggunakan *serial/bigserial* akan selalu bertambah 1, tidak akan pernah kembali mundur.

C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows XP
2. Program aplikasi XAMPP dengan PhpMyAdmin
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah-langkah Praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buka Command Prompt dan login sebagai root ke MySQL seperti di langkah pada Modul 1.
4. Buat database baru dengan perintah berikut ini.
create database perbankan;
5. Hubungkan ke dalam database yang telah dibuat dengan perintah berikut. Sehingga akan muncul pemberitahuan “database changed”.
use perbankan;

6. Membuat tabel nasabah dengan script berikut.

```
CREATE TABLE nasabah (  
    id_nasabah INTEGER PRIMARY KEY,  
    nama_nasabah VARCHAR(45) NOT NULL,  
    alamat_nasabah VARCHAR(255) NOT NULL  
);
```

7. Membuat tabel cabang_bank dengan script berikut.

```
CREATE TABLE cabang_bank (  
    kode_cabang VARCHAR(20) PRIMARY KEY,  
    nama_cabang VARCHAR(45) UNIQUE NOT NULL,  
    alamat_cabang VARCHAR(255) NOT NULL  
);
```

8. Membuat tabel rekening dengan script berikut.

```
CREATE TABLE rekening (  
    no_rekening INTEGER PRIMARY KEY,  
    kode_cabangFK VARCHAR(20) REFERENCES cabang_bank(kode_cabang)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
    pin VARCHAR(20) DEFAULT '1234' NOT NULL,  
    saldo INTEGER DEFAULT 0 NOT NULL  
);
```

9. Membuat tabel transaksi dengan script berikut ini.

```
CREATE TABLE transaksi (  
    no_transaksi SERIAL PRIMARY KEY,  
    id_nasabahFK INTEGER REFERENCES nasabah(id_nasabah)  
    ON DELETE SET NULL ON UPDATE CASCADE,  
    no_rekeningFK INTEGER REFERENCES rekening(no_rekening)  
    ON DELETE SET NULL ON UPDATE CASCADE,
```

```

jenis_transaksi VARCHAR(20) DEFAULT 'debit' NOT NULL,
tanggal DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
jumlah INTEGER NOT NULL CHECK (jumlah>=20000)
);

```

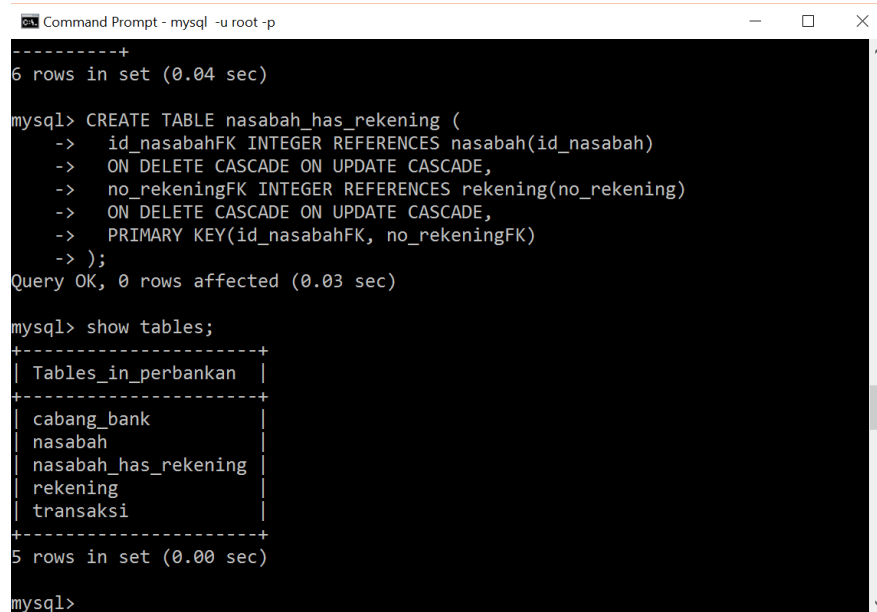
10. Membuat tabel nasabah_has_rekening dengan script berikut ini.

```

CREATE TABLE nasabah_has_rekening (
    id_nasabahFK INTEGER REFERENCES nasabah(id_nasabah)
    ON DELETE CASCADE ON UPDATE CASCADE,
    no_rekeningFK INTEGER REFERENCES rekening(no_rekening)
    ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY(id_nasabahFK, no_rekeningFK)
);

```

11. Untuk mengecek hasil pembuatan database gunakan perintah show tables;



```

Command Prompt - mysql -u root -p
-----+
6 rows in set (0.04 sec)

mysql> CREATE TABLE nasabah_has_rekening (
    -> id_nasabahFK INTEGER REFERENCES nasabah(id_nasabah)
    -> ON DELETE CASCADE ON UPDATE CASCADE,
    -> no_rekeningFK INTEGER REFERENCES rekening(no_rekening)
    -> ON DELETE CASCADE ON UPDATE CASCADE,
    -> PRIMARY KEY(id_nasabahFK, no_rekeningFK)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+-----+
| Tables_in_perbankan |
+-----+
| cabang_bank         |
| nasabah             |
| nasabah_has_rekening |
| rekening            |
| transaksi           |
+-----+
5 rows in set (0.00 sec)

mysql>

```

12. Kemudian untuk melihat struktur tiap tabel dapat dilakukan dengan perintah describe. Misalkan untuk melihat struktur tabel nasabah dapat dilakukan dengan perintah describe nasabah;

```
Command Prompt - mysql -u root -p
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+-----+
| Tables_in_perbankan |
+-----+
| cabang_bank         |
| nasabah              |
| nasabah_has_rekening |
| rekening            |
| transaksi            |
+-----+
5 rows in set (0.00 sec)

mysql> describe nasabah;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id_nasabah     | int(11)       | NO   | PRI | NULL    |       |
| nama_nasabah   | varchar(45)   | NO   |     | NULL    |       |
| alamat_nasabah | varchar(255)  | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)

mysql>
```

E. Tugas

Implementasikan hasil rancangan database yang menangani data kuliah pada tugas modul 2 ke dalam program mysql.

Modul 5

Data Manipulation Language (DML)

A. Tujuan

1. Mahasiswa mampu memahami dan mengisi tabel dalam database.
2. Mahasiswa mampu memahami dan memanipulasi data dalam database.

B. Landasan Teori

Data Manipulation Language (DML) adalah suatu statement yang dijalankan pada saat kita memerlukan :

- penambahan baris baru pada table.
- memodifikasi baris yang ada pada table.
- menghapus baris yang ada pada table.

DML Statement identik dengan operasi INSERT, MODIFY dan DELETE. Istilah Transaksi mengandung pengertian kumpulan Statement DML yang membentuk suatu fungsi tertentu.

1. Menambahkan baris baru ke dalam tabel (INSERT)

Menambahkan baris baru ke dalam table menggunakan perintah INSERT.

```
INSERT INTO table [(column [, column ...] ) ]
```

```
VALUES (value [, value...] );
```

Contoh :

```
INSERT INTO nasabah(id_nasabah, nama_nasabah, alamat_nasabah)
VALUES(1, 'Sutopo',' Jl. Jendral Sudirman 12');
```

2. Perubahan data dalam tabel (UPDATE).

Untuk memodifikasi baris data yang ada pada table digunakan perintah UPDATE. Sintak dari perintah UPDATE :

```
UPDATE table
```

```
SET column = value [, column = value, ...]
```

```
[WHERE condition];
```

Contoh :

```
UPDATE nasabah
```

```
SET alamat_nasabah = 'Jalan Jenderal Sudirman No.12'
```

```
WHERE nama_nasabah = 'Sutopo';
```

3. Menghapus data dalam tabel (DELETE).

Baris data yang ada pada table dapat dihapus dengan menggunakan perintah DELETE.

```
DELETE FROM table
```

```
[WHERE condition]
```

Contoh :

```
DELETE FROM nasabah
```

```
WHERE nama_nasabah = 'Sutopo';
```

4. Transaksi dalam database

Jika DML berkaitan dengan manipulasi data pada table, maka DDL (Data Definition Language) berkaitan dengan pendefinisian table, sedangkan DCL (Data Control Language) berkaitan kontrol transaksi. DDL dan DCL secara otomatis akan di-commit (dilakukan perubahan secara permanen) pada akhir dari transaksi. Ada 2 statement DCL yang penting yaitu COMMIT dan ROLLBACK, selain dari itu ada SAVEPOINT. Perintah COMMIT menandai perubahan secara permanen pada data. Sedangkan ROLLBACK mengembalikan keadaan sesuai dengan titik (keadaan) yang ditandai dengan SAVEPOINT, atau jika ROLLBACK tidak diberi parameter maka keadaan akan dikembalikan pada titik perubahan yang terakhir. Transaksi akan diproses secara implicit atau dilakukan operasi COMMIT secara otomatis, untuk keadaan berikut :

- Setelah Statement DDL diberikan.
- Setelah Statement DCL diberikan.
- Proses exit secara normal dari SQL*PLUS.

Sedangkan perintah ROLLBACK secara otomatis akan dijalankan jika terjadi kondisi yang abnormal atau terjadi system failure.

C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows 7
2. Program aplikasi XAMPP dengan PhpMyAdmin
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah-langkah praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buka Command Prompt dan login sebagai root ke MySQL seperti di langkah pada Modul 1.
4. Pilih database perbankan dengan perintah “use perbankan”. Sehingga akan muncul pemberitahuan “database changed”
5. Kemudian masukan data-data berikut ini ke dalam database perbankan.

Tabel : nasabah

id_nasabah	nama_nasabah	alamat_nasabah
1	Sutopo	Jl. Jendral Sudirman 12
2	Maryati	Jl. MT. Haryono 31
3	Suparman	Jl. Hasanudin 81
4	Kartika Padmasari	Jl. Manggis 15
5	Budi Eko Prayogo	Jl. Kantil 30
6	Satria Eka Jaya	Jl. Slamet Riyadi 45
7	Indri Hapsari	Jl. Sutoyo 5
8	Sari Murti	Jl. Pangandaran 11
9	Canka Lokananta	Jl. Tidar 86
10	Budi Murtono	Jl. Merak 22

Tabel : cabang_bank

kode_cabang	nama_cabang	alamat_cabang
BRUS	Bank Rut Unit Surakarta	Jl. Slamet Riyadi 18
BRUM	Bank Rut Unit Magelang	Jl. P. Tendean 63
BRUB	Bank Rut Unit Boyolali	Jl. Ahmad Yani 45
BRUK	Bank Rut Unit Klaten	Jl. Suparman 23
BRUY	Bank Rut Unit Yogyakarta	Jl. Anggrek 21
BRUW	Bank Rut Unit Wonogiri	Jl. Untung Suropati 12

Tabel : rekening

no_rekening	kode_cabang	pin	saldo
101	BRUS	1111	500000
102	BRUS	2222	350000
103	BRUS	3333	750000
104	BRUM	4444	900000
105	BRUM	5555	2000000
106	BRUS	6666	3000000

107	BRUS	7777	1000000
108	BRUB	0000	5000000
109	BRUB	9999	0
110	BRUY	1234	550000
111	BRUK	4321	150000
112	BRUK	0123	300000
113	BRUY	8888	255000

Tabel : nasabah_has_rekening

id_nasabah	no_rekening
1	104
2	103
3	105
3	106
4	101
4	107
5	102
5	107
6	109
7	109
8	111
9	110
10	113
8	112
10	108

Tabel : transaksi

no_trans aksi	no_rekening	id_nasabah	jenis_trans aksi	tanggal	jumlah
1	105	3	debit	2009-11-10	50000
2	103	2	debit	2009-11-10	40000
3	101	4	kredit	2009-11-12	20000
4	106	3	debit	2009-11-13	50000
5	107	5	kredit	2009-11-13	30000
6	104	1	kredit	2009-11-15	200000
7	110	9	kredit	2009-11-15	150000
8	102	5	debit	2009-11-16	20000
9	105	3	kredit	2009-11-18	50000
10	107	4	debit	2009-11-19	100000
11	103	2	debit	2009-11-19	100000
12	104	1	debit	2009-11-19	50000
13	107	4	kredit	2009-11-20	200000
14	105	3	debit	2009-11-21	40000
15	104	1	kredit	2009-11-22	100000
16	101	4	kredit	2009-11-22	20000
17	103	2	debit	2009-11-22	50000
18	102	5	debit	2009-11-25	50000
19	108	10	debit	2009-11-26	100000
20	106	3	kredit	2009-11-27	50000
21	103	2	kredit	2009-11-28	200000
22	105	3	kredit	2009-11-28	100000
23	102	5	debit	2009-11-30	20000
24	104	1	debit	2009-12-1	50000
25	103	2	debit	2009-12-2	40000
26	101	4	debit	2009-12-4	50000
27	103	2	kredit	2009-12-5	100000
28	102	5	kredit	2009-12-5	200000

29	109	7	debit	2009-12-5	100000
30	110	9	debit	2009-12-6	20000

6. Lakukan update untuk kasus-kasus berikut ini.
 - a. Nasabah dengan nama “Indri Hapsari” pindah alamat ke “Jalan Slamet Riyadi No.34”.
 - b. Cabang dengan kode “BRUW” pindah ke alamat “Jalan A. Yani No.23”.
7. Lakukan penghapusan untuk kasus-kasus berikut.
 - a. Nasabah dengan id “7” menutup rekeningnya.
 - b. Cabang dengan nama_cabang “Bank Rut Unit Magelang” menutup kantornya.

E. Tugas

Masukan beberapa *record* ke setiap tabel dalam database yang telah anda buat pada modul 4. Print out hasil implementasi rancangan tersebut dan analisa hasilnya. (Masing-masing 10 record untuk tabel master dan 20 record untuk tabel transaksi)

Modul 6

Query Standar dan Variasinya

A. Tujuan

1. Mahasiswa mampu menggunakan intruksi seleksi standard an beberapa variasi dasarnya.

B. Landasan Teori

SELECT STATEMENTS digunakan untuk menentukan atau memilih data yang akan ditampilkan ketika melakukan *query* terhadap basis data. Struktur *select statements* di postgresQL adalah sebagai berikut:

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]  
  
* | expression [ AS output_name ] [, ...]  
  
[ FROM from_item [, ...] ]  
  
[ WHERE condition ]  
  
[ GROUP BY expression [, ...] ]  
  
[ HAVING condition [, ...] ]  
  
[ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]  
  
[ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]  
  
[ FOR UPDATE [ OF tablename [, ...] ] ]  
  
[ LIMIT { count | ALL } ]  
  
[ OFFSET start ]
```

1. Contoh *SELECT* :

- Untuk melihat semua kolom dari suatu tabel:
`SELECT * FROM nasabah;`
- Untuk melihat kolom(-kolom) tertentu:
`SELECT nama_nasabah FROM nasabah;`
`SELECT id_nasabah, nama_nasabah FROM nasabah;`
- Secara umum:
`SELECT <nama kolom, ...> FROM <nama tabel>;`

2. Columns Alias (AS)

AS digunakan untuk mengganti nama kolom pada tampilan SELECT.

Contoh:

```
SELECT nama_nasabah AS "Nama Nasabah" FROM nasabah;
```

```
SELECT nama_nasabah AS "Nasabah", alamat_nasabah AS "Alamat Nasabah" FROM nasabah;
```

3. Where

Digunakan untuk membatasi hasil SELECT yang ditampilkan berdasarkan kondisi yang ditentukan.

Contoh:

```
SELECT nama_nasabah FROM nasabah WHERE nama_nasabah='Ali Topan';
```

```
SELECT nama_nasabah, alamat_nasabah FROM nasabah WHERE id_nasabah=2;
```

- Bisa menggunakan >, <, <> (atau !=), >=, <=

Gunakan AND atau OR untuk lebih dari satu kondisi:

```
SELECT*FROM nasabah WHERE nama_nasabah = 'Rina Marsudi' AND alamat_nasabah='Jl. Kusumanegara 30';
```

```
SELECT*FROM nasabah WHERE nama_nasabah='Ali Topan' OR id_nasabah = 2;
```

- Pencarian NULL

Gunakan IS NULL untuk mencari NULL:

```
SELECT*FROM rekening WHERE kode_cabang IS NULL;
```

Gunakan IS NOT NULL untuk mencari yang tidak NULL:

```
SELECT*FROM rekening WHERE kode_cabang IS NOT NULL;
```

- Pencarian String

Gunakan LIKE untuk mencari string tertentu:

```
SELECT*FROM nasabah WHERE nama_nasabah LIKE 'Ali Topan';
```

Bisa menggunakan %:

```
SELECT*FROM nasabah WHERE alamat_nasabah LIKE '%negara%';
```

Bisa menggunakan _ untuk 1 huruf:

```
SELECT*FROM nasabah WHERE nama_nasabah LIKE 'Ali T_p_n';
```

Untuk pencarian yang *case insensitive* (tidak mempedulikan huruf besar atau kecil), gunakan ILIKE:

```
SELECT*FROM nasabah WHERE nama_nasabah ILIKE '% marsudi';
```

4. Order by

Digunakan untuk mengurutkan hasil SELECT. Jenis-jenisnya adalah sebagai berikut ini :

Untuk mengurutkan dari kecil ke besar:

```
SELECT*FROM nasabah ORDER BY nama_nasabah;
```

Untuk mengurutkan dari besar ke kecil:

```
SELECT*FROM nasabah ORDER BY nama_nasabah DESC;
```

Untuk melakukan pengurutan lebih dari satu kolom, pisahkan dengan tanda koma:

```
SELECT*FROM nasabah_has_rekening ORDER BY no_rekening,  
id_nasabah;
```

Anda bisa menentukan DESC untuk kolom(-kolom) tertentu, misalnya:

```
SELECT*FROM nasabah_has_rekening ORDER BY no_rekening, id_nasabah  
DESC;
```

```
SELECT*FROM nasabah_has_rekening ORDER BY no_rekening DESC,  
id_nasabah;
```

5. Limit dan Offset

- Digunakan untuk membatasi jumlah baris yang ditampilkan dalam SELECT.

Contoh:

Hanya menampilkan 3 baris pertama:

```
SELECT*FROM nasabah ORDER BY id_nasabah LIMIT 3;
```

Menampilkan 2 baris setelah melewati 2 baris pertama:

```
SELECT*FROM nasabah ORDER BY id_nasabah LIMIT 2 OFFSET 2;
```

- Perhatian: penggunaan LIMIT sebaiknya selalu digunakan bersama dengan ORDER BY, sehingga urutan yang ditampilkan akan selalu konsisten.
- LIMIT dan OFFSET sangat berguna dalam tampilan yang berbasis web (melalui web browser dengan menggunakan PHP atau JSP) agar tampilan data tidak terlalu besar dan bisa lebih rapi. Tampilan data yang banyak bisa diatur dan dibagi menjadi beberapa halaman (pages).

6. Tabel Join

- Cross Join
 - ✓ Menggabungkan semua record dari tabel pertama dengan semua record di tabel kedua.
 - ✓ Banyaknya record dari cross join = jumlah record tabel pertama X jumlah record tabel kedua

Contoh:

```
SELECT*FROM rekening CROSS JOIN cabang_bank;
```

- Inner Join

- ✓ Menghubungkan 2 (atau lebih) tabel berdasarkan attribute penghubung.
- ✓ Metode 1:

```
SELECT*FROM rekening INNER JOIN cabang_bank
USING(kode_cabang);
```
- ✓ Metode 2:

```
SELECT*FROM rekening INNER JOIN cabang_bank ON
rekening.kode_cabang = cabang_bank.kode_cabang;
```
- ✓ Metode 3:

```
SELECT * FROM rekening NATURAL INNER JOIN cabang_bank;
```
- ✓ Metode 4:

```
SELECT * FROM rekening, cabang_bank WHERE
rekening.kode_cabang= cabang_bank.kode_cabang;
```
- ✓ Perhatian: Untuk INNER JOIN, Anda dapat menghilangkan kata 'INNER'. Jadi, cukup dengan kata 'JOIN' saja.
- ✓ Dengan metode 4, jika kolom yang ingin ditampilkan ada di lebih dari 2 tabel, maka Anda harus menentukan tabel mana yang diinginkan.

Contoh:

```
SELECT nasabah.id_nasabah, nama_nasabah, no_rekening FROM
nasabah, nasabah_has_rekening WHERE nasabah.id_nasabah=
nasabah_has_rekening.id_nasabah;
```

- Tabel Alias

- ✓ Untuk kemudahan penulisan SQL, kita bisa membuat table alias.
- ✓ Contoh:

```
SELECT*FROM nasabah A, nasabah_has_rekening B WHERE
A.id_nasabah=B.id_nasabah;
```

```
SELECT A.id_nasabah, nama_nasabah, no_rekening FROM nasabah
A, nasabah_has_rekening B WHERE A.id_nasabah=B.id_nasabah;
```

- Distinct

- ✓ Dalam table join, kadang-kadang ada informasi yang berulang. Untuk menghilangkan pengulangan tersebut, gunakan DISTINCT.

Contoh:

```
SELECT DISTINCT nama_nasabah, alamat_nasabah FROM nasabah
NATURAL JOIN nasabah_has_rekening;
```


- ✓ Perhatikan perbedaan dengan berikut:

```
SELECT nama_nasabah, alamat_nasabah FROM nasabah NATURAL
JOIN nasabah_has_rekening;
```
- Right Outer Join
 - ✓ Menampilkan hasil join tabel pertama (sisi kiri) dengan tabel kedua (sisi kanan), serta semua *record* di tabel kedua (sisi kanan/*right*):

```
SELECT*FROM rekening NATURAL RIGHT OUTER JOIN
cabang_bank;
```
 - ✓ Ketiga metode pertama yang telah disebutkan untuk INNER JOIN juga berlaku untuk RIGHT OUTER JOIN, yaitu dengan menggunakan USING, ON, atau NATURAL.
- Left Outer Join
 - ✓ Menampilkan hasil join tabel pertama (sisi kiri) dengan tabel kedua (sisi kanan), serta semua *record* di tabel pertama (sisi kiri/*left*):

```
SELECT*FROM rekening NATURAL LEFT OUTER JOIN cabang_bank;
```
 - ✓ Ketiga metode yang telah disebutkan untuk RIGHT OUTER JOIN juga berlaku untuk LEFT OUTER JOIN, yaitu dengan menggunakan USING, ON, atau NATURAL.
- Full Outer Join
 - ✓ Menampilkan hasil join tabel pertama dengan tabel kedua, serta semua record di kedua tabel tersebut:

```
SELECT*FROM rekening NATURAL FULL OUTER JOIN cabang_bank;
```
 - ✓ Ketiga metode yang telah disebutkan untuk LEFT/RIGHT OUTER JOIN juga berlaku untuk FULL OUTER JOIN, yaitu dengan menggunakan USING, ON, atau NATURAL.
- Inner vs Outer Join
 - ✓ Dalam *Inner Join*: yang ditampilkan hanyalah hasil dari *table join* yang berhasil, yaitu semua *record* yang berhubungan di kedua tabel yang digabungkan.
 - ✓ Dalam *Outer Join*: selain menampilkan hasil dari *Inner Join*, *Outer Join* juga menampilkan semua *record* yang tidak berhubungan di kedua tabel yang digabungkan.
- Multiple Join
 - ✓ Untuk lebih dari 2 tabel, tinggal diteruskan saja JOINnya. Misalnya:

```
SELECT*FROM nasabah NATURAL JOIN nasabah_has_rekening
NATURAL JOIN rekening;
```

Cara lain:

```
SELECT*FROM nasabah A, nasabah_has_rekening B, rekening C
WHERE          A.id_nasabah=B.id_nasabah          AND
B.no_rekening=C.no_rekening;
```

- ✓ Jika melakukan *multiple join* (lebih dari 2 tabel), Anda harus memperhatikan urutan *join*. Urutan *table join* perlu mengikuti alur *relationship* yang tertera di ER Diagram.
- ✓ Oleh karena itu, sebaiknya Anda menggunakan ER Diagram agar bisa menghasilkan *table join* yang benar.

C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows 7
2. Program aplikasi XAMPP dengan PhpMyAdmin
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah-langkah Praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buka Command Prompt dan login sebagai root ke MySQL seperti di langkah pada Modul 1.
4. Pilih database perbankan dengan perintah “use perbankan”. Sehingga akan muncul pemberitahuan “database changed”
5. Tampilkan nama bank dan alamat bank untuk semua cabang bank dan diurutkan berdasarkan nama bank dengan kode berikut :

```
SELECT nama_cabang, alamat_cabang FROM cabang_bank ORDER BY
nama_cabang;
```

6. Tampilkan nomor rekening, pin, dan jumlah saldo untuk semua rekening dan diurutkan berdasarkan jumlah saldo dari yang paling besar ke yang paling kecil dengan kode berikut
SELECT no_rekening, pin, saldo FROM rekening ORDER BY saldo DESC;
7. Tampilkan nomor rekening, nama nasabah, dan alamat nasabah dari semua nasabah yang memiliki rekening dan diurutkan berdasarkan nama nasabah dengan kode berikut.

```
SELECT          rekening.no_rekening,          nasabah.nama_nasabah,
nasabah.alamat_nasabah FROM rekening, nasabah, nasabah_has_rekening
WHERE nasabah.id_nasabah= nasabah_has_rekening.id_nasabahFK AND
rekening.no_rekening=nasabah_has_rekening.no_rekeningFK ORDER BY
nasabah.nama_nasabah;
```

8. Tampilkan nomor rekening, nama nasabah, dan jumlah saldo untuk semua rekening yang dimiliki oleh nasabah dan diurutkan berdasarkan nama nasabah dengan kode berikut :

```
SELECT rekening.no_rekening, nasabah.nama_nasabah, rekening.saldo
FROM rekening, nasabah, nasabah_has_rekening WHERE
```

nasabah.id_nasabah= nasabah_has_rekening.id_nasabahFK AND
rekening.no_rekening= nasabah_has_rekening.no_rekeningFK ORDER BY
nasabah.nama_nasabah;

E. Tugas

1. Tampilkan nama nasabah, alamat nasabah, jenis transaksi dan jumlah transaksi dimana jenis transaksinya adalah kredit dan diurutkan berdasarkan nama nasabah!
2. Tampilkan nomor rekening, nama nasabah, jenis transaksi dan jumlah transaksi yang melakukan transaksi pada tanggal 21 November 2009 dan diurutkan berdasarkan nama nasabah!
3. Tampilkan nomor rekening, nama nasabah, jenis transaksi dan jumlah transaksi dimana jumlah transaksi = Rp 20.000!
4. Tampilkan nomor rekening, nama nasabah dan alamat nasabah dimana nama nasabah diawali dengan kata 'Su'!
5. Tampilkan nomor rekening dengan alias 'Nomor Rekening', nama nasabah dengan alias 'Nama Nasabah', jumlah transaksi dengan alias 'Jumlah Transaksi' dimana jenis transaksinya adalah debit! Urutkan berdasarkan nama nasabah!

Modul 7

Aggregasi

A. Tujuan

1. Mahasiswa mampu menggunakan beberapa fungsi agregasi dalam melakukan seleksi data.

B. Landasan Teori

1. IN

- Contoh:
`SELECT*FROM rekening WHERE kode_cabang IN ('BRUM', 'BRUL');`
- Perintah SQL di atas sama dengan:
`SELECT*FROM rekening WHERE kode_cabang = 'BRUM' OR
kode_cabang = 'BRUL';`
- Tidak ada batas banyaknya nilai yang bisa ada di dalam IN (...).

2. NOT IN

- Contoh:
`SELECT*FROM rekening WHERE kode_cabang NOT IN ('BRUS', 'BRUM');`
- Perintah SQL di atas sama dengan:
`SELECT*FROM rekening WHERE kode_cabang <> 'BRUS' AND
kode_cabang <> 'BRUM';`
- Nilai NULL tidak akan tampil dalam IN dan NOT IN.
- Perhatikan perbedaan penggunaan OR dan AND dalam IN dan NOT IN.

3. BETWEEN

- Contoh:
`SELECT*FROM rekening WHERE saldo BETWEEN 500000 AND
1000000;`
- Perintah SQL di atas sama dengan:
`SELECT*FROM rekening WHERE saldo>=500000 AND saldo<= 1000000;`
- Nilai yang pertama dalam BETWEEN harus lebih kecil dari nilai yang kedua.
- Bisa untuk *string*.

4. NOT BETWEEN

- Contoh:
`SELECT*FROM rekening WHERE saldo NOT BETWEEN 500000 AND
1000000;`
- Perintah SQL di atas sama dengan:
`SELECT*FROM rekening WHERE saldo<500000 OR saldo>1000000;`

- Perhatikan perbedaan penggunaan AND dan OR dalam BETWEEN dan NOT BETWEEN.

5. AGGREGATE FUNCTIONS

Fungsi-fungsi untuk *aggregate*:

- **MIN()**
 - Digunakan untuk mencari nilai terkecil dari sekumpulan *record*.
Contoh:
SELECT MIN(saldo) FROM rekening;
 - Bisa dibatasi dengan WHERE clause sehingga hanya *record(-record)* tertentu yang ditelusuri:
SELECT MIN(saldo) FROM rekening WHERE kode_cabang='BRUS';
- **MAX()**
 - Digunakan untuk mencari nilai terbesar dari sekumpulan *record*.
Contoh:
SELECT MAX(saldo) FROM rekening;
 - Juga bisa dibatasi dengan WHERE clause:
SELECT MAX(saldo) FROM rekening WHERE kode_cabang='BRUS';
- **COUNT()**
 - Digunakan untuk menghitung banyaknya *record*.
Contoh:
SELECT COUNT(*) FROM nasabah;
SELECT COUNT(nama_nasabah) FROM nasabah;
SELECT COUNT(alamat_nasabah) FROM nasabah;
 - Juga bisa dibatasi dengan WHERE clause.
 - Jika kita ingin menghitung banyaknya *record* yang unik (tidak ada pengulangan), gunakan DISTINCT:
SELECT COUNT(DISTINCT alamat_nasabah) FROM nasabah;
- **SUM()**
 - Digunakan untuk menjumlahkan nilai-nilai dari sekumpulan *record*.
Contoh:
SELECT SUM(saldo) FROM rekening;
 - Bisa dibatasi dengan WHERE clause:
SELECT SUM(saldo) FROM rekening WHERE kode_cabang='BRUS';
- **AVG()**
 - Digunakan untuk menghitung rata-rata nilai dari sekumpulan *record*.
Contoh:

```
SELECT AVG(saldo) FROM rekening;
```

- Bisa dibatasi dengan *WHERE clause*:

```
SELECT AVG(saldo) FROM rekening WHERE kode_cabang='BRUS';
```

- Beberapa *aggregate functions* bisa digabungkan dalam satu perintah SQL:

```
SELECT MIN(saldo), MAX(saldo), AVG(saldo) FROM rekening;
```

- Bisa menambahkan ekspresi aritmatika:

```
SELECT SUM(saldo + 1000) FROM rekening;
```

```
SELECT SUM(saldo) + 1000 FROM rekening;
```

```
SELECT MAX(saldo) - MIN(saldo) FROM rekening;
```

- Bisa menggunakan *Column Alias (AS)* untuk membuat tampilan lebih profesional.

- **GROUP BY**

- Digunakan untuk mengelompokkan sekumpulan *record* berdasarkan (kolom-kolom) tertentu.

Contoh:

```
SELECT jenis_transaksi FROM transaksi GROUP BY jenis_transaksi;
```

```
SELECT jenis_transaksi, tanggal FROM transaksi GROUP BY  
jenis_transaksi, tanggal;
```

- Hasil yang sama bisa didapatkan dengan menggunakan **DISTINCT**:

```
SELECT DISTINCT jenis_transaksi, tanggal FROM transaksi;
```

- Jika menggunakan **GROUP BY**, semua *field* yang ingin ditampilkan dalam **SELECT** harus tercantum di **GROUP BY**.

Contoh yang salah:

```
SELECT jenis_transaksi, tanggal FROM transaksi GROUP BY  
jenis_transaksi;
```

```
SELECT jenis_transaksi, tanggal FROM transaksi GROUP BY tanggal;
```

Contoh yang benar:

```
SELECT jenis_transaksi, tanggal FROM transaksi GROUP BY  
jenis_transaksi, tanggal;
```

- **HAVING**

- Merupakan pasangan dari **GROUP BY**, digunakan untuk membatasi kelompok yang ditampilkan:

```
SELECT jenis_transaksi, tanggal FROM transaksi GROUP BY  
jenis_transaksi, tanggal HAVING jenis_transaksi='kredit';
```

- Hasil yang sama bisa didapatkan dengan:

```
SELECT jenis_transaksi, tanggal FROM transaksi WHERE
jenis_transaksi='kredit' GROUP BY jenis_transaksi, tanggal;
```

- Jika menggunakan HAVING, maka pembatasan dilakukan setelah hasil dikelompokkan dalam GROUP BY.
- Jika menggunakan WHERE, maka pembatasan dilakukan sebelum hasil dikelompokkan dalam GROUP BY.
- *Field (-field)* yang disebut di HAVING harus ada di GROUP BY, atau berupa *aggregate functions*.

Contoh yang salah:

```
SELECT jenis_transaksi, tanggal FROM transaksi GROUP BY
jenis_transaksi, tanggal HAVING jumlah=50000;
```

Contoh yang benar:

```
SELECT jenis_transaksi, tanggal FROM transaksi WHERE
jumlah=50000 GROUP BY jenis_transaksi, tanggal;
```

- **GROUP BY dan AGGREGATE**

- GROUP BY sangat cocok untuk *aggregate functions*. Dengan menggunakan GROUP BY, kita bisa mengelompokkan record-record dan menghitung *min*, *max*, *count*, *sum*, dan *avg* untuk masing-masing kelompok.

Contoh:

```
SELECT kode_cabang, MIN(saldo), MAX(saldo), COUNT(*),
SUM(saldo), AVG(saldo) FROM rekening GROUP BY kode_cabang;
```

- Bisa digabungkan dengan tabel join dan ORDER BY:

```
SELECT nama_cabang, SUM(saldo) FROM rekening NATURAL JOIN
cabang_bank GROUP BY nama_cabang ORDER BY nama_cabang;
```
- Hasil di atas menampilkan total saldo untuk masing-masing cabang_bank.
- Perintah SQL di bawah menampilkan banyaknya nasabah yang dilayani oleh masing-masing cabang bank:

```
SELECT nama_cabang, COUNT(DISTINCT id_nasabah) FROM
cabang_bank NATURAL JOIN rekening NATURAL JOIN
nasabah_has_rekening GROUP BY nama_cabang ORDER BY
nama_cabang;
```

- Contoh dengan HAVING:

```
SELECT kode_cabang, SUM(saldo), COUNT(*) FROM rekening
GROUP BY kode_cabang HAVING SUM(saldo) >= 5000000 ORDER
BY kode_cabang;
```

Karena SUM(saldo) hanya bisa dihitung setelah hasil dikelompokkan dengan GROUP BY, maka kita harus menggunakan HAVING untuk membatasi hasil berdasarkan SUM(saldo) >= 5000000. Kita tidak bisa menggunakan WHERE.

C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows 7
2. Program aplikasi XAMPP dengan PhpMyAdmin
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah-langkah Praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buka Command Prompt dan login sebagai root ke MySQL seperti di langkah pada Modul 1.
4. Pilih database perbankan dengan perintah “use perbankan”. Sehingga akan muncul pemberitahuan “database changed”
5. Tampilkan tanggal transaksi, jenis transaksi, dan jumlah transaksi untuk semua transaksi yang dilakukan oleh Sutopo dan Canka Lokananta dan diurutkan berdasarkan tanggal transaksi dengan kode berikut :

```
SELECT transaksi.tanggal, transaksi.jenis_transaksi, transaksi.jumlah
FROM nasabah, transaksi WHERE
nasabah.id_nasabah=transaksi.id_nasabahFK AND nasabah.nama_nasabah
IN ('Sutopo','Canka Lokananta') ORDER BY transaksi.tanggal;
```

6. Tampilkan tanggal transaksi, nama nasabah, jenis transaksi, dan jumlah transaksi untuk semua transaksi yang terjadi dari 15 November sampai 20 November 2009 dan diurutkan berdasarkan tanggal transaksi dan nama nasabah dengan kode berikut :

```
SELECT transaksi.tanggal, nasabah.nama_nasabah,
transaksi.jenis_transaksi, transaksi.jumlah FROM nasabah, transaksi
WHERE transaksi.tanggal BETWEEN '2009-11-15' AND '2009-11-20' AND
nasabah.id_nasabah=transaksi.id_nasabahFK ORDER BY
transaksi.tanggal, nasabah.nama_nasabah;
```

7. Tampilkan jenis transaksi dan total jumlah transaksi (dalam rupiah) untuk tiap jenis transaksi dan diurutkan berdasarkan jenis transaksi dengan kode berikut :

```
SELECT transaksi.jenis_transaksi AS "Jenis Transaksi", SUM(jumlah) AS
"Jumlah (Rp)" FROM transaksi GROUP BY transaksi.jenis_transaksi ORDER
BY transaksi.jenis_transaksi;
```

8. Tampilkan jenis transaksi, jumlah transaksi yang terbesar serta yang terkecil untuk tiap jenis transaksi dan diurutkan berdasarkan jenis transaksi dengan kode berikut :

SELECT jenis_transaksi AS "Jenis Transaksi", MAX(jumlah) AS "Transaksi Terbesar", MIN(jumlah) AS "Transaksi Terkecil" FROM transaksi GROUP BY transaksi.jenis_transaksi ORDER BY transaksi.jenis_transaksi;

9. Tampilkan jenis transaksi, total jumlah transaksi (dalam rupiah), dan banyaknya transaksi yang tercatat untuk tiap jenis transaksi yang terjadi sebelum bulan Desember 2009 dan diurutkan berdasarkan jenis transaksi dengan kode berikut :

SELECT jenis_transaksi AS "Jenis Transaksi", SUM(jumlah) AS "Jumlah (Rp)", COUNT(jumlah) AS "Jumlah Transaksi" FROM transaksi WHERE tanggal BETWEEN '2009-11-1' AND '2009-11-30' GROUP BY transaksi.jenis_transaksi ORDER BY transaksi.jenis_transaksi;

E. Tugas

1. Tampilkan jenis transaksi, jumlah transaksi dalam Rp dan total transaksi untuk nasabah yang bernama akhiran 'Kartika Padmasari' untuk masing-masing jenis transaksi!
2. Berapa jumlah total saldo yang dimiliki oleh Maryati?
3. Tampilkan jumlah transaksi yang ditangani oleh masing-masing cabang bank!
4. Tampilkan nama nasabah dan jumlah saldo yang memiliki saldo antara Rp 500.000 sampai Rp 2.000.000!
5. Tampilkan nama nasabah, tanggal transaksi dan jumlah transaksi dalam Rp dimana jumlah transaksi di atas Rp 100.000 dan urutkan berdasarkan jumlah transaksi dari yang besar ke yang kecil!

Modul 8

Data Control Language

A. Tujuan

1. Mahasiswa memahami perintah-perintah Data Control Language.
2. Mahasiswa mampu mengimplementasikan perintah GRANT dan REVOKE.

B. Landasan Teori

Data Control Language (DCL) merupakan perintah-perintah yang dapat digunakan untuk menjaga keamanan basis data. Perintah tersebut dapat dipakai untuk menentukan akses basis data hanya dapat dilakukan oleh orang-orang tertentu dan dengan macam akses yang dibatasi pula. Adapun Objek-Objek DCL dalam Mysql diantaranya :

1. Tabel User dari Database MySQL

Tabel user adalah tabel yang ada dalam database MySQL. Tabel user hanya diperuntukkan bagi seorang Administrator (root). Tabel user bersifat global, apapun perubahan yang terjadi pada tabel ini akan mempengaruhi jalannya keseluruhan system MySQL. Tabel ini yang nantinya digunakan untuk mendefinisikan user dari basis data. Adapun isi dari tabel tersebut adalah sebagai berikut.

Field	Type	Null	Key	Default	Extra
Host	Varchar(60) binary		Pri		
User	Varchar(16) binary		Pri		
Password	Varchar(16)				
Select_priv	Enum('N', 'Y')			N	
Insert_priv	Enum('N', 'Y')			N	
Update_priv	Enum('N', 'Y')			N	
Delete_priv	Enum('N', 'Y')			N	
Drop_priv	Enum('N', 'Y')			N	
Reload_priv	Enum('N', 'Y')			N	
Shutdown_priv	Enum('N', 'Y')			N	
Process_priv	Enum('N', 'Y')			N	
File_priv	Enum('N', 'Y')			N	
Grant_priv	Enum('N', 'Y')			N	
References_priv	Enum('N', 'Y')			N	
Index_priv	Enum('N', 'Y')			N	
Alter_priv	Enum('N', 'Y')			N	
Show_db_priv	Enum('N', 'Y')			N	
Create_priv	Enum('N', 'Y')			N	

Sintaks Umum :

SELECT, UPDATE, DELETE, INSERT, FLUSH

Contoh Penggunaan :

```
SELECT host, user, password FROM user;
```

```
DELETE FROM user WHERE user='';
```

```
UPDATE user SET password=password('xxxxxxxxxx') WHERE user='root';
```

```
insert into user (User,Password) values('Winda','w1nd');
```

```
FLUSH PRIVILEGES;
```

Perintah FLUSH PRIVILEGES adalah suatu perintah untuk mengaktifkan perubahan-perubahan yang terjadi pada user, seperti hak akses, penggantian password pada user, dsb. Perintah FLUSH PRIVILEGES ini hukumnya wajib dilaksanakan setelah Anda melakukan perubahan (apapun juga) secara langsung ke dalam tabel user atau ke dalam database mysql.

2. Tabel Table_Priv dari database MySQL

Tabel_priv berfungsi mengatur tabel apa saja yang dapat diakses oleh seorang user, berikut jenis izin aksesnya. Tingkat akses hanya untuk tabel. Pada prinsipnya hanya bekerja seperti db table, kalau tidak digunakan untuk tabel sebagai ganti database.

Sintaks Umum :

SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT, REFERENCES,
INDEX, ALTER, CREATE VIEW, SHOW VIEW

Contoh :

```
Insert into tables_priv (User, table_name, table_priv)          values ('Febe,  
Elfrida', 'Point_Of_Sales.jenis', 'select');
```

3. GRANT

Grant berfungsi untuk membuat user baru dan memberikan hak istimewa. Grant adalah salah satu privilege untuk tabel. Grant digunakan untuk memberikan privilege kepada tabel yang didefinisikan kepada pemakai lain. Privilege untuk pemakai dalam perintah grant didefinisikan dengan menggunakan nama-nama privilege. Nama privilege memudahkan administrator untuk dapat memberikan privilege tanpa harus tahu apa nama field dan tabel yang harus diisi. Perintah grant secara otomatis akan menambah data pemakai apabila data nama pemakai yang disertakan pada perintah tersebut belum ada dalam tabel user. Perintah grant memudahkan administrator untuk tidak perlu melakukan perintah pendefinisian privilege dengan menggunakan sql. Karena dengan menggunakan sql, kita harus hafal nama tabel yang harus diisi, field apa saja yang harus diisi, jumlah field yang harus diisi. Kesalahan mudah dilakukan dengan menggunakan perintah sql karena 38 ketidaktelitian atau ketidakhafalan nama tabel dan nama field yang harus diisi.

Sintaks umum :

GRANT hak_akses ON nama_tabel TO pemakai;

Contoh :

GRANT SELECT ON Point_Of_Sales.jenis TO Febe;

GRANT ALL PRIVILEGES ON Point_Of_Sales.User TO Admin;

SHOW GRANTS FOR Admin;

4. REVOKE

Untuk menghapus batasan hak akses yang telah diatur dengan menggunakan perintah GRANT, digunakan perintah REVOKE.

Sintaks Umum :

REVOKE hak_akses ON nama_tabel FROM namaAccount@namaHost;

REVOKE hak_akses ON nama_database.nama_tabel FROM user; \

Contoh Penggunaan :

REVOKE SELECT ON Point_Of_Sales.jenis FROM Febe;

REVOKE ALL PRIVILEGES ON Point_Of_Sales.user FROM Admin;

5. Show Processlist

Digunakan untuk menampilkan kegiatan apa saja yang terjadi pada MySQL server atau menampilkan informasi mengenai thread yang dieksekusi di server. Bila terdapat kegiatan yang membahayakan kita sebagai admin dapat menghentikan dengan perintah KILL atau MySQLAdmin.

Sintaks Umum :

SHOW PROCESSLIST;

6. Kill

Kill berfungsi menghentikan thread server / untuk membunuh proses yang sedang berjalan

Sintaks Umum :

KILL nomor_Id;

Contoh Penggunaan :

KILL 2;

7. Optimize Table

Tabel yang sering mengalami proses penghapusan dan penambahan akan menyebabkan struktur yang tidak teratur secara fisik atau telah terjadi fragmentasi. Penghapusan data dalam jumlah besar mempunyai peluang terjadinya fragmentasi. Terutama untuk data bertipe VARCHAR, TEXT atau BLOB. Tidak semua DBMS dapat melakukan fragmentasi, kita dapat melihat dukungan setiap DBMS Untuk mengatasi masalah fragmentasi solusinya adalah melakukan OPTIMIZE TABLE Untuk MySQL versi 3.23 ke atas mendukung fasilitas OPTIMIZE TABLE. Perlu diketahui pada saat OPTIMIZE TABLE dikerjakan, semua tabel

akan di-lock (terkunci) Proses fragmentasi sebaiknya dilakukan secara berkala, misalnya setiap minggu atau setiap bulan.

Sintaks Umum :

```
OPTIMIZE TABLE tabel_1, tabel_2, tabel_3, tabel_n;
```

Contoh Penggunaan :

```
OPTIMIZE TABLE Point_Of_Sales.User, Point_Of_Sales.item,  
Point_Of_Sales.jualGlobal;
```

C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows 7
2. Program aplikasi XAMPP dengan PhpMyAdmin
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah-langkah Praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buka Command Prompt dan login sebagai root ke MySQL seperti di langkah pada Modul 1.
4. Pilih database perbankan dengan perintah “use mysql”. Sehingga akan muncul pemberitahuan “database changed”
5. Masukkan user baru ke tabel user dengan sintaks berikut ini.

```
insert into user (user,password) values ('myuser','myuser');
```
6. Berikan hak akses kepada user “myuser” dengan sintaks berikut ini.

```
GRANT SELECT on perbankan.nasabah to myuser;  
FLUSH PRIVILEGES;  
GRANT SELECT on perbankan.cabang_bank to myuser;  
FLUSH PRIVILEGES;
```
7. Keluar sebagai user root dengan syntax “exit;”.
8. Login lagi sebagai “myuser” dengan sintaks sebagai berikut.

```
mysql -u myuser
```
9. Ubah database yang digunakan ke database perbankan dengan sintaks “use perbankan”. Sehingga akan muncul konfirmasi “database changed”.
10. Coba lakukan select pada tabel nasabah dengan sintaks berikut.

```
select * from nasabah;
```

Selanjutnya coba lakukan insert pada tabel nasabah dengan sintaks berikut.

```
insert into nasabah values('31','Susilo','Sukoharjo');
```
11. Kemudian ketikkan perintah “exit”. Kemudian login lagi sebagai root.
12. Kemudian hilangkan salah satu privilege untuk user myuser dengan sintaks berikut ini.

```
REVOKE SELECT on perbankan.nasabah from myuser;
```

FLUSH PRIVILEGE;

13. Kemudian exit dan kembali login sebagai user “myuser”.
14. Gunakan database perbankan dengan sintaks “use perbankan”.
15. Selanjutnya silahkan dicoba untuk menggunakan perintah SELECT pada tabel nasabah.

E. Tugas

1. Buat user baru sesuai dengan nama anda masing-masing.
2. Beri Privilege untuk user tersebut dengan ketentuan sebagai berikut ini.
 - a. Dapat melakukan INSERT, UPDATE, dan DELETE pada tabel nasabah.
 - b. Hanya dapat melakukan SELECT pada tabel cabang_bank.
3. Coba lakukan kasus berikut ini dengan login sebagai user yang dibuat dan screenshot hasilnya.
 - a. Lakukan perintah SELECT pada tabel nasabah.
 - b. Lakukan perintah INSERT pada tabel nasabah.
 - c. Lakukan perintah INSERT pada tabel cabang_bank.
4. Analisa hasil percobaan kalian tersebut.

Modul 9

SubQuery

A. Tujuan

1. Mahasiswa mampu mengimplementasikan SubQuery untuk menyelesaikan kasus pengambilan data dari suatu basis data.

B. Landasan Teori

Dalam aplikasi yang menggunakan database tidak akan pernah terlepas dalam perolehan kembali data. Untuk memunculkan data yang dikemas lebih baik agar memudahkan pengguna data untuk mengerti dan memahami data tentunya tidak akan terlepas dari query-query sql yang lebih kompleks apalagi data antara table satu dengan table lainnya saling berhubungan.

Sangat panjang jika kita menulis subquery dalam satu baris(baris tunggal) dan multiple-row subqueris(banyak baris) dimana hanya satu kolom saja yang ingin ditampilkan dengan perintah select dan ini digunakan untuk mengevaluasi dari expression dalam induk perintah select. Jika kita ingin membandingkan dua atau lebih kolom, kita harus menulis sebuah gabungan klausa WHERE menggunakan operator logika. Menggunakan multiple-subqueries, kita dapat mengkombinasikan duplikat kondisi where ke dalam satu klausa WHERE.

Sintaks Umum.

```
select kolom, kolom,...  
from table  
where (kolom, kolom,...) IN  
        (select kolom, kolom,..  
        From table  
        Where kondisi);
```

1. SubQuery Perbandingan Kolom

Multiple-kolom membandingkan ke dalam subquery dan dijadikan campuran perbandingan atau pasangan perbandingan. Jika kita mempertimbangkan contoh “menampilkan secara detail dari pekerja yang bekerja di beberapa department dan memiliki beberapa manager seperti ‘Daniel’”, kita dapat melihat hasilnya dari statement berikut:

```
Select first_name, last_name, manager_id,  
department_id from employees  
        where manager_id IN (select manager_id from employees  
        where first_name='Daniel')  
        AND department_id IN (select department_id  
        From employees where first_name='Daniel');
```

2. SubQuery Pasangan Perbandingan Subquery

Menampilkan secara detail dari pekerja yang di manage oleh beberapa manager and bekerja di beberapa department sebagai pekerja yang first_name nya adalah “John”.

```
select employee_id, manager_id, department_id
from employees where
      (manager_id, department_id) IN
      ( select manager_id, department_id from employees
      where first_name='John')
      AND first_name <> 'John'
```

Pada contoh diatas adalah contoh tampilan hasil dari query kombinasi nilai dari kolom manager_id dan kolom department_id untuk masing-masing baris pada table employees dengan nilai pada kolom manager_id dan kolom department_id untuk pekerja dengan first_name adalah “John”, subquery untuk mendapatkan kembali nilai dari manager_id dan department_id untuk employees dengan first_name “John” dieksekusi.

Nilai perbandingan dengan kolom manager_id dan department_id untuk masing-masing baris dalam table employees. Jika kombinasi sesuai, baris akan ditampilkan. Dalam output, record dari employees dengan first_name “John” tidak akan ditampilkan.

3. SubQuery NonPasangan Perbandingan Subquery

Menampilkan secara detail dari pekerja yang dimanage beberapa manager pekerja dengan first_name “John” dan bekerja di beberapa department dengan first_name John

```
select employee_id, manager_id, department_id
from employees
where manager_id IN
(select manager_id
from employees
where first_name='John')
AND
department_id IN
(select department_id
from employees
where first_name='John')
AND
first_name<>'John';
```


4. Ekspresi-ekspresi Scalar Subquery

Scalar subquery adalah subquery yang mengembalikan tepat satu nilai kolom dari satu baris. Scalar subquery dapat digunakan di:

- Bagian kondisi dan ekspresi dari DECODE and CASE
- Semua klausa dari SELECT kecuali GROUP BY

Contoh scalar subquery pada CASE expression :

```
select employee_id, last_name,(
CASE
when department_id=
(select department_id from departments
where location_id=1800)
THEN 'Canada' ELSE 'USA' END) location from employees
```

Contoh subquery dalam klausa order by :

```
select employee_id, last_name from employees e
order by ( select department_name from departments d
where e.department_id=d.department_id);
```

5. Korelasi Subquery

Korelasi subquery digunakan untuk proses baris-perbaris. Masing-masing subquery di eksekusi setelah setiap baris setelah query paling luar. Oracle server menampilkan sebuah korelasi query mengacu sebuah kolom dari table yang menunjuk pada parent statement. Korelasi subquery dievaluasi ketika setiap baris diproses oleh statement induk. Induk statement dapat berupa select, update atau delete.

Sintaks umum :

```
Select kolom1, kolom2,...
From table1 outer
Where kolom1 operator
(select kolom1, kolom2,..
From table2
Where ekspresi=outer.ekspresi2);
```

Menampilkan semua pekerja yang mendapatkan lebih dari rata-rata gaji dari department tertentu

```
select last_name, salary, department_id from
employees outer
```

```

where salary >
(select AVG(salary)
from employees
where department_id=outer.department_id);

```

6. Menggunakan Operator Exists

Operator EXISTS mengecek keberadaan dari baris dalam hasil subquery.

Jika Nilai baris dari subquery ditemukan:

- o Pencarian tidak akan dilanjutkan dalam query terdalam
- o Kondisi tetap benar

Jika Nilai baris dari subquery tidak ditemukan:

- o Kondisi akan salah
- o Pencarian akan diteruskan samapi pada query paling dalam.

Menemukan pekerja yang paling sedikit memberikan laporan :

```

select employee_id, last_name, job_id, department_id
from employees outer
where EXISTS ( select 'X' from employees
where manager_id=outer.employee_id);

```

7. Korelasi Update

Menggunakan korelasi query untuk update baris berdasar pada baris dari table lain

Sintaks Umum.

Update table1 alias

Set kolom=(select expression

From table2 alias

Where alias1.kolom=alias2.kolom);

Denormalisasi table EMP16 dengan menambahkan sebuah kolom untuk memasukan pada department name.

```
Create table emp16 as select * from employees;
```

```
Alter table emp16 add(department_name varchar2(25));
```

```
Select department_name from emp16;
```

```
update emp16 e set department_name=
```

```

(select department_name from departments d
where e.department_id=d.department_id);

```

```
Select department_name from emp16;
```

8. Korelasi Delete

Menggunakan korelasi subquery untuk mendelete row berdasarkan pada baris dari table lain

Sintaks umum :

```
DELETE from table1 alias
WHERE kolom operator
(select expression from table alias2
Where alias1.kolom=alias2.kolom);
DELETE from emp16 E where employee_id=(
select      employee_id      from      employees      where
employee_id=E.employee_id);
select * from emp16;
```

C. Alat dan Bahan

1. Komputer dengan sistem operasi Windows 7
2. Program aplikasi XAMPP dengan PhpMyAdmin
3. Modul Praktikum Sistem Berkas dan Basis Data

D. Langkah Praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buka Command Prompt dan login sebagai root ke MySQL seperti di langkah pada Modul 1.
4. Pilih database perbankan dengan perintah “use perbankan”. Sehingga akan muncul pemberitahuan “database changed”
5. Tampilkan detail data nasabah yang pernah melakukan transaksi kredit dengan menggunakan syntax subquery dibawah ini:

```
SELECT * FROM nasabah where nasabah.id_nasabah IN (select DISTINCT
transaksi.id_nasabahFK from transaksi where jenis_transaksi='kredit');
```

6. Tampilkan detail data nasabah yang tidak pernah melakukan segala jenis transaksi dengan menggunakan syntax sub query dibawah ini:

```
SELECT * FROM nasabah where nasabah.id_nasabah NOT IN (select
DISTINCT transaksi.id_nasabahFK from transaksi);
```

7. Tampilkan cabang bank yang memiliki nilai rata-rata saldo paling besar untuk seluruh rekening dalam cabang tersebut menggunakan syntax sub query dibawah ini:

```
SELECT      cabang_bank.kode_cabang,      cabang_bank.nama_cabang,
AVG(rekening.saldo) as rata2 from cabang_bank, rekening where
cabang_bank.kode_cabang=rekening.kode_cabangFK      group      by
cabang_bank.kode_cabang having rata2 >= ALL (select AVG (saldo) FROM
`rekening` group by kode_cabangFK);
```

8. Tampilkan data transaksi yang melibatkan jumlah uang lebih besar dari rata-rata nilai keseluruhan transaksi yang pernah dilakukan menggunakan syntax sub query dibawah ini:

```
Select * from transaksi where jumlah > (SELECT AVG(jumlah) FROM `transaksi`);
```

E. Tugas

Gunakan sub query untuk mendapatkan data-data berikut dari databse data kuliah yang telah anda bangun pada modul 4 dan 5.

1. Ambil salah satu mata kuliah. Tampilkan daftar mahasiswa yang tidak mengambil mata kuliah tersebut.
2. Satu dosen dapat mengampu lebih dair satu mata kuliah dan satu mahasiswa dapat mengambil lebih dari satu mata kuliah. Tampilkan daftar mahasiswa yang mengambil semua mata kuliah yang diampu oleh salah satu dosen!
3. Karena salah satu mata kuliah (A) dihilangkan, seluruh mahasiswa yang mengambil matakuliah tersebut dipindahkan untuk mengambil mata kuliah lain (B). Lakukan update data menggunakan sub query!

Modul 10

MySQL dan Python

A. Tujuan

1. Mahasiswa memahami cara koneksi MySQL pada Python
2. Mahasiswa memahami penggunaan MySQL pada Python

B. Landasan Teori

1. MySQL Connector

MySQL connector adalah framework yang dibuat oleh MySQL untuk dapat mendukung berbagai macam bahasa pemrograman. Dengan memanfaatkan MySQL connector, pengembang perangkat lunak dapat mengolah database yang dibangun menggunakan bahasa pemrograman lain. Berikut adalah beberapa connector yang disediakan oleh MySQL :

- ADO.NET Driver (Connector/NET)
- ODBC Driver (Connector/ODBC)
- JDBC Driver (Connector/J)
- Python Driver (Connector/Python)
- C++ Driver (Connector/C++)
- C Driver (Connector/C)
- C API (mysqlclient)

2. Bahasa Pemrograman Python

Python adalah bahasa pemrograman yang diciptakan oleh Guido van Rossum pada tahun 1991. Python adalah salah satu bahasa pemrograman tingkat tinggi yang bersifat interpreter, interaktif, object oriented, dan multi-platform. Sebagai bahasa tingkat tinggi python termasuk salah satu bahasa pemrogramman yang mudah di pelajari karena syntaks yang jelas dan elegan, di kombinasikan dengan penggunaan module-module siap pakai dan struktur data tingkat tinggi yang efisien.

Bahasa pemrograman ini mudah dibaca dan terstruktur karena menggunakan indentasi. Berbeda dengan java atau C yang memanfaatkan kurung kurawal untuk menandai blok program, pada bahasa pemrograman ini setiap blok kode dipisahkan dengan susunan indentasi. Untuk memasukkan sub program dalam sebuah blok program, sub program tersebut harus diletakkan satu atau lebih spasi dari kolom suatu blok.

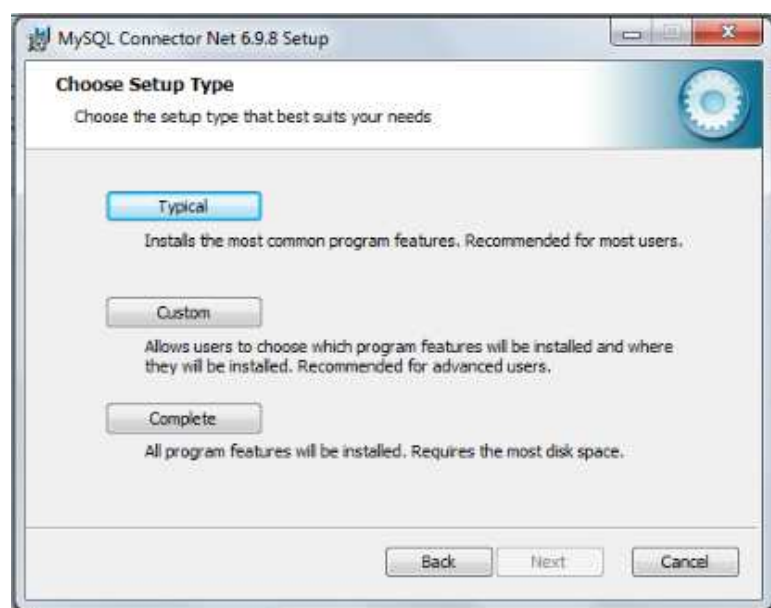
3. Koneksi MySQL pada Python

Sebelum dapat melakukan instalasi MySQL Connector, terlebih dahulu lakukan instalasi Python. Python telah secara otomatis terinstall pada sistem operasi UNIX dan sistem operasi sejenis UNIX seperti Linux, OS X, dan FreeBSD. Pada sistem operasi Windows, installer python dapat diakses pada <http://python.org/download/>.

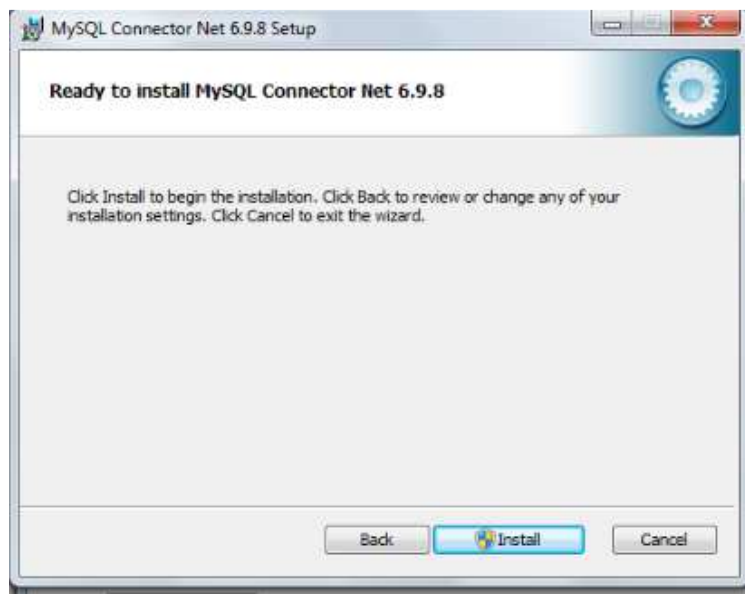
- Unduh konektor untuk python pada halaman berikut <http://dev.mysql.com/downloads/connector/python/>. Pastikan untuk mengunduh versi konektor yang sesuai dengan versi python yang terinstall pada perangkat anda.
- Lakukan instalasi package tersebut.



- Pilih mode instalasi yang diinginkan



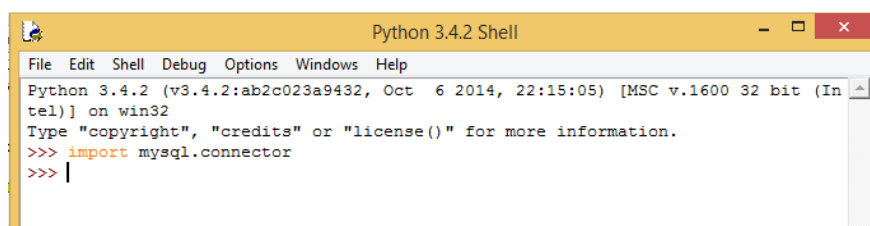
- Pilih tombol install untuk memulai instalasi



- Tunggu beberapa saat sampai proses instalasi selesai dilakukan. Kemudian tekan finish.



- Untuk melakukan pengecekan, masukkan syntax “import mysql.connector” pada python command line. Apabila tidak terdapat pemberitahuan kesalahan, maka instalasi konektor berhasil dilakukan.



C. Alat dan Bahan

1. Python yang sudah terinstall.
2. MySQL connector yang sudah terinstall.

D. Langkah-Langkah Praktikum

1. Jalankan XAMPP Control Panel.
2. Jalankan server Apache dan MySQL.
3. Buat sebuah file python dengan ekstensi .py
4. Masukkan kode berikut ini untuk memasukkan data transaksi baru pada database perbankan.

```
from datetime import date, datetime, timedelta
import mysql.connector

cnx = mysql.connector.connect(user='root', database='perbankan')
cursor = cnx.cursor()
tanggal = datetime.now().date()
tambah_transaksi = ("INSERT INTO transaksi "
                    "(`id_nasabahFK`, `no_rekeningFK`, `jenis_transaksi`, `tanggal`, `jumlah`) "
                    "VALUES (%s, %s, %s, %s, %s)")
data_transaksi = ('9', '110', 'kredit', tanggal, '50000')
cursor.execute(tambah_transaksi, data_transaksi)

cnx.commit()
cursor.close()
cnx.close()
```


5. Lakukan query untuk mengetahui transaksi yang dilakukan hari ini dengan menggunakan k

```
from datetime import date, datetime, timedelta

import mysql.connector

cnx = mysql.connector.connect(user='root', database='perbankan')
cursor = cnx.cursor()

query = ("SELECT `id_nasabahFK`, `jenis_transaksi`, `tanggal`, `jumlah`"
        "FROM transaksi "
        "WHERE YEAR(tanggal)='2016'")

cursor.execute(query)

for (id_nasabahFK, jenis_transaksi, tanggal, jumlah) in cursor:
    print("nasabah dengan ID {} melakukan transaksi {} pada {:%d %b %Y} sejumlah {}".format(
        id_nasabahFK, jenis_transaksi, tanggal, jumlah))

cursor.close()

cnx.close()
```

E. Tugas

1. Buat kode program python untuk melakukan perintah INSERT, UPDATE, dan DELETE pada data transaksi.
2. Buatlah kode program python untuk mendapatkan :
 - a. Data nasabah.
 - b. Data nasabah yang melakukan transaksi antara bulan oktober sampai desember.