BAB 2 *DATA DEFINITION LANGUAGE* (DDL)

TUJUAN BELAJAR:

- Memahami tipe data yang di dukung oleh DBMS
- Memahami dan menerapkan sintaks DDL
- Memahami dan menerapkan constraint

2.1 TIPE DATA

Tipe-tipe data yang didukung oleh basis data tidak jauh berbeda dengan yang telah dikenal dalam pemrograman pada umumnya. Namun, antara satu *Database Management System* (DBMS) dengan DBMS yang lain bisa memiliki perbedaan, meskipun memiliki standardisasi SQL. Berikut ini adalah beberapa tipe data yang digunakan di SQL Server.

Tipe data String:

Tipe data	Keterangan
CHAR(N)	*String* dengan panjang tetap *n* (maks. 8000 karakter)
VARCHAR(N)	*String* dengan panjang tidak tetap maks *n* (maks. 8000 karakter)
TEXT	*String* dengan panjang tidak tetap
NCHAR	*String* Unicode dengan panjang tetap (maks. 4000 karakter)
NVARCHAR	*String* Unicode dengan panjang tidak tetap (maks. 4000 karakter)

Tipe data	Keterangan
NTEXT	String Unicode dengan panjang tidak tetap hingga 2 GB
BIT	Berisi 0, 1, dan NULL
BINARY(N)	*String* biner dengan panjang tetap *n*
VARBINARY	*String* biner dengan panjang tidak tetap *n*
IMAGE	*String* biner dengan panjang tidak tetap untuk menyimpan *image*

Tipe data angka:

Tipe data	Keterangan
TINYINT	Bilangan bulat 0..255 (8 bit/1 byte)
SMALLINT	Bilangan bulat 16 bit/2 byte
INT	Bilangan bulat 32 bit/4 byte
BIGINT	Bilangan bulat 64 bit/8 byte
DECIMAL(P, S) NUMERIC(P, N)	Bilangan desimal 5-17 bytes (tergantung ketelitian desimal) – bernilai pasti (*exact*) hingga 38 digit

Tipe data	Keterangan
FLOAT	Bilangan desimal 4 bytes (bernilai aproksimasi)
REAL	Bilangan desimal 8 bytes (bernilai aproksimasi)
MONEY	Menyimpan bilangan untuk uang 8 bytes
SMALLMONEY	Menyimpan bilangan untuk uang 4 bytes

Tipe data tanggal/waktu:

Tipe data	Keterangan
DATETIME	Dari 1 January 1753 s.d. 31 December 9999 dengan akurasi 3,33 milidetik
DATETIME2	Dari 1 January 0001 s.d. 31 December 9999 dengan akurasi 100 nanodetik
SMALLDATETIME	Dari 1 January 1900 s.d. 31 December 9999 dengan akurasi 1 menit
DATE	Menyimpan tanggal saja dari 1 January 1753 s.d. 31 December 9999

Tipe data	Keterangan
DATETIMEOFFSET	Sama dengan datetime2 tetapi menyimpan *offset* zona waktu
TIME	Menyimpan waktu saja dengan akurasi 100 nanodetik
TIMESTAMP	Menyimpan angka unik yang diperbarui tiap kali baris *record* dibuat/ dimodifikasi berdasarkan *internal clock*

SQL Server juga mendukung beberapa tipe data yang lain misalkan `uniqeidentifier` berupa *globally unique identifier* (GUID) sebagai kode unik yang dibuat secara acak, `xml` untuk menyimpan data berupa XML, dan masih ada yang lain[1].

2.2 MEMBUAT DATABASE DENGAN *CREATE*

Untuk membuat basis data digunakan sintaksis `CREATE DATABASE` diikuti nama *database*. Contoh pembuatan basis data sebagai berikut:

```
CREATE DATABASE mydatabase
```

Maka akan membuat basis data bernama `mydatabase`. Untuk memilih dan menggunakan basis data `mydatabase`, gunakan sintaksis sebagai berikut.

```
use mydatabase
```

2.3 MEMBUAT TABLE DENGAN CREATE

Setelah kita memilih basis data yang akan digunakan, maka kita bisa membuat tabel di dalam basis data tersebut. Untuk membuat tabel maka digunakan sintaksis `CREATE TABLE`.

2.3.1 *Table* Sederhana

```
CREATE TABLE mytable(
        col int
)
```

Contoh di atas digunakan untuk membuat tabel `mytable` dengan satu atribut bernama `col` dengan tipe data int (*integer* 32 bit)

2.3.2 Table dengan *Default Value*

Pada tabel `Users` berikut, *field* profession diisi dengan nilai *default* yaitu berupa *string* "Student". Nantinya saat penambahan *record* baru, apabila *field* profession ini tidak diisi secara eksplisit maka secara otomatis isinya adalah "Student".

[1] *Data Types (Transact-SQL)* (https://msdn.microsoft.com/en-us/library/ms187752.aspx)

```
CREATE TABLE Users (
    name       CHAR(20),
    age        INTEGER,
    profession VARCHAR(30) default 'Student'
)


CREATE TABLE Department (
    deptno     SMALLINT      NOT NULL  IDENTITY (100, 1),
    deptname   VARCHAR(36)   NOT NULL,
    mgrno      CHAR(6),
    admrdept   SMALLINT      NOT NULL,
    location   CHAR(30)
)
```

Pada tabel Department, field deptno diberi nilai *default* mulai dari angka 100 dan apabila ada record baru maka record berikutnya memiliki deptno dengan penambahan (increment) 1. Misalnya apabila tabel masih kosong, apabila ada *record* baru (baris pertama), maka deptno secara otomatis bernilai 100. Apabila ada *record* baru lagi (baris kedua), maka deptno dari record baru bernilai 101, deptno baris ketiga bernilai 102, dst.

2.3.3 Table dengan NOT NULL Value

Pada tabel users dan department di atas, NOT NULL mengindikasikan bahwa *field* tersebut tidak boleh bernilai kosong/tidak berisi. Berbeda apabila tidak ada NOT NULL, misalnya field age pada tabel users, bila tidak diisi nilainya secara eksplisit maka bernilai NULL.

2.3.4 Table dengan *Constraint*

Contraint pada suatu tabel mendefinisikan aturan-aturan yang membatasi suatu field. Ada beberapa *constraint* antara lain:

a. UNIQUE

Fungsinya adalah menjaga suatu *field* pada suatu tabel tidak boleh berisi nilai yang sama (duplikasi). Namun NULL diperbolehkan menjadi nilai suatu *field* yang UNIQUE. Contoh tabel dengan UNIQUE:

```
CREATE TABLE Persons (
    P_Id int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
)

CREATE TABLE Persons (
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    UNIQUE (P_Id)
)
```

```
CREATE TABLE Persons (
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CONSTRAINT uc_PersonID UNIQUE (P_Id, LastName)
)
```

b. PRIMARY KEY

Fungsinya menjaga suatu *field* tidak boleh berisi sama dan tidak boleh berisi NULL. Fungsi dari *primary key* adalah sebagai pembeda antara satu *record* (entitas) dengan *record* yang lainnya (sebagai suatu *identifier*)

```
CREATE TABLE Persons (
    P_Id int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
)
```

c. REFERENTIAL

Fungsinya untuk menjaga hubungan (*relationship*) antar tabel. Tabel yang dirujuk memiliki *primary key*, sedangkan tabel yang merujuk memiliki *foreign key,* dimana *primary key* dan *foreign key* memiliki domain nilai yang sama.

```
CREATE TABLE Orders (
    O_Id int NOT NULL PRIMARY KEY,
    OrderNo int NOT NULL,
    P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

Pada contoh di atas, *field* P_Id tabel Orders akan merujuk ke *primary key* P_Id pada tabel Persons. Pada *referential integrity* ini, nilai *field* pada *foreign key* P_id tabel Orders **harus ada** nilainya di *primary key* P_Id tabel Persons.

d. CHECK

CHECK berfungsi membatasi nilai/data apa saja yang bisa dimasukkan pada suatu *field*. Selain itu juga berfungsi menjaga supaya data yang dimasukkan pada suatu *field* sesuai dengan aturan yang dibuat.

```
CREATE TABLE Employee (
    ID          INTEGER     NOT NULL    PRIMARY KEY,
    Name        VARCHAR(9),
    ZipCode     CHAR(5)     CHECK (ZipCode     LIKE '[0-9][0-9][0-9][0-9][0-9]'),
    Dept        SMALLINT    CHECK (Dept BETWEEN 10 AND 100),
    Job         CHAR(5)     CHECK (Job IN ('Sales','Manager','Clerk')),
    HireDate    DATE,
    Salary      DECIMAL(7,2),
    CONSTRAINT  YEARSAL     CHECK ( YEAR(HireDate) > 1986 OR Salary > 40500 )
)
```

2.4 MELIHAT DATA PADA TABLE

Untuk melihat data yang sudah kita masukkan pada tabel, maka digunakan perintah SELECT. Misalnya untuk melihat seluruh baris data pada seluruh *field* yang ada di tabel Employee, *query*-nya:

```
SELECT * FROM Employee
```

Apabila ingin mengambil data pada sebagian *field* saja (operasi *project*) dari tabel Employee, contohnya:

```
SELECT Name, Dept, HireDate FROM Employee
```

2.5 MEMBUAT VIEW DENGAN CREATE

View adalah tabel virtual yang berisi *result-set* (hasil *query* berisi 1 atau banyak baris dengan banyak kolom) dari suatu *statement* SQL. *View* seperti halnya tabel juga terdiri dari kolom dan baris, namun kolom dan isi data pada *view* sebenarnya berasal dari tabel. *View* ini bisa dibuat dengan *statement* SQL yang sederhana hingga kompleks, yang nantinya bisa berisi klausa WHERE, JOIN, dll yang akan dibahas pada bab-bab berikutnya.
Contoh pembuatan *view* bernama Employee2010 yang mana datanya berasal dari tabel Employee yang menampilkan semua pegawai yang dipekerjakan setelah tahun 2010:

```
CREATE VIEW Employee2010 AS
SELECT ID, Name, HireDate FROM Employee
WHERE YEAR(HireDate) > 2010
```

Cara untuk mengambil data dari *view* Employee2010 sama seperti pada tabel, misalnya:

```
SELECT * FROM Employee2010
```

2.6 MEMODIFIKASI OBJEK *DATABASE* DENGAN *ALTER*

Basis data yang telah dibuat masih bisa dimodifikasi. Untuk memodifikasi objek basis data dan tabel yang telah dibuat digunakan sintaksis ALTER.

2.6.1 Menambah Kolom/*Field* Baru

Sintaksis untuk menambah kolom/*field* baru pada tabel adalah:

```
ALTER TABLE [nama_tabel]
ADD [nama_kolom] [tipe_data] (constraint);
```

Contoh penambahan field baru State pada tabel Persons:

```
ALTER TABLE Persons
ADD State varchar(20);
```

2.6.2 Mengubah Tipe Data Dari Kolom/*Field*

Sintaksis untuk mengubah kolom/*field* yang sudah ada adalah:

```
ALTER TABLE [nama_tabel]
ALTER COLUMN [nama_kolom] [tipe_data];
```

Contoh pengubahan tipe data pada tabel Persons:

```
ALTER TABLE Persons
ALTER COLUMN City VARCHAR(75) NOT NULL;
```

2.6.3 Menghapus Kolom/*Field*

Sintaksis untuk menghapus kolom/*field* adalah:

```
ALTER TABLE [nama_tabel]
DROP COLUMN [nama_kolom]
```

Contoh penghapusan kolom:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth
```

2.6.4 Menambah Constraint

Sintaksis penambahan *constraint*:

```
ALTER TABLE Persons
ADD CONSTRAINT [nama_kolom] [jenis_constraint] [kondisi_constraint];
```

Contoh penambahan constraint *primary key*:

```
ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id, LastName)
```

2.6.5 Mengubah Nama Objek *Database*

Sintaksis pengubahan nama basis data adalah:

```
ALTER DATABASE [nama_basisdata]
MODIFY NAME = [nama_basisdata_baru];
GO
```

Contoh pengubahan nama basis data mydatabase ke my_newdatabase:

```
use master;
GO

ALTER DATABASE mydatabase
MODIFY NAME = my_newdatabase;
GO
```

Pada contoh di atas, apabila basis data mydatabase masih digunakan/terkoneksi maka aktifkan basis data yang lain, dalam hal ini adalah master. Setelah dijalankan melalui perintah GO (sebagai pembatas suatu blok baris perintah, GO bisa dikustomisasi) maka jalankan pengubahan nama sebenarnya.

2.7 MENGHAPUS OBJEK *DATABASE* DENGAN *DROP*

Sintaksis penghapusan objek basis data menggunakan DROP. Penghapusan bisa dilakukan untuk basis data, tabel, atau objek lainnya.

CATATAN:
Hati-hati untuk melakukan penghapusan objek, tabel, basis data, karena apabila suatu objek dihapus maka:

a. Semua data dan struktur tabel akan dihapus
b. Semua indeks akan dihapus
c. Semua transaksi *pending* akan di-*commit*
d. Perintah *drop* tidak bisa di-*rollback* (dikembalikan atau *undo/undrop*) secara permanen

2.7.1 Menghapus *Table*

Sintaksis menghapus tabel adalah:

```
DROP TABLE [nama_tabel]
```

Contoh menghapus tabel Persons adalah:

```
DROP TABLE Persons
```

2.7.2 Menghapus *Database*

Sintaksis menghapus basis data adalah:

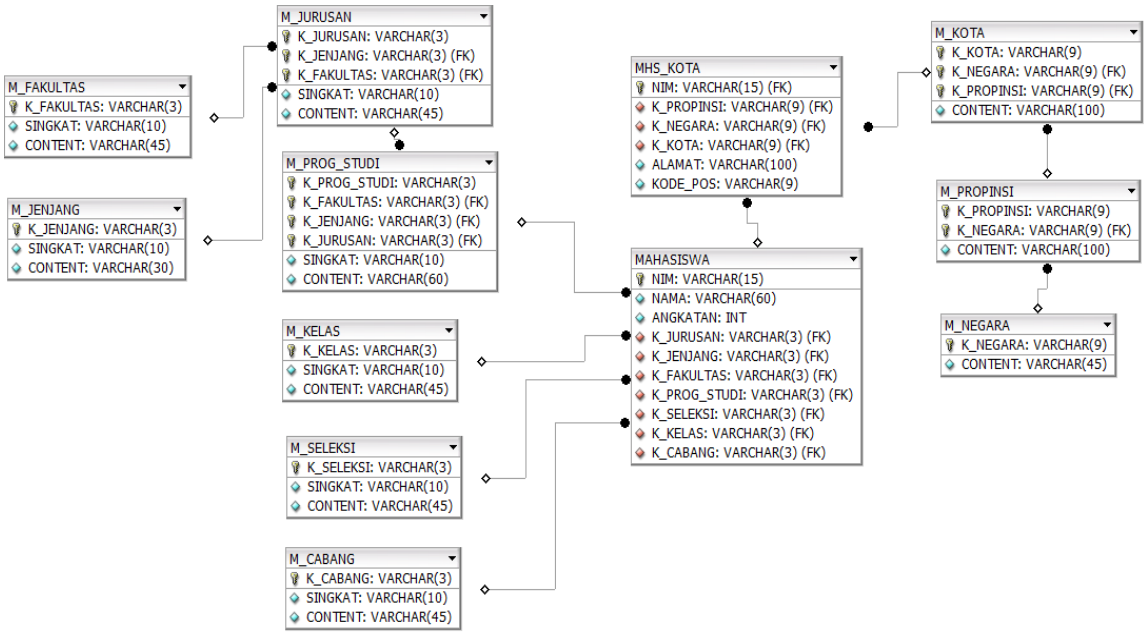
```
DROP DATABASE [nama_basisdata]
```

Contoh menghapus tabel mydatabase adalah:

```
DROP DATABASE mydatabase
```

2.8 LATIHAN

1. Implementasikan model relasional berikut ke DBMS.



2. Modifikasi *table* MHS_KOTA sehingga mempunyai struktur seperti berikut (perlihatkan hasilnya):

Name	Null?	Type	Constr.
NIM	NOT	VARCHAR(15)	FK PK
K_PROPINSI		VARCHAR(9)	FK
K_NEGARA		VARCHAR(9)	FK
K_KOTA		VARCHAR(9)	FK
ALAMAT		VARCHAR(110)	
KODE_POS		VARCHAR(6)	

3. Buatlah *table* MHS_KOTA2 yang memiliki struktur sama seperti *table* MHS_KOTA!
4. *Drop table* MHS_KOTA!