BAB 4
PENGELOLAAN DATABASE, TABEL, DAN INDEX

MATERI
a. Membuat, Memodifikasi, dan Menghapus Database
b. Membuat Tabel, Mendefinisikan Kolom, Menambah Batasan-batasan pada Tabel, dan Menspesifikasikan Tipe Tabel yang Dibuat
c. Membuat dan Menghapus Index pada Tabel
d. Memperoleh Informasi Database dan Tabel dan Cara Mengkonfigurasinya

MENGELOLA DATABASE

Setelah MySQL berhasil Anda instal, berikutnya Anda bisa membuat database dalam lingkungan MySQL. Database-database yang Anda buat, maka akan dibuatkan subdirektori yang berkaitan dengan database pada direktori **data**. Tabel-tabel yang Anda tambahkan ke database akan muncul sebagai file-file dalam subdirektori. Jika Anda menghapus database, maka subdirektori yang berkaitan dan file-filenya akan ikut dihapus.
Langkah pertama untuk men-*setup* database adalah membuat obyek database. Dari sini Anda bisa memodifikasi definisinya atau menghapus database.

MEMBUAT DATABASE

Membuat database dalam MySQL merupakan satu tugas atau aktivitas yang cukup mudah. Untuk mendefinisikan database hanya diperlukan satu kunci CREATE DATABASE, diikuti dengan nama database baru yang ingin Anda buat, sebagaimana terlihat pada sintaks berikut ini:

```
<definisi database>::=
CREATE DATABASE (IF NOT EXISTS) <nama database>
((DEFAULT) CHARACTER SET <nama character set>)
((DEFAULT) COLLATE <nama collation>)
```

Sebagaimana terlihat pada sintaks, hanya sedikit saja komponen yang diperlukan dalam pernyataan CREATE DATABASE. Dalam pernyataan tersebut juga disertakan beberapa elemen-elemen opsional. Pertama adalah IF NOT EXISTS, menandakan bagaimana MySQL merespon terhadap pernyataan CREATE DATABASE jika database dengan nama yang sama sudah ada. Jika database sudah ada dan klausa IF NOT EXISTS tidak digunakan, MySQL akan menampilkan suatu *error*. Jika klausa digunakan, MySQL akan menampilkan suatu peringatan, tanpa menampilkan suatu *error*. Baik klausa tersebut disertakan pada pernyataan yang Anda buat, efek pada database adalah sama: Jika database dengan nama yang sama telah ada, maka tidak akan dibuatkan database baru.

Komponen opsional lainnya adalah klausa CHARACTER SET dan COLLATE. Anda dapat menspesifikasikannya satu atau keduanya. Klausa CHARACTER SET menspesifikasikan karakter default yang digunakan untuk database baru, dan klausa COLLATE menspesifikasikan *collation* (pembandingan) default yang digunakan. Se-set karakter merupakan kumpulan dari huruf, angka, dan simbol-simbol untuk membuat nilai-nilai dalam database. Misal, A,B,C,a,b,c,1,2,3,>,+, dan * adalah merupakan bagian dari set karakter. Sedangkan *collation* adalah urutan pengurutan yang digunakan untuk set karakter tertentu. Misal, beberapa *collation* berkaitan dengan set karakter default, termasuk latin1_bin, latin1_general_ci, dan latin1_swedish-ci, dimana merupakan *collation* default.

Anda dapat menampilkan set karakter dan collation yang tersedia di sistem Anda dengan mengeksekusi pernyataan SHOW CHARACTER dan SHOW COLLATION di utilitas mysql client. Perhatikan juga bahwa set karakter dan collation hanya berpengaruh pada data string (huruf, angka, dan simbol), tidak sebaliknya dengan data numerik atau data yang berhubungan dengan tanggal dan waktu.

Sekarang Anda sudah sedikit mengerti tentang sintaks yang digunakan untuk membuat database. Berikut ini contoh pernyataan dasar CREATE DATABASE yang tidak menyertakan komponen opsional apapun.

```
CREATE DATABASE TokoBuku;
```

Ketika Anda mengeksekusi pernyataan tersebut, database dengan nama TokoBuku akan ditambahkan pada sistem Anda. Database menggunakan set karakter dan *collation* default karena Anda tidak menspesifikasikannya.

Ketika Anda membuat database dan tabel dalam Windows, semua nama dikonversi ke lowercase (huruf kecil). Karena nama file dan direktori di Windows bersifat tidak sensitif (case insensitive). Namun berbeda dengan sistem operasi berbasi Unix. Anda harus benar-benar memperhatikan tulisannya, karena bersifat case-sensitif.

Pada contoh berikut, pernyataan CREATE DATABASE menspesifikasikan set karakter dan *collation*:

```
CREATE DATABASE TokoBuku
DEFAULT CHARACTER SET latin1
DEFAULT COLLATE latin1_bin;
```

MEMODIFIKASI DATABASE

Suatu saat Anda mungkin ingin mengubah Eaton karakter dan *collation* yang digunakan dalam database Anda. Untuk melakukannya, Anda dapat menggunakan pernyataan ALTER DATABASE untuk menspesifikasikan *setting* baru. Berikut ini sintaksnya:

```
ALTER DATABASE <nama database>
((DEFAULT) CHARACTER SET <nama character set>)
((DEFAULT) COLLATE <nama collation>)
```

Dalam pernyataan ini, Anda harus menspesifikasikan kata kunci ALTER DATABASE dan nama dari database, bersamaan dengan klausa CHARACTER SET, klausa COLLATE, atau keduanya. Misal, mengubah set karakter ke latin1 untuk database TokoBuku:

```
ALTER DATABASE TokoBuku
CHARACTER SET latin1;
```

Dalam pernyataan di atas hanya set karakter yang diubah, sedangkan *collation* tetap yang dulu.

MENGHAPUS DATABASE

Untuk menghapus database Anda dapat menggunakan pernyataan DROP DATABASE. Sintaksnya adalah:

```
DROP DATABASE (IF EXISTS) <nama database>
```

Pernyataan di atas hanya memerlukan kata kunci DROP DATABASE dan nama database. Dengan tambahan, Anda dapat menentukan klausa IF EXISTS. Jika Anda menspesifikasikan klausa ini dan nama database yang tidak terdapat di sistem, Anda akan menerima pesan peringatan, bukan sebuah *error*.

```
DROP DATABASE TokoBuku
```

Contoh di atas menghapus database TokoBuku dari sistem. Ketika Anda menghapus database, berarti juga menghapus tabel-tabel dalam database tersebut dan data yang terdapat dalam tabel. Sehingga Anda harus benar-benar berhati-hati dalam mengeksekusi perintah DROP DATABASE.

MENGELOLA TABEL

Langkah berikutnya untuk mengatur database, setelah membuat obyek database aktual, adalah menambahkan tabel-tabel yang diperlukan ke database tersebut. Tabel-tabel tersebut menyediakan sebuah struktur untuk menyimpan dan mengamankan data. Semua data berada di dalam struktur tabel, dan semua tabel

berada dalam struktur database. Sebagai tambahan untuk membuat tabel, Anda juga dapat memodifikasi definisi tabel atau menghapus tabel dari database.

MEMBUAT TABEL

Untuk membuat sebuah tabel dalam MySQL, Anda harus menggunakan pernyataan CREATE TABLE untuk mendefinisikan kolom-kolom dalam tabel dan mengkonfigurasi batasan-batasan yang sesuai dalam tabel tersebut. Pernyataan CREATE TABLE adalah salah satu pernyataan SQL yang paling kompleks dalam MySQL. Dia berisi komponen-komponen yang banyak dan menyediakan banyak pilihan untuk menentukan sifat yang sebenarnya dari suatu tabel tertentu. Sintaks berikut merepresentasikan elemen-elemen yang daat digunakan dalam membangun pernyataan CREATE TABLE:

```
<definisi tabel>::=
CREATE (TEMPORARY) TABLE (IF NOT EXISTS) <nama tabel>
(<elemen tabel> ({, < elemen tabel >}...))
(<opsi tabel> (<opsi tabel >...))

<elemen tabel>::=
<definisi kolom>
| {(CONSTRAINT <nama constraint>) PRIMARY KEY
(<nama kolom> ({, < nama kolom >}...))}
| {(CONSTRAINT < nama constraint >) FOREIGN KEY (<nama indeks>)
(<nama kolom > ({, < nama kolom >}...)) <definisi referensi>}
| {(CONSTRAINT < nama constraint >) UNIQUE (INDEX) (<nama indeks>)
(<nama kolom > ({, < nama kolom >}...))}
| {{INDEX | KEY} (<nama indeks>) (<nama kolom > ({, < nama kolom >}...))}
| {FULLTEXT (INDEX) (<nama indeks >) (<nama kolom > ({, < nama kolom >}...))}

<definisi kolom>::=
<nama kolom> <type> (NOT NULL | NULL) (DEFAULT <value>) (AUTO_INCREMENT)
(PRIMARY KEY) (COMMENT `<string>') (<definisi referensi >)

<type>::=
<tipe data numerik>
| <tipe data string>
| <tipe data tanggal/waktu>

<definisi referensi>::=
REFERENCES <nama tabel> ((<nama kolom > ({, < nama kolom >}...)))
(ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT })
(ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT })
(MATCH FULL | MATCH PARTIAL)

<opsi tabel>::=
{ENGINE = {BDB | MEMORY | ISAM | INNODB | MERGE | MYISAM}}
| <opsi tabel tambahan>
```

Sebagaimana Anda lihat, banyak elemen untuk membuat pernyataan CREATE TABLE. Dalam kenyataanya, sintaks yang nampak di sini tidak seluruhnya dipakai untuk membuat pernyataan CREATE TABLE.
Sekarang kita lihat lebih dekat pernyataan CREATE TABEL berikut:

```
<table definition>::=
CREATE (TEMPORARY) TABLE (IF NOT EXISTS) <table name>
(<table element> ({, <table element>}...))
(<table option> (<table option>...))
```

Baris pertama dari pernyataan hanya memerlukan kata kunci CREATE TABLE diikuti oleh nama dari tabel baru. Baris ini juga berisi dua komponen pilihan. Pertama-TEMPORARY- menandakan bahwa ini adalah tabel temporer yang digunakan hanya selama sesi terkini oleh pengguna terkini. Sebuah tabel temporer ada hanya selama sesi sedang terbuka atau tabel secara eksplisit terhapus. Pilihan kedua adalah klausa IF NOT EXISTS. Klausa ini sama dengan klausa sebelumnya yang pernah Anda lihat dalam pernyataan CREATE DATABASE.
Elemen tabel adalah obyek individu yang didefinisikan dalam sebuah tabel, misal kolom atau batasan PRIMARY KEY. Setiap pernyataan CREATE TABLE menyertakan satu atau lebih elemen tabel. Jika terdapat lebih ddari satu elemen tabel, mereka dipisahkan dengan menggunakan tanda koma. Berapapun jumlah elemen tabel yang ada, mereka semua ditutup dalam sebuah kumpulan tanda kurung.

Untuk baris terakhir mengijinkan Anda untuk mendefinisikan opsi-opsi tabel secara individual. Opsi tabel adalah opsi yang diterapkan pada keseluruhan tabel. Misal, Anda dapat mendefinisikan tipe tabel yang ingin Anda buat. Semua opsi tabel adalah opsional; bagaianapun, Anda dapat menentukannya sebanyak yang Anda perlukan.
Untuk membuat tabel secara minimal, sintaksnya adalah sebagai berikut:

```
CREATE TABLE <nama tabel> (<elemen tabel>)
```

Karena elemen tabel memerlukan komponen, lihat pada sintaks berikut:

```
<elemen tabel>::=
<definisi kolom>
 | {(CONSTRAINT <nama constraint>) PRIMARY KEY
(<nama kolom> ({, < nama kolom >}...))}
 | {(CONSTRAINT < nama constraint >) FOREIGN KEY (<nama indeks>)
(<nama kolom > ({, < nama kolom >}...)) <definisi referensi>}
 | {(CONSTRAINT < nama constraint >) UNIQUE (INDEX) (<nama indeks>)
(<nama kolom > ({, < nama kolom >}...))}
 | {{INDEX | KEY} (<nama indeks>) (<nama kolom > ({, < nama kolom >}...))}
 | {FULLTEXT (INDEX) (<nama indeks >) (<nama kolom > ({, < nama kolom >}...))}
```

Elemen tabel dapat merepresentasikan satu atau lebih opsi-opsi yang berbeda. Yang paling umum digunakan adalah opsi yang direpresentasikan dalam <definisi kolom>, dimana mengijinkan Anda untuk mendefinisikan kolom yang disertakan dalam definisi tabel Anda.

Membuat Definisi Kolom

Definisi kolom adalah salah satu tipe elemen tabel yang dapat Anda definisikan dalam definisi tabel. Anda harus membuat definisi kolom untuk setiap kolom yang ingin anda sertakan dalam tabel Anda. Sintaks berikut ini menyediakan pada Anda dengan struktur yang sebaiknya Anda gunakan ketika membuat definisi kolom:

```
<definisi kolom>::=
<nama kolom> <type> (NOT NULL | NULL) (DEFAULT <value>) (AUTO_INCREMENT)
(PRIMARY KEY) (COMMENT `<string>') (<definisi referensi >)
```

Sebagaimana Anda lihat, hanya terdapat dua elemen yang diperlukan dalam sebah definisi kolom: nama kolom (direpresentasikan dengan <nama kolom>) dan tipe data (direpresentasikan dengan <type>). Nama dapat berupa *indentifier* yang dapat diterima dalam MySQL, dan database dapat berupa tipe-tipe data yang didukung. Setiap elemen tambahan dari definisi kolom adalah opsional.

Mendefinisikan Tipe Data
MySQL mendukung tiga kategori dari tipe data, sebagaimana direpresentasikan dalam sintaks berikut:

```
<type>::=
<tipe data numerik>
 | <tipe data string>
 | <tipe data tanggal/waktu>
```

Kapanpun Anda menambahkan sebuah kolom ke tabel Anda, Anda harus mendefinisikan kolom dengan tipe data yang terdapat dalam salah satu dari tiga kategori ini. Masing-masing kategori memiliki karakteristik sendiri-sendiri.

Tipe Data Numerik

Tipe data ini berhubungan dengan data angka.
Anda dapat membagi tipe data numerik dalam dua kategori, sebagaimana dalam sintaks berikut:

```
<tipe data numerik>::=
<tipe data integer> ((<panjang>)) (UNSIGNED) (ZEROFILL)
 | <tipe data fraksional> ((<panjang>, <desimal>)) (UNSIGNED) (ZEROFILL)
```

Untuk elemen <panjang>, mengindikasikan jumlah maksimum dari karakter yang ditampilkan untuk kolom tertentu. Anda dapat menspesifikasikan panjang dengan angka mulai dari 1 sampai 255. Tipe data fraksional juga menyertakan elemen <desimal>. Nilai di dalamnya menandakan jumlah desimal yang ditampilkan untuk

sebuah nilai dalam kolom tertentu. Anda dapat menspesifikasikan jumlah desimal mulai dari 0 sampai 30.
Opsi berikutnya adalah UNSIGNED. Jika opsi ini mengikuti tipe data numerik, tidak ada nilai negatif yang diijinkan berada dalam kolom. Jika Anda menspesifikasikan ZEROFILL, nol akan ditambahkan di awal nilai sehingga nilainya ditampilkan dengan sejumlah karakter yan direpresentasikan oleh <panjang>. Misal, jika Anda mendefinisikan <panjang> dengan 4 dan menspesifikasikan ZEROFILL, angka 53 akan ditampikan sebagai 0053.
Tipe data integer mengijinkan Anda menyimpan seluruh bilangan dalam kolom (tidak mengandung pecahan atau desimal). MySQL mendukung tipe data integer seperti yang ditampilkan dalam sintaks berikut:

```
<tipe data integer>::=
TINYINT | SMALLINT | MEDIUMINT | INT | INTEGER | BIGINT
```

Data type	Acceptable values	Storage requirements
TINYINT	Signed: –128 to 127 Unsigned: 0 to 255	1 byte
SMALLINT	Signed: –32768 to 32767 Unsigned: 0 to 65535	2 bytes
MEDIUMINT	Signed: –8388608 to 8388607 Unsigned: 0 to 16777215	3 bytes
INT	Signed: –2147483648 to 2147483647 Unsigned: 0 to 4294967295	4 bytes
INTEGER	Same values as the INT data type. (INTEGER is a synonym for INT.)	4 bytes
BIGINT	Signed: –9223372036854775808 to 9223372036854775807 Unsigned: 0 to 18446744073709551615	8 bytes

Sedangkan untuk tipe data fraksional, sintaksnya adalah:

```
<tipe data fraksional>::=
FLOAT | DOUBLE | DOUBLE PRECISION | REAL | DECIMAL | DEC | NUMERIC | FIXED
```

Data type	Description
FLOAT	An approximate numeric data type that uses 4 bytes of storage. The data type supports the following values: –3.402823466E+38 to –1.175494351E–38 0 1.175494351E–38 to 3.402823466E+38
DOUBLE	An approximate numeric data type that uses 8 bytes of storage. The data type supports the following values: –1.7976931348623157E+308 to –2.2250738585072014E–308 0 2.2250738585072014E–308 to 1.7976931348623157E+308
DOUBLE PRECISION	Synonym for the DOUBLE data type
REAL	Synonym for the DOUBLE data type
DECIMAL	An exact numeric data type whose range storage requirements depend on the <length> and <decimals> values specified in the column definition
DEC	Synonym for the DECIMAL data type
NUMERIC	Synonym for the DECIMAL data type
FIXED	Synonym for the DECIMAL data type

Contoh dalam pembuatan tabel dengan menggunakan tipe data numerik.

```
CREATE TABLE Katalog
(
IDProduk SMALLINT(4) UNSIGNED ZEROFILL,
Jumlah INT UNSIGNED,
Harga DECIMAL(7,2),
Berat FLOAT(8,4)
);
```

Tipe Data String

Tipe data string menyediakan fleksibilitas untuk menyimpan berbagai macam tipe data dari bit-bit individual sampai file-file besar. Tipe data string secara normal digunakan untuk menyimpan nama dan *titles* dan nilai apapun yang meliputi huruf dan angka. MySQL mendukung empat kategori dari tipe data string.

```
<tipe data string>::=
<tipe data karakter>
 | <tipe data biner>
 | <tipe data teks>
 | <tipe data list atau daftar>
```

Tipe data karakter mngkin salah satu yang akan sering Anda gunakan. Sintaksnya adalah sebagai berikut:

```
<tipe data karakter>::=
CHAR (<panjang>) (BINARY | ASCII | UNICODE)
VARCHAR (<panjang>) (BINARY)
```

Tipe data CHAR adalah tipe data *fixed-length* yang dapat menyimpan sampai 255 karakter. Tanda <panjang> menspesifikasikan jumlah karakter yang disimpan. Meskipun pada keadaan sebenarnya nilainya terdiri atas jumlah karakter yang lebih sedikit dari jumlah yang dispesifikasikan, ruang penyimpanan aktualnya adalah tetap (*fixed*) sebagaimana jumlah yang ditentukan.
Contoh:

```
CREATE TABLE Katalog
(
IDProduk SMALLINT,
Deskripsi VARCHAR(40),
Kategori CHAR(3),
Harga DECIMAL(7,2)
);
```

Kolom Kategori didefinisikan dengan tipe data CHAR(3). Hasilnya, kolom tersebut akan menyimpan sebanyak nol sampai 3 karakter per nilainya, namun jumlah penyimpanan yang dialokasikan untuk nilai tersebut selalu tiga byte, satu untuk setiap karakter.
Tipe data CHAR adalah tipe data yang sesuai digunakan saat Anda mengetahui berapa karakter terbanyak dari suatu nilai dalam kolom yang ada dan jika nilainya terdiri atas karakter alphanumerik. Jika Anda tidak tahu jumlah karakter dari setiap nilainya, Anda sebaiknya menggunakan tipe data VARCHAR. Tipe data VARCHAR juga mengijinkan Anda menspesifikasikan jumlah atau panjang maksimum; bagaimanapun, kebutuhan penyimpanan berdasarkan pada jumlah karakter yang sebenarnya, bukan pada nilai <panjang>.
Kembali pada contoh sebelumnya. Perhatikan bahwa kolom Deskripsi dikonfigurasi dengan tipe data VARCHAR(40). Artinya bahwa nilainya dapat mempunyai panjang yang bermacam-macam, sampai sepanjang 40 karakter. Hasilnya, jumlah penympanan yang sebenarnya adalah antara nol dan 40 byte, tergantung pada nilai aktualnya. Misal, sebuah nilai "Buku baru" memerlukan byte yang lebih sedikit daripada nilai "Buku teknologi informasi". Tipe data VARCHAR, sebagaimana tipe data CHAR, dapat menyimpan sampai 255 karakter.

Sebagai tambahan dari tipe data string, MySQL mendukung empat tipe data biner:

```
<tipe data biner>::=
TINYBLOB | BLOB | MEDIUMBLOB | LONGBLOB
```

Tipe data biner mendukung penyimpanan sejumlah besar data, seperti file gambar dan suara. Tipe ini sangat berguna ketika Anda memperkirakan nilainya akan tumbuh besar atau melebar.

Data type	Maximum size
TINYBLOB/TINYTEXT	255 characters (355 bytes)
BLOB/TEXT	65,535 characters (64 KB)
MEDIUMBLOB/MEDIUMTEXT	16,777,215 characters (16 MB)
LONGBLOB/LONGTEXT	4,294,967,295 characters (4 GB)

Contoh tabel:

```
CREATE TABLE Inventori
(
IDProduk SMALLINT UNSIGNED,
Nama VARCHAR(40),
Foto BLOB,
Jumlah INT UNSIGNED
);
```

Kolom Foto dapat menyimpan data biner dengan ukuran sampai 64 KB. Asumsi dalam kasus ini bahwa foto dapat diambil sesuai produk dan disimpan dalam ukuran file yang cukup kecil untuk menyesuaikan dengan kolom ini. Jika Anda memperkirakan bahwa foto dapat berukuran lebih besar, Anda harus menaikkannya ke tipe MEDIUMBLOB.

Tipe data teks sama dengan tipe data biner, sintaksnya:

```
<tipe data teks>::=
TINYTEXT | TEXT | MEDIUMTEXT | LONGTEXT
```

Tipe data teks juga mempunyai batasan ukuran dan keperluan penyimpanan yang sama sebagaimana tipe data biner.
Contoh:

```
CREATE TABLE Katalog
(
IDProduk SMALLINT UNSIGNED,
Nama VARCHAR(40),
DokumenDeskripsi TEXT CHARACTER SET latin1 COLLATE latin1_bin
);
```

Sebagaimana Anda lihat, kolom DokumenDeskripsi menyertakan klausa CHARACTER SET dan COLLATE.

Sekarang kita lihat tipe data *list*/daftar. Sintaksnya adalah:

```
<tipe data list/daftar>::=
{ENUM | SET} (<nilai> ({, <nilai>}...))
```

Tipe data ENUM mengijinkan Anda untuk menspesifikasikan daftar nilai yang dapat Anda gunakan dalam kolom yang dikonfigurasi dengan tipe tersebut. Ketika Anda memasukkan sebuah baris dalam tabel, Anda juga dapat memasukkan salah satu nilai yang ditentukan untuk tipe data dalam kolom. Kolom hanya dapat berisi satu nilai, dan dia harus salah satu dari yang terdapat di *list*/daftar. Tipe data SET juga menspesifikasikan daftar nilai yang dimasukkan dalam kolom. Tidak seperti tipe data ENUM, dimana Anda hanya dapat menspesifikasikan hanya satu nilai, tipe data SET mengijinkan Anda menspesifikasikan banyak nilai dari daftar.

Contoh:

```
CREATE TABLE Pembelian
(
IDPembelian SMALLINT UNSIGNED,
ModelSepeda VARCHAR(40),
WarnaSepeda ENUM('merah', 'biru', 'hijau', 'kuning'),
OpsiSepeda SET('rak', 'senter', 'helm', 'gembok')
);
```

Perhatikan bahwa daftar nilai mengikuti tipe data. Nilai ditutup dalam tanda petik tunggal dan dipisahkan oleh tanda koma, dan semua nilai tertutup dalam tanda kurung. Untuk tipe data ENM, Anda dapat menspesifikasikan sampai 65.535 nilai. Untuk tipe data SET, Anda dapat menspesifikasikan sampai 64 nilai.

Tipe Data Tanggal/Waktu

Sintaksnya:

```
<tipe data tanggal/waktu>::=
DATE | TIME | DATETIME | YEAR | TIMESTAMP
```

Tipe data tanggal/waktu mengijinkan Anda menspesifikasikan kolom yang berisi data spesifik seperti tanggal dan waktu. Tipe data ini mendukung jangkauan dan format seperti nampak pada tabel berikut:

Data type	Format	Range
DATE	YYYY-MM-DD	1000-01-01 through 9999
TIME	HH:MM:SS	–838:59:59 to 838:59:59
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 through 9999
YEAR	YYYY	1901 to 2155 (and 0000)
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00 to partway through 2037

Jika Anda ingin menyimpan data hanya tanggalnya saja bukan tahun, Anda dapat menggunakan tipe data DATE. Nilai yang Anda masukkan harus sesuai dengan format yang didefinisikan oleh tipe data tersebut. Bagaimanapun, salah satu tipe data yang menarik adalah TIMESTAMP, dimana begitu berbeda dengan tipe data lainnya. Saat Anda mengkonfigurasi sebuah kolom dengan tipe data ini, sebuah baris, ketika dimasukkan data atau update, secara otomatis menyediakan sebuah nilai berdasarkan waktu dan tanggal terkini.
Contoh:

```
CREATE TABLE PembelianBuku
(
IDPembelian SMALLINT UNSIGNED,
IDBuku SMALLINT UNSIGNED,
Copyright YEAR,
TanggalBeli TIMESTAMP
);
```

Kolom Copyright mengijinkan Anda untuk menambahkan nilai ke kolom yang terdapat pada range 1901 sampai 2155; bagaimanapun, Anda dilarang menambahkan nilai selsin dari tipe data ini. Kolom TanggalBeli secara otomatis menyimpan data dan waktu terkini ketika baris tertentu dimasukkan atau diupdate, jadi Anda tidak perlu memasukkan nilai apapun ke kolom ini.

Menentukan Sifat Null (*Nullability*) dari Kolom

Sampai pada poin ini, fokus hanya pada identifikasi elemen-elemen yang diperlukan pada pernyataan CREATE TABLE dan definisi kolom. Hasilnya, definisi kolom telah menyertakan hanya nama kolom dan tipe data yang diberikan pada kolom tersebut. Komponen berikut dari definisi kolom yang dibahasa adalah nullabilitas kolom, dimana dispesifikasikan melalui kata kunci NULL dan NOT NULL.
Nullabilitas kolom menunjuk pada kemampuan kolom untuk menerima nilai null. Nilai null mengindikasikan bahwa nilai tersebut tidak ditentukan atau tidak diketahui. Tidak sama dengan nol atau kosong. Ketika Anda menyertakan NOT NULL dalam definisi kolom Anda, Anda mengatakan bahwa kolom tersebut tidak mengijinkan nilai null. Dengan kata lain, sebuah nilai spesifik harus selalu disediakan untuk kolom tersebut. Sedangkan di lain hal, opsi NULL mengijinkan nilai null. Jika tidak ada dari kedua-duanya yang dispesifikasikan, nilai defaultnya adalah NULL, dan nilai null diijinkan berada dalam kolom.
Berikut ini contoh tabel yang menyertakan dua kolom NOT NULL:

```
CREATE TABLE Katalog
(
IDProduk SMALLINT UNSIGNED NOT NULL,
Nama VARCHAR(40) NOT NULL
);
```

Anda harus menyediakan sebuah nilai untuk kedua kolom, IDProduk dan Nama. Kapanpun Anda memasukkan baris-baris dalam tabel atau mengupdate baris dalam tabel, Anda tidak dapat menggunakan NULL sebagai nilai untuk kedua kolom tersebut.
Secara umum, kapanpun Anda mengkonfigurasi sebuah kolom sebagai NOT NLL, Anda harus menyediakan sebuah nilai selain dari NULL ke kolom ketika memasukkan atau memodifikasi baris.

Menentukan Nilai Default

Situasi bisa saja berubah dimana Anda menginginkan sebuah kolom untuk menggunakan nilai default ketika memasukkan atau mengupdate suatu baris, jika tidak ada nilai apapun yang disediakan. Hal ini sangat berguna ketika sebuah nilai seringkali berulang-ulang dalam sebuah kolom atau merupakan nilai yang paling sering digunakan dalam kolom tersebut. MySQL mengijinkan Anda untuk memberikan nilai default melalui penggunaan klausa DEFAULT. Misal, definisi tabel berikut menyertakan sebuah kolom yang didefinisikan dengan nilai default 'Kosong':

```
CREATE TABLE Biografi
(
IDPenulis SMALLINT UNSIGNED NOT NULL,
TahunLahir YEAR NOT NULL,
KotaKelahiran VARCHAR(40) NOT NULL DEFAULT 'Kosong'
);
```

Untuk kolom KotaKelahiran dikonfigurasi dengan tipe data VARCHAR dan opsi NOT NULL. Dengan tambahan, definisi kolom juga menyertakan klausa DEFAULT. Dalam klausa tersebt, kata kunci DEFALT dispesifikasi, diikuti oleh nilai default yang sebenarnya, dimana dalam kasus ini adalah Kosong. Jika Anda memasukkan sebuah baris dalam tabel dan tidak menspesifikasikan suatu nilai pada kolom KotaKelahiran, nilai Kosong secara otomatis dimasukkan ke dalam kolom tersebut.

Anda juga dapat menspesifikasikan nilai default dalam sebuah kolom yang dikonfigurasi dengan sebuah tipe data numerik. Contoh:

```
CREATE TABLE Biografi
(
IDPenulis SMALLINT UNSIGNED NOT NULL,
TahunLahir YEAR NOT NULL,
JumlahBuku SMALLINT NOT NULL DEFAULT 1
);
```

Perhatikan bahwa Anda tidak perlu menutup nilai default tersebut dalam tanda kutip tunggal. Tanda petik hanya digunakan untuk nilai default string.

Anda juga dapat menspesifikasikan NULL sebagai nilai default. Kolom tersebut tentu saja mengijinkn nilai null dikarenakan telah ditentukan sebagai default.

Jika Anda tidak memberikan nilai default ke kolom, MySQL secara otomatis memberikan nilai defaultnya ke kolom. Jika suatu kolom menerima nilai null, nilai defaultnya adalah NULL. Jika kolom tidak menerima nilai null, nilai defaultnya tergantung bagaimanan kolom tersebut didefinisikan:

- Untuk kolom yang dikonfigurasi dengan tipe data TIMESTAMP, nilai default untuk kolom TIMESTAMP pertama adalah tanggal dan waktu sekarang. Nilai default untuk kolom TIMESTAMP yang lain dalam tabel adalah nilai nol di tempat tanggal dan waktu.
- Untuk kolom yang dikonfigurasi dengan tipe data tanggal/waktu selain TIMESTAMP, nilai defaultnya adalah nol di tempat tanggal dan waktu.
- Untuk kolom yang dikonfigurasi dengan opsi AUTO_INCREMENT, nilai defaultnya adalah nilai berikut dalam urutan inkrementasi angka.
- Untuk kolom numerik yang tidak dikonfigurasi dengan opsi AUTO_INCREMENT, nilai defaultnya adalah 0.
- Untuk kolom yang dikonfigurasi dengan tipe data ENUM, nilai default adalah nilai yang pertama dispesifikasikan dalam definisi kolom.
- Untuk kolom yang dikonfigurasi dengan tipe data string selain tipe ENUM, nilai defaultnya adalah string kosong.

Menentukan *Primary Keys* (Kunci Utama)

Primary Key (Kunci Primer/Utama) adalah satu atau lebih kolom dalam tabel yang secara unik mengidentifikasi setiap baris dalam tabel tersebut. Hampir untuk setiap tabel, Anda sebaiknya menentukan kunci primer dari tabel tersebut.
Cara yang paling mudah untuk menentukan kunci primer dari kolom tunggal adalah dengan menspesifikasikan opsi PRIMARY KEY dalam definisi kolom.

Contoh:

```
CREATE TABLE Pembelian
(
IDPembelian SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
IDModel SMALLINT UNSIGNED NOT NULL,
DeskripsiModel VARCHAR(40)
);
```

Dalam definisi tabel ini, kunci primer untuk tabel Pembelian ditentukan pada kolom IDPembelian. Anda hanya perlu menambahkan klausa PRIMARY KEY ke definisi kolom. Selain itu, untuk menentukan suatu kolom sebagai kunci primer, kolom tersebut harus dikonfigurasi sebagai NOT NULL. Jika Anda secara eksplisit tidak menspesifikasikan opsi NOT NULL, maka diasumsikan sebagai NOT NULL. Dengan tambahan, sebuah tabel hanya dapat memiliki satu kunci primer, yang dapat Anda definisikan dalam definisi kolom atau sebagai batasan terpisah, sebagaimana terlihat dalam sintaks berikut:

```
(CONSTRAINT <constraint name>) PRIMARY KEY (<column name> ({, <column name>}...))
```

Saat Anda menentukan sebuah kunci primer sebagai batasan terpisah, Anda menyertakannya sebagai elemen tabel dalam definisi tabel Anda. Contoh:

```
CREATE TABLE Pembelian
(
IDPembelian SMALLINT UNSIGNED NOT NULL,
IDModel SMALLINT UNSIGNED NOT NULL,
DeskripsiModel VARCHAR(40),
PRIMARY KEY (IDPembelian)
);
```

Jika Anda membuat kunci primer pada lebih dari satu kolom, Anda harus menyertakan kedua kolom tersebut dalam tanda kurung, sebagaimana terihat pada definisi tabel berikut:

```
CREATE TABLE Pembelian
(
IDPembelian SMALLINT UNSIGNED NOT NULL,
IDModel SMALLINT UNSIGNED NOT NULL,
DeskripsiModel VARCHAR(40),
PRIMARY KEY (IDPembelian, IDModel)
);
```

Perhatikan bahwa batasan PRIMARY KEY kini menspesifikasikan kolom IDPembelian dan IDModel. Sehingga, kunci primer untuk tabel Pembelian akan dibuat pada dua kolom ini, yang berarti tidak ada 2 pasang nilai yang sama, meskipun nilai-nilai tersebut dapat diulang dalam kolom-kolom secara individual. Kapan saja Anda ingin menentukan kunci primer pada dua atau lebih kolom, Anda harus menggunakan batasan pada level tabel. Anda tidak dapat menentukan kunci primer pada banyak kolom pada level kolom, sebagaimana dapat Anda lakukan saat Anda menyertakan hanya satu kolom dalam kunci primer.

Menentukan Kolom *Auto-Increment*

Dalam beberapa kasus, Anda mungkin ingin membangkitkan angka-angka dalam kunci primer Anda secara otomatis. Misal, setiap kali Anda menambahkan sebuah pembelian ke tabel baru, Anda ingin memberikan nilai baru untuk mengidentifikasi pembelian tersebut. Semakin banyak isi dalam tabel, semakin banyak jumlah atau nomor pembelian di sana. Untuk alasan inilah, MySQL mengijinkan Anda menentukan kolom kunci primer dengan opsi AUTO_INCREMENT. Opsi AUTO_INCREMENT mengijinkan Anda untuk menspesifikasikan angka-angka tersbeut dibangkitkan secara otomatis untuk kolom kunci tamu Anda.
Misal, kolom kunci primer dalam contoh berikut dikonfigurasi dengan opsi AUTO_INCREMENT:

```
CREATE TABLE Katalog
(
IDProduk SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
Nama VARCHAR(40) NOT NULL,
PRIMARY KEY (IDProduk)
);
```

Dalam contoh ini, kolom IDProduk dikonfigurasi dengan tipe data SMALLINT dan dikonfigurasi sebagai kunci primer (lewat penggunaan batasan PRIMARY KEY). Definisi kolom juga menyertakan opsi NOT NULL dan opsi AUTO_INCREMENT. Sehingga, kapanpun Anda menambahkan sebuah baris baru ke tabel Katalog, nomor baru secara otomatis ditambahkan ke kolom IDProduk. Nomor dinaikkan sebanyak 1, berdasarkan pada nilai tertinggi yang terdapat dalam kolom tersebut. Misal, jika dalam sebuah baris terdapat IDProduk dengan nilai 125, dan ini adalah nilai IDProduk yang tertinggi dalam tabel, pada baris berikutnya yang ditambahkan ke dalam tabel akan diberikan nilai IDProduk 126.
Anda dapat menggunakan opsi AUTO_INCREMENT hanya pada kolom yang dikonfigurasi dengan tipe data integer dan opsi NOT NULL. Dengan tambahan, tabel harus di set terdapat kunci primer atau dengan indeks unik, dan di sana hanya boleh satu kolom AUTO_INCREMENT per tabel. Juga, kolom AUTO_INCREMENT tidak dapat didefinisikan dengan suatu nilai default.

Menentukan *Foreign Keys* (Kunci Tamu)

Anda telah mempelajari dalam database relasional bahwa terdapat relasi atau hubungan antara data (relasi) satu dengan data (relasi) lainnya atau untuk membantu memastikan integritas data. Ketika Anda membuat model data, relasi ini ditunjukkan dengan menghubungkan tabel-tabel yang berkaitan dengan garis-garis yang menandakan tipe relasi.
Untuk mengimplementasikan relasi ini dalam MySQL, Anda haru menentukan kunci tamu pada tabel yang berkaitan. Anda menentukan kunci tamu pada satu kolom atau kolom-kolom dalam tabel mengkait dengan kolom atau kolom-kolom pada tabel yang terkait.

Tabel yang mengkait, yaitu tabel yang berisi kunci tamu, seringkali ditunjuk sebagai *child table* (tabel anak), dan tabel terkait seringkali ditunjuk sebagai *parent table* (tabel induk).
Kunci tamu mengelola konsistensi data antara tabel anak dan tabel induk. Sehingga untuk memasukkan sebuah baris dalam tabel anak atau mengupdate baris dalam tabel tersebut, nilai dalam kolom kunci tamu haru ada di kolom terkait dalam tabel induk.
Misal, anggap Anda memiliki tabel untuk mendata penjualan di toko buku. Satu kolom dalam tabel menyimpan ID untuk buku-buku yanng telah dijual. Data tentang buku-buku itu sendiri sebenarnya disimpan dalam tabel terpisah, dan setiap buku diidentifikasi dalam tabel tersebut oleh ID-nya. Sehingga, kolom ID bku dalam tabel penjualan mengkait kolom ID buku dalam tabel buku. Untuk menghubungkan data dalam kedua kolom ini, kolom ID buku dalam tabel penjualan dikonfigurasi sebagai kunci tamu. Dikarenakan kunci tamu, tidak ada ID buku yang dapat ditambahkan ke tabel penjualan yang tidak terdapat dalam tabel buku. Tabel penjualan, untuk kemudian, adalah tabel anak, dan tabel buku adalah tabel induk.

Untuk menambahkan kunci tamu ke sebuah tabel, Anda dapat mendefinisikannya dalam definisi kolom atau sebagai batasan dalam elemen tabel terpisah. Untuk menambahkan kunci tamu dalam definisi kolom, Anda harus menambahkan definisi referensi, dimana terlihat pada sintaks berikut:

```
<definisi referensi>::=
REFERENCES <nama tabel> [(<nama kolom> [{, < nama kolom >}...])]
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT }]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT }]
```

Sebagaimana terlihat pada sintaks, klausa menyertakan beberapa elemen-elemen yang diperlukan: kata kunci REFERENCES, nama dari tabel (induk) terkait, dan minimal satu kolom dalam tabel tersebut, ditutup dalam tanda kurung.
Dalam sintaks juga disertakan klausa opsional ON DELETE dan ON UPDATE. Klausa ON DELETE menspesifikasikan bagaimana MySQL memperlakukan data yang terhubung dalam tabel anak ketika sebuah baris dalam tabel induk dihapus. Klausa ON UPDATE menspesifikasikan bagaimana MySQL memperlakukan data yang terhubung dalam tabel anak ketika sebuah baris dalam tabel induk diupdate. Untuk setiap klausa, lima opsi tersedia. Anda hanya dapat menspesifikasikan satu opsi untuk setiap klausa. Opsi-opsi ini diterangkan dalam tabel berikut:

OPSI	DESKRIPSI
RESTRICT	Jika tabel anak berisi nilai dalam kolom yang mengkait yang nilainya sama dengan di kolom terkait pada tabel induk, baris dalam tabel induk tidak bisa dihapus, dan nilai di kolom terkait tidak dapat diupdate. Ini adalah opsi default jika klausa ON DELETE atau ON UPDATE tidak dispesifikasikan.

CASCADE	Baris-baris dalam tabel anak yang berisi nilai-nilai yang juga terdapat dalam kolom terkait dari tabel induk dihapus ketika baris-baris yang berkaitan dihapus dari tabel induk. Baris-baris dalam tabel anak yang berisi nilai-nilai yang juga terdapat dalam kolom terkait dari tabel induk diupdate ketika nilai-nilai yang berkaitan diupdate dalam tabel induk.
SET NULL	Nilai-nilai dalam kolom yang mengkait dari tabel anak diset ke NULL saat baris-baris dengan data terkait dalam tabel induk dihapus dari tabel induk atau ketika data terkait dalam tabel induk diupdate. Untuk menggunakan opsi ini, semua kolom-kolom yang mengkait dalam tabel anak harus mengijinkan nilai NULL.
NO ACTION	Tidak ada aksi yang diambil dalam tabel anak ketika baris-baris dihapus dari tabel induk atau nilai-nilai dalam kolom terkait dalam tabel induk diupdate.
SET DEFAULT	Nilai-nilai dalam kolom-kolom yang mengkait dari tabel anak diset ke nilai default mereka ketika baris-baris dihapus dari tabel induk atau kolom terkait dari tabel induk diupdate.

Ketika Anda menentkan kunci tamu, Anda dapat menyertakan klausa ON DELETE, ON UPDATE, atau keduanya. Jika Anda menyertakan keduanya, Anda dapat mengkonfigurasinya dengan opsi yang sama atau dengan opsi yang berbeda. Jika Anda tidak menyertakan salah satu atau keduanya, opsi RESTRICT diasumsikan dalam kasus satunya, yang berarti bahwa *update* dan *delete* dibatasi ke baris-baris dengan nilai-nilai yang tidak terkait. Dengan tambahan, ketika menentukan kunci tamu, kolom-kolom yang mengkait harus memiliki tioe data yang kompatibel dengan kolom-kolom terkait. Untuk tipe data integer, kuran dan status *signed*/*unsigned* harus sama. Panjang dari tipe data string, bagaimanapun, tidak harus sama. Secara umum merupakan ide yang bagus untuk mengkonfigurasi kolom-kolom yang mengkait dan yang terkait dengan tipe data yang sama dan opsi-opsi yang berkaitan yang sama pula.
Contoh:

```
CREATE TABLE Pembelian
(
IDPembelian SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
IDModel SMALLINT UNSIGNED NOT NULL REFERENCES Model (IDModel),
DeskripsiModel VARCHAR(40)
);
```

Dalam contoh ini, kolom IDModel dikonfigurasi dengan tipe data SMALLINT (*unsigned*), opsi NOT NULL, dan klausa REFERENCES, yang menspesifikasikan nama dari tabel induk (Model) dan nama dari kolom terkait (IDModel) dalam tabel induk. Sehingga, kolom IDModel dari tabel Pembelian dapat menyertakan hanya nilai-nilai yang terdapat dalam kolom IDModel dari tabel Model.
Anda juga dapat menentkan kunci tamu ini sebagai elemen tabel terpisah dengan menambahkan batasan FOREIGN KEY ke definisi tabel Anda. Berikut ini sintaksnya:

```
(CONSTRAINT <constraint name>) FOREIGN KEY (<index name>)
                (<column name> ({, <column name>}...)) <reference definition>
```

Dalam sintaks tersebut, Anda harus menyertakan kata kunci FOREIGN KEY, nama dari kolom yang mengkait dalam tabel anak, tertutup dalam tanda kurung, dan definisi referensi.
Contoh:

```
CREATE TABLE Pembelian
(
IDPembelian SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
IDModel SMALLINT UNSIGNED NOT NULL,
DeskripsiModel VARCHAR(40)
FOREIGN KEY (IDModel) REFERENCES Model (IDModel)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Dalam contoh ini, batasan kunci tamu ditambahkan sebagai elemen tabel, bersamaan dengan definisi kolom.
Kolom yang sama (IDModel) dikonfigurasi sebagai kunci tamu yang mereferensi kolom IDModel dari tabel Model. Perbedaan antara comtoh ini dan contoh sebelumnya hanyalah bahwa pada definisi referensi pada contoh terakhir menyertakan klausa ON DELETE dan ON UPDATE, keduanya dikonfigurasi dengan opsi CASCADE. Sehingga, tabel anak mencerminkan perubahan ke tabel induk.

Jika Anda ingin menentukan kunci tamu pada lebih dari satu kolom, Anda harus menggunakan batasan FOREIGN KEY, daripada menambahkan definisi referensi ke definisi kolom. Dengan tambahan, Anda harus memisahkan nama-nama kolom dengan koma dan tertutup semuanya dalam tanda kurung.

Menentukan Tipe Tabel

Salah satu opsi tabel yang cukup penting ketika belajar mengenai MySQL adalah salah satunya yang mengijinkan Anda menentukan tipe tabel yang Anda buat dalam definisi tabel. MySQL mengijinkan Anda membuat enam tipe tabel yang berbeda, sintaksnya adalah:

```
ENGINE = {BDB | MEMORY | ISAM | INNODB | MERGE | MYISAM}
```

Untuk menentukan tipe tabel, Anda harus menyertakan sebuah klausa ENGINE di akhir definisi tabel Anda, setelah tanda kurung yang menutup elemen-elemen tabel Anda. Misal, definisi tabel berikut ini menspesifikasikan tipe tabel InnoDB:

```
CREATE TABLE Biografi
(
IDPenulis SMALLINT UNSIGNED NOT NULL,
TahunLahir YEAR NOT NULL,
KotaLahir VARCHAR(40) NOT NULL DEFAULT 'Kosong'
)
ENGINE=INNODB;
```

Dalam definisi ini, sebuah klausa ENGINE ditambahkan setelah definisi kolom terakhir dan tanda kurung tutup. Perhatikan bahwa Anda dengan mudah menspesifikasikan kata kunci ENGINE, tanda sama dengan, dan satu dari enam tipe tabel.
Setiap tipe tabel dalam MySQL mendukung kumpulan fungsionalitas spesifik dan melayani tujuan spesifik. Dengan tambahan, setiap tabel dihubungkan dengan *engine* penyimpanan (*handler*) yang bersangkutan yang memproses data dalam tabel. Misalnya, *engine* MyISAM memproses data dalam tabel MyISAM. Berikut ini mendiskusikan masing-masing dari enam tipe tabel.

Table type	Description
BDB	A transaction-safe table that is managed by the Berkeley DB (BDB) handler. The BDB handler also supports automatic recovery and page-level locking. The BDB handler does not work on all the operating systems on which MySQL can operate. For the most part, InnoDB tables have replaced BDB tables.
MEMORY	A table whose contents are stored in memory. The data stored in the tables is available only as long as the MySQL server is available. If the server crashes or is shut down, the data disappears. Because these types of tables are stored in memory, they are very fast and are good candidates for temporary tables. MEMORY tables can also be referred to as HEAP tables, although MEMORY is now the preferable keyword.
InnoDB	A transaction-safe table that is managed by the InnoDB handler. As a result, data is not stored in a .MYD file, but instead is managed in the InnoDB tablespace. InnoDB tables also support full foreign key functionality in MySQL, unlike other tables. In addition, the InnoDB handler supports automatic recovery and row-level locking. InnoDB tables do not perform as well as MyISAM tables.
ISAM	A deprecated table type that was once the default table type in MySQL. The MyISAM table type has replaced it, although it is still supported for backward compatibility. Eventually, ISAM tables will no longer be supported.
MERGE	A virtual table that is made up of identical MyISAM tables. Data is not stored in the MERGE table, but in the underlying MyISAM tables. Changes made to the MERGE table definition do not affect the underlying MyISAM tables. MERGE tables can also be referred to as MRG_MyISAM tables
MyISAM	The default table type in MySQL. MyISAM tables, which are based on and have replaced ISAM tables, support extensive indexing and are optimized for compression and speed. Unlike other table types, BLOB and TEXT columns can be indexed and null values are allowed in indexed columns. MyISAM tables are not transaction safe, and they do not support full foreign key functionality.

LATIHAN
Desain Database Penyewaan DVD

Model data yang Anda desain di sini adalah membuat toko fiksi yang menyewakan DVD ke pelanggannya. Database mencatat inventori DVD, menyediakan informasi mengenai DVD, menyimpan transaksi penyewaan, dan menyimpan nama-nama dari pelanggan dan pekerja. Untuk membuat model data, Anda perlu menggunakan peraturan bisnis berikut:

- Database menyimpan informasi mengenai ketersediaan DVD untuk disewa. Setiap DVD, informasinya meliputi nama DVD, jumlah disk dalam satu set, tahun DVD di-*release*, tipe film (misal:aksi), studio yang memiliki hak penerbitan DVD (misal: Colmbia Pictures), rating film (PG dan seterusnya), format DVD (misal: *Widescreen*), dan status ketersediaan (Keluar). Banyak salinan DVD yang sama diperlakukan sebagai produk individu.
- Database seharusnya menyimpan nama-nama aktor, sutradara, produser, eksekutif produser, ko-produser, asisten produser, penulis, dan komposer yang berpartisipasi dalam pembuatan film yang tersedia untuk disewa. Informasi mencakup nama lengkap pemain dan peranan yang dimainkan dalam pembuatan film.
- Database seharusnya meliputi nama lengkap pelanggan yang menyewa DVD dan para pekerja yang bekerja di toko tersebut. Data rekod pelanggan harus dibedakan dengan data rekod pekerja.
- Database seharusnya mencakup informasi mengenai setiap transaksi penyewaan DVD. Informasi meliputi siapa saja pelanggan yang menyewa DVD, pekerja yang menangani transaksi, DVD yang disewa, tanggal penyewaan, tanggal DVD itu harus kembali, dan tanggal DVD tersebut telah benar-benar kembali. Setiap penyewaan DVD harus disimpan sebagai transaksi individu.
  Setiap transaksi merupakan bagian dari tepat satu order. Satu atau lebih transaksi diperlakukan sebagai order tunggal di bawah kondisi berikut: [1] transaksi (satu atau banyak) adalah untuk pelanggan tunggal meminjam satu atau lebih DVD pada waktu yang sama dan [2] transaksi (satu atau banyak) dijalankan oleh satu pekerja.

Peraturan bisnis yang disediakan di sini bukan berarti telah memakai semua spesifikasi yang dibutuhkan untuk membuat sebuah database. Namun peraturan bisnis yang Anda lihat di sini, bagaimanapun, sudah cukup untuk Anda memulai membuat model data.

Langkah-langkah yang diperlukan untuk membuat database secara sederhana adalah sebagai berikut:

- Normalisasi data menurut bentuk normal satu sampai tiga.
- Menentukan relasi antar tabel.
- Melakukan langkah-langkah yang perlu untuk membuat model data, termasuk mengidentifikasi entitas potensial untuk disertakan dalam database, menormalisasi desain data, mengidentifikasi relasi antar tabel, dan memperbaiki atau menghaluskan model data.

SELAMAT MENCOBA!!!