CIS 363 MySQL

Chapter 4 MySQL Connectors

Chapter 5 Data Types

Chapter 6 Identifiers

Ch. 4 MySQL Connectors

***** Chapter 3 MySQL Query Browser is Not covered on MySQL certification exam. For details regarding the exam, please check**

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=332

Ch. 4 MySQL Connectors:

- MySQL AB provides several applications programming interfaces (API) for accessing the MySQL server. The interface included with distributions of MySQL itself is libmysqlclient, the C client library.
- MySQL AB also provides Drivers that act as bridge to the MySQL server for client programs that communicating by using a particular protocol.
- MySQL connectors are shipped separately from MySQL server distributions by MySQL AB. (CD Practice Question 2). It means the MySQL download doesn't include any MySQL connectors. You have to download each connector separately from

  http://www.mysql.com/downloads/.

Ch. 4 MySQL Connectors

- The MySQL connectors are available for Windows and Unix. To use a connector, you must install it on the client host. MySQL connectors are useful for providing MySQL connectivity in host heterogeneous environment. (different operating systems)
- In addition to MySQL connector/ODBC, MySQL connector/J, and MySQL connector/Net, many third-party clients interfaces are supported by MySQL AB. Most of them are based on the C Client library and provide a binding for some other language. These include the mysql and mysqli extensions for PHP, the DBD::mysql driver for the Perl DBI module, and interfaces for other languages such as Python, Ruby, Pascal, and Tcl.

Ch. 4 MySQL Connectors

MySQL Connectors

Connector Type	Based on	Connection	Operating System
MySQL Connector/ODBC	C client Library	1)TCP/IP 2)Unix socket files 3)Name pipes	Windows & Unix
MySQL Connector/J	Java	1)TCP/IP 2)Name pipes	Windows & Unix
MySQL Connector/NET	C#	1)TCP/IP 2)Unix socket files 3)Name pipes 4)shared memory	Windows (Linux)

Note: If you use Mono, the Open source implement of .Net developed by Novell, MySQL Connector/Net is also on Linux p.57

Ch. 5 Data Types

Date values can be grouped into

1) Numeric values:

- Integer data types-INT
- Floating-point data types-FLOAT, DOUBLE
- Fixed-point data types-DECIMAL
- The *BIT* data type

2) String values

- Binary – raw bytes
- Non-binary –characters –character set &collation –should be quoted

3) Temporal values

- Dates
- Times

4) Spatial values (Not covered in the textbook)

Ch. 5 Data Types

- INT -- TINYINT, SMALLINT, MEDIUMINT, BIGINT
- DECIMAL (p,s) -- Fixed point data types. AKA: NUMERIC
- FLOAT - 32 bit / 4 bytes floating point
- DOUBLE - 64 bit / 8 bytes floating point
- BIT(n) - n bit data type, precisely how many bits to store.
- CHAR -- Fixed length, non binary string.
- VARCHAR -- Variable length non binary
- TEXT -- Variable length non binary
- BINARY -- Fixed length binary
- VARBINARY -- Variable length binary
- BLOB -- Variable length binary string
- ENUM -- Enumeration fixed set of legal values. One value per entry.
- SET -- Set consisting of a fixed set of legal values. Multiple values per entry.
- DATE(date+time) -- (DATE, TIME, DATETIME, TIMESTAMP, YEAR)

Ch. 5 Data Types

```
mysql> CREATE TABLE people (
id INT,
first_name CHAR(30),
last_name CHAR(30)
);
```

```
mysql>CREATE TABLE people (
id INT UNSIGNED NOT NULL,
first_name CHAR(30),
last_name CHAR(30)
);
```

- **To disallow negative values in the id column, use the UNSUGNED attribute.**
- **To disallow missing or unknown values in the id columns, use NOT NULL attribute.**

Ch. 5 Data Types

5.2 Numeric Data Types

For storing numeric data, MySQL provides integer data types, floating-point types that store approximate-value numbers, a fixed-point type that stores exact-value numbers, and a `BIT` type for bit-field values. When you choose a numeric data type, consider the following factors:

- The range of values the data type represents
- The amount of storage space that column values require
- The display width indicating the maximum number of characters to use when presenting column values in query output
- The column precision and scale for floating-point and fixed-point values

Precision and scale are terms that apply to floating-point and fixed-point values, which can have both an integer part and a fractional part. Precision is the number of significant digits. Scale is the number of digits to the right of the decimal point.

Ch. 5 Data Types

5.2.1 Integer Data Types

Integer data types include `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`. Smaller integer types require less storage space, but are more limited in the range of values they represent. For example, `TINYINT` column values take only one byte each to store, but the type has a small range (–128 to 127). `INT` column values require four bytes each, but the type has a much larger range (–2,147,483,648 to 2,147,483,647). The integer data types are summarized in the following table, which indicates the amount of storage per value that each type requires as well as its range. For integer values declared with the `UNSIGNED` attribute, negative values are not allowed, and the high end of the range shifts upward to approximately double the maximum positive value of the signed range.

Ch. 5 Data Types

Type	Storage Required	Signed Range	Unsigned Range
TINYINT	1 byte	-128 to 127	0 to 255
SMALLINT	2 bytes	-32,768 to 32,767	0 to 65,535
MEDIUMINT	3 bytes	-8,388,608 to 8,388,607	0 to 16,777,215
INT	4 bytes	-2,147,683,648 to 2,147,483,647	0 to 4,294,967,295
BIGINT	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073, 709,551,615

Ch. 5 Data Types

- Century INT(4) p.62
  - The result is that values in the century column are displayed 4 digits wide.
  - Display width is unrelated to the range of data type. If you inserted the `57622` into the century column, MySQL display the entire value, not just the first 4 digits of the value.
  - The display width for integer types also is unrelated to storage requirement. INT(4) and INT(8) require 4 bytes, same storage per value.

- If no display width is specified for an integer type, MySQL chooses a default based on the number of characters needed to displayed the full range of values for the type (including the minus sign, for signed types).
  - SMALLINT has a default display width of 6 because the widest possible value is -32768.

Ch. 5 Data Types

Floating-Point Data Types:

- The Floating-point data types include **FLOAT** and **DOUBLE.**
- FLOAT and DOUBLE data types represent value in the native binary floating-point format used by the server host's CPU.
- They are very efficient type for storage and computation.
- Values are subject to rounding error.
- Floating-point values are stored using mantissa/exponent representation. The precision is defined by the width of the mantissa and the scale varies depending on the exponent value.

Data Type	Precision and Scale values in the column definition	Storage
weight FLOAT (10,3)	**A single-precision column with a precision of 10 digits and scale of 3 decimals**	**4 bytes**
score DOUBLE (20,7)	**A double-precision column with a precision of 20 digits and scale of 7 decimals**	**8 bytes**

Ch. 5 Data Types

Fixed-Point Data Types:

- The Floating-point data type is **DECIMAL**
- DECIMAL represents exact-value numbers and has a fixed-decimal storage format.
- Values are ***NOT*** subject to rounding error. More accurate and good for financial applications.
- The amount of storage required for DECIMAL values depends on the precision and scale, approximately 4 bytes are required per nine digits on each side of the decimal point.

Ch. 5 Data Types

The BIT Data Type: p.64

- The Bit data type represents bit-field values.
- BIT column specifications take a width indicating the number of bits per value, from 1 to 64 bits.
  - BIT_col1 BIT(4) → store 4 **bits** per value
  - BIT_col2 BIT(20) → store 20 **bits** per value
- For a BIT(n) column, the range of values is 0 to $2^n - 1$
- For a BIT(n) column, the storage requirement is approximately INT((n+7)/8) **bytes** per value.
  - For BIT_col BIT(3), the storage requirement is 1 byte. The range is 0 to 7.
- BIT column can be assigned values using numeric expressions. To write literal bit values in binary format, the literal-value notation b'val' can be used, where val indicates a value consisting of the binary digits 0 to 1.
  - b'1111' equals 15
  - b'100000' equals 64

Ch. 5 Data Types

String Data Types

The following table summarizes the non-binary string data types. For the storage requirement values, *M* represents the maximum length of a column. *L* represents the actual length of a given value, which may be 0 to *M*.

Type	Storage Required	Maximum Length
CHAR(*M*)	*M* characters	255 characters
VARCHAR(*M*)	*L* characters plus 1 or 2 bytes	65,535 characters (subject to limitations)
TINYTEXT	*L* characters + 1 byte	255 characters
TEXT	*L* characters + 2 bytes	65,535 characters
MEDIUMTEXT	*L* characters + 3 bytes	16,777,215 characters
LONGTEXT	*L* characters + 4 bytes	4,294,967,295 characters

Ch. 5 Data Types

- Non-binary String Data Type: **CHAR**, **VARCHAR**, and **TEXT** are data types that store non-binary strings.
  - Non-Binary strings have a character set and collation, and non-binary columns by default are assigned the character set and collation of the table that contains them.
- Binary String Data Type: **BINARY**, **VARBINARY** and **BLOB** are data types that store binary strings.
  - Binary strings consist simply of bytes that are distinguished only by their numeric values.

- The most general difference non-binary and binary strings is that non-binary strings have a character set and consist of characters in that character set, whereas binary strings consist simply of bytes that are distinguished only by their numeric values.

Ch. 5 Data Types

Character Set Support: p.65

- Non-binary string is a sequence of characters that belong to a specific character set. MySQL's default character is latin1. The latin1 character set uses 1 byte per character.
- Multi-byte character sets may require a fixed or variable number of bytes per character. The ucs2 Unicode character set uses 2 bytes per character, whereas the utf8 Unicode character set uses from 1 to 3 bytes character.
- Non-binary string comparisons are based on the collation of the character set associated with the string. A given character set may have one or more collations, but a given string has only one of those collations.
- The collation determines whether uppercase and lowercase versions of a given character are equivalent.
- The collation also determines whether to treat instances of a given character with different accent marks as equivalent.
- A collation can be a binary collation. In this case, comparisons are based on numeric character values. Comparisons for binary strings are always byte-based.

Ch. 5 Data Types

mysql> SHOW COLLATION LIKE 'latin1%';

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5		Yes	1
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15		Yes	1
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48		Yes	1
latin1_general_cs	latin1	49		Yes	1
latin1_spanish_ci	latin1	94		Yes	1

```
_ci      = case insensitive
_cs      = case sensitive
_bin     = binary
```

Ch. 5 Data Types

Characteristics of binary strings:

- A binary string is treated as a sequence of byte values. It can appear to contain character as a quoted string.
- Binary strings contain bytes, not characters, comparisons of binary strings are performed on the basis of the byte values in the string.
- Binary strings appear to be case sensitive, because uppercase and lowercase versions of a given character have different numeric byte values.
- A binary string appear to be accent sensitive.
- A multi byte character, if stored in a binary string, is treated simply as multiple individual bytes. Character boundaries of the original data no longer apply.

Ch. 5 Data Types

The difference in how non-binary and binary string are treated with respect to lettercase

```
mysql> SELECT UPPER('AaBb'), UPPER(BINARY 'AaBa');
+----------------+-----------------------+
| UPPER('AaBb') | UPPER(BINARY 'AaBa') |
+----------------+-----------------------+
| AABB           | AaBa                  |
+----------------+-----------------------+
```

A way to set up character sets and collations:

mysql> CREATE TABLE auth_info (
login CHAR(32) CHARACTER SET LATIN1,
password CHAR(32) ***CHARACTER SET LATIN1 COLLATE latin1_general_cs***,
picture MEDIUMBLOB,
);

Ch. 5 Data Types

- **Non-Binary String Data Types: CHAR, VARCHAR, TEXT**

The following table summarizes the non-binary string data types. For the storage requirement values, *M* represents the maximum length of a column. *L* represents the actual length of a given value, which may be 0 to *M*.

Type	Storage Required	Maximum Length
CHAR(*M*)	*M* characters	255 characters
VARCHAR(*M*)	*L* characters plus 1 or 2 bytes	65,535 characters (subject to limitations)
TINYTEXT	*L* characters + 1 byte	255 characters
TEXT	*L* characters + 2 bytes	65,535 characters
MEDIUMTEXT	*L* characters + 3 bytes	16,777,215 characters
LONGTEXT	*L* characters + 4 bytes	4,294,967,295 characters

Ch. 5 Data Types

- **Binary String Data Types: BINARY, VARBINARY, BLOB**
  - Binary string data types store strings that consist of bytes, and they have no character set or collation.

Type	Storage Required	Maximum Length
BINARY(*M*)	*M* bytes	255 bytes
VARBINARY(*M*)	*L* bytes plus 1 or 2 bytes	65,535 bytes (subject to limitations)
TINYBLOB	*L* + 1 bytes	255 bytes
BLOB	*L* + 2 bytes	65,535 bytes
MEDIUMBLOB	*L* + 3 bytes	16,777,215 bytes
LONGBLOB	*L* + 4 bytes	4,294,967,295 bytes

Ch. 5 Data Types

The ENUM and SET Data Types:

- The ENUM and SET string data types are used when the values to be store in a column are chosen from a fixed set of values.
- ENUM is an enumeration type. An ENUM column definition includes a list of allowable values. Each value in the list is called a “member”.
- An ENUM column definition may list up to 65535 members.
- Enumerations with up to 255 members require 1 byte of storage per value. Enumerations with 256 to 65535 members require 2 bytes per value.

```
mysql> CREATE TABLE booleans (
yesno       ENUM (‘Y’, ‘N’),
truefalse ENUM (‘T,’ ‘F’));

mysql> CREATE TABLE booleans (
yesno       ENUM (‘yes`, ‘no’),
truefalse ENUM (‘true,’ ‘false’));
```

Ch. 5 Data Types

```
mysql> CREATE TABLE countries (
name CHAR(30),
continent ENUM('Asia', 'Europe', 'North America', 'Africa',
'Oceania', 'Antarctica', 'South America'));

mysql> INSERT INTO countries VALUES (‘Kenya`, `Africa’);
mysql> SET @@SQL_MODE=‘’;   (note: clear SQL mode)
mysql> INSERT INTO countries VALUES (‘San Mateo`, `USA’);
```

- Internally, MySQL stores the strings as integers, using the values 1 to n for a column with n enumeration members.
- MySQL reserves the internal values 0 as an implicit member of all ENUM columns. It ‘s used to represent illegal values assigned to an enumeration column. For example, if you assign ‘USA’ to the continent column, MySQL will store the value 0. IF you select to column later, MySQL display 0 values as the empty string **‘’;** (In strict SQL mode, an error occurs if you try to store an illegal ENUM value)

Ch. 5 Data Types

- The **SET** data type, like **ENUM**, is declared using a comma-separated list of quoted strings that define its valid members. Unlike ENUM, set may have any combination of valid values.

```
mysql> CREATE TABLE allergy (symptom SET( 'sneezing', 'runny
    nose', 'stuffy head', 'red eyes'));
```

- MySQL stores SET columns as a bitmap using one bit per member: 1, 2, 4, and 8. Again 0 is reserved for invalid entry.
- INSERT INTO allergy (symptom) VALUES (''); -- 0 bit
  INSERT INTO allergy (symptom) VALUES ('stuffy head'); -- 4 bits
  INSERT INTO allergy (symptom) VALUES ('sneezing red eyes'); -- bit 1 + bit 8 = 9 bits
- A **SET** may contain up to 64 members. Invalid entries are generally ignored or may cause an error depending on the SQL mode you are running in

Ch. 5 Data Types

```
mysql> CREATE TALBE t (
age INT, siblings ENUM('0', '1', '2', '3', '>3')
);

mysql> INSERT INTO t (age, siblings) VALUES(14,'3');
mysql> INSERT INTO t (age, siblings) VALUES(14,3);
```

NOTE: The above two are not the same. The number 3 is an index pointing to value '2‘

```
mysql> SELECT * FROM t WHERE siblings = '3';
mysql> SELECT * FROM t WHERE siblings = 3;
```

Ch. 5 Data Types

Temporal Data Types

MySQL provides data types for storing different kinds of temporal information. In the following descriptions, the terms *YYYY*, *MM*, *DD*, *hh*, *mm*, and *ss* stand for a year, month, day of month, hour, minute, and second value, respectively.

The following table summarizes the storage requirements and ranges for the date and time data types.

Type	Storage Required	Range
DATE	3 bytes	'1000-01-01' to '9999-12-31'
TIME	3 bytes	'-838:59:59' to '838:59:59'
DATETIME	8 bytes	'1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP	4 bytes	'1970-01-01 00:00:00' to mid-year 2037
YEAR	1 byte	1901 to 2155 (for YEAR(4)), 1970 to 2069 (for YEAR(2))

Ch. 5 Data Types

Each temporal data type also has a "zero" value that's used when you attempt to store an illegal value. The "zero" value is represented in a format appropriate for the type (such as `'0000-00-00'` for `DATE` values and `'00:00:00'` for `TIME`) values.

MySQL represents date values in *`'YYYY-MM-DD'`* format when it displays them. This representation corresponds to the ANSI SQL date format, also known as ISO 8601 format. If necessary, you can reformat date values into other display formats using the `DATE_FORMAT()` function.

```
mysql> select now() as 'Old';
+---------------------+
| Old                 |
+---------------------+
| 2011-01-27 21:05:41 |
+---------------------+
1 row in set (0.00 sec)
```

```
mysql> select DATE_FORMAT(now(), '%m/%d/%y') as 'New';
+----------+
| New      |
+----------+
| 01/27/11 |
+----------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

The TIMESTAMP Data Type

- TIMESTAMP data type. Similar to DATETIME but shorter range 1970-01-01 to mid 2037.
- MySQL Server stores TIMESTAMP values internally in UTC. (Coordinated Universal Time). It coverts TIMESTAMP values from the server's current time zones for storage, and converts back to the current time zone for retrieval.
- Three time zone formats:

1. Signed hour/minute offset, i.e. '***+hh:mm***' or '***-hh:mm***'
2. Named time zone, i.e. US/Eastern, US/Pacific, etc.
3. SYSTEM time zone, the time zone of the host server

Ch. 5 Data Types

```
mysql> CREATE TABLE ts_test1 (
ts1 TIMESTAMP,
ts2 TIMESTAMP,
data CHAR(30) );
```

```
mysql> DESCRIBE ts_test1;
+-------+-----------+------+-----+---------------------+-------+
| Field | Type      | Null | Key | Default             | Extra |
+-------+-----------+------+-----+---------------------+-------+
| ts1   | timestamp | NO   |     | CURRENT_TIMESTAMP   |       |
| ts2   | timestamp | NO   |     | 0000-00-00 00:00:00 |       |
| data  | char(30)  | YES  |     | NULL                |       |
+-------+-----------+------+-----+---------------------+-------+
```

Ch. 5 Data Types

mysql> INSERT INTO ts_test1 (data) VALUES ('original value');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test1;

```
+---------------------+---------------------+----------------+
| ts1                 | ts2                 | data           |
+---------------------+---------------------+----------------+
| 2008-02-04 14:20:58 | 0000-00-00 00:00:00 | original value |
+---------------------+---------------------+----------------+
1 row in set (0.00 sec)
```

mysql> UPDATE ts_test1 SET data='updated_value';

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM ts_test1;

```
+---------------------+---------------------+---------------+
| ts1                 | ts2                 | data          |
+---------------------+---------------------+---------------+
| 2008-02-04 14:21:51 | 0000-00-00 00:00:00 | updated_value |
+---------------------+---------------------+---------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

mysql>CREATE TABLE ts_test2 (
created_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
data CHAR(30));

mysql> INSERT INTO ts_test2 (data) VALUES ('original_value');

Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test2;

```
+---------------------+----------------+
| created_time        | data           |
+---------------------+----------------+
| 2008-02-04 14:23:18 | original_value |
+---------------------+----------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

mysql> UPDATE ts_test2 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM ts_test2;

```
+---------------------+---------------+
| created_time        | data          |
+---------------------+---------------+
| 2008-02-04 14:23:18 | updated_value |
+---------------------+---------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

```
CREATE TABLE ts_test3 (
updated_time TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
data CHAR(30) );
```

```
mysql> INSERT INTO ts_test3 (data) VALUES ('original_value');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM ts_test3;
+---------------------+----------------+
| updated_time        | data           |
+---------------------+----------------+
| 0000-00-00 00:00:00 | original_value |
+---------------------+----------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

mysql> UPDATE ts_test3 SET data='updated_value';

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM ts_test3;

```
+---------------------+---------------+
| updated_time        | data          |
+---------------------+---------------+
| 2008-02-04 14:25:54 | updated_value |
+---------------------+---------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

mysql> **CREATE TABLE ts_test4 (**
created TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
data CHAR(30));

ERROR 1293 (HY000): Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDAT E clause

- An error will occur if you use DEFAULT CURRENT_TIMESTAMP with one column and ON UPDATE CURRENT_TIMESTAMP with another. To resolve this error, use the follow syntax

mysql>CREATE TABLE ts_test5 (
created TIMESTAMP DEFAULT 0,
updated TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
data CHAR(30));

Ch. 5 Data Types

mysql> INSERT INTO ts_test5 (created, data) VALUES (null, 'original_value');

Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test5;

```
+---------------------+---------------------+----------------+
| created             | updated             | data           |
+---------------------+---------------------+----------------+
| 2008-02-04 14:29:10 | 0000-00-00 00:00:00 | original_value |
+---------------------+---------------------+----------------+
```

mysql> UPDATE ts_test5 SET data='updated_value';

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM ts_test5;

```
+---------------------+---------------------+---------------+
| created             | updated             | data          |
+---------------------+---------------------+---------------+
| 2008-02-04 14:29:10 | 2008-02-04 14:29:48 | updated_value |
+---------------------+---------------------+---------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

Per-Connection Time Zone Support

- **There are three time zone formats available to use with MySQL.**
  - The signed hour/minute offset of a time zone is expressed as '**+hh:mm**' or '**-hh:mm**'.
  - The name time zone for a given location is defined by a string such as 'US/Eastern', which is translated into the correct time zone by the server.
  - System time zone: this stands for the time zone value that the MySQL server retrieves from the server host.
- **Time zone settings are determined by the time_zone variable. The server maintain a global time_zone value. A session time_zone value for each client that connects.**

Ch. 5 Data Types

The default setting for the global value is SYSTEM, which thus also becomes each client's initial session time_zone value.

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
+--------------------+---------------------+
| @@global.time_zone | @@session.time_zone |
+--------------------+---------------------+
| SYSTEM             | SYSTEM              |
+--------------------+---------------------+
1 row in set (0.00 sec)
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @@session.time_zone;
+---------------------+
| @@session.time_zone |
+---------------------+
| +00:00              |
+---------------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

mysql> CREATE TABLE ts_test (ts TIMESTAMP);

Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO ts_test (ts) VALUES (NULL);

Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test;

```
+---------------------+
| ts                  |
+---------------------+
| 2008-02-05 03:22:11 |
+---------------------+
1 row in set (0.00 sec)
```

mysql> SET time_zone = '+02:00';

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM ts_test;

```
+---------------------+
| ts                  |
+---------------------+
| 2008-02-05 05:22:11 |
+---------------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

```
mysql> SET time_zone = '-02:00';
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM ts_test;
+---------------------+
| ts                  |
+---------------------+
| 2008-02-05 01:22:11 |
+---------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CONVERT_TZ('2008-01-28 13:30:00', '+01:00', '+03:00');
+-----------------------------------------------------------+
| CONVERT_TZ('2008-01-28 13:30:00', '+01:00', '+03:00') |
+-----------------------------------------------------------+
| 2008-01-28 15:30:00                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

Ch. 5 Data Types

Column Attributes

```
mysql> CREATE TABLE t (
i INT UNSIGNED NOT NULL,
c CHAR(10) CHARACTER SET utf8,
d DATE DEFAULT '2011-01-01' );
```

- Numeric Column Attributes
  - UNSIGNED: negative values is NOT allowed.
  - ZEROFILL: left-padded with leading zeros.
  - AUTO_INCREMENT: applies to integer data types. The column with AUTO_INCREMENT attribute must be **indexed** and be defined as **NOT NULL**
- String Column Attributes
  - Character sets and Collation: applies for non-binary strings data types.
- General Column Attributes
  - **Null** and **not Null**
  - **Default** value

Ch. 5 Data Types

- **Default can be used with all data type with the exception of TEXT and BLOB columns, or integer columns that have the AUTO_INCREMENT attribute.**
- **A DEFAULT value must be a constant, not an expression whose value is calculated at record-creation time.**
- **It is an error to specify a default value of NULL for a NOT NULL columns**
- **It is an error to specify a default value that is out of range for the data type.**
- **Implicit default value are defined as follows:**

  - For numeric columns, the default is zero.
  - For string columns other than `ENUM`, the default is the empty string. For `ENUM` columns, the default is the first enumeration member.
  - For temporal columns, the default value is the "zero" value for the data type, represented in whatever format is appropriate to the type (for example, `'0000-00-00'` for `DATE` and `'00:00:00'` for `TIME`). For `TIMESTAMP`, the implicit default is the current timestamp if the column is defined to be automatically initialized, or the "zero" value otherwise.

Ch. 5 Data Types

AUTO_INCREMENT Attribute

- The AUTO_INCREMENT attribute is used in conjunction with an index. (usually a primary key)
- MySQL provides a **LAST_INSERT_ID()** function that returns the most recent generated AUTO_INCREMENT value.
- The column must have an integer data type when you create an AUTO_INCREMENT column.
- AN AUTO_INCREMENT sequence contains only positive values. (UNSIGNED)
- An AUTO_INCREMENT column must be NOT NULL.

Ch. 5 Data Types

```
mysql> CREATE TABLE attendee (
att_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
att_name CHAR(100),
att_title CHAR(40),
PRIMARY KEY (att_id) );

mysql> CREATE TABLE seminar (
att_id INT UNSIGNED NOT NULL,
sem_title ENUM('Database Design', 'Query Optimization', 'SQL Standards', 'Using
    Replication'),
INDEX (att_id) );

mysql> INSERT INTO attendee (att_name, att_title) VALUES ('Charles Loviness', 'IT
    Manager');
mysql> SELECT * FROM attendee WHERE att_id = LAST_INSERT_ID();
```

```
+--------+------------------+------------+
| att_id | att_name         | att_title  |
+--------+------------------+------------+
|      1 | Charles Loviness | IT Manager |
+--------+------------------+------------+
1 row in set (0.01 sec)
```

Ch. 5 Data Types

```
mysql> INSERT INTO seminar (att_id, sem_title) VALUES (LAST_INSERT_ID(), 'Database Design');
mysql> INSERT INTO seminar (att_id, sem_title) VALUES (LAST_INSERT_ID(), 'SQL Standards');
mysql> INSERT INTO seminar (att_id, sem_title) VALUES (LAST_INSERT_ID(), 'Using Replication');
```

```
mysql> SELECT * FROM seminar WHERE att_id = LAST_INSERT_ID();
```

```
+--------+-------------------+
| att_id | sem_title         |
+--------+-------------------+
|      1 | Database Design   |
|      1 | SQL Standards     |
|      1 | Using Replication |
+--------+-------------------+
3 rows in set (0.00 sec)
```

Ch. 5 Data Types

AUTO_INCREMENT must be NOT NULL, the two below are equivalent:
mysql> INSERT INTO t (id, name) VALUES (NULL, 'Hans');
mysql>INSERT INTO t (name) VALUES ('Hans');

You can however explicitly enter a number.
mysql> CREATE TABLE t (id INT AUTO_INCREMENT, PRIMARY KEY (id));
mysql>INSERT INTO t (id) VALUES (NULL), (17), (NULL), (NULL);
mysql> SELECT id FROM t;

```
+----+
| id |
+----+
|  1 |
| 17 |
| 18 |
| 19 |
+----+
4 rows in set (0.00 sec)
```

Ch. 5 Data Types

Compound or Composit keys

```
mysql> CREATE TABLE multisequence (
name CHAR(10) NOT NULL,
name_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (name, name_id) );
```

```
mysql> INSERT INTO multisequence (name) VALUES ('Petr'), ('Ilya'), ('Yuri'), ('Ilya'), ('Petr');
Query OK, 5 rows affected (0.00 sec)

Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM multisequence ORDER BY name, name_id;
+------+---------+
| name | name_id |
+------+---------+
| Ilya |       1 |
| Ilya |       2 |
| Petr |       1 |
| Petr |       2 |
| Yuri |       1 |
+------+---------+
5 rows in set (0.00 sec)
```

Ch. 5 Data Types

Handling Missing or Invalid Data Values

- Assume the server is running in MySQL's "forgiving" SQL mode.
  - MySQL converts a negative value to ZERO into an UNSIGNED column.

```
mysql> set @@SQL_MODE='';
mysql> create table r (j INT  unsigned not null);
mysql> insert into r values (12), (-23), (null), ('abc');
mysql> select * from r;
+-----+
|  j   |
+-----+
|  12  |
|   0   | → convert -23 to 0
|   0   | → convert null string to 0
|   0   | → convert string value to 0
+-----+
4 rows in set (0.00 sec)
```

Ch. 5 Data Types

- In MySQL, INSERT statements maybe incomplete in the sense of not specifying a vlaue for every column.

```
mysql> create table t(
i int null,
j int not null,
k int default -1);

mysql> INSERT INTO t (i) VALUES (0);
mysql> INSERT INTO t (I, k) VALUES (0);
mysql> INSERT INTO t  VALUES ();
mysql>> select * from t;
```

```
+------+---+------+
| i    | j | k    |
+------+---+------+
|    0 | 0 |   -1 |
|    1 | 0 |    2 |
| NULL | 0 |   -1 |
+------+---+------+
3 rows in set (0.00 sec)
```

Ch. 5 Data Types

MySQL handles missing values as following:

- If the column definition contains a DEFAULT clause, MySQL inserts the values specified by that clause.
- If the column definition has no DEFAULT clause, missing-value handling depends on whether strict SQL mode is in effect and whether the table is transactional.
  - If strict mode is not in effect, MySQL inserts the implicit default values for the column data type and generate a warning.
  - If strict mode is in effect, an error occurs for transactional tables.

Ch. 5 Data Types

- When MySQL is operating in non-strict mode, it adjusts input values to legal values. p93
  - Conversion of out-of-range values to in-range values: If you insert a value less than -128 in a TINTINT column, MySQL stores -128 instead.
  - String truncation: If you inset ‘Sakila’ into a CHAR(4) column, MySQL store it as ‘Saki’
  - Enumeration and set value conversion: If a value that’s assigned to an ENUM column isn’t listed, MySQL convert it to ‘’ (the empty string). If a value that’s assigned to s SET column contains elements that aren’t listed, MySQL discards those elements, retaining only the legal elements.
  - Conversion to data type default: **if you attempt to store a value that cannot be converted to the column data type, MySQL stores the implicit default value for the type.**
  - Handling assignment of NULL to NOT NULL columns.

Ch. 5 Data Types

The following table shows how several types of string values are handled when converted to DATE or INT data types. It demonstrates several of the points just discussed. Note that only string values that look like dates or numbers convert properly without loss of information.

String Value	Converted to DATE	Converted to INT
'2010-03-12'	'2010-03-12'	2010
'03-12-2010'	'0000-00-00'	3
'0017'	'0000-00-00'	17
'500 hats'	'0000-00-00'	500
'bartholomew'	'0000-00-00'	0

Ch. 5 Data Types

Handling Invalid Values in Strict Mode

Enabling strict mode turns on general input value restrictions. In strict mode, the server rejects values that are out of range, have an incorrect data type, or are missing for columns that have no default. Strict mode is enabled using the `STRICT_TRANS_TABLES` and `STRICT_ALL_TABLES` mode values.

`STRICT_TRANS_TABLES` enables strict behavior for errors that can be rolled back or canceled without changing the table into which data is being entered.

`STRICT_ALL_TABLES` is similar to `STRICT_TRANS_TABLES` but causes statements for non-transactional tables to abort even for errors in the second or later rows of a multiple-row insert. This means that a partial update might occur, because rows earlier in the statement will already have been inserted.

Ch. 5 Data Types

- If you want your MySQL server to be as restrictive as possible about input data checking, set the sql_mode system varibale to TRADITIONAL.

SET sql_mode = ‘traditional’;

- To overirde input data restrictions that may be enabled, use INSERT IGNORE or UPDATE IGNORE.

If you want relaxed date checking that requires only that month and day values be in the respective ranges of 1 to 12 and 1 to 31, enable the ALLOW_INVALID_DATES SQL mode value:

```
SET sql_mode = 'ALLOW_INVALID_DATES';
```

You can use ALLOW_INVALID_DATES for relaxed date checking even in strict mode:

```
SET sql_mode = 'STRICT_ALL_TABLES,ALLOW_INVALID_DATES';
```

Ch. 6 Identifiers

- Identifiers are names given to database objects such as tables, stored routines, triggers, columns, etc. They are essentially the same thing as a variable or property in programming languages such as Java.

- Identifiers can be ***quoted*** or ***unquoted***.

Ch. 6 Identifiers

If unquoted, an identifier must follow these rules:

- An identifier may contain alpha numeric characters plus the $ and _
- An identifier may begin with any of the legal characters including a digit, this is different from most programming languages including Java which must begin with an alpha or underscore (_). However I recommend sticking to the same naming conventions you use for languages like Java and PHP.
- An identifier cannot consist entirely of digits

Ch. 6 Identifiers

- An Identifier may be quoted. To quote an identifier, enclose it within backtick (`). If the ANSI_QUOTES SQL mode is enabled, you may also quote an identifier by enclosing it with double quotes(“). When quoting identifiers they may contain any character, including spaces. Again, I recommend against these names for several reasons. One is simplicity, identifiers should be long enough to be meaningful, but not so long that they cannot be quickly and accurately typed.
- In general, any character may be used in a quoted identifier. Exception are that an identifier cannot contain a byte with a numberic value of 0 or 225, and database and table names cannot contain . (period), / (slash) or \ (back slash).
- A quoted identifier may consist entirely of digits.

Ch. 6 Identifiers

The following are all legal identifiers:

- Last_name
- $last_name
- Amount$
- `Street Address 1`
- `City, State  Zip-Code:`
- 2009_YTD
- `56789895`

Ch. 6 Identifiers

- Alias identifiers.

An alias identifier can include any character, but should be quoted if

1)it’s a reserved word (such as SELECT or DESC)

2) Consists entirely of digits.

Aliases may be quoted within single quotes, double quotes or backticks. p98

Ch. 6 Identifiers

- Case Sensitivity:

In general database identifiers are not case sensitive. In some cases tables names may be case sensitive depending on the operating system you are using and MySQL database engine you are using. MyISAM on Unix/Linux/Mac OS X would be case sensitive for table names, but not column names.

Ch. 6 Identifiers

- **Qualified Name:**

Both column and table names can be written in qualified form and sometimes must be written that way if necessary to eliminate ambiguity. This will come into play later on when we start doing multi-table joins. Tables may be qualified with the name of the database.

SELECT * FROM Country; -- May be re-written by adding the database name (world or in our case test).

SELECT * FROM world.Country; -- Or SELECT * FROM test.Country.

Columns can be qualified as such:

SELECT Country.Name FROM Country;
SELECT test.Country.Name FROM test.Country

Ch. 6 Identifiers

- **Using ReservedWords as Identifiers (JUST SAY NO):**

You cannot use reserved words such as: ORDER, SELECT, UPDATE, DELETE, … as reserved words unless you quote them. While quoting is legal, I recommend against it.

CREATE TABLE junk (order INT, dt DATE); -- This is illegal and will cause an error.

mysql> CREATE TABLE junk (order INT, dt DATE);

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'order INT, dt DATE)' at line 1

mysql> CREATE TABLE junk (`order` INT, dt DATE);

Query OK, 0 rows affected (0.07 sec)

-- This is legal, just NOT A GOOD IDEA!

Ch. 6 Identifiers

Note: p.100

- **To use a reserved word as a database, table, column, index identifier, you can quote it within backticks (`) or double quotes (“)**

mysql> create table t (`order` int not null, name varchar(50));

mysql> create table t (“order” int not null, name varchar(50));

- **To use a reserved word as an alias, you can use single quotes (‘), double quotes (“) or backticks (`)**

mysql> select 1 as `INTEGER`;

mysql> select 1 as “INTEGER”;.

mysql> select 1 as ‘INTEGER’;..