69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01713 432003

Lecture Sheet on "PHP+MySQL"
Day-7: (MySQL)

Objectives	Topics
❖ Choose column types and other column properties ❖ Create databases and tables ❖ Insert records, select, update and delete data ❖ Use conditionals and functions	❖ MySQL ❖ Choosing column types ❖ Creating databases and tables ❖ Inserting records ❖ Selecting data ❖ Using conditionals ❖ Using LIKE and NOT LIKE ❖ Sorting and limiting query results ❖ Updating and deleting data ❖ Using text and numeric functions ❖ Formatting date and time functions

What is MySQL?

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers. It is a second generation Open Source company that unites Open Source values and methodology with a successful business model. The MySQL Web site (http://www.mysql.com/) provides the latest information about MySQL software and MySQL AB.

Data Types

MySQL supports a number of data types in several categories: numeric types, date and time types, and string (character) types. This lecture first gives an overview of these data types, and then provides a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the allowable formats in which you can specify values.

Numeric Types

MySQL supports all of the standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keyword DEC is a synonym for DECIMAL. For numeric type storage requirements, see Section 10.5, "Data Type Storage Requirements". The numeric types used for the results of calculations depends on the operations being performed and the numeric types of the operands; for more information, see Section 11.5.1, "Arithmetic Operators". The BIT data type stores bit-field values and is supported for MyISAM, MEMORY, InnoDB, and NDBCLUSTER tables. As an extension to the SQL standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each of the integer types.

Type	Bytes	Minimum Value (Signed/Unsigned)	Maximum Value (Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

Another extension is supported by MySQL for optionally specifying the display width of integer data types in parentheses following the base keyword for the type (for example, INT(4)). This optional display width is used to display integer values having a width less than the width specified for the column by left-padding them with spaces. The display width does not constrain the range of values that can be stored in the column, nor the number of digits that are displayed for values having a width exceeding that specified for the column. For example, a column specified as SMALLINT(3) has the usual SMALLINT range of -32768 to 32767, and values outside the range allowed by three characters are displayed using more than three characters. When used in conjunction with the optional extension attribute ZEROFILL, the default padding of spaces is replaced with zeros. For example, for a column declared as INT(5) ZEROFILL, a value of 4 is retrieved as 00004. Note that if you store larger values than the display width in an integer column, you may experience problems when MySQL generates temporary tables for some complicated joins, because in these cases MySQL assumes that the data fits into the original column width.

Date and Time Types

The date and time types for representing temporal values are DATETIME, DATE, TIMESTAMP, TIME, and YEAR. Each temporal type has a range of legal values, as well as a “zero” value that may be used when you specify an illegal value that MySQL cannot represent. The TIMESTAMP type has special automatic updating behavior, described later on. For temporal type storage requirements, see Section 10.5, “Data Type Storage Requirements”. MySQL gives warnings or errors if you try to insert an illegal date. By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See Section 5.1.7, “SQL Modes”.) You can get MySQL to accept certain dates, such as '1999-11-31', by using the ALLOW_INVALID_DATES SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 0 to 12 and that the day is in the range from 0 to 31. These ranges are defined to

include zero because MySQL allows you to store dates where the day or month and day are zero in a DATE or DATETIME column. This is extremely useful for applications that need to store a birthdate for which you do not know the exact date. In this case, you simply store the date as '1999-00-00' or '1999-01-00'. If you store dates such as these, you should not expect to get correct results for functions such as DATE_SUB() or DATE_ADD() that require complete dates. (If you do not want to allow zero in dates, you can use the NO_ZERO_IN_DATE SQL mode). Prior to MySQL 5.1.18, when DATE values are compared with DATETIME values, the time portion of the DATETIME value is ignored, or the comparison could be performed as a string compare. Starting from MySQL 5.1.18, a DATE value is coerced to the DATETIME type by adding the time portion as '00:00:00'. To mimic the old behavior, use the CAST() function to cause the comparison operands to be treated as previously. For example: date_col = CAST(NOW() AS DATE); MySQL also allows you to store '0000-00-00' as a "dummy date" (if you are not using the NO_ZERO_DATE SQL mode). This is in some cases more convenient (and uses less data and index space) than using NULL values. Here are some general considerations to keep in mind when working with date and time types:

- ✓ MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). Only the formats described in the following sections are supported. It is expected that you supply legal values. Unpredictable results may occur if you use values in other formats.
- ✓ Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
  - ⇒ Year values in the range 70-99 are converted to 1970-1999.
  - ⇒ Year values in the range 00-69 are converted to 2000-2069.
- ✓ Although MySQL tries to interpret values in several formats, dates always must be given in year-month-day order (for example, '98-09-04'), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, '09-04-98', '04-09-98').
- ✓ MySQL automatically converts a date or time type value to a number if the value is used in a numeric context and vice versa.
- ✓ By default, when MySQL encounters a value for a date or time type that is out of range or otherwise illegal for the type (as described at the beginning of this section), it converts the value to the "zero" value for that type. The exception is that out-of-range TIME values are clipped to the appropriate endpoint of the TIME range.

The following table shows the format of the "zero" value for each type. Note that the use of these values produces warnings if the NO_ZERO_DATE SQL mode is enabled.

Data Type	"Zero" Value
DATETIME	'0000-00-00 00:00:00'

DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000

- ✓ The "zero" values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values '0' or 0, which are easier to write.
- ✓ "Zero" date or time values used through MyODBC are converted automatically to NULL in MyODBC 2.50.12 and above, because ODBC cannot handle such values.

String Types

The string types are CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET. This section describes how these types work and how to use them in your queries. For string type storage requirements, see Section 10.5, "Data Type Storage Requirements".

The CHAR and VARCHAR Types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained. The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters. The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed unless the PAD_CHAR_TO_FULL_LENGTH SQL mode is enabled. Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. In contrast to CHAR, VARCHAR values are stored as a one-byte or two-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values Data Types 637may require more than 255 bytes. If strict SQL mode is not enabled and you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of non-space characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, "SQL Modes". For VARCHAR columns, excess trailing spaces are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For CHAR columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode. VARCHAR values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL. The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns (assuming that the column uses a single-byte character set such as latin1):

Value	CHAR(4)	**Storage Required**	VARCHAR(4)	**Storage Required**

69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01713 432003

''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

The values shown as stored in the last row of the table apply only when not using strict mode; if MySQL is running in strict mode, values that exceed the column length are not stored, and an error results. If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval.

The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than non-binary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in the values.

The allowable maximum length is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR, except that the length for BINARY and VARBINARY is a length in bytes rather than in characters.

The BINARY and VARBINARY data types are distinct from the CHAR BINARY and VARCHAR BINARY data types. For the latter types, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains non-binary character strings rather than binary byte strings. For example, CHAR(5) BINARY is treated as CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin, assuming that the default character set is latin1. This differs from BINARY(5), which stores 5-bytes binary strings that have no character set or collation.

The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold. The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements.

BLOB columns are treated as binary strings (byte strings). TEXT columns are treated as non-binary strings (character strings). BLOB columns have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. TEXT columns have a character set, and values are sorted and compared based on the collation of the character set.

The ENUM Type

An ENUM is a string object with a value chosen from a list of allowed values that are enumerated explicitly in the column specification at table creation time.

An enumeration value must be a quoted string literal; it may not be an expression, even one that evaluates to a string value. For example, you can create a table with an ENUM column like this:

```
CREATE TABLE sizes (
    name ENUM('small', 'medium', 'large')
);
```

However, this version of the previous CREATE TABLE statement does *not* work:

```
CREATE TABLE sizes (
    c1 ENUM('small', CONCAT('med','ium'), 'large')
);
```

The SET Type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (“,”). A consequence of this is that SET member values should not themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

''
'one'
'two'
'one,two'

A SET can have a maximum of 64 different members.