This lecture covers the main types of MySQL.

Although learning how to use the MySQL types is important, there are a lot of types and it would be difficult to concentrate on all of them in one lecture (and the lecture would be pretty dull).

Therefore although they are all grouped together for easy reference, we will actually go over part of this material in a number of different lectures.

Types

Using any typed language means learning the available types.

Because there is no general mechanism to build new types out of old ones (like building classes in Java or structures in C) in MySQL there are a lot of types in MySQL many of which have several variants.

There are four major groups of datatypes

- Numeric Types
- String Types
- Date and Time Types
- Spatial Types

Integer Types

MySQL supports a variety of integer types

- INT, INTEGER
  A normal 32-bit integer with values from $-2^{31}$ to $2^{31} - 1$ just like a Java int.
- MEDIUMINT and SMALLINT
  24-bit and 16-bit integers respectively.
- TINYINT
  An 8-bit integer (like Java’s byte type).
- BIGINT
  A 64-bit integer (like Java’s long type).
Signed types

Unlike Java (but like C) you can declare any of these types to be unsigned for the situation where business logic dictates that the number can never be negative.

For example, the type TINYINT holds values in the range 
\[-128 \text{ to } 127\]

while TINYINT UNSIGNED holds values in the range

\[0 \text{ to } 255\].

Normally it makes sense to declare things like id-numbers to be unsigned.

Non-integer types

MySQL provides three basic non-integer types

- **FLOAT** or **FLOAT(M,D)**
  
  Single-precision floating point numbers. If \(M\) and \(D\) are specified, then the precision can be reduced to a total of \(M\) decimal digits with \(D\) after the decimal point.

- **DOUBLE** or **DOUBLE(M,D)**
  
  Double-precision floating point numbers

- **DECIMAL** or **DECIMAL(M)** or **DECIMAL(M,D)**

  Fixed-point numbers with \(M\) decimal digits and \(D\) after the point

For example, we might use **DECIMAL(8,2)** to store dollars-and-cents values.

String Types

MySQL has 5 basic string types

- **CHAR** and **VARCHAR**
- **BINARY** and **VARBINARY**
- **BLOB** and **TEXT**
- **ENUM**
- **SET**
Character Strings

CHAR (M) is a fixed-length string of M characters, while VARCHAR (M) is a variable-length string of at most M characters.

A CHAR (M) field always uses the full M characters no matter what it contains, and so can be wasteful of space if there are many fields shorter than the longest.

However the fixed size of a CHAR (M) column makes some internal operations much faster and so if the values all happen to be about the same size, then CHAR (M) is the best choice.

Something like student login names would be a perfect candidate for a CHAR (8) type.

Note: The full details of the differences between CHAR and VARCHAR are more complicated and technical than this brief description.

Character sets and collations

The CHAR and VARCHAR types represent strings of characters — but of course, representing characters in a computer requires specifying a character set.

For example, a string might be a string of ASCII characters (from the character set ascii or a string of Chinese characters (from the character set big5).

Moreover even among languages that use the same character set it is possible that the order of the characters is considered to be different — a collation is a set of rules that specify how to order strings.

MySQL allows the user to specify both a character-set and a collation individually for any string type fields.

The importance of collations

By default MySQL uses the latin1 character set, but even that has more than one different collation.

mysql> create table collationtest (    -> f1 CHAR(8) COLLATE latin1_swedish_ci,    -> f2 CHAR(8) COLLATE latin1_bin);

The collation latin1_swedish_ci is the default “English language” collation, while latin1_bin is the binary order according to the encoding of the character set.

Why is this important?

An example

mysql> SELECT * FROM collationtest;
+-------+-------+
| f1    | f2    |
+-------+-------+
| apple | apple |
| Apple | Apple |
+-------+-------+

The table contains just two rows both the same.
### Collation affects selection

```sql
SELECT * FROM collationtest
WHERE f1 = 'apple';
+-------+-------+
<table>
<thead>
<tr>
<th>f1</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>apple</td>
</tr>
<tr>
<td>Apple</td>
<td>Apple</td>
</tr>
</tbody>
</table>
+-------+-------+
2 rows in set (0.00 sec)
```

```sql
SELECT * FROM collationtest
WHERE f2 = 'apple';
+-------+-------+
<table>
<thead>
<tr>
<th>f1</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>apple</td>
</tr>
</tbody>
</table>
+-------+-------+
1 row in set (0.01 sec)
```

The default collation gives case-insensitive string comparisons, while the binary collation treats `a` and `A` as different.

---

### Binary Types

It is quite common to use characters and strings to encode data as well as for human-readable strings.

In this case, collations make no sense and to ensure that queries are not mangled by default collations changing, such data types can be declared as **BINARY** or **VARBINARY** with the same syntax as **CHAR** and **VARCHAR**.

The binary types are then just treated as a sequence of *bytes* with no particular interpretation as characters, and comparisons and sorting are based purely on their numerical value.

---

### BLOB and TEXT

For (more) truly variable-sized fields, you can use either **BLOB** or **TEXT** types.

The difference between them is that **TEXT** is viewed as a string of characters hence has a character set and a collation, whereas a **BLOB** (Binary Large Object) is a string of bytes.

Therefore a **TEXT** field is used for large amounts of textual data (for example, books, blog-entries, web-pages etc) while a **BLOB** is used for large amounts of pure data (for example, compressed files, images etc).

---

### Variants of BLOB and TEXT

There are four variants of each type which determine the maximum amount of storage.

- **TINYBLOB**, **TINYTEXT**
  - Maximum size of $2^8 - 1$ bytes/characters.
- **BLOB**, **TEXT**
  - Maximum size of $2^{16} - 1$ bytes/characters (64 K).
- **MEDIUMBLOB**, **MEDIUMTEXT**
  - Maximum size of $2^{24} - 1$ bytes/characters (16 Mb).
- **LONGBLOB**, **LONGTEXT**
  - Maximum size of $2^{32} - 1$ bytes/characters (4 Gb).
Inserting BLOBs

One immediate question is how to get binary data into and out of a table. For example, suppose we have a table `student` that will store student information including their student photograph.

```
CREATE TABLE student (
    snum INT NOT NULL,
    sname VARCHAR(40),
    photo BLOB
);

INSERT INTO student
VALUES(1058475, "John Smith",
    LOAD_FILE('/tmp/smith.jpg'));
```
Using sets

Sets are implemented in MySQL by using bit-patterns, and the syntax for accessing set-valued data is heavily dependent on the bitwise operators and these implementation details.

Consider the SET of interests (maybe from an online dating site)

- 'reading', 'movies', 'sport', 'gardening'

We can represent a subset of these 4 items by a collection of 4 bits as follows:

- 1000 represents 'reading'
- 0100 represents 'movies'
- 0010 represents 'sport'
- 0001 represents 'gardening'

(GF Royle 2006-8, N Spadaccini 2008)

Databases - MySQL Types 21 / 23

Multiple values

Now a set of interests is represented simply by setting the appropriate bits to 1.

- 1100 represents 'reading, movies'
- 0111 represents 'movies, sport, gardening'

Internally these are simply stored as numbers, but MySQL translates the numbers into readable sets of values for output purposes.

The numbers that are used are the bit-patterns viewed as binary numbers in reverse — thus 'movies, sport, gardening' has the value 14.

(GF Royle 2006-8, N Spadaccini 2008)

Databases - MySQL Types 22 / 23

Querying sets

A query involving a set-valued field can be done in a "pattern-matching" fashion or using the underlying binary representation. For example, to answer the query

Who is interested in gardening?

we could use either

```
SELECT * FROM people
WHERE interests LIKE '%gardening%';
```

or

```
SELECT * FROM people
WHERE interests & 8 <> 0;
```

(GF Royle 2006-8, N Spadaccini 2008)

Databases - MySQL Types 23 / 23