86

SQL Primer as Used in MySQL

The Structured Query Language provides the application programmer interface that allows data to be entered into, edited, and selected from most relational databases. It also provides users with suitable authorisation with the ability to create and edit the structures that hold the data.

For teaching purposes, we're using the mysql command to access the database which is being accessed by the mysql daemon, although practical applications will embed commands in the SQL language within commands (typically function calls) in another language, and to write practical applications you'll need to have a good understanding of both SQL and the other language of your choice.

86.1 The structure of a database as seen by SQL

A single database engine can handle more than one database at a time. It would be inefficient to run a whole series of **`mysqld`** daemons on the same computer at the same time.

Within a database, you'll have one or more tables of data. Each table will contain a number of records, and the data for each record will be divided into a series of fields.

Here's a diagram that shows how these components fit together:

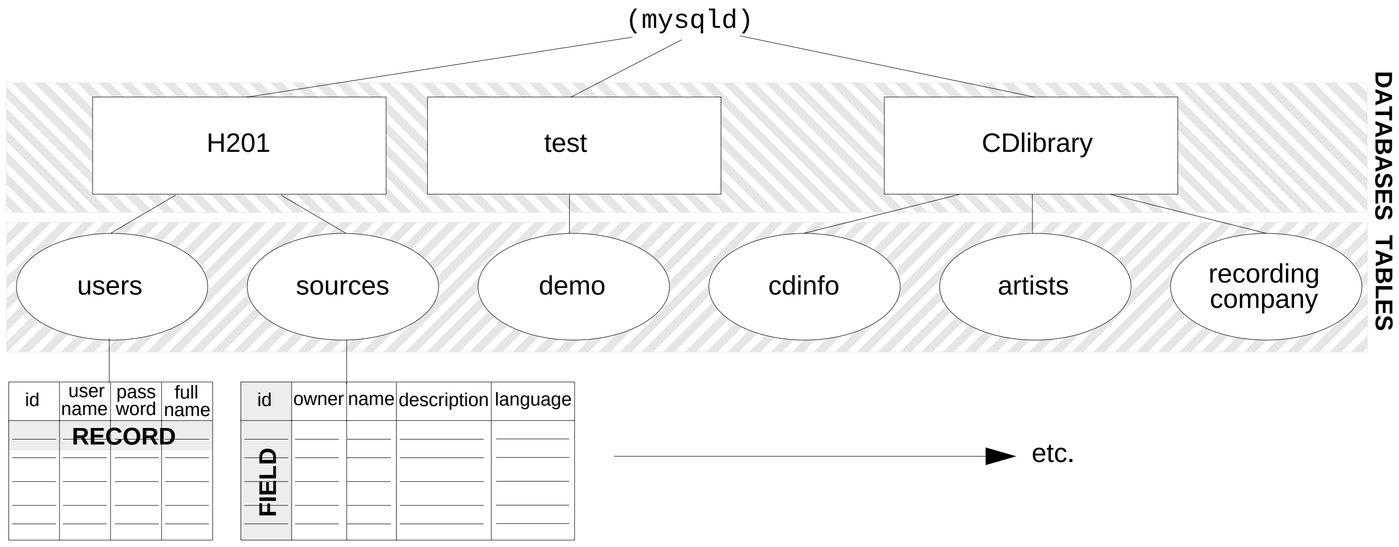


Figure 422 How database components fit together. Note the highlighted areas showing "Databases", "Tables", Record" and "Field".

86.2 SQL introduction

SQL commands all start with a keyword from the following list:

ALTER	CREATE	DELETE	DESCRIBE	
DROP	EXPLAIN	FLUSH	GRANT	
INSERT	KILL	LOAD	LOCK	
OPTIMIZE	REPLACE	REVOKE	SELECT	
SET	SHOW	UNLOCK	UPDATE	USE

You'll notice that all these are "doing" words, or verbs. SQL is designed to look as English-like as is practical for a computer language.

Although most users have got into the habit of writing these words in capitals (and then using lower case letters in the names of databases, tables and fields), they are not case significant in MySQL. Names of databases, tables and fields *are* case sensitive, however.

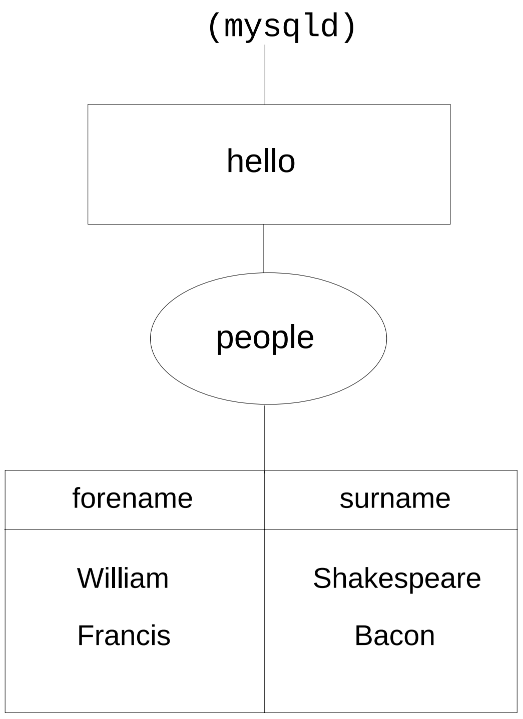
Other English-like words follow the command word (what's valid will vary from command to command), and table names, field names and data values will also be included.

Hello SQL World

Let's start our tour of SQL commands by following through the simplest of examples. Let's create a database, and a table within that database, with two fields of data. Let's add two records into that table and then perform a query that will return one of the records to us. Let's conclude this simple example by dropping the table and dropping the database.

Here is what our data will look like once we've created the table and added our records into it:

Figure 423 Structure of a simple database



```
bash-2.04$ mysql -uroot -pabc123
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.13

Type 'help' for help.

mysql> CREATE DATABASE hello;
Query OK, 1 row affected (0.00 sec)

mysql> use hello;
Database changed
mysql> CREATE TABLE people (forename TEXT, surname TEXT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO people VALUES ("William", "Shakespeare");
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO people VALUES ("Francis", "Bacon");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM people;
+----------+-------------+
| forename | surname     |
+----------+-------------+
| William  | Shakespeare |
| Francis  | Bacon       |
+----------+-------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM people WHERE surname = "Bacon";
+----------+---------+
| forename | surname |
+----------+---------+
| Francis  | Bacon   |
+----------+---------+
1 row in set (0.00 sec)

mysql> DROP TABLE people;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP DATABASE hello;
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
bash-2.04$
```

To start us off, we need to run the mysql program through which we'll talk to the database engine. Although there are sometimes SQL commands that can be run by unauthorised users, we'll be doing some fairly major modifications to the data help by MySQL and so for this example we've chosen to log in as the highest authority user - also known as root.

Note that the accounts used within SQL are different accounts to operating system login accounts; they're maintained in a different way and have different passwords. Because of the SQL background, though, Unix, LInux and MySQL all share the default name "root" for the administrator's account that can do almost anything!

```
bash-2.04$ mysql -uroot -pabc123
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.13

Type 'help' for help.

mysql>
```

SQL commands are usually sent, one by one, from another programming language to the SQL server and there's no need to add any special termination character. Within the mysql command itself, which has a command line interface, we use a semicolon. The initial prompt reminds us of this!

Let's now create a complete database:

```
mysql> CREATE DATABASE hello;
Query OK, 1 row affected (0.00 sec)

mysql>
```

Yes, that easy if we're logged in as root, even if this is a major action that you won't perform very often. MySQL maintains an internal table of databases (thus the "one row affected" message).

Lets's now create a table within that database; we need to move into that database first with a `use` statement:

```
mysql> use hello;
Database changed
mysql> CREATE TABLE people (forename TEXT, surname TEXT);
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Once we've selected "hello" as our current database, we can perform actions on it. As it's a brand new database, our first action is likely to be to create a table.

Creating a table requires more parameters than creating a data base - as well as the table name, more information is required. Within round brackets, we have specified the name of each field that is to be in the table, and we have specified the type of data that MySQL will hold in that particular column. Multiple words may be used to specify the field type, so a comma is used to indicate the end of one field definition and the start of the next.

Having created our table, we can add data into it:

```
mysql> INSERT INTO people VALUES ("William", "Shakespeare");
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO people VALUES ("Francis", "Bacon");
Query OK, 1 row affected (0.00 sec)

mysql>
```

That's two records (rows) inserted into the table. The values that we insert into the table may include spaces and commas, so we delimit the text parameters with quotes; either single or double quotes can be used. In this form of the `INSERT` command, we specify the value for each field in turn, in the order that the fields are named in the database.

The majority of accesses to your database are likely to be true queries; let's use mysql to look retrieve a table:

```
mysql> SELECT * FROM people;
+----------+-------------+
| forename | surname     |
+----------+-------------+
| William  | Shakespeare |
| Francis  | Bacon       |
+----------+-------------+
2 rows in set (0.00 sec)

mysql>
```

We've asked mysql to select all fields (thus the asterisk) from the people table. Mysql reports them back as a neatly formatted table in the current window.

In a live application, a table may contain a very large number of records and you'll want to use the database engine to filter the records rather than have to do so in a program:

```
mysql> SELECT * FROM people WHERE surname = "Bacon";
+----------+---------+
| forename | surname |
+----------+---------+
| Francis  | Bacon   |
+----------+---------+
1 row in set (0.00 sec)

mysql>
```

This is perhaps the simplest of "Where" clauses; in this case we knew exactly what we were looking for in the Surname field. Note that the report will include all records that match the condition.

If you're a programmer, you might expect us to have written == to check for equality, since a single = sign is used to set a variable. SQL is different; a single = sign is used as a comparison operator. There is an implicit loop in this SQL statement.

```
mysql> DROP TABLE people;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP DATABASE hello;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Once we're finished with our enquiries, we'll complete this example by deleting the table and database that we've been using. In practice, these are not common actions, but they complete our example and leave the data managed by our database engine in the same condition it was in before we started this example.

In summary, we have:

- created a database
- connected to that database
- created a table
- populated a table
- queried that table
- deleted the table
- deleted the database

It just remains for us to exit from the mysql client program:

```
mysql> exit
Bye
bash-2.04$
```

Initial administration and error handling

If you're the first user of the MySQL database on a system, chances are that you'll need to do some admin work, perhaps including starting up the database engine.

Check with your system administrator (or refer to other material) if you're unsure about this.

```
bash-2.04$ /usr/libexec/mysqld &
[1] 1408
bash-2.04$ /usr/libexec/mysqld: ready for connections

bash-2.04$
```

The mysqld program may vary in location on some systems. The extra "&" character on the command starts the daemon in the background, so that you get a prompt back and can carry on working in the window in which you stared it running.

Let's now see a brief session with two common errors:

```
bash-2.04$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.13

Type 'help' for help.

mysql> CREATE DATABASE hello
                -> ;
ERROR 1044: ?Access denied for user: '@localhost' to database 'hello'
mysql> exit
Bye
```

Firstly, we forgot to add the ";" on the end of the **CREATE** command. SQL commands can be very long, so mysql gives you the option of typing them in over a number of lines, thus the extra **->** prompt which asks us to continue typing. Oops, we just finished with our ";" !

We also forgot to specify the user name and password for mysql, and so we were denied access when we tried to create our new database.

86.3 Field types and modifiers in MySQL

In our first example table, we held strings of text - but we might want to use our database to store other types of information too. In this section, we'll look at the data types that are available:

- Floating point (real) numbers
- Whole (Integer) numbers
- Dates and Times
- Text Strings
- Blobs
- One or more values from a pre-defined list

In all cases, there are a number of data types available within this grouping, and modifiers can be applied to provide extra facilities and/or fine-tune the field specification.

We make no apologies for spending quite some time of field types and definitions; the careful design of your database tables is extremely important to a successful application.

Floating point (real) numbers

Floating point numbers are used to hold values that may have a decimal part - for example, if you wanted to accurately hold the distance from your home to our training centre in kilometres, or the temperature at a given location and time.

Computers hold floating point numbers internally to a certain level of accuracy, and within a certain numeric range.

In MySQL, you can specify:

type	accuracy	range	
single precision	7 significant figures	+- 3.4 x 10^{38}	maximum
		+- 3.4 x 10^{-38}	minimum non-zero
double precision	14 significant figures	+- 2.2 x 10^{308}	maximum
		+- 2.2 x 10^{-308}	minimum non-zero

Your choice can make a significant difference to storage requirements. A single precision number takes four bytes of storage and a double precision takes eight.

The following all specify single precision floating point numbers:

```
FLOAT
FLOAT(4)
FLOAT(length,decimal)
FLOAT4(length,decimal)
```

In the latter cases, the length really should be 4 (it's the length in bytes) but if you use another figure in the MySQL flavour of SQL, you'll get away with it. The decimal value gives the number of decimal places to be used; you might want to do this so that your database will only report figures to a sensible accuracy, even if someone attempts to put in something with many more decimal places than they should.

For double precision numbers, you may specify:

```
FLOAT(8)
FLOAT8(length,decimal)
DOUBLEPRECISION(length,decimal)
REAL(length,decimal)
```

In all cases, the length and decimal parameters are optional; omit them, and a default is used.

Whole (Integer) numbers

Integers are used for whole numbers - in other words numbers where it wouldn't be sensible to have decimal or fraction parts. You wouldn't use an integer for a distance or a temperature, but you would use it for the number of people booked on a training course, or for a payroll number for an employee. You can specify no less than five different lengths of integer, depending on the minimum and maximum number that you want to store.

	type	bytes	range
	INT1	1	-128 to + 127 (+-2^7)
or	TINYINT		
	INT2	2	-32768 to +32767 (+-2^{15})
or	SMALLINT		
	INT3	3	-8388608 to +8388607 (+-2^{23})
or	MEDIUMINT		
or	MIDDLEINT		
	INT	4	+-2^{31}
or	INTEGER		
or	INT4		
	INT8	8	+-2^{63}
or	BIGINT		

In all cases, you can specify a length if you wish (not that it changes anything) and other modifiers. The **`UNSIGNED`** modifier, which can only be applied to this group of whole number fields, disallows any number below zero, and doubles the maximum number that can be stored. For example, the range of an **`INT2 UNSIGNED`** is 0 to 65535.

Text Strings

Text Strings can be held in a number of different field types, depending on the maximum possible length of the text:

	type	bytes	max length
	TINYTEXT	length + 1	255 characters
	TEXT	length + 2	64 Kbytes
or	LONG VARCHAR		
	MEDIUMTEXT	length + 3	16 Mbytes
	LONGTEXT	length + 4	4 GBytes

If you're always going to have the same number of characters in a field, you can use

CHAR(length)	length	size specified by the parameter

and if you want to specify a variable number of characters with a user pre-defined maximum, you can specify:

	VARCHAR(length)	length	255
or	CHAR	(length)	VARYING

The main differences here are:

- a **CHAR** field always takes the number of bytes specified when it is defined, and space-fills short fields, whereas a **VARCHAR** will only take up the number of bytes it needs, but
- a **CHAR** field can be as long as you like, but a **VARCHAR** is limited to 255 characters.

If you'll be sorting using a text field later on, note that text field sorts are care insensitive by default. You can specify the **BINARY** modifier, or use **BINARY** in place of **CHAR** and **VARBINARY** in place of **VARCHAR** to make sorts case sensitive.

DECIMAL and NUMERIC fields are also available to hold floating point numbers as strings of characters. They're treated as **CHAR** variables, and you specify a length and a number of decimal places as you specify the field.

Caution: If you are looking to store truly binary data in your table, you should use a Blob; we'll be looking at those shortly.

One or more values from a pre-defined list

If you want to select a single value from a pre-defined list of possibles, then you should use an **ENUM** field. You are allowed a maximum or 65,535 possible different values in your list, and **NULL** can always be specified. You can set the value by name, or by the position in the list. **ENUM** fields usually occupy 1 byte, unless there are over 255 options.

If you want to be able to select multiple values from a pre-defined list of possibilities, you should use a **SET** field. A maximum of 64 different options are available in SET fields, and SET fields occupy one byte for every eight possible values.

Example: If you were holding a table of vehicle insurance information, you might well use:

ENUM	for the vehicle manufacturer (one and only one for each vehicle)
SET	for additional risks (could be 0 or more from: – driver under 25 – kept on public road – driver has poor record – driver lives in high crime area)

Dates and Times

Four different field types are available for you to store dates and/or times:

type	bytes	output as	input as
DATE	3	YYYY-MM-DD	YY-MM-DD
			YYYY-MM-DD
			YYMMDD
			YYYYMMDDHHMMSS (time ignored)
TIME	3	HH:MM:SS	HH:MM:SS
		HHMMSS	
		HHMM (secs set to 0)	
		HH (mins and secs set to 0)	
DATETIME	8	YYYY-MM-DD HH:MM:SS	YYYY-MM-DD HH:MM:SS
YEAR	1	YYYY	YYYY
		YY	YY

With **YEAR**, if you originally put a two-figure year into the database, that's what you'll get out. For sorting and selection purposes, two-digit years are assumed to be between 1970 and 2069. Four-digit years must be between 1901 and 2155.

With the other time and date formats, the output format is identical no matter which of the possible input formats you have used.

A **TIMESTAMP** field is also available, and this is updated every time the row of the table is modified. You can also force the value in this field if you want. The output format depends on the length you specify:

length	output	example
14 (default)	YYYYMMDDHHMMSS	20030416123521
12	YYMMDDHHMMSS	030416123521
8	YYYYMMDD	20030416
6	YYMMDD	030416

Each of these examples is for 16th April 2003, as 12:35 and 21 seconds.

Blobs

The group of data types are "blobs". A Blob is a field in which you can hold anything you want – "user data" if you like, including true binary information. You would use a blob if you wanted to store a .gif image or a word document within an SQL database, for example.

	type	bytes	max length
	TINYBLOB	length + 1	255 bytes
	BLOB	length + 2	64 KBytes
or	LONG VARBINARY		
	MEDIUMBLOB	length + 3	16 Mbytes
	LONGBLOB	length + 4	4 GBytes

Note that many computers run operating systems that have a file size limit of two Gbytes, so that although (in theory) you can define a `LONGBLOB` in any version of MySQL, on such systems you'll be severely limited by the operating system.

Other Modifiers

We've just seen the various types of fields that we can define in our tables, divided up into six groupings to help you determine which you should be using to hold any particular data. We've already examined certain modifiers which apply to individual groups, but there are other modifiers such as:

AUTO_INCREMENT	automatically assign the next available number to a numeric field. Data can be inserted normally if required.
DEFAULT value	The default value to use if this field isn't given a value. If not specified and no value given, NULL is used
NOT NULL	Specifies that a field may not contain a NULL Value
PRIMARY KEY	Specifies that this field is the primary key for the table

Example

There's a lot of field types we can define there. Let's set up an SQL database to hold data on our DVD collection. What do we want to hold, and in what type of field? Here are some fields:

title	**VARCHAR(100)**	A piece of text, and no film title will exceed 100 characters!
length	**FLOAT**	Length in minutes of main presentation
cert	**ENUM**	Must be one of U, PG, 12, 12A, 15, 18, Unclass and banned
videotype	**ENUM**	Must be one of PAL, NTSC or SECAM
lang	**SET**	One or more of English, French, German, Italian, Spanish, Dutch, Polish, Hungarian, Hindi, Finnish, Swedish and Hebrew
bought	**DATE**	The date we purchased the DVD (we don't care what time of day it was!)

It would also be a good idea to assign a unique reference number to each DVD, and to keep a timestamp when the record was last changed:

```
id      SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT NOT NULL
```

It's possible that we may have over 256 DVDs at some time, but 65535 seems a reasonable limit.

changed **TIMESTAMP** Quite happy with the default 14 character format!

The commands used to create this table will be quite long; we're likely to make a few typing errors as we write them, and we're likely to have to test them out and refine them more than once, so we've written them into plain text files that we've used as the out inputs to mysql. We've chosen to use two files:

	make_dvd.sql	to create the database and table
and	**pop_dvd.sql**	to insert sample records into the table

We'll then look at the data using a select command directly. Let's see how that looks ...

```
bash-2.04$ mysql -uroot -pabc123 < make_dvd.sql
bash-2.04$ mysql -uroot -pabc123 < pop_dvd.sql
bash-2.04$ mysql -uroot -pabc123
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 14 to server version: 4.0.13

Type 'help' for help.

mysql> use entertainment;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker start-up with -A
Database changed
```

```
mysql> select * from dvd ;
+----+---------------+--------+------+-----------+-----------------------------------------+------------+----------------+
| id | title         | length | cert | videotype | lang                                    | bought     | changed        |
+----+---------------+--------+------+-----------+-----------------------------------------+------------+----------------+
|  1 | Dinosaur      |   79.5 | PG   | PAL       | English                                 | 2001-03-15 | 20010704000000 |
|  2 | Chicken Run   |     81 | U    | PAL       | English                                 | 2001-02-07 | 20010704000000 |
|  3 | Groundhog Day |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010704000000 |
+----+---------------+--------+------+-----------+-----------------------------------------+------------+----------------+
3 rows in set (0.00 sec)

mysql> exit
Bye
bash-2.04$
```

That report looks fine. It's a good idea to test your table setups and check the integrity of your data through the mysql program, even if you're using other methods to write and maintain it.

Let' see what was in the command files, firstly **`make_dvd.sql`**:

```
drop database if exists entertainment;
create database entertainment;
use entertainment;

drop table if exists dvd;
create table dvd (
        id      SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT NOT NULL,
        title   VARCHAR(100),
        length  FLOAT,
        cert    ENUM ("U","PG","12","12A","15","18","UNCLASS","BANNED"),
        videotype       ENUM ("PAL","NTSC","SECAM"),
        lang    SET ("English","French","German","Italian","Spanish",
                "Dutch","Polish","Hungarian","Hindi","Finnish","Swedish","Hebrew"),
        bought  DATE,
        changed TIMESTAMP );
```

and **`pop_dvd.sql`**:

```
use entertainment;

insert into dvd values (
1, "Dinosaur", 79.5, "PG", "PAL", "English", "2001-03-15", "2001-07-04");

insert into dvd values (
2, "Chicken Run", 81.0, "U", "PAL", "English", "2001-02-07", "2001-07-04");

insert into dvd values (
3, "Groundhog Day", 97.0, "PG", "PAL",
                "English,French,German,Italian,Spanish", "2000-12-08", "2001-07-04");
```

Indexes

Indexes can help the database engine store large tables is a more easily searchable way. You define indexes as you create a table using the **`INDEX`** and **`UNIQUE INDEX`** commands within your table create, or a separate **`CREATE INDEX`** command. MySQL also supports a **`PRIMARY KEY`**, which is a unique key field that's set up as a unique index by virtue of that specification.

Choose your primary key with care, as you'll be able to use that field later to join two table together if you need to do so.

86.4 Other matters

A note on reserved words

As SQL is an English-like language. If you attempt to use the certain English words that it understands as your field names, you will get some strange-looking error messages. You can't use the command names as field names either.

At first glance you might think "Who in their right mind would call a field **`AS`**, **`FROM`**, **`IF`** or **`INTO`**, but you would be surprised how easy it is to fall into this trap. We've tried ...

... in a table of public transport services to name fields "from" and "to" ....

... in a table of information about stately homes to hold a description in a field called "desc" .... (check it out – it's a MySQL command provided for Oracle compatibility).

If you get a weird error when you're creating a table, try changing field names.

You can use function names as field names, by the way. You'll find that we have an example on this course that has a field called "length", and that we use the **`length`** function too.

Exercise

You're going to create a database with a single table to hold information about the reference books that are available on this course. As well as things like the title and authors, you want to hold the length in pages, the publication date, and a list of the subjects that the book relates to from a specified list (Linux, PHP, Perl, Java, Tcl/Tk, HTML and MySQL).

Work out what fields you need, create the database and table, and add in some sample records to check that everything is as it should be.

86.5 Commands to enter new rows of data

The INSERT command

The command you'll normally use to add rows of data into a table is the **INSERT** command.

The insert command can be told the values for every field in the record as we did in the previous section:

```
        insert into dvd values (
        1, "Dinosaur", 79.5, "PG", "PAL", "English", "2001-03-15", "2001-07-04");
```

or you can specify the names of fields individually if you wish. All fields not specified will be defaulted:

```
bash-2.04$ mysql -uroot -pabc123
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21 to server version: 4.0.13

Type 'help' for help.

mysql> use entertainments;
ERROR 1049: ?Unknown database 'entertainments'
mysql> use entertainment;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker start-up with -A

Database changed
mysql> insert into dvd (title, length, cert, lang)
    -> values ("Erin Brockovich", 126, "15", "English,German");
Query OK, 1 row affected (0.00 sec)

mysql> select * from dvd;
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
| id | title           | length | cert | videotype | lang                                 | bought     | changed        |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
|  1 | Dinosaur        |   79.5 | PG   | PAL       | English                              | 2001-03-15 | 20010704000000 |
|  2 | Chicken Run     |     81 | U    | PAL       | English                              | 2001-02-07 | 20010704000000 |
|  3 | Groundhog Day   |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010704000000 |
|  4 | Erin Brockovich |    126 | 15   | NULL      | English,German                       | NULL       | 20010701104404 |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
4 rows in set (0.00 sec)

mysql> exit
Bye
bash-2.04$
```

You'll notice how the videotype and bought fields that we didn't specify were set to **NULL**, but because of modifiers used the id was set to 4, and because "changed" is a timestamp field, it was set to reflect the date and time of the update.

On recent versions of MySQL, you can specify multiple sets of values if you wish to insert more than one row at a time, but you can't rely on this working on other SQL database engines.

On busy systems, you can specify a priority level after the **INSERT** command:

DELAYED	will delay the change until all incoming **SELECTS** are completed
LOW_PRIORITY	will delay the change until all other client operations are complete

The REPLACE command

The **REPLACE** command works like the **INSERT** command, except that it will override old conflicting data if your new records are specified with an existing unique key. Fewer formats are available for the replace command

The LOAD command

The **LOAD** command is a quick way to insert multiple rows into a table from an incoming data file. By default, **LOAD** assumes that the incoming file is in the current directory, has tabs between the fields, a \ is used to escape special characters, and each line is terminated with a new line.

Here's a tab-delimited file containing data about some more DVDs:

```
The Piano          15      115
The Color Purple   15      148
Billy Elliott      15      106
Patch Adams        15      111
Toy Story 2        U       89
```

And we'll now load that into our database:

```
bash-2.04$ mysql -uroot -pabc123
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 22 to server version: 4.0.13

Type 'help' for help.

mysql> use entertainment
Reading table information for completion of table and column names
You can turn off this feature to get a quicker start-up with -A

Database changed
mysql> LOAD DATA INFILE "/home/graham/mysql/dvdinfo.txt" INTO TABLE dvd
    -> (title, cert, length);
Query OK, 5 rows affected (0.04 sec)
Records: 5  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from dvd;
+----+------------------+--------+------+-----------+----------------------------------------+------------+----------------+
| id | title            | length | cert | videotype | lang                                   | bought     | changed        |
+----+------------------+--------+------+-----------+----------------------------------------+------------+----------------+
|  1 | Dinosaur         |   79.5 | PG   | PAL       | English                                | 2001-03-15 | 20010704000000 |
|  2 | Chicken Run      |     81 | U    | PAL       | English                                | 2001-02-07 | 20010704000000 |
|  3 | Groundhog Day    |     97 | PG   | PAL       | English,French,German,Italian,Spanish  | 2000-12-08 | 20010704000000 |
|  4 | Erin Brockovich  |    126 | 15   | NULL      | English,German                         | NULL       | 20010701104404 |
|  5 | The Piano        |    115 | 15   | NULL      | NULL                                   | NULL       | 20010701114551 |
|  6 | The Color Purple |    148 | 15   | NULL      | NULL                                   | NULL       | 20010701114551 |
|  7 | Billy Elliott    |    106 | 15   | NULL      | NULL                                   | NULL       | 20010701114551 |
|  8 | Patch Adams      |    111 | 15   | NULL      | NULL                                   | NULL       | 20010701114551 |
|  9 | Toy Story 2      |     89 | U    | NULL      | NULL                                   | NULL       | 20010701114551 |
+----+------------------+--------+------+-----------+----------------------------------------+------------+----------------+
9 rows in set (0.00 sec)

mysql> exit
```

Quick, clean, easy. But you'll notice that our original table definition should have had a default value of "PAL" for the Video Type, and "English" for the language! We can come back and correct that later, but first we need to learn a little more about reading data back out from our database.

86.6 Enquiry commands

The **SELECT** command is the main function of extracting data from database tables; it has many options and we'll start with some simple examples.

As a minimum, we write:

```
        SELECT {whatever} FROM {table}
```

In most uses, **{whatever}** is either a "*" or a list of column names, and **{table}** is the name of the table.

Let's read back our DVD table, but only look at the title, length and certificate fields:

```
mysql> select title, length, cert FROM dvd;
+------------------+--------+------+
| title            | length | cert |
+------------------+--------+------+
| Dinosaur         |   79.5 | PG   |
| Chicken Run      |     81 | U    |
| Groundhog Day    |     97 | PG   |
| Erin Brockovich  |    126 | 15   |
| The Piano        |    115 | 15   |
| The Color Purple |    148 | 15   |
| Billy Elliott    |    106 | 15   |
| Patch Adams      |    111 | 15   |
| Toy Story 2      |     89 | U    |
+------------------+--------+------+
```

Conditional Clauses

Good, by what if we're looking for a young visitor on holiday with us to see a DVD, we really don't want to offer him the heavy adult content of Patch Adams; in fact, we would rather he chose a "U".

```
mysql> select title, length, cert FROM dvd WHERE cert = "U";
+-------------+--------+------+
| title       | length | cert |
+-------------+--------+------+
| Chicken Run |     81 | U    |
| Toy Story 2 |     89 | U    |
+-------------+--------+------+
```

Perhaps a PG would be acceptable too?

```
mysql> select title, length, cert FROM dvd WHERE cert IN("U", "PG");
+---------------+--------+------+
| title         | length | cert |
+---------------+--------+------+
| Dinosaur      |   79.5 | PG   |
| Chicken Run   |     81 | U    |
| Groundhog Day |     97 | PG   |
| Toy Story 2   |     89 | U    |
+---------------+--------+------+
```

And perhaps we're short of time and should offer the shorter DVDs first?

```
mysql> select title, length, cert FROM dvd WHERE cert IN("U", "PG") ORDER BY length;
+---------------+--------+------+
| title         | length | cert |
+---------------+--------+------+
| Dinosaur      |   79.5 | PG   |
| Chicken Run   |     81 | U    |
| Toy Story 2   |     89 | U    |
| Groundhog Day |     97 | PG   |
+---------------+--------+------+
```

You're starting to see some of the flexibility, but just starting!

The **WHERE** clause understands mathematical operators, and parentheses to combine those operators. It understands the words **NOT**, **AND** and **OR** (alternatives are **! &&** and **||** if you're familiar with them from another language), and comparators:

= **!=** (also **<>**)
<= **<**
>= **>**

and where necessary it converts between data types for you (like PHP, unlike Perl 5 in this instance).

It also understands a number of other words such as

BETWEEN looking for a range
IN looking for a value from a list
LIKE which does a match using **%** to mean "anything" and **_** to mean "any one character"
NOT LIKE
REGEXP
or **RLIKE** which matches to a regular expression
and
NOT REGEXP

Let's select all movies of less than two hours with titles starting with the letter T:

```
mysql> select title, length from dvd where length < 120 and title LIKE "T%";
+-------------+--------+
| title       | length |
+-------------+--------+
| The Piano   |    115 |
| Toy Story 2 |     89 |
+-------------+--------+
```

Any all movies with two words in their title:

```
mysql> select title, length from dvd where title REGEXP "^[[:graph:]]+[[:space:]][[:graph:]]+$";
+-----------------+--------+
| title           | length |
+-----------------+--------+
| Chicken Run     |     81 |
| Groundhog Day   |     97 |
| Erin Brockovich |    126 |
| The Piano       |    115 |
| Billy Elliott   |    106 |
| Patch Adams     |    111 |
+-----------------+--------+
```

There's a wide range of mathematical, text and date functions available too. The next example shows all dvds that were bought on a Thursday (careful: day 1 is Sunday in SQL):

```
mysql> select * from dvd where (dayofweek(bought) = 5);
+----+----------+--------+------+-----------+---------+------------+----------------+
| id | title    | length | cert | videotype | lang    | bought     | changed        |
+----+----------+--------+------+-----------+---------+------------+----------------+
|  1 | Dinosaur |   79.5 | PG   | PAL       | English | 2001-03-15 | 20010704000000 |
+----+----------+--------+------+-----------+---------+------------+----------------+
```

And the final example shows all DVDs that have more than 12 characters in their title:

```
mysql> select * from dvd where (length(title) > 12);
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
| id | title           | length | cert | videotype | lang                                 | bought     | changed        |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
|  3 | Groundhog Day   |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010704000000 |
|  4 | Erin Brockovich |    126 | 15   | NULL      | English,German                       | NULL       | 20010701104404 |
|  6 | The Color Purple |   148 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  7 | Billy Elliott   |    106 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
```

Returning table report information

The Select command can also be used to run functions on fields, and return an overall result. In this next example, we look firstly for the average length of our DVDs, then for the shortest and longest. Finally, we count how many of our DVDs actually have the date that they were bought recorded in the database.

```
mysql> select avg(length) from dvd;
+-----------------+
| avg(length)     |
+-----------------+
| 105.83333333333 |
+-----------------+
1 row in set (0.01 sec)

mysql> select min(length),max(length) from dvd;
+-------------+------------+
| min(length) | max(length) |
+-------------+------------+
|        79.5 |        148 |
+-------------+------------+
1 row in set (0.00 sec)

mysql> select count(bought) from dvd;
+---------------+
| count(bought) |
+---------------+
|             3 |
+---------------+
1 row in set (0.00 sec)
```

Exercise

Add some further data to the reference book table that you created in the previous exercise, using a `load` command to read that data from a tab-delimited file.

Use SQL enquiries to generate:

1. A list of books sorted by publication date
2. A list of all books about any form of SQL
3. A count of the number of books about SQL.

86.7 Commands to modify existing rows of data

The UPDATE command

We introduced the **REPLACE** command earlier which allows you to override a row by replacing the whole row upon matching a unique key.

The **UPDATE** command allows you to update (alter) specific values in a table. Used with a **WHERE** clause, only certain records in the table are affected; used without a **WHERE** clause, changes are applied to every row.

In this next example, we modify all the most recent records (those with an id greater than 6) to indicate that the DVD is in English. We then add 30 minutes to the length of each DVD (perhaps because we now want the length field to reflect how long we should allow to watch the DVD, including time to make popcorn and settle down before we press the play button?).

```
mysql> select * from dvd;
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
| id | title           | length | cert | videotype | lang                                 | bought     | changed        |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
|  1 | Dinosaur        |   79.5 | PG   | PAL       | English                              | 2001-03-15 | 20010704000000 |
|  2 | Chicken Run     |     81 | U    | PAL       | English                              | 2001-02-07 | 20010704000000 |
|  3 | Groundhog Day   |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010704000000 |
|  4 | Erin Brockovich |    126 | 15   | NULL      | English,German                       | NULL       | 20010701104404 |
|  5 | The Piano       |    115 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  6 | The Color Purple |   148 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  7 | Billy Elliott   |    106 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  8 | Patch Adams     |    111 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  9 | Toy Story 2     |     89 | U    | NULL      | NULL                                 | NULL       | 20010701114551 |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
9 rows in set (0.00 sec)

mysql> update dvd set lang="English" where id > 6;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from dvd;
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
| id | title           | length | cert | videotype | lang                                 | bought     | changed        |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
|  1 | Dinosaur        |   79.5 | PG   | PAL       | English                              | 2001-03-15 | 20010704000000 |
|  2 | Chicken Run     |     81 | U    | PAL       | English                              | 2001-02-07 | 20010704000000 |
|  3 | Groundhog Day   |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010704000000 |
|  4 | Erin Brockovich |    126 | 15   | NULL      | English,German                       | NULL       | 20010701104404 |
|  5 | The Piano       |    115 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  6 | The Color Purple |   148 | 15   | NULL      | NULL                                 | NULL       | 20010701114551 |
|  7 | Billy Elliott   |    106 | 15   | NULL      | English                              | NULL       | 20010701132929 |
|  8 | Patch Adams     |    111 | 15   | NULL      | English                              | NULL       | 20010701132929 |
|  9 | Toy Story 2     |     89 | U    | NULL      | English                              | NULL       | 20010701132929 |
+----+-----------------+--------+------+-----------+--------------------------------------+------------+----------------+
9 rows in set (0.00 sec)

mysql> update dvd set length=length+30 ;
Query OK, 9 rows affected (0.00 sec)
Rows matched: 9  Changed: 9  Warnings: 0
```

```
mysql> select * from dvd;
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+
| id | title            | length | cert | videotype | lang                                  | bought     | changed        |
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+
|  1 | Dinosaur         |  109.5 | PG   | PAL       | English                               | 2001-03-15 | 20010701133049 |
|  2 | Chicken Run      |    111 | U    | PAL       | English                               | 2001-02-07 | 20010701133049 |
|  3 | Groundhog Day    |    127 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010701133049 |
|  4 | Erin Brockovich  |    156 | 15   | NULL      | English,German                        | NULL       | 20010701133049 |
|  5 | The Piano        |    145 | 15   | NULL      | NULL                                  | NULL       | 20010701133049 |
|  6 | The Color Purple |    178 | 15   | NULL      | NULL                                  | NULL       | 20010701133049 |
|  7 | Billy Elliott    |    136 | 15   | NULL      | English                               | NULL       | 20010701133049 |
|  8 | Patch Adams      |    141 | 15   | NULL      | English                               | NULL       | 20010701133049 |
|  9 | Toy Story 2      |    119 | U    | NULL      | English                               | NULL       | 20010701133049 |
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+
9 rows in set (0.00 sec)

mysql> update dvd set length=length-30;
Query OK, 9 rows affected (0.00 sec)
Rows matched: 9  Changed: 9  Warnings: 0
```

The DELETE command

The delete command is used to delete records from a table, and can take a **WHERE** clause.

Beware: Without a **WHERE** clause, the **DELETE** command will delete all rows from a table!

86.8 Commands to modify the metadata

The ALTER Command

The **ALTER** command allows you to change the structure of a table. This is a more fundamental operation than just changing the data a table contains, and it may result in other applications and scripts being rendered incompatible with the new data structure.

A common use of **ALTER** is with the **ADD** subcommand, allowing you to provide additional columns of data in a table. **ALTER** is also useful to set or drop default values.

In the following example, we change the default language to English; you'll note that this updates all our timestamps. We then add two new columns of data – the number of times that we've seen the movie and the company who produced it.

Finally, we've added yet another DVD to our table and displayed the whole table to check that the new and default values are working. Note that setting a default does not cause any retrospective actions. In other words, by setting the language to English by default, we have not revisited old records that have any old default value.

```
mysql> ALTER TABLE dvd ALTER lang SET DEFAULT "English";
Query OK, 9 rows affected (0.02 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> select * from dvd;
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+
| id | title            | length | cert | videotype | lang                                  | bought     | changed        |
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+
|  1 | Dinosaur         |   79.5 | PG   | PAL       | English                               | 2001-03-15 | 20010701133113 |
|  2 | Chicken Run      |     81 | U    | PAL       | English                               | 2001-02-07 | 20010701133113 |
|  3 | Groundhog Day    |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010701133113 |
|  4 | Erin Brockovich  |    126 | 15   | NULL      | English,German                        | NULL       | 20010701133113 |
|  5 | The Piano        |    115 | 15   | NULL      | NULL                                  | NULL       | 20010701133113 |
|  6 | The Color Purple |    148 | 15   | NULL      | NULL                                  | NULL       | 20010701133113 |
|  7 | Billy Elliott    |    106 | 15   | NULL      | English                               | NULL       | 20010701133113 |
|  8 | Patch Adams      |    111 | 15   | NULL      | English                               | NULL       | 20010701133113 |
|  9 | Toy Story 2      |     89 | U    | NULL      | English                               | NULL       | 20010701133113 |
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+
9 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE dvd ADD (timeseen INT DEFAULT 1,company TEXT);
Query OK, 9 rows affected (0.00 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> select * from dvd;
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+----------+---------+
| id | title            | length | cert | videotype | lang                                  | bought     | changed        | timeseen | company |
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+----------+---------+
|  1 | Dinosaur         |   79.5 | PG   | PAL       | English                               | 2001-03-15 | 20010701133113 |        1 | NULL    |
|  2 | Chicken Run      |     81 | U    | PAL       | English                               | 2001-02-07 | 20010701133113 |        1 | NULL    |
|  3 | Groundhog Day    |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010701133113 |        1 | NULL    |
|  4 | Erin Brockovich  |    126 | 15   | NULL      | English,German                        | NULL       | 20010701133113 |        1 | NULL    |
|  5 | The Piano        |    115 | 15   | NULL      | NULL                                  | NULL       | 20010701133113 |        1 | NULL    |
|  6 | The Color Purple |    148 | 15   | NULL      | NULL                                  | NULL       | 20010701133113 |        1 | NULL    |
|  7 | Billy Elliott    |    106 | 15   | NULL      | English                               | NULL       | 20010701133113 |        1 | NULL    |
|  8 | Patch Adams      |    111 | 15   | NULL      | English                               | NULL       | 20010701133113 |        1 | NULL    |
|  9 | Toy Story 2      |     89 | U    | NULL      | English                               | NULL       | 20010701133113 |        1 | NULL    |
+----+------------------+--------+------+-----------+---------------------------------------+------------+----------------+----------+---------+
9 rows in set (0.00 sec)

mysql> INSERT INTO dvd(title, length, cert, company, timeseen)
    -> values ("The Perfect Storm", 125, "12", "Warner Bros.", 0);
Query OK, 1 row affected (0.00 sec)

mysql> select * from dvd;
+----+-------------------+--------+------+-----------+---------------------------------------+------------+----------------+----------+--------------+
| id | title             | length | cert | videotype | lang                                  | bought     | changed        | timeseen | company      |
+----+-------------------+--------+------+-----------+---------------------------------------+------------+----------------+----------+--------------+
|  1 | Dinosaur          |   79.5 | PG   | PAL       | English                               | 2001-03-15 | 20010701133113 |        1 | NULL         |
|  2 | Chicken Run       |     81 | U    | PAL       | English                               | 2001-02-07 | 20010701133113 |        1 | NULL         |
|  3 | Groundhog Day     |     97 | PG   | PAL       | English,French,German,Italian,Spanish | 2000-12-08 | 20010701133113 |        1 | NULL         |
|  4 | Erin Brockovich   |    126 | 15   | NULL      | English,German                        | NULL       | 20010701133113 |        1 | NULL         |
|  5 | The Piano         |    115 | 15   | NULL      | NULL                                  | NULL       | 20010701133113 |        1 | NULL         |
|  6 | The Color Purple  |    148 | 15   | NULL      | NULL                                  | NULL       | 20010701133113 |        1 | NULL         |
|  7 | Billy Elliott     |    106 | 15   | NULL      | English                               | NULL       | 20010701133113 |        1 | NULL         |
|  8 | Patch Adams       |    111 | 15   | NULL      | English                               | NULL       | 20010701133113 |        1 | NULL         |
|  9 | Toy Story 2       |     89 | U    | NULL      | English                               | NULL       | 20010701133113 |        1 | NULL         |
| 10 | The Perfect Storm |    125 | 12   | NULL      | English                               | NULL       | 20010701135612 |        0 | Warner Bros. |
+----+-------------------+--------+------+-----------+---------------------------------------+------------+----------------+----------+--------------+
10 rows in set (0.00 sec)
```

The DROP command

DANGER. In real life, if you drop something it breaks. In SQL if you drop something, it is deleted. That can include a complete database including all the tables that it contains. You are not asked for any confirmation, provided that you have the privileges needed to drop a table!

86.9 Other commands

The SHOW command

The **SHOW** command lets you examine and list out the Metadata, and the operation of the SQL system as a whole.

```
mysql> SHOW DATABASES;
+---------------+
| Database      |
+---------------+
| H201          |
| content       |
| demo          |
| entertainment |
| mysql         |
| test          |
+---------------+
6 rows in set (0.00 sec)
```

```
mysql> SHOW STATUS;
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Aborted_clients          | 0     |
| Aborted_connects         | 0     |
| Bytes_received           | 10175 |
| Bytes_sent               | 24083 |
| Connections              | 32    |
| Created_tmp_tables       | 0     |
| Delayed_insert_threads   | 0     |
| Delayed_writes           | 0     |
| Delayed_errors           | 0     |
| Flush_commands           | 1     |
| Handler_delete           | 0     |
| Handler_read_first       | 8     |
| Handler_read_key         | 1     |
| Handler_read_next        | 3     |
| Handler_read_prev        | 0     |
| Handler_read_rnd         | 4     |
| Handler_read_rnd_next    | 351   |
| Handler_update           | 21    |
| Handler_write            | 46    |
| Key_blocks_used          | 8     |
| Key_read_requests        | 40    |
| Key_reads                | 1     |
| Key_write_requests       | 41    |
| Key_writes               | 21    |
| Max_used_connections     | 0     |
| Not_flushed_key_blocks   | 0     |
| Not_flushed_delayed_rows | 0     |
| Open_tables              | 5     |
| Open_files               | 10    |
| Open_streams             | 0     |
| Opened_tables            | 28    |
| Questions                | 250   |
| Slow_launch_threads      | 0     |
| Slow_queries             | 0     |
| Slave_running            | OFF   |
| Threads_cached           | 0     |
| Threads_connected        | 1     |
| Threads_running          | 1     |
| Uptime                   | 63099 |
+--------------------------+-------+
39 rows in set (0.00 sec)

mysql> use entertainment;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker start-up with -A
```

```
Database changed
mysql> SHOW TABLES;
+------------------------+
| Tables_in_entertainment |
+------------------------+
| dvd                    |
+------------------------+
1 row in set (0.00 sec)
```

```
mysql> SHOW COLUMNS FROM dvd;
+-----------+------------------------------------------------------------------------------------------------------------------+------+-----+---------+----------------+-
-------------------------------+
| Field     | Type                                                                                             | Null | Key | Default | Extra          | Privileges                      |
+-----------+------------------------------------------------------------------------------------------------------------------+------+-----+---------+----------------+-
-------------------------------+
| id        | smallint(5) unsigned                                                                                                                 |      | PRI | NULL    | auto_increment |
select,insert,update,references |
| title     | varchar(100)                                                                                    | YES  |     | NULL    |                | select,insert,update,references |
| length    | float                                                                                           | YES  |     | NULL    |                | select,insert,update,references |
| cert      | enum('U','PG','12','12A','15','18','UNCLASS','BANNED')                                                                                  | YES  |     | NULL    |                |
select,insert,update,references |
| videotype | enum('PAL','NTSC','SECAM')                                                                                                          | YES  |     | NULL    |                |
select,insert,update,references |
| lang      | set('English','French','German','Italian','Spanish','Dutch','Polish','Hungarian','Hindi','Finnish','Swedish','Hebrew') | YES  |     | English |                |
select,insert,update,references |
| bought    | date                                                                                            | YES  |     | NULL    |                | select,insert,update,references |
| changed   | timestamp(14)                                                                                   | YES  |     | NULL    |                | select,insert,update,references |
| timeseen  | int(11)                                                                                         | YES  |     | 1       |                | select,insert,update,references |
| company   | text                                                                                            | YES  |     | NULL    |                | select,insert,update,references |
+-----------+------------------------------------------------------------------------------------------------------------------+------+-----+---------+----------------+-
-------------------------------+
10 rows in set (0.00 sec)
```

The **DESC** or **DESCRIBE** is provided only for compatibility with Oracle; the **SHOW** command provides all its functionality and much more.

On limiting selection

Chances are that if you're selecting within an application, you want to receive back and work on all the rows that match. If you're using the mysql client though, you might want to see just a few lines of your result set.

You can add a **limit** clause onto the end of your select command, for example:

LIMIT 5,10

which will give you ten rows back starting at matching row number five.

Notes:

- The first matching row is numbered 0 not 1.
- The second parameter is the number of rows you want, and not the number of the last row you want.
- If the table is updated between two limited selects to give you everything between them, you might not get every match back and/or you may get some duplicates.
- It's not a problem if you ask for more rows than are available.
- If you give a single parameter, you get that many matched rows from the first match (`match 0`)

You can find out the number of matches to a **select** by counting a column. For example:

```
        select (count(Host)) from user where Insert_Priv="Y"
```

86.10 Joining tables when selecting rows

If you have a requirement to select values from one table based on the presence or absence of a value, you can do so using a `table join`. Here's an example.

```
use test;

drop table if exists main;
drop table if exists iused;

create table main (mode text, uses int default  0, c text);
create table iused ( f text );

insert into main (mode, c) values ("Train", "public");
insert into main (mode, c) values ("Bus", "public");
insert into main (mode, c) values ("Car", "private");
insert into main (mode, c) values ("Byke", "private");
insert into main (mode, c) values ("Ferry", "public");

insert into iused (f) values ("Train");
insert into iused (f) values ("Car");
insert into iused (f) values ("Ferry") ;

select * from main left join iused on main.mode = iused.f
        where iused.f is NOT NULL;
```

We've created two tables, one called "main", which describes a number or different types of public transport, and the other called "iused" which lists types that I have used in the past week. I want to select all the information from the main table, but only for those types of public transport that I have used in the iused table.

The **`left join`**, which is an option on the select command, allows me to do this. Here are my results from running the code above:

```
mode    uses    c       f
Train   0       public  Train
Car     0       private Car
Ferry   0       public  Ferry
```

You'll notice that this is an option on the `select` command, so it doesn't help you in updating records directly. With a unique key in the main table, though, it would be easy enough to update a field in each of the records returned by this selection, simply looping through the keys.

Extra use of SELECT

Remember that we told you that **`select`** was very flexible, and went on to illustrate it? We didn't quite show you everything.

`select` can be used to select from multiple tables at the same time; if our example had been extended to include our video collection too in a separate table in the same database, carefully crafted **`SELECT`** statements could be used to give our young visitors a choice of appropriate DVDs, or Videos (or we could even look at the TV schedule too if that's in another table).

`select` can also write its output to a named file. Just as we loaded data in using the LOAD command, so we can get it back out.

As our final action with our DVD table, let's write the data out to a tab-delimited file:

```
mysql> select * INTO OUTFILE "/tmp/showem" FROM dvd;
Query OK, 10 rows affected (0.00 sec)
mysql>
```

No display back to our client, but we do have a data file:

```
1       Dinosaur        79.5    PG      PAL     English 2001-03-15      20010701133113  1       \N
2       Chicken Run     81      U       PAL     English 2001-02-07      20010701133113  1       \N
3      Groundhog Day  97      PG      PAL     English,French,German,Italian,Spanish   2000-12-08      20010701133113  1      \N
4       Erin Brockovich 126     15      \N      English,German  \N      20010701133113  1       \N
5       The Piano       115     15      \N      \N      \N      20010701133113  1       \N
6       The Color Purple        148     15      \N      \N      \N      20010701133113  1       \N
7       Billy Elliott   106     15      \N      English \N      20010701133113  1       \N
8       Patch Adams     111     15      \N      English \N      20010701133113  1       \N
9       Toy Story 2     89      U       \N      English \N      20010701133113  1       \N
10      The Perfect Storm       125     12      \N      English \N      20010701135612  0       Warner Bros.
```

Just as with load, subclasses let us specify different separators, etc.

Other commands

The following commands are also available in SQL; you may wish to use them at a later date as you get more deeply involved with SQL, privileges and larger data tables.

`EXPLAIN`	provides verbose information about a **`SELECT`** command
`FLUSH`	Flushes / resets internal MySQL processes and variables
`GRANT`	Grant privileges
`KILL`	Terminates a thread (get ref. no from **`SHOW PROCESSES`**)
`LOCK`	Locks a table for use by a specific thread
`OPTIMIZE`	Repacks a table, eliminating any wasted space
`REVOKE`	Removes privileges from a user
`SET`	Defines an option setting for the current session
`UNLOCK`	Unlocks all table previously locked by the current session

To find the columns in a table:

`SELECT COLUMNS` from **`tables_name`**

86.11 Some more advanced SQL

Let's have a look at loading some data from file, and connecting that data using joins. We had better start with some data:

file tab1:

```
06:35,Swindon,1
06:51,Chippenham,1
08:46,Swindon,2
09:02,Chippenham,2
14:23,Swindon,3
14:39,Chippenham,3
17:53,Swindon,4
18:08,Chippenham,4
22:01,Swindon,5
22:17,Chippenham,5
```

file tab2:

```
07:01,1
09:11,2
18:18,4
22:27,5
14:49,6
09:21,7
```

86.12 Loading data from file

If you're loading data from file, the file must be on the same computer as the daemon, you must have load privileges, and you'll need to specify the full path to the file. Here's a load example for loading the two data files above:

```
drop database if exists joindemo;
create database joindemo;
use joindemo;
create table leave (
        departs time,
        away text,
        codenum int,
        refno int primary key not null auto_increment);
create table arrive (
        arrives time,
        refno int primary key not null);
load data infile "/home/trainee/coursework/tab1" into table leave
        fields terminated by ','
        (departs, away, codenum);
load data infile "/home/trainee/coursework/tab2" into table arrive
        fields terminated by ','
        (arrives, refno);
```

86.13 More on joining

A regular "join"

If you specify two tables in your **SELECT**, by default you'll get a result set which includes each record of the first table combine with each record of the second table. This can make for a huge result set. For example, if you have tables that you "join" in this way with 6 and 10 rows, the result set will be 6 * 10 = 60 rows.

```
mysqlselect * from leave, arrive;
+----------+------------+---------+-------+----------+-------+
| departs  | away       | codenum | refno | arrives  | refno |
+----------+------------+---------+-------+----------+-------+
| 06:35:00 | Swindon    |       1 |     1 | 07:01:00 |     1 |
| 06:51:00 | Chippenham |       1 |     2 | 07:01:00 |     1 |
| 08:46:00 | Swindon    |       2 |     3 | 07:01:00 |     1 |
| 09:02:00 | Chippenham |       2 |     4 | 07:01:00 |     1 |
| 14:23:00 | Swindon    |       3 |     5 | 07:01:00 |     1 |
| 14:39:00 | Chippenham |       3 |     6 | 07:01:00 |     1 |
| 17:53:00 | Swindon    |       4 |     7 | 07:01:00 |     1 |
| 18:08:00 | Chippenham |       4 |     8 | 07:01:00 |     1 |
| 22:01:00 | Swindon    |       5 |     9 | 07:01:00 |     1 |
| 22:17:00 | Chippenham |       5 |    10 | 07:01:00 |     1 |
| 06:35:00 | Swindon    |       1 |     1 | 09:11:00 |     2 |
| 06:51:00 | Chippenham |       1 |     2 | 09:11:00 |     2 |
| 08:46:00 | Swindon    |       2 |     3 | 09:11:00 |     2 |
| 09:02:00 | Chippenham |       2 |     4 | 09:11:00 |     2 |
| 14:23:00 | Swindon    |       3 |     5 | 09:11:00 |     2 |
| 14:39:00 | Chippenham |       3 |     6 | 09:11:00 |     2 |
| 17:53:00 | Swindon    |       4 |     7 | 09:11:00 |     2 |
| 18:08:00 | Chippenham |       4 |     8 | 09:11:00 |     2 |
| 22:01:00 | Swindon    |       5 |     9 | 09:11:00 |     2 |
| 22:17:00 | Chippenham |       5 |    10 | 09:11:00 |     2 |
| 06:35:00 | Swindon    |       1 |     1 | 18:18:00 |     4 |
| 06:51:00 | Chippenham |       1 |     2 | 18:18:00 |     4 |
| 08:46:00 | Swindon    |       2 |     3 | 18:18:00 |     4 |
| 09:02:00 | Chippenham |       2 |     4 | 18:18:00 |     4 |
| 14:23:00 | Swindon    |       3 |     5 | 18:18:00 |     4 |
| 14:39:00 | Chippenham |       3 |     6 | 18:18:00 |     4 |
| 17:53:00 | Swindon    |       4 |     7 | 18:18:00 |     4 |
| 18:08:00 | Chippenham |       4 |     8 | 18:18:00 |     4 |
| 22:01:00 | Swindon    |       5 |     9 | 18:18:00 |     4 |
| 22:17:00 | Chippenham |       5 |    10 | 18:18:00 |     4 |
| 06:35:00 | Swindon    |       1 |     1 | 22:27:00 |     5 |
| 06:51:00 | Chippenham |       1 |     2 | 22:27:00 |     5 |
| 08:46:00 | Swindon    |       2 |     3 | 22:27:00 |     5 |
| 09:02:00 | Chippenham |       2 |     4 | 22:27:00 |     5 |
| 14:23:00 | Swindon    |       3 |     5 | 22:27:00 |     5 |
| 14:39:00 | Chippenham |       3 |     6 | 22:27:00 |     5 |
| 17:53:00 | Swindon    |       4 |     7 | 22:27:00 |     5 |
| 18:08:00 | Chippenham |       4 |     8 | 22:27:00 |     5 |
| 22:01:00 | Swindon    |       5 |     9 | 22:27:00 |     5 |
| 22:17:00 | Chippenham |       5 |    10 | 22:27:00 |     5 |
| 06:35:00 | Swindon    |       1 |     1 | 14:49:00 |     6 |
| 06:51:00 | Chippenham |       1 |     2 | 14:49:00 |     6 |
| 08:46:00 | Swindon    |       2 |     3 | 14:49:00 |     6 |
| 09:02:00 | Chippenham |       2 |     4 | 14:49:00 |     6 |
| 14:23:00 | Swindon    |       3 |     5 | 14:49:00 |     6 |
| 14:39:00 | Chippenham |       3 |     6 | 14:49:00 |     6 |
| 17:53:00 | Swindon    |       4 |     7 | 14:49:00 |     6 |
| 18:08:00 | Chippenham |       4 |     8 | 14:49:00 |     6 |
| 22:01:00 | Swindon    |       5 |     9 | 14:49:00 |     6 |
| 22:17:00 | Chippenham |       5 |    10 | 14:49:00 |     6 |
| 06:35:00 | Swindon    |       1 |     1 | 09:21:00 |     7 |
| 06:51:00 | Chippenham |       1 |     2 | 09:21:00 |     7 |
| 08:46:00 | Swindon    |       2 |     3 | 09:21:00 |     7 |
| 09:02:00 | Chippenham |       2 |     4 | 09:21:00 |     7 |
| 14:23:00 | Swindon    |       3 |     5 | 09:21:00 |     7 |
| 14:39:00 | Chippenham |       3 |     6 | 09:21:00 |     7 |
| 17:53:00 | Swindon    |       4 |     7 | 09:21:00 |     7 |
| 18:08:00 | Chippenham |       4 |     8 | 09:21:00 |     7 |
| 22:01:00 | Swindon    |       5 |     9 | 09:21:00 |     7 |
| 22:17:00 | Chippenham |       5 |    10 | 09:21:00 |     7 |
+----------+------------+---------+-------+----------+-------+
60 rows in set (0.01 sec)
```

Clearly, this is not usually what you require, so you'll use a `where` clause to select only those records in the result set above where there's a matching key of some sort:

```
mysql> select * from arrive, leave where leave.codenum = arrive.refno;
+----------+-------+----------+------------+---------+-------+
| arrives  | refno | departs  | away       | codenum | refno |
+----------+-------+----------+------------+---------+-------+
| 07:01:00 |     1 | 06:35:00 | Swindon    |       1 |     1 |
| 07:01:00 |     1 | 06:51:00 | Chippenham |       1 |     2 |
| 09:11:00 |     2 | 08:46:00 | Swindon    |       2 |     3 |
| 09:11:00 |     2 | 09:02:00 | Chippenham |       2 |     4 |
| 18:18:00 |     4 | 17:53:00 | Swindon    |       4 |     7 |
| 18:18:00 |     4 | 18:08:00 | Chippenham |       4 |     8 |
| 22:27:00 |     5 | 22:01:00 | Swindon    |       5 |     9 |
| 22:27:00 |     5 | 22:17:00 | Chippenham |       5 |    10 |
+----------+-------+----------+------------+---------+-------+
8 rows in set (0.00 sec)
```

A left join

The join that we've just looked at correctly identified the eight records where the keys matched. Sometimes, we'll want a result set that includes at least one record for every record in one of the incoming tables (let's say the left table) even if there's no corresponding entry in the right table. If we use a **`LEFT JOIN`**, we can do this:

```
mysql> select * from arrive left join leave on leave.codenum =
arrive.refno;
+----------+-------+----------+------------+---------+-------+
| arrives  | refno | departs  | away       | codenum | refno |
+----------+-------+----------+------------+---------+-------+
| 07:01:00 |     1 | 06:35:00 | Swindon    |       1 |     1 |
| 07:01:00 |     1 | 06:51:00 | Chippenham |       1 |     2 |
| 09:11:00 |     2 | 08:46:00 | Swindon    |       2 |     3 |
| 09:11:00 |     2 | 09:02:00 | Chippenham |       2 |     4 |
| 18:18:00 |     4 | 17:53:00 | Swindon    |       4 |     7 |
| 18:18:00 |     4 | 18:08:00 | Chippenham |       4 |     8 |
| 22:27:00 |     5 | 22:01:00 | Swindon    |       5 |     9 |
| 22:27:00 |     5 | 22:17:00 | Chippenham |       5 |    10 |
| 14:49:00 |     6 | NULL     | NULL       |    NULL |  NULL |
| 09:21:00 |     7 | NULL     | NULL       |    NULL |  NULL |
+----------+-------+----------+------------+---------+-------+
10 rows in set (0.00 sec)
```

The additional records that left join creates are "completed" with nulls, so that you can use a check for null on a **`NOT NULL`** field to identify them; great if you're looking for holes in your structure:

```
mysql> select * from arrive left join leave on leave.codenum =
arrive.refno
    -> where leave.codenum is NULL;
+----------+-------+---------+------+---------+-------+
| arrives  | refno | departs | away | codenum | refno |
+----------+-------+---------+------+---------+-------+
| 14:49:00 |     6 | NULL    | NULL |    NULL |  NULL |
| 09:21:00 |     7 | NULL    | NULL |    NULL |  NULL |
+----------+-------+---------+------+---------+-------+
2 rows in set (0.00 sec)
```

In more recent versions of MySQL, there's a right join too.

Joining across databases?

You should try to design your databases so that each database contains the tables that relate to one another, and different databases are used for applications that do not interact. However, you may sometimes find that you need to relate tables across databases, especially as an application matures.

You can join across databases by pre-pending "databasename." in front of the table name. Thus:

```
use fred;
select nearby.id, tom.faraway.hisid from nearby, tom.faraway;
```

Exercise