MySQL Primer

Introduction

MySQL is a free, sql-compliant server and client package that is very fast and robust. There are 3 different databased models: hierarchical, network, and relational. mySQL is based on the relational database model, which nowadays is the dominatn one. The relational model was first formalized by E. F Codd (1970) A Relational Model of Data for Large Shared Data Banks, Vol. 13, No. 6, pp. 377-387. The publication of Codd's rules resulted in a considerable amount of relational database research done in the early 1970s. By 1974, IBM had surfaced with a prototype of a relational database called System/R. The System/R project ended in 1979, but two significant accomplishments are accredited to that project. The relational data model's viability was sufficiently proven to the world and the project included significant work on a database query language. By the end of the System/R project, IBM had implemented a language to create and manipulate relational databases, th Structured English Query Language (SEQUEL). The name later was shortened to Structured Query Language (SQL).

Relational databases are are collection of tables. Each table contains records, which are the horizontal rows in the table. These are also called tuples. Each record contains fields, which are the vertical columns of the table. These are also called attributes. Unlike the hierarchical and network models, in the relational model there are no explicit pointers.

A relation between two tables is defined by creating a common field to the two tables. Relational databases can be used in ways not anticipated by their developers, which is the reason why they are so popular nowadays.

It came from an IBM Research project entitled "SEQUEL'.

The Swedish company MySQL AB writes and maintains the system, selling support and service contracts, as well as commercially-licensed copies of MySQL, and employing people all over the world who communicate over the internet. Two Swedes and a Finn founded MySQL AB: David Axmark, Allan Larsson and Michael "Monty" Widenius. MySQL, as free software, utilises the GNU General Public License.

How MySQL Represents Information

When you first log on to the MySQL server, you will be shown a prompt. After you select a database, you can drop commands to the MySQL prompt. First let's show the databases in this server

```
SHOW DATABASES;
```

The list of databases is probably different on your machine, but the mysql and test databases are likely to be among them. The mysql database is required because it describes user access privileges. The test database is often provided as a workspace for users to try things out. Let's access the test database.

```
 USE test;
```

After we have selected a dataset we can create a table:

```
  CREATE TABLE PERSON(
    person_id INT NOT NULL AUTO_INCREMENT,
    first_name VARCHAR( 100 ),
    last_name VARCHAR( 100 ),
    age INT NOT NULL,
    email_address VARCHAR( 100 ) DEFAULT 'default@default.com',
    PRIMARY KEY( person_id ));
```

MySQL keywords/command syntax are in all caps, and variable names are in lower case. Note that MySQL is *not* case-senstive for command syntax, but *is* case-sensitive for table and field names . The command above will create a table in the database called "PERSON" which has five fields: person_id, first_name, last_name, age, and email_address. A "VARHCAR" datatype is a string that is case-insensitive. In the table above we specified that our VARCHAR types are limited to 100 characters. The default and max length for a VARCHAR is 255 characters. Any string longer than the specified length will be cut after 255 characters. Our "person_id" field is an INT which can never be NULL. It also has the AUTO_INCREMENT flag. AUTO_INCREMENTS are used for PRIMARY KEYs and mean that when you insert a new row of values into a table, fields that have an AUTO_INCREMENT flag will automatically be set to the next greatest value available in the table for that field. MySQL will not allow you to manually enter in duplicate values of PRIMARY KEY types. Our last line in the table definition specifies that person_id is our PRIMARY KEY. If, for example, we had ommited the last line, MySQL would have rejected our command for not specifying a PRIMARY KEY. You can have more than one PRIMARY KEY.

A key is the tool to unlock access to database tables. By knowing the key, we know how to locate specific records, and traverse the relationships between tables. A primary key is the candidate key that has been chosen to identify unique records in a particular table.

Now that we have created our table, it would be useful to put data into that table. We can do so with the INSERT command.

```
  INSERT INTO PERSON VALUES( NULL, "Etienne", "Pelaprat", 20,
    "etienne@markov.ucsd.edu" );
```

The NULL argument tells MySQL that it should generate an appropriate key for this new row of fields and enter it as the person_id. The rest of the parameters are appropriate to the field to which they correspond. Again, all the uppercase words except "PERSON" (which is the name of the table) are MySQL keywords.

If we want to insert a row with partial information

```
  INSERT INTO PERSON (first_name, age)  VALUES("Ian", 29 );
```

The cells in this row without information will show as "NULL".

Select, Show, Describe, Drop, Alter

To see the tables in a database

```
 show tables;
```

To see the fields of the table "PERSON"

```
 show PERSON;
```

To delete a table

```
 drop table PERSON;
```

The example belows add columsnt to a table at the end, begining and after a given column

```
ALTER TABLE person ADD COLUMN education VARCHAR(100) FIRST;

ALTER TABLE person ADD COLUMN modified TIMESTAMP FIRST;

ALTER TABLE person ADD COLUMN married  ENUM( 'yes, 'no') AFTER age;
```

To delete a column

```
ALTER TABLE person  DROP COLUMN education;
```

To delete the database named "test"

```
 drop database test;
```

The most common way to retrieve data from an SQL database is to use the SELECT command and perform a *query*. Here is the most basic command:

```
  SELECT * from PERSON;
```

This command retrieves all the fields of every row in the table PERSON. A query returns an ASCII table of rows from the MySQL prompt. If you wanted to retrieve only certain fields from a query, you could do this:

```
  SELECT first_name from PERSON;
```

. Which would return an ASCII table of all the rows in the table PERSON but would only list the first_name of that person. You could also do:

```
SELECT first_name, last_name from PERSON;
```

It's fun to just do these commands and retrieve a lot of info, but it's often more desired to query the database for a subset of the entire set of information. We can add qualifiers to our queries:

```
SELECT * from PERSON where first_name="Etienne";
```

Would return only the rows from the table PERSON where the first_name is "Etienne." If you wanted to search for people whos names begin with the letter "B", you could do this query:

```
SELECT * from PERSON where first_name like "B%";
```

Notice that we have replaced our '=' sign with a 'like' and that we have added the wildcard character "%". The reason we use 'like' is so that we can use wildcards. Using an '=' sign would treat the '%' as any other character. If we wanted to search for people whos names contained the letter 'e' and who were less than 30 years of age, we could do:

```
SELECT * from PERSON where first_name like "%e%" and age<30;
```

Note that this query would still return fields where the first_name is "Ethan" because the VARCHAR datatype is case-insensitive.

If we just want to see a subset of fields

```
SELECT email_address, first_name from PERSON;
```

Here is another query to get the email address of all people younger than 30 and with ane "e" in their first names.

```
SELECT email_address from PERSON where age < 30 and first_name like ``%e%'';
```

1 Keywords and Operators

- SELECT Retrieves fields from one or more tables.
- FROM Tables containing the fields.
- WHERE Criteria to restrict the records returned.
- GROUP BY Determines how the records should be grouped.
- HAVING Used with GROUP BY to specify the criteria for the grouped records.
- ORDER BY Criteria for ordering the records.
- LIMIT Limit the number of records returned.
- = Equal to
  <> or ! = Not equal to
- < Less than
- <= Less than or equal to
- > greater than
- >= greater than or equal to

- *LIKE* Used to compare strings
- *BETWEEN* Checks for values between a range
- *IN* Checks for values in a list
- *NOTIN* Ensures the value is not in the list
- % may be used as a wildcard
- The underscore character may be used as a placeholder.

  ```
  ``Select * from person wher first_name LIKE ``J___;''
  ```

  searches for a record with first name beginning with J followed by 3 characters.

Data Types

- CHAR(size) Stores up to 255 characters. If the content is smaller than the field size, the content will have trailing spaces appended.
- VARCHAR(size) Stores up to 255 characters, and a minimum of 4 characters. No trailing spaces are appended to the end of this datatype. MySQL keeps track of a delimiter to keep track of the end of the field.
- TINYTEXT Equivalent to VARCHAR(255). TEXT Stores up to 65,535 characters. MEDIUMTEXT Stores up to 16,777,215 characters. LONGTEXT Stores up to 4,294,967,295 characters.
- ENUM ('black', 'white', 'green') Stores up to 65,535 enumerated types. The DEFAULT modifier may be used to specify the default value for this field.
- INT Stores a signed or unsigned integer number. Unsigned integers have a range of 0 to 4,294,967,295, and signed integers have a range of -2,147,438,648 to 2,147,438,647. By default, the INT data type is signed. To create an unsigned integer, use the UNSIGNED attribute. fieldName INT UNSIGNED. The ZEROFILL attribute may be used to left-pad any of the integer with zero's. The AUTO_INCREMENT attribute may be used with any of the Integer data types.
- TINYINT Stores a signed or unsigned byte. Unsigned bytes have a range of 0 to 255, and signed bytes have a range of -128 to 127. By default, the TINYINT data type is signed. MEDIUMINT Stores a signed or unsigned medium sized integer. Unsigned fields of this type have a range of 0 to 1,677,215, and signed fields of this type have a range of -8,388,608 to 8,388,607. By default, the MEDIUMINT data type is signed. BIGINT Stores a signed or unsigned big integer. Unsigned fields of this type have a range of 0 to 18,446,744,073,709,551,615, and signed fields of this type have a range of -9,223,372,036,854,775,808 to 9,223,327,036,854,775,807. By default, the BIGINT data type is signed.
- FLOAT Used for single precision floating point numbers. DOUBLE Used for double precision floating point numbers.
- DATE Stores dates in the format YYYY-MM-DD.
- TIME Stores times in the format HH:MM:SS.
- YEAR(size) Stores the year as either a 2 digit number, or a 4 digit number, depending on the size provided.

- TIMESTAMP(size) Stores dates and times in the format YYYY-MM-DD HH:MM:SS. Automatically keeps track of the time the record was last ammended. The following table shows the formats depending on the size of TIMESTAMP: 2 YY, 4 YYMM, 6 YYMMDD, 8 YYYYMMDD, 10 YYYYMMDDHH, 12 YYYYMMDDHHMM, 14 YYYYMMDDHHMMSS

Designing a Database

A one-to-one (1:1) relationship occurs where, for each instance of table A, only one instance of table B exists, and vice-versa. For example, each vehicle registration is associated with only one engine number, and vice-versa

A one-to-many (1:m) relationship is where, for each instance of table A, many instances of the table B exist, but for each instance of table B, only once instance of table A exists. For example, for each artist, there are many paintings. Since it is a one-to-many relationship, and not many-to-many, in this case each painting can only have been painted by one artist.

A many to many (m:n) relationship occurs where, for each instance of table A, there are many instances of table B, and for each instance of table B, there are many instances of the table A. For example, a poetry anthology can have many authors, and each author can appear in many poetry anthologies.

A relation between two tables is created by creating a common field to the two tables. The common field must be a primary key to the one table (the table that would be the one component of the one-to-many relationship). Consider a relation between a poet table and a poetry anthology table. The relation is of little use if instead of using the primary key from the poet table, code, to create the relationship with the anthology table, we use another field that is not unique, such as the poet's surname. We would never know for certain which poet we're referring to in the poetry anthology. The poet_code field is called the foreign key in the anthology table, which means it's the primary key(code) in the poet table.

1 to N

We create a dataset of students and homeworks. Each student can have multiple homeworks. Each homework is done by a single student, thus stablishing a 1-to-n relationship.

```
drop table if exists student;
drop table if exists homework;

Create table student(
 id int(3) default '0' not null auto_increment,
 first_name varchar(100) default '',
 last_name varchar(100) default '',
 primary key (id)
);

insert into student values (null, 'Tom', 'Smith');
insert into student values (null, 'Peter', 'Pan');
insert into student values (null, 'Susan', 'Neverland');
insert into student values (null, 'Maria', 'Spinoza');

Create table homework(
 id int(3) default '0' not null auto_increment,
 name char(30) default '' not null,
 student_id int(3) not null,
 grade int(1),
 comments varchar(255) default 'no comments',
 key student_id(student_id),
 primary key (id),
);
```

```
insert into homework  values (null, 'Homework 1', '1', 5, null);
insert into homework  values (null, 'Homework 1', '2', 6, null);
insert into homework  values (null, 'Homework 2', '1', 6, null);
```

Below is a query and results

```
mysql> select first_name, last_name, homework.name, grade from student,
homework where student.id = homework.student_id order by student.id;
+------------+-----------+------------+-------+
| first_name | last_name | name       | grade |
+------------+-----------+------------+-------+
| Tom        | Smith     | Homework 1 |     5 |
| Tom        | Smith     | Homework 2 |     6 |
| Peter      | Pan       | Homework 1 |     6 |
+------------+-----------+------------+-------+
```

1.1 N to N

We will develop a three table database. There are two entity tables: "image" stores propoerties of individual images. The second table stores properties of image banks. Each bank has many images and each image may belong to many banks. There is one relationship table: "link" that is used to link the two entity tables.

The "image" table has attributes for imageID, name, width, height, and format. The imageID is be the "Primary Key," an attribute that uniquely identifies each record in the table. The "bank" table has attributes for bankID, name, size, url. It's Primary Key is bankID. The relationship between "images" and "banks" is stored by the "link" table links the student table with the the course table. It has atributes iID and bID, that correspond to the Primary Keys of "image" and "bank". These two keys serve as the joint primary Key for the "link" table.

```
CREATE TABLE image (
  imageID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  width  INT,
  height INT,
  format VARCHAR(10)
);

CREATE TABLE bank (
  bankID INT NOT NULL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  size INT,
  url VARCHAR(100)
);

CREATE TABLE link (
  iID INT NOT NULL,
  bID INT NOT NULL,
  PRIMARY KEY (iID, bID)
);
```

Using MySQL with Matlab

We have installed on markov.ucsd.edu the MySQL extensions for Matlab. Here is a very simple Matlab program that uses the MySQL module:

```
  mysql("open", "localhost", "etienne", "hi.etienne" );
  mysql("use compaq_database");

  A=mysql("select image_id,x1,y1,x2,y2 from BOX where person_id=2");

  result_matrix = zeros( length(A), 5 );

  for k=1:length(A),
          result_matrix(k,:) = [A(k).image_id, A(k).x1, A(k).y1, A(k).x2, A(k).y2 ];
  end;

  mysql('close');
```

The first two lines of this program connect you to the compaq_database. The third lines drops a query to MySQL. The MySQL module for Matlab returns from a query a 1-dimensional array. Each location in the array holds a struct. The fields of the struct correspond to the fields you queryied for in your array. So in the case above, our struct contains the fields: *image_id, x1, x2, y1, y2*. Our for-loop goes through each slot in the array A and inserts the values in each field into a regular Matlab array.

2 History

- The first version of this document was written by Etienne Pelaprat in June 2001. At the time the document was 2 pages long.
- In March 2004 Javier R. Movellan made revisions and added additional materials mostly based on examples from other tutorials found on the Web.
- The document was made open source under the GNU Free Documentation License Version 1.2 on March 2004, as part of the Kolmogorov project.