ILGLabs

Mysql Rdbms

Concepts & Usage

RDBMS

- A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. Most popular commercial and open source databases currently in use are based on the relational database model.
- RDBMS may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables.
- At a minimum, these systems:

  * presented the data to the user as relations (a presentation in tabular form, i.e. as a collection of tables with each table consisting of a set of rows and columns, can satisfy this property)

  * provided relational operators to manipulate the data in tabular form.

- The most popular definition of an RDBMS is a product that presents a view of data as a collection of rows and columns.
- Most commercial relational DBMSes employ SQL as their query language

Primary keys,Foreign keys,Stored procedure

- A primary key uniquely defines a relationship within a database.

  In order for an attribute to be a good primary key it must not repeat. While natural attributes are sometimes good primary keys, Surrogate keys are often used instead. A surrogate key is an artificial attribute assigned to an object which uniquely identifies it (for instance, in a table of information about students at a school they might all be assigned a Student ID in order to differentiate them). The surrogate key has no intrinsic (inherent) meaning, but rather is useful through its ability to uniquely identify a tuple.

- A foreign key is a reference to a key in another relation, meaning that the referencing table has, as one of its attributes, the values of a key in the referenced table. Foreign keys need not have unique values in the referencing relation. Foreign keys effectively use the values of attributes in the referenced relation to restrict the domain of one or more attributes in the referencing relation.

Stored procedure

- A stored procedure is *executable code* that is associated with, and generally stored in, the database.
- Stored procedures usually collect and customize common operations, like inserting a tuple into a relation, gathering statistical information about usage patterns, or encapsulating complex business logic and calculations.
- Frequently they are used as an application programming interface (API) for security or simplicity. Implementations of stored procedures on SQL DBMSs often allow developers to take advantage of procedural extensions (often vendor-specific) to the standard declarative SQL syntax.
- Stored procedures are not part of the relational database model, but all commercial implementations include them.

Indices

- An index is one way of providing quicker access to data.
- Indices can be created on any combination of attributes on a relation.
- Queries that filter using those attributes can find matching tuples randomly using the index, without having to check each tuple in turn. This is analogous to using the index of a book to go directly to the page on which the information you are looking for is found i.e. you do not have to read the entire book to find what you are looking for.
- Relational databases typically supply multiple indexing techniques, each of which is optimal for some combination of data distribution, relation size, and typical access pattern.
- Indices are usually not considered part of the database, as they are considered an implementation detail, though indices are usually maintained by the same group that maintains the other parts of the database.

Normalization

- Normalization was first proposed by Codd as an integral part of the relational model.
- It encompasses a set of best practices designed to eliminate the duplication of data, which in turn prevents data manipulation anomalies and loss of data integrity.
- The most common forms of normalization applied to databases are called the normal forms.
- Normalization trades reducing redundancy for increased information entropy.
- Normalization is criticised because it increases complexity and processing overhead required to join multiple tables representing what are conceptually a single item

Creating a database

CREATE DATABASE publications;

SHOW DATABASES; - for viewing databases

SHOW TABLES; - for viewing the table within database

Now that you've created the database, you want to work with it, so issue:

USE publications;

- Select database();
- Select user();
- Select version();

Creating users

- GRANT ALL ON publications.* TO 'ilg'@'<ipaddress>' IDENTIFIED BY 'ilg007';

- What this does is allow the user ilg@localhost (the localhost is implied by omitting it) full access to the publications database using the password ilg007.

Creating a table

- For creating the above table within publications database,

  mysql> use publications;

- Creating a table called classics

  mysql> CREATE TABLE classics (

  author VARCHAR(128),

  title VARCHAR(128),

  type VARCHAR(16),

  year CHAR(4)) ENGINE MyISAM;

- To check whether your new table has been created, type:

  mysql> DESCRIBE classics;

  mysql> EXPLAIN classics;

A MySQL session: Creating and checking a new table

```
mysql> USE publications;
Database changed
mysql> CREATE TABLE classics (
    -> author VARCHAR(128),
    -> title VARCHAR(128),
    -> type VARCHAR(16),
    -> year CHAR(4)) ENGINE MyISAM;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> DESCRIBE classics;
+--------+--------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| author | varchar(128) | YES |      | NULL    |       |
| title | varchar(128) | YES |       | NULL    |       |
| type   | varchar(16) | YES |       | NULL    |       |
| year   | char(4)      | YES |      | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Describe, Explain

- The DESCRIBE command is an invaluable debugging aid when you need to ensure that you have correctly created a MySQL table. You can also use it to remind yourself about a table's field or column names and the types of data in each one. Let's look at each of the headings in detail:

  Field : The name of each field or column within a table.

  Type : The type of data being stored in the field.

  Null : Whether a field is allowed to contain a value of NULL.

  Key : MySQL supports keys or indexes, which are quick ways to look up and search for data. The Key heading shows what type of key (if any) has been applied.Default The default value that will be assigned to the field if no value is specified when a new row is created.

  Extra : Additional information, such as whether a field is set to auto-increment.

Data types

- VARCHAR stands for VARiable length CHARacter string and the command takes a numeric value that tells MySQL the maximum length allowed to a string stored in this field.
- Both CHAR and VARCHAR accept text strings and impose a limit on the size of the field.The difference is that every string in a CHAR field has the specified size. If you put in a smaller string, it is padded with spaces. A VARCHAR field does not pad the text; it lets the size of the field vary to fit the text that is inserted. But VARCHAR requires a small amount of overhead to keep track of the size of each value. So CHAR is slightly more efficient if the sizes are similar in all records, whereas VARCHAR is more efficient if sizes can vary a lot and get large. In addition, the overhead causes access to VARCHAR data to be slightly slower than to CHAR data.

- Varchar and Char

Data type	Bytes used	Examples
CHAR(*n*)	Exactly *n* (<= 255)	CHAR(5) "Hello" uses 5 bytes
		CHAR(57) "New York" uses 57 bytes
VARCHAR(*n*)	Up to *n* (<= 65535)	VARCHAR(100) "Greetings" uses 9 bytes
		VARCHAR(7) "Morning" uses 7 bytes

The BINARY data type

- The BINARY data type is used for storing strings of full bytes that do not have an associated character set.Use the BINARY data type to store a GIF image.

Data type	Bytes used	Examples
BINARY(*n*) or BYTE(*n*)	Exactly *n* (<= 255)	As CHAR but contains binary data
VARBINARY(*n*)	Up to *n* (<= 65535)	As VARCHAR but contains binary data

The TEXT and VARCHAR data types

- Mysql's text data types

Data type	Bytes used	Attributes
TINYTEXT(*n*)	Up to *n* (<=255)	Treated as a string with a character set
TEXT(*n*)	Up to *n* (<=65535)	Treated as a string with a character set
MEDIUMTEXT(*n*)	Up to *n* (<=16777215)	Treated as a string with a character set
LONGTEXT(*n*)	Up to *n* (<=4294967295)	Treated as a string with a character set

The BLOB data type

- The term BLOB stands for Binary Large OBject and therefore, as you would think, the BLOB data type is most useful for binary data in excess of 65,536 bytes in size. The other main difference between the BLOB and BINARY data types is that BLOBs cannot have default values

Data type	Bytes used	Attributes
TINYBLOB(*n*)	Up to *n* (<=255)	Treated as binary data—no character set
BLOB(*n*)	Up to *n* (<=65535)	Treated as binary data—no character set
MEDIUMBLOB(*n*)	Up to *n* (<=16777215)	Treated as binary data—no character set
LONGBLOB(*n*)	Up to *n* (<=4294967295)	Treated as binary data—no character set

Numeric data types

- Mysql's numeric data type

Data type	Bytes used	Minimum value (Signed/Unsigned)	Maximum value (Signed/Unsigned)
TINYINT	1	−128	127
		0	255
SMALLINT	2	−32768	32767
		0	65535
MEDIUMINT	3	−8388608	8388607
		0	16777215
INT or INTEGER	4	−2147483648	2147483647
		0	4294967295
BIGINT	8	−9223372036854775808	9223372036854775807
		0	18446744073709551615
FLOAT	4	−3.402823466E+38	3.402823466E+38
		(no unsigned)	(no unsigned)
DOUBLE or REAL	8	−1.7976931348623157E+308	1.7976931348623157E+308
		(no unsigned)	(no unsigned)

DATE and TIME

- MySQL's DATE and TIME data types

Data type	Time/date format
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (Only years 0000 and 1901–2155)

The DATETIME and TIMESTAMP data types display the same way.

The main difference is that TIMESTAMP has a very narrow range (from the years 1970 through 2037), whereas DATETIME will hold just about any date you're likely to specify TIMESTAMP is useful, however, because you can let MySQL set the value for you.

If you don't specify the value when adding a row, the current time is automatically inserted.You can also have MySQL update a TIMESTAMP column each time you change a row.

The AUTO_INCREMENT data type

- **Adding the auto-incrementing column id**

  ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;

- This is your introduction to the ALTER command, which is very similar to CREATE. ALTER operates on an existing table, and can add, change, or delete columns. Our example adds a column named id with the following characteristics:

- INT UNSIGNED: Makes the column take an integer large enough for you to store more than 4 billion records in the table.

- NOT NULL: Ensures that every column has a value. Many programmers use NULL in a field to indicate that the field doesn't have any value. But that would allow duplicates, which would violate the whole reason for this column's existence. So we disallow NULL values.

- AUTO_INCREMENT: Causes MySQL to set a unique value for this column in every row, as described earlier. We don't really have control over the value that this column will take in each row, but we don't care: all we care about is that we are guaranteed a unique value.

- KEY: An auto-increment column is useful as a key, because you will tend to search for rows based on this column. This will be explained in the section

- Adding the auto-incrementing id column at table creation

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 type VARCHAR(16),
 year CHAR(4),
 id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE MyISAM;
```

- Removing id column

```
ALTER TABLE classics DROP id;
```

Adding data to a table

- To add data to a table, use the INSERT command.

  INSERT INTO classics(author, title, type, year) VALUES('Mark Twain','The Adventures of Tom Sawyer','Fiction','1876');

  INSERT INTO classics(author, title, type, year) VALUES('Jane Austen','Pride and Prejudice','Fiction','1811');

  INSERT INTO classics(author, title, type, year) VALUES('Charles Darwin','The Origin of Species','Non-Fiction','1856');

  INSERT INTO classics(author, title, type, year) VALUES('Charles Dickens','The Old Curiosity Shop','Fiction','1841');

  INSERT INTO classics(author, title, type, year) VALUES('William Shakespeare','Romeo and Juliet','Play','1594');

- Use the following SQL to see what have you inserted in DB.

  Select * from classics;

Renaming a table

- Renaming a table, like any other change to the structure or meta-information about atable, is achieved via the ALTER command.

```
ALTER TABLE classics RENAME pre1900;
```

- Rename the table back to classics for examples to work.

```
ALTER TABLE pre1900 RENAME classics;
```

- Changing the data type of a column

```
ALTER TABLE classics MODIFY year SMALLINT;
```

Adding a new column

- To add the new column pages, which will be used to store the number of pages in a publication

  ALTER TABLE classics ADD pages SMALLINT UNSIGNED;

- Renaming a column

  ALTER TABLE classics CHANGE type category VARCHAR(16);

- Removing a column

  ALTER TABLE classics DROP pages;

Deleting a table – A test to settle our curiosity

- Creating, viewing, and deleting a table (This is a temp)

  CREATE TABLE disposable(trash INT);

  DESCRIBE disposable;

  DROP TABLE disposable;

  SHOW tables;

Indexes

- As things stand, the table classics works and can be searched without problem by MySQL—until it grows to more than a couple hundred rows, that is.
  - At that point, database accesses will get slower and slower with every new row added, because MySQL has to search through every row whenever a query is issued. This is like searching through every book in a library whenever you need to look something up.
- Indexes are special tables -> unlike normal data tables, are *kept in a specific order. Instead of containing all of the data about an entity.*

  An index contains only the column (or columns) used to locate rows in the data table, along with information describing where the rows are physically located.

  role of indexes is to facilitate the retrieval of a subset of a table's rows and columns without the need to inspect every row in the table.

  All database servers allow you to look at the available indexes.

  MySQL users can use the show command to see all of the indexes on a specific table, as in:

  mysql> SHOW INDEX FROM <table name>;

Creating an Index

- The way to achieve fast searches is to add an index, either when creating a table or at any time afterward.
- Adding indexes to the classics table

```
ALTER TABLE classics ADD INDEX(author(20));

ALTER TABLE classics ADD INDEX(title(20));

ALTER TABLE classics ADD INDEX(category(4));

ALTER TABLE classics ADD INDEX(year);

DESCRIBE classics;
```

Using CREATE INDEX

- An alternative to using ALTER TABLE to add an index is to use the CREATE INDEX command.
- They are equivalent, except that CREATE INDEX cannot be used to create a PRIMARY KEY
- These two commands are equivalent

```
ALTER TABLE classics ADD INDEX(author(20));

CREATE INDEX author ON classics (author(20));
```

Adding indexes when creating tables

Creating the table classics with indexes

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 category VARCHAR(16),
 year SMALLINT,
 INDEX(author(20)),
 INDEX(title(20)),
 INDEX(category(4)),
 INDEX(year)) ENGINE MyISAM;
```

Primary keys

- Populating the isbn column with data and using a primary key (single unique key for each publication to enable instant accessing of a row.)

```
ALTER TABLE classics ADD isbn CHAR(13);

UPDATE classics SET isbn='9781598184891' WHERE   year='1876';

UPDATE classics SET isbn='9780582506206' WHERE   year='1811';

UPDATE classics SET isbn='9780517123201' WHERE   year='1856';

UPDATE classics SET isbn='9780099533474' WHERE   year='1841';

UPDATE classics SET isbn='9780192814968' WHERE   year='1594';

ALTER TABLE classics ADD PRIMARY KEY(isbn);

DESCRIBE classics;
```

Querying a MySQL Database

- SELECT:command is used to extract data from a table.The basic syntax is:

  SELECT something FROM tablename;

  Two different SELECT statements

  ```
  SELECT author,title FROM classics;
  ```

  ```
  SELECT title,isbn FROM classics;
  ```

- SELECT COUNT : Counting rows

  ```
  SELECT COUNT(*) FROM classics;
  ```

DELETE

- When you need to remove a row from a table, use the DELETE command. Its syntax is similar to the SELECT command and allows you to narrow down the exact row or rows to delete using qualifiers such as WHERE and LIMIT.

  Removing the new entry

```
DELETE FROM classics WHERE title='Little Dorrit';
```

WHERE

- The WHERE keyword enables you to narrow down queries by returning only those where a certain expression is true.
- Using the WHERE keyword

```
SELECT author,title FROM classics WHERE author="Mark Twain";

SELECT author,title FROM classics WHERE isbn="9781598184891 ";
```

- Using the LIKE qualifier

```
SELECT author,title FROM classics WHERE author LIKE "Charles%";

SELECT author,title FROM classics WHERE title LIKE "%Species";

SELECT author,title FROM classics WHERE title LIKE "%and%";
```

LIMIT

- The LIMIT qualifier enables you to choose how many rows to return in a query, and where in the table to start returning them. When passed a single parameter, it tells MySQL to start at the beginning of the results and just return the number of rows given in that parameter. If you pass it two parameters, the first indicates the offset from the start of the results where MySQL should start the display, and the second indicates how many to return. You can think of the first parameter as saying, “Skip this number of results at the start.”
- Limiting the number of results returned

  SELECT author,title FROM classics LIMIT 3;

  SELECT author,title FROM classics LIMIT 1,2;

  SELECT author,title FROM classics LIMIT 3,1;

UPDATE...SET

- This construct allows you to update the contents of a field. If you wish to change the contents of one or more fields, you need to first narrow in on just the field or fields to be changed, in much the same way you use the SELECT command.
- Using UPDATE...SET

  UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)' WHERE author='Mark Twain';

  UPDATE classics SET category='Classic Fiction' WHERE category='Fiction';

ORDER BY

- ORDER BY sorts returned results by one or more columns in ascending or descending order.
- Using ORDER BY

  SELECT author,title FROM classics ORDER BY author;

  SELECT author,title FROM classics ORDER BY title DESC;

  The first query returns the publications by author in ascending alphabetical order (the default), and the second returns them by title in descending order.

- f you wanted to sort all the rows by author and then by descending year of publication (to view the most recent first), you would issue the following query:

  SELECT author,title,year FROM classics ORDER BY author,year DESC;

GROUP BY

- In a similar fashion to ORDER BY, you can group results returned from queries using GROUP BY, which is good for retrieving information about a group of data.

  SELECT category,COUNT(author) FROM classics GROUP BY category;

Joining Tables Together

- It is quite normal to maintain multiple tables within a database, each holding a different type of information. For example, consider the case of a customers table that needs to be able to be cross-referenced with publications purchased from the classics table. Creating and populating the customers table

```
CREATE TABLE customers (
    name VARCHAR(128),
    isbn VARCHAR(128),
    PRIMARY KEY (isbn)) ENGINE MyISAM;
```

- Login and connect to DB and issue following.

```
INSERT INTO customers(name,isbn) VALUES('Joe Bloggs','9780099533474');
INSERT INTO customers(name,isbn)  VALUES('Mary Smith','9780582506206');
INSERT INTO customers(name,isbn)  VALUES('Jack Wilson','9780517123201');
SELECT * FROM customers;
```

Joining two tables into a single SELECT

Single Select

```
SELECT name,author,title from customers,classics  WHERE
customers.isbn=classics.isbn;
```

- NATURAL JOIN

Using NATURAL JOIN, you can save yourself some typing and make queries a little clearer.This kind of join takes two tables and automatically joins columns that have the same name.

```
SELECT name,author,title FROM customers NATURAL JOIN classics;
```

- JOIN...ON

If you wish to specify the column on which to join two tables, use the JOIN...ON construct

```
SELECT name,author,title FROM customers JOIN classics ON
customers.isbn=classics.isbn;
```

Using Logical Operators

- You can also use the logical operators AND, OR, and NOT in your MySQL WHERE queries to further narrow down your selections.
- Using logical operators

```
SELECT author,title FROM classics WHERE  author LIKE "Charles%" AND author LIKE "%Darwin";

SELECT author,title FROM classics WHERE  author LIKE "%Mark Twain%" OR author LIKE "%Samuel Langhorne Clemens%";

SELECT author,title FROM classics WHERE  author LIKE "Charles%" AND author NOT LIKE "%Darwin";
```

- Primary Keys: The Keys to Relational Databases
- Normalization:

  The process of separating your data into tables and creating primary keys is called normalization. Its main goal is to make sure each piece of information appears in the database only once. Duplicating data is very inefficient, because it makes databases larger than they need to be and therefore slows down access.

- First Normal Form

  For a database to satisfy the First Normal Form, it must fulfill three requirements:

  1. There should be no repeating columns containing the same kind of data.
  2. All columns should contain a single value.
  3. There should be a primary key to uniquely identify each row.

- The Second Normal Form is all about redundancy across multiple rows. In order to achieve Second Normal Form, your tables must already be in First Normal Form.Once this has been done, Second Normal Form is achieved by identifying columns whose data repeats in different places and then removing them to their own tables.
- Third normal form.....forget it....

When Not to Use Normalization

- Why you should throw these rules out of the window on high-traffic sites. That’s right — you should never fully normalize your tables on sites that will cause MySQL to thrash.
- Normalization requires spreading data across multiple tables, and this means making multiple calls to MySQL for each query. On a very popular site, if you have normalized tables, your database access will slow down considerably once you get above a few dozen concurrent users, because they will be creating hundreds of database accesses between them

Relationships

- MySQL is called a relational database management system because its tables store not only data but the relationships among the data. There are three categories of relationships.
- One to One

Table 9-8a (Customers)

CustNo	Name
1	Emma Brown
2	Darren Ryder
3	Earl B. Thurston
4	David Miller

Table 9-8b (Addresses)

Address	Zip
1565 Rainbow Road	90014
4758 Emily Drive	23219
862 Gregory Lane	40601
3647 Cedar Lane	02154

- **One-to-Many**:One-to-many (or many-to-one) relationships occur when one row in one table is linked to many rows in another table.
- **Many-to-Many** :In a many-to-many relationship, many rows in one table are linked to many rows in another table.

Table 9-8a (Customers)

CustNo	Name
1	Emma Brown
2	Darren Ryder
	(etc...)
3	Earl B. Thurston
4	David Miller

Table 9-7. (Purchases)

CustNo	ISBN	Date
1	0596101015	Mar 03 2009
2	0596527403	Dec 19 2008
2	0596101015	Dec 19 2008
3	0596005436	Jun 22 2009
4	0596006815	Jan 16 2009

Transactions

- In some applications, it is vitally important that a sequence of queries runs in the correct order and that every single query successfully completes. For example, suppose that you are creating a sequence of queries to transfer funds from one bank account to another. You would not want either of the following events to occur:

  • You add the funds to the second account, but when you try to subtract them from the first account the update fails, and now both accounts have the funds.

  • You subtract the funds from the first bank account, but the update request to add them to the second account fails, and the funds have now disappeared into thin air.

Transaction Storage Engines

- In order to be able to use MySQL's transaction facility, you have to be using MySQL's InnoDB storage engine.

  Creating a transaction-ready table

  ```
  CREATE TABLE accounts (
  number INT, balance FLOAT, PRIMARY KEY(number)
  ) ENGINE InnoDB;
  DESCRIBE accounts;
  ```

- Populating the accounts table

  ```
  INSERT INTO accounts(number, balance) VALUES(12345, 1025.50);
  INSERT INTO accounts(number, balance) VALUES(67890, 140.00);
  SELECT * FROM accounts;
  ```

Using BEGIN

- Transactions in MySQL start with either a BEGIN or a START TRANSACTION statement
- ***A MySQL transaction***

```
BEGIN;

UPDATE accounts SET balance=balance+25.11 WHERE number=12345;

COMMIT;

SELECT * FROM accounts;
```

Using COMMIT

- When you are satisfied that a series of queries in a transaction has successfully completed, issue a COMMIT command to commit all the changes to the database.Until a COMMIT is received, all the changes you make are considered to be merely temporary by MySQL. This feature gives you the opportunity to cancel a transaction by not sending a COMMIT but by issuing a ROLLBACK command instead.

Using ROLLBACK

- Using the ROLLBACK command, you can tell MySQL to forget all the queries made since the start of a transaction and to end the transaction.
- A funds transfer transaction

```
BEGIN;

UPDATE accounts SET balance=balance-250 WHERE number=12345;

UPDATE accounts SET balance=balance+250 WHERE number=67890;

SELECT * FROM accounts;
```

- Cancelling a transaction using ROLLBACK

```
ROLLBACK;

SELECT * FROM accounts;
```

Backing Up and Restoring

- **Using mysqldump** :With mysqldump, you can dump a database or collection of databases into one or more files containing all the instructions necessary to recreate all your tables and repopulate them with your data. It can also generate files in CSV (Comma-Separated Values) and other delimited text formats, or even in XML format.

  Drawback

  is that you must make sure that no one writes to a table while you're backing it up. There are various ways to do this, but the easiest is to shut down the MySQL server before mysqldump and start up the server again after mysqldump finishes.

- You can lock the tables you are backing up before running mysqldump. To lock tables for reading (as we want to read the data), from the MySQL command line issue the command:

  ```
  LOCK TABLES tablename1 tablename2 ... READ
  ```

  Then, to release the lock(s), enter:

  ```
  UNLOCK TABLES;
  ```

- By default, the output from mysqldump is simply printed out, but you can capture it in a file through the > redirect symbol.

  The basic format of the mysqldump command is:

  # mysqldump -u root -p database

- Dumping the publications database to screen / Stdout

  # mysqldump -u root -p publications

- **Creating a Backup File :Now that you have mysqldump working, and have verified it outputs correctly to the screen, you can send the backup data directly to a file using the > redirect symbol.**

  # mysqldump -u root -p publications > publications.sql

Working Example

- Dumping just the classics table from publications

  $ mysql -u root -p

  mysql> LOCK TABLES classics READ

  mysql> QUIT

  $ mysqldump -u root -p publications classics > classics.sql

  $ mysql -u root -p

  mysql> UNLOCK TABLES

  mysql> QUIT

- Dumping all the MySQL databases to file

  mysqldump -u root -p --all-databases > all_databases.sql

Restoring from a Backup File

- To perform a restore from a file, call the mysql executable, passing it the file to restore from using the < symbol. So, to recover an entire database that you dumped using the --all-databases option
- Restoring an entire set of databases

```
$ mysql -u root -p  < all_databases.sql
```

- Restoring the publications database

```
$ mysql -u root -p -D publications < publications.sql
```

- Restoring the classics table to the publications database

```
$ mysql -u root -p   -D publications < classics.sql
```

KEEP
CALM
AND
KEEP
PRACTICING