INTRO TO MYSQL

in 24 hours!

TECH SPECS

- 24 hours, lecture/lab format
- Hours: 8:30 - 5:00
- Lunch: 11:45 - 1:00
- Breaks every hour or so.. :)

ABOUT THE INSTRUCTOR

- Nathan Isburgh
  - instructor@edgecloud.com
- Unix user 15+ years, teaching it 10+ years
- Sysadmin, developer, geek
- Forgetful, goofy, patient :)

ABOUT THE COLLEGE

- Rackspace Parking Sticker = good to go
- Breakroom downstairs - labeled “Laundry”
- Sodas - bottles in machine ($1.25) or cans in mini-fridge ($0.50)
- Cafeteria
- Do not speed!
- No smoking anywhere. Can only smoke sitting in car.

ABOUT THE STUDENTS

- Name?
- Time served, I mean employed, at Rackspace?
- Department?
- Unix skill level?
- Why are you taking a class on MySQL and what are you expecting from the experience?

EXPECTATIONS OF STUDENTS

- Basic foundation in computer use
- Ask Questions!
- Complete the labs
- Email if you're going to be late/miss class
- Have fun
- Learn something

```
slideshow.end();
```

MYSQL OVERVIEW

History, Architecture and more

MYSQL

- MySQL (pronounced “My-es-que-el”) is an open source Relational DataBase Management System (RDBMS).

- Michael “Monty” Widenius was the original author of the MySQL database system. It all started back in 1994 when his employer needed a new database system and none of the existing options were acceptable. For a while, MySQL was an internal product.

- In 1996, MySQL 3.11.1 was released publicly in binary forms for Linux and Solaris. Over time, MySQL moved towards a dual-license model, with an open source GNU version and a commercial licensing scheme as well.

MYSQL VERSIONS

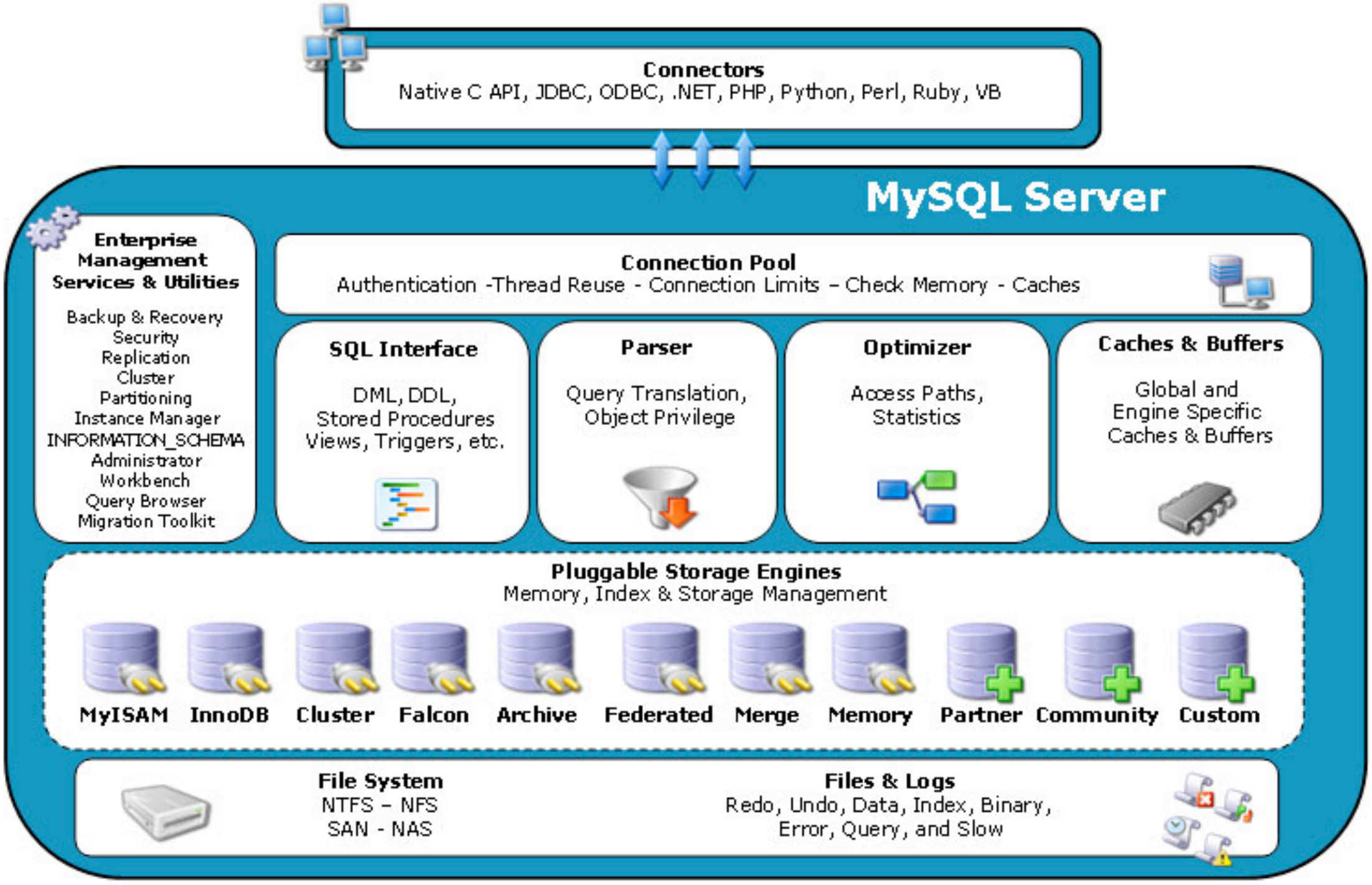
- **3.23** - First major, popular release. c. 2001
- **4.0** - Updates and new features, including Unions. c. 2003
- **4.1** - R-tree, B-trees, subqueries, prepared statements. c. 2004
- **5.0** - Cursors, stored procedures, triggers and more. c. 2005
- **5.1** - Event scheduler, partitioning, plugins and more. c. 2008
- **5.5** - New InnoDB, enhanced tuning, better replication. c. 2010

MYSQL ARCHITECTURE

- MySQL follows a traditional client/server model. The server is in charge of managing all of the data while the clients use/update the data.

- MySQL multi-processing is implemented with threads, providing some benefits and drawbacks; notably better inter-thread communication, but at the cost of more difficult implementation.

- There are many parts to the MySQL system, so a handy picture yanked from the documentation will work well:


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl, Ruby, VB
MySQL Server
Enterprise Management Services & Utilities
Backup & Recovery
Security
Replication
Cluster
Partitioning
Instance Manager
INFORMATION_SCHEMA
Administrator
Workbench
Query Browser
Migration Toolkit
Connection Pool
Authentication -Thread Reuse - Connection Limits – Check Memory - Caches
SQL Interface
DML, DDL,
Stored Procedures
Views, Triggers, etc.
Parser
Query Translation,
Object Privilege
Optimizer
Access Paths,
Statistics
Caches & Buffers
Global and
Engine Specific
Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
Cluster
Falcon
Archive
Federated
Merge
Memory
Partner
Community
Custom
File System
NTFS – NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary,
Error, Query, and Slow

OBTAINING MYSQL

- The MySQL source code is available for download from `www.mysql.com`

- Additionally, most distributions of Linux maintain one or more packages for the MySQL system.

- MySQL is even available to the Microsoft world! If running on Windows, MySQL 5.5 should be strongly considered due to massive overhaul of performance on Windows systems in that version.

COMPILING

- Compiling from source is an attractive option because:
  - Users can tweak advanced settings and features
  - Users can wring extra performance from the server
- Compiling from source cons:
  - Compiling. :)
  - Understanding all of the options and implications to achieve an effective database server for the user's application.

COMPILING

- Compiling is a multi-step, often complicated process.
- Fortunately, MySQL is a very well maintained project, and the process of building it has been finely tuned.
- The short form:
  - `./configure; make; make install`
- Note that as of MySQL 5.5, the build processes utilizes `cmake`.
- In the lab, you will get to try this out

PRE-COMPILED

- Pre-compiled software is an attractive option because:
  - Generally available in package form, therefore bringing all of the benefits of packaged software (dependencies, tracking, upgrades, management)
  - Do not need up front knowledge on MySQL compile time options and settings
- The only real drawbacks to pre-compiled software are:
  - Often not compiled to target user's specific hardware
  - No control over compile time configuration options and features

PACKAGES

- Installing a package is very simple on most distributions of linux:
  - `yum install mysql`
  - `apt-get install mysql`
- Windows users just download the installer and run it.

LAB

1) Experienced Linux users: download the latest source code for MySQL from `www.mysql.com`. Extract and compile the software, but do not install it. Windows users, watch a nearby Linux user while they do this.

2) Everyone: install the mysql packages. This includes at least packages `mysql` and `mysql-server`.

```
slideshow.end();
```

RDBMS BASICS

Tables, rows, databases, oh my!

DATA RELATIONSHIPS

- Why does everyone always talk about data relationships and relational databases? What's all this relationship business?

- Easy! Most information that humans care about is related somehow. One piece of information logically ties to another. Consider:

  - Accounting systems: accounts, transactions, customers

  - Recipes: ingredients, processes, categories

  - Movie collections: Titles, genres, actors, ratings

SO?

- Most requirements to store data can do so by expressing the various entities of information and then expressing the relationships between them. For example:
  - For a given Movie, there will be:
    - A Title
    - One or more Genres
    - One or more Actors
    - etc…

RDBMS'S

- RDBMS's were developed decades ago to manage the underlying storage and organization for all of the information bits and their relationships. Some major examples include:
  - MySQL, Oracle, Postgresql, mSQL, SQLServer
- There is even a special language that was invented to interact with relational databases, known as the Structured Query Language, or SQL. Most RDBMS's implement some form of SQL. Further discussion of SQL to occur in a later lecture.

VOCABULARY TIME!

- **Database**: Highest level container. Holds zero or more tables, views, stored procedures, etc. “Movie Collection”

- **Table**: Primary data storage container. Holds zero or more *rows* of information, each row representing one collection of data bits and relationships. “Titles”

- **Row**: Individual chunk of data, comprised of one or more *columns* of information. “id: 2600, title: Hackers”

- **Column**: One tiny part of data in a row. “Hackers”

WHITEBOARD TIME

- Let's draw some of this out on the whiteboard to get a better idea of what's going on, how relationships are commonly expressed, and how data is stored.

- We will also start sketching out a rough idea for how to organize a movie collection database.

LAB

1) In your own words, define what a table is and it's relationship with columns and rows.

2) On a sheet of paper, and using the sample movie collection database design as a guide, design a simple book collection database, trying to follow some of the ideas discussed in lecture for data reuse and proper organization.

```
slideshow.end();
```

MYSQL BASICS

mysql or mysqld?

MYSQL COMPONENTS

- MySQL is comprised of several components:
  - `mysqld`: The server process, providing most of the functionality expected from an RDBMS.
  - `mysql`: Important command line interface client, connecting to MySQL servers locally or remotely.
  - Libraries/Connectors: There are several MySQL libraries which can be used by developers to connect custom source code to the MySQL system, allowing full access to databases managed by a local or remote `mysqld`.

COMMAND LINE UTILITIES

- `mysql`: Important command line interface client, connecting to MySQL servers locally or remotely.
- `mysqladmin`: Administration tool
- `mysqldump`: Backup tool
- `mysqlcheck`: Table maintenance and repair tool
- `myisamchk`: MyISAM maintenance tool
- `mysqlshow`: View databases, tables and column information

SERVER CONTROL

- Shutting down a database properly is one of the most important steps in avoiding corruption. Consider:
  - Open files, database caches, ongoing transactions, operating system caches, hardware caches
- If the database is not given a chance to flush out caches, write out log messages and close out all operations, any number of nasty corruption issues can spring up.
- Exercising proper startup and shutdown procedures is a must:

SERVER CONTROL

- On Linux systems, a control script should have been included in the package:
  - `/etc/init.d/mysqld`
- To start up or shut down the database, simply run this script with an argument of “start” or “stop”
- Additionally, the `mysqladmin` command can be used by a properly privileged user to shutdown the database.
- In dire cases, sending `SIGTERM` to mysqld will also provide MySQL with notice and the time to shut down cleanly.

CONNECTING TO MYSQL

- In a later lecture, we will discuss users and access control within MySQL. For the time being, we will be logging in to MySQL with the default root account.

- To attach to the MySQL database, simply run:

  - `mysql -p -u root`

- This tells the `mysql` command line client to connect to the locally running `mysqld` process as the root user (`-u`), and prompt for the password (`-p`).

SENDING A QUERY

- Once a connection is established to the server, queries can be sent in for processing. Examples:
  - `SELECT NOW();`
  - `SHOW DATABASES;`
  - `SELECT USER()\G`

LAB

1) Start up your database, connect as root and try out the examples mentioned previously. What is the difference between ending a query with a semicolon or backslash G?

2) Try running "`\h`" or "`\?`". Figure out how to exit from the `mysql` client, then shutdown your database. Verify you can no longer connect via the `mysql` client. Read the documentation for `mysqladmin`, then run "`mysqladmin ping`" and verify your server is shut down.

3) Start up your database once more and check that it's running both with `mysql` and `mysqladmin`.

```
slideshow.end();
```

SQL BASICS

```
SELECT * FROM movie;
```

SQL

- The Structured Query Language is a powerful tool for interfacing with RDBMS's.
- SQL is not the most straightforward language, though, as the syntax is strict, and the wording a little foreign for first time users.
- We will explore SQL by using it to create the Movie database designed earlier, then add and manage some records within the new database.

CREATE

- The first command to cover is the `CREATE` statement.
- `CREATE` can be used to create a number of different RDBMS entities. A few examples:
  - `CREATE DATABASE`
  - `CREATE TABLE`
  - `CREATE VIEW`
  - `CREATE INDEX`

CREATE

- We first need to create a database container for our Movie Collection project:
  - `CREATE DATABASE MovieCollection;`
- The name is case sensitive, so we shall use capital letters to denote each word.
- Database naming is a personal or organizational policy decision. MySQL truly does not care what the name is, so long as it follows the rules laid out in the documentation “Schema Object Names”. Speaking of…

DOCUMENTATION! AND LAB!

- Now is a good time to learn about the vast and excellent documentation available for the MySQL system.
  - `http://dev.mysql.com/doc`
- Learn it, Live it, Love it
- Take some time now to peruse the documentation and familiarize yourself with the layout of the site. Maybe read up on the `mysql` or `mysqladmin` command.
- Find the section on Schema Object Names and browse through it.

USING A DATABASE

- Now that the MovieCollection database has been created, we need to let the `mysql` client know that is the database we want to work on. This is done with the `mysql` client "`use`" command. But first, let's see what database we're currently using:
  - `SELECT DATABASE();`
- Null? That's not what we want...
  - `use MovieCollection;`
  - `SELECT DATABASE();`

CREATE TABLE

- With a working database, we're ready to create our tables.
- Creating tables involves naming the table and describing the various column names and types. Generic syntax:
- ```
  CREATE TABLE table_name (
    column_name  column_type_and_parameters,
    ...
  )
  ```
  ```

MOVIE TABLE

```
CREATE TABLE movie (

    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,

                PRIMARY KEY ( id ),

    title       VARCHAR(50) NOT NULL,

    acquired    DATETIME,

    borrow_id   INT

);
```

LAB

1) Create the rest of the tables for the Movie Collection database.

2) Verify all of your tables by comparing to the Movie Collection database schema handout.

3) If you run into problems, ask me for the schema creation handout for guidance.

INSERT

- Now that we have a database and some tables to work with, let's start adding data! This is accomplished with the `INSERT` statement:
  - `INSERT INTO table_name VALUES ( ... );`
- There are several variations to the `INSERT` statement, so let's take an example insertion and rewrite it in several ways...

INSERT VARIATIONS

- ```
  INSERT INTO movie
  VALUES ( NULL, 'Hackers', NOW(), NULL );
  ```

- ```
  INSERT INTO movie ( title, acquired )
  VALUES ( 'Hackers', NOW() );
  ```

- ```
  INSERT INTO movie
  SET title='Hackers', acquired=NOW();
  ```
  ```

LAB

1) Insert the data presented in the handout for the **movie** and **person** tables. You can ignore the other tables for now, as that's too much tedious typing. :)

   Make sure each row goes in successfully, and if you make a mistake, ignore it for now - in a few slides you'll learn how to correct it.

SELECT

- With data in the database, now the question is how to view it. This is where the `SELECT` statement comes in.
  - `SELECT columns FROM table`
- The `SELECT` statement is a very powerful tool, and there is quite a bit to it. A simple example:
  - `SELECT * FROM movie;`
- This will return all columns for every row in movie

WHERE CLAUSE

- The bare `SELECT` statement is occasionally useful for a full dump of the information in a table, but normally, we are only interested in specific rows. Filtering the result set in a `SELECT` statement is handled via the `WHERE` clause:
  - `SELECT columns FROM table WHERE conditions`
- Example:
  - `SELECT * FROM person WHERE first = 'John';`

WHERE CONDITIONS

Column aliases
Basic operators and functions
Booleans
Basic pattern matching

- The conditions for a `WHERE` clause can be simple or extraordinarily complex.
- Let's spend a few minutes on our live systems to experiment with and discover various `SELECT` statements and `WHERE` clauses.

ORDERING

- `SELECT` statements also support the ability to order results, via the `ORDER BY` clause:
  - `SELECT columns FROM table ORDER BY specification`
- Example:
  - `SELECT * FROM person ORDER BY last ASC;`
- This will return all columns for all rows from the person table ordered by the column ‘last’ in ascending order.

LIMITING RESULTS

- Sometimes a user is only interested in seeing a subset of the results. This is achieved with the `LIMIT` clause:
  - `SELECT columns FROM table LIMIT number`
- Example:
  - `SELECT * FROM person LIMIT 5;`
- This will return 5 rows from the person table (no guarantee on which rows are returned! Why?)

LAB

1) Look through the MySQL documentation on the `SELECT` statement. Can you start to see the power and complexity of this command?

2) How many people in the database have a last name that starts with 'B'? Do this with a proper `SELECT` statement and `WHERE` clause, don't count by hand!

3) Bonus: Create a listing of people where the output is of the form "Last, First Middle". For example: "Jolie, Angelina". Look up the `CONCAT` functions, and possibly `IFNULL`.

UPDATING DATA

- With the foundation of WHERE clauses, the final two major SQL statements can be explored: UPDATE and DELETE.
- To change information in a table, one makes use of the UPDATE statement:
  - `UPDATE table SET name=value... WHERE conditions`
- Example:
  - `UPDATE movie SET acquired=NOW()`

    `WHERE title = 'Tron';`

DELETING DATA

- Finally, to remove rows from a table, use the `DELETE` statement:
  - `DELETE FROM table WHERE conditions`
- Example:
  - `DELETE FROM person WHERE last LIKE 'B%';`
- This will delete every row in person where the column 'last' begins with the letter 'B'.

DELETING STRUCTURES

- When a table, database or any other structure in the RDBMS needs to be completely removed, this is done with the `DROP` statement:
  - `DROP type name`
- Example:
  - `DROP TABLE genre;`
- This will **completely** remove the genre table and all data in it. ***This is irreversible! Be very careful with the* `DROP` *statement!***

LESS TYPING!

- Until this point, we have been typing every command in manually into the mysql client tool. This gets old.
- There is a better way!
  - `source filename`
- This will read "`filename`" as a series of SQL commands and run them in the current mysql session.
- A SQL script!

LAB

1) Drop your MovieCollection database.

2) Go to `http://server1.example.com/mysql`

3) Download "`create-moviedb.sql`" and "`load-moviedb.sql`"

4) Source `create-moviedb.sql`

5) Source `load-moviedb.sql`

6) Perform some queries looking at the data. If you wish to experiment some more with `UPDATE`, `DELETE` and `DROP`, just repeat steps 4 and 5 when you are done.

JOINS

- Now that we have more data in our tables, we can discuss joins.

- A join occurs when data is pulled from more than one table. The rows from each table are then *joined* together somehow, generally by matching values.

- Consider the actor, movie and person tables. What if we wanted to produce a listing of movie title and actor name. Can it be done?

- Of course!

JOINS

```
SELECT movie.title AS Title,
CONCAT_WS( '', person.first, ' ', person.last ) AS Name
FROM movie INNER JOIN actor ON
   actor.movie_id = movie.id
INNER JOIN person ON
   actor.person_id = person.id
ORDER BY Title ASC;
```

- Think about that one for a minute...

JOINS

- The rabbit hole that is joining goes on very, very deep. There are inner joins, outer joins, left and right joins and more.

- A good book and some experimentation is the best way to master these more advanced topics. The MovieCollection database, with some additional rows could serve as an excellent learning ground should you be interested.

LAB

1) Come up with a single SELECT statement which reports each movie title and it's genres.

2) Bonus: come up with a SELECT statement which shows director first and last names and what genre's they have developed.

```
slideshow.end();
```

MYSQL ACCESS CONTROLS

Access Denied!

USERS

- For the entire course up until this point, we have been logging in to the database as user root.
- Generally, this is not desirable behavior for a number of reasons:
  - One user and one password limits database use to one person, as sharing passwords is bad karma.
  - Security - root user by default can do *anything*.
  - Accountability.

CREATING USERS

- To create a new user, use the CREATE USER statement:
  - `CREATE USER account IDENTIFIED BY 'password'`
- Example:
  - `CREATE USER 'moviedba'@'localhost' IDENTIFIED BY 'popcorn';`
- This creates a new account *with no privileges* called "moviedba" with a password of "popcorn", allowed to login from "localhost".

PRIVILEGES

- The various actions that can be performed in MySQL are categorized into privileges, and the ability to perform an action or not is controlled by the privileges that have been granted to a user. Examples of some privileges:

- `SELECT`: Issue SELECT statements
- `INSERT`: Insert new data
- `DROP`: Dropping structures within the database
- `SHUTDOWN`: Allowed to initiate a shutdown of the server

PRIVILEGES

- The privileges granted can be limited in various ways:
  - Global: All databases and tables - think superuser
  - Database: All tables within a specific database
  - Table: A specific table in a specific database
  - Column: Specific columns in a table and database

GRANT

- The `GRANT` statement is used to control privileges:
  - `GRANT privileges ( columns ) ON what TO account`
- Example:
  - `GRANT SELECT ON MovieCollection.* TO 'moviedba'@'localhost';`
- This `GRANT` allows "moviedba" connected from "localhost" to perform `SELECT` statements on all tables in the MovieCollection database.

PRIVILEGE LEVELS

- As mentioned previously, privileges can be granted on several different levels. This is achieved with the *what* parameter to the `ON` clause in the `GRANT` statement:

- `GRANT ALL ON *.* ...`
- `GRANT ALL ON MovieCollection.* ...`
- `GRANT ALL ON MovieCollection.movie ...`
- `GRANT UPDATE ( title ) ON MovieCollection.movie ...`

PRIVILEGE TABLES

- In the mysql database, there are tables which describe the various access controls:
  - `user`: The user accounts and global privileges
  - `db`: Database level privileges
  - `host`: Host level privileges (generally not used)
  - `tables_priv`: Table level privileges
  - `columns_priv`: Column level privileges

LAB

1) Consult the documentation for details on privileges. Read up on the 26 or so privileges available and their meanings. Also, peruse the documentation on the `GRANT` statement.

2) Create a "movieclerk" user which has full select privileges on all tables in MovieCollection, and insert/update/delete privileges on just the actor table in MovieCollection. Test the account.

3) Create a full power "moviedba" user with full access to everything in the MovieCollection database. Test the account.

4) Create a "movieremote" account which has clerk level access from a neighbor machine. Get your neighbor to test the account. Read up on the `mysql` client command to figure out how to do this.

5) Read up on the `REVOKE` command and remove the modification abilities on the clerk accounts. Also, use SET PASSWORD to change the password. Verify.

```
slideshow.end();
```

MYSQL BASIC ADMINISTRATION

Logs? Configs? `mysqladmin`...

MY.CNF

- The main MySQL configuration file is `my.cnf`
- On many Linux systems, when MySQL is installed from package, this file is located in `/etc/my.cnf`
- Users can also have a `.my.cnf` in their home directory, which will be parsed for client options when using the `mysql` command.
- The format for the `my.cnf` file is similar to a windows ini file format. Sections are headed with `[sectionname]` and settings are simple `name=value` pairings.
- `my.cnf` settings are simply command line argument defaults

IMPORTANT MY.CNF SETTINGS

- There are lots of settings available in my.cnf. Some important ones include:
  - `datadir`: Filesystem path to data files
  - `log-error`: Filesystem path for error log file
  - `max_allowed_packet`: Sets maximum packet size for data exchanges between server and client.
- To see all parameters for `mysqld`:
  - `mysqld --help --verbose`

LAB

1) Browse through your global `my.cnf` file. Look up some of the parameters in the MySQL documentation.

2) Set up a user `.my.cnf` file so that you can connect to your MovieCollection database as moviedba automatically, without needing to type a password.

MYSQLADMIN

- `mysqladmin` is a very useful command line administration tool for the MySQL system. Some of the operations an admin can perform include:
  - Database creation/deletion
  - Cache flushing
  - Server shutdown
  - Password management

LAB

1) Use `mysqladmin` to ping your server. An exit code of 0 means the server is running, 1 means it is not. The exit code can be viewed by typing "echo $?". Shutdown your server and ping again. Can you imagine how you could write a simple server monitoring script?

2) Explore the `processlist` subcommand and theorize as to it's output. In another window, connect to the server. Then run the `processlist` subcommand again. See your second session? Use the `kill` subcommand to destroy your second connection. Very useful for runaway queries. We will explore this further in a future lecture.

LOGS

- MySQL maintains a log file with useful information on the server's activity.
- The path name to the log file can be determined by examining the global my.cnf file.
- During a future lab, take a few minutes to examine your log file. Use google for more information on any cryptic log messages

NETWORKING

- By default, the MySQL server will grab every network interface on a machine and listen on port 3306.
- To control the IP addresses MySQL will listen for traffic on, adjust the “bind-address” parameter. Example:
  - `bind-address = 192.168.1.15`
- If this setting is not specified, or if it is set to `0.0.0.0`, MySQL will listen on all interfaces. Note that those are the only options - all interfaces or one.

ROOT PASSWORD RESET

- The class has already covered changing passwords, but how to reset the root password when it is lost?
- There are two ways of doing this:
  - Disable all access controls, restart the server and change it manually. Then restart again with access controls enabled. Not the best solution.
  - Write a short SQL script to reset the password, then add an init-file option to my.cnf.
- Let's see examples of both methods...

INSECURE ROOT PASSWORD RESET

- `/etc/init.d/mysqld stop`
- `mysqld_safe --skip-grant-tables &`
- `mysql mysql`
- `UPDATE user SET password = '' where user = 'root' AND host = 'localhost';`
- `mysqladmin shutdown`
- `/etc/init.d/mysqld start`

SECURE ROOT PASSWORD RESET

- Create a new text file with the following contents:
  - `SET PASSWORD FOR`

    `'root'@'localhost' = PASSWORD('new_password');`
- Name it `/var/lib/mysql/mysql-init`
- Add under `[mysqld]` to `/etc/my.cnf:`
  - `init-file=/var/lib/mysql/mysql-init`
- Restart the `mysqld` service
- Remove the `init-file` line from `/etc/my.cnf`
- Remove `/var/lib/mysql/mysql-init`

LAB

1) Practice resetting the root password on your database using both the secure and insecure method. Verify each time by using a password of “secure” for the secure method, and “insecure” for the insecure method. When you are finished, you can reset the password to whatever you wish.

2) Change your MySQL configuration to not accept any connections from the network, only local connections. Test with a neighbor.

BACKUPS

- All of this data...
- What would happen if someone were to SIGKILL your mysqld process? Or if a hard drive detonated?
- Sad, sad days, that's what!
- So instead of worrying about such things, we *backup* our data. Sometimes weekly, sometimes daily, sometimes constantly..

BACKUPS IN MYSQL

- There are several backup techniques available:
  - `mysqldump`: Tried and true - everything dumped as SQL
  - `mysqlhotcopy`: For MyISAM tables only - faster than `mysqldump`.
  - Filesystem backups/snapshots
  - Replication

MYSQLDUMP

- By far, `mysqldump` is the most common simple backup strategy, as the backup is portable (all just SQL) and fast for most databases.

- `mysqldump` is very easy to use. Often no arguments are necessary besides credentials and the database to dump.

- Once dumped, the backup can be used for replication setup, experimentation, or simply compressed and stored in archive as a point in time backup.

COPYING AND SNAPSHOTS

- Backups can also be made by copying data files. If the database is being stored on a filesystem or device that supports snapshotting, backups can be easily performed with minimal downtime:
  - On the server, execute:

    ```
    FLUSH TABLES WITH READ LOCK
    ```

  - Take a snapshot of the data directory filesystem (or if snapshots are unavailable, start copying data files)
  - Then run:

    ```
    UNLOCK TABLES
    ```

  - Backup from the snapshot at leisure, then destroy it.

REPLICATION

- Replication is common in larger environments, where the cost to run multiple servers is outweighed by the need for instantaneous backups and failover capability.

- Replication involves streaming every change from the master database to all slave databases. In this manner, every database is a complete copy of the master.

- Backups can be easily pulled from slaves without impacting production at all, with the extra advantage that if the master goes down, a slave can take over.

SETTING UP REPLICATION

- Setting up replication is actually quite easy with MySQL. Short form:
  - Enable binary logging; assign unique server id's
  - Stop all writes to Master, grab status and perform backup
  - Restore backup to Slave
  - Point Slave to Master using status information
  - Start replication on Slave. Verify.

BINARY LOGGING AND SERVER ID'S

- To enable the binary log, add a `log-bin` line to `my.cnf`
- Every server must have a unique server identifier specified in the `my.cnf` file:
  - *Master*
    - `[mysqld]`
    - `server-id=5`
    - `log-bin=master`
  - *Slave*
    - `[mysqld]`
    - `server-id=10`

LOCK, STATUS AND BACKUP

- `FLUSH TABLES WITH READ LOCK`
- `SET GLOBAL read_only = ON;`
- `SHOW MASTER STATUS;`
  - Write down log file and log position
- Perform backup
- `UNLOCK TABLES`
- `SET GLOBAL read_only = OFF;`

CREATE SLAVE ACCOUNT ON MASTER

- Create the slave account on the Master:
- ```
  GRANT REPLICATION SLAVE ON *.*
  TO ‘slave’@‘slave-server.mycompany.com’
  IDENTIFIED BY ‘slavepass’;
  ```
- This account will be used by the Slave to connect to the Master to transfer the binary logs.
  ```

RESTORE TO SLAVE

- Restore the Master backup to the Slave server:

- `mysql -u root -p < backup.sql`

SET UP SLAVE

- Point the Slave to the Master:
- `CHANGE MASTER TO`

```
MASTER_HOST = ‘master-server.mycompany.com’,

MASTER_USER = ‘slave’,

MASTER_PASSWORD = ‘slavepass’,

MASTER_LOG_FILE = ‘master.000001’,

MASTER_LOG_POS = 238;
```

START THE SLAVE AND VERIFY

- Start up the Slave threads:
  - `START SLAVE;`
- Verify using:
  - `SHOW SLAVE STATUS\G`

LAB

1) Use `mysqldump` to back up your MovieCollection database. Take a look at the backup file and note how it works. Read the documentation on `mysqldump` and play with some of the options.

2) Once you have a good, solid backup, `DROP` the MovieCollection database and restore from backup. Verify everything is correct.

3) Work with a partner to establish replication from one machine to the other.

```
slideshow.end();
```

STORAGE ENGINES

MyISAM, InnoDB, Archive, Memory

STORAGE ENGINES

- As discussed throughout the course, MySQL tables are all backed to a pluggable storage engine.

- The storage engine defines the features available to the table, as well as the underlying storage and organization of the data.

- There are a number of engines available for use with MySQL, which will be explored in the following slides.

MYISAM

- The MyISAM storage engine was the original engine for all MySQL systems from 3.23 until 5.5, when it became InnoDB.

- MyISAM is a direct descendent of the ancient ISAM table format (deprecated), with a focus on speed.

- MyISAM is non-transactional.

- Transactional? What now?

TO TRANSACTION OR NOT TO TRANSACTION

- **Transactional**
  - Safer from server crash
  - Run multiple changes in a single transaction
  - Rollback your changes if something goes wrong
  - Better write concurrence
- **Non-Transactional**
  - Faster – No transaction overhead
  - Smaller file system footprint
  - Smaller RAM requirements

MYISAM

- Each table is composed of 3 files
  - `table.frm` Format File: Stores table definition
  - `table.MYD` Data File: Stores the data
  - `table.MYI` Index File: Stores index data
- Table files can be moved to another database/server by copying. Should only be performed under set conditions as discussed in backups lecture.
- Supports full text searching and extensive indexing capabilities
- No deadlocks – Uses full table lock for Inserts - at the cost of reduced concurrency.

MYISAM

- Three possible table storage formats:
  - **Fixed**
    - All rows the same size - fast lookups, but takes more space on disk
  - **Variable**
    - Not all rows the same size, so takes less space to store
  - **Compressed**
    - Packed to save space, fast retrieval, **read-only**

MYISAM PROS

- Supports FULLTEXT/Spatial Indexes
- Low filesystem foot prints
- Uses less RAM
- Fast SELECT/INSERT (append) performance
- Maintains an internal row count (COUNT(*) is fast)

MYISAM CONS

- No transactional support
- Table level locking only
- No crash recovery
- Blocking backups
- No support for Foreign Keys

INNODB

- Transactional Engine - Supports COMMIT, SAVEPOINT, ROLLBACK
- By default data/index stored in `$DATADIR/ib_data*` files.
- Like all MySQL Storage engines, there will be a table format file (`.frm`)
- By default the tablespace is shared for all database/tables in server. But tables can be configured to use individual tablespaces (see 13.2.2.1).
  - NOTE: Even with this option the default tablespace `/var/lib/mysql/ibdata` is needed. NEVER delete an `ib_data` file if unsure of what you're doing.
- InnoDB logs are used to store transaction activity. Can be deleted/resized if MySQL is stopped properly.

INNODB

- Full Atomicity, Consistency, Isolation, Durability or **ACID** compliance, for more on ACID go to:
  - `http://en.wikipedia.org/wiki/ACID`
- Auto recovery in case of server crash (Durability)
- Multi-versioning concurrency control (MVCC)
- Row-level locking - Great for WRITE concurrency
- Supports Foreign Keys!

INNODB PROS

- ACID Compliance
- Crash auto recovery
- High Storage Limit (64TB per tablespace!)
- Row-level locking
- Foreign Keys
- MVCC Support
- Clustered Indexes
- Non-blocking online backups

INNODB CONS

- No Fulltext/Spatial indexes
- Higher filesystem footprint (+2x)
- RAM hungry. Performance depends on having large buffers.

MEMORY

- Data and indexes are stored in RAM! Super performance!
- The table has a .frm file on disk. A restart will maintain the table structure but the data/index will be gone.
- Limited by max_heap_table_size setting (default 16M)
- Table level locking
- Cannot contain TEXT or BLOB.
  - NOTE: This is one of the reason why lots of temporary tables can be created on disk when we use a JOIN on tables containing TEXT or BLOB!

MEMORY PROS

- Fast reads/writes
- Support for Hash and Tree indexes

MEMORY CONS

- Volatile data
- Limited size
- No transactions
- Table level locking
- No support for foreign keys
- No TEXT/BLOB fields

ARCHIVE

- Stores large amounts of data without indexes in a very small footprint
- Supports `INSERT` and `SELECT`, but not `DELETE`, `REPLACE`, or `UPDATE`
- Table composed of three files:
  - `.frm` format file
  - `.ARZ` data file
  - `.ARM` meta data
- Uses row level locking
- Rows are compressed on insertion and uncompressed on retrieval
- Designed for efficient archival storage of large amounts of data
- data warehouse applications, data archiving, data auditing

ENGINES

- To view the available engines on a server:
  - SHOW ENGINES;
- To view the engine a table is using:
  - `SHOW CREATE TABLE db.table`
  - `SHOW TABLE STATUS`
  - `SHOW TABLE STATUS LIKE 'db.table'`

USING ENGINES

- To create a table with a non-default storage engine:
  - `CREATE TABLE table ( ... ) ENGINE=engine`
- Example:
  - ```
    CREATE TABLE test (
      testcol INT
    ) ENGINE=InnoDB;
    ```
    ```

CHANGING ENGINES

- It is possible to change the engine of an existing table using the `ALTER TABLE` statement:
  - `ALTER TABLE test ENGINE=MyISAM;`
- Will increase CPU load and I/O latency while running
- Conversion process may fail if:
  - The target engine doesn't support all features used in the original engine.
  - The table data exceeds the capabilities of the new engine.
- In the event of a conversion failure, MySQL will simply continue using the original table engine.

LAB

1) Convert the movie table to InnoDB.

2) Bonus: Refer to sections 13.2.3 and 13.2.5 in the MySQL Reference Manual and carefully add an additional 100M auto-extending tablespace for InnoDB.

3) Consider the possible performance impact of separating out tablespaces as well as index files for InnoDB across multiple disks.

```
slideshow.end();
```

BASIC SERVER PERFORMANCE TUNING

RAID 10!

DB SERVER HARDWARE

- Some best practice considerations for hardware:
  - 64 bit cpu! More memory, more registers!
  - Tons of RAM!
  - 4-8 cpu cores max - more is not always better because of concurrency contention issues, though as MySQL improves, this guideline might change.
  - RAID everywhere.
    - RAID 10 for MySQL data directory - best performance.
    - RAID 1 or 5 for the operating system

64 BIT, EH?

- MySQL is threaded - PAE (Physical Address Extension) doesn't work very well:
  - Each thread in MySQL could only use about 2.5GB of RAM
  - That means global buffers could not be larger than 2.5GB at best.
  - Performance would suffer because of the additional overhead from PAE
- 8 more GPR/SSE registers in 64-bit

RAID COMPARISON

	Min # of Drives	Capacity	Fault Tolerance	Random Reads	Random Writes	Seq. Reads	Seq. Writes
RAID0	1	N	NONE	Good	Fast	Fast	Fast
RAID1	2	1	N-1	Fast	Good	Fast	Good
RAID5	3	N-1	1	OK	Bad	Good	OK
RAID10	4	N/2	1+	Fast	Fast	Fast	Fast

OPERATING SYSTEM

- As mentioned previously, the operating system should preferably be installed on some sort of RAID storage, and in a perfect world separate from the database RAID system.

- Also preferred to use Linux, but if you have to use Windows, so be it. ;)

- As for flavor/version, there isn't a terrible amount of concern here, so long as you have a relatively recent 64 bit kernel and any additional features you need.

ADDITIONAL OS CONSIDERATIONS

- Care should be taken to ensure the database is properly started and stopped with the operating system state.
- Logs should be monitored and rotated as necessary.
- Simple system and database monitoring scripts can be utilized to alert on issues including:
  - Server and database health
  - Low disk space
  - Poor performance

MYSQL SERVER TUNABLES

- There are many, many variables in the MySQL server available for tuning and tweaking.
- With minor adjustments on a handful of these variables, one can often optimize the server to within a few percent of perfect. Beyond this point, additional changes will require extensive benchmarking and analysis to squeeze a minimal amount of additional performance.
- We will cover the major options here; consult a google beyond that. :)
- After this point, database structure and query structure should be scrutinized, which is the topic of the following lecture.

SEEING DEFAULT SETTINGS

- To see all of the default settings for the tuning variables:
  - `/usr/libexec/mysqld --verbose --help`
- This will produce a list of every tunable variable parameter that you can plug in to `my.cnf`, as well as a long table of default values for every setting.
- An **excellent** resource documenting all parameters, including those not available via the command line, is section 5.1.3.

SEEING CURRENT SETTINGS

- To see all of the current settings for the tuning variables:
  - `SHOW [GLOBAL] VARIABLES;`
- This will show the *current* running values for all of the tuning variables for the current session. The `GLOBAL` parameter shows server-wide settings.
- Note that some of these settings can be changed dynamically with:
  - `SET [GLOBAL] name = value;`

FIRST, CHECKING STATUS

- Before tuning values, it is important to consider the server's current *status* and operation metrics.
  - `SHOW GLOBAL STATUS`
    - Across all connections
  - `SHOW SESSION STATUS`
    - Just this connection
- Provides a report with over 250 metrics on server operation!

SERVER STATUS

- `Connections` — Number of new connections established
- `Max_used_connections` — Check if it matches max_connections
- `Threads_cached` — Number of threads in “thread_cache”
- `Threads_connected` — Number of concurrent connections
- `Threads_created` — Threads created (thread cache misses)
- `Threads_running` — Queries currently executing

SERVER STATUS

- `Open_tables`
  - Number of currently open tables
  - Single table opened twice counts as two
  - Check that table_cache is large enough to accommodate open_tables
- `Opened_table`
  - Number of times table was opened (table_cache miss)
  - Check how many opens per seconds are happening:

```
mysqladmin –i 1 –r extended-status | fgrep opened_table
```

SERVER STATUS

- `Slow_queries`
  - Queries considered to be slow (`long_query_time`)
  - Logged in slow query log if it is enabled (discussed in next lecture)

ENGINE STATUS

- `SHOW ENGINE InnoDB STATUS\G`
  - Great way to see what is going on with InnoDB!
  - File IO
  - Buffer Pool
  - Log Activity
  - Row Activity
  - Lock information, deadlocks, transaction status, pending operations, etc.

PERSISTENT VARIABLES

- To change one of the tuning variables permanently, simply put it in `my.cnf` and restart the server:
  - `table_cache = 128`
  - `max_connection = 200`
  - `...`
- Some of the many available tuning variables:

TUNING VARIABLES

- `max_connections`
  - Maximum number of connections to the server.
- `thread_cache_size`
  - Cache up to this many treads after disconnect.
- `table_cache`
  - Number of tables MySQL can keep open at the same time. Closing/opening table is expensive, but it does eat RAM to keep too many open..

TUNING VARIABLES

- There are lots of variables out there (recall section 5.1.3), and fortunately, many, many sample configurations to consider - MySQL even ships with several configuration variations to use as templates.
- There are also literally hundreds of guidelines, white papers, blogs, books and forums to help with tweaking every little parameter.
- Many factors will go into the final configuration, including hardware setup, operating system, database size/use/features/demands and **testing**

REVERSE DNS

- Due to the flexibility of the ACL system, MySQL normally performs a reverse DNS lookup when a client connects, in order to determine the host name. But the lookup process can be slow, resulting in sluggish connection speeds.

- Unfortunately, these lookups are needed if grant tables use host names instead of IPs..

- So the best practice, if possible, is to use IPs, not host names, for MySQL accounts!

- To fully disable these reverse lookups:

  - Set `skip-name-resolve` in `my.cnf`, restart MySQL

LAB

1) Adjust the table cache to 128, thread cache to 32 and disable DNS reverse lookups.

2) Scroll through the status output for your server. Look up the meaning of at least 5 variables that are not immediately apparent to you.

```
slideshow.end();
```

BASIC DATABASE STRUCTURE AND QUERY TUNING

Index, please!

INDEXES

- An index is a fancy way of describing a structure used in a database to speed up searches. Consider:
  - `SELECT account_id FROM account`

```
WHERE balance = 12345.67;
```

- Without an index, the database would have to crawl through every single row in the account table to locate each account with a current balance of 12345.67. This is called a *full table scan*, and it is one of the most expensive database operations out there, to be avoided at almost all costs!

INDEXES

- Fortunately, with an *index*, the database can locate the records many, many times faster (often many orders of magnitude faster)

- How the heck?

- Internally, an index uses one of several different complex computer science data structures to boil down and organize the information, allowing for rapid searches.

- How cool is that?!

WHY BOTHER WITH TABLES THEN?

- You might think “Why don’t we just use indexes instead of tables?”

- Well, interestingly enough, the index won’t actually hold the data in the table - it holds a reference to the *row* that holds the data.

- So while you might build an index on the balance column of the account table, when the database uses that index to quickly find every account with a balance of 12345.67, it’s actually just getting referred back to unique rows in the table which it knows have the right value.

INDEX EVERYTHING?

- Now you might think “Why don’t we index everything? Every column of every table. Index it all so any search I might do will be extremely fast!”
- On the outside, this seems like a fair idea. But, there are several considerations:
- Indexes only help with certain *types* of searches (such as range searches and equality searches)
- Indexes take up disk space
- Indexes have to be updated any time the table is changed
- Indexes can only be created on certain data types in certain engines

WHAT TO INDEX, THEN?!

- Frustrated, you might wonder “Well, what should I index, then?!”
- There is no perfect answer to this question, as it depends on:
  - Types of common queries
  - Data types
  - Size of table
- Rough idea: index identifier fields and consider indexing fields that are regularly searched by equality or range.

CREATING INDEXES

- Creating an index on an existing table is actually quite simple:
  - `CREATE INDEX name ON table ( columns... )`
- Example:
  - `CREATE INDEX bal_idx ON account ( balance );`
- But normally, we create indexes with table creation...

CREATING INDEXES

```
CREATE TABLE account (

    id          INT UNSIGNED NOT NULL,

    balance   DECIMAL( 15, 4 ),

              INDEX ( balance )

)
```

VIEWING INDEXES

- To see the indexes defined on a table:
  - `SHOW INDEX FROM table`
- Try:
  - `SHOW INDEX FROM movie;`

LAB

1) Create indexes on the actor and director tables for the *_id columns. Does it make sense to have an index for each field separately, or one combined index? Justify your answer.

2) Browse the documentation on creating indexes and some of the advanced topics presented therein.

EXPLAIN

- In a DBA's lifelong quest to tune `SELECT` queries, it is often helpful to see what the MySQL optimizer is doing.
- This is what the EXPLAIN command does:
  - `EXPLAIN select_statement`
- Example:
  - `EXPLAIN SELECT * FROM movie;`

EXPLAIN OUTPUT

- Section 7.2.2 covers the output in detail, but in short form:
- **Select type** - categorizes select
- **Table** - name of table being explained
  - The order indicates MySQL's read order
  - Remember, the FROM clause does not specify order
  - The order is chosen by MySQL's built-in optimizer, otherwise can be “hinted”

EXPLAIN OUTPUT

- **Type** - join type; how efficiently MySQL scans, ordered from best to worst:
  - *system* – fastest type
  - *eq_ref* - "=" referenced by a primary key or unique key (1 row)
  - *ref* - "=" by non-unique key (multiple rows)
  - *range* - references by <> or complex ranges
  - *index*
  - *ALL* - full table scan – slowest type

EXPLAIN OUTPUT

- **Possible keys** - which columns the optimizer could have used as indexes
- **Key** - index MySQL actually selected
- **Key length** - used key length in bytes
  - Check expected length is used for multiple column indexes
- **Ref** - the column or constant, key is matched against
  - Nulls in these columns indicate they are targets for improvement

EXPLAIN OUTPUT

- **Rows** - estimation of rows read
- **Extra** – extra information
  - *using index* - A covering index is used – a good thing!
  - *using where* - a where clause is used for filtering
  - *using filesort* - external sort is used
  - *using temporary* - temporary table will be used

EXPLAIN

- The output from EXPLAIN can be analyzed to find:
  - Candidates for indexes, as scans are bad!
  - Possible query rewriting, maybe even breaking into multiple queries in some cases.
  - Playing with join ordering.
- See section 7.2.1 for a preliminary discussion of optimization techniques for EXPLAIN output, and google for further details.

LAB

1) Put together a few SELECT statements, at least one or two including joins. Use EXPLAIN to see how the optimizer would execute your queries.

2) Bonus: Try to create some indexes that improve your execution plan reported by EXPLAIN.

3) Bonus bonus: Write a short script to insert a few hundred thousand fake records into the MovieCollection database. Then experiment with EXPLAIN and indexing to boost search performance.

QUERY PROFILER

- Way beyond a simple EXPLAIN, the query profiler can provide exceptionally detailed metrics on the actual execution of a statement.
  - Shows resource usage of the execution of a query
  - Introduced in mysql-server 5.0.37
  - Run on a per-session basis
  - Stores results in information_schema.PROFILING, a memory table unique to the session which is destroyed at disconnect

ENABLING PROFILING

- To enable the query profiler, just set a session variable:
  - `SET profiling = 1;`
- The profiler has a finite set of queries it can track data on:
  - 15 queries saved by default
  - Max is 100
  - `SET profiling_history_size = 100`

USING THE PROFILER

- To see all stored profiles:
  - `SHOW PROFILES;`
- And to view the metrics on a given query id:
  - `SHOW PROFILE type FOR QUERY query_id`
- Where type is one of the following...

PROFILE TYPES

- **ALL** – all information
- **BLOCK IO** – disk I/O mostly
- **CONTEXT SWITCHES** – (in)voluntary context switches
- **CPU** – CPU time, both system and user
- **IPC** – counts for msgs sent and received
- **MEMORY** – doesn't work
- **PAGE FAULTS** – counts for major/minor page faults
- **SOURCE** – shows functions from source code
- **SWAPS** – swap counts

LAB

1) Enable profiling for your mysql instance.

2) Experiment with a few queries and familiarize yourself with the profiler output. Research a bit into the documentation for the profiler and the current limitations.

ANALYZE

- The `ANALYZE` statement tells the server to re-analyze an index, computing a new key distribution and storing this information in the index summary.
- The key distribution is used by the optimizer in deciding the order for `JOIN`'s connected with something besides a constant.
- What?!
- Yeah, I know. Short of a looong discussion on index design, suffice it to say that performing a periodic `ANALYZE` on a very dynamic table will improve the optimizer's performance.

ANALYZE SYNTAX

- The syntax is simple:
  - `ANALYZE TABLE table`

OPTIMIZE

- As data is deleted and updated in a database, holes start to form in the underlying storage due to the changes.
- Normally, the storage engines continue refilling most of the holes, but this creates a largely fragmented table.
- The `OPTIMIZE TABLE` statement asks the storage engine to clean house:
  - `OPTIMIZE TABLE table`

SLOW QUERIES

- All of this talk about improving query performance.. If only there were some way to easily and quickly identify these slow queries..
- Oh wait, there is! That's what the slow query log is used for!

SLOW QUERIES

- The slow query log is not enabled by default. To enable it, add the following to `/etc/my.cnf` under the `[mysqld]` section:
  - `log-slow-queries=file`
  - `long_query_time=seconds`
- Records all queries taking longer than `long_query_time` to execute. The default is 10 seconds.
- These queries are often prime candidates for optimization!
- Use `mysqldumpslow` to summarize the log file.

LAB

1) Analyze, then optimize all tables in MovieCollection.

2) Enable the slow query log on your server, then simulate some slow queries using the `SLEEP(seconds)` function.

3) Use `mysqldumpslow` to view a report on your "slow" queries.

```
slideshow.end();
```