Database administration tutorial for non-DBAs

Patrick Lambert

[http://dendory.net]

November 7, 2014

Abstract

This tutorial will show you the basics of administering, configuring, querying and troubleshooting SQL databases in a professional environment, and is aimed at people with no formal database administration background who are thrust into that role. We will mostly focus on the MS SQL Server and MySQL engines.

1 Introduction

Database administrators (DBAs) are in high demand these days. With *big data* being what it is, every organization has more and more stuff to index, catalog, process and interact with. Each situation is different, but typically that involves working with databases. Large enterprises will typically hire one or more DBAs to handle the administration. This is good, because that way a single point of contact can be reached and is responsible for the continued operation of the organization's database systems, typically the life blood of the organization. In many places these days, if the database goes down, operations halt across the board.

But this is the ideal situation. In reality, things rarely happen that way. Many organizations aren't large enough to warrant a full time DBA, or they want to save a few bucks and ask a random IT pro to handle databases as a side project, regardless of how crucial that task is. Perhaps that IT person is you, and this is what this tutorial aims to address. For most tasks, you don't need a college degree to work with databases. By the end of this tutorial you should have the skills to select a database, install it, configure it according to your needs, do basic queries and administration tasks, and some troubleshooting.

1.1 Audience

This document was written for non-DBAs because I was in that situation myself. With no formal database administration training, I worked with dozens of different databases throughout my IT career for various projects, from web back-ends to application databases and big data processing, so I decided to write this document for people who are in the same situation, since that seems to happen so often, even in places you would not suspect.

Perhaps you're working as a help desk person, and a project manager comes to you needing a database set up for a new project. Maybe you're training to become an IT professional but the position you apply to also requires some database experience. Or maybe you're not even in the IT field, let's say a financial guru, but suddenly find yourself having to deal with a large amount of data, with no one to help you set up a structured system to manage all that data, and you want to learn how to do it by yourself.

This tutorial assumes no previous database experience. It does however assume some computer knowledge, namely the ability to download and install a software application, and the ability to start and stop system processes or services. We will mostly cover Windows and Linux setups, but this could be applied to any back-end system as well.

1.2 Disclaimer

As I mentioned before, I have no formal database education. This document is based on over 15 years of real world experience and focuses on real scenarios you may encounter yourself. However, I do not claim to be the foremost expert on the subject. All information here is provided as-is, with no warranty of fitness for a particular use. Your mileage may vary.

We will also not cover advanced topics such as database clustering or disaster recovery. If your organization depends on the database running to keep the lights on, you probably want a proper DBA, which is outside the scope of this document.

2 Popular databases

There are two main types of databases: *relational* databases and *integrated*, sometimes called *flat file* databases. The second type is pretty easy to understand and deal with, and more common than you may imagine. These typically consist of a single file that contains the database and is accessed directly by a software application. This could be as simple as a comma-separated values (CSV) file, but the most popular option is SQLite[1]. The web page describes this type of database nicely:

> SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

Many software applications come with their own built-in database, often ending with *.db*. Even your web browser may have a SQLite engine running in the background to store configuration values. It's a very efficient and low impact method for dealing with data. But it does have some drawbacks, namely the fact that you can only access it locally through a library. There's no way to distribute it across the network other than manually copying the file, and its performance is bound by I/O access speeds.

I personally am a big fan of SQLite, but in most real world situations, the type of database you will need to work with is called Relational Database Management Systems (RDBMS). There are many popular options here: Microsoft SQL Server, Oracle Server, MySQL, PostgreSQL, and IBM DB2. In this tutorial, the examples I will give will focus on MS SQL Server and MySQL because these seem to be the most popular options on Windows and Linux respectively. A lot of the same concepts can be applied to other engines however, and you shouldn't underestimate how popular some of the other choices are. For example, there are some industries where almost all large corporations rely on Oracle.

Finally, it's worth noting that a popular new type you may have heard of is the so-called *NoSQL* databases. These are databases that can be accessed using other means than the traditional SQL query language, instead focusing on objects, graphs or documents. They are mostly useful when dealing with big data, and are often found in education fields. However, they are fairly new and hardly ever found in businesses. We won't address this type here.

3 Installation considerations

You may never have to install a database engine. If you're already working as part of an organization, you may find yourself only having to configure it, or create a new table for a project. If that's the case feel free to jump right to the appropriate section. But if you're dealing with a small project, you may have to install the database yourself, and this section will give you some tips on how to do that.

3.1 Hardware resources

Unlike many other software applications, database engines require a large amount of forethought before you launch the *setup* file. Three factors impact the performance of a database: CPU speed, system memory, and storage space. It's almost never the case that a database system goes down in resource requirements, or even stays still. In almost every case, you will want your database to have access to more and more resources as you input more data into it. This is why virtual machines are so useful when it comes to running databases. That way you can increase the resources as need be.

Remember that a relational database is a server, and as such it should be run on proper hardware. The more CPU power you have, the faster your queries will be run. The more memory, the more data can be kept in RAM which also speeds queries up. Also consider ECC memory, as the extra error correction features may ensure better reliability. As for storage, this is a no-brainer, as the more data you put in the database, the more disk space you will need.

The minimum requirements can be found online. For example, Microsoft recommends[2] 6 GB of hard drive space, 1 GB of RAM and a 2 GHz or higher processor. But this is a place you should not skimp on budget. A good starting point for a mid-range database would be a 2 GHz 64 bits processor, 16 GB of RAM, and a SAN storage RAID array. If you do use network storage, then make sure you have a fast networking setup. Gigabit Ethernet cards are becoming common these days.

If however you don't have access to such an expensive storage option, you should really look into a Solid State Drive (SSD) as opposed to traditional SATA drives. This is because databases are more often disk bound than CPU bound. There's a good discussion[3] on the StackExchange site about database threading and performance.

3.2 Cloud vs on-premise

Another area of concern these days is the option of a cloud setup. You don't actually have to own your database server, you can instead spin up a virtual machine on Amazon's AWS, Microsoft Azure or one of the other cloud options. In many cases, that may be the better option. If you're working on a short term project, there's no need to spend a lot of money on hardware. It will likely cost you less to use the cloud. It's also much faster, you can spin up a virtual machine in a few minutes, or even use SQL database cloud options directly, such as Google Cloud SQL.

But the cloud is not always your best bet. If your database will be critical to your organization, then you may not want to rely on a perfect Internet connection. If you need to input or output a lot of data, having the database live in another city may also not be the best for performance. Finally, while costs are low for small databases, cloud prices go up rather fast when dealing with large data sets. You may be better off with a local option if you need to run it for a long time.

3.3 Running setup

Modern databases can be installed rather easily. For open source options like MySQL you can also compile the engine from source, but unless you have specific needs this is usually not worth it. Each Linux distribution has a MySQL package that takes care of installing each part of the engine to the proper location along with providing a good starting configuration. For example, on Ubuntu you can get MySQL installed by typing:

```
sudo apt-get install mysql-server
```

A Windows installation wizard is also available.

If you're just starting and want something quick to follow along, what I would recommend is that you download MS SQL Server Express[5]. It's a free version of SQL Server and comes with the graphical management console. On the download page, make sure you select *Express With Tools*, because if you just get the *Express* download it won't include any client software and you won't be able to access it.

The installation process usually involves setting up the engine, which is just a binary that will accept incoming connections, a library which will handle the queries, support files, and configuration options. Make sure you write down the information you get during setup, things such as the port you need to connect to, the administrator (typically called *sa*, *sysdba* or *root*) user name and password, along with the default database. One caveat here is that on Windows, there are two different authentication methods: Windows integrated and SQL authentication (or mixed, for both methods). In almost all cases, you should never select just Windows integrated, you should always go for a mixed option, so clients that don't support Windows integrated can still connect.

Typically, databases run as system services. So after following the installation wizard, the first thing you should do is go to your services and make sure the database is running. Some engines give you the option to run as a normal process, and not as a service. You rarely need that option, instead running as a service is best. On Windows, you can access services from the *Start* menu, *Run* (or *Search* for Windows 8) and by typing:

```
services.msc
```

On Linux, you can find it in the process list:

```
ps -aux | grep mysql
```

4 Connecting to an instance

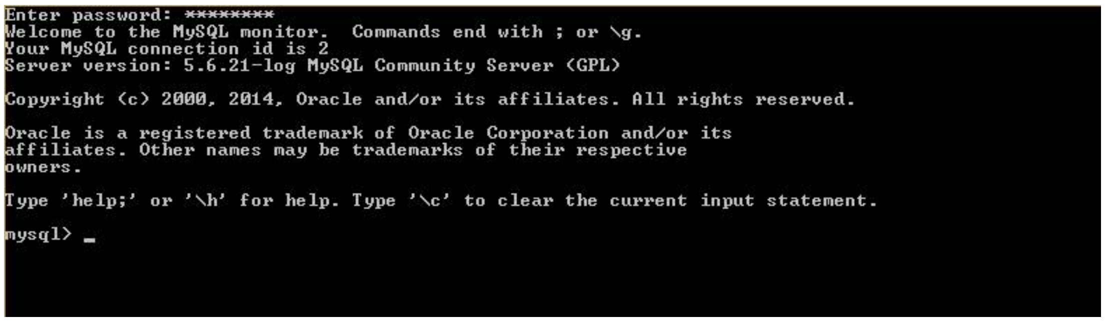
Connecting to your database and running queries is a pretty basic task you will often have to do. Whether you only just finished installing your own database, or if you've been given an IP address and login credentials, you need to be able to connect. Here we will see the two most popular methods for connecting to a database, which is over a network connection, using a MySQL text-based client, and the MS SQL Management Studio.

4.1 MySQL Client

While there are graphical frontends for MySQL, most people run this engine on Linux or Unix systems, and as such use the text-based client. The default client can be started simply by typing:

```
mysql
```

If it's not installed, you will need to find the package for your particular distribution, such as *mysql* or *mysql-client*. By default, it will try to connect to the local host with your current user name, and ask you for the password. If all goes well, you should see:



```
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.21-log MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

The client binary takes several command line options. For example, to connect to the remote host *db.mycorp.local* with the user name *root* (which is the default administrator name) type in:

```
mysql --host=db.mycorp.local --user=root
```

Note that MySQL also includes a special client for administration purposes called *mysqladmin* along with other utilities. If you would like to use a graphical client, I recommend MySQL Workbench[4]. However, since the text client is far more commonly used, we will focus on that.

4.2 MS SQL Management Studio

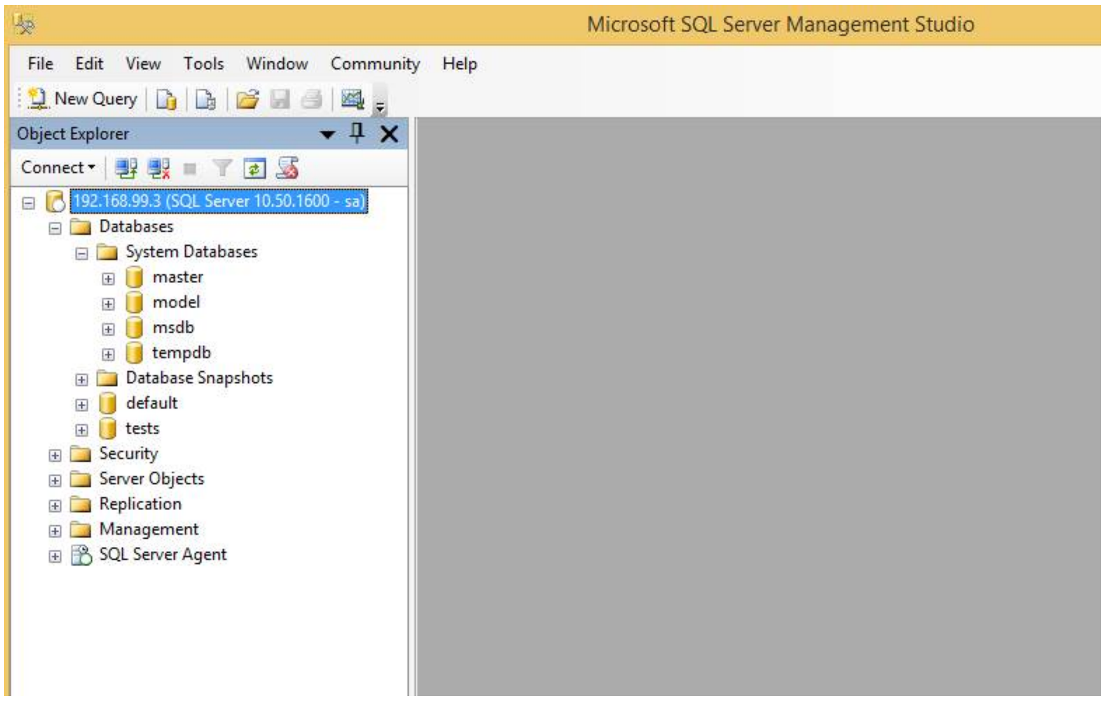
If you've installed SQL Server locally then you have the Management Studio in your Start menu. If not, and you need to connect to a remote server, you can download it from the same link[5] as the SQL Express page, but simply select the *SQL Server Management Studio Express* option, which will only download the client and not the database engine.

Once you start it, first you will need to log in. Simply enter the database server, the user name and password. You can also use Windows integrated authentication if you're on an enterprise domain and whoever set up the database gave you permission. For SQL authentication, the default name for the administrator is *sa*.

One important fact to understand here is that you can run more than one *server instance* on a single machine. When you installed your database, you likely used the default instance name such as *SQLEXPRESS* for MS SQL Express, or *SQL2008* for MS SQL Server 2008. When you enter the server name, you may be able to connect with entering just the host name or IP address, but if more than one instance is running, you should type in both the host and instance. For example, to connect to the *SQLEXPRESS* instance on the local machine, type in:

```
localhost\SQLEXPRESS
```

If all went well, you should be greeted with a list of database objects:



4.3 Alternative connection methods

It's important to understand that most modern databases offer more than just network login options. In fact, there are many ways to interact with a database. An application may access the database's API directly through a library, it may use what's called a named pipe, or it may use the Open Database Connectivity (ODBC)[6] standard. If you're trying to setup a database for a development team, chances are ODBC is actually how they will end up connecting, so you need to be aware of it.

ODBC is a way to access a database through a common protocol. It uses drivers in order to link the application's own code to the database engine. On Windows, you can access those drivers by launching the *odbcad32* binary. There, you will see the installed drivers along with some configuration options. You can install an *ODBC connector* in order to access a database remotely, and that connector will handle all the necessary work for the application. Even a Linux application can connect to a MS SQL Server database using the ODBC connector that Microsoft provides.

5 Creating and selecting databases

Now that you know how to install an engine and connect to it, it's time to learn basic SQL queries. We'll first learn how to list databases, use one, list tables, and query those tables. Each database instance can contain any number of databases. Typically, you will want to create a new database for each project. Each database can then contain any number of tables, which in turn contain columns and rows containing data.

5.1 MySQL

In the MySQL client, you can get a list of databases by typing:

```
SHOW DATABASES;
```

Notice how SQL queries always end with a *;* character, and key words are capitalized, although that isn't a requirement. The databases you're likely to see are all system databases containing system information, users, and so on. To select a database, you will need to type the *USE* keyword with the name of the database:

```
USE mysql;
```

Then, you can list tables with:

```
SHOW TABLES;
```

From there on, you have selected a database and know the table names. You can start using standard SQL queries to interact with that data, which we will do in the next section. First however, let's create a database of our own to run further tests:

```
CREATE DATABASE tests; USE tests;
```

5.2 SQL Server

If you're working with the SQL Server Management Studio, then everything is far more visual. On the left side, you can see the list of databases. You can expand it, and then select a specific database. The system databases are in their own container. Once expanded, you can see a list of tables. System tables are separated from normal ones, and should usually not be touched directly.

The management studio has two ways to interact with data. If you right click on an object on the left pane, you can see a couple of useful functions. For example, right clicking on a table will give you the options to *Select Top 1000 Rows* and *Edit Top 200 Rows*. This can be useful to quickly get an at-a-glance view of the data. If you want to enter a specific SQL query however, simply click the *New Query* button on the toolbar.

Before continuing, let's create a new database for us to use. Right click on the top level *Databases* object and select *New Database*. Call it *tests*, then expand it to see the objects inside of it. You can see quite a few items were made by default, and we'll touch on those later on.

6 SQL Queries

Now that you have your database and know how to select it, you need basic SQL knowledge in order to work with data. If your role is only to create the database for someone else to use, you may not need to do a lot of data manipulation. In fact, more often than not, an application is going to do the actual queries. But it's still important to know basic queries in order to browse the database and troubleshoot any issue.

6.1 Tables and data types

The first thing to understand about tables is that they are two-dimensional. Think of a database table like a spreadsheet, with columns and rows. Before you can store data, you have to define the columns and the type of data that can go in them. This is called the *schema*, or the blueprint of how your data is going to be structured. This is key to SQL databases, and what distinguishes them from NoSQL or other schema-free data storage engines. A traditional RDBMS requires a structure before you can enter data.

When you create a table, one thing you need to define is not only column titles, but also what kind of data these columns will accept. So before making our first table, let's see the more common data types:

Data Type	Description
varchar(n)	Variable string of n characters
char(n)	Fixed string of n characters
varbinary(n)	Variable binary value of n bytes
binary(n)	Fixed binary value of n bytes
tinyint	Integer number of 1 byte
smallint	Integer number of 2 bytes
int	Integer number of 4 bytes
bigint	Integer number of 8 bytes
float	A floating point number
date	A date value formatted `YYYY-MM-DD`
time	A time value formatted `HH:MM:SS`

Note that these data types apply to both MS SQL Server and MySQL, along with most RDBMS servers out there. Of course there are a lot more possible data types, but they vary a lot between engines. For example, the *text* type is actually similar to *varchar(max)* in MySQL, except that the actual data is stored as a reference pointer rather than inline, whereas MS SQL Server has deprecated *text* and recommends against using it.

There are two other useful tidbits of information to know before we make a table. First, tables have indexes such as the *primary key* which speeds up queries. The only constraint is that each value of the key must be unique. Most databases can use an internal structure as primary key, or you can define it yourself. The second piece of information is the concept of a *null* value. When querying a table, we need a way to distinguish between an empty value and whether the value is undefined, and this is where null comes in. You can specify whether a value has to be defined or not when you create your table.

So now that we have some basic data types, let's create a table. If you want to follow along, make sure you have your *tests* database selected and type the following SQL query:

```
  CREATE TABLE users (id int NOT NULL PRIMARY KEY, name varchar(255)
NOT NULL, age smallint, email varchar(255), note varchar(255));
```

Here, we're creating a table named *users* with five columns. The first will have the column name *id*, be of data type *int*, be a primary key for the table, and each row will have to be defined. The next column is called *name* and be a variable length string up to 255 characters. Each row will also require the name to be set. The next column is called *age* and be a 2 bytes integer. The next ones are *email* and *note*. Remember that the order is also important, just like the order of a spreadsheet typically matters.

Let's create one more table, this time containing a list of projects with the columns *title* for the project title, *owner* for the user owning the project, *summary* for a project summary, and finally *due_date* for the date the project is due:

```
  CREATE TABLE projects (title varchar(255) NOT NULL, owner int NOT
NULL, summary varchar(1000), due_date date);
```

If you use the `SHOW TABLES` command again, or refresh the left pane in MS SQL Server Management Studio, you should see your two new tables.

6.2 Inserting data

Now that our structure is done, we can start inserting data. This is done with the `INSERT` keyword, like this:

```
  INSERT INTO users VALUES (1, 'John Doe', 28, 'johndoe@corp.local',
'');
```

This will be our first user with id being *1*, the name is *John Doe*, an age of *28*, an email of *johndoe@corp.local*, and an empty note. As you can see, strings should always be between quotes. Note that you may get an error in SQL Server Management Studio, since Microsoft uses brackets to define objects. You may need to replace `users` for its full name: `[tests].[dbo].[users]`. Also, be sure you use single quotes, as double quotes will not work inside of Management Studio.

But you don't always need to enter every column, you can specify which fields you insert data into. Let's enter a second user:

```
INSERT INTO users (id, name, age) VALUES (2, 'Mary Rose', 36);
```

As you can see, here we add a row but only define a couple of columns. We can do this because the missing rows don't have the `NOT NULL` argument.

6.3 Selecting data

To display the data we entered, we can use the `SELECT` keyword. Here is how you can list everything inside of a table:

```
SELECT * FROM users;
```

Once again, make sure you replace this with the following if you use the SQL Server Management Console:

```
SELECT * FROM [tests].[dbo].[users];
```

If the previous commands were successful, you should see a nice spreadsheet with our data if you use a graphical interface, or the following inside the text-based MySQL client:

```
mysql> SELECT * FROM users;
+----+-----------+------+--------------------+------+
| id | name      | age  | email              | note |
+----+-----------+------+--------------------+------+
|  1 | John Doe  |   28 | johndoe@corp.local |      |
|  2 | Mary Rose |   36 | NULL               | NULL |
+----+-----------+------+--------------------+------+
2 rows in set (0.00 sec)
```

As you can see, both users are displayed. You may also notice the notes are different. For the first user, we entered an empty string, but for the second user, we didn't even specify anything, so it's left undefined. Of course, you can also select only specific columns:

```
SELECT id, name, age FROM users;
```

This can be useful if your table has a lot of columns. Finally, you can also specify conditions with the `WHERE` keyword. The following statement will select only users with an age of 28:

```
SELECT * FROM users WHERE age = 28;
```

This statement will select entries that have an id below 3:

```
SELECT * FROM users WHERE id < 3;
```

This next statement will select users with the name *Mary Rose* aged above 30:

```
SELECT * FROM users WHERE name = 'Mary Rose' AND age > 30;
```

Finally, this statement will select users that have an email defined:

```
SELECT * FROM users WHERE note IS NOT NULL;
```

As you can see, these statements are very similar to the English language and easy to remember. There are quite a bit more permutations we could do, but this should allow you to do basic querying.

6.4 Updating data

You can of course update data already inserted with the `UPDATE` command. Here, we will define a note and email for our second user:

```
UPDATE users SET email = '', note = '' WHERE id = 2;
```

Let's also assign an actual note to all users. This should update both rows since we don't add any condition:

```
UPDATE users SET note = 'Part of the corporation.';
```

6.5 Deleting data

The final task you often find yourself having to do, especially after doing tests, is deleting data from tables with `DELETE`, or entire tables and databases with `DROP`. Obviously, use these commands carefully. This query will delete any entry in our table where the email is empty:

```
DELETE FROM users WHERE email = '';
```

This will delete every entry from the table:

```
DELETE FROM users;
```

This will delete the actual table:

```
DROP TABLE users;
```

And finally, this will delete the entire database:

```
DROP DATABASE tests;
```

There are many more SQL commands out there, but for administration purposes, you now know the basics necessary to create databases, tables, inserting data, updating that data, and querying it. If you're interested in more, I recommend the W3Schools[7] site which contains a full list of SQL commands along with examples.

7 Logins and users

Now that we know how databases work and how to interact with data, we will cover a few useful administration topics that you may have to deal with as a non-DBA database expert, starting with access control.

So far, we've logged in as the administrator to our database, but that is quite dangerous. As you saw, we can delete entire databases with a single command. In normal operation, you want to restrict access more than that. You certainly don't want to give full access to a team if they only need a single database. This is where the concept of users comes in.

Users are handled differently between database engines. MS SQL Server, for example, has the concept of logins for access to the database server, and users, which are specific to a single database. In MySQL, you only create users, but specify where they can connect from. In both cases, you can interact directly with the system tables to add users, but it's preferable to use the designated interface to do it instead.

7.1 MySQL

In MySQL, you create a user with the following command:

```
CREATE USER 'name'@'%' IDENTIFIED BY 'password';
```

This will create a new user with the name *name* and the specified password. Note that here we specify that this user can connect from any host. If you want to restrict access to just a specific IP address, you would replace *%* with the one you want.

Once a user is created, you need to grant access with the `GRANT` command. The following command will grant our new user access to all tables of the *tests* database:

```
GRANT ALL PRIVILEGES ON tests.* TO 'name'@'%';
```

If instead we only want the user to be able to query and update data, but not insert or do other modifications on the database, we can use this statement:

```
GRANT SELECT,UPDATE ON tests.* TO 'name'@'%';
```

Finally, let's see the privileges that the root user has:

```
SHOW GRANTS FOR 'root'@'localhost';
```

You may notice that the root user actually only has access from the local host. Adding proper permissions to users is key to ensure security for your database. If you ever need to modify an existing user, you can do so with the `ALTER USER` command. For example, this command will expire the password:

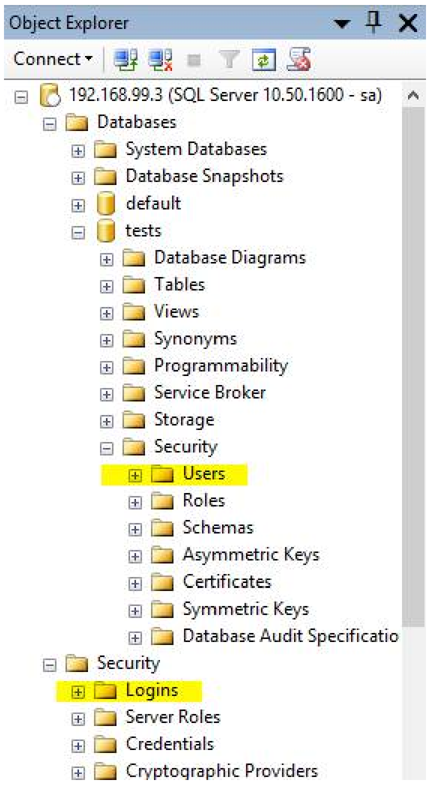
```
ALTER USER 'name'@'%' PASSWORD EXPIRE;
```

This will force the user to enter a new password with the `SET PASSWORD` command or be stuck in restricted mode.

7.2 SQL Server

In SQL Server, logins live under the *Security* top level branch on the left side of the Management Studio. Simply expand the container to see all current logins, and right click on it to create a new login. You will see that there are quite a few options you can define on that dialog. First, you can select whether you need a SQL authentication login, or a Windows integrated login. For SQL authenticated logins you will need to input a password, and then should define the default database.

Once you have a new login, you need to map it to a user. Go to the database you need this user to access, and inside of that container, you will find another container called *Security*. In there, you can find the list of users, or if you right click again, create a new user. Here you enter a name, and select which login is mapped to it. You can also select what permission this user will have such as reading data, writing data, and so on.



8 Server logs

When things break, the first thing you will typically want to do is look at server logs to find out what happened. Obviously, in order to do that you need to know how to access and make sense of them. All RDBMS can write error logs, but they aren't always enabled by default. For example, on Linux MySQL writes logs to the console, which is not particularly helpful. To have log files, you need to specify the file where to send them with the `--logerror=filename` option when you start the server.

On Windows, both MySQL and SQL Server write logs to the Event Viewer. You can also access logs from within Management Studio. These can be useful if the server process crashed, or some other strange behavior occurred with the actual instance. But if you're trying to find out what went wrong with specific queries, either from a user manually typing them in or from an application that failed somehow, then you may need what's called a general SQL query log.

MySQL is able to create a query log with the `--log=filename` option when you start the server. SQL Server however doesn't save them by default, unless you're willing to dig into the server cache, but several third party options exists. A popular one is the SSMS Tools Pack[8].

Another useful type of log is a session log. In SQL Server Management Studio, if you right click on the server name, then select *Reports* and then *Standard Reports*, you can see a list of all activity on the server, such as who connected, which applications, how much time their queries took, and so on.

Finally, as your database system grows, consider running audits. This may allow you to find problems before they happen. SQL Server has audits built into the interface, while MySQL comes with a plugin called *audit_log* that you can use.

9 Backups

When the worst happen, even logs may not help you. That's why you need backups. There are many different ways to do backups. Some people like to do a backup of the entire system (or VM), others like to use specialized tools to back the database up, or even scripts to run `BACKUP` commands. The way you do it is up to you, but there are a few things you should know.

First, backups are pointless if you don't know how to restore them. Never trust a backup solution until you tried restoring it. You can create a test system where you will test your backups and make sure they can be restored promptly. Also, never keep your backups locally. The best scenario is to put them on a network file share, and then also keep a copy off site. Finally, use a scheduling solution, because doing manual backups is not a good idea. Human beings forget.

9.1 MySQL

One of the area where Oracle makes money with MySQL is by offering an Enterprise version that includes more than the basic utilities you get in the community edition. This includes the *MySQL Enterprise Backup*. If you use the free version, you can still do backups using the `mysqldump` utility. This is probably the easiest way to do it, since all the utility does is output a list of `CREATE` and `INSERT` statements that you can use later on to recreate data. This shell command would make a manual backup of the *tests* database:

```
mysqldump -u root -ppassword tests > backup.sql
```

Then, you simply need to import the file as a list of SQL queries with the client in order to restore it:

```
mysql -u root -ppassword tests < backup.sql
```

The advantage of a professional solution is that it can make incremental backups, hot backups, do selective restores, and so on. This is far more flexible than using a script, and if you can spend the money for it, I would suggest doing so.

9.2 SQL Server

In the SQL Server Management Studio, if you expand the *Management* container, you will find the *Maintenance Plans* section. This can be used to set scheduled tasks such as performance indexing, integrity checks, and backups. Right click on it and select *Maintenance Plan Wizard*. This can guide you to creating a backup. You can select to do a full backup, a partial one, or just the transaction log. You can then select a single database or all of them, and when the backup will run. This is a very easy way to make backups of your SQL Server.

If you prefer, you can also do backups of the entire virtual machine. There are many paid options for that, or you can use the features that come with your virtual machine hypervisor. One popular option is Veeam Free Edition[9], but many others exist.

10 Conclusion

In this tutorial, we've covered the basics of administrating a database server, focusing on two of the most popular engines, MySQL and Microsoft SQL Server. The realm of the DBA is of course much more vast, including topics such as scaling, clustering, reporting, disaster recovery, database optimization, server instancing, data profiling, and more. Reading through this document doesn't make you an expert DBA, and won't make you pass any certification, but hopefully by now you realize that handling a database isn't that complicated.

Whether you need to create a database for a personal project, collect data from a team project, handle requests from co-workers, or deal with data in a more structured way, you should now have the skills to select a database engine, install it, query it and administer it. We've reviewed the most important concepts for dealing with databases, and you can certainly stop here, but if you want to expand your knowledge, there are many free online resources, and many sites with a community where you can ask questions such as the DBA StackExchange[10].

Whether you run a single flat-file SQLite database, or a clustered enterprise solution on hundreds of nodes, the same concepts we covered will hopefully help you in your day to day activities.

11 References

References

[1] SQLite: *A self-contained SQL database engine*
http://www.sqlite.org/

[2] MS SQL Server 2014: *Hardware and software requirements*
http://msdn.microsoft.com/en-us/library/ms143506.aspx

[3] StackExchange: *About single threaded versus multithreaded databases performance*
http://dba.stackexchange.com/questions/2918/about-single-threaded-versus-multithreaded-databases-performance

[4] MySQL: *MySQL Workbench Download*
http://dev.mysql.com/downloads/workbench

[5] Microsoft: *Microsoft SQL Server 2014 Express*
http://msdn.microsoft.com/en-ca/evalcenter/dn434042.aspx

[6] Wikipedia: *Open Database Connectivity*
http://en.wikipedia.org/wiki/Open_Database_Connectivity

[7] W3Schools: *SQL Tutorial*
http://www.w3schools.com/sql/

[8] SSMS: *Tools Pack*
http://www.ssmstoolspack.com/

[9] Veeam: *Veeam Backup Free Edition*
http://www.veeam.com/virtual-machine-backup-solution-free.html

[10] StackExchange: *DBA*
http://dba.stackexchange.com/