1 Backup and restore

Backups should be a part of any good IT policy. Backups are included in disaster recovery plans, which are used when critical systems fail. The number of times you create a backup is determined by your own policy. If you would lose too much money from a day's worth of data loss, then you should have daily backups. Smaller companies and startups can get away with fewer backups and even the occasional weekly backup, but this time frame is determined by the business and database administrators.

1.1 Best Practices for Backing up Your Database

It's almost the end of the workday and the employees are cranking out the last of the reports when every computer screen in the office goes dark. Some employees are checking the CPU connections while others are tapping on keyboards to see if something happens. When the database finally comes back online, there are large amounts of data missing. Half the workers cannot continue with their reports due to the missing data and the others will have to spend long weekends re-entering necessary data for office operations.

1.1.1 Developing Database Backup Procedures

The database is the heart of your computing operations. Having current backup procedures in place ensures that accurate information is saved and can be retrieved quickly to minimize downtime. Your database administrator needs to create effective backup and recovery plans in case any failure occurs to the hardware or software of the database. Best practices for Oracle database backups should involve 5 key aspects:

- Figure out what should be backed up in the database
- Deciding the types of backups to perform
- Choosing where to store the backups
- Creating a schedule for the backups
- Establishing a backup retention policy

1.1.2 Components to Back Up

Database administrators working with Oracle systems can select the top components to back up that can restore operations in case of a failure. Passwords, operating system software, relational database management system (RDBMS), and application software are all typical software systems to focus on. Specific Oracle database components to take note during the backup include the control files; redo log files, parameter file, network files, data files and password files.

1.1.2.1 *Type of Backups and Storage*

There are three types of backups that database administrators can perform: offline backups, online backups and logical backups. It is ideal to back up the database to disk and then transfer the disks to tape. Storing the tapes at a secure offsite location ensures that the data information can be retrieved if something monumental happens to the database and systems at the primary office location.

1.1.2.2 *Scheduling Backups*

Creating an appropriate backup schedule should not hamper daily activities for database users while ensuring that the backup disks and tapes hold the most current information. An ideal backup schedule would involve setting up transaction log backups every few hours, incremental backups during the week and full database backups once a week during the appropriate time when the database is the most inactive.

1.1.2.3 *Backup Retention Policies*

There will always be certain data information that becomes outdated and no longer required for the operations of the business. When setting up a service level agreement (SLA) with a database administrator, specify the length of time to retain the old backup information until the time comes to delete and replace it with the new backup data. You can decide between a time frame of months or years for the backup retention policy.

1.1.2.4 *Managing and Testing Backups*

While we would like to think that scheduled automated backup processes will perform correctly, minor glitches can occur when the backup procedure fails. Proper management of database backup procedures requires the database administrator to monitor and validate backups. Setting up alerts about failed backup processes allows for immediate remediation.

In addition, just because the system claims that the backup occurred does not mean that some type of error happened to cause the data to become unreadable during a system restore task. Testing and validating that database backups can be successfully restored to systems is another best practice that all database administrators should perform.

1.2 MYSQLDUMP: DUMPING A DATABASE AS SQL STATEMENTS

MySQL has an internal command named "mysqldump" that lets you create a full backup using SQL statements. None of your constraints, triggers, or primary keys are preserved. You only have the data. What happens when you need to restore the database schema? This is where the mysqldump command is useful.

You can create scripts for a database, a table or all databases on your MySQL server. We'll use the sampledatabase as an example, and we'll assume that this database has a Customer and an Order table.

Suppose we want to dump the sampledatabase as a full copy including SQL scripts. The following statement exports all tables and data to an sql file.

```
mysqldump -u root -p$sw0rdf1sh sampledatabase > db_backup.sql
```

In the above statement, we specify a user name and password. We are using root to dump the database, but other users with administrative rights can also export the database. The sampledatabase is scripted as SQL statements to the file named db_backup.sql.

The file looks similar to the following output.

```
--
-- Table structure for table `Customer`
--

DROP TABLE IF EXISTS ` Customer`;
CREATE TABLE ` Customer` (
`customer id` int NOT NULL AUTO_INCREMENT,
`first_name` varchar(36) default NULL,
`last_name` varchar(36) default NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
--
-- Dumping data for table `Customer `
--
LOCK TABLES `Customer` WRITE;
INSERT INTO `Customer` VALUES (1, 'John', 'Smith');
```

This is just a small example. We only included the Customer table schema and 1 record. If this was a full mysqldump, you would have all tables constructed and all data using INSERT statements.

One part of this dump file to note is the DROP TABLE statement. If the table exists, the table is first dropped and then recreated. This means that you don't want to accidentally run this file on a production database unless you're absolutely sure that you want the table to be recreated. The data restored is only from the last backup, so you can lose data and crash a production application if you run this script in error.

Also notice the LOCK TABLES command. If you recall from previous chapters, you can lock tables from other sessions as long as you only work with the specified table. In this statement, the Customer table is locked for you to insert data into the table.

The above example only dumps tables from one database, but you often need to dump every database and corresponding tables associated with each database.

Before you run a dump on all databases, you should take a look at the list of databases. Use the following command to view a list of databases.

```
SHOW DATABASES;
```

You'll see output similar to the below output.

```
Databases
---------------------------------------
information_schema
mysql
sampledatabase
```

The sampledatabase is our own. Since database preferences, configurations, users and privileges are stored in system databases, you should occasionally back them up. You don't need to back them up as regularly as application tables, but they should be stored at least once a month or as often as you make changes to the system.

Now that you know three tables need backups, you can use the "all databases" option in the mysqldump command. This tells MySQL to export all table schemas and data to an external sql file. The following command exports all databases.

```
mysqldump -u root -p$sw0rdf1sh --all-databases > db_backup.sql
```

Notice that a double dash is what indicates the option. This is somewhat counter intuitive since double dashes indicate comments.

You can also choose to back up several databases in one dump file. While this is convenient, it shouldn't be used for large databases. Since the schema and data are dumped into one file, the file can enlarge to several megabytes in size. This could prove a problem if you need to open the file in a text editor to review contents.

To export data for only specific tables, use the following command.

```
mysqldump -u root -p$sw0rdf1sh --databases sampledatabase mysql>
db_backup.sql
```

In the above statement, only the sampledatabase and mysql databases are exported. They are both exported to the db_backup.sql file. You need the "--databases" option to specify database names one-by-one. Just like the "all databases" option, the databases option also needs a double dash to trigger MySQL.

We mentioned that exporting multiple databases could create a large file that could cause problems. It could even be too large when you export the file and take up too much hard drive space. To get around this issue, mysqldump also has a compression option. MySQL uses Gzip to compress the file to a more manageable size. It doesn't help if you want to review the file content in a text editor, but it does minimize the size of the file when you have limited storage space.

The following command shows you how to compress your backup files.

```
mysqldump -u root -p$sw0rdf1sh --databases sampledatabase mysql | gzip >
db_backup.sql.gz
```

The pipe character and gzip command should come before the greater than symbol. MySQL first compresses the data and then stores it into a compressed file with the gz extension. You'll need an external Gzip extractor to open the file and review content outside of the MySQL database engine.

1.3 Loading Content from the MySQLDump File

If your database crashes, you'll need to import schema and data from the sql file created by mysqldump. Remember, this file uses DROP commands, so you should only use it if you're positive that you need to recreate the database structure. You can also open the file and copy only the queries that import specific tables. You can choose this option if just a few of your tables are corrupted, and you don't need to recreate the entire database.

Importing from a mysqldump file takes a huge amount of resources, and locks tables during the process. For this reason, you shouldn't use this command during busy hours when customers or employees are working with database data. You also want to be sure that you have the latest backup, because the table's data is replaced. Any data you stored between the last database backup and the time you restore information is lost. For this reason, importing from the mysqldump command is usually the last resort when a database crashes.

Let's assume we just need to restore our custom database named sampledatabase. We're importing from the db_backup.sql file created by the mysqldump command. Take a look at the following command.

```
mysql -u root -p -h localhost sampledatabase < db_backup.sql
```

Notice the use of the less than symbol to indicate that you want to import data rather than export when we used the mysqldump command in the previous section. In this example, no password is given, so it indicates that root has no password assigned. Of course, in a real-world scenario, root should always have a password. We use no password to give you an example of an SQL statement that uses no password.

Another important part of the statement is the host name. When we exported statements to the sql file, we didn't specify a host. In this import statement, a host name is specified. This is because you can import the SQL statements to several different hosts provided you have access. In this example, we import to localhost. You can also import to the IP address or host name.

If you exported your SQL statements to a compressed file, you first need to extract the file content. The default Unix command to extract a file is the gunzip command. If you run Windows, you need a third-party tool such as Winzip or 7-Zip. These tools first extract the file content to a specified directory, and then you can use the MySQL command shell to import content.

If you run Unix, you just need to run the following command line.

```
gunzip -c db_backup.sql.gz |mysql -u root –p
```

The "c" option tells the unzip tool to use standard output, which is a concept in basic coding libraries.

Once you have the file extracted, you can them import the content to your MySQL server. The mysqldump file doesn't have commands to create the database. For this reason, if you're restoring an entire database on your server and it's not created yet, you must first create the database before importing the data.

```
CREATE DATABASE sampledatabase;
```

After the database is created, you can now import your database schema and data. The following command imports your backup data.

```
mysql -u root –psampledatabase < sampledatabase.sql
```

Once you import the database schema and data, you still need to check your database for any errors and create users, if this is a new database.

1.4 The MySQL Binary Log

For Windows users, you're probably familiar with the Event Viewer. The Event Viewer takes a record of each security and system change. It also logs any errors recorded on the system.

The MySQL binary log is similar except it's specific to the MySQL server. The binary log records any statements run against a database server. For instance, if you delete a record, the binary log keeps track of the command. It records the length of time it takes to run the statement, and the number of records affected. It's only used for statements that actually change data. For instance, the binary log wouldn't show when a SELECT statement was executed. It wouldn't show when you used the SHOW command. Any command that doesn't not remove, add or change data isn't recorded in the binary log.

The binary log option makes MySQL performance slightly worse. It's not an option that destroys performance, but there is some slight difference. However, if your database fails or you need to restore data, this log file can be used to restore edited information instead of using a backup file.

Each log file has a numeric file extension, and you need this file name to review its content. The "mysqlbinlog" command displays contents of the log file based on the database name. Before you restore records, it's best to perform a quick overview of the log file information to ensure it's not corrupted. To view the content of the log file, use the following command.

```
SHOW BINLOG EVENTS IN log.006;
```

In this example, the log file "log.006" is reviewed on localhost.

Once you review the data and ensure its format, you can then import data based on the log files. The following command imports data from your log file.

```
mysqlbinlog /var/lib/mysql-bin.000016 | mysql –u root –p
```

Again, the mysqlbinlog utility is used. This example also shows you the default location where MySQL stores the log file. In this example, the log file has the 000016 extension. This extension number continues to increase as the log files fill up.

1.5 Check and Repair MySQL Tables

Data corruption is one of the worst technical issues for a database administrator, but MySQL includes a check and repair system that helps make reviews easier. MySQL also attempts to fix any corrupt data using its own binary logs and backups. The "mysqlcheck" command does both of these tasks, and it's a benefit for database administrators who think any table is corrupted. It can help database administrators mitigate damages before they happen, so you don't need to restore data from scratch using backup files. Database administrators should run this utility during slow production times such as at night or when customers aren't placing many orders. Using the utility can cause performance issues, so use it at slow hours.

The following statement is an example of the mysqlcheck statement for reviewing table stability.

```
mysqlcheck –c sampledatabase Customer –u root –p
```

You might recognize the "c" option. This option again tells the MySQL engine to use standard output when displaying your table information. We want to review the sampledatabase, which has all of our customer information in the Customer table. We've specified the Customer table for review. The user we use is root with no password.

The response you get is very simple. The following is an example of the mysqlcheck response.

```
sampledatabase.Customer                            OK
```

This response tells you that the MySQL check system verified the stability and validity of the database table Customer.

While checking one table is useful, you normally need to check all tables in a database. Running the mysqlcheck command on a monthly basis helps you identify issues before they become a major issue. The following command checks all tables in a database.

```
mysqlcheck –c sampledatabase -u root –p
```

Since we're checking all tables, this is an example output of our command.

```
sampledatabase.Customer                                     OK

sampledatabase.Order                                        OK
```

You're not just limited to one database either. You can also use the mysqlcheck command to check all databases on your server. The system will go through each database and each associated table. The following is the command for checking all databases.

```
mysqlcheck –c sampledatabase -u root –p --all-databases
```

What happens when you need to repair a table? The mysqlcheck utility makes it easy to repair tables if the review displays that tables are corrupted. MySQL runs its own repair system, which is the REPAIR TABLE command. In essence, running the mysqlcheck utility runs the REPAIR TABLE utility without specifying a table.

Note that repairs are only made to MyISAM tables and archive tables. The following statement repairs your sampledatabase database.

```
mysqlcheck -r sampledatabase Customer -u root -p
```

The above statement repairs just the Customer table, but you can also run a check and repair on the entire database. The following command performs a check, optimization and repair on the sampledatabase database in one shot.

```
mysqlcheck -u root -p --auto-repair -c -o sampledatabase
```