DATA DEFINITION LANGUAGE (DDL)

As stated above the Data Definition Language (DDL) is a sublanguage of SQL and is used to implement the database design or schema. Through SQL commands the designer can CREATE, DROP or ALTER objects within the database Data Dictionary, for example, create a table.

Create Table

Let's look at that first. The syntax for creating a table in MySQL is

```
CREATE [TEMPORARY] TABLE table_name {
    column definition, ...
) [TYPE=table_type];
```

An implementation of this might be

```
CREATE TABLE students (
    student_id  INT,
    first_name  VARCHAR(50),
    last_name   VARCHAR(50)
);
```

This creates a three column table with a student identifier that is of data-type integer and 2 columns of data-type character with variable length up to 50 characters. Notice the comma at the end of each column definition to separate the definitions.

Now we have touched on them lets have a better look at some of the data types we might find in MySQL.

There are three main data-types in MySQL, character, numeric and date. Within these three categories the most common used are:

- ***Character***
  - CHAR(length) — fixed-length character data
  - VARCHAR(length) — variable-length character data (length is the maximum number of characters that can be stored)
- ***Numeric***
  - INTEGER — a whole number
  - DECIMAL (precision, scale) — floating point number precision is the number of digits in the number, scale is the number of digits to the right of the decimal point eg DECIMAL(7,2) could hold a number up to 99999.99
- **Date**
  - DATE — a valid date

Having looked at the syntax and the data-types lets look at a more complex example of a table creation. In this next example we have an employee database that consists of 2 tables where the two tables are relationally linked via a foreign key.

```
CREATE TABLE department (
    departmentId INT          NOT NULL AUTO_INCREMENT,
    name         VARCHAR(30),
    PRIMARY KEY (departmentId)
) TYPE=INNODB;


CREATE TABLE employee (
    employeeId   INT          NOT NULL AUTO_INCREMENT,
    name         VARCHAR(80),
    job          VARCHAR(30),
    departmentId INT          NOT NULL,
    PRIMARY KEY (employeeId),
    FOREIGN KEY (departmentId) REFERENCES department(departmentId)
) TYPE=INNODB;
```

There a a number of addition in this table from the first example. Lets look at them here

- NOT NULL
  This is optional and can be omitted. A column can be defined as NOT NULL. Use of this option means that the data must include a value for that column. If a column is defined as NOT NULL, and no value is given when inserting data, then an error will occur. The row will not be stored. Primary key columns (see below) must be defined as NOT NULL.

- AUTO_INCREMENT
  This automatically generates a sequence number if a number has not been provided. The value automatically generated will be one greater than the current largest value in the table with the first row inserted having the sequence number 1. You can have only one `AUTO_INCREMENT` column per table, and it must be indexed. In this instance we are using it with a `PRIMARY KEY` and as such will automatically have an index.

- PRIMARY KEY
  This part of the command specifies the column, or columns, that make up the primary key. In a relational database all tables **must have one, and only one**, **primary key** defined. Should a table have more than one column that makes up the primary key then a comma separates them.
  For example: **PRIMARY KEY (student_number, unit_number).**

- FOREIGN KEY
  This is optional and can be omitted. Specifies the column, or columns, that make up a foreign key and the table to which it relates. The link is established with the primary key of that linked table. If the primary key is a compound key, the foreign key must be a compound key. It is good practice to use the same name for both foreign key and corresponding primary key, type and length must be the same. It is this clause that enforces referential integrity between this and the linked table.

It should be noted, the foreign key departmentId establishes the link to the department table. The primary key in that table is also called departmentId.

Note that we can use this foreign key syntax because the employee table is an InnoDB table type. If you do not specify a type, the tables will default to being MyISAM tables. Currently MyISAM tables do not support foreign keys but are planned for a future version of MySQL, probably version 5.1 according to the development schedule.

You may have also noticed in the Create Table syntax the optional TEMPORARY keyword. A temporary table is a table that lasts only for the current MySQL session. In other words the table will only last for as long as you are connected to the MySQL database. An example of this might be

```
CREATE TEMPORARY TABLE students (
    student_id  INT,
    first_name  VARCHAR(50),
    last_name   VARCHAR(50)
);
```

Temporary tables are useful if you need to manipulate data using temporary storage within an application and you do not want the tables you use to remain on the system when you are finished with them. We might also use a temporary table as a copy of another table so we can mess around with it without consequence.

To make a copy of a table you can use the LIKE keyword in the create statement. The syntax of which is:

```
CREATE [TEMPORARY] TABLE new_table LIKE old_table;
```

We have only really touched on a part of the Create table syntax. For a complete syntax of CREATE TABLE go to http://dev.mysql.com/doc/refman/5.0/en/create-table.html.

It is strongly recommended that command files or procedural files are used to hold the CREATE TABLE commands. These are text files that contain all your SQL commands procedurally down the page and can be copied into an SQL client to be run multiple times. This simplifies the process of having to recreate a table definition if your database gets messed up or you want to try it out at home on your own MySQl server. In the command file the following standard is a good practice to adopt:

- Use comments to aid understanding of command file.
- Type SQL reserved words (eg CREATE TABLE, NOT NULL, etc) in uppercase.
- Type table and column names in lowercase.
- Use indentations for clarity.
- Declare primary key and foreign key(s) after the column definitions.
- Be consistent in naming fields, eg empno, depno (both use ‘no’ to represent number).
- In the command file include a DROP TABLE command (see next section) before the CREATE TABLE command. This makes the command file re-runable.