# The City Lit Institute

##### Department of Computing

## Keeley Street, Holborn, London WC2B 4BA

**MySQL**

**(with Apache server)**

SUMMARY

**LECTURER** : **ALEXANDER ADU-SARKODIE**

MSc. Telematics (IT & Telecom), MSc. Eng., Dip. Russ. Lang., Teach. Cert, AMIAEng (UK), MBCS (UK), MIfL(UK)

**Blog** : <http://www.blogger.com/profile/14800490193632788559>

**Email:** [aadusarkodie1@gmail.com](mailto:aadusarkodie1@gmail.com)

**Summary**

* **MySQL** queries are classified into two main groups. **Data Definition Language** (DDL), and **Data Manipulation Language** (DML). These responses create four types of **requests** :
  + **Adding information :** Adding a **row** to a table
  + **Updating information:**  Changing information in an existing row. This includes adding data to a **blank** field in an existing row
  + **Retrieving information:** Looking at the data. This request does not remove data from the database
  + **Removing information:** Deleting data from database.
* **Identify a Unique Identifier:** Do not use a **customer’s name as a unique identifier** for a customer table. The identifier must remain the **same** as the database **persists**. A customer could change his or her name for various reasons at any time. You can misspell as well. That is why **social security numbers** are used as popular identifiers because they are **unique** and do not change. You will have to **invent** a **format** for your customer number that will not change in the lifetime of the table.
* **Primary Keys can be links:** Tables can be linked by including the primary key from **one table** as a **column** in **another table**. This is the **one - to -many** **relationships** that is the most common relationship between tables. A customer could be linked to his or her order by including a column containing the customer’s ID, such as customer number, in the order table. Many rows in the **order table** can have the same customer number because one customer can place many order.
* **Store Information in Smallest Reasonable Chunks:** Storing data in small chunks is more flexible for **future**, unexpected users. You can put two data fields together, much more easily than you can separate two pieces of data stored together in a field. For instance storing a customer’s first and last names together in one long field is a problem if you need the last name only. A better idea is to put a first name field and a last name field together when you need the complete full name. Using string concatenation manipulations.
* **Avoid Repeating Information:** Information should not be **repeated** in a table. If the information changes, you have to change it in several places, introducing an opportunity for **errors to creep** into the database and cause problems. If you realise you are **storing the same information repeatedly in a table row, you need to make another table or re-organise the data in another way.** If you have customers using the same addresses, you can store the **addresses** in a separate table, called **household**. Then you can link the people to the household by including the unique identifier for the household in the table row for each person who lives at that address.
* **One Piece of Information per Column:** Store only **one piece** of information in a column. For instance a table called **movie** with a column called **actors that lists all the actors is not a good idea**. The best thing to do is to **create a separate table actor** that has a row for each actor and a **link the actor row to the movie** table.
* **Use Descriptive Names:** Descriptive database names, table names and column names can make your database easy to understand by anyone who needs to use it. The names should describe clearly what is stored. Use names such as **person, customer** instead of say **‘table’**.
* **Most Numbers are Really Character Strings:** Numbers should be stored as numbers only when you decide to do **mathematical manipulations**. Phone numbers, social security numbers and zip codes are **never going to be added together,** so they should be stored as character strings.
* **Make Columns as Wide as Necessary:** Think carefully about the **widths of columns and do some research**. What’s the **longest** name or number in the phone book? What’s the longest city name?
* **Use ENUM Fields:**  Use ENUM fields to reduce entry errors. They accept only **certain values which keeps the number of typos in a database to a minimum**. MySQL allows 65,535 different values in an ENUM field, which is probably more than one can handle.