Database Designs

Section 1: Introduction to Databases

1.1 Why Do We Need A Database?

When we hear the term database we can think of it as a repository of data from which we can extract information very quickly. Data in the world is ubiquitous i.e. details about everything is stored somewhere especially in the modern connected world. There are a lot of data in the world and data is being constantly generated when we are online which is being stored somewhere. The question we can therefore ask is why do we need a database to store data? Or alternatively, what is it about a database that makes it a good choice for storing data?

As an example, lets say we are planning to setup a large e-commerce site like Amazon or Ebay. There is a lot of technology behind running a good e-commerce system i.e. there are a huge amount of interconnected systems that talk to each other rather than a sing monolithic software. For a e-commerce website this can be a website or a mobile app, a search system, recommendation system, cart, checkout & payment system, order management system, supply chain & logistics system, a catalogue of products and finally a seller system to onboard new merchants.

This provides an idea of of all the systems required to work together for a good working e-commerce site and all the data that these systems would need to hold. The Order Management System (OMS) is the core piece of system the user would interact with each time they purchase something on an e-commerce platform. The user does not need to know an Order Management System exists but it does exist in the background and it keeps track of all the orders that is placed on that website i.e. it keeps track of orders and inventory. With this information it can be used to power personalised searches, recommendations and a whole bunch of other things which can make the user experience more personal and tailored.

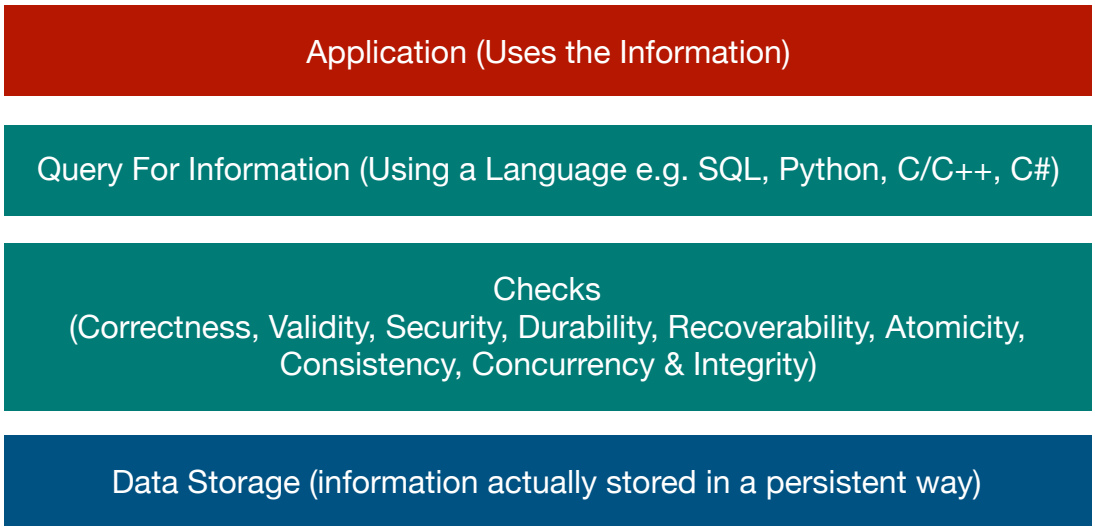
For the e-commerce Order Management System example it must record and store information related to the product such as description, specifications, units available, etc. as well as information about the customer such as name, address and contact details. All of this information needs to be stored and maintained by the OMS. All of this information related to the customer and the product together forms what we would call an order. The order brings together all of the information such as the customer name, product name, quantity, shipping address and a timestamp.

We would want the OMS to work correctly. What we mean by working correctly is that the OMS should allow an order to be placed only when there is available inventory. Each time a product is purchased we would want to reduce the available items when the order was successfully placed. This would also mean to ensure the reduction of inventory is equal to the number of orders placed for that product. There are also other data that needs to be correct such as the status for the order i.e. has it been packed, shipped or returned. We would also want to make sure the correct and valid product is in the order and it is

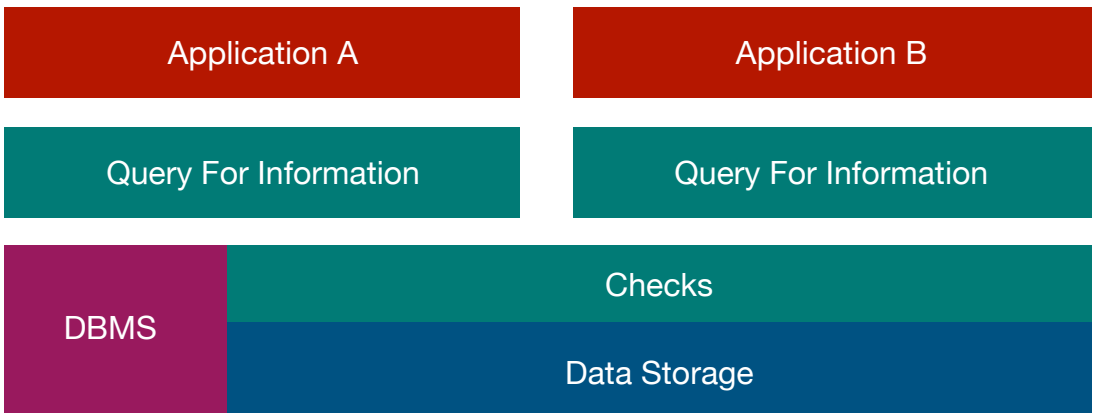
assigned to the correct and valid customer. This is not an exhaustive list of what is required for a OMS system.

The main take away we can see from the above example is that we require a system to implement that does not lose information ever and is backed up in the case of catastrophic failures and can be recovered from failures in order to return back to a sane state. The data must also be secure and not exposed for data protection. Finally, the data must be accessible to multiple authorised people to modify the data at the same time and the data to still make sense to everyone who uses it.

This is a formidable list of requirements for a OMS system. Below is a high level block diagram of how a system would look like:



Suppose we want to create a completely different application but uses the same data i.e. two applications are two different entity but uses the exactly same information. The checks are closely tied with the data itself and should not be a separate system or a separate block and should not be replicated in different systems. The database layer and all the checks should be included within a system and this is known as a Database Management System (DBMS).



DBMS is not purely the storage of data but also incorporates all of the system checks such as security checks, integration checks, consistency checks, etc. Therefore, we can define a DBMS as the following:

A database is a collection of structured data, an abstract representation of some domain.

A Database Manage System is a complex piece of software that sits in front of a collection of data and mediate access to the data guaranteeing many properties of the data and how it is accessed.

The “abstract representation of some domain” simply means a database holds some information in a structure/form that represents something in the real world. While “guaranteeing many properties” means the DBMS carrying out the various checks as seen in the first block diagram above.

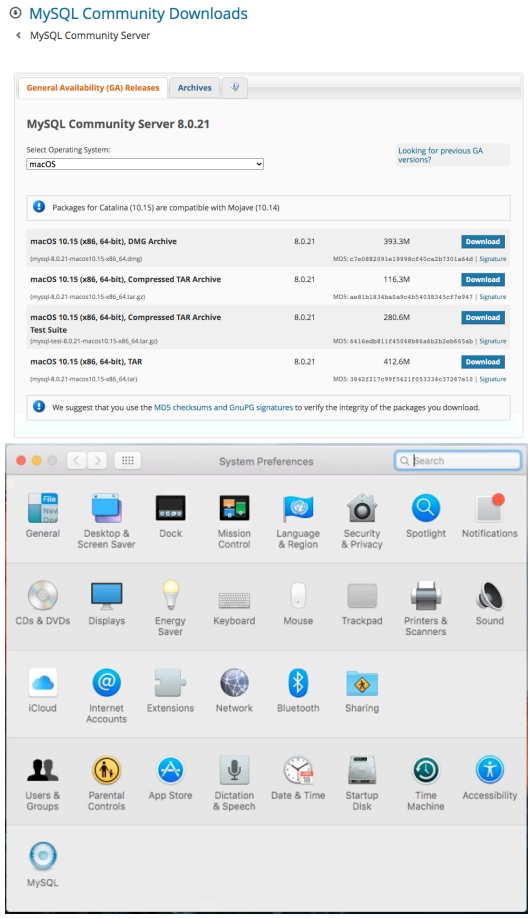
To conclude, the above demonstrates why we need a database or a Database Management System to store data for our applications.

1.2 MySQL and Workbench Installation and Setup

MySQL is an open source Relational Database Management System (RDBMS) and was acquired by Oracle. There are two versions a paid Enterprise edition as well as a free Community edition.

Go to the MySQL website (https://www.mysql.com/) to download and install the free community edition of MySQL Server. The MySQL Workbench is a nice Graphical User Interface (GUI) to interact with your MySQL Server database to write queries, manage the database and other useful utilities offered by MySQL Workbench.

We can go to the following page to download the necessary MySQL Community Server and Workbench tools on your operating system: https://dev.mysql.com/downloads/.
You will be asked to create a free Oracle login in order to download the application. For a Mac you will need to download the DMG archive file.

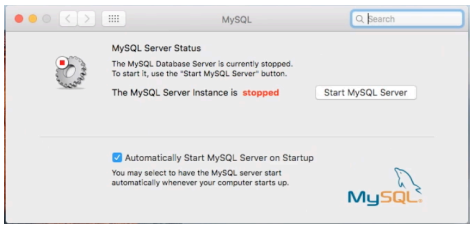


Follow the installation steps to install on your machine.

At the very end of the installation you will be provided with a message containing the root administrator account details for your database. You should make note of this detail and update the temporary password to a password you will remember to login as a root administrator for your database. We now have a MySQL Server installed on our machine. If you navigate to the System Preferences on the Mac we should now see MySQL installed.

If we click on MySQL we will see the status of our MySQL Server as well as some options such as start the server on startup or manually run the

server by clicking on the Start MySQL Server button as seen on the screenshot to the right. This button allows us to start and stop the server whenever we want. We would need the server running in order to work with our database to perform queries and write to our database.



Ensure no MySQL processes is running and our server is stopped in order to carry out the next step. We are going to edit our bash_profile script so that we can tell our terminal where to find the MySQL commands whenever we enter a MySQL command in the terminal.

Open the terminal and type in the following command:

```
$ nano ~/.bash_profile
```

Once the .bash_profile is open in the terminal we can enter the mysql path as seen in the screenshot below:

We can save this change by clicking Ctrl + X and then Y on your keyboard. Finally close the terminal shell for a restart. Whenever we now open our terminal the above line will execute and the PATH variable will be set so that it includes this directory. This will now allow us to type in MySQL commands in our terminal and this directory will be referred to by the terminal shell. If you skip this step then any commands in the terminal will return a Command Not Found error in our terminal whenever we run any MySQL commands.

To enable the Root User on our Mac go to System Preferences > Users and Groups > Login Options (unlock to make changes) > click Network Account Server: Join button > click the Open Directory Utility… button > unlock to make changes > Edit > Enable Root User > Enter a password and save.

We can now type in our terminal the su command and type in the super user (root user) password we just setup to login as the root user for our MySQL Server database.

The following commands will allow us to interact with our MySQL Server Database.

My SQL Commands To Reset Root admin password and Setup New User	
mysqld_safe --skip-grant-tables &	Start the MySQL Server with an option called skip-grant-tables which will allow anyone to login to MySQL Server without any username or password. This option should only be used when you want to reset the root user's password itself.
mysql	This command is usually followed by a username and password to run the MySQL Server. However, with the above command this can be run on its own without any user login

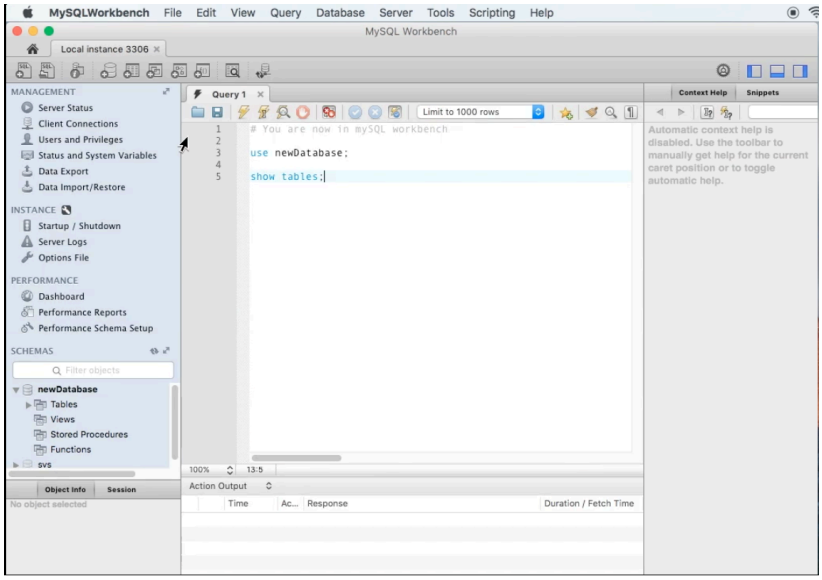
My SQL Commands To Reset Root admin password and Setup New User	
FLUSH PRIVILEGES;	Clear any privileges in the cache and allow you to edit privileges of different users.
ALTER USER 'root'@'localhost' IDENTIFIED BY 'NewPassword';	Change the root user password.
CREATE USER 'your-username'@'localhost' IDENTIFIED BY 'YourPassword';	Add new user and password.
GRANT ALL PRIVILEGESON *.* to 'your-username'@'localhost' WITH GRANT OPTION;	Grant all privileges to the user.
GRANT RELOAD, PROCESS ON *.* to 'your-username'@'localhost';	Reload changes made to permissions.
exit	Exit MySQL Server in the Terminal Shell.
kill 4368 kill mysqld	This is not a MySQL command but a generic OS command to terminate a process. To find the MySQL process ID (PID) go to the Activity Monitor to find the MySQL process running and this will provide the PID number for that process e.g. 4368. We can alternatively write kill mysqld to also terminate the SQL Server process. We can also go to the System Preferences > MySQL route to click the Stop MySQL Server button.
mysql -hlocalhost -uyour-username -pYourPasswrd	Login to the MySQL Server database using the -u and -p options to enter the username and password to the user account you want to sign into.

We would now want to restart the server again and sign into our database with all the privileges assigned to the new user we created. Once signed into the account and into our MySQL Server database we can now run any MySQL commands to our database e.g. show databases, creating a database, create a new table, inserting a new record, etc.

MySQL Database Commands Examples	
show databases;	Show a list of existing databases.
create database NewDatabaseName;	Create a new database.
use NewDatabaseName;	Switch to use the selected database.
create table NewTableName (id int, name archer(256));	Create a new table in the database. In the example this is a 2 column table containing an ID and Name column fields.
desc NewTableName;	Provide a description of a table i.e. what columns it contains and the data types for the column fields.

To install MySQL Workbench we would download the application and install like any other application. This will provide a GUI to work with our database rather than using the terminal shell. Once installed we would need to use the Database > Manage Connections… option to setup the database connection to our MySQL Server. We would need to enter a username and password to connect to our MySQL Database. We can

then finally test to see if we can successfully connect to our database or not. Once we have a successful connection we can use the GUI tool to interact with our MySQL Server database rather than the terminal.



We now have MySQL Server and Workbench installed on our machines (this is the Mac instructions, you will need to follow the instructions for Windows installations which is as simple as running the installation wizard and going through the steps to install both the MySQL Server and Workbench applications).

1.3 Entities and Attributes

Entities in databases are objects with specific characteristics which define/represent the object. For example, students, teachers, staff, customers, bank accounts, branches, transactions, tests, subjects, sports, etc.

We can define an entity as a real-world object or thing either animate or inanimate that can be easily identifiable and is distinguishable.

An entity set is a collection of similar type of entities i.e. an entity set contains entities with the same properties. For example an entity set of teachers; all teachers are human beings with a name, age, gender and other identifiable properties such as a unique id, department, subjects, etc. Therefore, every teacher has the same set of properties but the values for these properties which classifies each teacher differs from teacher to teacher. Each teacher is an object (entity) within that teachers entity set.

The characteristics which define an entity is known as the attribute. For example, if we had an entity of a customer the characteristics such as the name, address, phone number, etc are all attributes of the customer entity.

We can define an attribute as properties of entities which hold specific values. These attributes and values may differ from entities to entities.

Every attribute has a range of values or domain which can be assigned to it. For example the name attribute has a type value/domain of any String value while the age has a domain/value range of a number between 1 to 100.

We can categorise attributes into different categories: Simple, Single Valued, Multi-Valued, Composite or Derived.

A Simple attribute is an attribute that are atomic values which cannot be divided into further sub-values or parts. An example of this is a phone number, email address, bank branch code, etc.

A singled values attribute is an attribute that can contain only one value. An example of this is a NI number, bank branch code, ID, etc.

A Multi-Valued attribute is an attribute that can contain more than one value for the same attribute. An example of this is a person's phone number (e.g. home number, work number, etc.), a person's email address, etc.

A Composite attribute is where the attribute itself is made up of more than one simple attribute. An example of this is a Name (e.g. Title, First Name, Middle Name and Last Name), Address (e.g. Flat/Building Name, House Number, Street/Road Name, City, County, Country, Postal/Zip code), etc.

A Derived attribute is an attribute that does not explicitly exist as part of the data but can be derived from other related data available. An example of this is a person's age where the age can be derived/deduced from their date of birth attribute.

All this comes together to perform something called the Entity Relationship model.

1.4 Identifying Entities Using Keys

A lot of the data we add to a database requires association of the information with a specific entity. The problem that we want to solve is how do we uniquely identify an entity amongst all other entities of the same type?

This is solved via attributes. A single attribute or a combination of attributes of an entity should be unique amongst the entity set i.e. it should be unique amongst all other entities of the same type. For example, if we have a million customers of a bank we should be able to uniquely identify one customer amongst the million customers.

This ability of uniquely identifying an entity is a basic requirement of the real world and also within databases which model the real world. This will allow us to identify an entity without any ambiguity. The unique identifier is called a Key.

From an employee's attributes the Employee ID, NI number, email, phone number can be good choices for a Key while the first name, first name & last name, gender are all bad attributes for a key. Uniqueness is crucial for defining a Key to an entity.

A Key is an attribute or a set of attributes which uniquely identifies an entity in an entity set. The are different categories which a Key can belong: Super Key, Candidate Key, Primary Key.

A Super Key is a set of attributes which is sufficient to uniquely distinguish entities from one another. For example a email address is sufficient to identify a person as is a phone number. However, we can use the combination of these attributes (email address + phone number) which is sufficient to uniquely identify a person. Another example is a NI Number + name attributes to form a Super Key. The NI number is sufficient alone to create a Super Key while the name is not but joining both together is sufficient to also create a Super Key. When you add on another piece of information, they together form a Super Key.

A Candidate Key is a minimal set of attributes which form a Key. So in the example of the email address + phone number to create a Super Key. The email address (or phone number) alone is the minimum attribute required to create a Candidate Key. In the second example of the NI number + name, the minimum requirement is the NI number alone which is enough to make it unique and be a Candidate Key.

A Primary Key is where one of the Candidate Keys which the database designer designates as the key to uniquely identify an entity. Therefore, when we design a database we decide what the Primary Key for the entity is going to be. We typically choose a key that has the minimal set of attributes to uniquely identify an entity which is a Candidate Key (i.e. we can have a whole host of Candidate Keys but we are choosing which one of those will become the primary key for that entity).