**mySQL Datatype Research**

**Block 3 - Module 1/lab1**

**Introduction**

The research obtained throughout this document will serve to better understand and utilize database datatypes. Findings are determined solely from online sources and are deemed up-to-date and thus credible - as of the current date (7/09/2019). Within this document the common reader will find answers pertaining database datatypes such as int, varchar, date, tiny int, primary keys and foreign keys.

The ***int*** datatype in my SQL is known as an integer. The range of this number can be quite large and contains a minimum value of 0 to 2147483647. Moreover, the ***int*** datatype can also be a whole negative number up to – 2147483647. The ***varchar*** datatype is a very commonly used datatype. It holds textual values that range between 1 and 65, 535 characters. In terms of memory the varchar value is stored as a 1 or 2-byte length in addition to its actual data. The ***date*** datatype does precisely that, receive a date. MySQL retrieves and displays this date value as 'YYYY-MM-DD' format. Furthermore, the tiny int datatype is rather similar to its big brother “int” with some minute variation. The length is vastly reduced by comparison. More specifically, tiny int ranges between 0 and 255 accordingly. Any value that exceeds this amount will end with a value error. ***The primary*** key datatype is used to identify unique values within a database table**.** Containing identical values when assigned a primary is in fact an impossibility. Therefore, it is safe to say that the authenticity of a primary value is rightfully reserved. Primary keys (unique value) cannot contain NULL values and subsequently, there can only be one primary key in a database table. Contrary to primary keys, ***unique*** keys can be more than one. A ***foreign key*** is created for direct reference to another table, usually called the parent table. These children to parent tables become linked to one another. Database relationships can be used to help the user form a link between two or more tables. The ***1 to many relationship*** is most probably the most commonly used relation and is used to create a direct link between two tables. By creating a primary key in one table and a foreign key in another, we essentially can bond these tables together. Note that even though a 1 to many may state multiple subsequent tables, this doesn’t mean that there must always be more than just one table linked to the primary key table. The ***1 to 1 relationship*** is similar to the 1 to 1 relationship however, like the link between two tables, the 1 to 1 can only be branched into 1 existing table as opposed to many. The least used relationship but still utilized a fair amount is the ***many to many relationships***. The many to many relationship is when a parent table can have multiple child tables and vice versa. This relationship can sometimes be tricky and requires some thought process in order to identify.

**Conclusion**

This documentation that contained vital information is in its very essence is a beginner guide to many of the most utilized datatypes within databases. The information I’ve provided also contains an intro into the tree main relationships between tables and how they can be linked together. I conclude by saying that this documentation is valuable information for any who seek guidance through the use of databases and its tables.

**Research**

**Int:** In MySQL, INT stands for the integer that is a whole number. An integer can be written without a fractional component e.g., it is 1, 100, 4, -10, etc., and it cannot be 1.2, 5/3, etc. An integer can be zero, positive, and negative.

**Varchar:** MySQL VARCHAR is the variable-length string whose length can be up to 65,535. MySQL stores a VARCHAR value as a 1-byte or 2-byte length prefix plus actual data.

The length prefix specifies the number of bytes in the value. If a column requires less than 255 bytes, the length prefix is 1 byte. In case the column requires more than 255 bytes, the length prefix is two length bytes.

The maximum length, however, is subject to maximum row size (65,535 bytes) and the [character set](http://www.mysqltutorial.org/mysql-character-set/) used. It means that the total length of all columns should be less than 65,535 bytes.

**Date:** The DATE, DATETIME, and TIMESTAMP types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes DATE,DATETIME, and TIMESTAMP values in several formats. For the DATE and DATETIME range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The DATE type is used for values with a date part but no time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

The DATETIME type is used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in ***'YYYY-MM-DD hh:mm:ss'*** format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The TIMESTAMP data type is used for values that contain both date and time parts. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07'UTC.

A DATETIME or TIMESTAMP value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. In particular, any fractional part in a value inserted into a DATETIME or TIMESTAMP column is stored rather than discarded. With the fractional part included, the format for these values is '***YYYY-MM-DD hh:mm:ss***[.***fraction***]', the range for DATETIME values is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999', and the range for TIMESTAMP values is '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999'. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized.

**Tiny int:** The TINYINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations. You can explicitly specify TINYINT as UNSIGNED, but the UNSIGNED modifier has no effect as the type is always unsigned. The range for TINYINT values is 0 to 28 - 1, or 0 to 255.

In embedded SQL, TINYINT columns should not be fetched into variables defined as char or unsigned char, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program. Instead, TINYINT columns should be fetched into 2-byte or 4-byte integer columns. Also, to send a TINYINT value to a database from an application written in C, the type of the C variable should be integer. When converting a string to a TINYINT, leading and trailing spaces are removed. If the leading character is '+' it is ignored. If the leading character is '-' the remaining digits are interpreted as a negative number. Leading '0' characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

**Primary keys:**  A primary key is a column or a set of columns that uniquely identifies each row in the table. You must follow the rules below when you define a primary key for a table:

* A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.
* A primary key column cannot contain NULL values. It means that you have to declare the primary key column with the NOT NULL attribute. If you don’t, MySQL will force the primary key column as NOT NULL implicitly.
* A table has only one primary key.

Because MySQL works faster with integers, the [data type](http://www.mysqltutorial.org/mysql-data-types.aspx) of the primary key column should be the integer e.g., [INT](http://www.mysqltutorial.org/mysql-int/), BIGINT. However, you should make sure that the value ranges of the integer type for the primary key is sufficient for storing all possible rows that the table may have.

A primary key column often has the AUTO\_INCREMENT attribute that generates a unique [sequence](http://www.mysqltutorial.org/mysql-sequence/)for the key automatically. The primary key of the next row is greater than the previous one.

When you define a primary key for a table, MySQL automatically [creates an index](http://www.mysqltutorial.org/mysql-index/mysql-create-index/) named PRIMARY.

**Unique:** To enforce the uniqueness value of one or more columns, you often use the [PRIMARY KEY](http://www.mysqltutorial.org/mysql-primary-key/) constraint. However, each table can have only one primary key. Hence, if you want to have a more than one column or a set of columns with unique values, you cannot use the primary key constraint.

Luckily, MySQL provides another kind of [index](http://www.mysqltutorial.org/mysql-index/mysql-create-index/) called UNIQUE index that allows you to enforce the uniqueness of values in one or more columns. Unlike the PRIMARY KEY index, you can have more than one UNIQUE index per table.

**Foreign keys**: A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

**1 to 1 relationship:** One-to-One (1-1) relationship is defined as the relationship between two tables where both the tables should be associated with each other based on only one matching row. This relationship can be created using **Primary key-Unique foreign key constraints**.

With One-to-One Relationship in SQL Server, for example, a person can have only one passport.

**1 to many relationship:** The One-to-Many relationship is defined as a relationship between two tables where a row from one table can have multiple matching rows in another table. This relationship can be created using **Primary key- Foreign key relationship**.

In the One-to-Many Relationship in SQL Server, for example, a book can have multiple authors.

**many to many relationship:** A many-to-many relationship refers to a relationship between tables in a database when a parent row in one table contains several child rows in the second table, and vice versa. Many-to-many relationships are often tricky to represent.  
  
The many-to-many relationship is usually a mirror of the real-life relationship between the objects the two tables represent.

**FLOAT(M,D)** − A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.

**TINYINT** − A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.

**SMALLINT** − A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.

**MEDIUMINT** − A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.

**BIGINT** − A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.

**DOUBLE(M,D)** − A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.

**DECIMAL(M,D)** − An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

**TIMESTAMP** − A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 ( YYYYMMDDHHMMSS ).

**TIME** − Stores the time in a HH:MM:SS format.

**YEAR(M)** − Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

**CHAR(M)** − A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

**BLOB or TEXT** − A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are **case sensitive** on BLOBs and are **not case sensitive** in TEXT fields. You do not specify a length with BLOB or TEXT.

**TINYBLOB or TINYTEXT** − A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.

**MEDIUMBLOB or MEDIUMTEXT** − A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

**LONGBLOB or LONGTEXT** − A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.

**ENUM** − An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

**REAL -** The REAL data type accepts approximate numeric values, up to a precision of 64. No parameters are required when declaring a REAL data type. If you attempt to assign a value with a precision greater than 64 an error is raised.

BOOLEAN - The BOOLEAN data type supports the storage of two values: TRUE or FALSE. No parameters are required when declaring a BOOLEAN data type.

Use the case insensitive keywords TRUE or FALSE to assign a value to a BOOLEAN data type. Comparisons using the BOOLEAN data type should also use these keywords. If you attempt to assign any other value to a BOOLEAN data type, an error is raised.

**References**

<https://www.w3schools.com/sql/sql_foreignkey.asp>

<http://www.mysqltutorial.org/mysql-primary-key/>

<https://dev.mysql.com/doc/refman/8.0/en/integer-types.html>

<https://dev.mysql.com/doc/refman/8.0/en/numeric-type-overview.html>

<https://www.tech-recipes.com/rx/56738/one-to-one-one-to-many-table-relationships-in-sql-server/>

<https://www.techopedia.com/definition/27291/many-to-many-relationship>