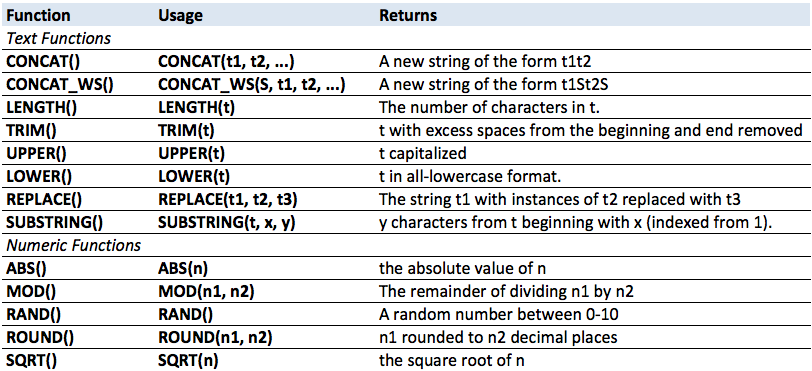
# Conventions

1. **make the table name plural and ALL lowercase**- make it plural (ex. users, leads, sites, clients, chapters, courses, modules)
2. **use "id" as the primary key**- name it *id*(also make it auto-incremented).
3. **name foreign keys with singular\_table\_name\_id**when referencing to a primary key in another table name it [*singular name of the table you're referring to]\_id* (ex. user\_id, lead\_id, site\_id, client\_id, chapter\_id, course\_id, module\_id).
4. **use *created\_at* and*updated\_at*** as columns for the timestamp in EVERY table you create.

When we do things in ORM or in Ruby on Rails, it becomes extremely important that we follow these naming conventions.

## ****Simple Data Types:****

* **VARCHAR(number of characters)**
  + Used to store non-numeric values that can be up to 255 characters. It is called a VARCHAR because it can store a variable number of characters and will only use the space required for each record that is stored in the database. VARCHAR should be used for values with different character lengths like an email, first\_name, or last\_name.
* **CHAR(number of characters)**
  + Also used to store non-numeric values, however, it will use up all space for the set number of characters regardless of what value is added. For instance, if I set CHAR(15), and I try to store the value "Coding", it will use up the equivalent of 15 characters even though "Coding" is only 6 characters long. Char is good to use for things that will always be a given number of characters. Char would work well for something like a state\_abbreviation.
* **INT**
  + Used to store integers.
  + The columns that you will find mostly using the INT are things like a unique identifier for each table. The majority of rows in a table will not exceed 2.1 billion records. INT is good to use for most normal number values like a phone\_number or a zip\_code.
  + **unsigned** (positive numbers only) - can store numerical values from 0 up to 4294967295
  + **signed** (positive and negative numbers) - can store numerical values from -2147483648 up to 2147483647
* **BIGINT**
  + BIGINT would be used for columns that would need to store huge numbers. In most cases you wouldn't need BIGINT, but if you wanted to store something like a Facebook id when using Facebook's API, since they have over a billion users the id will need to be a data type of BIGINT.
  + **unsigned**(again positive numbers only) -can store numerical values from 0 up to 18446744073709551615
  + **signed**(positive and negative numbers) - can store numerical values from 9223372036854775807 to -9223372036854775808.
* **TINYINT**
  + TINYINT would be good to use for numbers that will be relatively small. A good example of something that would use a TINYINT is user level identifier (0 - inactive user, 1 - active user, 9 - admin).
  + **unsigned -**can store numerical values from 0 up to 255
  + **signed -**can store numerical values from -128 up to 127
* **FLOAT**
  + Used to store floating point numbers (numbers that need to have decimal places). An example column for this would be like an item\_cost.
* **TEXT**
  + Used to store a large amount of text, like a description, message, or comment. Use this for any text that VARCHAR() is too small to handle.
* **DATETIME**
  + used to store a date and time in the format YYYY-MM-DD hh:mm:ss



CREATE – INSERT INTO table name (column name1, column name2) VALUES (value1, value2);

INSERT INTO users (username, email, age) VALUES (“briankk123”, “briankk123@gmail.com”, 27);

READ – SELECT *column name1, column name2* FROM *table name* WHERE *conditionals*

SELECT username, email FROM users WHERE age > 26;

UPDATE – UPDATE *table name* SET *column name = “ ”* WHERE *conditional*

UPDATE comments SET content = “Boo” WHERE comment\_id = 6;

DELETE – DELETE FROM *table name* WHERE *conditional*

DELETE FROM likes WHERE user\_id = 1 and like\_id = 4