# Introduction to SQL

* Structured Query Language
* It is not a programming language (by itself). It is a **declarative query** language
  + You tell the program what needs to be done instead of delivering the implementation details to do what needs to be done.
* It performs operations against a relational database

## Database

* It is just an organize collection of data stored in some organized format.
* They allow us to input, manage, organize, and retrieve data quickly
* Traditionally, it is organized into “tables” and each table will have a row and column
  + Rows will be the same as “records”
  + Columns will be the same as “fields”
* Data itself is the intersection between the row and column

## RDBMS

* It stands for Relational Database Management System
  + SQL is a specific style of RDBMS
* It upholds specified relationships between tables or our data
* It includes functions that maintain the security, accuracy, integrity, and consistency of the data

# SQL Sublanguages

## DDL

* Data Definition Language
* It is the sublanguage in SQL that is responsible for creating/altering the tables in your database
* Create – most used to create a table and their columns
  + Can also be used to create views, schema, etc.
* Alter – Will change the column of the table
  + Can also change certain properties or constraints
* Drop – will drop the table from the database out of existence

## DML

* Data Manipulation Language
* It is the sublanguage in SQL that is responsible for adding/changing/modifying the data within a table
* Insert – Adds row(s) to your table
* Select – gets data from your table
* Delete – remove data from your table
* Update – Modify pre-exist data in our table

## TCL

* Transaction Control Language
* They are used to manage transactions
* What are transactions?
  + Like a function so it can run multiple SQL statements, but it has the added benefit to either run every SQL statement or none at all
* Begin – it will start our transaction (so think of “{“ in C#)
* Commit – it will end our transaction and commit all the changes it did with the database (so think of “}” in C#)
* Savepoint – it is used to save the current state of the database
* Rollback – It will rollback/go back to a specified save point

# Constraints

* They are a way for you limit what data can be inputted in column
* It will specify one or more rules that the data must follow within a column

## Some commonly used constraints

* Type
  + Restrict what data type of value can be inputted in a column
* Unique
  + Data in this column cannot have repeating values
* Not null
  + Will make sure that every data you input cannot be null
* Primary Key
  + It is that every data inside of the column will be unique
  + It is implicitly unique and not null
  + Acts as the unique identifier for the records in the table
* Foreign Key
  + Data in this column will reference a data from another table/column
  + Establishes relationships between 2 columns in the same table or different tables
* Default
  + If you didn’t supply any value, default constraint will supply some default value
* Check
  + Adds an extra condition on the data
  + Ex: age column and it must have only above 18 age (age >= 18)
* Identity(1,1) – Good to know for you
  + Automatically fill in numerical values into that column
  + Essentially great way to have unique value in each row for your primary key
  + First parameter, it is the starting point the row
  + Second parameter, it is how much you want to increment by

# Multiplicity

* It is way to describe relationships between two tables
* We will be using primary and foreign key constraints to establish these relationships
* There are 3 main categories:

## One to One

* One row in Table A is directly relation to one row in Table B and it goes both ways (bi-direction)
* You have to make both columns unique constraint is needed
* Ex: One person can only have one heart

## One to Many

* One row in Table A is relation to many rows in Table B
* Make sure your foreign key is not unique
* Ex: One person has many fingers

## Many to Many

* Many rows in Table A are related to many rows in Table B
* You must use a join table to create this relationship
  + Join tables are just separate, independent table used to establish many to many
  + Consists of 2 columns are that both foreign keys
* Ex: Many students have many classes

# Normalization

* It is a design pattern that reduce/eliminate data redundancy and data duplication

## 0NF – Zero Normal Form

* No normalization is being utilized
* Data redundancy and data duplication

## 1NF – 1st Normal Form

* Each table must have a primary key
* All data must be atomic (One cell should only hold one value (NEVER A LIST))

## 2NF – 2ND Normal Form

* If you are in 2nd normal form, you are already in 1NF
* Remove partial dependencies
  + **Don’t create composite primary keys**
  + You need every column inside of that composite primary key to be dependent on the other columns of your table

## 3NF – 3rd Normal Form

* Remove transitive dependencies
* What are transitive dependencies? It is when a column in the table doesn’t depends on the primary key (they are unrelated and should be in two different tables)
* Make sure every column relates to the table you are putting in

## Referential Integrity

* It is a concept that is used to maintain our relationships between our table without missing information
* Essentially, we can’t delete or update data in a table that have existing relationships with another table
  + You can change/delete data from a table with a foreign key
* If you really want to drop the table holding relationships regardless of the data inconsistency it will bring, you can use cascade and it will auto fill any reference to that table with null value

# Joins

* Allows us to bring together data from multiple tables
* They will join two rows based on some condition (usually when the primary key from one table matches the foreign key of another table)

## Inner join

* Returns rows with matching values in both tables

## Left join

* Returns every row from the left table and returns some row from the right table with matching values

## Right join

* Return every row from the right table and returns some row from the left table with matching values

## Full join

* Returns every row from both tables regardless of if it matched or not

## Cross join

* Returns the cartesian product of the joined table
* Useful for returning all possible combinations of all rows from each table

# Subquery

* Allows you to add a query that is nested inside another query

# Introduction to T-SQL

* It is a superset of SQL
* It is Microsoft’s way of extending the functionality of SQL

# Set Operation

* Another way of combining data together
* It doesn’t match with or compare other columns in different queries to match it in a single row
* It needs to have the same data type and # of column from both queries that you are combining

## Union

* It will give you all the rows from both queries **except** duplicated rows

## Union All

* It will give you all the rows from both queries **including** duplicated rows

## Except

* It will give you all the rows from the left queries that is unique compared to the right query

## Intersect

* It will give you all the rows from the left query that is duplicated from the right query

# Functions

* Also known as User Defined Functions
* They are like methods but with some different restrictions
* You can perform multiple statements like a transaction
* You cannot start a transaction in a function
  + Main reason being is that a function cannot modify the database state
* Mostly used to do some algorithm logic that can be re-usable

## Different types of functions

* Scalar function
  + Take in one or more parameters and return a single value
* Aggregate function
  + Takes in multiple data (mostly from a column in a table) and only return a single value
* Tabular function
  + When a function returns a table data type

# Stored Procedures

* It is like a function except it can take in any input and output multiple things
* It can have input parameters but also has output parameters
* Essentially it can return multiple data unlike functions
* It can modify the database state
* You can also have optional parameters
* It has a special output parameter (with specific syntax) to return if the stored procedure was successful or not

# Triggers

* They are a special type of stored procedure that will run when a certain even has happen
  + The event might be before or after an insert, update, or delete queries
* You can specify when you want to trigger the trigger such as before or after
* DML triggers – triggers on any DML operations
* DDL triggers – triggers on any DDL operations
* Logon triggers – triggers when a user logs in and grab their information to be log/save somewhere else

# ACID properties

* Set of properties of a database transaction that is intended to guarantee data consistency in your database

## Atomicity

* Either all SQL statements run or none it all.

## Consistency

* There should be a data consistency in your database
* Transferring money to your saving accounts and making sure the total of money didn’t change
* Ex: $100 checking account => (after transferring $30 to saving accounts) => Checking: $70 and a Saving: $30

## Isolation

* The state of a transaction should be invisible to other transaction
* Can’t access the result of another transaction until it is completed

### Different phenomenon that that occur during transactions happening at the same time

* Dirty read – Reading that has not been committed
  + If transaction 1 updated a row followed by transaction 2 reading that updated data and suddenly transaction 1 rollback its changes to the database (due to failure) and now transaction 2 has read data that basically never existed
  + Ex: You are in a process of getting paid and your bank account is updating that row associated with your checking. So, you went to your bank app or bank website and **read** that checking account balance with the updated data, so you went out and try to buy something. Suddenly, your checking account that was updated rollback its changes and now you basically have your previous money.
* Non-repeatable read – When data is read twice but comes out different each time
  + If transaction 1 read a data (let’s say 5) and transaction 2 updated that row (to be 6) and transaction 1 reads the same data but it will give 6 instead.
  + Main difference is that transaction 1 and 2 will not roll back changes
  + Ex: You are in the process of transferring money to your savings, you read how much you have in your checking account (lets say $500) and you went and did a transfer of $300 to your savings. So during your transfer transaction a logic in your transaction might be to check if $500 > $300 and in this case it passes. Suddenly transaction 2 from your previously brought item went and updated your checking account to deduct how much you paid for it (lets say $400). Transfer transaction and it will proceed to do its update statement and all you have left in your bank account is a -$200.
* Phantom read – when data is removed or added by another transaction
  + If transaction 1 finds the average of a column and then transaction 2 comes in and add a new number in this column and transaction 1 tries to find the average again, it will come out different than the first one.
  + Main difference is you will be dealing with problems when you add or remove rows in a table while another transaction is reading the table

## How to solve each phenomenon (Degrees of isolation levels)

* Read uncommitted – no isolation at all
* Read committed – It forces the second transaction trying to read data to wait until that first transaction has been committed.
* Repeated reads – it forces the second transaction to wait until the first transaction finishes reading the data all the way through its transaction.
  + Will wait if there is a select statement
* Serializable - forces the second transaction to wait until the first transaction finishes reading multiple data in its transaction.
  + Will wait if there is an insert/delete statement
  + So this will basically stop concurrent transaction

## Durability

* Once a transaction is completed, the changes to the database will be permanent