Class 2 objectives

- Pair Programming discussion
- Create Table
- Insert
- Select
- Reminder:
    - Definitions, Class Notes, Homeworks and Discussions are all source material for Quizzes and are required reading *(even if in-class work skips something)*.
    - Quiz #1 will take place during the beginning of class 3
- Preview Homework

Class Code

Pair UP!

- **Choose a programming partner, spend a couple minutes - to get everyone paired.**
- **Anyone still need a partner? Raise your hands.**
- **Expectations for pairs:**
    - Pair programming is an agile software development technique in which two developers work as a pair together on one workstation. The driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The navigator also considers the “strategic" direction of the work and considers improvements and likely future problems. The two developers switch roles frequently. Read more here: https://en.wikipedia.org/wiki/Pair_programming
    - Deliverables (once a week AND the final project) will **require** the full name of each partner in the top of the content delivered.
        - Turn in only 1 homework per pair.
        - Missing partner names will result in no credit for the partner.
        - Grades will be equal and shared.
        - Pair team databases need to match after each pair homework. So that individual assignments are current, for testing code.
            - You both need to *run* the code that changes data structures and data, in your own databases.
    - Quizzes will be individual effort
    - Do not change partners without prior discussion with instructor.
    - You are *individually* responsible to ensure your homeworks are delivered by one of the two partners, but not both.
    - Homeworks are not accepted late without prior arrangement. Grades WILL be reduced for both team members, if late delivery is approved.

Definitions (Definitions Presentation)

- **Constraint**: An explicit requirement added to table column definitions, to ensure certain properties that data in a column must comply with. They help provide data integrity since they define conditions -that restricts the column content- to remain true while inserting or updating or deleting data in the column. Constraints can be used together. Some types of constraints we will use in class are shown below.
    - DEFAULT: if the developer *ignores* the column during insert of the data values, the default will be used. DEFAULT is applied to one column at a time and affects any row added to the table.

    - NOT NULL: NULL data is never allowed (NULL will be discussed more in future classes, but is defined as “no data”. Thus a NOT NULL constraint is saying that putting data in the column is required for all rows. NOT NULL is applied to one column at a time and affects any row added to the table.
    - UNIQUE: no two rows can have the same data. UNIQUE constraints can be defined on multiple columns, and multiple UNIQUE constraints can be created on one table.

- **Primary Key**: A specific type of column constraint. A primary key can be defined for column(s) in a table. It will then provide a constraint for the data in the column(s). After it is defined, it constrains the data in the column to only contain unique content [amongst all of the rows.] Each record then has key values that can not occur twice in one table. With a key, you can find at most one row. Primary Keys are different from UNIQUE constraints because Primary Keys can also be “referenced” by other tables (they exist as something different from mere UNIQUE constraints, because they provide the potential to make relationships *between* tables.)

- **Composite Key**: When 2 or more columns represent a unique row.

- **Foreign Key**: A specific type of column constraint. A foreign key can be defined for column(s) in a table. It will then provide a constraint for the data in the column(s). After it is defined, it constrains the data in the column to only allow values that already exists in another table’s Primary Key. Foreign keys ensure *referential integrity*. Note: Foreign Keys can be NULL or NOT NULL… i.e. if you put data into a Foreign Key column, that data has to be an actual value from the table you’ve referenced, but if you don’t put data there, it’s may be allowed to be empty - based on the nullability of the column itself.

- **Referential Integrity**: A relational database concept, which states that table relationships must always be consistent. In other words, any foreign key field must agree with the primary key that is referenced by the foreign key.

Just for fun: https://tmblr.co/Z14uHt1BE_s9u

References:
https://dev.mysql.com/doc/refman/5.7/en/create-table.html
https://dev.mysql.com/doc/refman/5.7/en/alter-table.html