

# Agenda

## Data Model Building Blocks

- Entities
- Attributes
- Relationships
- Constraints

## Database Keys

### Basic SQL operations:

- Create a database
- Create tables within the database
- Populate those tables with data
- Perform some basic operations

# Notes

- AWS credits should have been sent to everyone who requested them. If you haven't yet received them contact:  
[Softwarerequest@labstaff.dce.harvard.edu](mailto:Softwarerequest@labstaff.dce.harvard.edu)
- if you have strange connectivity issues, try:
  - using another region
  - creating a new EC2 instance
- We are using the most recent version of MySQL 8.0 - don't let the Linux version in the package name throw you off
- next section will be on Sunday at 10ET with Sergei
- Homework 3 will be released tomorrow with a two week deadline, and no new assignment next week

# Building Up Our First Database

## **What we know so far:**

We know how to set up and connect to a server

We know how to install MySQL on it

We know how to configure MySQL

## **What we want to do next:**

Create a database

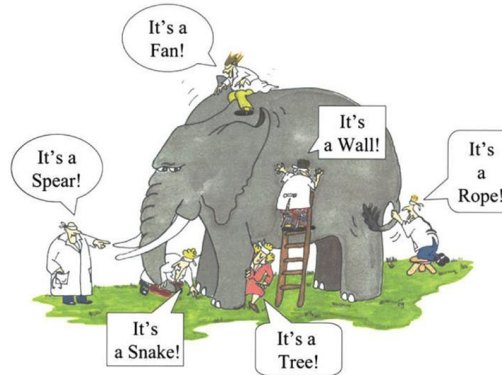
Create tables in that database

Insert some data into those tables

# Data Model

Data modeling is the first step in database design. It serves to map real-world objects to the computer database.

A model is an abstraction of of a more complex real-world object or event. A data model is a relatively simple representation, usually graphical, of more complex real-world data structures. In more general terms, a model is an abstraction of a more complex real-world object or event. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain. <sup>1</sup>



John Godfrey Saxe (1816-1887), *Blind Men and the Elephant*

<https://www.patheos.com/blogs/driventoabstraction/2018/07/blind-men-elephant-folklore-knowledge/>

(1) Coronel, Morris. *Database Systems Design, Implementation, & Management*. Cengage, 2019

# Data Model Building Blocks

The basic building blocks of all data models are:

- entities
- attributes
- relationships
- constraints

Let's run through what each of those are...

# Database Basics

- Collection of tables
- Heading: table name and column names
- Body: rows, occurrences of data

Table Name: Students

Primary Key: StudentID

<u>StudentID</u>	StudentFirstName	StudentMiddleInitial	StudentLastName	StudentGPA
123	John	J	Doe	3.0
456	Jane	T	Smith	3.5
789	Galen	NULL	Erso	4.0

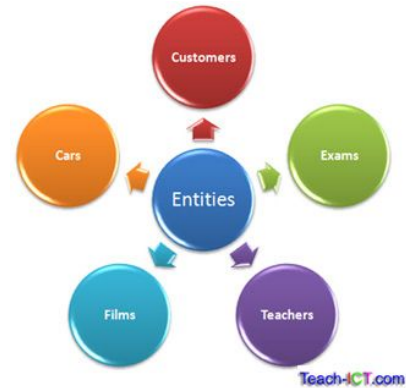
# Entities

An entity is a person, place, thing, object or event about which data will be collected and stored.

Only those things about which data will be captured or stored is considered an entity. If you aren't going to capture data about something, there's no point in creating an entity in a database.

It would make sense to create an entity in a school database for general Instructors and the data associated with them, it would not make sense to create an entity for a very specific “Database Class Instructor”. An Instructor entity would have many different occurrences, such as Greg Misicko, Marina Popova, etc

Entities do not have to be physical objects, they can also be abstractions such as a Harvard Extension Course, or a Lecture.



# Attributes

An attribute is a characteristic or trait of an entity type that describes the entity. For example, a Person entity type could have a Date of Birth attribute, a First Name attribute, a Last Name attribute, etc.

An attribute has a name (Date of Birth, First Name, Last Name), and a data type (date, string, etc).

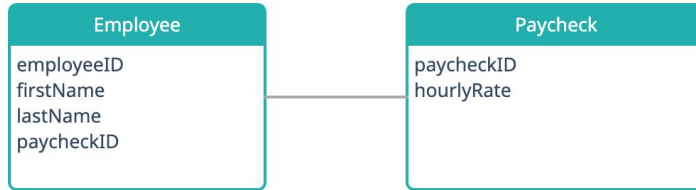
Some attributes are required, some may be optional.



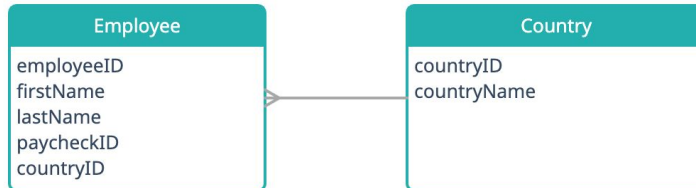
# Relationships

A relationship describes an association among entities. There are **three types of relationships** in relational database design:

## 1) One-to-One (1:1 or 1..1)

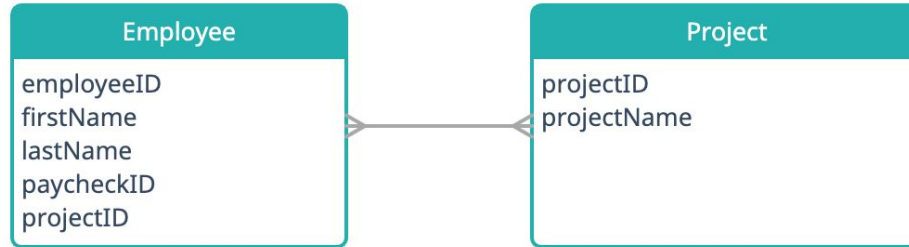


## 2) One-to-Many (Many-to-One or 1:M or 1..\*)



# Relationships

## 3) Many-to-Many (M:N or \*.\*.)



# Constraints

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints are usually expressed in the form of rules, for example:

- a student's GPA must be between 0.0 and 4.0
- an email address must be unique
- a value cannot be NULL (cannot be empty)
- if a value is not provided, set to a default value

[https://www.w3schools.com/sql/sql\\_constraints.asp](https://www.w3schools.com/sql/sql_constraints.asp)

# Data Types


Similar to declaring data types in some programming languages, every column in your database needs to identify the type of data it is representing; for example - are you storing a number, or a word?

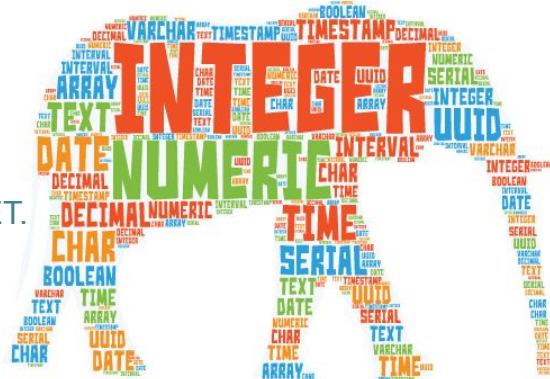
Why should we have to specify the data types we use? Why can't we just store whatever we want in a column, and let the DBMS figure out what to do with it? It has to do with how this data is stored and managed by the system, such as how memory is allocated and what kind of operations will be available.

# Data Types

## What are the data types supported in MySQL?

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

- Numeric Data Types
    - Integer Types (Exact Value) - INTEGER (INT), SMALLINT, TINYINT, MEDIUMINT, BIGINT
    - Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC
    - Floating-Point Types (Approximate Value) - FLOAT, DOUBLE
    - Bit-Value Type - BIT
  - Date and Time Data Types
  - String Data Types
    - CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET.
  - Spatial Data Types: Used for geometric data storage
  - The JSON Data Type: must be a **valid** JSON document or it will be rejected
- 



# What is SQL

Structured Query Language is the language used both for data definition and data manipulation in an RDBMS. It is one of the most, if not THE most, popular languages in use in the industry.

SQL uses English-like statements to perform operations, unlike procedural database languages which also require you to explicitly define the navigational access path to data.

<https://techdifferences.com/difference-between-procedural-and-non-procedural-language.html>



<https://ittutorial.org/wp-content/uploads/2019/11/rdtyuj%C4%B1.png>

# What can SQL do

[https://www.w3schools.com/sql/sql\\_intro.asp](https://www.w3schools.com/sql/sql_intro.asp)

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views



<https://moralis.io/structured-query-language-sql-explained-what-is-sql/>

# History of SQL

Disclaimer: the following summary is based on my own internet research and may be inaccurate or wrong.

In Lecture 1 we talked about the origins of the Relational Database Model.

<https://en.wikipedia.org/wiki/SQL#History>

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce after learning about the relational model from Ted Codd in the early 1970s. This version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original quasi-relational database management system, System R, which a group at IBM San Jose Research Laboratory had developed during the 1970s.

The acronym “SEQUEL” had to be changed to “SQL” due to the fact that someone else was already using the SEQUEL acronym.

SQL showed potential early on with its ability to fetch data using English-like syntax, but struggled due to hardware limitations of the 1970's. A simple query could take minutes to complete.



# History of SQL

In 1977 a company called Relational Software, Inc was formed in Menlo Park. They developed a relational database based on SQL and named that database “Oracle”, beating IBM to market by two years. Oracle was the first commercially available RDBMS. IBM announced an RDBMS in 1982 named SQL/DS, and released it to the market in 1982.

Other companies were actively working on their own relational database management systems, and in 1982 ANSI formed a technical committee to establish a standard. Several standards evolved over the years: SQL/86, SQL/89, SQL/92 , SQL:1999, SQL:2003,2008,2011,2016

The ANSI/ISO standard is the most widely accepted, but there were several others over the years: X/OPEN, SAA, FIPS, ODBC

# What are we going to cover in SQL...?

Seems like there must be a lot to SQL if it's been around since the 70's and has gone through that many revisions... the latest SQL standard is over 5000 pages long!

However, SQL is here to stay and the basics are not evolving. As updates are added to the standards, they are generally advanced, progressive updates.

Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

We'll focus on getting a solid grounding in the basics, covering the areas most beneficial to a beginner.

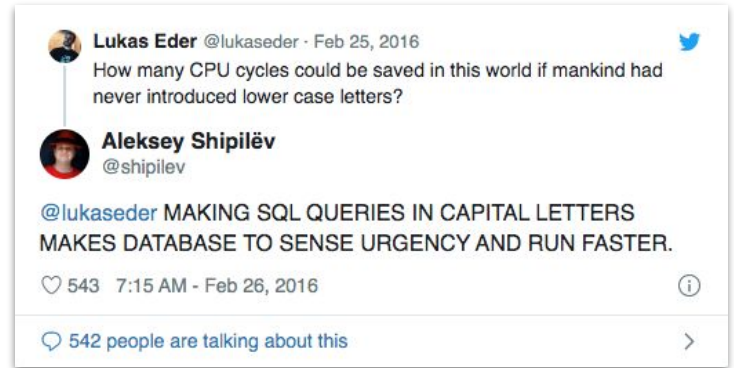
# Naming Conventions

There is no standard, universally accepted naming convention used in SQL. You can find examples of **PascalCase**, **camelCase**, and **snake\_case** being used.

Even “best practices” can be the topics of fierce debates online.

We won't mandate any specific naming conventions in this course because I understand that people may already have a preferred format that they are accustomed to, or are currently using in their work environment. However, *you do need to be consistent*: do not mix and match naming conventions.

I plan to stick to lower **snake\_case** in my examples.



<https://dzone.com/articles/a-guide-to-sql-naming-conventions>

# SQL Reference

When learning a new language you'll want to have solid reference materials to help you along. Fortunately, MySQL has some very solid support available.

There is the MySQL Reference Manual Online:

<https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html>

And possibly even better is the MySQLTutorial:

<https://www.mysqltutorial.org/basic-mysql-tutorial.aspx>

And as shown earlier, this great site with live interaction:

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

The MySQLTutorial is very well documented with many clear examples. I've used plenty of information from that site in preparing the following slides.

# Keys

Keys are an extremely important concept because they are used to ensure that each row in a table is uniquely identifiable, and to establish relationships between tables and secure the integrity of our data.

A key is one or more attributes used to identify other attributes. In our previous example a student has a unique identifier associated with them. Students can have the same first name, the same last name, even the same combination of those two attributes, but their student ID is unique to only them.



<https://findnerd.s3.amazonaws.com/imagdata/8921/8921.jpg>

@GregMisicko

# Key Example

Imagine we have some data about people.

student_id	first_name	middle_name	last_name	ssn
1	John	J	Doe	123-45-6789
2	Jane	A	Doe	987-65-4321
3	Alex		Fraser	333-33-3333

It's not difficult at all to imagine how we'd manage and identify these people, because our dataset is very small. However, what if this dataset grows and eventually we find multiple unique people with exactly the same name? How would you distinguish one 'Alex Fraser' from another? What if any of these people change their names?

# Types of Keys

- **Primary Key:** a column or group of columns which uniquely identifies every row in a table. The primary key must be unique - the same value cannot appear more than once.
- **Super Key:** can uniquely identify any row in the table. A super key is a group of single or multiple keys which identifies rows in a table. However, a superkey may contain additional attributes that are not necessary for unique identification.
- **Candidate Key:** a candidate key is a minimal super key; one without any repeated attributes. The name “candidate key” refers to the fact that the database designer has this as an option when identifying a primary key.
- **Alternate Key:** a candidate key which is not selected to be the primary key.
- **Foreign Key:** a foreign key is a column which is added to create a relationship with another table. It's a field in the table that is the primary key of another table. Every relationship in the model needs to be supported by a foreign key.
- **Secondary Key:** a key that is used strictly for data retrieval.
- **Composite Key:** a key that uses multiple attributes to uniquely identify rows in a table.
- **Compound Key:** The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or may not be a part of the foreign key.
- **Surrogate Key:** An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.

# Types of Keys

So what? Why go into so much detail on these keys when we can just set an auto-increment from the start?

We want to build our databases to be space efficient and highly performant. There are various things we'll focus on throughout the course to address best design practices; keys are just the beginning.



[https://www.google.com/url?sa=i&url=https%3A%2F%2Ftwitter.com%2Fbeakertheangry&psig=AOvVaw3Mtf7VpcD0mFxKEymuC\\_Ukv&ust=1644618310417000&source=images&cd=vfe&ved=0CA\\_sQjRxqFwoTCPjG0e2W9vUCFQAAAAAdAAAAABAE](https://www.google.com/url?sa=i&url=https%3A%2F%2Ftwitter.com%2Fbeakertheangry&psig=AOvVaw3Mtf7VpcD0mFxKEymuC_Ukv&ust=1644618310417000&source=images&cd=vfe&ved=0CA_sQjRxqFwoTCPjG0e2W9vUCFQAAAAAdAAAAABAE)



# Application

Let's say we apply the things we've learned so far to a scenario we know and understand fairly well: Harvard Extension Courses.

Without putting any emphasis on an elegant design yet (we'll get into that later) let's think of what some tables might look like if we needed to store some information about our class.

How might we organize the data model representing students and their classes?

Let's roll up our sleeves and get our hands dirty, as they say...



<https://static1.squarespace.com/static/534f40dfe4b0a74166d6ca5d/t/5b2a72ab8a922d5cdf419d1f/1529508528963/Roll-Up-Our-Sleeves-e1489688927759.jpg?format=1500w>

# Create A Database

The first thing we want to do is create a database inside of our RDBMS. We should know at this point what a database is, and a database management system.

Let's say that we want this particular database to be responsible for data related to school. We're not going to talk through a design process for it, we're not going to plan it carefully - we really just want to slap something together for a quick proof-of-concept.

We can first list what is currently available with our standard MySQL installation, and possibly even look into some of what exists in there:

```
SHOW DATABASES;
```

This command is really simple, just use the following command to create an empty database:

```
CREATE DATABASE <database name>;
```

and then we'll want to start using it:

```
USE <database name>;
```

# Create A Database

A database is of no value if it doesn't hold anything. To see what our newly created database is holding for us we can use this command:

```
SHOW TABLES;
```

And... we see that we have an empty database. With nothing inside of it, it is currently just an empty container or bucket and doesn't serve any useful purpose yet.

We will next create some entities/tables which will contain some attributes, and we'll have these tables built with some relationships to each other.

We are going to start simple here in order to give ourselves the opportunity to do some hands on work before getting into the best practices, design principles and so on.

# Populate A Database

Creating a database was pretty easy, we only needed to think of a name.

Creating a table in our database is a bit more involved. We need to define its attributes, set data types, and identify a primary key.

We will define the tables in the next slide, and after that we'll want to identify the data types to be used with each of the attributes.

# A Simple Database

**students**

<u>student_id</u>	first_name	middle_initial	last_name	gpa
123	John	J	Doe	3.0
456	Jane	T	Smith	3.5
789	Galen	NULL	Erso	4.0

**courses**

<u>course_id</u>	course_name
123	Big Data
456	Databases
789	Python

**enrollments**

<u>course_id</u>	<u>student_id</u>
123	123
456	123
789	456

# Which Data Type should I use?

Remember this from earlier... the link below provides us online reference to MySQL data types and their definitions

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

- Numeric Data Types
  - Integer Types (Exact Value) - INTEGER (INT), SMALLINT, TINYINT, MEDIUMINT, BIGINT
  - Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC
  - Floating-Point Types (Approximate Value) - FLOAT, DOUBLE
  - Bit-Value Type - BIT
- Date and Time Data Types
- String Data Types
  - CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET.
- Spatial Data Types: Used for geometric data storage
- The JSON Data Type: must be a valid JSON document or it will be rejected

# Which Data Type should I use?

Student ID is a number, but which one of the options should I use?

Take a look at your numeric options. There are integer options, and options for numbers requiring decimal values. A student ID is pretty clearly going to be an integer value, but there are five types of integers to choose from. The difference is very clear from the reference material; they vary by the range of values each will support:

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

# Which Data Type should I use for Strings?

Student name fields are strings. Strings are going to be CHAR or VARCHAR data types, but which one should I use? CHAR and VARCHAR differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

<https://dev.mysql.com/doc/refman/8.0/en/char.html>

Value	CHAR ( 4 )	Storage Required	VARCHAR ( 4 )	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Let's be honest - nobody cares much about "storage required" these days. So why would you ever want to use CHAR??? What kind of benefit does it offer? Performance. Its faster for search and for indexing, so it's a good choice *when you have a fixed size field*.



# Which Data Type should I use for Strings? (cont)

Note:

If strict SQL mode is not enabled and you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of non space characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode.

And how do I set up strict SQL mode? There are many options available, but it's clearly documented here:

<https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html>

# Which Data Type should I use for decimals?

We have fixed-point, and floating-point options available. Fixed-point types give us an exact value (the value is not rounded up or down), whereas floating-point values are approximated.

<https://dev.mysql.com/doc/refman/8.0/en/problems-with-float.html>

*Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The FLOAT and DOUBLE data types are subject to these issues. For DECIMAL columns, MySQL performs operations with a precision of 65 decimal digits, which should solve most common inaccuracy problems.*

Fixed-point data types should be used when you need exact results for fractional numbers - for example, when storing financial data. Floating-point data types are more efficient within the RDBMS as they require less storage space and less computational cost.

# Need More Info?

If you're confused, not convinced, or just curious about how it can be that floating-point operations are potentially inaccurate, there are plenty of online resources with more information:

[https://en.wikipedia.org/wiki/Floating-point\\_arithmetic#Accuracy\\_problems](https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_problems)

And sometimes these decisions really matter:

## Incidents [\[ edit \]](#)

- On February 25, 1991, a [loss of significance](#) in a [MIM-104 Patriot](#) missile battery [prevented it from intercepting](#) an incoming [Scud](#) missile in [Dhahran](#), [Saudi Arabia](#), contributing to the death of 28 soldiers from the U.S. Army's [14th Quartermaster Detachment](#).<sup>[31]</sup>

# CREATE TABLE

Full reference for MySQL CREATE TABLE is here:

<https://dev.mysql.com/doc/refman/8.0/en/create-table.html>

... and it's pretty confusing for a beginner. So let's start from where it should make more sense for a beginner.

The basic syntax for the CREATE TABLE statement is as follows:

```
CREATE TABLE <table name> (  
  <column1> <data type> [optional constraint],  
  <column2> <data type> [optional constraint],  
  PRIMARY KEY (<column1>, <column2>, ...),  
  FOREIGN KEY (<column1>, <column2>, ...) REFERENCES <table name>,  
  CONSTRAINT <constraint(s)> );
```

Which still doesn't look very clear, but it should start to make a bit more sense when we look at a real example:

# Populate A Database

Going back to our student table, this is how we might translate it into a SQL statement

<u>student_id</u>	first_name	middle_initial	last_name	gpa
123	John	J	Doe	3.0
456	Jane	T	Smith	3.5
789	Galen	NULL	Erso	4.0

```
CREATE TABLE students
(
    student_id          INT unsigned NOT NULL AUTO_INCREMENT, # Unique ID for the record
    first_name          VARCHAR(20) NOT NULL,                  # Student first name
    middle_initial       CHAR(1),                               # Optional middle initial
    last_name            VARCHAR(20) NOT NULL,                  # Student last name
    gpa                  DECIMAL(2,1) NOT NULL                 # Student grade point average
    CHECK(gpa <= 4 AND gpa >=0),                                # Set a constraint to ensure data integrity
    PRIMARY KEY         (student_id)                           # Make the StudentID the primary key
);
```

@GregMisicko

# Some things to note...

Note the placement of the commas - those are very important. For example, the “CHECK” under gpa is actually the constraint associated with gpa and is effectively a part of that same line. ‘

What is that AUTO\_INCREMENT? Remember we talked about a type of key known as a Surrogate Key: An artificial key which aims to uniquely identify each record.

The following rules are applied when you use the AUTO\_INCREMENT attribute:

- A table can have only one AUTO\_INCREMENT column
- The AUTO\_INCREMENT column must be indexed, which means it can be either PRIMARY KEY or UNIQUE index.
- The AUTO\_INCREMENT column must have a NOT NULL constraint. When you set the AUTO\_INCREMENT attribute to a column, MySQL automatically adds the NOT NULL constraint to the column implicitly.

To ensure a proper GPA value is entered, I use a type of constraint called CHECK, the syntax of which is:  
`CHECK (expression);`

# What if I make a mistake?

If you create a table and realize that it has an issue, of course there are ways to correct it. The **ALTER TABLE** statement will allow you to add a column, alter a column, rename a column, drop a column and even rename a table. As you can imagine, there is a lot of syntax to learn when performing various operations such as those. The quickest and easiest way to do a hard reset when you're just getting started is by deleting the table entirely, and trying again. To remove a table and all of its associated data **permanently** (be careful now), you can simply drop it from the database using:

```
DROP TABLE <table name>;
```

# Work With A Database

We have a database, and it has a table. How do we put data into that table?

```
INSERT INTO students ( first_name, last_name, gpa) VALUES
  ( 'Greg', 'Misicko', '4.0' ),
  ( 'Alex', 'Misicko', '4.0' ),
  ( 'Marina', 'Popova', '3.5' );
```

What happens if we try to insert a record where the GPA doesn't meet our constraint?

What happens if I try to insert the exact same data a second time?

What happens if we try to insert only student\_first\_name, student\_last\_name, ...but not GPA?

(If you've been performing the previous steps against a running MySQL instance you can easily test to find out the answers)



# The UNIQUE Constraint

Sometimes, you want to ensure values in a column or a group of columns are unique. For example, email addresses of users in the users table, or social security numbers of customers in a students table should probably be unique. To enforce this rule, you use a UNIQUE constraint.

A UNIQUE constraint is an integrity constraint that ensures values in a column or group of columns to be unique.

# Populate A Database

## courses

<u>course_id</u>	course_name
123	Big Data
456	Databases
789	Python

```
CREATE TABLE courses
```

```
(
```

```
  course_id INT UNSIGNED NOT NULL,  
  course_name VARCHAR(50) NOT NULL,  
  PRIMARY KEY (course_id),  
  CONSTRAINT UNIQUE (course_name)
```

```
);
```

# The UNIQUE Constraint

To define a UNIQUE constraint for a column when you create a table, you use this syntax:

```
CREATE TABLE table_name(  
    ...,  
    column_name data_type UNIQUE,  
    ...  
);
```

To define a UNIQUE constraint for two or more columns, you use the following syntax:

```
CREATE TABLE table_name(  
    ...  
    column_name1 column_definition,  
    column_name2 column_definition,  
    ...,  
    UNIQUE(column_name1,column_name2)  
);
```