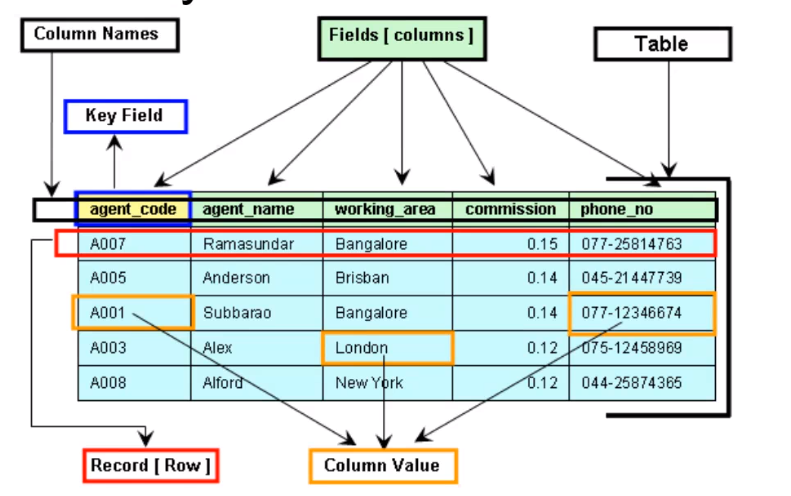
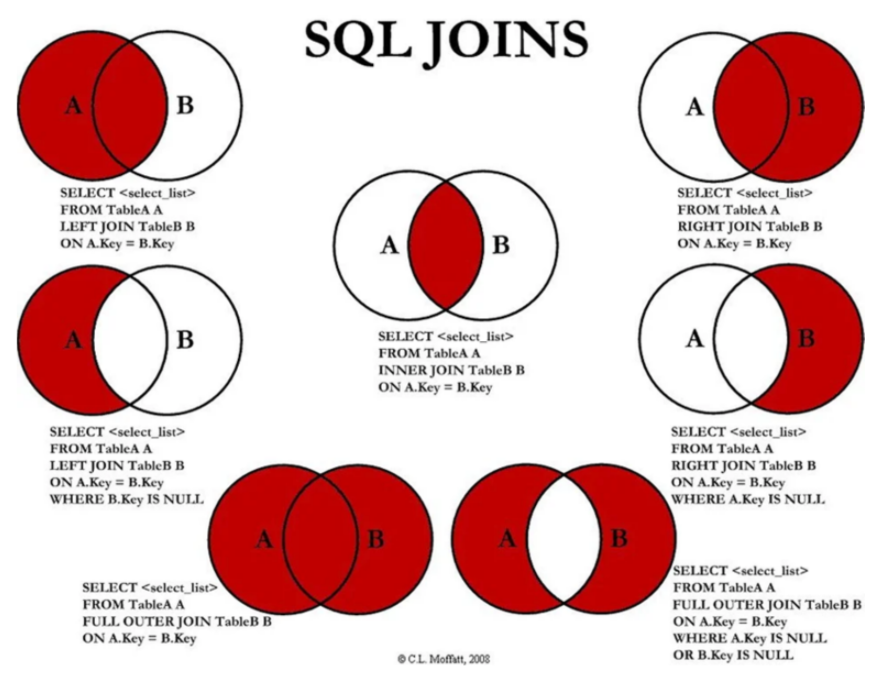
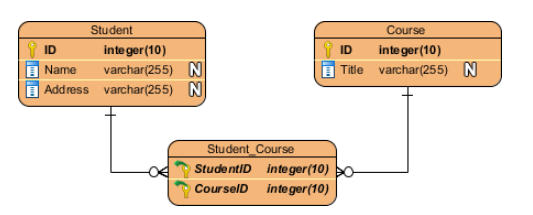
SQL

* Databases contain tables of rows and columns and primary keys
* Primary key: unique identifier for a row
* 
* Relational database: tables can be related to each other by keys
  + Ex: let’s say you have information about student biographical, grades, attendance. Putting them all in one table is disorganized. And having redundant data points creates opportunities for mistakes. So, each type of data can be placed in its own table. A foreign key (ID) can be used to find the relationship between the different tables to figure out the desired aspect for a specific student.
* Logging in to a mysql database: mysql -h <host> -u <username> -p
* Running sql commands stored in a script: mysql -h <host> -u <username> -p -D <database> < script.sql > output.txt
* Primitive data types
  + Numeric
    - INT: 11 digits
    - TINYINT: 4 digits
    - SMALLINT: 5 digits
    - MEDIUMINT: 9 digits
    - BIGINT: 20 digits
    - FLOAT(M,D): M length with D decimals
    - DOUBLE(M,D): M length with D decimals
    - DECIMAL(M,D): M length with D decimals
  + Time and Date
    - DATE: YYYY-MM-DD
    - DATETIME: YYYY-MM-DD HH:MM:SS
    - TIMESTAMP: YYYYMMDDHHMMSS
    - TIME: HH:MM:SS
    - YEAR(M): either 2 digit or 4 digit
  + Text and String
    - CHAR(M): 255 characters
    - VARCHAR(M): 255 characters
    - BLOB: 65535 characters
    - TINYBLOB: 255 characters
    - MEDIUMBLOB: 16777251 characters
    - LONGBLOB: 4294967295 characters
    - ENUM: limit defined by the table definition
  + Spatial
    - GEOMETRY: any geometry value
    - POINT: single value
    - LINESTRING: at least 2 points
    - POLYGON: at least 4 points
    - MULTIPOINT: a collection of points
    - MULTILINESTRING: a collection of linestrings
    - MULTIPOLYGON: a collection of polygons
    - GEOMETRYCOLLECTION: a collection of any spatial types
  + JSON
    - 255 characters limit
* Commands for read operations
  + SHOW DATABASES; to show the databases on the server
  + USE database\_name; to enter into the database named database\_name
  + SHOW TABLES; to show the tables in the current database
  + SELECT column\_names FROM table\_names; to print a table containing the listed column\_names from table\_names
    - SELECT
      * Use \* for the column name to get everything from the table
      * SELECT COUNT(column\_name) returns a count for the number of records selected
      * Doing math: SELECT num\_column\_1 <math operation> num\_column\_2: do math using values from num\_column\_1 and num\_column\_2
      * max(), min(), avg(), sum(), std()
      * AS alias\_name: in the resulting table, the column selected is named alias\_name
    - FROM
      * 
      * INNER JOIN: data from both tables where the foreign key is found in both tables
      * LEFT JOIN: data from both tables where the foreign key is found in the left table
      * RIGHT JOIN: data from both tables where the foreign key is found in the right table
  + WHERE condition: add to a select statement to filter the selected rows by condition
    - Can create compound conditions using AND, OR
    - number\_type\_column <an in/equality operator> <numeric value>
    - text\_type\_column LIKE ‘a string’
      * % included in the string is treated as a wildcard
      * There’s also a NOT LIKE operator
    - column\_name IN (an, array, of, stuff): match only records where column\_name’s value is found in the array
  + DESCRIBE table; to get information about table
  + ORDER BY column: add to a select statement. Orders output rows based on the value of a column in increasing order, non-case sensitive
  + DESC: add to an order statement. Flips the order to decreasing order. Case sensitive
  + GROUP BY column\_name: groups records by the value of column\_name, then perform the query on each group
    - To apply WHERE restrictions to a group, use HAVING
    - Aggregate functions can be used in HAVING
  + TIMESTAMPDIFF(unit, begin, end): calculate time between begin and end in unit
    - CURDATE(): the current date
* Commands for write operations
  + Creating tables
    - CREATE TABLE table\_name (field\_name\_1 TYPE PRIMARY KEY, field\_name\_2 TYPE, field\_name\_3 TYPE);
    - Add AUTOINCRIMENT after PRIMARY KEY to get it to autoincriment
  + Inserting content into tables
    - INSERT INTO table\_name (field\_name\_1, field\_name\_2, field\_name\_3) VALUES (value\_1, value\_2, value\_3)
  + Updating records
    - UPDATE table\_name SET field\_name = new\_value WHERE <condition to select the records to update>
  + Deleting records
    - DELETE FROM table\_name WHERE <condition to select the record to delete>
  + Altering a table
    - ALTER TABLE table\_name ADD field\_name TYPE; to add a field to a table
      * Add default default\_value to the end to define a default value
    - ALTER TABLE table\_name DROP COLUMN column\_name; to drop a column
    - ALTER TABLE table\_name MODIFY COLUMN column\_name data\_type; to change the data type of a column
  + Deleting a table
    - DROP TABLE table\_name;
* Other commands
  + SHOW GRANTS FOR username;
    - Show the privileges available to username
    - SELECT, INSERT, UPDATE, DELETE, DROP, CREATE, ALTER
  + GRANT permission\_name ON database\_name TO username;
    - Grants permission\_name on database\_name to username
    - permission\_name of ALL grants all possible privileges, except for using GRANT itself
    - permission\_name of GRANT OPTION grants all possible privileges, including for using GRANT itself
  + (bash) exporting a sql table to a file: mysqldump --single-transaction -p -h <hostname> -u <username> <database name> <table name> > <output file name>
    - Name the file <table name>.sql
  + (bash) importing a sql table from a file: mysql -h <hostname> -u <username> -p -D <database name to import to> < <file to import>
    - Created table has name <table name> from <table name>.sql
  + CREATE VIEW view\_name AS <query>
    - Stores the result of <query> as view\_name
    - To use, just put the view\_name into the FROM section like with any table name
* Schema
  + Diagrams used to describe systems
  + Unified Modeling Language (UML): indicates the name of an entity, what fields it has

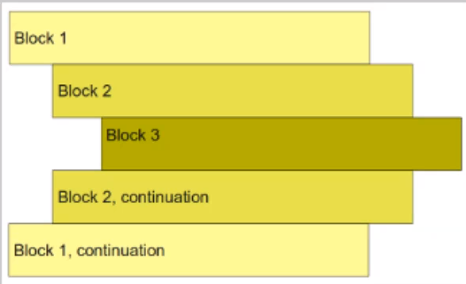
| some\_table |
| --- |
| some\_field\_1  some\_field\_2  some\_field\_3 |

* + Entity Relationship Diagram (ERD): describes tables and fields and how they’re related
    - Describes normalized databases: data that is broken down into tables of related data
    - Highlights foreign key relationships: keys from different tables that shows correspondence between records on different tables
      * Ex: the author of a book can be associated with the bio of the author through the author name (foreign key)
    - Cardinal relationships
      * One to one
        + One record on one table can only correspond to one record on the other table
        + Ex: a table for usernames and a table for passwords
        + 
      * One to many
        + One record on one table can correspond to many records on the other table
        + Ex: a table for mothers and a table for children
        + 
      * Many to many
        + Many records from one table can correspond to many records on the other table
        + Requires going through a junction table

Ex: a table for student info, a table for course info. Many to many relationship goes through a junction table for that associates student ids with course codes

* + - * + 

Python

* Popular for data science
* Both compiled and interpreted
  + Compiled python files have a \*.pyc extension
  + Usually importing a function from another file will create compiled files as cache to help performance
* Used as a scripting language and programming language
* Object-oriented programming
  + Non-scalar object: made up of scalars
    - Ex: Lists, tuplets,
  + Scalar object: can’t be subdivided anymore
    - Ex: Int, Float, Bool, NoneType
* Type python3 in the command line to access the python REPL
* Use # to start a comment
* No semicolons
* Use input(prompt) to prompt for input
* Naming convention is snake\_case for variables and functions
* PascalCase for classes
* Scripts
  + #!/usr/bin/python3 as the first line of a file on Unix systems to point to the Python interpreter
  + Files typically end in .py
  + python3 file.py or ./file.py to execute
* Math
  + Follows PEMDAS
  + +: add
  + -: subtract
  + \*: multiply
  + /: divide
  + \*\*: raise to the power of
  + %: mod
  + not a: represents the opposite value of a
  + a and b:
  + a or b:
  + >: greater than
  + <: less than
  + >=: greater than or equal to
  + <=: less than or equal to
  + ==: equal to
  + !=: not equal to
  + Can do compound inequalities: ex: 1 < x < 5
  + math module
    - Import math
    - Has constants and functions (ex: acos(), log2(), etc)
    - Would access using the . operator on math. Ex: math.log2(1)
* Casting
  + Explicit casting
  + <destination type>(input)
  + Ex: int(3.9) casts the float 3.9 to int 3
  + All values involved in a concatenation must be of the same type
  + Implicit casting
  + In a math operation, if at least one of the values involved is a float, then the result would be a float. Else, int
  + Except that in division, the result is always a float
* Flow control
  + 
  + Grouping blocks by indentation
  + Typically 4 spaces are used per indentation but must be consistent
  + if <condition>:
  + <expression>
  + elif <condition>:
  + <condition>
  + else:
  + <condition>
  + #without the < >
  + while <condition>:
  + <expression>
  + for <variable> in <iteratable>:
  + <expression>
  + # range(num) returns a list from 0 inclusive to num exclusive
  + # range(start, stop)
  + # range(start, stop, step)
  + # break will end the loop in the context where break was put in
  + # Use for loops if you know it should iterate a certain number of counts
  + # use while loops if you want to iterate something until something happens
* String
  + Is a list of individual characters, so can be iterated in a loop, and can use [ ] notation
  + Immutable: when a variable stores a different string, a new string was created and stored. The original string couldn’t be modified
  + len(string) returns the length of string
  + Can do array[index] to get the element at index
    - A negative index would wrap around, with the last element being -1, second to last being -2, etc
    - Index numbers larger than length will result in an error
  + Slicing: array[start:stop:step]
    - Can omit step to use default step size of 1
    - Returns a list starting from start inclusive, counting by step, until stop exclusive
    - “abcdefg”[3:6] = “def”
    - “abcdefg”[3:6:2] = “df”
    - Can count backwards using a negative step size
    - [::] implies start at 0 and go to the end, step by 1
    - [::-1] implies start at the end, step by -1 to 0
* Tuples
  + A compound data type of indexed elements
  + Elements can be iterated, accessed and sliced just like strings and lists
  + Conventionally, can contain mixed data types
  + immutable
  + empty = ()
  + populated = (“a”, 2, 1.5)
  + Can be concatenated like (1, 2, 3) + (“a”, “b”, “c”) == (1, 2, 3, “a”, “b”, “c”)
  + Lonely tuples, which contain only one element, have a comma like (1,)
  + Can also pass a tuple into len to get its length
  + Tuples can be used to swap variables, like (x, y) = (y, x)
  + Functions that return multiple values can return them in a tuple
    - Ex: (quotient, remainder) = divide(1, 2)
* Lists
  + Just like a tuple, but with some key differences
    - Uses [ ] instead of ( )
    - Is mutable
    - Conventionally, does not mix data types
  + Storing a list in a variable only stores the memory address (shallow copy)
    - Use copy.deepcopy(list) to create new copy in memory
  + Can concatenated, spliced, be accessed, iterated just like tuples
    - Concatenating lists creates a new list in memory
  + To add an element to the list, use <list>.append(el)
  + To add multiple elements to the list, use <list>.extend([list, of, what, to, add])
  + To delete an element from a certain index, use del(<list>[index])
  + To delete the last element and return it, use <list>.pop()
  + To remove the first instance of el found, use <list>.remove(el)
* Functions
  + def name(param):
  + “””
  + Multiline
  + Documentation
  + “””
  + Body
  + name(args) # to call
  + To import a function from another file, do from <filename without .py> import <function name>
    - Call it by that function name
  + Variables inside a function are scoped globally within that function
* Programming practices
  + Define the docs for a function before implementing it, so you know what you’re doing
  + Use version control to be able to backtrack on things
  + When functions get complicated, put each function into its own file for organization
  + Abstraction: the user of an algorithm doesn’t need to know how it’s implemented; they only need to know inputs and outputs
  + Problem decomposition
* Classes
  + class ClassName:
  + field\_1 = value
  + field\_2 = value
  + # fields are public, so can be accessed like instance.field\_1
  + def \_\_init\_\_(self, some, args):
  + # this is the constructor
  + # use `self` to refer to this instance
  + # to construct, do `instance = ClassName(some, args)`
  + # note that `self` is passed in implicitly
  + def method\_1(self, some, args):
  + # also needs the implicit `self` in order to refer to this instance
  + # to use, do `instance.method\_1(some, args)`
  + def method\_2():
  + # stuff

Machine Learning