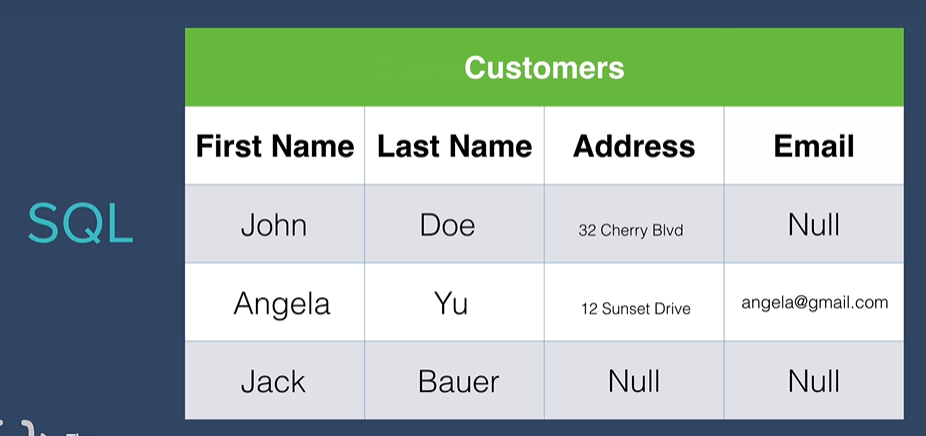
Section 24: Databases

# SQL vs NoSQL

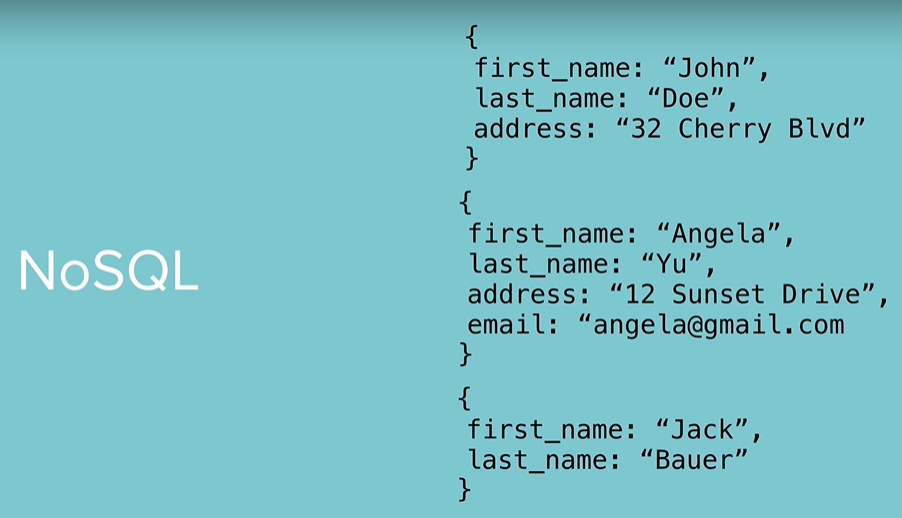
**SQL:** Structured Query Language

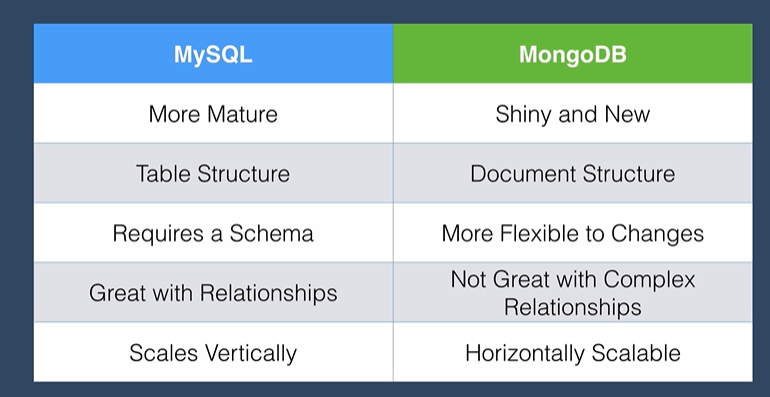
* Pro: Data is sorted in tables, in a similar style to excel. This makes it easy to understand. Relationships between data well established. Separate tables can be brought together later on.
* Con: not so flexible. When there is missing information, it automatically adds a “null”. If one object has a new category, it is added to the table. All other objects that do not have this category will need to put a “null” in its place. Not very scalable.
* Best for: order, customer details, products, inventory
* Examples of SQL Databases: MySQL, PostgreSQL



**NoSQL**: Not Only Structured Query Language

* Pro: Data is sorted as JSON objects. Therefore, SQL’s issue with null datatypes does not occur. More scalable
* Con: relationships between data is not well established.
* Best for: users, profiles with posts, one user with many posts
* Examples of NoSQL Databases: mongoDB, redis





**CRUD**: **C**reate **R**ead **U**pdate **D**estroy

# SQL

|  |  |
| --- | --- |
| **Command** | **Description** |
| CREATE TABLE *table\_name* (  *column\_name1* *datatype*,  *column\_name2* *datatype*,  …  PRIMARY KEY(*column\_name*)  FOREIGN KEY(*column\_name*) REFERENCES table\_name2(*column\_name*)  ); | Create a new table with specified columns. Each column is represented by a specified datatype.  Adding a “NOT NULL” after the datatype indicates that the column cannot be null  The PRIMARY KEY is used to identify each record. The FOREIGN KEY is optional and used to create connections between tables |
| INSERT INTO *table\_name*  VALUES (*value1*, *value2*, …);  INSERT INTO *table\_name* (*column\_name1*, *column\_name2*, …)  VALUES (*value1*, *value2*, …); | Insert specified values into specified table.  Insert specified values into specified table and columns. |
| SELECT \* FROM ‘*tableName’*;  SELECT *column\_name1*, *column\_name2*, …  FROM *table\_name*;  WHERE *condition*; | Show all records in specified table  Show records in specified columns of specified table  Adding WHERE at the end of the code will make it so that a condition has to be evaluated |
| UPDATE *table\_name*  SET *column\_name1* = *value1*, *column\_name2* = *value2*, …  WHERE *condition;* | Update specified table to set values for specified columns  Adding WHERE at the end of the code will make it so that a condition has to be evaluated. This is important if you want to specify which record should be updated (if not specified, all values in a column will be updated) |
| DELETE FROM *table\_name*  WHERE *condition;* | Delete records from specified table based on a condition |
| ALTER TABLE *table\_name*  ADD *column\_name datatype*; | Create a new column in a specified table |
| SELECT *table1*.*column\_name, table2.column\_name1, table2.column\_name2, …*  FROM *table1*  INNER JOIN *table2* ON *table1*.*column\_name* = *table2*.*corresponding\_column\_name*; | Joins two tables together. Searching for certain information in each table and combine it into one table  The INNER JOIN specifies a column that is shared between the tables (ex. orders.customer\_id = customers.id ) |

**Primary Key**: a column that uniquely identifies each record in a database table. Each record has an original value for this column, no two records are the same. (Typically, it is the “id” column.)

**Foreign Key**: used to connect a column in the current table to a column from a different table

Example for creating a table:

CREATE TABLE orders(

id INT NOT NULL,

order\_number INT,

customer\_id INT,

product\_id INT,

PRIMARY KEY(id),

FOREIGN KEY(customer\_id) REFERENCES customers(id),

FOREIGN KEY(product\_id) REFERENCES products(id)

);

# MongoDB

Data directory: C:\Program Files\MongoDB\Server\4.4\data\

|  |  |
| --- | --- |
| **Command** | **Description** |
| mongod | Used to start up the mongo server |
| mongo | Used to tap into the mongo shell. You can tell it is working when the carrot > shows up on the left |
| help | If you need help: cannot find a command, want to see list of commands, etc. |
| show dbs | Show database names |
| show collections | Shows collections in current database |
| db | Show which database we are currently working in |
| use *database\_name* | Create a new database. Automatically switches to the new database as well. |
| db.collection.insertOne(  {  *field\_name*: *field\_value,*  *field\_name*: *field\_value,*  *…*  }  )  db.*collection\_name*.insertMany(  { … },  { … },  ….  ) | Insert a new entry  Insert multiple new entries |
| db.*collection\_name*.find()  db.*collection\_name*.find(  { *search\_field*: { *condition; }* },  { *field\_to\_return*: 1; }  ….  ).limit(*num*); | See all entries in a collection  Used to find specific entries.  You can find fields by specified value or using a conditional in the first line. You use query operators for conditionals.  In the second line, you specify which field you would or would not like to get back and set it to either 1 (true) or 0 (false). If you don’t add this, you will get all fields returned  .limit is optional and used to find only a specific number of entries |
| db.*collection\_name*.updateOne(  { *field\_name*: *value*},  { $set: {*field\_name*: *value*}}  ) | Used to update an entry. The first line is used to find the entry. The second line is where you specified what to update. |
| db.collection\_nane.deleteOne(  { *field\_name*: *value\_or\_*{*condition*};  )  db.collection\_name.deleteMany() | Delete an entry  Same as other method but used to delete multiple entries |
| db.dropDatabase() | Delete the current database (note: make sure you are in the database. “use” command puts you in the database. |

|  |  |
| --- | --- |
| **Query Operators** | **Description** |
| $eq / $ne | Equal to / Not equal to |
| $gt / $gte | Greater than / Greater than equal to |
| $lt / $lte | Less than / less than equal to |
| $in / $nin | If in a specified array / If not in a specified array |
| $and | Both conditions true |
| $or | Either condition true |
| $not | Not a certain condition |
| $nor | Both conditions not true |

# Mongoose

Mongoose is an **ODM** (Object Document Mapper). It allows your Node.js app, which speaks the language of Javascript objects, to be able to talk to your MongoDB database, which speaks in the language of documents, collections, and databases.

Mongoose must be installed using “npm install mongoose”.

# Resources

* W3Schools on SQL: <https://www.w3schools.com/sql/>
  + To find various datatypes, go to the “SQL Data Types” page
* SQL Playground: <https://sqliteonline.com/>
* Mongoose Documentatino: <https://docs.mongodb.com/>