# PostgreSQL: Node.js Client

# Node-postgres

- **npm library:** `npm install pg --save`
- ***database driver***
- **implements the postgres protocol in a Node module (JS!)**
- **Gives us a `client` object that we can pass SQL to**
- **Asynchronously talks via postgres protocol / TCP to postgres**
- **gives us a callback with `rows` array of resulting table**

# Read from PostgreSQL with Node.js

```
//include the node postgres library
var pg = require('pg');
//connect to a database
pg.connect('postgres://user:password@localhost/my_db', function(err, client, done) {
  //request all of the hats
  client.query(`select * from hats`, function(err, result) {
    console.log(result.rows);
    //let pg know we're done with this client
    done();
    //close the pg pool entirely.
    //this is done so our node process will exit.
    pg.end();
  });
});
```

# Write to PostgreSQL with Node.js

```
//include the node postgres library
var pg = require('pg');
//connect to a database
pg.connect('postgres://user:password@localhost/my_db', function(err, client, done) {
  //add a new hat
  client.query(`insert into hats
        (name, material, height, brim)
        values
        ('cowboy', 'straw', '4', true)`, function(err, result) {
    //should print 'INSERT: 1'
    console.log(`${result.command}: ${result.rowCount}`);
    //call done and end, same as the read example
    done();
    pg.end();
  });
});
```

# Query Parameter Substitution

- Instead of constructing strings, `pg` will do substitution for you
- Values are passed as an array to the `.query()` method
- Substitutions are denoted by a dollar sign and a number corresponding to their 1-based position in the array

```
client.query('select * from hats where material = $1', ['felt'], function(err, result) {
    //result now has rows where the hat material is `felt`
});
```

# Exercise

- Create a Node.js application that takes in one parameter from the command line (`process.argv[2]`), which is a user's name.
- It then finds all the hats that belong to that user.

# Error Checking

- An exception can happen either when the connection configuration is incorrect...

```
//this will cause an error
pg.connect('postgres://bad:user@localhost/not-a-database', function(err, client, done) {
    //`err` will contain error information including a message: "authentication failed for user"
    if(err){
        //passing `client` to `done` will remove it from the connection pool.
        if(client) {
            done(client);
        }
        return;
    }
})
```

- ... or when a query has a problem.

```
client.query(`select * from does_not_exist`, function(err, result) {
    //`err` will contain error information, including  amessage letting you know that `does_not_exist` does not exist.
    if(err){
        return done (client);
    } else {
        done();
    }
})
```

# Separate Adapter

- Rather than having each module manage a client, reuse a single adapter
- Prevents clients from leaking
- Keeps error handling in one place

# Separate Adapter (part two)

```
//saved as `query.js`
var pg = require('pg');
var connectionString = "postgres://user:password@localhost/my_db";

//export the adapter function
module.exports = function(queryString, queryParameters, onComplete) {
  //normalize parameters, allowing only passing a query string and an optional `onComplete` handler
  if (typeof queryParameters == 'function') {
    onComplete = queryParameters;
    queryParameters = [];
  }

  //everything else is almost the same as before, replacing hard-coded strings and arrays with parameters
  pg.connect(connectionString, function(err, client, done) {
    if (err) {
      console.log(`error: connection to database failed. connection string: "${connectionString}" ${err}`);
      if (client) {
        done(client);
      }
      //check if `onComplete` exists before calling
      if (onComplete) {
        onComplete(err);
      }
      return;
    }

    client.query(queryString, queryParameters, function(err, result) {
      if (err) {
        done(client);
        console.log(`error: query failed: "${queryString}", "${queryParameters}", ${err}`);
      } else {
        done();
      }
      //check if `onComplete` exists before calling
      if (onComplete) {
        onComplete(err, result);
      }
    });
  });
};
```

# Separate Adapter (part three)

- The adapter can now be `required` from anywhere in the application.

```
//say we wanted to define another file in the same directory as `query.js`
var query = require('./query');

query('select * from hats where material = $1', ['felt'], function(err, results){
      //handle the error and results as appropriate.
});
```