Quantum Database Jumpers

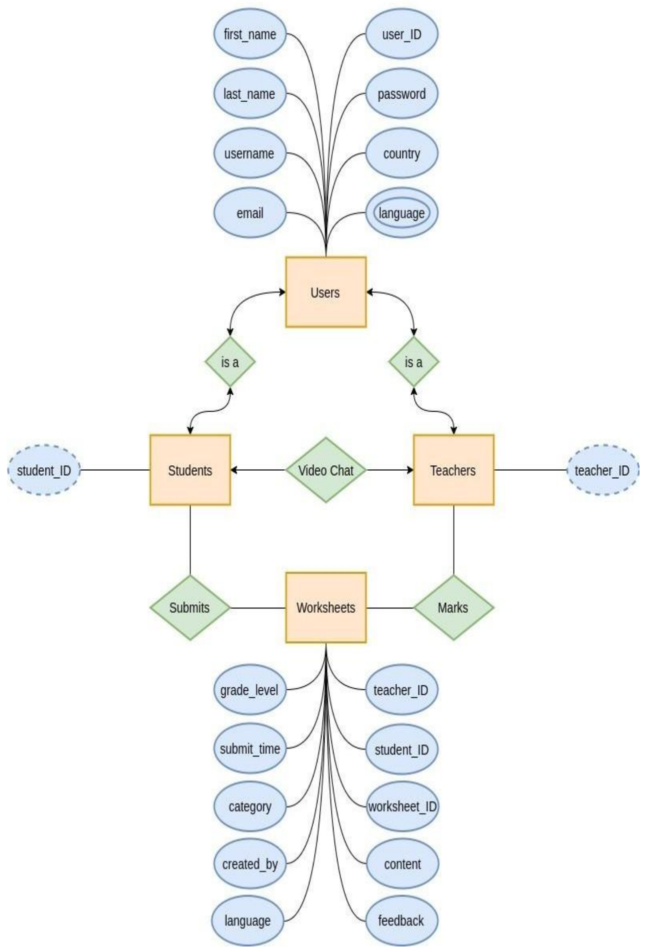
Oliver Tonnesen
Parm Johal
Nat Dring
Paige Loffler
Braydon Horcoff

Scenario 1 - Math education website

Assignment 1

- Analysed and developed a schema design for our website
- Determined several assumptions that would be safe to make for our chosen scenario
- Came up with a preliminary entity relation diagram to model our schema

Assignment #1: Entity-Relationship Diagram



Requirements:

- Contact details
- User level in home country
- Student progress on the site
    - Level of math worksheets
    - Marks allocated
    - Teacher comments
- Teacher details
    - Contact details
    - Availability
    - Languages
- Worksheet details
    - Level
    - Created by
    - Category

Assumptions:

- Upload PDF of worksheets to a hosting service
- Teachers can also be students (And vice versa)
- Platform allows for video conferences between teachers and students

Main objectives of the system:

- The system will help registered students (maybe even people not going to school) to become educated in all grade levels, including university/college.
- The system will allow anyone with internet access to study and learn math using worksheets, allowing users to receive feedback on their work to aid in their understanding

Tasks performed by different users during a typical day:

For students, the main tasks will relate to handling the worksheets provided to them by the teachers/system. They will also be provided with the resources to understand and submit their work. The tasks pertaining to students include:

- Registration (new students)
- Provide necessary information (ie. Full name, credit card/payment information, grade level to work in)
- Downloading the worksheet
- Working on the worksheet
- Completing the worksheet
- Using resources to help with work
- Submitting the worksheet
- Deciding whether to move to the next grade level, stay at the same grade level, or move down a grade level

Tasks performed by different users during a typical day:

For teachers, they will mainly provide the user with feedback and marking the worksheets submitted by students. Their tasks include:

- Registration (new teachers)
- Provide necessary information
- Downloading student-submitted worksheets
- Marking worksheets
- Giving grades and feedback for each submitted worksheet

Data types associated with these tasks:

For the website:

- PDFs
- JPEGs
- A skype profile

For the database:

- unsigned-Integers - Grades on papers
- String - Names of students, assignments and teachers

Scope of the project and relevant data:

For materials:

- Multiple worksheets for each grade level
- Recommended resources for extra help
- Marked worksheets with feedback

For users:

- Student information
- Teacher information
- Skype accounts

Possible outputs with the given data:

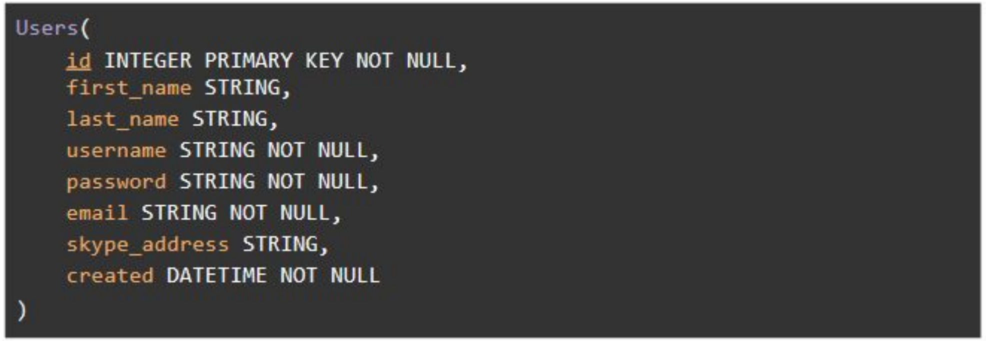
- Worksheets downloaded as a pdf file, then printed off
- Submitted work scanned/uploaded as a jpeg/pdf file.
- Student id after registration
- Submitted worksheets assigned to teachers to mark in a pdf format
- Feedback to be output as a text file or attached as a pdf to be returned with the marked worksheet

Assignment 2

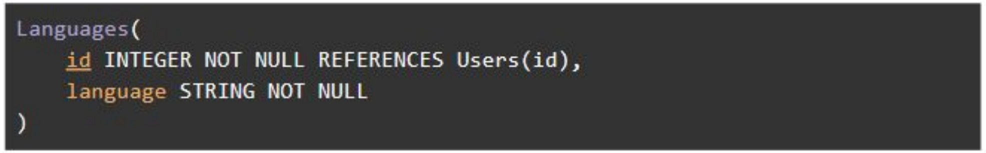
- Finalized entity relationship diagram
- Converted entity relationship diagram from assignment 1 to a relational model
- Normalized the resulting relational model in preparation for assignment 3

Normalized relational model

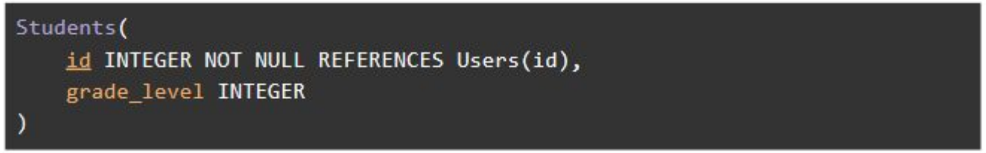
- Used primary keys from entity relationship model
- Added foreign keys to certain tables
- Added extra tables for multiple attributes in entity relationship model



```
Users(
    id INTEGER PRIMARY KEY NOT NULL,
    first_name STRING,
    last_name STRING,
    username STRING NOT NULL,
    password STRING NOT NULL,
    email STRING NOT NULL,
    skype_address STRING,
    created DATETIME NOT NULL
)
```



```
Languages(
    id INTEGER NOT NULL REFERENCES Users(id),
    language STRING NOT NULL
)
```



```
Students(
    id INTEGER NOT NULL REFERENCES Users(id),
    grade_level INTEGER
)
```

Assignment 3

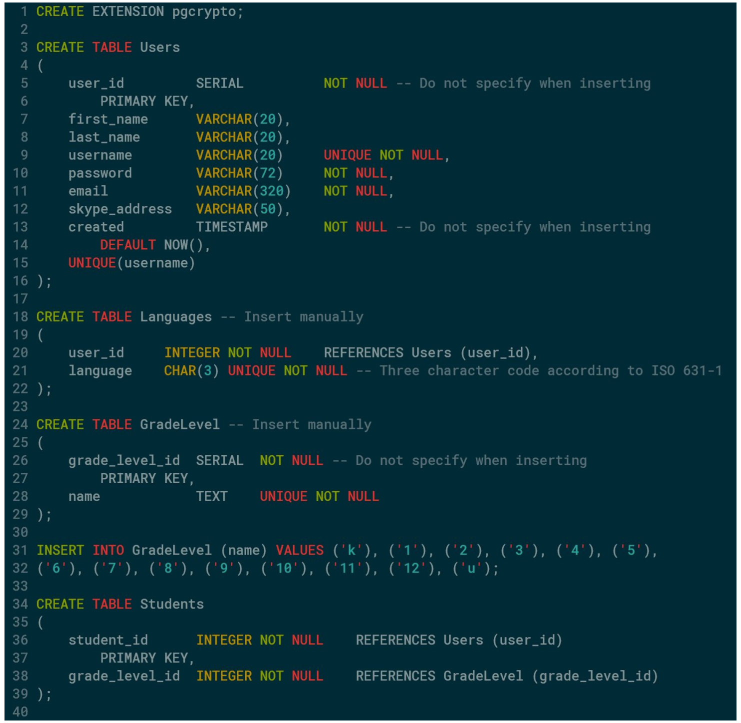
- Web app
  - Allowed easier interfacing with database
  - More closely resembled what a final product for this scenario might look like
- HTTP API
  - Communication between web client and python server
- psycopg2
  - Interface between python server and postgres server

Challenges

- UVic linux server environment too restricted
  - Unable to install crucial python packages
  - With some effort, we circumvent this issue by setting up postgres servers to run in our own personal development environments
- Interfacing between several languages, clients and servers
  - Had to coordinate Python, SQL, JavaScript, and HTML using several different APIs and interfaces

Database schema

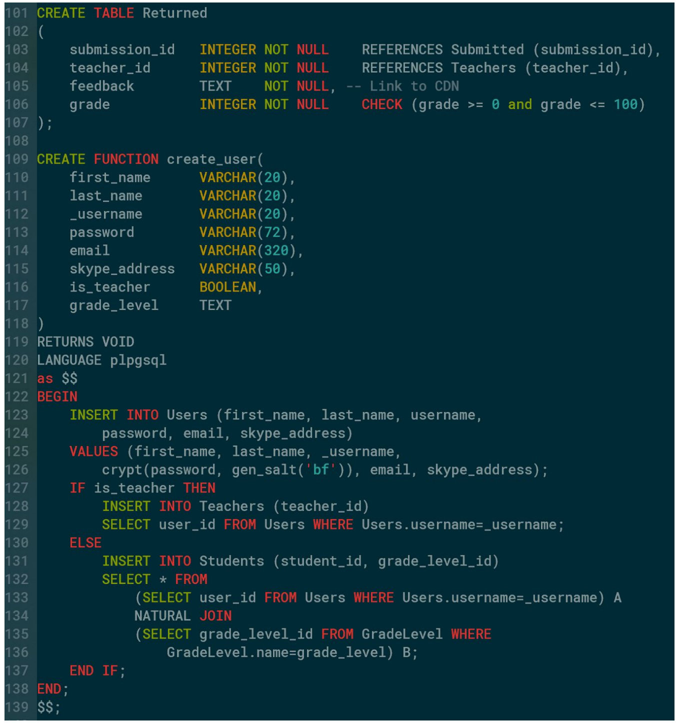
- Schema was normalized for assignment 2
- Straightforward implementation, mostly boilerplate
- Wrote some sample stored procedures to simplify data insertion from python



```sql
CREATE EXTENSION pgcrypto;

CREATE TABLE Users
(
    user_id         SERIAL          NOT NULL -- Do not specify when inserting
        PRIMARY KEY,
    first_name      VARCHAR(20),
    last_name       VARCHAR(20),
    username        VARCHAR(20)     UNIQUE NOT NULL,
    password        VARCHAR(72)     NOT NULL,
    email           VARCHAR(320)    NOT NULL,
    skype_address   VARCHAR(50),
    created         TIMESTAMP       NOT NULL -- Do not specify when inserting
        DEFAULT NOW(),
    UNIQUE(username)
);

CREATE TABLE Languages -- Insert manually
(
    user_id     INTEGER NOT NULL    REFERENCES Users (user_id),
    language    CHAR(3) UNIQUE NOT NULL -- Three character code according to ISO 631-1
);

CREATE TABLE GradeLevel -- Insert manually
(
    grade_level_id  SERIAL  NOT NULL -- Do not specify when inserting
        PRIMARY KEY,
    name            TEXT    UNIQUE NOT NULL
);

INSERT INTO GradeLevel (name) VALUES ('k'), ('1'), ('2'), ('3'), ('4'), ('5'),
('6'), ('7'), ('8'), ('9'), ('10'), ('11'), ('12'), ('u');

CREATE TABLE Students
(
    student_id      INTEGER NOT NULL    REFERENCES Users (user_id)
        PRIMARY KEY,
    grade_level_id  INTEGER NOT NULL    REFERENCES GradeLevel (grade_level_id)
);

```

Password hashing

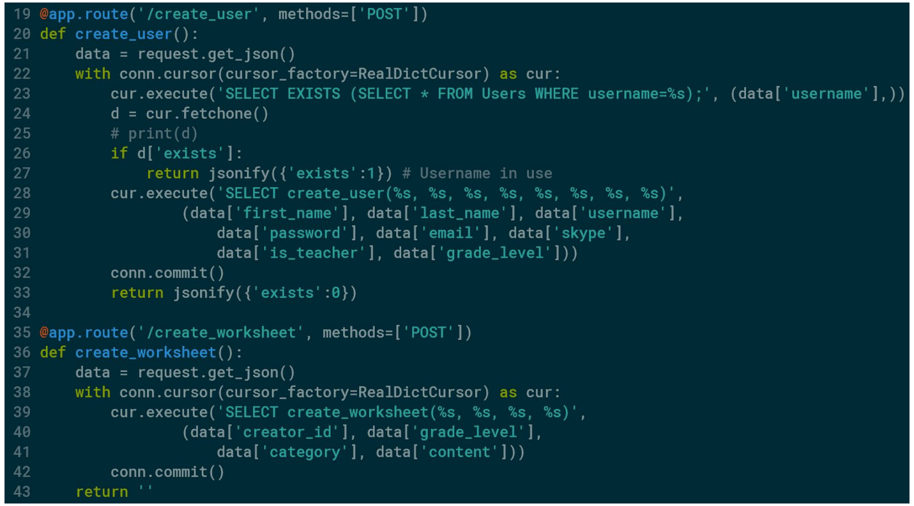
- Hashed with the blowfish cipher
- Salted with built in gen_salt method
- Password salting/hashing is done directly on the database automatically when a user is added



```sql
CREATE TABLE Returned
(
    submission_id   INTEGER NOT NULL    REFERENCES Submitted (submission_id),
    teacher_id      INTEGER NOT NULL    REFERENCES Teachers (teacher_id),
    feedback        TEXT    NOT NULL, -- Link to CDN
    grade           INTEGER NOT NULL    CHECK (grade >= 0 and grade <= 100)
);

CREATE FUNCTION create_user(
    first_name      VARCHAR(20),
    last_name       VARCHAR(20),
    _username       VARCHAR(20),
    password        VARCHAR(72),
    email           VARCHAR(320),
    skype_address   VARCHAR(50),
    is_teacher      BOOLEAN,
    grade_level     TEXT
)
RETURNS VOID
LANGUAGE plpgsql
as $$
BEGIN
    INSERT INTO Users (first_name, last_name, username,
        password, email, skype_address)
    VALUES (first_name, last_name, _username,
        crypt(password, gen_salt('bf')), email, skype_address);
    IF is_teacher THEN
        INSERT INTO Teachers (teacher_id)
        SELECT user_id FROM Users WHERE Users.username=_username;
    ELSE
        INSERT INTO Students (student_id, grade_level_id)
        SELECT * FROM
            (SELECT user_id FROM Users WHERE Users.username=_username) A
            NATURAL JOIN
            (SELECT grade_level_id FROM GradeLevel WHERE
                GradeLevel.name=grade_level) B;
    END IF;
END;
$$;
```

Python HTTP server

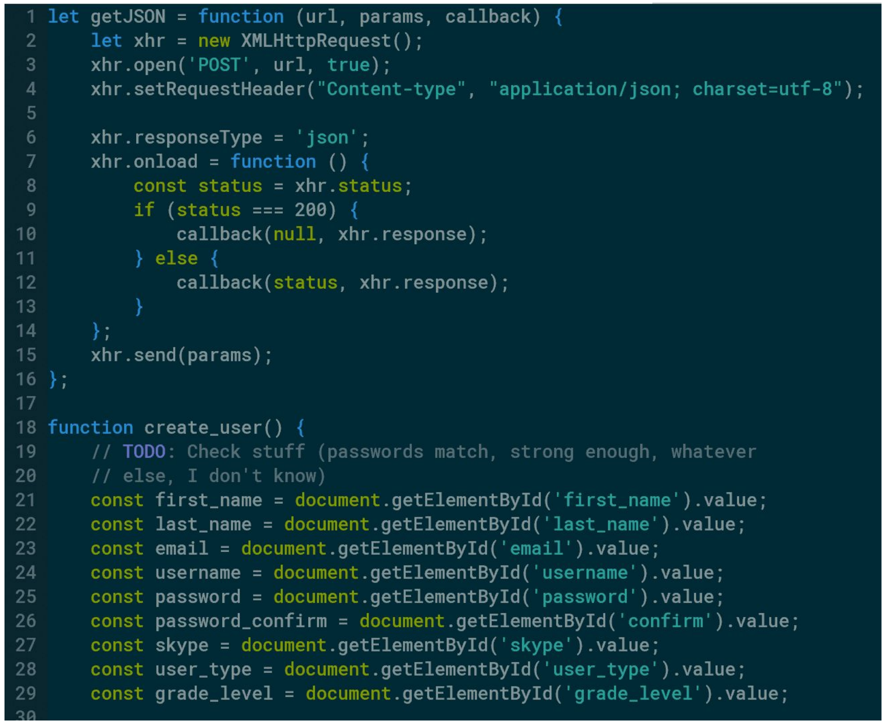
- Simple Flask HTTP server
- Accepts data in POST requests and inserts it into postgres server
- Renders templates to show results from a selection of queries



```python
@app.route('/create_user', methods=['POST'])
def create_user():
    data = request.get_json()
    with conn.cursor(cursor_factory=RealDictCursor) as cur:
        cur.execute('SELECT EXISTS (SELECT * FROM Users WHERE username=%s);', (data['username'],))
        d = cur.fetchone()
        # print(d)
        if d['exists']:
            return jsonify({'exists':1}) # Username in use
        cur.execute('SELECT create_user(%s, %s, %s, %s, %s, %s, %s, %s)',
                (data['first_name'], data['last_name'], data['username'],
                    data['password'], data['email'], data['skype'],
                    data['is_teacher'], data['grade_level']))
        conn.commit()
        return jsonify({'exists':0})

@app.route('/create_worksheet', methods=['POST'])
def create_worksheet():
    data = request.get_json()
    with conn.cursor(cursor_factory=RealDictCursor) as cur:
        cur.execute('SELECT create_worksheet(%s, %s, %s, %s)',
                (data['creator_id'], data['grade_level'],
                    data['category'], data['content']))
        conn.commit()
    return ''
```

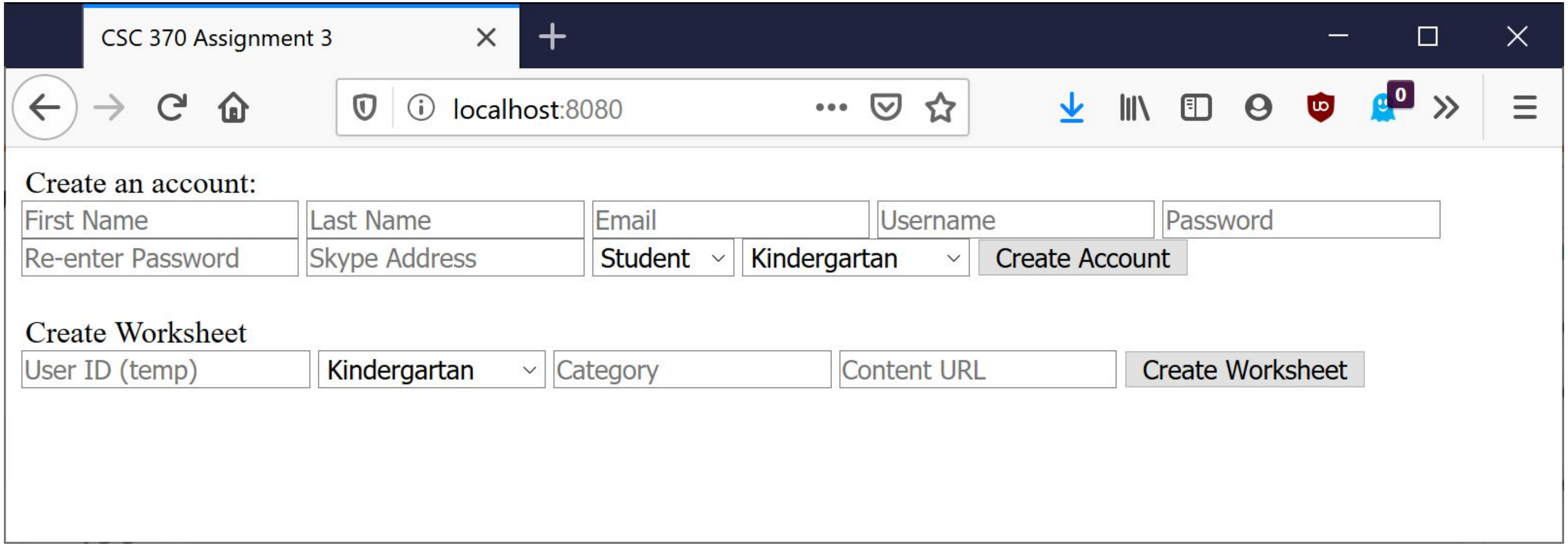
JavaScript client

- Mostly boilerplate to send POST requests to the python server



```javascript
let getJSON = function (url, params, callback) {
    let xhr = new XMLHttpRequest();
    xhr.open('POST', url, true);
    xhr.setRequestHeader("Content-type", "application/json; charset=utf-8");

    xhr.responseType = 'json';
    xhr.onload = function () {
        const status = xhr.status;
        if (status === 200) {
            callback(null, xhr.response);
        } else {
            callback(status, xhr.response);
        }
    };
    xhr.send(params);
};

function create_user() {
    // TODO: Check stuff (passwords match, strong enough, whatever
    // else, I don't know)
    const first_name = document.getElementById('first_name').value;
    const last_name = document.getElementById('last_name').value;
    const email = document.getElementById('email').value;
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;
    const password_confirm = document.getElementById('confirm').value;
    const skype = document.getElementById('skype').value;
    const user_type = document.getElementById('user_type').value;
    const grade_level = document.getElementById('grade_level').value;
```

Data insertion interface


CSC 370 Assignment 3

localhost:8080

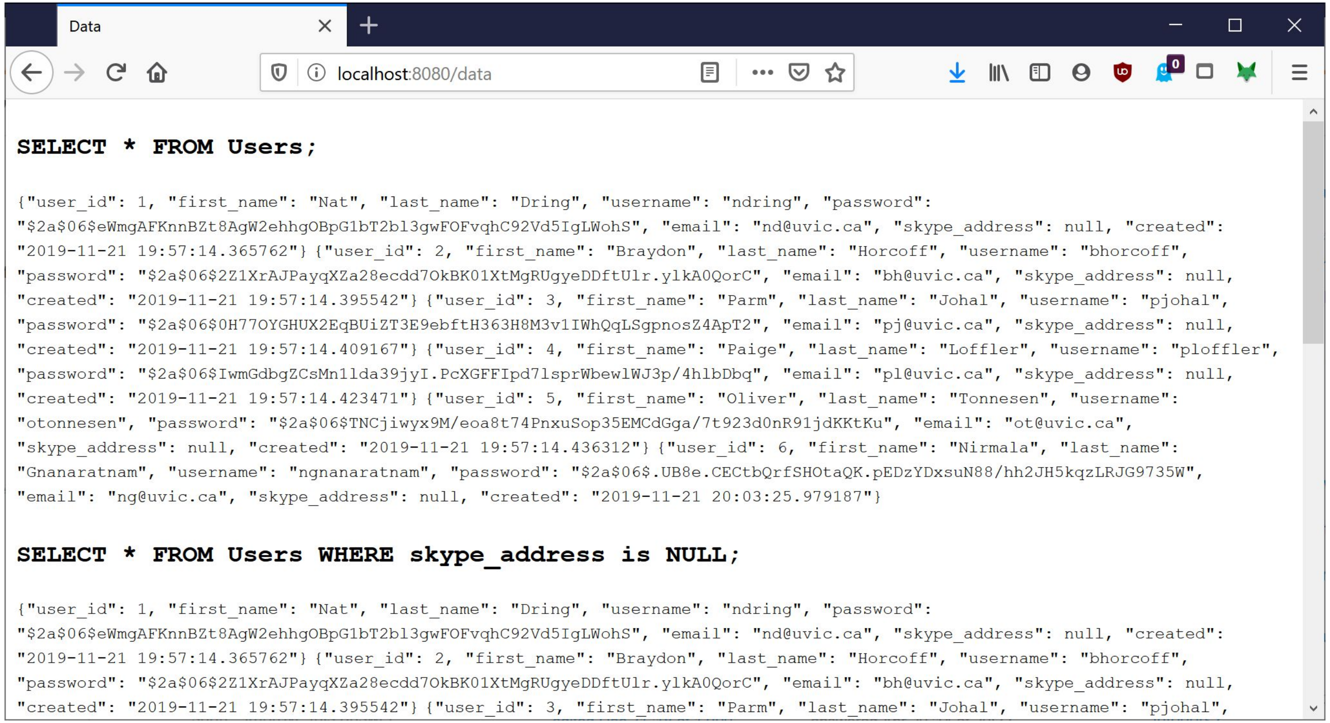
Create an account:

First Name	Last Name	Email	Username	Password
Re-enter Password	Skype Address	Student	Kindergartan	Create Account

Create Worksheet

User ID (temp)	Kindergartan	Category	Content URL	Create Worksheet

Sample queries viewable from web client


Data

localhost:8080/data

```
SELECT * FROM Users;
```

```
{"user_id": 1, "first_name": "Nat", "last_name": "Dring", "username": "ndring", "password":
"$2a$06$eWmgAFKnnBZt8AgW2ehhgOBpG1bT2bl3gwFOFvqhC92Vd5IgLWohS", "email": "nd@uvic.ca", "skype_address": null, "created":
"2019-11-21 19:57:14.365762"} {"user_id": 2, "first_name": "Braydon", "last_name": "Horcoff", "username": "bhorcoff",
"password": "$2a$06$2Z1XrAJPayqXZa28ecdd7OkBK01XtMgRUgyeDDftUlr.ylkA0QorC", "email": "bh@uvic.ca", "skype_address": null,
"created": "2019-11-21 19:57:14.395542"} {"user_id": 3, "first_name": "Parm", "last_name": "Johal", "username": "pjohal",
"password": "$2a$06$0H77OYGHUX2EqBUiZT3E9ebftH363H8M3v1IWhQqLSgpnosZ4ApT2", "email": "pj@uvic.ca", "skype_address": null,
"created": "2019-11-21 19:57:14.409167"} {"user_id": 4, "first_name": "Paige", "last_name": "Loffler", "username": "ploffler",
"password": "$2a$06$IwmGdbgZCsMn1lda39jyI.PcXGFFIpd7lsprWbewlWJ3p/4hlbDbq", "email": "pl@uvic.ca", "skype_address": null,
"created": "2019-11-21 19:57:14.423471"} {"user_id": 5, "first_name": "Oliver", "last_name": "Tonnesen", "username":
"otonnesen", "password": "$2a$06$TNCjiwyx9M/eoa8t74PnxuSop35EMCdGga/7t923d0nR91jdKKtKu", "email": "ot@uvic.ca",
"skype_address": null, "created": "2019-11-21 19:57:14.436312"} {"user_id": 6, "first_name": "Nirmala", "last_name":
"Gnanaratnam", "username": "ngnanaratnam", "password": "$2a$06$.UB8e.CECtbQrfSHOtaQK.pEDzYDxsuN88/hh2JH5kqzLRJG9735W",
"email": "ng@uvic.ca", "skype_address": null, "created": "2019-11-21 20:03:25.979187"}
```

```
SELECT * FROM Users WHERE skype_address is NULL;
```

```
{"user_id": 1, "first_name": "Nat", "last_name": "Dring", "username": "ndring", "password":
"$2a$06$eWmgAFKnnBZt8AgW2ehhgOBpG1bT2bl3gwFOFvqhC92Vd5IgLWohS", "email": "nd@uvic.ca", "skype_address": null, "created":
"2019-11-21 19:57:14.365762"} {"user_id": 2, "first_name": "Braydon", "last_name": "Horcoff", "username": "bhorcoff",
"password": "$2a$06$2Z1XrAJPayqXZa28ecdd7OkBK01XtMgRUgyeDDftUlr.ylkA0QorC", "email": "bh@uvic.ca", "skype_address": null,
"created": "2019-11-21 19:57:14.395542"} {"user_id": 3, "first_name": "Parm", "last_name": "Johal", "username": "pjohal",
```

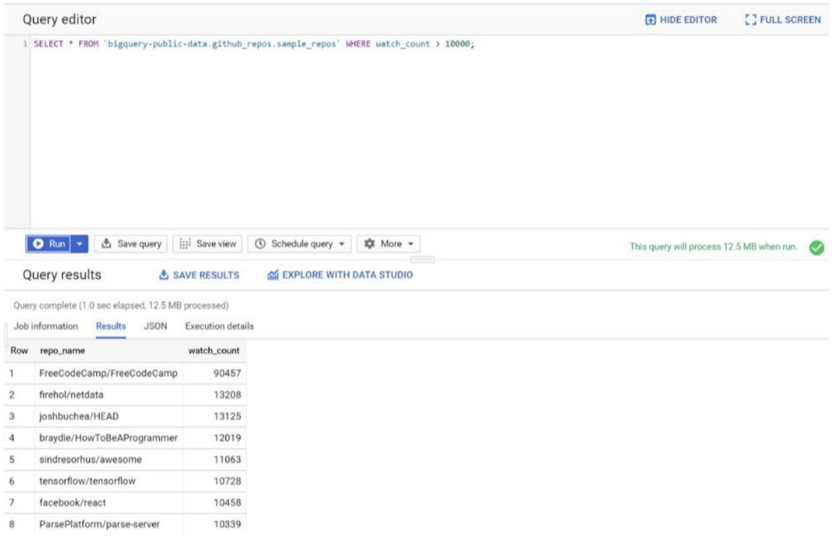
Assignment 4

- Google BigQuery
- Queried GitHub repository data
- Example questions asked:
    - Rates of language use
    - Most common licenses
    - Most watched projects

Using BigQuery to Query GitHub data

1. **What are all the repos with watch counts larger than ten thousand?**

SELECT * FROM `bigquery-public-data.github_repos.sample_repos` WHERE watch_count > 10000;



```
SELECT * FROM `bigquery-public-data.github_repos.sample_repos` WHERE watch_count > 10000;
```

This query will process 12.5 MB when run.

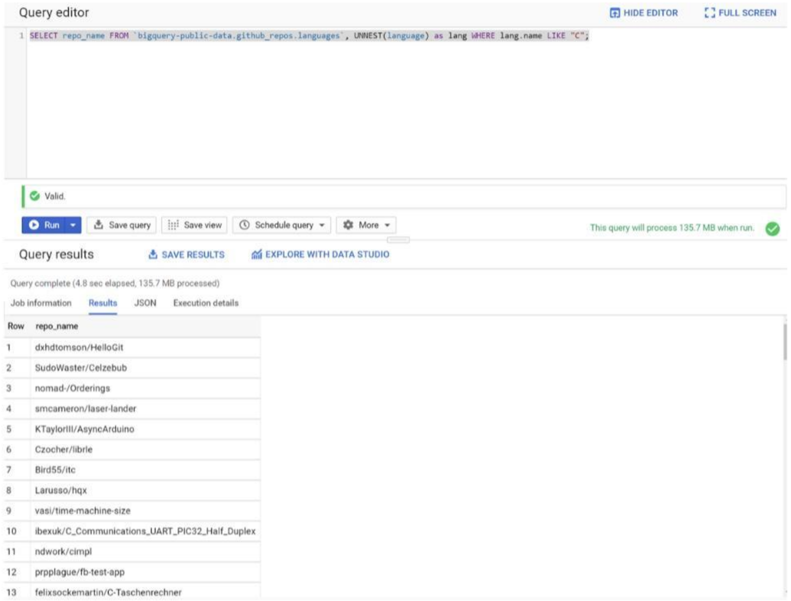
Query complete (1.0 sec elapsed, 12.5 MB processed)

Row	repo_name	watch_count
1	FreeCodeCamp/FreeCodeCamp	90457
2	firehol/netdata	13208
3	joshbuchea/HEAD	13125
4	braydie/HowToBeAProgrammer	12019
5	sindresorhus/awesome	11063
6	tensorflow/tensorflow	10728
7	facebook/react	10458
8	ParsePlatform/parse-server	10339

Using BigQuery to Query GitHub data

2. What are the repo names that have projects written in C?

SELECT repo_name FROM `bigquery-public-data.github_repos.languages`, UNNEST(language) as lang WHERE lang.name LIKE "C";



```
SELECT repo_name FROM `bigquery-public-data.github_repos.languages`, UNNEST(language) as lang WHERE lang.name LIKE "C";
```

Valid.

This query will process 135.7 MB when run.

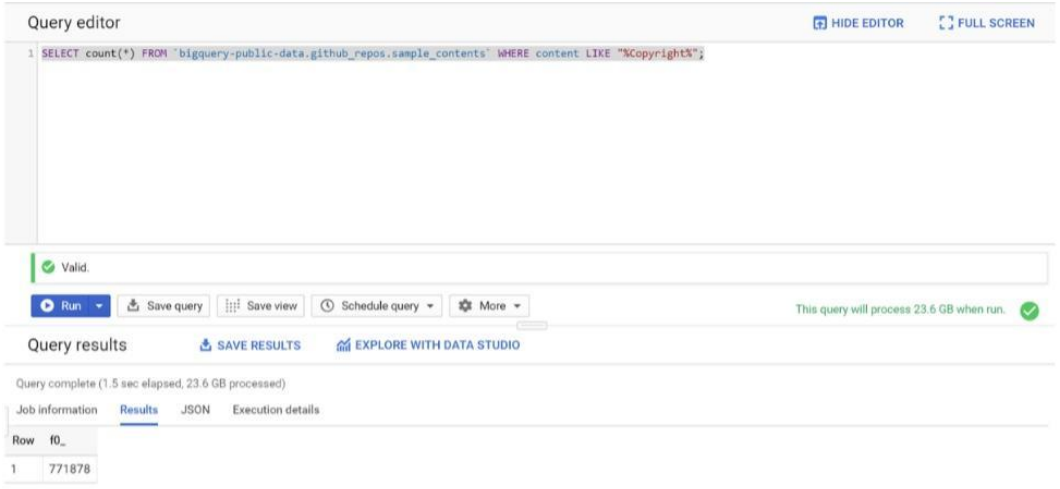
Query complete (4.8 sec elapsed, 135.7 MB processed)

Row	repo_name
1	dxhdtomson/HelloGit
2	SudoWaster/Celzebub
3	nomad-/Orderings
4	smcameron/laser-lander
5	KTaylorIII/AsyncArduino
6	Czocher/librle
7	Bird55/itc
8	Larusso/hqx
9	vasi/time-machine-size
10	ibexuk/C_Communications_UART_PIC32_Half_Duplex
11	ndwork/cimpl
12	prpplague/fb-test-app
13	felixsockemartin/C-Taschenrechner

Using BigQuery to Query GitHub data

3. How many users have a project that is contains a copyright?

SELECT count(*) FROM `bigquery-public-data.github_repos.sample_contents`
WHERE content LIKE "%Copyright%";



```
SELECT count(*) FROM `bigquery-public-data.github_repos.sample_contents` WHERE content LIKE "%Copyright%";
```

Valid.

This query will process 23.6 GB when run.

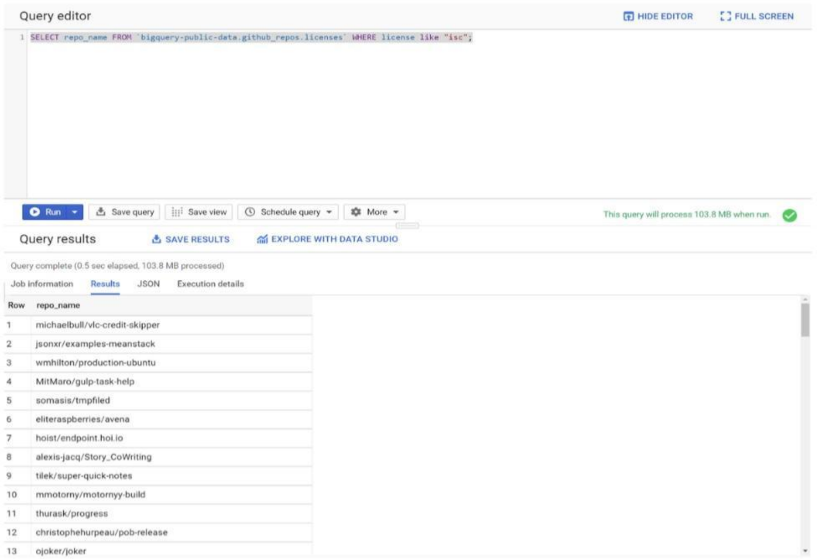
Query complete (1.5 sec elapsed, 23.6 GB processed)

Row	f0_
1	771878

Using BigQuery to Query GitHub data

4. What are the names of every repo with an isc license?

SELECT repo_name FROM `bigquery-public-data.github_repos.licenses` WHERE license like "isc";



```
SELECT repo_name FROM `bigquery-public-data.github_repos.licenses` WHERE license like "isc";
```

This query will process 103.8 MB when run.

Query complete (0.5 sec elapsed, 103.8 MB processed)

Row	repo_name
1	michaelbull/vlc-credit-skipper
2	jsonxr/examples-meanstack
3	wmhilton/production-ubuntu
4	MitMaro/gulp-task-help
5	somasis/tmpfiled
6	eliteraspberries/avena
7	hoist/endpoint.hoi.io
8	alexis-jacq/Story_CoWriting
9	tilek/super-quick-notes
10	mmotorny/motornyy-build
11	thurask/progress
12	christophehurpeau/pob-release
13	ojoker/joker