# jSearch

Our project allows users to track job applications for a given requirement cycle, as desired. There's a page for each table in our database that allows users to add, modify, and delete rows. In addition, we have a custom "insights" page that allows users to perform more interesting queries. These include...

- Determine all jobs with compensation above some minimum threshold, to demonstrate selection
- Select a give column from the applicant table (e.g., only applicant names), to demonstrate projection
- Determine a specialization from an applicant's email address, to demonstrate a join
- Determine the number of applicants who graduated in the earliest year, to demonstrate aggregation
- Determine the number of applicants per job position, to demonstrate nested aggregation
- Determine the job positions that *all* applicants applied to, to demonstrate division

It was written using React on the frontend and Java on the backend. The tables are responsive to user input and the display works on both desktop and mobile devices.

We hope this project can be used by UBC co-op students to facilitate an effective recruiting season, and can hopefully extend the number of "insightful" queries going forward!

## Schema

Here are the changes made to the final schema. Generally speaking, our codebase didn't differ from our proposed schema earlier, as many of the decisions made were straightforward to implement. We made two simplifications to ease implementation in the end, listed below.

- Removed enumeration type for `status_description`; instead, `status_description` was changed to be of type `varchar(100)`
- `year_hired` was changed from `Date` to `Integer`; using the date class was overly complex for the task at hand

## Queries

### Insertion

Insert a co-op applicant into the `applicant` table, given the `applicant_id`, `applicant_name`, `applicant_phone`, `applicant_email`, `spec_id`, `supervisor_id`, and `university_name`.

```
INSERT INTO applicant(applicant_id, applicant_name, applicant_phone, applicant_email, spec_i
VALUES (?, ?, ?, ?, ?, ?, ?)
```

### Deletion

Deletes a single `applicant` from the applicant table by `applicant_id`. Note the `applicant_id` is parameterized.

```
DELETE FROM applicant
WHERE applicant_id = ?;
```

### Update

Updates an applicant by supplying its attributes. Can update one or more fields, except for `applicant_id`.

```
UPDATE applicant
SET applicant_id = ?, applicant_name = ?, applicant_phone = ?, applicant_email = ?, spec_id
WHERE applicant_id = ?"
```

### Selection

Gets `company_id`, `position_id`, and salaries of jobs with salaries over a minimum threshold.

```
SELECT company_id, position_title, salary
FROM job_position_compensation
WHERE salary > ?
```

### Projection

Gets a requested field of an applicant, such as `applicant_id`, `applicant_email`, etc.

### Join

Gets specializations details, including major, minor, is_honors, and the degree type of the applicant whose email address matches the input email address.

```
SELECT major, minor, is_honours, degree_type
FROM specialization_info, applicant
WHERE specialization_info.spec_id = applicant.spec_id AND applicant.applicant_email = ?
```

### Aggregation

Gets the number of distinct applicants who has attended or is attended a university, as well as gets the minimum graduation year out of every applicant

```
SELECT COUNT(distinct applicant_id), MIN(graduation_year)
FROM attends
```

### Nested Aggregation

Gets number of applicants for each job position.

```
SELECT position_id, COUNT(applicant_id)
FROM application_made
GROUP BY position_id
```

### Division

Get the jobs to which all the applicants in the applicant table have applied.

```
SELECT position_title
FROM job_position_belongs_to J
WHERE NOT EXISTS ((SELECT A.applicant_id
                   FROM Applicant A)
                EXCEPT
                  (SELECT AM.applicant_id
                  FROM Application_Made AM
                  WHERE AM.position_id = J.position_id))
```