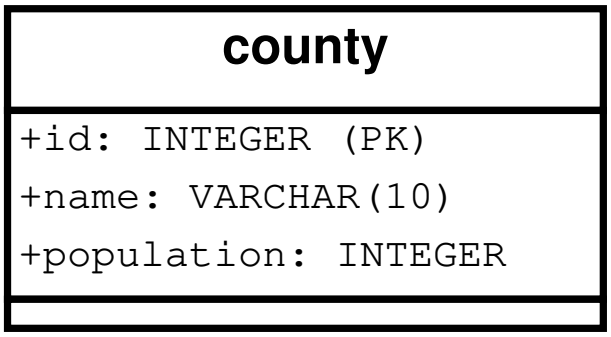
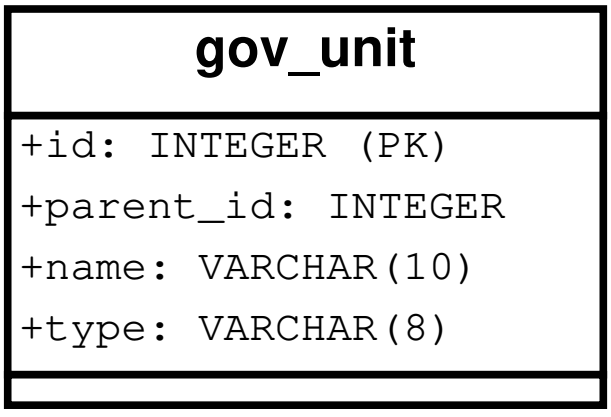
Graded Exercise 2

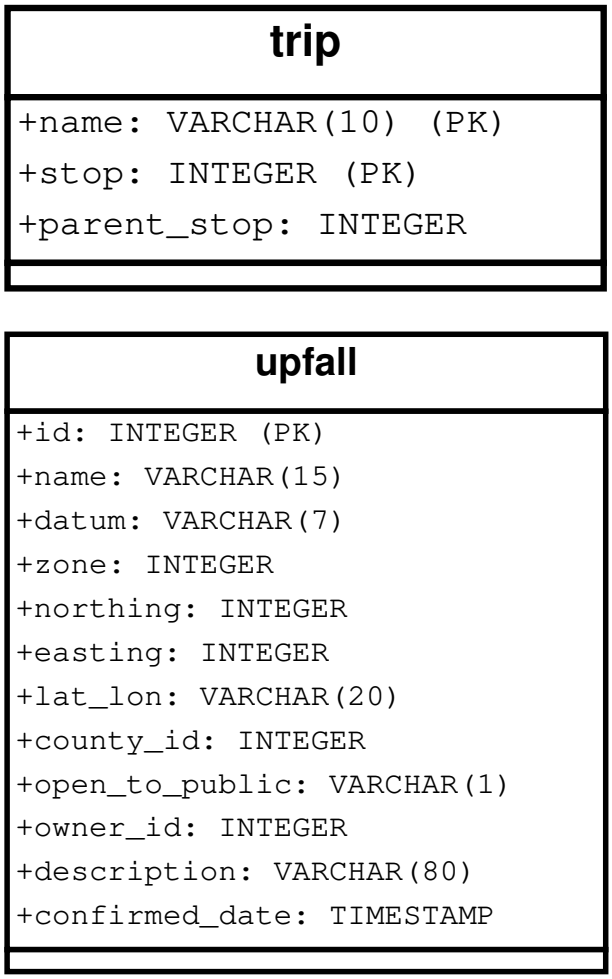
CSC230 Database Technologies for Analytics

08 November 2019

1 The waterfalls database.







2 Questions

1. Before we begin writing queries on our database of waterfalls, we might want to take a look at the structure and contents of our tables.

   Write a query that returns the whole upfall table (all rows and all columns).

```
SELECT * FROM upfall;
```

2. Here is a query that produces in each row the name of a waterfall and the name of the county in which that waterfall is located. It orders the results by county in ascending alphabetical order.

   Edit the query to make it label the columns in the result with the words 'Name of waterfall' and 'Name of county.'

```
/*
 * name of waterfall, name of county,
 * ordered by county
*/
SELECT f.name,
       c.name
  FROM upfall f INNER JOIN county c ON f.county_id = c.id
  ORDER BY c.name, f.name;
```

```
/*
 * name of waterfall, name of county,
 * ordered by county
*/
SELECT f.name AS 'Name of waterfall',
       c.name AS 'Name of county'
  FROM upfall f INNER JOIN county c ON f.county_id = c.id
  ORDER BY c.name, f.name;
```

3. Here is the start of a query that produces in each row the name of a county and the number of waterfalls in that county. It orders the results by county in ascending alphabetical order.

   Add what is needed to make it possible for **COUNT**(*) to tally for each county the number of rows that contain the name of that county.

```
/*
 * name of county, number of waterfalls in county,
 * ordered by county
*/
SELECT c.name AS 'Name of county',
       COUNT(*) AS 'Number of waterfalls in county'
  FROM upfall f INNER JOIN county c ON f.county_id= c.id

  ORDER BY c.name;
```

```
/*
 * name of county, number of waterfalls in county,
 * ordered by county
*/
SELECT c.name AS 'Name of county',
       COUNT(*) AS 'Number of waterfalls in county'
  FROM upfall f INNER JOIN county c ON f.county_id= c.id
  GROUP BY c.name
  ORDER BY c.name;
```

4. Here is the start of a query that produces the names of counties that have only one waterfall.

   This query searches for matching data in a table that another query produces. That inner **SELECT** statement must join two tables.

   Complete the query by providing the code that constructs the join (in the space that follows **FROM** in the inner **SELECT**).

```
/* names of counties that have only one waterfall */
SELECT name AS 'Name of county' FROM
   (SELECT c.name AS name, COUNT(*) AS numberOfFalls
       FROM
       GROUP BY c.name) AS t
  WHERE numberOfFalls = 1
  ORDER BY name;
```

```
/* names of counties that have only one waterfall */
SELECT name AS 'Name of county' FROM
   (SELECT c.name AS name, COUNT(*) AS numberOfFalls
       FROM upfall f INNER JOIN county c ON f.county_id = c.id
       GROUP BY c.name) AS t
  WHERE numberOfFalls = 1
  ORDER BY name;
```

5. Suppose that you have a table T that contain in each row the name of a county and the number of waterfalls in that county. The columns are 'name' and 'numberOfFalls.'

   Compose a query that produces the names of counties that have more than one waterfall.

```
SELECT name AS 'Name of county' FROM
    T
  WHERE numberOfFalls > 1
  ORDER BY name;
```

6. Here is the start of a query that produces in each row the name of a city or township and the name of the county in which that city/township is located.

   Modify the query to make it order the results first by county and then by city/township in ascending alphabetic order.

```
/*
 * name of city or township, name of county ordered by county,
 * city/township
*/
SELECT a.name AS 'City/Township', b.name AS 'County'
  FROM gov_unit a INNER JOIN gov_unit b ON a.parent_id = b.id
  WHERE a.type IN ('City', 'Township') AND b.type = 'County';
```

```
/*
 * name of city or township, name of county ordered by county,
 * city/township
*/
SELECT a.name AS 'City/Township', b.name AS 'County'
  FROM gov_unit a INNER JOIN gov_unit b ON a.parent_id = b.id
  WHERE a.type IN ('City', 'Township') AND b.type = 'County'
  ORDER BY b.name, a.name;
```

7. Here is the start of a query that produces in each row the name of a city or township, the name of the county in which that city/township is located, and the name of the state in which that county is located. It orders the results by state, county, and city in ascending alphabetic order.

   Supply the code that is needed after instances of the reserved word **ON**.

```
/*
 * name of city or township, name of county, name of state
 * ordered by state, county, city
*/
SELECT a.name AS 'City/Township',
       b.name AS 'County',
       c.name AS 'State'
  FROM gov_unit a INNER JOIN gov_unit b
    ON            INNER JOIN gov_unit c
    ON
  WHERE a.type IN ('City', 'Township') AND b.type = 'County'
  ORDER BY c.name, b.name, a.name;
```

```
/*
 * name of city or township, name of county, name of state
 * ordered by state, county, city
*/
SELECT a.name AS 'City/Township',
       b.name AS 'County',
       c.name AS 'State'
  FROM gov_unit a INNER JOIN gov_unit b
    ON a.parent_id = b.id INNER JOIN gov_unit c
    ON b.parent_id = c.id
  WHERE a.type IN ('City', 'Township') AND b.type = 'County'
  ORDER BY c.name, b.name, a.name;
```

8. Here is the start of a query that produces in each row the name of a waterfall together with the length of that waterfall's description.

   Modify the query to make it order the results by the length of the description in descending order, and then by the name in ascending alphabetic order.

```
/*
 * name of waterfall, length of waterfall's description
```

```
 * ordered by length of description in descending order, name
*/
SELECT name, LENGTH(description) AS 'Length of description'
  FROM upfall;
```

```
/*
 * name of waterfall, length of waterfall's description
 * ordered by length of description in descending order, name
*/
SELECT name, LENGTH(description) AS 'Length of description'
  FROM upfall ORDER BY LENGTH(description) DESC, name;
```

9. Here is the start of a query that produces in each row the name of a trip and the name of a waterfall that we will visit on that trip.

   Add what is needed between **FROM** and **ORDER BY**.

```
/*
 * name of trip, name of waterfall
*/
SELECT t.name, f.name FROM

  ORDER BY t.name, f.name;
```

```
/*
 * name of trip, name of waterfall
*/
SELECT t.name, f.name FROM
   upfall f INNER JOIN trip t ON t.stop = f.id
  ORDER BY t.name, f.name;
```

10. Here is the start of a query that produces in each row the name of a trip together with the name of the first waterfall that we will visit on that on trip.

    It is missing a **WHERE** clause. Write a **WHERE** clause that will include in the results table only those records that contain a **NULL** in the parent_stop column.

```
/*
 * name of trip, name of first waterfall on trip
*/
SELECT t.name, f.name FROM
  upfall f INNER JOIN trip t ON t.stop = f.id;
```

```
/*
 * name of trip, name of first waterfall on trip
*/
SELECT t.name, f.name FROM
  upfall f INNER JOIN trip t ON t.stop = f.id
 WHERE parent_stop IS NULL;
```

11. Here is a query that finds the minimum easting value in the upfall table.

    ```
    SELECT MIN(easting) FROM upfall;
    ```

    Here is a query that finds the maximum easting value in the upfall table.

    ```
    SELECT MAX(easting) FROM upfall;
    ```

    Both of these queries returns a table with a single row and a single column.

    Compose a query that returns the difference between the maximum and minimum values of easting. This result will also be just one number—a table with single row and a single column. It is a measure of the distance between the western-most waterfall and the eastern-most waterfall.

```
SELECT MAX(easting) - MIN(easting) FROM upfall;
```

12. Compose a query that returns the name of the waterfall whose easting is smaller than the easting of another easting, together with the value of that easting.

    This query will produce a table with a two columns and a single row. The first column will contain the name of the waterfall. The second column will contain the value of the waterfall's easting.

    Compose your query by placing these words in the right order.

- FROM
- FROM
- MIN(easting)
- SELECT
- WHERE
- easting
- easting
- name,
- upfall
- upfall);
- =
- (SELECT

```
SELECT name, easting
    FROM upfall
    WHERE easting = (SELECT min(easting) FROM upfall);
```