```
/*
Homework 4 Script
Jason Conci
jconci@zagmail.gonzaga.edu
2 October 2018
CPSC-321

Description -
1. Drops tables specified in spec sheet, if they exist
2. Creates all tables in the spec sheet
3. Populates tables (examples pertaining to USA, CAN, and MEX)
4. Creates variables used in queries
5. Runs specified queries in spec sheet
*/
/** DROP TABLE SECTION **/
DROP TABLE IF EXISTS border;

DROP TABLE IF EXISTS city;

DROP TABLE IF EXISTS province;

DROP TABLE IF EXISTS country;

/** CREATE TABLE SECTION **/
/*
Creates our country table. Primary key (country_code) ensures no two countries have the same
country code. VARCHAR's used for the sake of simplicity. GDP capped at 1 trillion monetary
units, with 2 sigfigs. Inflation capped at 999.99% (assumedly, we won't exceed this).
*/
CREATE TABLE country
  (
     country_code VARCHAR(100) NOT NULL,
     country_name VARCHAR(100) NOT NULL,
     gdp          DECIMAL(14, 2) NOT NULL,
     inflation    DECIMAL(5, 2) NOT NULL,
     PRIMARY KEY (country_code)
  )
engine = innodb;

/*
Creates our province table. Primary key (province_name, country_code) ensures no two provinces
within thes same country have the same name. Foreign key constraint (country_code) -> country
(country code) ensures we don't add a province to the table, without its containing country
existing in country first. VARCHAR's used for consistency and simplicity. Area capped at 11
digits (roughly the area of the earth).
*/
CREATE TABLE province
  (
     province_name VARCHAR(100) NOT NULL,
     country_code  VARCHAR(100) NOT NULL,
     area          DECIMAL(11, 2) NOT NULL,
     PRIMARY KEY (province_name, country_code),
     FOREIGN KEY (country_code) REFERENCES country (country_code)
  )
engine = innodb;

/*
Creates our city table. Primary key (province_name, country_code, city_name) ensures no two
cities within the same province (follows, within the same country) have the same name. Foreign
key constraint (province_name, country_code) -> province(province_name, country_code) ensures
we don't add a city to this table, without its containing province existing in province first.
VARCHAR's used for consistency and simplicity. Population capped at 10 digits (roughly the pop
of the earth), with no sigfigs since fractions of people don't make sense.
*/
CREATE TABLE city
  (
     city_name     VARCHAR(100) NOT NULL,
     province_name VARCHAR(100) NOT NULL,
     country_code  VARCHAR(100) NOT NULL,
     population    DECIMAL(10),
     PRIMARY KEY (city_name, province_name, country_code),
```

```
    FOREIGN KEY (province_name, country_code) REFERENCES province (
    province_name, country_code)
  )
engine = innodb;

/*
Creates our border table. Primary key (country1_code, country2_code) ensures that no country border
is described more than once (ALTER TABLE constraint below this CREATE TABLE statement also ensures
symmetrical uniqueness across country codes). Foreign key constraints (country1_code) -> country
(country_code), and (country2_code) -> country(country_code) ensures we don't describe a border
between two countries, without those countries existing in table country first. VARCHAR's used
for consistency and simplicity. Border length capped at 11 digits (roughly the area of the earth),
fractions allowed to 2 decimal places.
*/
CREATE TABLE border
  (
    country1_code VARCHAR(100) NOT NULL,
    country2_code VARCHAR(100) NOT NULL,
    border_length DECIMAL(11, 2) NOT NULL,
    PRIMARY KEY (country1_code, country2_code),
    FOREIGN KEY (country1_code) REFERENCES country (country_code),
    FOREIGN KEY (country2_code) REFERENCES country (country_code)
  )
engine = innodb;

-- this line ensures that the primary keys are unique symmetrically
ALTER TABLE border
  ADD CONSTRAINT unique_pairing UNIQUE (country1_code, country2_code);

/** SECTION FOR POPULATING TABLES WITH EXAMPLE DATA **/
INSERT INTO country
VALUES      ("usa",
             "united states of america",
             57466.79,
             3.60);

INSERT INTO country
VALUES      ("can",
             "canada",
             42157.93,
             4.50);

INSERT INTO country
VALUES      ("mex",
             "mexico",
             8201.31,
             2.70);

INSERT INTO province
VALUES      ("ontario",
             "can",
             456789);

INSERT INTO province
VALUES      ("quebec",
             "can",
             43289);

INSERT INTO province
VALUES      ("oaxaca",
             "mex",
             34278);

INSERT INTO province
VALUES      ("california",
             "usa",
             321789);

INSERT INTO city
VALUES      ("san francisco",
             "california",
```

```
             "usa",
             800000);

INSERT INTO city
VALUES      ("los angeles",
             "california",
             "usa",
             6000000);

INSERT INTO city
VALUES      ("montreal",
             "quebec",
             "can",
             1700000);

INSERT INTO city
VALUES      ("sacramento",
             "california",
             "usa",
             1500000);

INSERT INTO border
VALUES      ("usa",
             "can",
             5525.02);

INSERT INTO border
VALUES      ("usa",
             "mex",
             1954.00);

/** SECTION FOR CREATING VARIABLES USED IN QUERIES **/
SET @gdp = 40000;

SET @inflation = 4;

SET @can_code = "CAN";

SET @usa_code = "USA";

SET @mex_code = "MEX";

SET @sf_code = "San Francisco";

/** SECTION CONTAINING QUERIES SPECIFIED IN HW SPECS **/
-- Select the country codes where gdp is less than @gdp, and inflation is less than @inflation
SELECT c.country_code
FROM   country c
WHERE  c.gdp > @gdp
       AND c.inflation < @inflation;

/*
First, joins province and city relations on (province_name, country_code), meaning
that a city, in a given row, exists within the province in the row. Then, selecting
all rows where the city's population is greater than 1000. Then, selecting the names
and areas of all (distinct) provinces containing 1 or more cities where pop > 1000.
*/
SELECT DISTINCT p.province_name,
                p.area
FROM   province p
       JOIN city c USING (province_name, country_code)
WHERE  c.population > 1000;

/*
Self explanatory - returns the sum of the areas of all provinces in the relation.
*/
SELECT Sum(area)
FROM   province;

/*
Selecting all provinces in province p, where p.country_code = CAN, and then
```

```
averaging the area of all provinces selected.
*/
SELECT Sum(p.area)
FROM   province p
WHERE  p.country_code = @can_code;

/*
Self explanatory - gathering the min, max, and avg of both gdp & inflation, over
all countries within our country relation
*/
SELECT Min(gdp),
       Max(gdp),
       Avg(gdp),
       Min(inflation),
       Max(inflation),
       Avg(inflation)
FROM   country;

/*
We select all rows in city where c.country_code = @usa_code, then count the
number of rows selected. This gathers the count of all cities in our database
within the USA.
*/
SELECT Count(c.city_name)
FROM   city c
WHERE  c.country_code = @usa_code;

/*
We select all rows where the USA is one of the two countries describing a border,
and count the number of occurences of these borders, as well as the average border
length.
*/
SELECT Count(b.country1_code),
       Avg(b.border_length)
FROM   border b
WHERE  b.country1_code = @usa_code
        OR b.country2_code = @usa_code;

/*
We join the tables city and province on the attributes province_name
and country_code, meaning all cities in a row exist within the listed province.
Then, we gather cities in province "California", that are NOT san francisco,
and average their populations.
*/
SELECT Avg(c.population)
FROM   city c
       JOIN province p USING (province_name, country_code)
WHERE  c.city_name <> @sf_code
       AND p.province_name = "california";

/*
This query is fairly convoluted. Essentially, we're making all possible country
pairings where c1 is NOT the usa, and c2 is. Then, via inner join, we're selecting
all of these country pairings that describe a border between the US and another country.
From this, we select those countries c1 (other) with a higher inflation, and lower gdp,
than c2 (USA).
*/
SELECT c1.country_code
FROM   country c1
       INNER JOIN country c2
               ON ( c1.country_code <> @usa_code
                    AND c2.country_code = @usa_code )
       INNER JOIN border b
               ON ( ( b.country1_code = @usa_code
                       OR b.country2_code = @usa_code )
                    AND ( b.country1_code = c1.country_code
                           OR b.country2_code = c1.country_code ) )
WHERE  c2.inflation < c1.inflation
       AND c2.gdp > c1.gdp;
```