```
/* ************************************************************************
                    DATABASE ADMINISTRATION FUNDAMENTALS:
                  INTRODUCTION TO STRUCTURED QUERY LANGUAGE
                    SF21SQL1001, 2021/11/02 - 2021/12/09
                  https://folvera.commons.gc.cuny.edu/?cat=29
 ************************************************************************

SESSION #1 (2021/11/02): COOKIES NOW, CODE LATER

 1. For this example of cookies, we first need to create the data.  At this
    point of the course, you should not be concerned about the code, but rather
    what the language can do.

    1.1. On the right side, there is a quick explanation what each line (part
         of a single statement) does.

    1.2. Note that a statement starts with an English verb and ends with a
         semicolon.
 ************************************************************************ */

CREATE DATABASE food_stores;                    -- 1. creating database of food
                                                --    stores (`food_stores`)
CREATE SCHEMA nyc;                              -- 2. creating schema for food
                                                --    stores in NYC (`nyc`)
CREATE TABLE nyc.snack_store (                  -- 3. creating table for stores
  record_id INT NOT NULL,                       --    where we are shopping
  store_id INT,                                 --    (`snack_store`) in schema
  snack_id INT,                                 --    `nyc`; in other words,
  food_type VARCHAR(50),                        --    another food store in NYC
  flavor VARCHAR(50),
  organic VARCHAR(1),
  dietary_restriction VARCHAR(50),
  package_size VARCHAR(50),
  brand_name VARCHAR(50)
  );

INSERT INTO nyc.snack_store                     -- 4. inserting values in table
VALUES (                                        --    `nyc.snack_store`
  1,
  1,
  1,
  'cookie',
  'chocolate',
  'y',
  'gluten free',
  'small',
  'Good Cookies'
  ),
  (
  2,
  1,
  1,
```

```
    'cookie',
    'chocolate',
    'y',
    '',
    'small',
    'Good Cookies'
    ),
    (
    3,
    1,
    1,
    'cookie',
    'chocolate',
    'y',
    'nut free',
    'small',
    'Good Cookies'
    ),
    (
    4,
    1,
    2,
    'donut',
    'chocolate',
    'y',
    'gluten free',
    'small',
    'Good Cookies'
    ),
    (
    5,
    2,
    1,
    'cookie',
    'chocolate',
    'y',
    '',
    'small',
    'Good Cookies'
    ),
    (
    6,
    2,
    1,
    'cookie',
    'vanilla',
    'y',
    'gluten free',
    'large',
    'Cool Snacks'
    ),
    (
```

```
105     7,
106     2,
107     1,
108     'cookie',
109     'vanilla',
110     'y',
111     'nut free',
112     'single serving',
113     'Cool Snacks'
114     ),
115     (
116     8,
117     1,
118     1,
119     'cookie',
120     'chocolate',
121     'n',
122     'gluten free',
123     'small',
124     'Cookies by Alice'
125     ),
126     (
127     9,
128     3,
129     1,
130     'cookie',
131     'chocolate',
132     'n',
133     'gluten free',
134     'small',
135     'Cookies by Alice'
136     ),
137     (
138     10,
139     3,
140     1,
141     'cookie',
142     'chocolate',
143     'y',
144     'gluten free',
145     'small',
146     'Good Cookies'
147     ),
148     (
149     11,
150     3,
151     1,
152     'cookie',
153     'vanilla',
154     'y',
155     'gluten free',
156     'small',
```

```
157     'Cool Snacks'
158     ),
159     (
160     12,
161     1,
162     2,
163     'donut',
164     'jelly',
165     'y',
166     'gluten free',
167     'small',
168     'Good Cookies'
169     ),
170     (
171     13,
172     1,
173     1,
174     'cookie',
175     'chocolate',
176     'n',
177     'gluten free',
178     'small',
179     'Cookies by Alice'
180     ),
181     (
182     14,
183     3,
184     1,
185     'cookie',
186     'chocolate',
187     'n',
188     'gluten free',
189     'small',
190     'Cookies by Alice'
191     ),
192     (
193     15,
194     3,
195     1,
196     'cookie',
197     'chocolate',
198     'n',
199     'gluten free',
200     'small',
201     'Cookies by Alice'
202     ),
203     (
204     16,
205     3,
206     1,
207     'cookie',
208     'chocolate',
```

```
    'y',
    '',
    'large',
    'Grand Cookies'
    ),
    (
    17,
    3,
    1,
    'cookie',
    'vanilla',
    'y',
    'vegan',
    'large',
    'Good Cookies'
    ),
    (
    18,
    3,
    1,
    'cookie',
    'vanilla',
    'y',
    'vegan',
    'large',
    'Good Cookies'
    ),
    (
    19,
    3,
    1,
    'cookie',
    'chocolate',
    'y',
    'gluten free',
    'small',
    'Grand Cookies'
    );

CREATE TABLE nyc.store_names (
  store_id INT,
  store_name VARCHAR(50)
  );

INSERT INTO nyc.store_names
VALUES (
  1,
  'The Nature Shop'
  ),
  (
  2,
  'Supermarket Cool Banana'
```

```
261    ),
262    (
263    3,
264    'The Corner Shop'
265    );
266
267  SELECT *
268  FROM nyc.store_names;
269
270
271  /* ***************************************************************************
272      1.3. In this example, you are an assistant to a boss who wants cookies from
273           `The Corner Shop`.
274
275           We will keep adding line by line to redefine what we need.  As you can
276           see, the English request is translated to SQL in an English-like
277           series of expressions.
278
279           Note that, if you read the statement out loud and it does not sound
280           right as an English imperative sentence (a command, in this case given
281           to a computer), most likely your syntax is wrong.
282   *************************************************************************** */
283
284  SELECT *                                        -- 1. getting all information
285  FROM nyc.snack_store                            -- 2. from specific population
286                                                  --    `nyc.snack_store`
287  WHERE food_type = 'cookie'                      -- 3. where specific
288                                                  --    `food_type` is `cookie`
289    AND store_id = 3;                             --    `store_id` is `3`, the
290                                                  --    identifier for `The
291                                                  --    Corner Shop`
292
293
294  /* ***************************************************************************
295      1.4. You go to `The Corner Shop` and bring multiple cookies including
296           doubles.  Your boss then says that you should not bring doubles.
297   *************************************************************************** */
298
299  SELECT DISTINCT *                               -- `distinct` = no doubles,
300                                                  -- unique cookies
301  FROM nyc.snack_store
302  WHERE food_type = 'cookie'
303    AND store_id = 3;
304
305
306  /* ***************************************************************************
307      1.5. You go back to the store and bring back one of each cookie.  Then your
308           boss says that he/she wants chocolate cookies.
309   *************************************************************************** */
310
311  SELECT DISTINCT *                               -- 1. still looking for all
312                                                  --    information
```

```
313 FROM nyc.snack_store                          -- 2. from specific population
314                                               --    `nyc.snack_store`
315 WHERE food_type = 'cookie'                    -- 3. still looking for a
316                                               --    `food_type` of `cookie`
317   AND store_id = 3                            --    and `store_id` 3
318   AND flavor = 'chocolate';                   -- 4. new column/value
319                                               --    `flavor` is `chocolate`
320                                               -- 5. hence looking for a
321                                               --    chocolate cookies
322
323
324 /* ************************************************************************
325     1.6. You go back to the store and bring back only chocolate cookies.  Then
326          your boss says that the cookies must be organic.
327  ************************************************************************ */
328
329 SELECT DISTINCT *
330 FROM nyc.snack_store
331 WHERE food_type = 'cookie'
332   AND store_id = 3
333   AND flavor = 'chocolate'
334   AND organic = 'y';
335
336
337 /* ************************************************************************
338     1.7. You go back to the store and bring back only organic chocolate
339          cookies.  Then your boss says that the cookies should also be
340          gluten-free.
341  ************************************************************************ */
342
343 SELECT DISTINCT *
344 FROM nyc.snack_store
345 WHERE food_type = 'cookie'
346   AND store_id = 3
347   AND flavor = 'chocolate'
348   AND organic = 'y'
349   AND dietary_restriction = 'gluten free';
350
351
352 /* ************************************************************************
353     1.8. You go back to the store and bring back only gluten-free and organic
354          chocolate cookies.  Then your boss says the package is small.
355  ************************************************************************ */
356
357 SELECT DISTINCT *
358 FROM nyc.snack_store
359 WHERE food_type = 'cookie'
360   AND store_id = 3
361   AND flavor = 'chocolate'
362   AND organic = 'y'
363   AND dietary_restriction = 'gluten free'
364   AND package_size = 'small';
```

```
365
366
367 /* ********************************************************************************
368     1.9. You go back to the store and bring back only small packages of
369          gluten-free and organic chocolate cookies.  Then your boss says that
370          the brand name should be `Good Cookies`.
371  ******************************************************************************** */
372
373 SELECT DISTINCT *
374 FROM nyc.snack_store
375 WHERE food_type = 'cookie'
376   AND store_id = 3
377   AND flavor = 'chocolate'
378   AND organic = 'y'
379   AND dietary_restriction = 'gluten free'
380   AND package_size = 'small'
381   AND brand_name = 'Good Cookies';
382
383
384 /* ********************************************************************************
385     1.10. You go back to the store and bring back only small packages of `Good
386           Cookies` gluten-free and organic chocolate cookies.  At this point,
387           your boss is finally satisfied.
388
389           Since the values of `cookies` and `donuts` are repeated in several
390           records (rows), these values could be moved to another table and
391           referred to with a unique identifier (a row ID).  The values of
392           `chocolate` and `vanilla` can also be moved to another table.
393
394           This is the concept of relational databases where tables have unique
395           values that can be referred by unique identifiers -- for example,
396           `food_type_id` for `cookies` and `donuts` and `flavor_id` for
397           `chocolate and `vanilla`.
398
399  2. Now that you had a brief introduction to SQL, we can take a look at the
400     most basic statement.  In SQL in order to retrieve data, we use a `SELECT`
401     statement where the simplest syntax is the following.
402
403                     SELECT field1, field2 ...
404                     FROM table1;
405
406     2.1. In the example below, we retrieve all columns (fields) and all rows
407          (records) from `AP1.ContactUpdates` calling each one of the columns.
408  ******************************************************************************** */
409
410 SELECT VendorID,
411   VendorName,
412   VendorAddress1,
413   VendorAddress2,
414   VendorCity,
415   VendorState,
416   VendorZipCode,
```

```
417    VendorPhone,
418    VendorContactLName,
419    VendorContactFName,
420    DefaultTermsID,
421    DefaultAccountNo
422  FROM AP1.ContactUpdates;
423
424
425  /* ***************************************************************************
426      2.2. In the example below, we retrieve all columns (fields) and all rows
427           (records) from `AP1.Vendors` using wild card `*` (read as `all`).
428   *************************************************************************** */
429
430  SELECT *                                        -- read as `all`
431  FROM AP1.Vendors;
432
433
434  /* ***************************************************************************
435      2.3. In the example below, we retrieve all columns and rows from tables
436           `AP1.ContactUpdates` and `AP1.Vendors` using wild card `*` (read as
437           `all`).
438
439           Since we are calling a second table, we have to `JOIN` them on the
440           common field (data, value) as this indicates the relation between the
441           two tables.  We are going to cover three `JOIN` alternatives.  Each
442           `JOIN` returns a different population.
443
444           `INNER JOIN` returns shared data (rows) between the two tables.
445
446           `LEFT [OUTTER] JOIN` returns all the data (rows) from the left table
447           (first table called, `AP1.ContactUpdates`) and any shared data (rows)
448           in the right table (second table called, `AP1.Vendors`).
449
450           `RIGHT [OUTTER] JOIN` returns all the data (rows) from the right table
451           (second table you call, `AP1.Vendors`) and any shared data (rows) in
452           the left table (first table called, `AP1.ContactUpdates`).
453
454           In the example below, we retrieve all shared data (rows) from tables
455           `AP1.ContactUpdates` and `AP1.Vendors`.
456   *************************************************************************** */
457
458  SELECT *
459  FROM AP1.ContactUpdates                         -- 1. all shared data (rows)
460                                                  --    from `AP1.ContactUpdates`
461  INNER JOIN AP1.Vendors                          -- 2. all shared data (rows)
462                                                  --    from `AP1.Vendors`
463    ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID;
464                                                  -- 3. on common data (rows)
465                                                  --    `VendorID`
466
467
468  /* ***************************************************************************
```

```
469      2.4. In the example below, we retrieve all data (rows) from table
470           `AP1.ContactUpdates` and any shared data (rows) from `AP1.Vendors`.
471   ********************************************************************** */
472
473  SELECT *
474  FROM AP1.ContactUpdates                          -- 1. all data (rows) from main
475                                                   --    table
476                                                   --    `AP1.ContactUpdates`
477  LEFT JOIN AP1.Vendors                            -- 2. any shared data (rows)
478                                                   --    from `AP1.Vendors`
479    ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID;
480                                                   -- 3. on common data (rows)
481                                                   --    `VendorID`
482
483
484  /* ********************************************************************
485      2.5. In the example below, we retrieve all data (rows) from table
486           `AP1.Vendors` and any shared data (rows) from `AP1.ContactUpdates`.
487
488      2.6. We will cover `LEFT JOIN` and `RIGHT JOIN` in more detail later in the
489           course.
490   ********************************************************************** */
491
492  SELECT *
493  FROM AP1.ContactUpdates                          -- 1. any shared data (rows)
494                                                   --    from `AP1.ContactUpdates`
495  RIGHT JOIN AP1.Vendors                           -- 2. all data (rows) from main
496                                                   --    table `AP1.Vendors`
497    ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID;
498                                                   -- 3. on common data (rows)
499                                                   --    `VendorID`
500
501  /* ********************************************************************
502   https://folvera.commons.gc.cuny.edu/?p=985
503   ********************************************************************** */
```