L1 E1 - Step 5

June 10, 2021

1 STEP 5: ETL the data from 3NF tables to Facts & Dimension Tables

IMPORTANT: The following exercise depends on first having successing completed Exercise 1: Step 4.

Start by running the code in the cell below to connect to the database. If you are coming back to this exercise, then uncomment and run the first cell to recreate the database. If you recently completed steps 1 through 4, then skip to the second cell.

```
In [10]: # !PGPASSWORD=student createdb -h 127.0.0.1 -U student pagila
         # !PGPASSWORD=student psql -q -h 127.0.0.1 -U student -d pagila -f Data/pagila-schema.s
         # !PGPASSWORD=student psql -q -h 127.0.0.1 -U student -d pagila -f Data/pagila-data.sql
```

```
In [11]: %load_ext sql

         DB_ENDPOINT = "127.0.0.1"
         DB = 'pagila'
         DB_USER = 'student'
         DB_PASSWORD = 'student'
         DB_PORT = '5432'

         # postgresql://username:password@host:port/database
         conn_string = "postgresql://{}:{}@{}:{}/{}" \
                                 .format(DB_USER, DB_PASSWORD, DB_ENDPOINT, DB_PORT, DB)

         print(conn_string)
         %sql $conn_string
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
postgresql://student:student@127.0.0.1:5432/pagila
```

```
Out[11]: 'Connected: student@pagila'
```

1.0.1 Introducing SQL to SQL ETL

When writing SQL to SQL ETL, you first create a table then use the INSERT and SELECT statements together to populate the table. Here's a simple example.

First, you create a table called test_table.

```
In [12]: %%sql
         CREATE TABLE test_table
         (
           date timestamp,
           revenue  decimal(5,2)
         );

 * postgresql://student:***@127.0.0.1:5432/pagila
Done.


Out[12]: []
```

Then you use the INSERT and SELECT statements to populate the table. In this case, the SELECT statement extracts data from the `payment` table and INSERTs it INTO the `test_table`.

```
In [13]: %%sql
         INSERT INTO test_table (date, revenue)
         SELECT payment_date AS date,
                amount AS revenue
         FROM payment;

 * postgresql://student:***@127.0.0.1:5432/pagila
32098 rows affected.


Out[13]: []
```

Then you can use a SELECT statement to take a look at your new table.

```
In [14]: %sql SELECT * FROM test_table LIMIT 5;

 * postgresql://student:***@127.0.0.1:5432/pagila
5 rows affected.


Out[14]: [(datetime.datetime(2017, 1, 24, 21, 40, 19, 996577), Decimal('1.99')),
          (datetime.datetime(2017, 1, 25, 15, 16, 50, 996577), Decimal('0.99')),
          (datetime.datetime(2017, 1, 28, 21, 44, 14, 996577), Decimal('6.99')),
          (datetime.datetime(2017, 1, 29, 0, 58, 2, 996577), Decimal('0.99')),
          (datetime.datetime(2017, 1, 29, 8, 10, 6, 996577), Decimal('4.99'))]
```

If you need to delete the table and start over, use the DROP TABLE command, like below.

```
In [15]: %sql DROP TABLE test_table

 * postgresql://student:***@127.0.0.1:5432/pagila
Done.


Out[15]: []
```

Great! Now you'll do the same thing below to create the dimension and fact tables for the Star Schema using the data in the 3NF database.

1.1 ETL from 3NF to Star Schema

1.1.1 3NF - Entity Relationship Diagram

1.1.2 Star Schema - Entity Relationship Diagram

In this section, you'll populate the tables in the Star schema. You'll `extract` data from the normalized database, `transform` it, and `load` it into the new tables.

To serve as an example, below is the query that populates the `dimDate` table with data from the `payment` table. * NOTE 1: The EXTRACT function extracts date parts from the payment_date variable. * NOTE 2: If you get an error that says that the `dimDate` table doesn't exist, then go back to Exercise 1: Step 4 and recreate the tables.

```
In [16]: %%sql
         INSERT INTO dimDate (date_key, date, year, quarter, month, day, week, is_weekend)
         SELECT DISTINCT(TO_CHAR(payment_date :: DATE, 'yyyyMMDD')::integer) AS date_key,
                date(payment_date)                                           AS date,
                EXTRACT(year FROM payment_date)                              AS year,
                EXTRACT(quarter FROM payment_date)                           AS quarter,
                EXTRACT(month FROM payment_date)                             AS month,
                EXTRACT(day FROM payment_date)                               AS day,
                EXTRACT(week FROM payment_date)                              AS week,
                CASE WHEN EXTRACT(ISODOW FROM payment_date) IN (6, 7) THEN true ELSE false END A
         FROM payment;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
40 rows affected.
```

```
Out[16]: []
```

TODO: Now it's your turn. Populate the `dimCustomer` table with data from the `customer`, `address`, `city`, and `country` tables. Use the starter code as a guide.

```
In [17]: %%sql
         INSERT INTO dimCustomer (customer_key, customer_id, first_name, last_name, email, addre
                                  address2, district, city, country, postal_code, phone, active,
                                  create_date, start_date, end_date)
         SELECT
             c.customer_id as customer_key,
             c.customer_id,
             c.first_name,
             c.last_name,
             c.email,
             a.address,
             a.address2,
             a.district,
             ci.city,
             co.country,
             postal_code,
             a.phone,
             c.active,
             c.create_date,
             now()         AS start_date,
             now()         AS end_date
         FROM customer c
         JOIN address a  ON (c.address_id = a.address_id)
         JOIN city ci    ON (a.city_id = ci.city_id)
         JOIN country co ON (ci.country_id = co.country_id);
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
599 rows affected.
```

```
Out[17]: []
```

TODO: Populate the dimMovie table with data from the film and language tables. Use the starter code as a guide.

```
In [18]: %%sql
         INSERT INTO dimMovie (movie_key, film_id, title, description, release_year, language, o
                               length, rating, special_features)
         SELECT
             f.film_id as movie_key,
             f.film_id,
             f.title,
             f.description,
             f.release_year,
             l.name as language,
             orig_lang.name as original_language,
             f.rental_duration,
             f.length,
             f.rating,
             f.special_features
         FROM film f
         JOIN language l              ON (f.language_id=l.language_id)
         LEFT JOIN language orig_lang ON (f.original_language_id = orig_lang.language_id);
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
1000 rows affected.
```

```
Out[18]: []
```

TODO: Populate the dimStore table with data from the store, staff, address, city, and country tables. This time, there's no guide. You should write the query from scratch. Use the previous queries as a reference.

```
In [20]: %%sql
         INSERT INTO dimStore (store_key, store_id, address, address2, district, city, country,
                               manager_first_name, manager_last_name, start_date, end_date)
         SELECT
             s.store_id as store_key,
             s.store_id,
             a.address,
             a.address2,
             a.district,
             c.city,
             co.country,
             a.postal_code,
             st.first_name as manager_first_name,
             st.last_name as manager_last_name,
             now() as start_date,
             now() as end_date
         FROM
             store s
         JOIN
             staff st ON (s.manager_staff_id = st.staff_id)
         JOIN
             address a ON (s.address_id = a.address_id)
         JOIN
             city c ON (a.city_id = c.city_id)
         JOIN
             country co ON (c.country_id = co.country_id);
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
2 rows affected.
```

```
Out[20]: []
```

TODO: Populate the `factSales` table with data from the `payment`, `rental`, and `inventory` tables. This time, there's no guide. You should write the query from scratch. Use the previous queries as a reference.

```
In [21]: %%sql
         INSERT INTO factSales (date_key, customer_key, movie_key, store_key, sales_amount)
         SELECT
             TO_CHAR(p.payment_date::DATE, 'yyyyMMDD')::integer as date_key,
             p.customer_id as customer_key,
             i.film_id as movie_key,
             i.store_id as store_key,
             p.amount
         FROM
             payment p
         JOIN
             rental r ON (p.rental_id = r.rental_id)
         JOIN
             inventory i ON (r.inventory_id = i.inventory_id)
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
32098 rows affected.
```

```
Out[21]: []
```