

Demo 01 - Sakila Star Schema & ETL

All the database tables in this demo are based on public database samples and transformations

- Sakila is a sample database created by MySQL [Link \(https://dev.mysql.com/doc/sakila/en/sakila-structure.html\)](https://dev.mysql.com/doc/sakila/en/sakila-structure.html)
- The postgresql version of it is called Pagila [Link \(https://github.com/devrimgunduz/pagila\)](https://github.com/devrimgunduz/pagila)
- The facts and dimension tables design is based on O'Reilly's public dimensional modelling tutorial schema [Link \(http://archive.oreilly.com/oreillyschool/courses/dba3/index.html\)](http://archive.oreilly.com/oreillyschool/courses/dba3/index.html)

STEP0: Using ipython-sql

- load ipython-sql: `%load_ext sql`
- To execute SQL queries you write one of the following atop of your cell:
 - `%sql`
 - For a one-liner SQL query
 - You can access a python var using `$`
 - `%%sql`
 - For a multi-line SQL query
 - You can **NOT** access a python var using `$`
- Running a connection string like: `postgresql://postgres:postgres@db:5432/pagila` connects to the database

STEP1 : Connect to the local database where Pagila is loaded

1.1 Create the pagila db and fill it with data

- Adding `!"` at the beginning of a jupyter cell runs a command in a shell, i.e. we are not running python code but we are running the `createdb` and `psql` postgresql command-line utilities

In []:

```
!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.sql
!PGPASSWORD=student psql -q -h 127.0.0.1 -U student -d pagila -f Data/pagila-data.sql
```

1.2 Connect to the newly created db

In []:

```
%load_ext sql
```

In []:

```
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://{user}:{password}@{host}:{port}/{db}" \
               .format(DB_USER, DB_PASSWORD, DB_ENDPOINT, DB_PORT, DB)

print(conn_string)
```

In []:

```
%sql $conn_string
```

STEP 1.1: Create and populate the star schema

STEP2 : Explore the 3NF Schema



2.1 How much? What data sizes are we looking at?

In []:

```
nStores = %sql select count(*) from store;
nFilms = %sql select count(*) from film;
nCustomers = %sql select count(*) from customer;
nRentals = %sql select count(*) from rental;
nPayment = %sql select count(*) from payment;
nStaff = %sql select count(*) from staff;
nCity = %sql select count(*) from city;
nCountry = %sql select count(*) from country;

print("nFilms\t\t=", nFilms[0][0])
print("nCustomers\t=", nCustomers[0][0])
print("nRentals\t=", nRentals[0][0])
print("nPayment\t=", nPayment[0][0])
print("nStaff\t\t=", nStaff[0][0])
print("nStores\t\t=", nStores[0][0])
print("nCITIES\t\t=", nCity[0][0])
print("nCountry\t\t=", nCountry[0][0])
```

2.2 When? What time period are we talking about?

In []:

```
%%sql
select min(payment_date) as start, max(payment_date) as end from payment;
```

2.3 Where? Where do events in this database occur?

In []:

```
%%sql
select district, sum(city_id) as n
from address
group by district
order by n desc
limit 10;
```

STEP3: Perform some simple data analysis

3.1 Insight 1: Top Grossing Movies

- Payments amounts are in table payment
- Movies are in table film
- They are not directly linked, payment refers to a rental, rental refers to an inventory item and inventory item refers to a film
- payment → rental → inventory → film

3.1.1 Films

In []:

```
%%sql
select film_id, title, release_year, rental_rate, rating from film limit 5;
```

3.1.2 Payments

In []:

```
%%sql
select * from payment limit 5;
```

3.1.3 Inventory

In []:

```
%%sql
select * from inventory limit 5;
```

3.1.4 Get the movie of every payment

In []:

```
%%sql
SELECT f.title, p.amount, p.payment_date, p.customer_id
FROM payment p
JOIN rental r    ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id )
JOIN film f ON ( i.film_id = f.film_id)
limit 5;
```

3.1.5 sum movie rental revenue

In []:

```
%%sql
SELECT f.title, sum(p.amount) as revenue
FROM payment p
JOIN rental r    ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id )
JOIN film f ON ( i.film_id = f.film_id)
GROUP BY title
ORDER BY revenue desc
limit 10;
```

3.2 Insight 2: Top grossing cities

- Payments amounts are in table `payment`
- Cities are in table `cities`
- `payment` → `customer` → `address` → `city`

3.2.1 Get the city of each payment

In []:

```
%%sql
SELECT p.customer_id, p.rental_id, p.amount, ci.city
FROM payment p
JOIN customer c ON ( p.customer_id = c.customer_id )
JOIN address a ON ( c.address_id = a.address_id )
JOIN city ci ON ( a.city_id = ci.city_id )
order by p.payment_date
limit 10;
```

3.2.2 Top grossing cities

In []:

```
%%sql
SELECT ci.city , sum(p.amount) as revenue
FROM payment p
JOIN customer c ON ( p.customer_id = c.customer_id )
JOIN address a ON ( c.address_id = a.address_id )
JOIN city ci ON ( a.city_id = ci.city_id )
group by ci.city
order by revenue desc
limit 10;
```

3.3 Insight 3 : Revenue of a movie by customer city and by month

3.3.1 Total revenue by month

In []:

```
%%sql
SELECT sum(p.amount) as revenue, EXTRACT(month FROM p.payment_date) as month
from payment p
group by month
order by revenue desc
limit 10;
```

3.3.2 Each movie by customer city and by month (data cube)

In []:

```
%%sql
SELECT f.title, p.amount, p.customer_id, ci.city, p.payment_date, EXTRACT(month FROM p.payment_date) as month
FROM payment p
JOIN rental r ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id )
JOIN film f ON ( i.film_id = f.film_id )
JOIN customer c ON ( p.customer_id = c.customer_id )
JOIN address a ON ( c.address_id = a.address_id )
JOIN city ci ON ( a.city_id = ci.city_id )
order by p.payment_date
limit 10;
```

3.3.3 Sum of revenue of each movie by customer city and by month

In []:

```
%%sql
SELECT f.title, ci.city, EXTRACT(month FROM p.payment_date) as month, sum(p.amount) as revenue
FROM payment p
JOIN rental r ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id )
JOIN film f ON ( i.film_id = f.film_id )
JOIN customer c ON ( p.customer_id = c.customer_id )
JOIN address a ON ( c.address_id = a.address_id )
JOIN city ci ON ( a.city_id = ci.city_id )
group by (f.title, ci.city, month)
order by month, revenue desc
limit 10;
```

STEP4 : Creating Facts & Dimensions



In []:

```
%%sql
CREATE TABLE dimDate
(
    date_key integer NOT NULL PRIMARY KEY,
    date date NOT NULL,
    year smallint NOT NULL,
    quarter smallint NOT NULL,
    month smallint NOT NULL,
    day smallint NOT NULL,
    week smallint NOT NULL,
    is_weekend boolean
);

CREATE TABLE dimCustomer
(
    customer_key SERIAL PRIMARY KEY,
    customer_id smallint NOT NULL,
    first_name varchar(45) NOT NULL,
    last_name varchar(45) NOT NULL,
    email varchar(50),
    address varchar(50) NOT NULL,
    address2 varchar(50),
    district varchar(20) NOT NULL,
    city varchar(50) NOT NULL,
    country varchar(50) NOT NULL,
    postal_code varchar(10),
    phone varchar(20) NOT NULL,
    active smallint NOT NULL,
    create_date timestamp NOT NULL,
    start_date date NOT NULL,
    end_date date NOT NULL
);

CREATE TABLE dimMovie
(
    movie_key SERIAL PRIMARY KEY,
    film_id smallint NOT NULL,
    title varchar(255) NOT NULL,
    description text,
    release_year year,
    language varchar(20) NOT NULL,
    original_language varchar(20),
    rental_duration smallint NOT NULL,
    length smallint NOT NULL,
    rating varchar(5) NOT NULL,
    special_features varchar(60) NOT NULL
);

CREATE TABLE dimStore
(
    store_key SERIAL PRIMARY KEY,
    store_id smallint NOT NULL,
    address varchar(50) NOT NULL,
    address2 varchar(50),
    district varchar(20) NOT NULL,
    city varchar(50) NOT NULL,
```

```
country          varchar(50) NOT NULL,  
postal_code      varchar(10),  
manager_first_name varchar(45) NOT NULL,  
manager_last_name varchar(45) NOT NULL,  
start_date       date NOT NULL,  
end_date         date NOT NULL  
);  
CREATE TABLE factSales  
(  
    sales_key      SERIAL PRIMARY KEY,  
    date_key       INT NOT NULL REFERENCES dimDate(date_key),  
    customer_key   INT NOT NULL REFERENCES dimCustomer(customer_key),  
    movie_key      INT NOT NULL REFERENCES dimMovie(movie_key),  
    store_key      INT NOT NULL REFERENCES dimStore(store_key),  
    sales_amount   decimal(5,2) NOT NULL  
);
```

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

In []:

```

%%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 AS is_weekend
FROM payment;

INSERT INTO dimCustomer (customer_key, customer_id, first_name, last_name, email, address,
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,
       a.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);

INSERT INTO dimMovie (movie_key, film_id, title, description, release_year, language, original_language, rental_duration, 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);

```

```
INSERT INTO dimStore (store_key, store_id, address, address2, district, city, country, postal_code, 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);

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                                          AS sales_amount
FROM payment p
JOIN rental r   ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id );
```

STEP 6: Repeat the computation from the facts & dimension table

6.1 Facts Table has all the needed dimensions, no need for deep joins

In [42]:

```
%%time
%%sql
SELECT movie_key, date_key, customer_key, sales_amount
FROM factSales
limit 5;
```

```
* postgresql://student:***@127.0.0.1:5432/pagila
5 rows affected.
CPU times: user 4.06 ms, sys: 527 µs, total: 4.59 ms
Wall time: 10 ms
```

Out[42]:

movie_key	date_key	customer_key	sales_amount
870	20170124	269	1.99
651	20170125	269	0.99
818	20170128	269	6.99
249	20170129	269	0.99
159	20170129	269	4.99

6.2 Join fact table with dimensions to replace keys with attributes

In []:

```
%%time
%%sql
SELECT dimMovie.title, dimDate.month, dimCustomer.city, sales_amount
FROM factSales
JOIN dimMovie on (dimMovie.movie_key = factSales.movie_key)
JOIN dimDate on (dimDate.date_key = factSales.date_key)
JOIN dimCustomer on (dimCustomer.customer_key = factSales.customer_key)
limit 5;
```

In []:

```
%%time
%%sql
SELECT dimMovie.title, dimDate.month, dimCustomer.city, sum(sales_amount) as revenue
FROM factSales
JOIN dimMovie on (dimMovie.movie_key = factSales.movie_key)
JOIN dimDate on (dimDate.date_key = factSales.date_key)
JOIN dimCustomer on (dimCustomer.customer_key = factSales.customer_key)
group by (dimMovie.title, dimDate.month, dimCustomer.city)
order by dimMovie.title, dimDate.month, dimCustomer.city, revenue desc;
```

In []:

```
%%time
%%sql
SELECT f.title, EXTRACT(month FROM p.payment_date) as month, ci.city, sum(p.amount) as revenue
FROM payment p
JOIN rental r ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id )
JOIN film f ON ( i.film_id = f.film_id)
JOIN customer c ON ( p.customer_id = c.customer_id )
JOIN address a ON ( c.address_id = a.address_id )
JOIN city ci ON ( a.city_id = ci.city_id )
group by (f.title, month, ci.city)
order by f.title, month, ci.city, revenue desc;
```

Conclusion

- We were able to show that a star schema is easier to understand
- Evidence that it is more performant

In [46]:

```
!PGPASSWORD=student pg_dump -h 127.0.0.1 -U student pagila > Data/pagila-star.sql
```