Mastering SQL using Postgresql

Durga Gadiraju

Dec 04, 2020

CONTENTS

This course is primarily designed to learn basic and advanced SQL using Postgresql Database.

CHAPTER ONE

ABOUT POSTGRESQL

Postgresql is one of the leading datatabase. It is an open source database and used for different types of applications.

- Web Applications
- Mobile Applications
- Data Logging Applications

Even though it is relational database and best suited for transactional systems (OLTP), it's flavors such as Redshift are extensively used for Analytical or Decision Support Systems.

CHAPTER TWO

COURSE DETAILS

This course is primarily designed to go through basic and advanced SQL using Postgres Database. You will be learning following aspects of SQL as well as Postgres Database.

- Setup Postgres Database using Docker
- Connect to Postgres using different interfaces such as psql, SQL Workbench, Jupyter with SQL magic etc.
- Understand utilities to load the data
- Overview of Normalization Principles and Relations (RDBMS Concepts)
- Performing CRUD or DML Operations
- Writing basic SQL Queries such as filtering, joins, aggregations, sorting etc
- Creating tables, constraints and indexes
- Different partitioning strategies while creating tables
- Using pre-defined functions provided by Postgresql
- Writing advanced SQL queries using analytic functions
- Overview of query performance tuning with emphasis on explain plans and different tuning techniques
- Difference between RDBMS and Data Warhousing with live examples.

CHAPTER

THREE

DESIRED AUDIENCE

Here are the desired audience for this course.

- College students and entry level professionals to get hands on expertise with respect to SQL to be prepared for the interviews.
- Experienced application developers to understand key aspects of Databases to improve their productivity.
- Data Engineers and Data Warehouse Developers to understand the relevance of SQL and other key concepts.
- Testers to improve their query writing abilities to validate data in the tables as part of running their test cases.
- Business Analysts to write ad-hoc queries to understand data better or troubleshoot data quality issues.
- Any other hands on IT Professional who want to improve their query writing and tuning capabilities.

Note: Developers from non CS or IT background at times struggle in writing queries and this course will provide required database skills to take their overall application development skills to next level.

CHAPTER FOUR

PREREQUISITES

Here are the prerequisites before signing up for the course.

Logistics

- Computer with decent configuration
  - At least 4 GB RAM
  - 8 GB RAM is highly desired
- Chrome Browser
- High Speed Internet

Desired Skills

- Engineering or Science Degree
- Ability to use computer
- Knowledge or working experience with databases is highly desired

CHAPTER

FIVE

KEY OBJECTIVES

The course is designed for the professionals to achieve these key objectives related to databases using Postgresql.

- Ability to interpret data models.
- Using database IDEs to interact with databases.
- Data loading strategies to load data into database tables.
- Write basic as well as advanced SQL queries.
- Ability to create tables, partition tables, indexes etc.
- Understand and use constraints effectively based up on the requirements.
- Effective usage of functions provided by Postgresql.
- Understand basic performance tuning strategies
- Differences between RDBMS and Data Warehouse concepts by comparing Postgresql with Redshift.

Attention: This course is primarily designed to gain key database skills for application developers, data engineers, testers, business analysts etc.

CHAPTER

SIX

TRAINING APPROACH

Here are the details related to the training approach.

- It is self paced with reference material, code snippets and videos.
- One can either use environment provided by us or setup their own environment using Docker.
- Modules will be published as and when they are ready. We would recommend to complete **2 modules every week** by spending **4 to 5 hours per week**.
- It is highly recommended to take care of the exercises at the end to ensure that you are able to meet all the key objectives for each module.
- Support will be provided either through chat or email.
- For those who signed up, we will have weekly monitoring and review sessions to keep track of the progress.

Attention: Spend 4 to 5 hours per week up to 8 weeks and complete all the exercises to get best out of this course.

6.1 Getting Started

As part of this section we will primarily understand different ways to get started with Postgres.

- Connecting to Database
- Using psql
- Setup Postgres using Docker
- Setup SQL Workbench
- SQL Workbench and Postgres
- SQL Workbench Features
- Data Loading Utilities
- Loading Data - Docker
- Exercise - Loading Data

Here are the key objectives of this section

- Connecting to Database using Jupyter based environment in our labs. This is relevant to only those who got our lab access.
- Ability to setup Postgres Database using Docker for those who does not have access to our labs.

- Relevance of IDEs such as SQL Workbench
- Understand the key features for IDEs such as SQL Workbench including connecting SQL Workbench to Postgres Database.
- How to load data into tables using Database native utilities?
- Exercise to ensure our understanding related to loading data into the tables using database native utilities.

6.1.1 Connecting to Database

We will be using JupyterHub based environment to master Postgresql. Let us go through the steps involved to get started using JupyterHub environment.

- We will use Python Kernel with sql magic command and for that we need to first load the sql extension.
- Create environment variable `DATABASE_URL` using SQL Alchemy format.
- Write a simple query to get data from information schema table to validate database connectivity.
- Here is the information you can leverage to connect to the database.
  - **User Name:** YOUR_OS_USER_sms_user
  - **Database Name:** YOUR_OS_USER_sms_db
  - **Password:** Your lab password provided by us

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%sql SELECT * FROM information_schema.tables LIMIT 10
```

6.1.2 Using psql

Let us understand how to use `psql` utility to perform database operations.

- `psql` is command line utility to connect to the Postgres database server. It is typically used for the following by advanced Database users:
  - Manage Databases
  - Manage Tables
  - Load data into tables for testing purposes
- We need to have at least Postgres Client installed on the server from which you want to use psql to connect to Postgres Server.
- If you are on the server where **Postgres Database Server** is installed, `psql` will be automatically available.
- We can run `sudo -u postgres psql -U postgres` from the server provided you have sudo permissions on the server. Otherwise we need to go with `psql -U postgres -W` which will prompt for the password.
- **postgres** is the super user for the postgres server and hence typically developers will not have access to it in non development environments.

- As a developer, we can use following command to connect to a database setup on postgres server using user credentials.

```
psql  -h <host_ip_or_dns_alias> -d <db_name> -U <user_name> -W

# Here is the example to connect to itversity_sms_db using itversity_sms_user
psql -h localhost -p 5432 -d itversity_sms_db -U itversity_sms_user -W
```

- We typically use `psql` to troubleshoot the issues in non development servers. IDEs such as **SQL Alchemy** might be better for regular usage as part of development and unit testing process.
- For this course, we will be primarily using Jupyter based environment for practice.
- However, we will go through some of the important commands to get comfortable with `psql`.
  - Listing Databases - `\l`
  - Switching to a Database - \c <DATABASE_NAME>
  - Get help for `psql` - `\?`
  - Listing tables - `\d`
  - Create table - `CREATE TABLE t (i SERIAL PRIMARY KEY)`
  - Get details related to a table - \d <table_name>
  - Running Scripts - \i <SCRIPT_PATH>
  - You will go through some of the commands over a period of time.

6.1.3 Setup Postgres using Docker

In some cases you might want to have postgres setup on your machine. Let us understand how we can setup Postgres using Docker.

- If you are using our labs, the database will be pre-created by us with all the right permissions.
- If you are using Windows or Mac, ensure that you have installed Docker Desktop.
- If you are using Ubuntu based desktop, make sure to setup Docker.
- Here are the steps that can be used to setup Postgres database using Docker.
  - Pull the postgres image using `docker pull`
  - Create the container using `docker create`.
  - Start the container using `docker start`.
  - Alternatively we can use `docker run` which will pull, create and start the container.
  - Use `docker logs` or `docker logs -f` to review the logs to ensure Postgres Server is up and running.

```
docker pull postgres

docker container create \
    --name itv_pg \
    -p 5433:5432 \
    -h itv_pg \
    -e POSTGRES_PASSWORD=itversity \
    postgres
```

(continues on next page)

(continued from previous page)

```
docker start itv_pg

docker logs itv_pg
```

- You can connect to Postgres Database setup using Docker with `docker exec`.

```
docker exec \
    -it itv_pg \
    psql -U postgres
```

- You can also connecto to Postgres directly with out using `docker exec`.

```
psql -h localhost \
    -p 5433 \
    -d postgres \
    -U postgres \
    -W
```

6.1.4 Setup SQL Workbench

Let us understand how to setup and use SQL Workbench.

Why SQL Workbench

Let us see the details why we might have to use SQL Workbench.

- Using Database CLIs such as psql for postgres, mysql etc can be cumbersome for those who are not comfortable with command line interfaces.
- Database IDEs such as SQL Workbench will provide required features to run queries against databases with out worrying to much about underlying data dictionaries.
- SQL Workbench provide required features to review databases and objects with out writing queries or running database specific commands.
- Also Database IDEs provide capabilities to preserve the scripts we develop.

  In short Database IDEs such as SQL Workbench improves productivity.

Alternative IDEs

There are several IDEs in the market.

- TOAD
- SQL Developer for Oracle
- MySQL Workbench and many others

Install SQL Workbench

Here are the instructions to setup SQL Workbench.

- Download SQL Workbench (typically zip file)
- Unzip and launch

Once installed we need to perform below steps which will be covered in detail as part of next topic.

- Download JDBC driver for the database we would like to connect.

- Get the database connectivity information and connect to the database.

6.1.5 SQL Workbench and Postgres

Let us connect to Postgres Database using SQL Workbench.

- Download the JDBC Driver
- Get the database connectivity information
- Configure the connection using SQL Workbench
- Validate the connection and save the profile

Connecting to Postgres

Here are the steps to connect to Postgres running on your PC or remote machine without Docker.

- We are trying to connect to Postgres Database that is running as part of remote machine or on your PC.
- We typically use ODBC or JDBC to connect to a Database from remote machines (our PC).
- Here are the pre-requisites to connect to a Database.
  - Make sure 5432 port is opened as part of the firewalls.
  - If you have telnet configured on your system on which SQL Workbench is installed, make sure to validate by running telnet command using ip or DNS Alias and port number 5432.
  - Ensure that you have downloaded right JDBC Driver for Postgres.
  - Make sure to have right credentials (username and password).
  - Ensure that you have database created on which the user have permissions.
- Once you have all the information required along with JDBC jar, ensure to save the information as part of the profile. You can also validate before saving the details by using **Test** option.

Postgres on Docker

Here are the steps to connect to Postgres running as part of Docker container.

- We are trying to connect to Postgres Database that is running as part of Docker container running in a Ubuntu 18.04 VM provisioned from GCP.
- We have published Postgres database port to port 5433 on Ubuntu 18.04 VM.
- We typically use ODBC or JDBC to connect to a Database from remote machines (our PC).
- Here are the pre-requisites to connect to a Database on GCP.
  - Make sure 5432 port is opened as part of the firewalls.
  - If you have telnet configured on your system on which SQL Workbench is installed, make sure to validate by running telnet command using ip or DNS Alias and port number 5433.
  - Ensure that you have downloaded right JDBC Driver for Postgres.
  - Make sure to have right credentials (username and password).
  - Ensure that you have database created on which the user have permissions.

- You can validate credentials and permissions to the database by installing postgres client on Ubuntu 18.04 VM and then by connecting to the database using the credentials.
- Once you have all the information required along with JDBC jar, ensure to save the information as part of the profile. You can also validate before saving the details by using **Test** option.

6.1.6 SQL Workbench Features

Here are some of the key features, you have to familiar with related to SQL Workbench.

- Ability to connect to different RDBMS, Data Warehouse and MPP Database servers such as Postgres, MySQL, Oracle, Redshift etc.
- Saving profiles to connect to multiple databases.
- Ability to access data dictionary or information schema using wizards to validate tables, columns, sequences, indexes, constraints etc.
- Generate scripts out of existing data.
- Ability to manage database objects with out writing any commands. We can drop tables, indexes, sequences etc by right clicking and then selecting drop option.
- Develop SQL files and preserve them for future usage.

Almost all leading IDEs provide all these features in similar fashion.

Usage Scenarios

Here are **some of the usage scenarios** for database IDEs such as SQL Workbench as part of day to day responsibilities.

- Developers for generating and validating data as part of unit testing.
- Testers to validate data for their test cases.
- Business Analysts and Data Analysts to run ad hoc queries to understand the data better.
- Developers to troubleshoot data related to production issues using read only accounts.

6.1.7 Data Loading Utilities

Let us understand how we can load the data into databases using utilities provided.

- Most of the databases provide data loading utilities.
- One of the most common way of getting data into database tables is by using data loading utilities provided by the underlying datatabase technology.
- We can load delimited files into database using these utilities.
- Here are the steps we can follow to load the delimited data into the table.
  - Make sure files are available on the server from which we are trying to load.
  - Ensure the database and table are created for the data to be loaded.
  - Run relevant command to load the data into the table.
  - Make sure to validate by running queries.
- Let us see a demo by loading a sample file into the table in Postgres database.

Loading Data

We can use COPY Command using `psql` to copy the data into the table.

- Make sure database is created along with the user with right permissions. Also the user who want to use `COPY` command need to have **pg_read_server_files** role assigned.
- Create the file with sample data. In this case data is added to **users.csv** under **/data/sms_db**

```
user_first_name,user_last_name,user_email_id,user_role,created_dt
Gordan,Bradock,gbradock0@barnesandnoble.com,A,2020-01-10
Tobe,Lyness,tlyness1@paginegialle.it,U,2020-02-10
Addie,Mesias,amesias2@twitpic.com,U,2020-03-05
Corene,Kohrsen,ckohrsen3@buzzfeed.com,U,2020-04-15
Darill,Halsall,dhalsall4@intel.com,U,2020-10-10
```

- Connect to Database.

```
psql -U itversity_sms_user \
  -h localhost \
  -p 5432 \
  -d itversity_sms_db \
  -W
```

- Create the `users` table.

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE
);
```

- Use copy command to load the data

```
COPY users(user_first_name, user_last_name,
    user_email_id, user_role, created_dt
) FROM '/data/sms_db/users.csv'
DELIMITER ','
CSV HEADER;
```

- Validate by running queries

```
SELECT * FROM users;
```

6.1.8 Loading Data - Docker

Let us understand how you can take care of loading data into Postgres Database running using Docker Container.

- Make sure database is created along with the user with right permissions. Also the user who want to use `COPY` command need to have **pg_read_server_files** role assigned.
  - Create file with sample data
  - Copy file into Docker container
  - Connect to Database
  - Create the table
  - Run `COPY` Command to copy the data.

Prepare Data

We need to create file with sample data and copy the files into the container.

- Sample File In this case data is added to users.csv under **~/sms_db**.

```
user_first_name,user_last_name,user_email_id,user_role,created_dt
Gordan,Bradock,gbradock0@barnesandnoble.com,A,2020-01-10
Tobe,Lyness,tlyness1@paginegialle.it,U,2020-02-10
Addie,Mesias,amesias2@twitpic.com,U,2020-03-05
Corene,Kohrsen,ckohrsen3@buzzfeed.com,U,2020-04-15
Darill,Halsall,dhalsall4@intel.com,U,2020-10-10
```

- Copy data

```
docker cp ~/sms_db/users.csv itv_pg:/tmp
```

Create Database

Here are the steps to create database.

- Connect to database as super user **postgres**

```
docker exec -it itv_pg psql -U postgres
```

- Create the database with right permissions.

```
CREATE DATABASE itversity_sms_db;
CREATE USER itversity_sms_user WITH PASSWORD 'sms_password';
GRANT ALL ON DATABASE itversity_sms_db TO itversity_sms_user;
GRANT pg_read_server_files TO itversity_sms_user;
```

- Exit using `\q`

Connect to Database

Use this command to connect to the newly created database.

```
psql -U itversity_sms_user \
  -h localhost \
  -p 5433 \
  -d itversity_sms_db \
  -W
```

Create Table

Here is the script to create the table.

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE
);
```

Load Data

Here are the steps to load and validate the data using `psql`.

- Load data using `COPY` Command

```
COPY users(user_first_name, user_last_name,
    user_email_id, user_role, created_dt
) FROM '/tmp/users.csv'
DELIMITER ','
CSV HEADER;
```

- Validate by running queries

```
SELECT * FROM users;
```

6.1.9 Exercise - Loading Data

As part of this exercise, you need to take care of loading data using `COPY` Command.

- You can connect to the database using following details in the environment provided by us.
  - Host: localhost
  - Port: 5342
  - Database Name: YOUR_OS_USER_hr_db
  - User Name: YOUR_OS_USER_hr_user
  - Password: YOUR_OS_USER_PASSWORD (provided by us).
- If you are using your own environment, make sure to create database for storing HR Data.
  - Database Name: hr_db
  - User Name: hr_user
  - You can create user with password of your choice.

```
CREATE DATABASE hr_db;
CREATE USER hr_user WITH PASSWORD 'hr_password';
GRANT ALL ON DATABASE hr_db TO hr_user;
GRANT pg_read_server_files TO hr_user;
```

- Create table using this script.

```
CREATE TABLE employees
   ( employee_id INTEGER
   , first_name VARCHAR(20)
   , last_name VARCHAR(25)
   , email VARCHAR(25)
   , phone_number VARCHAR(20)
   , hire_date DATE
   , job_id VARCHAR(10)
   , salary NUMERIC(8,2)
   , commission_pct NUMERIC(2,2)
   , manager_id INTEGER
   , department_id INTEGER
   ) ;
CREATE UNIQUE INDEX emp_emp_id_pk
         ON employees (employee_id) ;
ALTER TABLE employees ADD
   PRIMARY KEY (employee_id);
```

- Understand data.
  - Check for delimiters (record as well as field).
  - Check whether header exists or not.
  - Ensure number of fields for the table and data being loaded are same or not.
- Load data into the table using `COPY` Command. The file is under `/data/hr_db/employees`
- Validate by running these queries. You can also use SQL Workbench to run the queries to validate whether data is loaded successfully or not.

```
SELECT * FROM employees LIMIT 10;
SELECT count(1) FROM employees;
```

6.2 DML or CRUD Operations

Let us understand how to perform CRUD operations using Postgresql.

- Normalization Principles
- Tables as Relations
- Database Operations - Overview
- CRUD Operations
- Creating Table
- Inserting Data
- Updating Data
- Deleting Data
- Overview of Transactions
- Exercise - Database Operations

Here are the key objectives of this section.

- What are the different types of Database Operations?

- How DML is related to CRUD Operations?
- How to insert new records into table?
- How to update existing data in a table?
- How the data is typically deleted from a table?
- You will also get a brief overview about Database Operations?
- Self evaluate whether you gain enough skills related to performing CRUD or DML operations or not using exercieses

6.2.1 Normalization Principles

Let us get an overview about Normalization Principles.

Here are different normal forms we use. Provided links are from Wiki.

- 1st Normal Form
- 2nd Normal Form
- 3rd Normal Form
- Boyce Codd Normal Form

Most of the well designed Data Models will be in either 3rd Normal Form. BCNF is used in some extreme cases where 3rd Normal Form does not eliminate all insertion, updation and deletion anomalies.

Reporting Environments

While normalization is extensively used for transactional systems, they are not ideal for reporting or descision support systems. We tend to use dimensional modeling for reporting systems where tables will contain pre processed data as per the report requirements.

Normal Forms - Key Terms

Let us understand some of the key terms we use while going through the normal forms.

- Domain
- Attribute
- Atomic (indivisible)
- Functionally Dependent
- Prime Attribute
- Candidate Key
- Data Anomalies - potential issues to data due to the mistakes by users or developers
- Transitive Dependency

6.2.2 Tables as Relations

Let us understand details about relations and different types of relationships we typically use.

- In RDBMS - R stands for Relational.
- In the transactional systems, tables are created using normalization principles. There will be relations or tables created based on relationships among them.
- Here are the typical relationships among the tables.
  - 1 to 1
  - 1 to many or many to 1 (1 to n or n to 1)
  - many to many (m to n)
- To **enforce** relationships we typically define constraints such as **Primary Key** and **Foreign Key**.
- Here is the typical process we follow from requirements to physical database tables before building applications.
  - Identify entities based up on the requirements.
  - Define relationships among them.
  - Create ER Diagram (Entity Relationship Diagram). It is also called as Logical Data Model.
  - Apply Normalization Principles on the entities to identify tables and constraints to manage relationships among them.
  - Come up with Physical Data Model and generate required DDL Scripts.
  - Execute the scripts in the database on which applications will be eventually build based up on business requirements.
- Logical modeling is typically done by Data Architects.
- Physical modeling is taken care by Application Architect or Development lead.
- Let us go through data model related to HR and OE systems.
  - Identify the relationships between the tables.
  - Differentiate between transactional tables and non transactional tables.

6.2.3 Database Operations - Overview

Let us get an overview of Database Operations we typically perform on regular basis. They are broadly categorized into the following:

- DDL - Data Definition Language
  - CREATE/ALTER/DROP Tables
  - CREATE/ALTER/DROP Indexes
  - Add constraints to tables
  - CREATE/ALTER/DROP Views
  - CREATE/ALTER/DROP Sequences
- DML - Data Manipulation Language
  - Inserting new data into the table
  - Updating existing data in the table

  - Deleting existing data from the table
- DQL - Data Query Language
  - Read the data from the table

On top of these we also use TCL (Transaction Control Language) which include **COMMIT** and **ROLLBACK**.

As part of this section in the subsequent topics we will primarily focus on basic DDL and DML.

6.2.4 CRUD Operations

Let us get an overview of CRUD Operations. They are nothing but DML and queries to read the data while performing database operations.

- CRUD is widely used from application development perspective.
- C - CREATE (INSERT)
- R - READ (READ)
- U - UPDATE (UPDATE)
- D - DELETE (DELETE)

As part of the application development process we perform CRUD Operations using REST APIs.

6.2.5 Creating Table

Before getting into action with respect to basic DML and queries or CRUD operations, we need to prepare tables.

At this time we have not covered DDL yet. All database operations related to managing tables come under DDL.

For now, let's just create the table by copy pasting below `CREATE TABLE` statement. We will get into concepts as part of the subsequent sections.

- Connect to the database.
- Create the table.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_catalog = 'itversity_sms_db' AND table_schema = 'public'
LIMIT 10
```

```
2 rows affected.
```

```
[('itversity_sms_db', 'public', 'courses', 'BASE TABLE', None, None, None, None, None,
↪ 'YES', 'NO', None),
 ('itversity_sms_db', 'public', 'users', 'BASE TABLE', None, None, None, None, None,
↪'YES', 'NO', None)]
```

```
%%sql

DROP TABLE IF EXISTS users;
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_catalog = 'itversity_sms_db' AND table_schema = 'public'
LIMIT 10
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[('itversity_sms_db', 'public', 'courses', 'BASE TABLE', None, None, None, None, None,
↪ 'YES', 'NO', None)]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    create_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

- Let us validate the objects that are created in the underlying database. We can either run query against **information_schema** or use Database Explorer in **SQL Workbench** or even `psql`.

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_catalog = 'itversity_sms_db' AND table_schema = 'public'
LIMIT 10
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[('itversity_sms_db', 'public', 'users', 'BASE TABLE', None, None, None, None, None,
↪'YES', 'NO', None)]
```

```
%%sql

SELECT * FROM information_schema.columns
WHERE table_name = 'users'
LIMIT 10
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
10 rows affected.
```

```
[('itversity_sms_db', 'public', 'users', 'user_id', 1, "nextval('users_user_id_seq
↪'::regclass)", 'NO', 'integer', None, None, 32, 2, 0, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'int4',␣
↪None, None, None, None, '1', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES'),
 ('itversity_sms_db', 'public', 'users', 'user_first_name', 2, None, 'NO', 'character␣
↪varying', 30, 120, None, None, None, None, None, None, None, None, None, None, None,
↪ None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'varchar', None, None,␣
↪None, None, '2', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES
↪'),
 ('itversity_sms_db', 'public', 'users', 'user_last_name', 3, None, 'NO', 'character␣
↪varying', 30, 120, None, None, None, None, None, None, None, None, None, None, None,
↪ None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'varchar', None, None,␣
↪None, None, '3', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES
↪'),
 ('itversity_sms_db', 'public', 'users', 'user_email_id', 4, None, 'NO', 'character␣
↪varying', 50, 200, None, None, None, None, None, None, None, None, None, None, None,
↪ None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'varchar', None, None,␣
↪None, None, '4', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES
↪'),
 ('itversity_sms_db', 'public', 'users', 'user_email_validated', 5, 'false', 'YES',
↪'boolean', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'bool', None, None,␣
↪None, None, '5', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES
↪'),
 ('itversity_sms_db', 'public', 'users', 'user_password', 6, None, 'YES', 'character␣
↪varying', 200, 800, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'varchar', None,␣
↪None, None, None, '6', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER',␣
↪None, 'YES'),
 ('itversity_sms_db', 'public', 'users', 'user_role', 7, "'U'::character varying", 'NO
↪', 'character varying', 1, 4, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, 'itversity_sms_db', 'pg_catalog', 'varchar',␣
↪None, None, None, None, '7', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES'),
 ('itversity_sms_db', 'public', 'users', 'is_active', 8, 'false', 'YES', 'boolean',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_sms_db', 'pg_catalog', 'bool', None, None, None, None,
↪'8', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES'),
 ('itversity_sms_db', 'public', 'users', 'create_ts', 9, 'CURRENT_TIMESTAMP', 'YES',
↪'timestamp without time zone', None, None, None, None, None, 6, None, None, None,␣
↪None, None, None, None, None, None, None, None, 'itversity_sms_db', 'pg_catalog',
↪'timestamp', None, None, None, None, '9', 'NO', 'NO', None, None, None, None, None,
↪'NO', 'NEVER', None, 'YES'),
```

(continues on next page)

(continued from previous page)

```
 ('itversity_sms_db', 'public', 'users', 'last_updated_ts', 10, 'CURRENT_TIMESTAMP',
↪'YES', 'timestamp without time zone', None, None, None, None, None, 6, None, None,␣
↪None, None, None, None, None, None, None, None, None, 'itversity_sms_db', 'pg_
↪catalog', 'timestamp', None, None, None, None, '10', 'NO', 'NO', None, None, None,␣
↪None, None, 'NO', 'NEVER', None, 'YES')]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

6.2.6 Inserting Data

Let us see how to insert the data into the table.

- We need to use INSERT clause to insert the data. Here is the sample syntax.

```
INSERT INTO <table_name> (col1, col2, col3)
VALUES (val1, val2, val3)
```

- If we don't pass columns after table name then we need to specify values for all the columns. It is not good practice to insert records with out specifying column names.
- If we do not specify value for `SERIAL` field, a sequence generated number will be used.
- It is not mandatory to pass the values for those fields where `DEFAULT` is specified. Values specified in `DEFAULT` clause will be used.
- It is mandatory to specify columns and corresponding values for all columns where `NOT NULL` is specified.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%sql TRUNCATE TABLE users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Scott', 'Tiger', 'scott@tiger.com')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 11, 813351), datetime.datetime(2020, 11, 14, 15, 35,␣
↪11, 813351))]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Donald', 'Duck', 'donald@duck.com')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
2 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 11, 813351), datetime.datetime(2020, 11, 14, 15, 35,␣
↪11, 813351)),
 (2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 14, 495991), datetime.datetime(2020, 11, 14, 15, 35,␣
↪14, 495991))]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_role, is_
↪active)
VALUES ('Mickey', 'Mouse', 'mickey@mouse.com', 'U', true)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 11, 813351), datetime.datetime(2020, 11, 14, 15, 35,␣
↪11, 813351)),
 (2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 14, 495991), datetime.datetime(2020, 11, 14, 15, 35,␣
↪14, 495991)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', True, datetime.
↪datetime(2020, 11, 14, 15, 35, 15, 881686), datetime.datetime(2020, 11, 14, 15, 35,␣
↪15, 881686))]
```

```
%%sql

INSERT INTO users
    (user_first_name, user_last_name, user_email_id, user_password, user_role, is_
↪active)
VALUES
    ('Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true),
    ('Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true),
    ('Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 11, 813351), datetime.datetime(2020, 11, 14, 15, 35,␣
↪11, 813351)),
 (2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 14, 495991), datetime.datetime(2020, 11, 14, 15, 35,␣
↪14, 495991)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', True, datetime.
↪datetime(2020, 11, 14, 15, 35, 15, 881686), datetime.datetime(2020, 11, 14, 15, 35,␣
↪15, 881686)),
 (4, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', False, 'h9LAz7p7ub', 'U',␣
↪True, datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020,␣
↪11, 14, 15, 35, 20, 938583)),
 (5, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', False, 'oEofndp', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020, 11, 14,
↪ 15, 35, 20, 938583)),
 (6, 'Addie', 'Mesias', 'amesias2@twitpic.com', False, 'ih7Y69u56', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020, 11, 14,
↪ 15, 35, 20, 938583))]
```

6.2.7 Updating Data

Let us see how we can update data in the table.

- Typical syntax

```
UPDATE <table_name>
SET
    col1 = val1,
    col2 = val2
WHERE <condition>
```

- If WHERE condition is not specified all rows in the table will be updated.
- For now we will see basic examples for update. One need to have good knowledge about WHERE clause to take care of complex conditions. Using WHERE will be covered extensively as part of filtering the data at a later point in time.
- Set user role for user_id 1 as 'A'

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 11, 813351), datetime.datetime(2020, 11, 14, 15, 35,␣
↪11, 813351)),
 (2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 14, 495991), datetime.datetime(2020, 11, 14, 15, 35,␣
↪14, 495991)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', True, datetime.
↪datetime(2020, 11, 14, 15, 35, 15, 881686), datetime.datetime(2020, 11, 14, 15, 35,␣
↪15, 881686)),
 (4, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', False, 'h9LAz7p7ub', 'U',␣
↪True, datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020,␣
↪11, 14, 15, 35, 20, 938583)),
 (5, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', False, 'oEofndp', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020, 11, 14,
↪ 15, 35, 20, 938583)),
 (6, 'Addie', 'Mesias', 'amesias2@twitpic.com', False, 'ih7Y69u56', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020, 11, 14,
↪ 15, 35, 20, 938583))]
```

```
%%sql

UPDATE users
    SET user_role = 'A'
WHERE user_id = 1
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 14, 495991), datetime.datetime(2020, 11, 14, 15, 35,␣
↪14, 495991)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', True, datetime.
↪datetime(2020, 11, 14, 15, 35, 15, 881686), datetime.datetime(2020, 11, 14, 15, 35,␣
↪15, 881686)),
 (4, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', False, 'h9LAz7p7ub', 'U',␣
↪True, datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020,␣
↪11, 14, 15, 35, 20, 938583)),
 (5, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', False, 'oEofndp', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020, 11, 14,
↪ 15, 35, 20, 938583)),
 (6, 'Addie', 'Mesias', 'amesias2@twitpic.com', False, 'ih7Y69u56', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 35, 20, 938583), datetime.datetime(2020, 11, 14,
↪ 15, 35, 20, 938583)),
 (1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'A', False, datetime.
↪datetime(2020, 11, 14, 15, 35, 11, 813351), datetime.datetime(2020, 11, 14, 15, 35,␣
↪11, 813351))]
```

- Set user_email_validated as well as is_active to true for all users

```
%sql SELECT user_id, user_email_validated, is_active FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, False, False),
 (3, False, True),
 (4, False, True),
 (5, False, True),
 (6, False, True),
 (1, False, False)]
```

```
%%sql

UPDATE users
SET
    user_email_validated = true,
    is_active = true
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[]
```

```
%sql SELECT user_id, user_email_validated, is_active FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, True, True),
 (3, True, True),
 (4, True, True),
 (5, True, True),
 (6, True, True),
 (1, True, True)]
```

- Convert case of user_email_id to upper for all the records

```
%sql SELECT user_id, user_email_id FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, 'donald@duck.com'),
 (3, 'mickey@mouse.com'),
 (4, 'gbradock0@barnesandnoble.com'),
 (5, 'tlyness1@paginegialle.it'),
 (6, 'amesias2@twitpic.com'),
 (1, 'scott@tiger.com')]
```

```
%%sql

UPDATE users
SET
    user_email_id = upper(user_email_id)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[]
```

```
%sql SELECT user_id, user_email_id FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, 'DONALD@DUCK.COM'),
 (3, 'MICKEY@MOUSE.COM'),
 (4, 'GBRADOCK0@BARNESANDNOBLE.COM'),
 (5, 'TLYNESS1@PAGINEGIALLE.IT'),
 (6, 'AMESIAS2@TWITPIC.COM'),
 (1, 'SCOTT@TIGER.COM')]
```

- Add new column by name **user_full_name** and update it by concatenating **user_first_name** and **user_last_name**.

```
%%sql

ALTER TABLE users ADD COLUMN user_full_name VARCHAR(50)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%sql SELECT user_id, user_first_name, user_last_name, user_full_name FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, 'Donald', 'Duck', None),
 (3, 'Mickey', 'Mouse', None),
 (4, 'Gordan', 'Bradock', None),
 (5, 'Tobe', 'Lyness', None),
 (6, 'Addie', 'Mesias', None),
 (1, 'Scott', 'Tiger', None)]
```

```
%sql SELECT concat(user_first_name, ' ', user_last_name) FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[('Donald Duck',),
 ('Mickey Mouse',),
 ('Gordan Bradock',),
 ('Tobe Lyness',),
 ('Addie Mesias',),
 ('Scott Tiger',)]
```

```
%%sql

UPDATE users
    SET user_full_name = upper(concat(user_first_name, ' ', user_last_name))
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[]
```

```
%sql SELECT user_id, user_first_name, user_last_name, user_full_name FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, 'Donald', 'Duck', 'DONALD DUCK'),
 (3, 'Mickey', 'Mouse', 'MICKEY MOUSE'),
 (4, 'Gordan', 'Bradock', 'GORDAN BRADOCK'),
 (5, 'Tobe', 'Lyness', 'TOBE LYNESS'),
 (6, 'Addie', 'Mesias', 'ADDIE MESIAS'),
 (1, 'Scott', 'Tiger', 'SCOTT TIGER')]
```

6.2.8 Deleting Data

Let us understand how to delete the data from a table.

- Typical Syntax - `DELETE FROM <table> WHERE <condition>`.
- If we do not specify condition, it will delete all the data from the table.
- It is not recommended to use delete with out where condition to delete all the data (instead we should use `TRUNCATE`).
- For now we will see basic examples for delete. One need to have good knowledge about `WHERE` clause to take care of complex conditions.
- Let's see how we can delete all those records from users where the password is not set. We need to use `IS NULL` as condition to compare against Null values.

```
%sql SELECT user_id, user_password FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
6 rows affected.
```

```
[(2, None),
 (3, None),
 (4, 'h9LAz7p7ub'),
 (5, 'oEofndp'),
 (6, 'ih7Y69u56'),
 (1, None)]
```

```
%sql DELETE FROM users WHERE user_password IS NULL
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[]
```

```
%sql SELECT user_id, user_password FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(4, 'h9LAz7p7ub'), (5, 'oEofndp'), (6, 'ih7Y69u56')]
```

```
%sql SELECT count(1) FROM users
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(3,)]
```

6.2.9 Overview of Transactions

Let us go through the details related to Transactions.

- We typically perform operations such as `COMMIT` and `ROLLBACK` via the applications.
- `COMMIT` will persist the changes in the database.
- `ROLLBACK` will revert the uncommitted changes in the database.
- We typically rollback the uncommitted changes in a transaction if there is any exception as part of the application logic flow.
- For example, once the order is placed all the items that are added to shopping cart will be rolled back if the payment using credit card fails.
- By default every operation is typically committed in Postgres. We will get into the details related to transaction as part of application development later.
- Commands such as `COMMIT`, `ROLLBACK` typically comes under TCL (Transaction Control Language)

6.2.10 Exercises - Database Operations

Let's create a table and perform database operations using direct SQL.

Exercise 1 - Create Table

Create table - **courses**

- course_id - sequence generated integer and primary key
- course_name - which holds alpha numeric or string values up to 60 characters
- course_author - which holds the name of the author up to 40 characters
- course_status - which holds one of these values (published, draft, inactive).
- course_published_dt - which holds date type value.

Provide the script as answer for this exercise.

Exercise 2 - Inserting Data

- Insert data into courses using the data provided. Make sure id is system generated.

Course Name	Course Author	Course Status	Course Published Date
Programming using Python	Bob Dillon	published	2020-09-30
Data Engineering using Python	Bob Dillon	published	2020-07-15
Data Engineering using Scala	Elvis Presley	draft	
Programming using Scala	Elvis Presley	published	2020-05-12
Programming using Java	Mike Jack	inactive	2020-08-10
Web Applications - Python Flask	Bob Dillon	inactive	2020-07-20
Web Applications - Java Spring	Mike Jack	draft	
Pipeline Orchestration - Python	Bob Dillon	draft	
Streaming Pipelines - Python	Bob Dillon	published	2020-10-05
Web Applications - Scala Play	Elvis Presley	inactive	2020-09-30
Web Applications - Python Django	Bob Dillon	published	2020-06-23
Server Automation - Ansible	Uncle Sam	published	2020-07-05

Provide the insert statement(s) as answer for this exercise.

Exercise 3 - Updating Data

Update the status of all the **draft courses** related to Python and Scala to **published** along with the **course_published_dt using system date**.

Provide the update statement as answer for this exercise.

Exercise 4 - Deleting Data

Delete all the courses which are neither in draft mode nor published.

Provide the delete statement as answer for this exercise.

Validation - Get count of all published courses by author and make sure output is sorted in descending order by count.

```
SELECT course_author, count(1) AS course_count
FROM courses
WHERE course_status= 'published'
GROUP BY course_author
```

Course Author	Course Count
Bob Dillon	5
Elvis Presley	2
Uncle Sam	1

6.3 Writing Basic SQL Queries

As part of this section we will primarily focus on writing basic queries.

- Standard Transformations
- Overview of Data Model
- Define Problem Statement – Daily Product Revenue
- Preparing Tables
- Selecting or Projecting Data
- Filtering Data
- Joining Tables – Inner
- Joining Tables – Outer
- Performing Aggregations
- Sorting Data
- Solution – Daily Product Revenue

Here are the key objectives for this section

- What are different standard transformations and how they are implemented using Basic SQL?
- Understand the data model using which basic SQL features are explored?

- Setup the database, tables and load the data quickly
- How we typically select or project the data, filter the data, join data from multiple tables, compute metrics using aggregate functions, sort the data etc?
- While exploring basic SQL queries, we will define a problem statement and come up with a solution at the end.
- Self evaluate whether one understood all the key aspects of writing basic SQL queries using exercises at the end.

6.3.1 Standard Transformations

Here are some of the transformations we typically perform on regular basis.

- Projection of data
- Filtering data
- Performing Aggregations
- Joins
- Sorting
- Ranking (will be covered as part of advanced queries)

6.3.2 Overview of Data Model

We will be using retail data model for this section. It contains 6 tables.

- Table list
  - orders
  - order_items
  - products
  - categories
  - departments
  - customers
- **orders** and **order_items** are transactional tables.
- **products**, **categories** and **departments** are non transactional tables which have data related to product catalog.
- **customers** is a non transactional table which have customer details.
- There is 1 to many relationship between **orders** and **order_items**.
- There is 1 to many relationship between **products** and **order_items**. Each order item will have one product and product can be part of many order_items.
- There is 1 to many relationship between **customers** and **orders**. A customer can place many orders over a period of time but there cannot be more than one customer for a given order.
- There is 1 to many relationship between **departments** and **categories**. Also there is 1 to many relationship between **categories** and **products**.
- There is hierarchical relationship from departments to products - **departments** -> **categories** -> **products**

6.3.3 Define Problem Statement – Daily Product Revenue

Let us try to get daily product revenue using retail tables.

- daily is derived from orders.order_date.
- product has to be derived from products.product_name.
- revenue has to be derived from order_items.order_item_subtotal.
- We need to join all the 3 tables, then group by order_date, product_id as well as product_name to get revenue using order_item_subtotal.
- Get Daily Product Revenue using products, orders and order_items data set.
- We have following fields in **orders**.
  - order_id
  - order_date
  - order_customer_id
  - order_status
- We have following fields in **order_items**.
  - order_item_id
  - order_item_order_id
  - order_item_product_id
  - order_item_quantity
  - order_item_subtotal
  - order_item_product_price
- We have following fields in **products**
  - product_id
  - product_category_id
  - product_name
  - product_description
  - product_price
  - product_image
- We have one to many relationship between orders and order_items.
- **orders.order_id** is **primary key** and **order_items.order_item_order_id** is foreign key to **orders.order_id**.
- We have one to many relationship between products and order_items.
- **products.product_id** is **primary key** and **order_items.order_item_product_id** is foreign key to **products.product_id**
- By the end of this module we will explore all standard transformations and get daily product revenue using following fields.
  - **orders.order_date**
  - **order_items.order_item_product_id**
  - **products.product_name**

  - **order_items.order_item_subtotal** (aggregated using date and product_id).
- We will consider only **COMPLETE** or **CLOSED** orders.
- As there can be more than one product names with different ids, we have to include product_id as part of the key using which we will group the data.

6.3.4 Preparing Tables

Let us prepare retail tables to come up with the solution for the problem statement.

- Ensure that we have required database and user for retail data. We might provide the database as part of our labs. Here are the instructions to use `psql` for setting up the required tables.

```
psql -U postgres -h localhost -p 5432 -W
```

```
CREATE DATABASE itversity_retail_db;
CREATE USER itversity_retail_user WITH ENCRYPTED PASSWORD 'retail_password';
GRANT ALL ON DATABASE itversity_retail_db TO itversity_retail_user;
```

- Create Tables using the script provided. You can either use `psql` or **SQL Alchemy**.

```
psql -U itversity_retail_user \
  -h localhost \
  -p 5432 \
  -d itversity_retail_db \
  -W

\i /data/retail_db/create_db_tables_pg.sql
```

- Data shall be loaded using the script provided.

```
\i /data/retail_db/load_db_tables_pg.sql
```

- Run queries to validate we have data in all the 3 tables.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql SELECT current_database()
```

```
1 rows affected.
```

```
[('itversity_retail_db',)]
```

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_catalog = 'itversity_retail_db'
```

(continues on next page)

(continued from previous page)

```
    AND table_schema = 'public'
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[('itversity_retail_db', 'public', 'categories', 'BASE TABLE', None, None, None, None,
↪ None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'departments', 'BASE TABLE', None, None, None,␣
↪None, None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'products', 'BASE TABLE', None, None, None, None,␣
↪None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'customers', 'BASE TABLE', None, None, None, None,␣
↪None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'orders', 'BASE TABLE', None, None, None, None,␣
↪None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'order_items', 'BASE TABLE', None, None, None,␣
↪None, None, 'YES', 'NO', None)]
```

```
%sql SELECT * FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 11599, 'CLOSED'),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT'),
 (3, datetime.datetime(2013, 7, 25, 0, 0), 12111, 'COMPLETE'),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED'),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE'),
 (6, datetime.datetime(2013, 7, 25, 0, 0), 7130, 'COMPLETE'),
 (7, datetime.datetime(2013, 7, 25, 0, 0), 4530, 'COMPLETE'),
 (8, datetime.datetime(2013, 7, 25, 0, 0), 2911, 'PROCESSING'),
 (9, datetime.datetime(2013, 7, 25, 0, 0), 5657, 'PENDING_PAYMENT'),
 (10, datetime.datetime(2013, 7, 25, 0, 0), 5648, 'PENDING_PAYMENT')]
```

```
%sql SELECT * FROM order_items LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, 1, 957, 1, 299.98, 299.98),
 (2, 2, 1073, 1, 199.99, 199.99),
 (3, 2, 502, 5, 250.0, 50.0),
 (4, 2, 403, 1, 129.99, 129.99),
 (5, 4, 897, 2, 49.98, 24.99),
 (6, 4, 365, 5, 299.95, 59.99),
 (7, 4, 502, 3, 150.0, 50.0),
 (8, 4, 1014, 4, 199.92, 49.98),
 (9, 5, 957, 1, 299.98, 299.98),
 (10, 5, 365, 5, 299.95, 59.99)]
```

```
%sql SELECT * FROM products LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, 2, 'Quest Q64 10 FT. x 10 FT. Slant Leg Instant U', '', 59.98, 'http://images.
↪acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy'),
 (2, 2, "Under Armour Men's Highlight MC Football Clea", '', 129.99, 'http://images.
↪acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat'),
 (3, 2, "Under Armour Men's Renegade D Mid Football Cl", '', 89.99, 'http://images.
↪acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat'),
 (4, 2, "Under Armour Men's Renegade D Mid Football Cl", '', 89.99, 'http://images.
↪acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat'),
 (5, 2, 'Riddell Youth Revolution Speed Custom Footbal', '', 199.99, 'http://images.
↪acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet'),
 (6, 2, "Jordan Men's VI Retro TD Football Cleat", '', 134.99, 'http://images.
↪acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat'),
 (7, 2, 'Schutt Youth Recruit Hybrid Custom Football H', '', 99.99, 'http://images.
↪acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014'),
 (8, 2, "Nike Men's Vapor Carbon Elite TD Football Cle", '', 129.99, 'http://images.
↪acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat'),
 (9, 2, 'Nike Adult Vapor Jet 3.0 Receiver Gloves', '', 50.0, 'http://images.
↪acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves'),
 (10, 2, "Under Armour Men's Highlight MC Football Clea", '', 129.99, 'http://images.
↪acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat')]
```

```
%sql SELECT count(1) FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(68883,)]
```

```
%sql SELECT count(1) FROM order_items
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%sql SELECT count(1) FROM products
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1345,)]
```

6.3.5 Selecting or Projecting Data

Let us understand different aspects of projecting data. We primarily using `SELECT` to project the data.

- We can project all columns using `*` or some columns using column names.
- We can provide aliases to a column or expression using `AS` in `SELECT` clause.
- `DISTINCT` can be used to get the distinct records from selected columns. We can also use `DISTINCT *` to get unique records using all the columns.
- As part of `SELECT` clause we can have aggregate functions such as `count`, `sum` etc.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql SELECT * FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 11599, 'CLOSED'),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT'),
 (3, datetime.datetime(2013, 7, 25, 0, 0), 12111, 'COMPLETE'),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED'),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE'),
 (6, datetime.datetime(2013, 7, 25, 0, 0), 7130, 'COMPLETE'),
 (7, datetime.datetime(2013, 7, 25, 0, 0), 4530, 'COMPLETE'),
 (8, datetime.datetime(2013, 7, 25, 0, 0), 2911, 'PROCESSING'),
 (9, datetime.datetime(2013, 7, 25, 0, 0), 5657, 'PENDING_PAYMENT'),
 (10, datetime.datetime(2013, 7, 25, 0, 0), 5648, 'PENDING_PAYMENT')]
```

```
%%sql

SELECT * FROM information_schema.columns
WHERE table_catalog = 'itversity_retail_db'
    AND table_name = 'orders'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
4 rows affected.
```

```
[('itversity_retail_db', 'public', 'orders', 'order_id', 1, None, 'NO', 'integer',␣
↪None, None, 32, 2, 0, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', 'int4', None, None, None, None, '1
↪', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES'),
 ('itversity_retail_db', 'public', 'orders', 'order_date', 2, None, 'NO', 'timestamp␣
↪without time zone', None, None, None, None, None, 6, None, None, None, None, None,␣
↪None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'timestamp
↪', None, None, None, None, '2', 'NO', 'NO', None, None, None, None, None, 'NO',
↪'NEVER', None, 'YES'),
```

(continues on next page)

(continued from previous page)

```
 ('itversity_retail_db', 'public', 'orders', 'order_customer_id', 3, None, 'NO',
↪'integer', None, None, 32, 2, 0, None, None, None, None, None, None, None,␣
↪None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'int4', None, None,␣
↪None, None, '3', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES
↪'),
 ('itversity_retail_db', 'public', 'orders', 'order_status', 4, None, 'NO',
↪'character varying', 45, 180, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'varchar',␣
↪None, None, None, None, '4', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES')]
```

```
%%sql

SELECT order_customer_id, order_date, order_status
FROM orders
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(11599, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED'),
 (256, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT'),
 (12111, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE'),
 (8827, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED'),
 (11318, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE'),
 (7130, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE'),
 (4530, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE'),
 (2911, datetime.datetime(2013, 7, 25, 0, 0), 'PROCESSING'),
 (5657, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT'),
 (5648, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT')]
```

```
%%sql

SELECT order_customer_id,
    to_char(order_date, 'yyyy-MM'),
    order_status
FROM orders
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(11599, '2013-07', 'CLOSED'),
 (256, '2013-07', 'PENDING_PAYMENT'),
 (12111, '2013-07', 'COMPLETE'),
 (8827, '2013-07', 'CLOSED'),
 (11318, '2013-07', 'COMPLETE'),
 (7130, '2013-07', 'COMPLETE'),
 (4530, '2013-07', 'COMPLETE'),
 (2911, '2013-07', 'PROCESSING'),
 (5657, '2013-07', 'PENDING_PAYMENT'),
 (5648, '2013-07', 'PENDING_PAYMENT')]
```

```
%%sql

SELECT order_customer_id,
    to_char(order_date, 'yyyy-MM') AS order_month,
    order_status
FROM orders
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(11599, '2013-07', 'CLOSED'),
 (256, '2013-07', 'PENDING_PAYMENT'),
 (12111, '2013-07', 'COMPLETE'),
 (8827, '2013-07', 'CLOSED'),
 (11318, '2013-07', 'COMPLETE'),
 (7130, '2013-07', 'COMPLETE'),
 (4530, '2013-07', 'COMPLETE'),
 (2911, '2013-07', 'PROCESSING'),
 (5657, '2013-07', 'PENDING_PAYMENT'),
 (5648, '2013-07', 'PENDING_PAYMENT')]
```

```
%%sql

SELECT DISTINCT to_char(order_date, 'yyyy-MM') AS order_month
FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
13 rows affected.
```

```
[('2014-01',),
 ('2014-05',),
 ('2013-12',),
 ('2013-11',),
 ('2014-04',),
 ('2014-07',),
 ('2014-03',),
 ('2013-08',),
 ('2013-10',),
 ('2013-07',),
 ('2014-02',),
 ('2013-09',),
 ('2014-06',)]
```

```
%sql SELECT count(1) FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(68883,)]
```

```
%%sql

```

(continues on next page)

(continued from previous page)

```
SELECT count(DISTINCT to_char(order_date, 'yyyy-MM')) AS distinct_month_count
FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(13,)]
```

6.3.6 Filtering Data

Let us understand how we can filter the data as part of our queries.

- We use `WHERE` clause to filter the data.
- All comparison operators such as =, !=, >, <, <=, >= etc can be used to compare a column or expression or literal with another column or expression or literal.
- We can use operators such as `LIKE` with `%` or ~ with regular expressions for pattern matching.
- Boolean `OR` and `AND` can be performed when we want to apply multiple conditions.
  - Get all orders with order_status equals to COMPLETE or CLOSED. We can also use IN operator.
  - Get all orders from month 2014 January with order_status equals to COMPLETE or CLOSED
- We can also use `BETWEEN` along with `AND` to compare a column or expression against range of values.
- We need to use `IS NULL` and `IS NOT NULL` to compare against null values.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT * FROM orders
WHERE order_status = 'COMPLETE'
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(3, datetime.datetime(2013, 7, 25, 0, 0), 12111, 'COMPLETE'),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE'),
 (6, datetime.datetime(2013, 7, 25, 0, 0), 7130, 'COMPLETE'),
 (7, datetime.datetime(2013, 7, 25, 0, 0), 4530, 'COMPLETE'),
 (15, datetime.datetime(2013, 7, 25, 0, 0), 2568, 'COMPLETE'),
```

(continues on next page)

(continued from previous page)

```
 (17, datetime.datetime(2013, 7, 25, 0, 0), 2667, 'COMPLETE'),
 (22, datetime.datetime(2013, 7, 25, 0, 0), 333, 'COMPLETE'),
 (26, datetime.datetime(2013, 7, 25, 0, 0), 7562, 'COMPLETE'),
 (28, datetime.datetime(2013, 7, 25, 0, 0), 656, 'COMPLETE'),
 (32, datetime.datetime(2013, 7, 25, 0, 0), 3960, 'COMPLETE')]
```

```
%sql SELECT count(1) FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(68883,)]
```

```
%%sql

SELECT count(1)
FROM orders
WHERE order_status = 'COMPLETE'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(22899,)]
```

```
%%sql

SELECT DISTINCT order_status
FROM orders
WHERE order_status = 'COMPLETE'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('COMPLETE',)]
```

```
%%sql

SELECT DISTINCT order_status
FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9 rows affected.
```

```
[('COMPLETE',),
 ('ON_HOLD',),
 ('PENDING_PAYMENT',),
 ('PENDING',),
 ('CLOSED',),
 ('CANCELED',),
 ('PROCESSING',),
 ('PAYMENT_REVIEW',),
 ('SUSPECTED_FRAUD',)]
```

```
%%sql

SELECT * FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 11599, 'CLOSED'),
 (3, datetime.datetime(2013, 7, 25, 0, 0), 12111, 'COMPLETE'),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED'),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE'),
 (6, datetime.datetime(2013, 7, 25, 0, 0), 7130, 'COMPLETE'),
 (7, datetime.datetime(2013, 7, 25, 0, 0), 4530, 'COMPLETE'),
 (12, datetime.datetime(2013, 7, 25, 0, 0), 1837, 'CLOSED'),
 (15, datetime.datetime(2013, 7, 25, 0, 0), 2568, 'COMPLETE'),
 (17, datetime.datetime(2013, 7, 25, 0, 0), 2667, 'COMPLETE'),
 (18, datetime.datetime(2013, 7, 25, 0, 0), 1205, 'CLOSED')]
```

```
%%sql

SELECT count(1) FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(30455,)]
```

```
%%sql

SELECT count(1) FROM orders
WHERE order_status = 'COMPLETE' OR order_status = 'CLOSED'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(30455,)]
```

```
%%sql

SELECT * FROM orders
WHERE order_date = '2014-01-01'
LIMIT 3
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[(25876, datetime.datetime(2014, 1, 1, 0, 0), 3414, 'PENDING_PAYMENT'),
 (25877, datetime.datetime(2014, 1, 1, 0, 0), 5549, 'PENDING_PAYMENT'),
 (25878, datetime.datetime(2014, 1, 1, 0, 0), 9084, 'PENDING')]
```

Note: This query will not work as LIKE cannot be used to compare against columns with date data type

```
%%sql

SELECT * FROM orders
WHERE order_date LIKE '2014-01%'
LIMIT 3
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedFunction) operator does not exist: timestamp without time␣
↪zone ~~ unknown
LINE 2: WHERE order_date LIKE '2014-01%'
                         ^
HINT:  No operator matches the given name and argument types. You might need to add␣
↪explicit type casts.

[SQL: SELECT * FROM orders
WHERE order_date LIKE '2014-01%%'
LIMIT 3]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

SELECT * FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM-dd') LIKE '2014-01%'
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(25882, datetime.datetime(2014, 1, 1, 0, 0), 4598, 'COMPLETE'),
 (25888, datetime.datetime(2014, 1, 1, 0, 0), 6735, 'COMPLETE'),
 (25889, datetime.datetime(2014, 1, 1, 0, 0), 10045, 'COMPLETE'),
 (25891, datetime.datetime(2014, 1, 1, 0, 0), 3037, 'CLOSED'),
 (25895, datetime.datetime(2014, 1, 1, 0, 0), 1044, 'COMPLETE'),
 (25897, datetime.datetime(2014, 1, 1, 0, 0), 6405, 'COMPLETE'),
 (25898, datetime.datetime(2014, 1, 1, 0, 0), 3950, 'COMPLETE'),
 (25899, datetime.datetime(2014, 1, 1, 0, 0), 8068, 'CLOSED'),
 (25900, datetime.datetime(2014, 1, 1, 0, 0), 2382, 'CLOSED'),
 (25901, datetime.datetime(2014, 1, 1, 0, 0), 3099, 'COMPLETE')]
```

```
%%sql

SELECT count(1) FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM-dd') LIKE '2014-01%'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2544,)]
```

```
%%sql

SELECT * FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM') = '2014-01'
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(25882, datetime.datetime(2014, 1, 1, 0, 0), 4598, 'COMPLETE'),
 (25888, datetime.datetime(2014, 1, 1, 0, 0), 6735, 'COMPLETE'),
 (25889, datetime.datetime(2014, 1, 1, 0, 0), 10045, 'COMPLETE'),
 (25891, datetime.datetime(2014, 1, 1, 0, 0), 3037, 'CLOSED'),
 (25895, datetime.datetime(2014, 1, 1, 0, 0), 1044, 'COMPLETE'),
 (25897, datetime.datetime(2014, 1, 1, 0, 0), 6405, 'COMPLETE'),
 (25898, datetime.datetime(2014, 1, 1, 0, 0), 3950, 'COMPLETE'),
 (25899, datetime.datetime(2014, 1, 1, 0, 0), 8068, 'CLOSED'),
 (25900, datetime.datetime(2014, 1, 1, 0, 0), 2382, 'CLOSED'),
 (25901, datetime.datetime(2014, 1, 1, 0, 0), 3099, 'COMPLETE')]
```

```
%%sql

SELECT count(1) FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM') = '2014-01'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2544,)]
```

```
%%sql

SELECT count(1) FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM-dd') ~ '2014-01'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2544,)]
```

```
%%sql

SELECT count(1), min(order_date), max(order_date), count(DISTINCT order_date)
FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
    AND order_date BETWEEN '2014-01-01' AND '2014-03-31'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(7594, datetime.datetime(2014, 1, 1, 0, 0), datetime.datetime(2014, 3, 31, 0, 0),␣
↪89)]
```

```
%%sql

SELECT DISTINCT order_date
FROM orders
WHERE to_char(order_date, 'yyyy-MM') LIKE '2014-03%'
ORDER BY order_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[(datetime.datetime(2014, 3, 1, 0, 0),),
 (datetime.datetime(2014, 3, 2, 0, 0),),
 (datetime.datetime(2014, 3, 3, 0, 0),),
 (datetime.datetime(2014, 3, 4, 0, 0),),
 (datetime.datetime(2014, 3, 5, 0, 0),),
 (datetime.datetime(2014, 3, 6, 0, 0),),
 (datetime.datetime(2014, 3, 7, 0, 0),),
 (datetime.datetime(2014, 3, 8, 0, 0),),
 (datetime.datetime(2014, 3, 10, 0, 0),),
 (datetime.datetime(2014, 3, 11, 0, 0),),
 (datetime.datetime(2014, 3, 12, 0, 0),),
 (datetime.datetime(2014, 3, 13, 0, 0),),
 (datetime.datetime(2014, 3, 14, 0, 0),),
 (datetime.datetime(2014, 3, 15, 0, 0),),
 (datetime.datetime(2014, 3, 16, 0, 0),),
 (datetime.datetime(2014, 3, 17, 0, 0),),
 (datetime.datetime(2014, 3, 18, 0, 0),),
 (datetime.datetime(2014, 3, 19, 0, 0),),
 (datetime.datetime(2014, 3, 20, 0, 0),),
 (datetime.datetime(2014, 3, 21, 0, 0),),
 (datetime.datetime(2014, 3, 22, 0, 0),),
 (datetime.datetime(2014, 3, 23, 0, 0),),
 (datetime.datetime(2014, 3, 24, 0, 0),),
 (datetime.datetime(2014, 3, 25, 0, 0),),
 (datetime.datetime(2014, 3, 26, 0, 0),),
 (datetime.datetime(2014, 3, 27, 0, 0),),
 (datetime.datetime(2014, 3, 28, 0, 0),),
 (datetime.datetime(2014, 3, 29, 0, 0),),
 (datetime.datetime(2014, 3, 30, 0, 0),),
 (datetime.datetime(2014, 3, 31, 0, 0),)]
```

```
%%sql

DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql
```

(continues on next page)

(continued from previous page)

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    create_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Donald', 'Duck', 'donald@duck.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_role, is_
↪active)
VALUES ('Mickey', 'Mouse', 'mickey@mouse.com', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users
    (user_first_name, user_last_name, user_email_id, user_password, user_role, is_
↪active)
VALUES
    ('Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true),
    ('Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true),
    ('Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 38, 53, 352984), datetime.datetime(2020, 11, 14, 15, 38,␣
↪53, 352984)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', True, datetime.
↪datetime(2020, 11, 14, 15, 38, 54, 369402), datetime.datetime(2020, 11, 14, 15, 38,␣
↪54, 369402)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', False, 'h9LAz7p7ub', 'U',␣
↪True, datetime.datetime(2020, 11, 14, 15, 38, 55, 260250), datetime.datetime(2020,␣
↪11, 14, 15, 38, 55, 260250)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', False, 'oEofndp', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 38, 55, 260250), datetime.datetime(2020, 11, 14,
↪ 15, 38, 55, 260250)),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', False, 'ih7Y69u56', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 38, 55, 260250), datetime.datetime(2020, 11, 14,
↪ 15, 38, 55, 260250))]
```

Note: This will not return any thing and not the correct way to compare against NULL. NULL is specially treated by databases and it is not same as empty string.

```
%%sql

SELECT * FROM users
WHERE user_password = NULL
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users
WHERE user_password IS NULL
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.
↪datetime(2020, 11, 14, 15, 38, 53, 352984), datetime.datetime(2020, 11, 14, 15, 38,␣
↪53, 352984)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', True, datetime.
↪datetime(2020, 11, 14, 15, 38, 54, 369402), datetime.datetime(2020, 11, 14, 15, 38,␣
↪54, 369402))]
```

```
%%sql

SELECT * FROM users
WHERE user_password IS NOT NULL
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[(3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', False, 'h9LAz7p7ub', 'U',␣
↪True, datetime.datetime(2020, 11, 14, 15, 38, 55, 260250), datetime.datetime(2020,␣
↪11, 14, 15, 38, 55, 260250)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', False, 'oEofndp', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 38, 55, 260250), datetime.datetime(2020, 11, 14,
↪ 15, 38, 55, 260250)),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', False, 'ih7Y69u56', 'U', True,␣
↪datetime.datetime(2020, 11, 14, 15, 38, 55, 260250), datetime.datetime(2020, 11, 14,
↪ 15, 38, 55, 260250))]
```

6.3.7 Joining Tables – Inner

Let us understand how to join data from multiple tables.

- We will primarily focus on ASCII style join (**JOIN with ON**).
- There are different types of joins.
  - INNER JOIN - Get all the records from both the datasets which satisfies JOIN condition.
  - OUTER JOIN - We will get into the details as part of the next topic
- Example for INNER JOIN

```
SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
LIMIT 10
```

- We can join more than 2 tables in one query. Here is how it will look like.

```
SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
    JOIN products p
    ON p.product_id = oi.order_item_product_id
LIMIT 10
```

- If we have to apply additional filters, it is recommended to use WHERE clause. ON clause should only have join conditions.
- We can have non equal join conditions as well, but they are not used that often.
- Here are some of the examples for INNER JOIN:

- Get order id, date, status and item revenue for all order items.
- Get order id, date, status and item revenue for all order items for all orders where order status is either COMPLETE or CLOSED.
- Get order id, date, status and item revenue for all order items for all orders where order status is either COMPLETE or CLOSED for the orders that are placed in the month of 2014 January.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 299.98),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 199.99),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 250.0),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 129.99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 49.98),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 299.95),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 150.0),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 199.92),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 299.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 299.95)]
```

```
%sql SELECT count(1) FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(68883,)]
```

```
%sql SELECT count(1) FROM order_items
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%%sql

SELECT count(1)
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%%sql

SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 299.98),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 49.98),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 299.95),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 150.0),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 199.92),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 299.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 299.95),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 99.96),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 299.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 129.99)]
```

```
%%sql

SELECT count(1)
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(75408,)]
```

```
%%sql
```

(continues on next page)

(continued from previous page)

```
SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM') = '2014-01'
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(25882, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 299.97),
 (25882, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 100.0),
 (25882, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 79.98),
 (25882, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 399.98),
 (25888, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 299.98),
 (25889, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 99.96),
 (25889, datetime.datetime(2014, 1, 1, 0, 0), 'COMPLETE', 19.99),
 (25891, datetime.datetime(2014, 1, 1, 0, 0), 'CLOSED', 150.0),
 (25891, datetime.datetime(2014, 1, 1, 0, 0), 'CLOSED', 50.0),
 (25891, datetime.datetime(2014, 1, 1, 0, 0), 'CLOSED', 119.97)]
```

```
%%sql

SELECT count(1)
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND to_char(order_date, 'yyyy-MM') = '2014-01'
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(6198,)]
```

6.3.8 Joining Tables - Outer

Let us understand how to perform outer joins using SQL. There are 3 different types of outer joins.

- `LEFT OUTER JOIN` (default) - Get all the records from both the datasets which satisfies JOIN condition along with those records which are in the left side table but not in the right side table.
- `RIGHT OUTER JOIN` - Get all the records from both the datasets which satisfies JOIN condition along with those records which are in the right side table but not in the left side table.
- `FULL OUTER JOIN` - left union right
- When we perform the outer join (lets say left outer join), we will see this.
  - Get all the values from both the tables when join condition satisfies.
  - If there are rows on left side table for which there are no corresponding values in right side table, all the projected column values for right side table will be null.

- Here are some of the examples for outer join.
  - Get all the orders where there are no corresponding order items.
  - Get all the order items where there are no corresponding orders.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_order_id,
    oi.order_item_subtotal
FROM orders o LEFT OUTER JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
ORDER BY o.order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 1, 299.98),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 2, 129.99),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 2, 250.0),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 2, 199.99),
 (3, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', None, None),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 199.92),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 150.0),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 299.95),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 49.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 5, 299.98)]
```

```
%%sql

SELECT count(1)
FROM orders o LEFT OUTER JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(183650,)]
```

```
%%sql

SELECT count(1)
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%%sql

SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_order_id,
    oi.order_item_subtotal
FROM orders o LEFT OUTER JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE oi.order_item_order_id IS NULL
ORDER BY o.order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(3, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', None, None),
 (6, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', None, None),
 (22, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', None, None),
 (26, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', None, None),
 (32, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', None, None),
 (40, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', None, None),
 (47, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', None, None),
 (53, datetime.datetime(2013, 7, 25, 0, 0), 'PROCESSING', None, None),
 (54, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', None, None),
 (55, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING', None, None)]
```

```
%%sql

SELECT count(1)
FROM orders o LEFT OUTER JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE oi.order_item_order_id IS NULL
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(11452,)]
```

```
%%sql

SELECT count(1)
FROM orders o LEFT OUTER JOIN order_items oi
```

(continues on next page)

(continued from previous page)

```
    ON o.order_id = oi.order_item_order_id
WHERE oi.order_item_order_id IS NULL
    AND o.order_status IN ('COMPLETE', 'CLOSED')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(5189,)]
```

```
%%sql

SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_order_id,
    oi.order_item_subtotal
FROM orders o RIGHT OUTER JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 1, 299.98),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 2, 199.99),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 2, 250.0),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 'PENDING_PAYMENT', 2, 129.99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 49.98),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 299.95),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 150.0),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 'CLOSED', 4, 199.92),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 5, 299.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 'COMPLETE', 5, 299.95)]
```

```
%%sql

SELECT count(1)
FROM orders o RIGHT OUTER JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%%sql

SELECT o.order_id,
    o.order_date,
    o.order_status,
    oi.order_item_order_id,
    oi.order_item_subtotal
FROM orders o RIGHT OUTER JOIN order_items oi
```

(continues on next page)

(continued from previous page)

```
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id IS NULL
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

6.3.9 Performing Aggregations

Let us understand how to aggregate the data.

- We can perform global aggregations as well as aggregations by key.
- Global Aggregations
  - Get total number of orders.
  - Get revenue for a given order id.
  - Get number of records with order_status either COMPLETED or CLOSED.
- Aggregations by key - using `GROUP BY`
  - Get number of orders by date or status.
  - Get revenue for each order_id.
  - Get daily product revenue (using order date and product id as keys).
- We can also use `HAVING` clause to apply filtering on top of aggregated data.
  - Get daily product revenue where revenue is greater than $500 (using order date and product id as keys).
- Rules while using `GROUP BY`.
  - We can have the columns which are specified as part of `GROUP BY` in `SELECT` clause.
  - On top of those, we can have derived columns using aggregate functions.
  - We cannot have any other columns that are not used as part of `GROUP BY` or derived column using non aggregate functions.
  - We will not be able to use aggregate functions or aliases used in the select clause as part of the where clause.
  - If we want to filter based on aggregated results, then we can leverage `HAVING` on top of `GROUP BY` (specifying `WHERE` is not an option)
- Typical query execution - FROM -> WHERE -> GROUP BY -> SELECT

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql SELECT count(order_id) FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(68883,)]
```

```
%sql SELECT count(DISTINCT order_date) FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(364,)]
```

```
%%sql

SELECT *
FROM order_items
WHERE order_item_order_id = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[(2, 2, 1073, 1, 199.99, 199.99),
 (3, 2, 502, 5, 250.0, 50.0),
 (4, 2, 403, 1, 129.99, 129.99)]
```

```
%%sql

SELECT round(sum(order_item_subtotal::numeric), 2) AS order_revenue
FROM order_items
WHERE order_item_order_id = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(Decimal('579.98'),)]
```

```
%%sql

SELECT count(1)
FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(30455,)]
```

```
%%sql

SELECT order_date,
    count(1)
FROM orders
GROUP BY order_date
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 143),
 (datetime.datetime(2013, 7, 26, 0, 0), 269),
 (datetime.datetime(2013, 7, 27, 0, 0), 202),
 (datetime.datetime(2013, 7, 28, 0, 0), 187),
 (datetime.datetime(2013, 7, 29, 0, 0), 253),
 (datetime.datetime(2013, 7, 30, 0, 0), 227),
 (datetime.datetime(2013, 7, 31, 0, 0), 252),
 (datetime.datetime(2013, 8, 1, 0, 0), 246),
 (datetime.datetime(2013, 8, 2, 0, 0), 224),
 (datetime.datetime(2013, 8, 3, 0, 0), 183)]
```

```
%%sql

SELECT order_status,
    count(1) AS status_count
FROM orders
GROUP BY order_status
ORDER BY order_status
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9 rows affected.
```

```
[('CANCELED', 1428),
 ('CLOSED', 7556),
 ('COMPLETE', 22899),
 ('ON_HOLD', 3798),
 ('PAYMENT_REVIEW', 729),
 ('PENDING', 7610),
 ('PENDING_PAYMENT', 15030),
 ('PROCESSING', 8275),
 ('SUSPECTED_FRAUD', 1558)]
```

```
%%sql

SELECT order_item_order_id,
    sum(order_item_subtotal) AS order_revenue
FROM order_items
GROUP BY order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(44127, 179.97),
 (26264, 334.96000000000004),
 (37876, 699.97),
 (55864, 600.94),
 (31789, 129.99),
 (56903, 479.97),
 (40694, 1129.75),
 (48663, 969.9200000000001),
 (47216, 1219.89),
 (37922, 1029.9)]
```

Error: This query using `round` will fail as `sum(order_item_subtotal)` will not return the data accepted by `round`. We have to convert the data type of `sum(order_item_subtotal)` to `numeric`.

```
%%sql

SELECT order_item_order_id,
    round(sum(order_item_subtotal), 2) AS order_revenue
FROM order_items
GROUP BY order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedFunction) function round(double precision, integer) does␣
↪not exist
LINE 1: SELECT order_item_order_id, round(sum(order_item_subtotal), ...
                                    ^
HINT:  No function matches the given name and argument types. You might need to add␣
↪explicit type casts.

[SQL: SELECT order_item_order_id, round(sum(order_item_subtotal), 2) AS order_revenue
FROM order_items
GROUP BY order_item_order_id
LIMIT 10]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

SELECT order_item_order_id,
    round(sum(order_item_subtotal)::numeric, 2) AS order_revenue
FROM order_items
GROUP BY order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(44127, Decimal('179.97')),
 (26264, Decimal('334.96')),
 (37876, Decimal('699.97')),
 (55864, Decimal('600.94')),
 (31789, Decimal('129.99')),
 (56903, Decimal('479.97')),
```

(continues on next page)

(continued from previous page)

```
(40694, Decimal('1129.75')),
 (48663, Decimal('969.92')),
 (47216, Decimal('1219.89')),
 (37922, Decimal('1029.90'))]
```

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date,
    oi.order_item_product_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99'))]
```

Note: We cannot use the aliases in select clause in `WHERE`. In this case **revenue** cannot be used in `WHERE` clause.

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND revenue >= 500
GROUP BY o.order_date,
    oi.order_item_product_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedColumn) column "revenue" does not exist
LINE 5:     AND revenue >= 500
                ^


[SQL: SELECT o.order_date, oi.order_item_product_id, round(sum(oi.order_item_
↪subtotal::numeric), 2) AS revenue
```

(continues on next page)

(continued from previous page)

```
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND revenue >= 500
GROUP BY o.order_date,
    oi.order_item_product_id
LIMIT 10]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

Note: We cannot use aggregate functions in `WHERE` clause.

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND round(sum(oi.order_item_subtotal::numeric), 2) >= 500
GROUP BY o.order_date,
    oi.order_item_product_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.GroupingError) aggregate functions are not allowed in WHERE
LINE 5:     AND round(sum(oi.order_item_subtotal::numeric), 2) >= 50...
                      ^

[SQL: SELECT o.order_date, oi.order_item_product_id, round(sum(oi.order_item_
↪subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND round(sum(oi.order_item_subtotal::numeric), 2) >= 500
GROUP BY o.order_date,
    oi.order_item_product_id
LIMIT 10]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date,
    oi.order_item_product_id
HAVING round(sum(oi.order_item_subtotal::numeric), 2) >= 500
ORDER BY o.order_date, revenue DESC
LIMIT 25
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
25 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99')),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46')),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67')),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54')),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32')),
 (datetime.datetime(2013, 7, 26, 0, 0), 1014, Decimal('4798.08')),
 (datetime.datetime(2013, 7, 26, 0, 0), 502, Decimal('4250.00')),
 (datetime.datetime(2013, 7, 26, 0, 0), 1073, Decimal('3999.80')),
 (datetime.datetime(2013, 7, 26, 0, 0), 403, Decimal('3249.75')),
 (datetime.datetime(2013, 7, 26, 0, 0), 627, Decimal('3039.24')),
 (datetime.datetime(2013, 7, 27, 0, 0), 1004, Decimal('9599.52')),
 (datetime.datetime(2013, 7, 27, 0, 0), 191, Decimal('5999.40')),
 (datetime.datetime(2013, 7, 27, 0, 0), 957, Decimal('5699.62')),
 (datetime.datetime(2013, 7, 27, 0, 0), 1073, Decimal('5399.73')),
 (datetime.datetime(2013, 7, 27, 0, 0), 365, Decimal('5099.15')),
 (datetime.datetime(2013, 7, 27, 0, 0), 502, Decimal('5050.00'))]
```

```
%%sql

SELECT count(1) FROM (
    SELECT o.order_date,
        oi.order_item_product_id,
        round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
    FROM orders o JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
    WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    GROUP BY o.order_date,
        oi.order_item_product_id
) q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9120,)]
```

```
%%sql

SELECT count(1) FROM (
    SELECT o.order_date,
        oi.order_item_product_id,
        round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
    FROM orders o JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
```

(continues on next page)

(continued from previous page)

```
    WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    GROUP BY o.order_date,
        oi.order_item_product_id
    HAVING round(sum(oi.order_item_subtotal::numeric), 2) >= 500
) q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(3339,)]
```

6.3.10 Sorting Data

Let us understand how to sort the data using **SQL**.

- We typically perform sorting as final step.
- Sorting can be done either by using one field or multiple fields. Sorting by multiple fields is also known as composite sorting.
- We can sort the data either in ascending order or descending order by using column or expression.
- By default, the sorting order is ascending and we can change it to descending by using `DESC`.
- As part of composite sorting, we can sort the data in ascending order on some fields and descending order on other fields.
- Typical query execution order
  1. `FROM`
  2. `WHERE`
  3. `GROUP BY` and `HAVING`
  4. `SELECT`
  5. `ORDER BY`

```
SELECT order_date, count(1) AS order_count
FROM orders
WHERE order_status IN ('COMPLETE', 'CLOSED')
GROUP BY order_date
HAVING count(1) > 50
ORDER BY order_count DESC
```

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT * FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 11599, 'CLOSED'),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT'),
 (3, datetime.datetime(2013, 7, 25, 0, 0), 12111, 'COMPLETE'),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED'),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE'),
 (6, datetime.datetime(2013, 7, 25, 0, 0), 7130, 'COMPLETE'),
 (7, datetime.datetime(2013, 7, 25, 0, 0), 4530, 'COMPLETE'),
 (8, datetime.datetime(2013, 7, 25, 0, 0), 2911, 'PROCESSING'),
 (9, datetime.datetime(2013, 7, 25, 0, 0), 5657, 'PENDING_PAYMENT'),
 (10, datetime.datetime(2013, 7, 25, 0, 0), 5648, 'PENDING_PAYMENT')]
```

```
%%sql

SELECT * FROM orders
ORDER BY order_customer_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(22945, datetime.datetime(2013, 12, 13, 0, 0), 1, 'COMPLETE'),
 (33865, datetime.datetime(2014, 2, 18, 0, 0), 2, 'COMPLETE'),
 (67863, datetime.datetime(2013, 11, 30, 0, 0), 2, 'COMPLETE'),
 (15192, datetime.datetime(2013, 10, 29, 0, 0), 2, 'PENDING_PAYMENT'),
 (57963, datetime.datetime(2013, 8, 2, 0, 0), 2, 'ON_HOLD'),
 (56178, datetime.datetime(2014, 7, 15, 0, 0), 3, 'PENDING'),
 (57617, datetime.datetime(2014, 7, 24, 0, 0), 3, 'COMPLETE'),
 (23662, datetime.datetime(2013, 12, 19, 0, 0), 3, 'COMPLETE'),
 (22646, datetime.datetime(2013, 12, 11, 0, 0), 3, 'COMPLETE'),
 (35158, datetime.datetime(2014, 2, 26, 0, 0), 3, 'COMPLETE')]
```

```
%%sql

SELECT * FROM orders
ORDER BY order_customer_id ASC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(22945, datetime.datetime(2013, 12, 13, 0, 0), 1, 'COMPLETE'),
 (33865, datetime.datetime(2014, 2, 18, 0, 0), 2, 'COMPLETE'),
 (67863, datetime.datetime(2013, 11, 30, 0, 0), 2, 'COMPLETE'),
 (15192, datetime.datetime(2013, 10, 29, 0, 0), 2, 'PENDING_PAYMENT'),
 (57963, datetime.datetime(2013, 8, 2, 0, 0), 2, 'ON_HOLD'),
 (56178, datetime.datetime(2014, 7, 15, 0, 0), 3, 'PENDING'),
 (57617, datetime.datetime(2014, 7, 24, 0, 0), 3, 'COMPLETE'),
```

(continues on next page)

(continued from previous page)

```
 (23662, datetime.datetime(2013, 12, 19, 0, 0), 3, 'COMPLETE'),
 (22646, datetime.datetime(2013, 12, 11, 0, 0), 3, 'COMPLETE'),
 (35158, datetime.datetime(2014, 2, 26, 0, 0), 3, 'COMPLETE')]
```

```
%%sql

SELECT * FROM orders
ORDER BY order_customer_id,
    order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(22945, datetime.datetime(2013, 12, 13, 0, 0), 1, 'COMPLETE'),
 (57963, datetime.datetime(2013, 8, 2, 0, 0), 2, 'ON_HOLD'),
 (15192, datetime.datetime(2013, 10, 29, 0, 0), 2, 'PENDING_PAYMENT'),
 (67863, datetime.datetime(2013, 11, 30, 0, 0), 2, 'COMPLETE'),
 (33865, datetime.datetime(2014, 2, 18, 0, 0), 2, 'COMPLETE'),
 (22646, datetime.datetime(2013, 12, 11, 0, 0), 3, 'COMPLETE'),
 (61453, datetime.datetime(2013, 12, 14, 0, 0), 3, 'COMPLETE'),
 (23662, datetime.datetime(2013, 12, 19, 0, 0), 3, 'COMPLETE'),
 (35158, datetime.datetime(2014, 2, 26, 0, 0), 3, 'COMPLETE'),
 (46399, datetime.datetime(2014, 5, 9, 0, 0), 3, 'PROCESSING')]
```

```
%%sql

SELECT * FROM orders
ORDER BY order_customer_id,
    order_date DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(22945, datetime.datetime(2013, 12, 13, 0, 0), 1, 'COMPLETE'),
 (33865, datetime.datetime(2014, 2, 18, 0, 0), 2, 'COMPLETE'),
 (67863, datetime.datetime(2013, 11, 30, 0, 0), 2, 'COMPLETE'),
 (15192, datetime.datetime(2013, 10, 29, 0, 0), 2, 'PENDING_PAYMENT'),
 (57963, datetime.datetime(2013, 8, 2, 0, 0), 2, 'ON_HOLD'),
 (57617, datetime.datetime(2014, 7, 24, 0, 0), 3, 'COMPLETE'),
 (56178, datetime.datetime(2014, 7, 15, 0, 0), 3, 'PENDING'),
 (46399, datetime.datetime(2014, 5, 9, 0, 0), 3, 'PROCESSING'),
 (35158, datetime.datetime(2014, 2, 26, 0, 0), 3, 'COMPLETE'),
 (23662, datetime.datetime(2013, 12, 19, 0, 0), 3, 'COMPLETE')]
```

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
```

(continues on next page)

(continued from previous page)

```
GROUP BY o.order_date,
    oi.order_item_product_id
ORDER BY o.order_date,
    revenue DESC
LIMIT 25
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
25 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95')),
 (datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95')),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98'))]
```

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal::numeric), 2) AS revenue
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date,
    oi.order_item_product_id
HAVING round(sum(oi.order_item_subtotal::numeric), 2) >= 1000
ORDER BY o.order_date,
    revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73')),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'))]
```

```
%%sql

DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    user_country VARCHAR(2),
    is_active BOOLEAN DEFAULT FALSE,
    create_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_country)
VALUES ('Donald', 'Duck', 'donald@duck.com', 'IN')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_role, is_
↪active, user_country)
```

(continues on next page)

(continued from previous page)

```
VALUES ('Mickey', 'Mouse', 'mickey@mouse.com', 'U', true, 'US')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users
    (user_first_name, user_last_name, user_email_id, user_password, user_role, is_
↪active, user_country)
VALUES
    ('Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true, 'CA
↪'),
    ('Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true, 'FR'),
    ('Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true, 'AU')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users
ORDER BY user_country
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(5, 'Addie', 'Mesias', 'amesias2@twitpic.com', False, 'ih7Y69u56', 'U', 'AU', True,␣
↪datetime.datetime(2020, 11, 14, 15, 40, 12, 414932), datetime.datetime(2020, 11, 14,
↪ 15, 40, 12, 414932)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', False, 'h9LAz7p7ub', 'U',
↪'CA', True, datetime.datetime(2020, 11, 14, 15, 40, 12, 414932), datetime.
↪datetime(2020, 11, 14, 15, 40, 12, 414932)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', False, 'oEofndp', 'U', 'FR', True,␣
↪datetime.datetime(2020, 11, 14, 15, 40, 12, 414932), datetime.datetime(2020, 11, 14,
↪ 15, 40, 12, 414932)),
 (1, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', 'IN', False, datetime.
↪datetime(2020, 11, 14, 15, 40, 10, 878908), datetime.datetime(2020, 11, 14, 15, 40,␣
↪10, 878908)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', 'US', True, datetime.
↪datetime(2020, 11, 14, 15, 40, 11, 683887), datetime.datetime(2020, 11, 14, 15, 40,␣
↪11, 683887))]
```

```
%%sql

SELECT user_id,
    user_first_name,
    user_last_name,
    user_email_id,
```

(continues on next page)

(continued from previous page)

```
    user_country
FROM users
ORDER BY
    CASE WHEN user_country = 'US' THEN 0
        ELSE 1
    END, user_country
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(2, 'Mickey', 'Mouse', 'mickey@mouse.com', 'US'),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', 'AU'),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'CA'),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'FR'),
 (1, 'Donald', 'Duck', 'donald@duck.com', 'IN')]
```

6.3.11 Solution – Daily Product Revenue

Let us review the Final Solution for our problem statement **daily_product_revenue**.

- Prepare tables
  - Create tables
  - Load the data into tables
- We need to project the fields which we are interested in. We need to have **product_id** as well as **product_name** as there can be products with same name and can result in incorrect output.
  - order_date
  - order_item_product_id
  - product_name
  - product_revenue
- As we have fields from multiple tables, we need to perform join after which we have to filter for COMPLETE or CLOSED orders.
- We have to group the data by order_date and order_item_product_id, then we have to perform aggregation on order_item_subtotal to get product_revenue.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    p.product_name,
    round(sum(oi.order_item_subtotal::numeric), 2) AS product_revenue
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
    JOIN products p
        ON p.product_id = oi.order_item_product_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date,
    oi.order_item_product_id,
    p.product_name
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 24, 'Elevation Training Mask 2.0', Decimal(
↪'319.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, "Under Armour Men's Tech II T-Shirt",␣
↪Decimal('74.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 134, "Nike Women's Legend V-Neck T-Shirt",␣
↪Decimal('100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, "Nike Men's Free 5.0+ Running Shoe",␣
↪Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, 'Bowflex SelectTech 1090 Dumbbells',␣
↪Decimal('599.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, 'Perfect Fitness Perfect Rip Deck',␣
↪Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, "Nike Men's CJ Elite 2 TD Football Cleat
↪", Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, "Nike Men's Dri-FIT Victory Golf Polo",␣
↪Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 572, "TYR Boys' Team Digi Jammer", Decimal(
↪'119.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, "Nike Men's Kobe IX Elite Low Basketball␣
↪Shoe", Decimal('199.99'))]
```

```
%%sql

SELECT o.order_date,
    oi.order_item_product_id,
    p.product_name,
    round(sum(oi.order_item_subtotal::numeric), 2) AS product_revenue
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
    JOIN products p
        ON p.product_id = oi.order_item_product_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date,
    oi.order_item_product_id,
    p.product_name
```

(continues on next page)

(continued from previous page)

```
ORDER BY o.order_date,
    product_revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, 'Field & Stream Sportsman 16 Gun Fire␣
↪Safe', Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, "Nike Men's Free 5.0+ Running Shoe",␣
↪Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, "Diamondback Women's Serene Classic␣
↪Comfort Bi", Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, 'Perfect Fitness Perfect Rip Deck',␣
↪Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, 'Pelican Sunstream 100 Kayak', Decimal(
↪'2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, "O'Brien Men's Neoprene Life Vest",␣
↪Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, "Nike Men's CJ Elite 2 TD Football Cleat
↪", Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, "Nike Men's Dri-FIT Victory Golf Polo",␣
↪Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, "Under Armour Girls' Toddler Spine Surge␣
↪Runni", Decimal('1079.73')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, 'Bowflex SelectTech 1090 Dumbbells',␣
↪Decimal('599.99'))]
```

```
%%sql

SELECT count(1) FROM (
    SELECT o.order_date,
        oi.order_item_product_id,
        p.product_name,
        round(sum(oi.order_item_subtotal::numeric), 2) AS product_revenue
    FROM orders o
        JOIN order_items oi
            ON o.order_id = oi.order_item_order_id
        JOIN products p
            ON p.product_id = oi.order_item_product_id
    WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    GROUP BY o.order_date,
        oi.order_item_product_id,
        p.product_name
) q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9120,)]
```

6.3.12 Exercises - Basic SQL Queries

Here are some of the exercises for which you can write SQL queries to self evaluate.

- Ensure that we have required database and user for retail data. **We might provide the database as part of our labs.** Here are the instructions to use `psql` for setting up the required tables.

```
psql -U postgres -h localhost -p 5432 -W
```

```
CREATE DATABASE itversity_retail_db;
CREATE USER itversity_retail_user WITH ENCRYPTED PASSWORD 'retail_password';
GRANT ALL ON DATABASE itversity_retail_db TO itversity_retail_user;
```

- Create Tables using the script provided. You can either use `psql` or **SQL Workbench**.

```
psql -U itversity_retail_user \
  -h localhost \
  -p 5432 \
  -d itversity_retail_db \
  -W
```

- You can drop the existing tables.

```
DROP TABLE order_items;
DROP TABLE orders;
DROP TABLE customers;
DROP TABLE products;
DROP TABLE categories;
DROP TABLE departments;
```

- Once the tables are dropped you can run below script to create the tables for the purpose of exercises.

```
\i /data/retail_db/create_db_tables_pg.sql
```

- Data shall be loaded using the script provided.

```
\i /data/retail_db/load_db_tables_pg.sql
```

- Run queries to validate we have data in all the 3 tables.

Exercise 1 - Customer order count

Get order count per customer for the month of 2014 January.

- Tables - orders and customers
- Data should be sorted in descending order by count and ascending order by customer id.
- Output should contain customer_id, customer_first_name, customer_last_name and customer_order_count.

Exercise 2 - Dormant Customers

Get the customer details who have not placed any order for the month of 2014 January.

- Tables - orders and customers
- Data should be sorted in ascending order by customer_id
- Output should contain all the fields from customers

Exercise 3 - Revenue Per Customer

Get the revenue generated by each customer for the month of 2014 January

- Tables - orders, order_items and customers
- Data should be sorted in descending order by revenue and then ascending order by customer_id
- Output should contain customer_id, customer_first_name, customer_last_name, customer_revenue.
- If there are no orders placed by customer, then the corresponding revenue for a give customer should be 0.
- Consider only COMPLETE and CLOSED orders

Exercise 4 - Revenue Per Category

Get the revenue generated for each category for the month of 2014 January

- Tables - orders, order_items, products and categories
- Data should be sorted in ascending order by category_id.
- Output should contain all the fields from category along with the revenue as category_revenue.
- Consider only COMPLETE and CLOSED orders

Exercise 5 - Product Count Per Department

Get the products for each department.

- Tables - departments, categories, products
- Data should be sorted in ascending order by department_id
- Output should contain all the fields from department and the product count as product_count

6.4 Creating Tables and Indexes

Let us go through the details related to creating tables and indexes. We will also talk about how columns, constraints etc while going through the details related to tables and indexes.

- DDL - Data Definition Language
- Overview of Data Types
- Adding or Modifying Columns
- Different Types of Constraints
- Managing Constraints

- Indexes on Tables
- Indexes for Constraints
- Overview of Sequences
- Truncating Tables
- Dropping Tables
- Exercise - Managing Database Objects

Here are the key objectives of this section:

- How to create and manage tables?
- Get in depth understanding about columns and commonly used data types
- What are different types of constraints and how they are managed?
- What are indexes and how they are relevant to Prmary Key, Unique and Foreign Key constraints?
- What is a Sequence and how sequences are used to populate Surrogate Keys?
- Self evaluate whether one understood all the key aspects of managing tables and constraints.

6.4.1 DDL – Data Definition Language

Let us get an overview of DDL Statements which are typically used to create database objects such as tables.

- DDL Stands for Data Definition Language.
- We execute DDL statements less frequently as part of the application development process.
- Typically DDL Scripts are maintained separately than the code.
- Following are the common DDL tasks.
  - Creating Tables - Independent Objects
  - Creating Indexes for performance - Typically dependent on tables
  - Adding constraints to existing tables (`NOT NULL`, `CHECK`, `PRIMARY KEY`, `UNIQUE` etc)

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- Following are less common DDL tasks which can be taken care using `ALTER` command.
  - Adding columns to existing tables.
  - Dropping columns from existing tables.
  - Changing data types of existing columns.

- We can also define comments both at column level as well as table level. However in **postgres**, we can only add comments after table is created.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS users
```

```
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql COMMENT ON TABLE users IS 'Stores all user details'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql COMMENT ON COLUMN users.user_id IS 'Surrogate Key'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql COMMENT ON COLUMN users.user_first_name IS 'User First Name'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql COMMENT ON COLUMN users.user_role IS 'U for user A for admin'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'public', 'users', 'BASE TABLE', None, None, None, None,␣
↪None, 'YES', 'NO', None)]
```

```
%%sql

SELECT * FROM information_schema.columns
WHERE table_name = 'users'
ORDER BY ordinal_position
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('itversity_retail_db', 'public', 'users', 'user_id', 1, "nextval('users_user_id_seq
↪'::regclass)", 'NO', 'integer', None, None, 32, 2, 0, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'int4
↪', None, None, None, None, '1', 'NO', 'NO', None, None, None, None, None, 'NO',
↪'NEVER', None, 'YES'),
 ('itversity_retail_db', 'public', 'users', 'user_first_name', 2, None, 'NO',
↪'character varying', 30, 120, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'varchar',␣
↪None, None, None, None, '2', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES'),
 ('itversity_retail_db', 'public', 'users', 'user_last_name', 3, None, 'NO',
↪'character varying', 30, 120, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'varchar',␣
↪None, None, None, None, '3', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES'),
 ('itversity_retail_db', 'public', 'users', 'user_email_id', 4, None, 'NO',
↪'character varying', 50, 200, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'varchar',␣
↪None, None, None, None, '4', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES'),
```

(continues on next page)

(continued from previous page)

```
('itversity_retail_db', 'public', 'users', 'user_email_validated', 5, 'false', 'YES',
↪ 'boolean', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'bool', None,␣
↪None, None, None, '5', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER',␣
↪None, 'YES'),
('itversity_retail_db', 'public', 'users', 'user_password', 6, None, 'YES',
↪'character varying', 200, 800, None, None, None, None, None, None, None, None, None,
↪ None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'varchar',
↪ None, None, None, None, '6', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER
↪', None, 'YES'),
('itversity_retail_db', 'public', 'users', 'user_role', 7, "'U'::character varying",
↪'NO', 'character varying', 1, 4, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog',
↪'varchar', None, None, None, None, '7', 'NO', 'NO', None, None, None, None, None,
↪'NO', 'NEVER', None, 'YES'),
('itversity_retail_db', 'public', 'users', 'is_active', 8, 'false', 'YES', 'boolean',
↪ None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪ None, None, None, 'itversity_retail_db', 'pg_catalog', 'bool', None, None, None,␣
↪None, '8', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES'),
('itversity_retail_db', 'public', 'users', 'created_dt', 9, 'CURRENT_DATE', 'YES',
↪'date', None, None, None, None, None, 0, None, None, None, None, None, None, None,␣
↪None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'date', None, None,␣
↪None, None, '9', 'NO', 'NO', None, None, None, None, None, 'NO', 'NEVER', None, 'YES
↪'),
('itversity_retail_db', 'public', 'users', 'last_updated_ts', 10, 'CURRENT_TIMESTAMP
↪', 'YES', 'timestamp without time zone', None, None, None, None, None, 6, None,␣
↪None, None, None, None, None, None, None, None, None, 'itversity_retail_db',
↪'pg_catalog', 'timestamp', None, None, None, None, '10', 'NO', 'NO', None, None,␣
↪None, None, None, 'NO', 'NEVER', None, 'YES')]
```

6.4.2 Overview of Data Types

Let us get an overview of supported datatypes in Postgres.

- Here is the sample `CREATE TABLE` command for the review.

```sql
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- While creating tables in RDBMS databases, we should specify data types for the columns.
  - `SERIAL` is nothing but integer which is populated by a special database object called as sequence. It is typically used for surrogate primary key.
  - When `SERIAL` is specified, a sequence with **table_name_serial_column_seq** naming convention will be created. In our case it is `users_user_id_seq`.

  - INT or INTEGER is used to define columns with integer values. Most of the ids are defined as integer.
  - FLOAT or DOUBLE can be used to define columns used to store price, salary etc.
  - VARCHAR with length is used to define variable length columns such as name, email id etc.
  - CHAR can be used to define fixed length string columns - single character fields such as gender which store M or F, three character days or months etc.
  - BOOLEAN is used to store **true** and **false** values.
  - We can also use DATE or TIMESTAMP to store date or time respectively.
- We can add columns, drop columns, modify columns by changing data types as well as specify default values using ALTER TABLE command.
- Let us perform these tasks to understand about Data Types. Drop and recreate users table with the following details.
  - user_id - integer
  - user_first_name - not null and alpha numeric or string up to 30 characters
  - user_last_name - not null and alpha numeric or string up to 30 characters
  - user_email_id - not null and alpha numeric or string up to 50 characters
  - user_email_validated - true or false (boolean)
  - user_password - alpha numeric up to 200 characters
  - user_role - single character with U or A (for now we will use VARCHAR(1))
  - is_active - true or false (boolean)
  - created_dt - not null and date with out timestamp. It should be defaulted to system date.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
  user_id INT,
  user_first_name VARCHAR(30) NOT NULL,
  user_last_name VARCHAR(30) NOT NULL,
  user_email_id VARCHAR(50) NOT NULL,
  user_email_validated BOOLEAN,
  user_password VARCHAR(200),
```

(continues on next page)

(continued from previous page)

```
  user_role VARCHAR(1),
  is_active BOOLEAN,
  created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    column_name,
    data_type,
    character_maximum_length,
    column_default,
    is_nullable,
    ordinal_position
FROM information_schema.columns
WHERE table_name = 'users'
ORDER BY ordinal_position
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9 rows affected.
```

```
[('itversity_retail_db', 'users', 'user_id', 'integer', None, None, 'YES', 1),
 ('itversity_retail_db', 'users', 'user_first_name', 'character varying', 30, None,
↪'NO', 2),
 ('itversity_retail_db', 'users', 'user_last_name', 'character varying', 30, None, 'NO
↪', 3),
 ('itversity_retail_db', 'users', 'user_email_id', 'character varying', 50, None, 'NO
↪', 4),
 ('itversity_retail_db', 'users', 'user_email_validated', 'boolean', None, None, 'YES
↪', 5),
 ('itversity_retail_db', 'users', 'user_password', 'character varying', 200, None,
↪'YES', 6),
 ('itversity_retail_db', 'users', 'user_role', 'character varying', 1, None, 'YES',␣
↪7),
 ('itversity_retail_db', 'users', 'is_active', 'boolean', None, None, 'YES', 8),
 ('itversity_retail_db', 'users', 'created_dt', 'date', None, 'CURRENT_DATE', 'YES',␣
↪9)]
```

6.4.3 Different Types of Constraints

Let us understand details about different types of constraints used in RDBMS databases.

- Supported constraints:
  - NOT NULL constraint
  - CHECK constraint
  - UNIQUE constraint
  - PRIMARY KEY constraint
  - FOREIGN KEY constraint
- All constraints can be added while creating the table or on pre-created tables using `ALTER`.
- Typically we define `NOT NULL`, `CHECK` constraints while creating the tables. However, we can also specify **not null constraints** as well as **check constraints** to the columns while adding columns using `ALTER TABLE`.
- `FOREIGN KEY` constraints are created after the tables are created. It is primarily used to define relationship between 2 tables - example: users is parent table and user_login_details is child table with one to many relationship between them.
- `PRIMARY KEY` and `UNIQUE` constraints might be added as part of CREATE table statements or ALTER table statements. Both are commonly used practices.
- Let us compare and contrast `PRIMARY KEY` and `UNIQUE` constraints.
  - There can be only one `PRIMARY KEY` in a table where as there can be any number of `UNIQUE` constraints.
  - `UNIQUE` columns can have null values unless `NOT NULL` is also enforced. In case of `PRIMARY KEY`, both uniqueness as well as not null are strictly enforced. In other words a primary key column cannot be null where as unique column can be null.
  - `FOREIGN KEY` from a child table can be defined against `PRIMARY KEY` column or `UNIQUE` column.
  - Typically `PRIMARY KEY` columns are surrogate keys which are supported by sequence.
  - `PRIMARY KEY` or `UNIQUE` can be composite. It means there can be more than one column to define `PRIMARY KEY` or `UNIQUE` constraint.
- Let's take an example of LMS (Learning Management System).
  - **USERS** - it contains columns such as user_id, user_email_id, user_first_name etc. We can enforce primary key constraint on user_id and unique constraint on user_email_id.
  - **COURSES** - it contains columns such as course_id, course_name, course_price etc. Primary key constraint will be enforced on course_id.
  - **STUDENTS** - A student is nothing but a user who is enrolled for one or more courses. But he can enroll for one course only once.
    - It contains fields such as student_id, user_id, course_id, amount_paid, enrolled_dt etc.
    - Primary key constraint will be enforced on student_id.
    - A foreign key constraint can be enforced on students.user_id against users.user_id.
    - Another foreign key constraint can be enforced on students.course_id against courses.course_id.
    - Also we can have unique constraint enforced on students.user_id and students.course_id. It will be composite key as it have more than one column.

6.4.4 Managing Constraints

Let us understand how we can manage constraints.

- We can add constraints while creating the tables or after creating the tables.
- Constraints such as NOT NULL, CHECK, FOREIGN KEY are automatically dropped when we drop the table.
- Even PRIMARY KEY and UNIQUE constraints are dropped if they are not used to enforce constraints. When PRIMARY KEY or UNIQUE constraint is referred by child table then there can be errors.
- We can add constraints to existing table using `ALTER TABLE` with `ADD`. We can specify the name using `CONSTRAINT` keyword.
- Constraints from the table can be dropped using `ALTER TABLE` with `DROP`.
- Let us perform tasks to understand how we can use `ALTER TABLE` command to add or drop the constraints.
  - Use the prior users table with out any constraints.
  - Add primary key constraint on user_id.
  - Add unique constraint on user_email_id.
  - Add not null constraints user_email_validated, user_role, created_dt, last_updated_ts
  - Add check constraint to user_role with 'U' and 'A' as accepted values.
  - Add new table user_logins with below columns and establish foreign key relationship with users.
    * user_login_id - `SERIAL` and `PRIMARY KEY`
    * user_id - `INT`
    * user_login_time - `TIMESTAMP` defaulted to `CURRENT_TIMESTAMP`
    * **user_logins** is child table to **users** with many to one relationship. Hence, create **foreign key** between **user_logins.user_id** to **users.user_id**.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP SEQUENCE IF EXISTS users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id INT,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN,
    user_password VARCHAR(200),
    user_role VARCHAR(1),
    is_active BOOLEAN,
    created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql CREATE SEQUENCE users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql ALTER TABLE users ALTER COLUMN user_id SET DEFAULT nextval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users
    ALTER COLUMN user_email_validated SET DEFAULT FALSE,
    ALTER COLUMN is_active SET DEFAULT FALSE
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users
    ALTER COLUMN user_role SET DATA TYPE CHAR(1),
    ALTER COLUMN user_role SET DEFAULT 'U'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users
    ADD COLUMN last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[('itversity_retail_db', 'users', 'CHECK', '2200_17328_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_4_not_null')]
```

```
%sql ALTER TABLE users ADD PRIMARY KEY (user_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pkey'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_1_not_null'),
```

(continues on next page)

(continued from previous page)

```
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_4_not_null')]
```

```
%sql ALTER TABLE users DROP CONSTRAINT users_pkey
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
4 rows affected.
```

```
[('itversity_retail_db', 'users', 'CHECK', '2200_17328_1_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_4_not_null')]
```

```
%sql ALTER TABLE users ADD CONSTRAINT users_pk PRIMARY KEY (user_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pk'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_1_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_4_not_null')]
```

```
%sql ALTER TABLE users ADD UNIQUE (user_email_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pk'),
 ('itversity_retail_db', 'users', 'UNIQUE', 'users_user_email_id_key'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_1_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_4_not_null')]
```

```
%%sql

ALTER TABLE users
    ALTER COLUMN user_email_validated SET NOT NULL,
    ALTER COLUMN user_role SET NOT NULL,
    ALTER COLUMN created_dt SET NOT NULL,
    ALTER COLUMN last_updated_ts SET NOT NULL
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users
    ADD CHECK (user_role IN ('U', 'A') )
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
```

(continues on next page)

(continued from previous page)

```
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
11 rows affected.
```

```
[('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pk'),
 ('itversity_retail_db', 'users', 'UNIQUE', 'users_user_email_id_key'),
 ('itversity_retail_db', 'users', 'CHECK', 'users_user_role_check'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_1_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_4_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_5_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_7_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_9_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17328_10_not_null')]
```

```
%%sql

CREATE TABLE user_logins (
    user_login_id SERIAL PRIMARY KEY,
    user_id INT,
    user_login_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_ip_addr VARCHAR(20)
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'user_logins'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[('itversity_retail_db', 'user_logins', 'PRIMARY KEY', 'user_logins_pkey'),
 ('itversity_retail_db', 'user_logins', 'CHECK', '2200_17351_1_not_null')]
```

```
%%sql

ALTER TABLE user_logins
    ADD FOREIGN KEY (user_id)
    REFERENCES users(user_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'user_logins'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[('itversity_retail_db', 'user_logins', 'PRIMARY KEY', 'user_logins_pkey'),
 ('itversity_retail_db', 'user_logins', 'FOREIGN KEY', 'user_logins_user_id_fkey'),
 ('itversity_retail_db', 'user_logins', 'CHECK', '2200_17351_1_not_null')]
```

Error: This will fail as there is a child table user_logins for users table.

```
%%sql

DROP TABLE users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
```

```
---------------------------------------------------------------------------
DependentObjectsStillExist                Traceback (most recent call last)
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594

DependentObjectsStillExist: cannot drop table users because other objects depend on it
DETAIL:  constraint user_logins_user_id_fkey on table user_logins depends on table␣
↪users
HINT:  Use DROP ... CASCADE to drop the dependent objects too.


The above exception was the direct cause of the following exception:

InternalError                             Traceback (most recent call last)
```

(continues on next page)

(continued from previous page)

```
<ipython-input-60-a1fbf34721c3> in <module>
----> 1 get_ipython().run_cell_magic('sql', '', '\nDROP TABLE users\n')

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/interactiveshell.
↪py in run_cell_magic(self, magic_name, line, cell)
   2369             with self.builtin_trap:
   2370                 args = (magic_arg_s, cell)
-> 2371                 result = fn(*args, **kwargs)
   2372             return result
   2373 

<decorator-gen-135> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

<decorator-gen-134> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/magic.py in execute(self,␣
↪line, cell, local_ns)
    215 
    216         try:
--> 217             result = sql.run.run(conn, parsed["sql"], self, user_ns)
    218 
    219             if (

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/run.py in run(conn, sql,␣
↪config, user_namespace)
    365             else:
    366                 txt = sqlalchemy.sql.text(statement)
--> 367                 result = conn.session.execute(txt, user_namespace)
    368             _commit(conn=conn, config=config)
    369             if result and config.feedback:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in␣
↪execute(self, object_, *multiparams, **params)
   1009             )
   1010         else:
-> 1011             return meth(self, multiparams, params)
   1012 
   1013     def _execute_function(self, func, multiparams, params):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/sql/elements.py in␣
↪_execute_on_connection(self, connection, multiparams, params)
```

(continues on next page)

(continued from previous page)

```
    296     def _execute_on_connection(self, connection, multiparams, params):
    297         if self.supports_execution:
--> 298             return connection._execute_clauseelement(self, multiparams, params)
    299         else:
    300             raise exc.ObjectNotExecutableError(self)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _execute_clauseelement(self, elem, multiparams, params)
   1128                 distilled_params,
   1129                 compiled_sql,
-> 1130                 distilled_params,
   1131             )
   1132         if self._has_events or self.engine._has_events:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _execute_context(self, dialect, constructor, statement, parameters, *args)
   1315         except BaseException as e:
   1316             self._handle_dbapi_exception(
-> 1317                 e, statement, parameters, cursor, context
   1318             )
   1319 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1509             elif should_wrap:
   1510                 util.raise_(
-> 1511                     sqlalchemy_exception, with_traceback=exc_info[2], from_=e
   1512                 )
   1513             else:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/util/compat.py in raise_(***failed resolving arguments***)
    180 
    181         try:
--> 182             raise exception
    183         finally:
    184             # credit to

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _execute_context(self, dialect, constructor, statement, parameters, *args)
   1275                 if not evt_handled:
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )
   1279 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py in do_execute(self, cursor, statement, parameters, context)
    591 
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 
    595     def do_execute_no_params(self, cursor, statement, context=None):

InternalError: (psycopg2.errors.DependentObjectsStillExist) cannot drop table users because other objects depend on it
```

(continues on next page)

(continued from previous page)

```
DETAIL:  constraint user_logins_user_id_fkey on table user_logins depends on table␣
↪users
HINT:  Use DROP ... CASCADE to drop the dependent objects too.

[SQL: DROP TABLE users]
(Background on this error at: http://sqlalche.me/e/13/2j85)
```

Note: You can use `CASCADE` to drop foreign key constraints from child tables before dropping the table users.

```
%%sql

DROP TABLE users CASCADE
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'user_logins'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[('itversity_retail_db', 'user_logins', 'PRIMARY KEY', 'user_logins_pkey'),
 ('itversity_retail_db', 'user_logins', 'CHECK', '2200_17351_1_not_null')]
```

```
%sql DROP TABLE IF EXISTS user_logins
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

6.4.5 Indexes on Tables

Let us go through the details related to indexes supported in RDBMS such as Postgres.

- An index can be unique or non unique.
- Unique Index - Data will be sorted in ascending order and uniqueness is enforced.
- Non Unique Index - Data will be sorted in ascending order and uniqueness is not enforced.
- Unless specified all indexes are of type B Tree.

- For sparsely populated columns, we tend to create B Tree indexes. B Tree indexes are the most commonly used ones.
- For densely populated columns such as gender, month etc with very few distinct values we can leverage bit map index. However bitmap indexes are not used quite extensively in typical web or mobile applications.
- Write operations will become relatively slow as data have to be managed in index as well as table.
- We need to be careful while creating indexes on the tables as write operations can become slow as more indexes are added to the table.
- Here are some of the criteria for creating indexes.
  - Create unique indexes when you want to enforce uniqueness. If you define unique constraint or primary key constraint, it will create unique index internally.
  - If we are performing joins between 2 tables based on a value, then the foreign key column in the child table should be indexed.
    * Typically as part of order management system, we tend to get all the order details for a given order using order id.
    * In our case we will be able to improve the query performance by adding index on **order_items.order_item_order_id**.
    * However, write operation will become a bit slow. But it is acceptable and required to create index on **order_items.order_item_order_id** as we write once and read many times over the life of the order.
- Let us perform tasks related to indexes.
  - Drop and recreate retail db tables.
  - Load data into retail db tables.
  - Compute statistics (Optional). It is typically taken care automatically by the schedules defined by DBAs.
  - Use code to randomly fetch 2000 orders and join with order_items - compute time.
  - Create index for order_items.order_item_order_id and compute statistics
  - Use code to randomly fetch 2000 orders and join with order_items - compute time.
- Script to create tables and load data in case there are no tables in retail database.

```
psql -U itversity_retail_user \
  -h localhost \
  -p 5432 \
  -d itversity_retail_db \
  -W

DROP TABLE order_items;
DROP TABLE orders;
DROP TABLE products;
DROP TABLE categories;
DROP TABLE departments;
DROP TABLE customers;

\i /data/retail_db/create_db_tables_pg.sql
\i /data/retail_db/load_db_tables_pg.sql
```

```
!pip install psycopg2
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: psycopg2 in /opt/anaconda3/envs/beakerx/lib/python3.6/
↪site-packages (2.8.6)
```

```
import psycopg2
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT *
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    order_id = randrange(1, 68883)
    cursor.execute(query, (order_id,))
    ctr += 1
cursor.close()
connection.close()
```

```
CPU times: user 73.8 ms, sys: 31.4 ms, total: 105 ms
Wall time: 19.6 s
```

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

CREATE INDEX order_items_oid_idx
ON order_items(order_item_order_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT *
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    order_id = randrange(1, 68883)
    cursor.execute(query, (order_id,))
    ctr += 1
cursor.close()
connection.close()
```

```
CPU times: user 49.1 ms, sys: 32.9 ms, total: 82 ms
Wall time: 265 ms
```

6.4.6 Indexes for Constraints

Let us understand details related to indexes for constraints.

- Constraints such as primary key and unique are supported by indexes.
- **Primary Key** - Unique and Not Null.
- **Unique** - Unique and can be null.
- Unless data is sorted, we need to perform full table scan to enforce uniqueness. Almost all the databases will create indexes implicitly for Primary Keys as well as Unique constraints.
- We cannot define Primary Key or Unique constraint with out associated index.
- It is quite common that we explicitly create indexes on foreign key columns to improve the performance.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP SEQUENCE IF EXISTS users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id INT,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN,
    user_password VARCHAR(200),
    user_role VARCHAR(1),
    is_active BOOLEAN,
    created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[('itversity_retail_db', 'users', 'CHECK', '2200_17365_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_4_not_null')]
```

```
%%sql

SELECT * FROM pg_catalog.pg_indexes
```

(continues on next page)

(continued from previous page)

```
WHERE schemaname = 'public'
    AND tablename = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%sql CREATE SEQUENCE users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users
    ALTER COLUMN user_id SET DEFAULT nextval('users_user_id_seq'),
    ADD PRIMARY KEY (user_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pkey'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_1_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_4_not_null')]
```

```
%%sql

SELECT * FROM pg_catalog.pg_indexes
WHERE schemaname = 'public'
    AND tablename = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('public', 'users', 'users_pkey', None, 'CREATE UNIQUE INDEX users_pkey ON public.
↪users USING btree (user_id)')]
```

```
%%sql

SELECT tc.table_catalog,
    tc.table_name,
    tc.constraint_name,
    pi.indexname
FROM information_schema.table_constraints tc JOIN pg_catalog.pg_indexes pi
    ON tc.constraint_name = pi.indexname
WHERE tc.table_schema = 'public'
    AND tc.table_name = 'users'
    AND tc.constraint_type = 'PRIMARY KEY'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'users', 'users_pkey', 'users_pkey')]
```

```
%%sql

ALTER TABLE users
    ADD UNIQUE (user_email_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_name,
    constraint_type,
    constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pkey'),
 ('itversity_retail_db', 'users', 'UNIQUE', 'users_user_email_id_key'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_1_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_2_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_3_not_null'),
 ('itversity_retail_db', 'users', 'CHECK', '2200_17365_4_not_null')]
```

```
%%sql

SELECT * FROM pg_catalog.pg_indexes
WHERE schemaname = 'public'
    AND tablename = 'users'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[('public', 'users', 'users_pkey', None, 'CREATE UNIQUE INDEX users_pkey ON public.
↪users USING btree (user_id)'),
 ('public', 'users', 'users_user_email_id_key', None, 'CREATE UNIQUE INDEX users_user_
↪email_id_key ON public.users USING btree (user_email_id)')]
```

```
%%sql

SELECT tc.table_catalog,
    tc.table_name,
    tc.constraint_name,
    pi.indexname
FROM information_schema.table_constraints tc JOIN pg_catalog.pg_indexes pi
    ON tc.constraint_name = pi.indexname
WHERE tc.table_schema = 'public'
    AND tc.table_name = 'users'
    AND tc.constraint_type = 'UNIQUE'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'users', 'users_user_email_id_key', 'users_user_email_id_key
↪')]
```

Note: Query to get all the primary key and unique constraints along with indexes.

```
%%sql

SELECT tc.table_catalog,
    tc.table_name,
    tc.constraint_type,
    tc.constraint_name,
    pi.indexname
FROM information_schema.table_constraints tc JOIN pg_catalog.pg_indexes pi
    ON tc.constraint_name = pi.indexname
WHERE tc.table_catalog = 'itversity_retail_db'
    AND tc.constraint_type IN ('PRIMARY KEY', 'UNIQUE')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
8 rows affected.
```

```
[('itversity_retail_db', 'departments', 'PRIMARY KEY', 'departments_pkey',
↪'departments_pkey'),
 ('itversity_retail_db', 'categories', 'PRIMARY KEY', 'categories_pkey', 'categories_
↪pkey'),
 ('itversity_retail_db', 'products', 'PRIMARY KEY', 'products_pkey', 'products_pkey'),
 ('itversity_retail_db', 'customers', 'PRIMARY KEY', 'customers_pkey', 'customers_pkey
↪'),
 ('itversity_retail_db', 'orders', 'PRIMARY KEY', 'orders_pkey', 'orders_pkey'),
 ('itversity_retail_db', 'order_items', 'PRIMARY KEY', 'order_items_pkey', 'order_
↪items_pkey'),
```

(continues on next page)

(continued from previous page)

```
('itversity_retail_db', 'users', 'PRIMARY KEY', 'users_pkey', 'users_pkey'),
('itversity_retail_db', 'users', 'UNIQUE', 'users_user_email_id_key', 'users_user_
↪email_id_key')]
```

Error: It is not possible to drop the indexes that are automatically created to enforce primary key or unique constraints.

```
%sql DROP INDEX users_user_email_id_key
```

```
* postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
```

```
---------------------------------------------------------------------------
DependentObjectsStillExist                Traceback (most recent call last)
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 

DependentObjectsStillExist: cannot drop index users_user_email_id_key because␣
↪constraint users_user_email_id_key on table users requires it
HINT:  You can drop constraint users_user_email_id_key on table users instead.


The above exception was the direct cause of the following exception:

InternalError                             Traceback (most recent call last)
<ipython-input-89-7b38c07068a1> in <module>
----> 1 get_ipython().run_line_magic('sql', 'DROP INDEX users_user_email_id_key')

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/interactiveshell.
↪py in run_line_magic(self, magic_name, line, _stack_depth)
   2324                 kwargs['local_ns'] = sys._getframe(stack_depth).f_locals
   2325             with self.builtin_trap:
-> 2326                 result = fn(*args, **kwargs)
   2327             return result
   2328 

<decorator-gen-135> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):
```

(continues on next page)

(continued from previous page)

```
<decorator-gen-134> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/magic.py in execute(self,␣
↪line, cell, local_ns)
    215 
    216         try:
--> 217             result = sql.run.run(conn, parsed["sql"], self, user_ns)
    218 
    219             if (

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/run.py in run(conn, sql,␣
↪config, user_namespace)
    365             else:
    366                 txt = sqlalchemy.sql.text(statement)
--> 367                 result = conn.session.execute(txt, user_namespace)
    368             _commit(conn=conn, config=config)
    369             if result and config.feedback:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in␣
↪execute(self, object_, *multiparams, **params)
   1009             )
   1010         else:
-> 1011             return meth(self, multiparams, params)
   1012 
   1013     def _execute_function(self, func, multiparams, params):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/sql/elements.py in␣
↪_execute_on_connection(self, connection, multiparams, params)
    296     def _execute_on_connection(self, connection, multiparams, params):
    297         if self.supports_execution:
--> 298             return connection._execute_clauseelement(self, multiparams,␣
↪params)
    299         else:
    300             raise exc.ObjectNotExecutableError(self)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_clauseelement(self, elem, multiparams, params)
   1128             distilled_params,
   1129             compiled_sql,
-> 1130             distilled_params,
   1131         )
   1132         if self._has_events or self.engine._has_events:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1315         except BaseException as e:
   1316             self._handle_dbapi_exception(
-> 1317                 e, statement, parameters, cursor, context
```

(continues on next page)

(continued from previous page)

```
   1318             )
   1319 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1509             elif should_wrap:
   1510                 util.raise_(
-> 1511                     sqlalchemy_exception, with_traceback=exc_info[2], from_=e
   1512                 )
   1513             else:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/util/compat.py in␣
↪raise_(***failed resolving arguments***)
    180 
    181         try:
--> 182             raise exception
    183         finally:
    184             # credit to

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1275                 if not evt_handled:
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )
   1279 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    591 
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 
    595     def do_execute_no_params(self, cursor, statement, context=None):

InternalError: (psycopg2.errors.DependentObjectsStillExist) cannot drop index users_
↪user_email_id_key because constraint users_user_email_id_key on table users␣
↪requires it
HINT:  You can drop constraint users_user_email_id_key on table users instead.

[SQL: DROP INDEX users_user_email_id_key]
(Background on this error at: http://sqlalche.me/e/13/2j85)
```

6.4.7 Overview of Sequences

Let us go through some of the important details related to sequences.

- For almost all the tables in relational databases we define primary key constraints.
- Primary key is nothing but unique constraint with not null and there can be only one primary key in any given table.
- Many times, we might not have appropriate column in the table which can be used as primary key. In those scenarios we will define a column which does not have any business relevant values. This is called as **surrogate key**.
- Relational Database technologies provide sequences to support these **surrogate primary keys**.

- In postgres we can define **surrogate primary key** for a given table as `SERIAL`. Internally it will create a sequence.
- We can also pre-create a sequence and use it to populate multiple tables.
- Even if we do not specify the column and value as part of the insert statement, a sequence generated number will be populated in that column.
- Typically, the sequence generated number will be incremented by 1. We can change it by specifying a constant value using `INCREMENT BY`.
- Here are some of the properties that can be set for a sequence. Most of them are self explanatory.
  - `START WITH`
  - `RESTART WITH`
  - `MINVALUE`
  - `MAXVALUE`
  - `CACHE`
- We can use functions such as `nextval` and `currval` to explicitly generate sequence numbers and also to get current sequence number in the current session.
- We might have to use `RESTART WITH` to reset the sequences after the underlying tables are populated with values in surrogate key.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Note: Let us create a sequence which start with 101 with minimum value 101 and maximum value 1000.

```
%%sql

DROP SEQUENCE IF EXISTS test_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE SEQUENCE test_seq
START WITH 101
MINVALUE 101
MAXVALUE 1000
INCREMENT BY 100
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql SELECT currval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.ObjectNotInPrerequisiteState) currval of sequence "test_seq" is not
↪yet defined in this session

[SQL: SELECT currval('test_seq')]
(Background on this error at: http://sqlalche.me/e/13/e3q8)
```

```
%sql SELECT nextval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(101,)]
```

```
%sql SELECT currval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(101,)]
```

```
%sql SELECT nextval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(201,)]
```

```
%sql SELECT currval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(201,)]
```

```
%sql SELECT nextval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(301,)]
```

```
%sql SELECT currval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(301,)]
```

```
%%sql

ALTER SEQUENCE test_seq
INCREMENT BY 1
RESTART WITH 101
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql SELECT nextval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(101,)]
```

```
%sql SELECT currval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(101,)]
```

```
%sql SELECT nextval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(102,)]
```

```
%sql SELECT currval('test_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(102,)]
```

```
%sql DROP SEQUENCE test_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

Note: `SERIAL` will make sure user_id is populated using sequence and `PRIMARY KEY` will enforce not null and unique constraints.

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP SEQUENCE IF EXISTS users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN,
    user_password VARCHAR(200),
    user_role VARCHAR(1),
    is_active BOOLEAN,
    created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT * FROM information_schema.sequences
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'public', 'users_user_id_seq', 'integer', 32, 2, 0, '1', '1',
↪ '2147483647', '1', 'NO')]
```

```
%sql SELECT nextval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Donald', 'Duck', 'donald@duck.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,
↪11, 23))]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_role, is_
↪active)
VALUES ('Mickey', 'Mouse', 'mickey@mouse.com', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[(2, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23))]
```

```
%%sql

INSERT INTO users
    (user_first_name, user_last_name, user_email_id, user_password, user_role, is_
↪active)
VALUES
    ('Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true),
    ('Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true),
    ('Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(6,)]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(2, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23)),
 (4, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', None, 'h9LAz7p7ub', 'U',␣
↪True, datetime.date(2020, 11, 23)),
 (5, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', None, 'oEofndp', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (6, 'Addie', 'Mesias', 'amesias2@twitpic.com', None, 'ih7Y69u56', 'U', True,␣
↪datetime.date(2020, 11, 23))]
```

Warning: It is not a good idea to populate surrogate key fields by passing the values. Either we should specify sequence generated number or let database take care of populating the field.

```
%%sql

INSERT INTO users (user_id, user_first_name, user_last_name, user_email_id)
VALUES (7, 'Scott', 'Tiger', 'scott@tiger.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(6,)]
```

Note: When data is loaded with surrogate key values into the table from external sources, it is recommended to create sequence with maximum + 1 value using`START WITH`

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP SEQUENCE IF EXISTS users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

Note: `SERIAL` will make sure user_id is populated using sequence and `PRIMARY KEY` will enforce not null and unique constraints.

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN,
    user_password VARCHAR(200),
    user_role VARCHAR(1),
    is_active BOOLEAN,
    created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[ ]
```

```
%%sql

INSERT INTO users (user_id, user_first_name, user_last_name, user_email_id)
VALUES (1, 'Donald', 'Duck', 'donald@duck.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[ ]
```

```
%%sql

INSERT INTO users (user_id, user_first_name, user_last_name, user_email_id, user_role,
↪ is_active)
VALUES (2, 'Mickey', 'Mouse', 'mickey@mouse.com', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[ ]
```

```
%%sql

INSERT INTO users
    (user_id, user_first_name, user_last_name, user_email_id, user_password, user_
↪role, is_active)
VALUES
    (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true),
    (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true),
    (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[ ]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', None, 'h9LAz7p7ub', 'U',␣
↪True, datetime.date(2020, 11, 23)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', None, 'oEofndp', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', None, 'ih7Y69u56', 'U', True,␣
↪datetime.date(2020, 11, 23))]
```

```
%sql SELECT nextval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%sql ALTER SEQUENCE users_user_id_seq RESTART WITH 5
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%sql SELECT nextval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(5,)]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(5,)]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Scott', 'Tiger', 'scott@tiger.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%sql SELECT currval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(6,)]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', None, 'h9LAz7p7ub', 'U',␣
↪True, datetime.date(2020, 11, 23)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', None, 'oEofndp', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', None, 'ih7Y69u56', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (6, 'Scott', 'Tiger', 'scott@tiger.com', None, None, None, None, datetime.date(2020,␣
↪11, 23))]
```

```
%sql DROP SEQUENCE users_user_id_seq CASCADE
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', None, 'h9LAz7p7ub', 'U',␣
↪True, datetime.date(2020, 11, 23)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', None, 'oEofndp', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', None, 'ih7Y69u56', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (6, 'Scott', 'Tiger', 'scott@tiger.com', None, None, None, None, datetime.date(2020,␣
↪11, 23))]
```

```
%%sql

CREATE SEQUENCE users_user_id_seq
    START WITH 7
    MINVALUE 1
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER SEQUENCE users_user_id_seq
    OWNED BY users.user_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users
    ALTER COLUMN user_id
    SET DEFAULT nextval('users_user_id_seq')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Matt', 'Clarke', 'matt@clarke.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
7 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,
↪11, 23)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', None, 'h9LAz7p7ub', 'U',
↪True, datetime.date(2020, 11, 23)),
```

(continues on next page)

(continued from previous page)

```
(4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', None, 'oEofndp', 'U', True,␣
↪datetime.date(2020, 11, 23)),
(5, 'Addie', 'Mesias', 'amesias2@twitpic.com', None, 'ih7Y69u56', 'U', True,␣
↪datetime.date(2020, 11, 23)),
(6, 'Scott', 'Tiger', 'scott@tiger.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
(7, 'Matt', 'Clarke', 'matt@clarke.com', None, None, None, None, datetime.date(2020,␣
↪11, 23))]
```

```
%sql SELECT curval('users_user_id_seq')
```

```
* postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(7,)]
```

6.4.8 Truncating Tables

Let us understand details related to truncating tables.

- If you want to delete the data from a table entirely, then `TRUNCATE` is the fastest way to do so.
- Irrespective of size of the table, data can be cleaned up with in no time.
- Truncate operations can be rolled back.
- `TRUNCATE` is a DDL statement. In Postgres, DDL statements are not auto committed. In most of the databases, DDL statements are committed automatically.
- One cannot **truncate** the table with only DML permissions.
- As part of the web or mobile applications, we typically will not have `TRUNCATE` as part of the core logic.
- In Data Engineering or ETL applications, it is used more commonly to truncate intermediate or stage tables.
- If we have to truncate multiple related tables at the same time, then typically we truncate child tables first and then parent tables.
- We can also use `CASCADE` to truncate the data in child tables as well as in the parent.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS user_logins
```

```
* postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP SEQUENCE IF EXISTS users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN,
    user_password VARCHAR(200),
    user_role VARCHAR(1),
    is_active BOOLEAN,
    created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE user_logins (
    user_login_id SERIAL PRIMARY KEY,
    user_id INT,
    user_login_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_ip_addr VARCHAR(20)
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE user_logins
    ADD FOREIGN KEY (user_id)
    REFERENCES users(user_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

Warning: You will not be able to truncate parent table with out cascade (even when tables are empty)

```
%sql TRUNCATE TABLE users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
```

```
---------------------------------------------------------------------------
FeatureNotSupported                       Traceback (most recent call last)
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 

FeatureNotSupported: cannot truncate a table referenced in a foreign key constraint
DETAIL:  Table "user_logins" references "users".
HINT:  Truncate table "user_logins" at the same time, or use TRUNCATE ... CASCADE.


The above exception was the direct cause of the following exception:

NotSupportedError                         Traceback (most recent call last)
<ipython-input-154-a8605a816166> in <module>
----> 1 get_ipython().run_line_magic('sql', 'TRUNCATE TABLE users')

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/interactiveshell.
↪py in run_line_magic(self, magic_name, line, _stack_depth)
   2324                 kwargs['local_ns'] = sys._getframe(stack_depth).f_locals
   2325             with self.builtin_trap:
-> 2326                 result = fn(*args, **kwargs)
   2327             return result
   2328 

<decorator-gen-135> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):
```

(continues on next page)

(continued from previous page)

```
<decorator-gen-134> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/magic.py in execute(self,␣
↪line, cell, local_ns)
    215
    216         try:
--> 217             result = sql.run.run(conn, parsed["sql"], self, user_ns)
    218
    219             if (

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/run.py in run(conn, sql,␣
↪config, user_namespace)
    365             else:
    366                 txt = sqlalchemy.sql.text(statement)
--> 367                 result = conn.session.execute(txt, user_namespace)
    368             _commit(conn=conn, config=config)
    369             if result and config.feedback:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in␣
↪execute(self, object_, *multiparams, **params)
   1009             )
   1010         else:
-> 1011             return meth(self, multiparams, params)
   1012
   1013     def _execute_function(self, func, multiparams, params):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/sql/elements.py in␣
↪_execute_on_connection(self, connection, multiparams, params)
    296     def _execute_on_connection(self, connection, multiparams, params):
    297         if self.supports_execution:
--> 298             return connection._execute_clauseelement(self, multiparams,␣
↪params)
    299         else:
    300             raise exc.ObjectNotExecutableError(self)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_clauseelement(self, elem, multiparams, params)
   1128             distilled_params,
   1129             compiled_sql,
-> 1130             distilled_params,
   1131         )
   1132         if self._has_events or self.engine._has_events:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1315         except BaseException as e:
   1316             self._handle_dbapi_exception(
-> 1317                 e, statement, parameters, cursor, context
   1318             )
```

(continues on next page)

(continued from previous page)

```
   1319

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1509             elif should_wrap:
   1510                 util.raise_(
-> 1511                     sqlalchemy_exception, with_traceback=exc_info[2], from_=e
   1512                 )
   1513             else:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/util/compat.py in␣
↪raise_(***failed resolving arguments***)
    180
    181         try:
--> 182             raise exception
    183         finally:
    184             # credit to

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1275                 if not evt_handled:
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )
   1279

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    591
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594
    595     def do_execute_no_params(self, cursor, statement, context=None):

NotSupportedError: (psycopg2.errors.FeatureNotSupported) cannot truncate a table␣
↪referenced in a foreign key constraint
DETAIL:  Table "user_logins" references "users".
HINT:  Truncate table "user_logins" at the same time, or use TRUNCATE ... CASCADE.

[SQL: TRUNCATE TABLE users]
(Background on this error at: http://sqlalche.me/e/13/tw8g)
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Donald', 'Duck', 'donald@duck.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_role, is_
↪active)
```

(continues on next page)

(continued from previous page)

```
VALUES ('Mickey', 'Mouse', 'mickey@mouse.com', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users
    (user_first_name, user_last_name, user_email_id, user_password, user_role, is_
↪active)
VALUES
    ('Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true),
    ('Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true),
    ('Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO user_logins
    (user_id)
VALUES
    (1),
    (2),
    (3),
    (1),
    (1),
    (4)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 'Donald', 'Duck', 'donald@duck.com', None, None, None, None, datetime.date(2020,␣
↪11, 23)),
 (2, 'Mickey', 'Mouse', 'mickey@mouse.com', None, None, 'U', True, datetime.date(2020,
↪ 11, 23)),
 (3, 'Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', None, 'h9LAz7p7ub', 'U',␣
↪True, datetime.date(2020, 11, 23)),
 (4, 'Tobe', 'Lyness', 'tlyness1@paginegialle.it', None, 'oEofndp', 'U', True,␣
↪datetime.date(2020, 11, 23)),
 (5, 'Addie', 'Mesias', 'amesias2@twitpic.com', None, 'ih7Y69u56', 'U', True,␣
↪datetime.date(2020, 11, 23))]
```

(continues on next page)

(continued from previous page)

```
%sql SELECT * FROM user_logins
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[(1, 1, datetime.datetime(2020, 11, 23, 16, 44, 8, 289602), None),
 (2, 2, datetime.datetime(2020, 11, 23, 16, 44, 8, 289602), None),
 (3, 3, datetime.datetime(2020, 11, 23, 16, 44, 8, 289602), None),
 (4, 1, datetime.datetime(2020, 11, 23, 16, 44, 8, 289602), None),
 (5, 1, datetime.datetime(2020, 11, 23, 16, 44, 8, 289602), None),
 (6, 4, datetime.datetime(2020, 11, 23, 16, 44, 8, 289602), None)]
```

Note: `TRUNCATE` with `CASCADE` will truncate data from child table as well.

```
%sql TRUNCATE TABLE users CASCADE
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql SELECT * FROM users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%sql SELECT * FROM user_logins
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

6.4.9 Dropping Tables

Let us go through the details related to dropping tables.

- We can drop table using `DROP TABLE`.
- All the direct dependent objects such as indexes, primary key constraints, unique constraints, not null constraints will automatically be dropped.
- Sequences will be dropped only if the sequence is owned by the column.
- If there are child tables for the table being dropped, then we need to specify `CASCADE`.
- Using `CASCADE` will drop the constraints from the child table, but not the child tables themselves.

- We can also drop the foreign key constraints before dropping the parent table instead of using CASCADE.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS user_logins
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%sql DROP SEQUENCE IF EXISTS users_user_id_seq
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN,
    user_password VARCHAR(200),
    user_role VARCHAR(1),
    is_active BOOLEAN,
    created_dt DATE DEFAULT CURRENT_DATE
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql
```

(continues on next page)

(continued from previous page)

```
CREATE TABLE user_logins (
    user_login_id SERIAL PRIMARY KEY,
    user_id INT,
    user_login_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_ip_addr VARCHAR(20)
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE user_logins
    ADD FOREIGN KEY (user_id)
    REFERENCES users(user_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_name IN ('users', 'user_logins')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[('itversity_retail_db', 'public', 'users', 'BASE TABLE', None, None, None, None,␣
↪None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'user_logins', 'BASE TABLE', None, None, None,␣
↪None, None, 'YES', 'NO', None)]
```

```
%%sql

SELECT * FROM information_schema.sequences
WHERE sequence_name = 'users_user_id_seq'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'public', 'users_user_id_seq', 'integer', 32, 2, 0, '1', '1',
↪ '2147483647', '1', 'NO')]
```

```
%%sql

SELECT * FROM information_schema.sequences
WHERE sequence_name = 'user_logins_user_login_id_seq'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'public', 'user_logins_user_login_id_seq', 'integer', 32, 2,␣
↪0, '1', '1', '2147483647', '1', 'NO')]
```

Error: We will not be able to drop the parent tables with out dropping the child tables or specifying `CASCADE`. Using `CASCADE` will not drop child tables, it only drops the foreign key constraints.

```
%sql DROP TABLE users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
```

```
---------------------------------------------------------------------------
DependentObjectsStillExist                Traceback (most recent call last)
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 

DependentObjectsStillExist: cannot drop table users because other objects depend on it
DETAIL:  constraint user_logins_user_id_fkey on table user_logins depends on table␣
↪users
HINT:  Use DROP ... CASCADE to drop the dependent objects too.


The above exception was the direct cause of the following exception:

InternalError                             Traceback (most recent call last)
<ipython-input-175-46beae3783fd> in <module>
----> 1 get_ipython().run_line_magic('sql', 'DROP TABLE users')

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/interactiveshell.
↪py in run_line_magic(self, magic_name, line, _stack_depth)
   2324                 kwargs['local_ns'] = sys._getframe(stack_depth).f_locals
   2325             with self.builtin_trap:
-> 2326                 result = fn(*args, **kwargs)
   2327             return result
   2328 

<decorator-gen-135> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
```

(continues on next page)

(continued from previous page)

```
    188
    189         if callable(arg):

<decorator-gen-134> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/magic.py in execute(self,␣
↪line, cell, local_ns)
    215
    216         try:
--> 217             result = sql.run.run(conn, parsed["sql"], self, user_ns)
    218
    219             if (

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/run.py in run(conn, sql,␣
↪config, user_namespace)
    365             else:
    366                 txt = sqlalchemy.sql.text(statement)
--> 367                 result = conn.session.execute(txt, user_namespace)
    368             _commit(conn=conn, config=config)
    369             if result and config.feedback:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in␣
↪execute(self, object_, *multiparams, **params)
   1009             )
   1010         else:
-> 1011             return meth(self, multiparams, params)
   1012
   1013     def _execute_function(self, func, multiparams, params):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/sql/elements.py in␣
↪_execute_on_connection(self, connection, multiparams, params)
    296     def _execute_on_connection(self, connection, multiparams, params):
    297         if self.supports_execution:
--> 298             return connection._execute_clauseelement(self, multiparams,␣
↪params)
    299         else:
    300             raise exc.ObjectNotExecutableError(self)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_clauseelement(self, elem, multiparams, params)
   1128             distilled_params,
   1129             compiled_sql,
-> 1130             distilled_params,
   1131         )
   1132         if self._has_events or self.engine._has_events:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1315         except BaseException as e:
```

(continues on next page)

(continued from previous page)

```
   1316                 self._handle_dbapi_exception(
-> 1317                     e, statement, parameters, cursor, context
   1318                 )
   1319

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1509                 elif should_wrap:
   1510                     util.raise_(
-> 1511                         sqlalchemy_exception, with_traceback=exc_info[2], from_=e
   1512                     )
   1513                 else:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/util/compat.py in␣
↪raise_(***failed resolving arguments***)
    180
    181         try:
--> 182             raise exception
    183         finally:
    184             # credit to

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1275                     if not evt_handled:
   1276                         self.dialect.do_execute(
-> 1277                             cursor, statement, parameters, context
   1278                         )
   1279

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    591
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594
    595     def do_execute_no_params(self, cursor, statement, context=None):

InternalError: (psycopg2.errors.DependentObjectsStillExist) cannot drop table users␣
↪because other objects depend on it
DETAIL:  constraint user_logins_user_id_fkey on table user_logins depends on table␣
↪users
HINT:  Use DROP ... CASCADE to drop the dependent objects too.

[SQL: DROP TABLE users]
(Background on this error at: http://sqlalche.me/e/13/2j85)
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id)
VALUES ('Donald', 'Duck', 'donald@duck.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users (user_first_name, user_last_name, user_email_id, user_role, is_
↪active)
VALUES ('Mickey', 'Mouse', 'mickey@mouse.com', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users
    (user_first_name, user_last_name, user_email_id, user_password, user_role, is_
↪active)
VALUES
    ('Gordan', 'Bradock', 'gbradock0@barnesandnoble.com', 'h9LAz7p7ub', 'U', true),
    ('Tobe', 'Lyness', 'tlyness1@paginegialle.it', 'oEofndp', 'U', true),
    ('Addie', 'Mesias', 'amesias2@twitpic.com', 'ih7Y69u56', 'U', true)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO user_logins
    (user_id)
VALUES
    (1),
    (2),
    (3),
    (1),
    (1),
    (4)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[]
```

```
%sql DROP TABLE users CASCADE
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql
```

(continues on next page)

(continued from previous page)

```
SELECT * FROM information_schema.tables
WHERE table_name IN ('users', 'user_logins')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('itversity_retail_db', 'public', 'user_logins', 'BASE TABLE', None, None, None,␣
↪None, None, 'YES', 'NO', None)]
```

```
%%sql

SELECT * FROM information_schema.sequences
WHERE sequence_name = 'users_user_id_seq'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%sql SELECT * FROM user_logins
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
6 rows affected.
```

```
[(1, 1, datetime.datetime(2020, 11, 23, 16, 44, 45, 373009), None),
 (2, 2, datetime.datetime(2020, 11, 23, 16, 44, 45, 373009), None),
 (3, 3, datetime.datetime(2020, 11, 23, 16, 44, 45, 373009), None),
 (4, 1, datetime.datetime(2020, 11, 23, 16, 44, 45, 373009), None),
 (5, 1, datetime.datetime(2020, 11, 23, 16, 44, 45, 373009), None),
 (6, 4, datetime.datetime(2020, 11, 23, 16, 44, 45, 373009), None)]
```

6.4.10 Exercises - Managing Database Objects

This exercise is primarily to assess your capabilities related to put all important DDL concepts in practice by coming up with solution for a typical data migration problem from one database (mysql) to another (postgres).

- Here are the high level steps for database migration from one type of database to another type of database.
  - Extract DDL Statements from source database (MySQL).
  - Extract the data in the form of delimited files and ship them to target database.
  - Refactor scripts as per target database (Postgres).
  - Create tables in the target database.
  - Execute pre-migration steps (disable constraints, drop indexes etc).
  - Load the data using native utilities.
  - Execute post-migration steps (enable constraints, create or rebuild indexes, reset sequences etc).
  - Sanity checks with basic queries.
  - Make sure all the impacted applications are validated thoroughly.

- We have scripts and data set available in our GitHub repository. If you are using our environment the repository is already cloned under **/data/retail_db**.
- It have scripts to create tables with primary keys. Those scripts are generated from MySQL tables and refactored for Postgres.
  - Script to create tables: **create_db_tables_pg.sql**
  - Load data into tables: **load_db_tables_pg.sql**
- Here are the steps you need to perform to take care of this exercise.
  - Create tables
  - Load data
  - All the tables have surrogate primary keys. Here are the details.
    * orders.order_id
    * order_items.order_item_id
    * customers.customer_id
    * products.product_id
    * categories.category_id
    * departments.department_id
  - Get the maximum value from all surrogate primary key fields.
  - Create sequences for all surrogate primary key fields using maximum value. Make sure to use standard naming conventions for sequences.
  - Ensure sequences are mapped to the surrogate primary key fields.
  - Create foreign key constraints based up on this information.
    * orders.order_customer_id to customers.customer_id
    * order_items.order_item_order_id to orders.order_id
    * order_items.order_item_product_id to products.product_id
    * products.product_category_id to categories.category_id
    * categories.category_department_id to departments.department_id
  - Insert few records in `departments` to ensure that sequence generated numbers are used for `department_id`.
- Here are the commands to launch `psql` and run scripts to create tables as well as load data into tables.

```
psql -U itversity_retail_user \
  -h localhost \
  -p 5432 \
  -d itversity_retail_db \
  -W

\i /data/retail_db/create_db_tables_pg.sql

\i /data/retail_db/load_db_tables_pg.sql
```

- We use this approach of creating tables, loading data and then adding constraints as well as resetting sequences for large volume data migrations from one database to another database.

- Here are the commands or queries you need to come up with to solve this problem.

Exercise 1

Queries to get maximum values from surrogate primary keys.

Exercise 2

Commands to add sequences with `START WITH` pointing to the maximum value for the corresponding surrogate primary key fields. Make sure to use meaningful names to sequences **TABLENAME_SURROGATEFIELD_seq** (example: users_user_id_seq for users.user_id)

Exercise 3

Commands to alter sequences to bind them to corresponding surrogate primary key fields.

Exercise 4

Add Foreign Key constraints to the tables.

- Validate if the tables have data violataing foreign key constraints (Hint: You can use left outer join to find rows in child table but not in parent table)
- Delete the data which violates the parent child relationship or foreign key relationship.
- Alter tables to add foreign keys as specified.
- Here are the relationships for your reference.
  - orders.order_customer_id to customers.customer_id
  - order_items.order_item_order_id to orders.order_id
  - order_items.order_item_product_id to products.product_id
  - products.product_category_id to categories.category_id
  - categories.category_department_id to departments.department_id
- Solution should contain the following:
  - Queries for validation
  - Delete statements to delete the data
  - Commands to add foreign keys to the tables.

Exercise 5

Queries to validate whether constraints are created or not. You can come up with queries against `information_schema` tables such as `columns`, `sequences` etc.

6.5 Partitioning Tables and Indexes

As part of this section we will primarily talk about partitioning tables as well as indexes.

- Overview of Partitioning
- List Partitioning
- Managing Partitions - List
- Manipulating Data
- Range Partitioning
- Managing Partitions - Range
- Repartitioning - Range
- Hash Partitioning
- Managing Partitions - Hash
- Usage Scenarios
- Sub Partitioning
- Exercise - Paritioning Tables

Here are the key objectives of this section.

- Different partitioning strategies
- How to create and manage partitioned tables?
- How to manipulate data by inserting, updating and deleting data from managed tables?
- How to repartition the tables if partitioning strategy is changed (example: from yearly to monthly)?
- Learn about sub partitioning or nested partitioning or multi level partitioning with examples.
- Self evaluate whether one understood key skills related to partitioned tables or not using exercises.

6.5.1 Overview of Partitioning

Most of the modern database technologies support wide variety of partitioning strategies. However, here are the most commonly used ones.

- List Partitioning
- Range Partitioning
- Hash Partitioning
- List and Range are more widely used compared to Hash Partitioning.
- We can also mix and match these to have multi level partitioning. It is known as sub partitioning.
- We can either partition a table with out primary key or partition a table with primary key when partition column is prime attribute (one of the primary key columns).
- Indexes can be added to the partitioned table. If we create on the main table, it is global index and if we create index on each partition then it is partitioned index.

6.5.2 List Partitioning

Let us understand how we can take care of list partitioning of tables.

- It is primarily used to create partitions based up on the values.
- Here are the steps involved in creating table using list partitioning strategy.
  - Create table using `PARTITION BY LIST`
  - Add default and value specific partitions
  - Validate by inserting data into the table
- We can detach as well as drop the partitions from the table.

Create Partitioned Table

Let us create partitioned table with name `users_part`.

- It contains same columns as `users`.
- We will partition based up on `user_role` field.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%sql DROP TABLE IF EXISTS users
```

```
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[ ]
```

```
%sql DROP TABLE IF EXISTS users_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[ ]
```

```
%%sql

CREATE TABLE users_part (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_role, user_id)
) PARTITION BY LIST(user_role)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[ ]
```

Note: Additional indexes on the users_part table.

```
%%sql

CREATE INDEX users_part_email_id_idx
    ON users_part(user_email_id)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[ ]
```

Error: Below `INSERT` statement will fail as we have not added any partitions to the table `users_part` even though it is created as partitioned table.

```
%%sql

INSERT INTO users_part (user_first_name, user_last_name, user_email_id)
VALUES
```

(continues on next page)

(continued from previous page)

```
    ('Scott', 'Tiger', 'scott@tiger.com'),
    ('Donald', 'Duck', 'donald@duck.com'),
    ('Mickey', 'Mouse', 'mickey@mouse.com')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
```

```
---------------------------------------------------------------------------
CheckViolation                            Traceback (most recent call last)
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 

CheckViolation: no partition of relation "users_part" found for row
DETAIL:  Partition key of the failing row contains (user_role) = (U).


The above exception was the direct cause of the following exception:

IntegrityError                            Traceback (most recent call last)
<ipython-input-23-b06b3c83cab2> in <module>
----> 1 get_ipython().run_cell_magic('sql', '', "\nINSERT INTO users_part (user_first_
↪name, user_last_name, user_email_id)\nVALUES \n    ('Scott', 'Tiger', 'scott@tiger.
↪com'),\n    ('Donald', 'Duck', 'donald@duck.com'),\n    ('Mickey', 'Mouse',
↪'mickey@mouse.com')\n")

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/interactiveshell.
↪py in run_cell_magic(self, magic_name, line, cell)
   2369             with self.builtin_trap:
   2370                 args = (magic_arg_s, cell)
-> 2371                 result = fn(*args, **kwargs)
   2372             return result
   2373 

<decorator-gen-135> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

<decorator-gen-134> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
```

(continues on next page)

(continued from previous page)

```
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/magic.py in execute(self, line, cell, local_ns)
    215 
    216         try:
--> 217             result = sql.run.run(conn, parsed["sql"], self, user_ns)
    218 
    219             if (

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/run.py in run(conn, sql, config, user_namespace)
    365             else:
    366                 txt = sqlalchemy.sql.text(statement)
--> 367                 result = conn.session.execute(txt, user_namespace)
    368             _commit(conn=conn, config=config)
    369             if result and config.feedback:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in execute(self, object_, *multiparams, **params)
   1009             )
   1010         else:
-> 1011             return meth(self, multiparams, params)
   1012 
   1013     def _execute_function(self, func, multiparams, params):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/sql/elements.py in _execute_on_connection(self, connection, multiparams, params)
    296     def _execute_on_connection(self, connection, multiparams, params):
    297         if self.supports_execution:
--> 298             return connection._execute_clauseelement(self, multiparams, params)
    299         else:
    300             raise exc.ObjectNotExecutableError(self)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _execute_clauseelement(self, elem, multiparams, params)
   1128             distilled_params,
   1129             compiled_sql,
-> 1130             distilled_params,
   1131         )
   1132         if self._has_events or self.engine._has_events:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _execute_context(self, dialect, constructor, statement, parameters, *args)
   1315         except BaseException as e:
   1316             self._handle_dbapi_exception(
-> 1317                 e, statement, parameters, cursor, context
   1318             )
   1319 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1509             elif should_wrap:
```

(continues on next page)

(continued from previous page)

```
   1510                     util.raise_(
-> 1511                         sqlalchemy_exception, with_traceback=exc_info[2], from_=e
   1512                     )
   1513                 else:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/util/compat.py in␣
↪raise_(***failed resolving arguments***)
    180
    181         try:
--> 182             raise exception
    183         finally:
    184             # credit to

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1275                 if not evt_handled:
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )
   1279

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    591
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594
    595     def do_execute_no_params(self, cursor, statement, context=None):

IntegrityError: (psycopg2.errors.CheckViolation) no partition of relation "users_part
↪" found for row
DETAIL:  Partition key of the failing row contains (user_role) = (U).

[SQL: INSERT INTO users_part (user_first_name, user_last_name, user_email_id)
VALUES
    ('Scott', 'Tiger', 'scott@tiger.com'),
    ('Donald', 'Duck', 'donald@duck.com'),
    ('Mickey', 'Mouse', 'mickey@mouse.com')]
(Background on this error at: http://sqlalche.me/e/13/gkpj)
```

6.5.3 Managing Partitions - List

Let us understand how to manage partitions for a partitioned table using `users_part`.

- All users data with `user_role` as **'U'** should go to one partition by name `users_part_u`.
- All users data with `user_role` as **'A'** should go to one partition by name `users_part_a`.
- We can add partition to existing partitioned table using `CREATE TABLE partition_name PARTITION OF table_name`.
- We can have a partition for default values so that all the data that does not satisfy the partition condition can be added to it.
- We can have a partition for each value or for a set of values.
  - We can have one partition for `U` as well as `A` and default partition for all other values.

  - We can have individual partitions for `U`, `A` respectively and default partition for all other values.
  - We can use `FOR VALUES IN (val1, val2)` as part of `CREATE TABLE partition_name PARTITION OF table_name` to specify values for respective table created for partition.
- Once partitions are added, we can insert data into the partitioned table.
- We can detach using `ALTER TABLE` and drop the partition or drop the partition directly. To drop the partition we need to use `DROP TABLE` command.

Note: Here is how we can create partition for default values for a list partitioned table **users_part**.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%%sql

CREATE TABLE users_part_default
PARTITION OF users_part DEFAULT
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

Note: All the 3 records will go to default partition as we have not defined any partition for user_role 'U'.

```
%%sql

INSERT INTO users_part (user_first_name, user_last_name, user_email_id, user_role)
VALUES
    ('Scott', 'Tiger', 'scott@tiger.com', 'U'),
    ('Donald', 'Duck', 'donald@duck.com', 'U'),
    ('Mickey', 'Mouse', 'mickey@mouse.com', 'U')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part_default
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(2, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594)),
 (3, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594)),
 (4, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2020, 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594))]
```

```
%%sql

CREATE TABLE users_part_a
PARTITION OF users_part
FOR VALUES IN ('A')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

UPDATE users_part
SET
    user_role = 'A'
WHERE user_email_id = 'scott@tiger.com'
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(2, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594)),
 (3, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594)),
 (4, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2020, 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594))]
```

```
%%sql

SELECT * FROM users_part_a
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(2, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594))]
```

```
%%sql

SELECT * FROM users_part_default
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
2 rows affected.
```

```
[(3, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594)),
 (4, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2020, 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594))]
```

Error: This will fail as there are records with user_role 'U' in default partition.

```
%%sql

CREATE TABLE users_part_u
PARTITION OF users_part
FOR VALUES IN ('U')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
```

```
---------------------------------------------------------------------------
CheckViolation                            Traceback (most recent call last)
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1276                     self.dialect.do_execute(
-> 1277                         cursor, statement, parameters, context
   1278                     )

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 

CheckViolation: updated partition constraint for default partition "users_part_default
↪" would be violated by some row


The above exception was the direct cause of the following exception:

IntegrityError                            Traceback (most recent call last)
<ipython-input-35-fbb5e14aecbd> in <module>
----> 1 get_ipython().run_cell_magic('sql', '', "\nCREATE TABLE users_part_u \
↪nPARTITION OF users_part  \nFOR VALUES IN ('U')\n")

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/interactiveshell.
↪py in run_cell_magic(self, magic_name, line, cell)
   2369             with self.builtin_trap:
```

(continues on next page)

(continued from previous page)

```
   2370                 args = (magic_arg_s, cell)
-> 2371                 result = fn(*args, **kwargs)
   2372             return result
   2373 

<decorator-gen-135> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

<decorator-gen-134> in execute(self, line, cell, local_ns)

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/IPython/core/magic.py in
↪<lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/magic.py in execute(self,␣
↪line, cell, local_ns)
    215 
    216         try:
--> 217             result = sql.run.run(conn, parsed["sql"], self, user_ns)
    218 
    219             if (

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sql/run.py in run(conn, sql,␣
↪config, user_namespace)
    365             else:
    366                 txt = sqlalchemy.sql.text(statement)
--> 367                 result = conn.session.execute(txt, user_namespace)
    368             _commit(conn=conn, config=config)
    369             if result and config.feedback:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in␣
↪execute(self, object_, *multiparams, **params)
   1009             )
   1010         else:
-> 1011             return meth(self, multiparams, params)
   1012 
   1013     def _execute_function(self, func, multiparams, params):

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/sql/elements.py in␣
↪_execute_on_connection(self, connection, multiparams, params)
    296     def _execute_on_connection(self, connection, multiparams, params):
    297         if self.supports_execution:
--> 298             return connection._execute_clauseelement(self, multiparams,␣
↪params)
    299         else:
    300             raise exc.ObjectNotExecutableError(self)
```

(continues on next page)

(continued from previous page)

```
/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_clauseelement(self, elem, multiparams, params)
   1128                 distilled_params,
   1129                 compiled_sql,
-> 1130                 distilled_params,
   1131             )
   1132             if self._has_events or self.engine._has_events:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1315             except BaseException as e:
   1316                 self._handle_dbapi_exception(
-> 1317                     e, statement, parameters, cursor, context
   1318                 )
   1319 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1509                 elif should_wrap:
   1510                     util.raise_(
-> 1511                         sqlalchemy_exception, with_traceback=exc_info[2], from_=e
   1512                     )
   1513                 else:

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/util/compat.py in␣
↪raise_(***failed resolving arguments***)
    180 
    181             try:
--> 182                 raise exception
    183             finally:
    184                 # credit to

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/base.py in _
↪execute_context(self, dialect, constructor, statement, parameters, *args)
   1275                     if not evt_handled:
   1276                         self.dialect.do_execute(
-> 1277                             cursor, statement, parameters, context
   1278                         )
   1279 

/opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/sqlalchemy/engine/default.py␣
↪in do_execute(self, cursor, statement, parameters, context)
    591 
    592     def do_execute(self, cursor, statement, parameters, context=None):
--> 593         cursor.execute(statement, parameters)
    594 
    595     def do_execute_no_params(self, cursor, statement, context=None):

IntegrityError: (psycopg2.errors.CheckViolation) updated partition constraint for␣
↪default partition "users_part_default" would be violated by some row

[SQL: CREATE TABLE users_part_u PARTITION OF users_part
FOR VALUES IN ('U')]
(Background on this error at: http://sqlalche.me/e/13/gkpj)
```

Note: We can detach the partition, add partition for 'U' and load the data from detached partitione into the new partition created.

```
%%sql

ALTER TABLE users_part
    DETACH PARTITION users_part_default
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_part_u
PARTITION OF users_part
FOR VALUES IN ('U')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users_part
SELECT * FROM users_part_default
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
2 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part_a
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(2, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594))]
```

```
%%sql

SELECT * FROM users_part_u
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
2 rows affected.
```

```
[(3, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594)),
 (4, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2020, 11, 24), datetime.datetime(2020, 11, 24, 12, 11, 46, 894594))]
```

Note: We can drop and create partition for default or truncate and attach the existing default partition.

```
%%sql

DROP TABLE users_part_default
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_part_default
PARTITION OF users_part DEFAULT
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

6.5.4 Manipulating Data

Let us understand how we can manipulate data for a partitioned table.

- We can insert data using the table (most preferred way).
- As we define table for each partition, we can insert data using table created for specific partition.
- In the case of `users_part` partitioned table, we can either use table name`users_part` or partition name `users_part_u` to insert records with user_role **'U'**.

```
CREATE TABLE users_part_u
PARTITION OF users_part
FOR VALUES IN ('U')
```

- As part of the update, if we change the value in a partitioned column which will result in changing partition, then internally data from one partition will be moved to other.
- We can delete the data using the table or the table created for each partition (either by using table name `users_part` or partitions such as `users_part_u`, `users_part_a` etc

Note: DML is same irrespective of the partitioning strategy. This applies to all 3 partitioning strategies - **list**, **range** as well as **hash**.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%%sql

TRUNCATE TABLE users_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users_part (user_first_name, user_last_name, user_email_id, user_role)
VALUES
    ('Scott', 'Tiger', 'scott@tiger.com', 'U'),
    ('Donald', 'Duck', 'donald@duck.com', 'U'),
    ('Mickey', 'Mouse', 'mickey@mouse.com', 'U')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part_u
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(5, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850)),
 (6, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850)),
 (7, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2020, 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850))]
```

```
%%sql

INSERT INTO users_part_a (user_first_name, user_last_name, user_email_id, user_role)
VALUES
    ('Matt', 'Clarke', 'matt@clarke.com', 'A')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
4 rows affected.
```

```
[(8, 'Matt', 'Clarke', 'matt@clarke.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 9, 284614)),
 (5, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850)),
 (6, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850)),
 (7, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2020, 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850))]
```

```
%%sql

UPDATE users_part SET
    user_role = 'A'
WHERE user_email_id = 'donald@duck.com'
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part_a
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
2 rows affected.
```

```
[(8, 'Matt', 'Clarke', 'matt@clarke.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 9, 284614)),
 (6, 'Donald', 'Duck', 'donald@duck.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850))]
```

```
%%sql

DELETE FROM users_part WHERE user_email_id = 'donald@duck.com'
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

DELETE FROM users_part_u WHERE user_email_id = 'mickey@mouse.com'
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
2 rows affected.
```

```
[(8, 'Matt', 'Clarke', 'matt@clarke.com', False, None, 'A', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 9, 284614)),
 (5, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2020,
↪ 11, 24), datetime.datetime(2020, 11, 24, 12, 12, 8, 505850))]
```

6.5.5 Range Partitioning

Let us understand how we can take care of range partitioning of tables.

- It is primarily used to create partitions based up on a given range of values.
- Here are the steps involved in creating table using range partitioning strategy.
  - Create table using `PARTITION BY RANGE`
  - Add default and range specific partitions
  - Validate by inserting data into the table
- We can detach as well as drop the partitions from the table.

Create Partitioned Table

Let us create partitioned table with name `users_range_part`.

- It contains same columns as `users`.
- We will partition the table based up on `created_dt` field.
- We will create one partition per year with naming convention **users_range_part_yyyy** (users_range_part_2016).

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%sql DROP TABLE IF EXISTS users_range_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_range_part (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (created_dt, user_id)
) PARTITION BY RANGE(created_dt)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

Note: We will not be able to insert the data until we add at least one partition.

6.5.6 Managing Partitions - Range

Let us understand how to manage partitions for the table `users_range_part`.

- All users data created in a specific year should go to the respective partition created.
- For example, all users data created in the year of 2016 should go to `users_range_part_2016`.
- We can add partition to existing partitioned table using `CREATE TABLE partition_name PARTITION OF table_name`.
- We can have a partition for default values so that all the data that does not satisfy the partition condition can be added to it.
- We can have a partition for specific range of values using `FOR VALUES FROM (from_value) TO (to_value)` as part of `CREATE TABLE partition_name PARTITION OF table_name`.
- Once partitions are added, we can insert data into the partitioned table.

Note: Here is how we can create partition for default values for a range partitioned table **users_range_part**.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%%sql

CREATE TABLE users_range_part_default
PARTITION OF users_range_part DEFAULT
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_range_part_2016
PARTITION OF users_range_part
FOR VALUES FROM ('2016-01-01') TO ('2016-12-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

Error: As there is a overlap between the previous partition and below one, command to create partition for data ranging from 2016-01-01 till 2017-12-31 will fail.

```
%%sql

CREATE TABLE users_range_part_2017
PARTITION OF users_range_part
FOR VALUES FROM ('2016-01-01') TO ('2017-12-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
(psycopg2.errors.InvalidObjectDefinition) partition "users_range_part_2017" would␣
↪overlap partition "users_range_part_2016"

[SQL: CREATE TABLE users_range_part_2017 PARTITION OF users_range_part
FOR VALUES FROM ('2016-01-01') TO ('2017-12-31')]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

Note: This is how we can create partitions for the years **2017**, **2018**, **2019** etc

```
%%sql

CREATE TABLE users_range_part_2017
PARTITION OF users_range_part
FOR VALUES FROM ('2017-01-01') TO ('2017-12-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_range_part_2018
PARTITION OF users_range_part
FOR VALUES FROM ('2018-01-01') TO ('2018-12-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_range_part_2019
PARTITION OF users_range_part
FOR VALUES FROM ('2019-01-01') TO ('2019-12-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_range_part_2020
PARTITION OF users_range_part
FOR VALUES FROM ('2020-01-01') TO ('2020-12-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users_range_part
    (user_first_name, user_last_name, user_email_id, created_dt)
VALUES
```

(continues on next page)

(continued from previous page)

```
    ('Scott', 'Tiger', 'scott@tiger.com', '2018-10-01'),
    ('Donald', 'Duck', 'donald@duck.com', '2019-02-10'),
    ('Mickey', 'Mouse', 'mickey@mouse.com', '2017-06-22')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[]
```

```
%%sql

SELECT user_first_name, user_last_name, user_email_id, created_dt
FROM users_range_part_default
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT user_first_name, user_last_name, user_email_id, created_dt
FROM users_range_part_2017
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[('Mickey', 'Mouse', 'mickey@mouse.com', datetime.date(2017, 6, 22))]
```

```
%%sql

SELECT user_first_name, user_last_name, user_email_id, created_dt
FROM users_range_part_2018
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[('Scott', 'Tiger', 'scott@tiger.com', datetime.date(2018, 10, 1))]
```

```
%%sql

SELECT user_first_name, user_last_name, user_email_id, created_dt
FROM users_range_part_2019
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[('Donald', 'Duck', 'donald@duck.com', datetime.date(2019, 2, 10))]
```

```
%%sql
```

(continues on next page)

(continued from previous page)

```
SELECT user_first_name, user_last_name, user_email_id, created_dt
FROM users_range_part_2020
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

6.5.7 Repartitioning - Range

Let us understand how we can repartition the existing partitioned table.

- We will use **users_range_part** table. It is originally partitioned for each year.
- Now we would like to partition for each month.
- Here are the steps that are involved in repartitioning from year to month.
  - Detach all yearly partitions from **users_range_part**.
  - Add new partitions for each month.
  - Load data from detached partitions into the table with new partitions for each month.
  - Validate to ensure that all the data is copied.
  - Drop all the detached partitions.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

Note: Detach all yearly partitions

```
%%sql

ALTER TABLE users_range_part
    DETACH PARTITION users_range_part_2016
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users_range_part
    DETACH PARTITION users_range_part_2017
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users_range_part
    DETACH PARTITION users_range_part_2018
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users_range_part
    DETACH PARTITION users_range_part_2019
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE users_range_part
    DETACH PARTITION users_range_part_2020
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

Note: Add new partitions for every month between 2016 January and 2020 December.

```
!pip install psycopg2
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: psycopg2 in /opt/anaconda3/envs/beakerx/lib/python3.6/
↪site-packages (2.8.6)
```

```
import pandas as pd
from pandas.tseries.offsets import MonthBegin, MonthEnd
```

(continues on next page)

(continued from previous page)

```
months = pd.date_range(start='1/1/2016', end='3/31/2016', freq='1M')

for month in months:
    begin_date = month - MonthBegin(1)
    end_date = month + MonthEnd(0)
    print(str(month)[:7].replace('-', ''), end=':')
    print(str(begin_date).split(' ')[0], end=':')
    print(str(end_date).split(' ')[0])
```

```
201601:2016-01-01:2016-01-31
201602:2016-02-01:2016-02-29
201603:2016-03-01:2016-03-31
```

```
import psycopg2
```

```
import pandas as pd
from pandas.tseries.offsets import MonthBegin, MonthEnd

months = pd.date_range(start='1/1/2016', end='12/31/2020', freq='1M')

connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_sms_db',
    user='itversity_sms_user',
    password='sms_password'
)
cursor = connection.cursor()
table_name = 'users_range_part'
query = '''
CREATE TABLE {table_name}_{yyyymm}
PARTITION OF {table_name}
FOR VALUES FROM ('{begin_date}') TO ('{end_date}')
'''
for month in months:
    begin_date = month - MonthBegin(1)
    end_date = month + MonthEnd(0)
    print(f'Adding partition for {begin_date} and {end_date}')
    cursor.execute(
        query.format(
            table_name=table_name,
            yyyymm=str(month)[:7].replace('-', ''),
            begin_date=str(begin_date).split(' ')[0],
            end_date=str(end_date).split(' ')[0]
        ), ()
    )
connection.commit()
cursor.close()
connection.close()
```

```
Adding partition for 2016-01-01 00:00:00 and 2016-01-31 00:00:00
Adding partition for 2016-02-01 00:00:00 and 2016-02-29 00:00:00
Adding partition for 2016-03-01 00:00:00 and 2016-03-31 00:00:00
Adding partition for 2016-04-01 00:00:00 and 2016-04-30 00:00:00
```

(continues on next page)

(continued from previous page)

```
Adding partition for 2016-05-01 00:00:00 and 2016-05-31 00:00:00
Adding partition for 2016-06-01 00:00:00 and 2016-06-30 00:00:00
Adding partition for 2016-07-01 00:00:00 and 2016-07-31 00:00:00
Adding partition for 2016-08-01 00:00:00 and 2016-08-31 00:00:00
Adding partition for 2016-09-01 00:00:00 and 2016-09-30 00:00:00
Adding partition for 2016-10-01 00:00:00 and 2016-10-31 00:00:00
Adding partition for 2016-11-01 00:00:00 and 2016-11-30 00:00:00
Adding partition for 2016-12-01 00:00:00 and 2016-12-31 00:00:00
Adding partition for 2017-01-01 00:00:00 and 2017-01-31 00:00:00
Adding partition for 2017-02-01 00:00:00 and 2017-02-28 00:00:00
Adding partition for 2017-03-01 00:00:00 and 2017-03-31 00:00:00
Adding partition for 2017-04-01 00:00:00 and 2017-04-30 00:00:00
Adding partition for 2017-05-01 00:00:00 and 2017-05-31 00:00:00
Adding partition for 2017-06-01 00:00:00 and 2017-06-30 00:00:00
Adding partition for 2017-07-01 00:00:00 and 2017-07-31 00:00:00
Adding partition for 2017-08-01 00:00:00 and 2017-08-31 00:00:00
Adding partition for 2017-09-01 00:00:00 and 2017-09-30 00:00:00
Adding partition for 2017-10-01 00:00:00 and 2017-10-31 00:00:00
Adding partition for 2017-11-01 00:00:00 and 2017-11-30 00:00:00
Adding partition for 2017-12-01 00:00:00 and 2017-12-31 00:00:00
Adding partition for 2018-01-01 00:00:00 and 2018-01-31 00:00:00
Adding partition for 2018-02-01 00:00:00 and 2018-02-28 00:00:00
Adding partition for 2018-03-01 00:00:00 and 2018-03-31 00:00:00
Adding partition for 2018-04-01 00:00:00 and 2018-04-30 00:00:00
Adding partition for 2018-05-01 00:00:00 and 2018-05-31 00:00:00
Adding partition for 2018-06-01 00:00:00 and 2018-06-30 00:00:00
Adding partition for 2018-07-01 00:00:00 and 2018-07-31 00:00:00
Adding partition for 2018-08-01 00:00:00 and 2018-08-31 00:00:00
Adding partition for 2018-09-01 00:00:00 and 2018-09-30 00:00:00
Adding partition for 2018-10-01 00:00:00 and 2018-10-31 00:00:00
Adding partition for 2018-11-01 00:00:00 and 2018-11-30 00:00:00
Adding partition for 2018-12-01 00:00:00 and 2018-12-31 00:00:00
Adding partition for 2019-01-01 00:00:00 and 2019-01-31 00:00:00
Adding partition for 2019-02-01 00:00:00 and 2019-02-28 00:00:00
Adding partition for 2019-03-01 00:00:00 and 2019-03-31 00:00:00
Adding partition for 2019-04-01 00:00:00 and 2019-04-30 00:00:00
Adding partition for 2019-05-01 00:00:00 and 2019-05-31 00:00:00
Adding partition for 2019-06-01 00:00:00 and 2019-06-30 00:00:00
Adding partition for 2019-07-01 00:00:00 and 2019-07-31 00:00:00
Adding partition for 2019-08-01 00:00:00 and 2019-08-31 00:00:00
Adding partition for 2019-09-01 00:00:00 and 2019-09-30 00:00:00
Adding partition for 2019-10-01 00:00:00 and 2019-10-31 00:00:00
Adding partition for 2019-11-01 00:00:00 and 2019-11-30 00:00:00
Adding partition for 2019-12-01 00:00:00 and 2019-12-31 00:00:00
Adding partition for 2020-01-01 00:00:00 and 2020-01-31 00:00:00
Adding partition for 2020-02-01 00:00:00 and 2020-02-29 00:00:00
Adding partition for 2020-03-01 00:00:00 and 2020-03-31 00:00:00
Adding partition for 2020-04-01 00:00:00 and 2020-04-30 00:00:00
Adding partition for 2020-05-01 00:00:00 and 2020-05-31 00:00:00
Adding partition for 2020-06-01 00:00:00 and 2020-06-30 00:00:00
Adding partition for 2020-07-01 00:00:00 and 2020-07-31 00:00:00
Adding partition for 2020-08-01 00:00:00 and 2020-08-31 00:00:00
Adding partition for 2020-09-01 00:00:00 and 2020-09-30 00:00:00
Adding partition for 2020-10-01 00:00:00 and 2020-10-31 00:00:00
Adding partition for 2020-11-01 00:00:00 and 2020-11-30 00:00:00
Adding partition for 2020-12-01 00:00:00 and 2020-12-31 00:00:00
```

Note: Load data from detached yearly partitions into monthly partitioned table.

```
%%sql

INSERT INTO users_range_part
SELECT * FROM users_range_part_2016
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users_range_part
SELECT * FROM users_range_part_2017
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users_range_part
SELECT * FROM users_range_part_2018
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users_range_part
SELECT * FROM users_range_part_2019
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[]
```

```
%%sql

INSERT INTO users_range_part
SELECT * FROM users_range_part_2020
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_range_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2017, 6, 22), datetime.datetime(2020, 11, 24, 12, 12, 27, 94936)),
 (1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2018,
↪ 10, 1), datetime.datetime(2020, 11, 24, 12, 12, 27, 94936)),
 (2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2019,
↪ 2, 10), datetime.datetime(2020, 11, 24, 12, 12, 27, 94936))]
```

```
%%sql

SELECT * FROM users_range_part_201706
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2017, 6, 22), datetime.datetime(2020, 11, 24, 12, 12, 27, 94936))]
```

```
%%sql

SELECT * FROM users_range_part_201810
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2018,
↪ 10, 1), datetime.datetime(2020, 11, 24, 12, 12, 27, 94936))]
```

```
%%sql

SELECT * FROM users_range_part_201902
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2019,
↪ 2, 10), datetime.datetime(2020, 11, 24, 12, 12, 27, 94936))]
```

Note: As we are able to see the data in the monthly partitioned table, we can drop the tables which are created earlier using yearly partitioning strategy.

```
%%sql

DROP TABLE users_range_part_2016
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

DROP TABLE users_range_part_2017
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

DROP TABLE users_range_part_2018
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

DROP TABLE users_range_part_2019
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

DROP TABLE users_range_part_2020
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

SELECT table_catalog,
    table_schema,
    table_name FROM information_schema.tables
WHERE table_name ~ 'users_range_part_'
ORDER BY table_name
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
61 rows affected.
```

```
[('itversity_sms_db', 'public', 'users_range_part_201601'),
 ('itversity_sms_db', 'public', 'users_range_part_201602'),
 ('itversity_sms_db', 'public', 'users_range_part_201603'),
 ('itversity_sms_db', 'public', 'users_range_part_201604'),
 ('itversity_sms_db', 'public', 'users_range_part_201605'),
 ('itversity_sms_db', 'public', 'users_range_part_201606'),
 ('itversity_sms_db', 'public', 'users_range_part_201607'),
 ('itversity_sms_db', 'public', 'users_range_part_201608'),
 ('itversity_sms_db', 'public', 'users_range_part_201609'),
 ('itversity_sms_db', 'public', 'users_range_part_201610'),
 ('itversity_sms_db', 'public', 'users_range_part_201611'),
 ('itversity_sms_db', 'public', 'users_range_part_201612'),
 ('itversity_sms_db', 'public', 'users_range_part_201701'),
 ('itversity_sms_db', 'public', 'users_range_part_201702'),
 ('itversity_sms_db', 'public', 'users_range_part_201703'),
 ('itversity_sms_db', 'public', 'users_range_part_201704'),
 ('itversity_sms_db', 'public', 'users_range_part_201705'),
 ('itversity_sms_db', 'public', 'users_range_part_201706'),
 ('itversity_sms_db', 'public', 'users_range_part_201707'),
 ('itversity_sms_db', 'public', 'users_range_part_201708'),
 ('itversity_sms_db', 'public', 'users_range_part_201709'),
 ('itversity_sms_db', 'public', 'users_range_part_201710'),
 ('itversity_sms_db', 'public', 'users_range_part_201711'),
 ('itversity_sms_db', 'public', 'users_range_part_201712'),
 ('itversity_sms_db', 'public', 'users_range_part_201801'),
 ('itversity_sms_db', 'public', 'users_range_part_201802'),
 ('itversity_sms_db', 'public', 'users_range_part_201803'),
 ('itversity_sms_db', 'public', 'users_range_part_201804'),
 ('itversity_sms_db', 'public', 'users_range_part_201805'),
 ('itversity_sms_db', 'public', 'users_range_part_201806'),
 ('itversity_sms_db', 'public', 'users_range_part_201807'),
 ('itversity_sms_db', 'public', 'users_range_part_201808'),
 ('itversity_sms_db', 'public', 'users_range_part_201809'),
 ('itversity_sms_db', 'public', 'users_range_part_201810'),
 ('itversity_sms_db', 'public', 'users_range_part_201811'),
 ('itversity_sms_db', 'public', 'users_range_part_201812'),
 ('itversity_sms_db', 'public', 'users_range_part_201901'),
 ('itversity_sms_db', 'public', 'users_range_part_201902'),
 ('itversity_sms_db', 'public', 'users_range_part_201903'),
 ('itversity_sms_db', 'public', 'users_range_part_201904'),
 ('itversity_sms_db', 'public', 'users_range_part_201905'),
 ('itversity_sms_db', 'public', 'users_range_part_201906'),
 ('itversity_sms_db', 'public', 'users_range_part_201907'),
 ('itversity_sms_db', 'public', 'users_range_part_201908'),
 ('itversity_sms_db', 'public', 'users_range_part_201909'),
 ('itversity_sms_db', 'public', 'users_range_part_201910'),
 ('itversity_sms_db', 'public', 'users_range_part_201911'),
 ('itversity_sms_db', 'public', 'users_range_part_201912'),
 ('itversity_sms_db', 'public', 'users_range_part_202001'),
 ('itversity_sms_db', 'public', 'users_range_part_202002'),
 ('itversity_sms_db', 'public', 'users_range_part_202003'),
 ('itversity_sms_db', 'public', 'users_range_part_202004'),
 ('itversity_sms_db', 'public', 'users_range_part_202005'),
 ('itversity_sms_db', 'public', 'users_range_part_202006'),
```

(continues on next page)

(continued from previous page)

```
('itversity_sms_db', 'public', 'users_range_part_202007'),
('itversity_sms_db', 'public', 'users_range_part_202008'),
('itversity_sms_db', 'public', 'users_range_part_202009'),
('itversity_sms_db', 'public', 'users_range_part_202010'),
('itversity_sms_db', 'public', 'users_range_part_202011'),
('itversity_sms_db', 'public', 'users_range_part_202012'),
('itversity_sms_db', 'public', 'users_range_part_default')]
```

6.5.8 Hash Partitioning

Let us understand how we can take care of Hash partitioning of tables.

- It is primarily used to create partitions based up on modulus and reminder.
- Here are the steps involved in creating table using hash partitioning strategy.
  - Create table using `PARTITION BY HASH`
  - Add default and remainder specific partitions based up on modulus.
  - Validate by inserting data into the table
- We can detach as well as drop the partitions from the table.
- Hash partitioning is typically done on sparse columns such as `user_id`.
- If we want to use hash partitioning on more than one tables with common key, we typically partition all the tables using same key.

Create Partitioned Table

Let us create partitioned table with name `users_hash_part`.

- It contains same columns as `users`.
- We will partition the table based up on `user_id` field.
- We will create one partition for each reminder with modulus 8.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
%sql DROP TABLE IF EXISTS users_hash_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id)
) PARTITION BY HASH(user_id)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

Note: We will not be able to insert the data until we add at least one partition.

6.5.9 Managing Partitions - Hash

Let us understand how to manage partitions using table `users_hash_part` which is partitioned using **hash**.

- We would like to divide our data into 8 hash buckets.
- While adding partitions for **hash partitioned table**, we need to specify modulus and remainder.
- For each and every record inserted, following will happen for the column specified as partitioned key.
  - A hash will be computed. Hash is nothing but an integer.
  - The integer generated will be divided by the value specified in **modulus**.
  - Based up on the remainder, the record will be inserted into corresponding partition.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

Error: We cannot have a default partition for hash partitioned table.

```
%%sql

CREATE TABLE users_hash_part_default
PARTITION OF users_hash_part DEFAULT
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
(psycopg2.errors.InvalidTableDefinition) a hash-partitioned table may not have a
↪default partition

[SQL: CREATE TABLE users_hash_part_default PARTITION OF users_hash_part DEFAULT]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

Note: Let us add partitions using modulus as 8. For each remainder between 0 to 7. we need to add a partition.

```
%%sql

CREATE TABLE users_hash_part_0_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 0)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_1_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 1)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_2_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 2)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_3_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 3)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_4_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 4)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_5_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 5)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_6_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 6)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_hash_part_7_of_8
PARTITION OF users_hash_part
FOR VALUES WITH (modulus 8, remainder 7)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO users_hash_part
    (user_first_name, user_last_name, user_email_id, created_dt)
VALUES
    ('Scott', 'Tiger', 'scott@tiger.com', '2018-10-01'),
    ('Donald', 'Duck', 'donald@duck.com', '2019-02-10'),
    ('Mickey', 'Mouse', 'mickey@mouse.com', '2017-06-22')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[]
```

Note: user_id is populated by sequence. The hash of every sequence generated integer will be divided by modulus (which is 8) and based up on the remainder data will be inserted into corresponding partition.

```
%%sql

SELECT * FROM users_hash_part
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
3 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2018,
↪ 10, 1), datetime.datetime(2020, 11, 24, 12, 13, 6, 353736)),
 (3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2017, 6, 22), datetime.datetime(2020, 11, 24, 12, 13, 6, 353736)),
 (2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2019,
↪ 2, 10), datetime.datetime(2020, 11, 24, 12, 13, 6, 353736))]
```

```
%%sql

SELECT * FROM users_hash_part_0_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(1, 'Scott', 'Tiger', 'scott@tiger.com', False, None, 'U', False, datetime.date(2018,
↪ 10, 1), datetime.datetime(2020, 11, 24, 12, 13, 6, 353736))]
```

```
%%sql

SELECT * FROM users_hash_part_1_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(3, 'Mickey', 'Mouse', 'mickey@mouse.com', False, None, 'U', False, datetime.
↪date(2017, 6, 22), datetime.datetime(2020, 11, 24, 12, 13, 6, 353736))]
```

```
%%sql

SELECT * FROM users_hash_part_2_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
1 rows affected.
```

```
[(2, 'Donald', 'Duck', 'donald@duck.com', False, None, 'U', False, datetime.date(2019,
↪ 2, 10), datetime.datetime(2020, 11, 24, 12, 13, 6, 353736))]
```

```
%%sql

SELECT * FROM users_hash_part_3_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_hash_part_4_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_hash_part_5_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_hash_part_6_of_8
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM users_hash_part_7_of_8
```

```
* postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
0 rows affected.
```

```
[]
```

6.5.10 Usage Scenarios

Let us go through some of the usage scenarios with respect to partitioning.

- It is typically used to manage large tables so that the tables does not grow abnormally over a period of time.
- Partitioning is quite often used on top of log tables, reporting tables etc.
- If a log table is partitioned and if we want to have data for 7 years, partitions older than 7 years can be quickly dropped.
- Dropping partitions to clean up huge chunk of data is much faster compared to running delete command on non partitioned table.
- For tables like orders with limited set of statuses, we often use list partitioning based up on the status. It can be 2 partitions (CLOSED orders and ACTIVE orders) or separate partition for each status.
  - As most of the operations will be on **Active Orders**, this approach can significantly improve the performance.
- In case of log tables, where we might want to retain data for several years, we tend to use range partition on date column. If we use list partition, then we might end up in duplication of data unnecessarily.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

Note: Monthly partition using list. We need to have additional column to store the month to use list partitioning strategy.

```
%%sql

DROP TABLE IF EXISTS users_mthly
```

```
* postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_mthly (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    created_mnth INT,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (created_mnth, user_id)
) PARTITION BY LIST(created_mnth)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_mthly_201601
PARTITION OF users_mthly
FOR VALUES IN (201601)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_mthly_201602
PARTITION OF users_mthly
FOR VALUES IN (201602)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

Note: Monthly partition using range. Partition strategy is defined on top of **created_dt**. No additional column is required.

```
%%sql

DROP TABLE IF EXISTS users_mthly
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_mthly (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (created_dt, user_id)
) PARTITION BY RANGE(created_dt)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_mthly_201601
PARTITION OF users_mthly
FOR VALUES FROM ('2016-01-01') TO ('2016-01-31')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_mthly_201602
PARTITION OF users_mthly
FOR VALUES FROM ('2016-02-01') TO ('2016-02-29')
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

6.5.11 Sub Partitioning

We can have sub partitions created with different permutations and combinations. Sub Partitioning is also known as nested partitioning.

- List - List
- List - Range and others.

Note: Try different sub-partitioning strategies based up on your requirements.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

```
env: DATABASE_URL=postgresql://itversity_sms_user:sms_password@localhost:5432/
↪itversity_sms_db
```

List - List Partitioning

Let us understand how we can create table using list - list sub partitioning. We would like to have main partition per year and then sub partitions per quarter.

- Create table `users_qtly` with `PARTITION BY LIST` with `created_year`.
- Create tables for yearly partitions with `PARTITION BY LIST` with `created_month`.
- Create tables for quarterly partitions with list of values using `FOR VALUES IN`.

```
%%sql

DROP TABLE IF EXISTS users_qtly
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
```

(continues on next page)

(continued from previous page)

```
    created_year INT,
    created_mnth INT,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (created_year, created_mnth, user_id)
) PARTITION BY LIST(created_year)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly_2016
PARTITION OF users_qtly
FOR VALUES IN (2016)
PARTITION BY LIST (created_mnth)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly_2016q1
PARTITION OF users_qtly_2016
FOR VALUES IN (1, 2, 3)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly_2016q2
PARTITION OF users_qtly_2016
FOR VALUES IN (4, 5, 6)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

List - Range Partitioning

Let us understand how we can create table using list - Range sub partitioning using same example as before (partitioning by year and then by quarter).

- Create table with `PARTITION BY LIST` with `created_year`.
- Create tables for yearly partitions with `PARTITION BY RANGE` with `created_month`.
- Create tables for quarterly partitions with the range of values using `FOR VALUES FROM (lower_bound) TO (upper_bound)`.

```
%%sql

DROP TABLE IF EXISTS users_qtly
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly (
    user_id SERIAL,
    user_first_name VARCHAR(30) NOT NULL,
    user_last_name VARCHAR(30) NOT NULL,
    user_email_id VARCHAR(50) NOT NULL,
    user_email_validated BOOLEAN DEFAULT FALSE,
    user_password VARCHAR(200),
    user_role VARCHAR(1) NOT NULL DEFAULT 'U', --U and A
    is_active BOOLEAN DEFAULT FALSE,
    created_dt DATE DEFAULT CURRENT_DATE,
    created_year INT,
    created_mnth INT,
    last_updated_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (created_year, created_mnth, user_id)
) PARTITION BY LIST(created_year)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly_2016
PARTITION OF users_qtly
FOR VALUES IN (2016)
PARTITION BY RANGE (created_mnth)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly_2016q1
PARTITION OF users_qtly_2016
FOR VALUES FROM (1) TO (3)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users_qtly_2016q2
PARTITION OF users_qtly_2016
FOR VALUES FROM (4) TO (6)
```

```
 * postgresql://itversity_sms_user:***@localhost:5432/itversity_sms_db
Done.
```

```
[]
```

6.5.12 Exercises - Partitioning Tables

Here is the exercise to get comfort with partitioning. We will be using range partitioning.

- Use retail database. Make sure **orders** table already exists.
- You can reset the database by running these commands.
- Connect to retail database.

```
psql -U itversity_retail_user \
  -h localhost \
  -p 5432 \
  -d itversity_retail_db \
  -W
```

- Run these commands or scripts to reset the tables. It will take care of recreating **orders** table.

```
DROP TABLE IF EXISTS order_items;
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS products;
DROP TABLE IF EXISTS categories;
DROP TABLE IF EXISTS departments;


\i /data/retail_db/create_db_tables_pg.sql

\i /data/retail_db/load_db_tables_pg.sql
```

Exercise 1

Create table **orders_part** with the same columns as orders.

- Partition the table by month using range partitioning on **order_date**.
- Add 14 partitions - 13 based up on the data and 1 default. Here is the naming convention.
  - Default - orders_part_default
  - Partition for 2014 January - orders_part_201401

Exercise 2

Let us load and validate data in the partitioned table.

- Load the data from **orders** into **orders_part**.
- Get count on **orders_part** as well as all the 14 partitions. You should get 0 for default partition and all the records should be distributed using the other 13 partitions.

6.6 Pre-Defined Functions

Let us go through the pre-defined functions available in Postgresql.

- Overview of Pre-Defined Functions
- String Manipulation Functions
- Date Manipulation Functions
- Overview of Numeric Functions
- Data Type Conversion
- Handling Null Values
- Using CASE and WHEN
- Exercises - Pre-Defined Functions

Here are the key objectives of this section.

- How to use official documentation of Postgres to get syntax and symantecs of the pre-defined functions?
- Understand different categories of functions
- How to use functions effectively using real world examples?
- How to manipulate strings and dates?
- How to deal with nulls, convert data types etc?
- Self evaluate by solving the exercises by using multiple functions in tandem.

6.6.1 Overview of Pre-Defined Functions

Like any RDBMS, Postgres provides robust set of pre-defined functions to come up with solutions quickly as per the business requirements. There are many functions, but we will see the most common ones here.

- Following are the categories of functions that are more commonly used.
  - String Manipulation
  - Date Manipulation
  - Numeric Functions
  - Type Conversion Functions
  - CASE and WHEN
  - and more
- One can go to the official documentation from Postgres website.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql SELECT * FROM information_schema.routines LIMIT 10
```

```
10 rows affected.
```

```
[('itversity_retail_db', 'pg_catalog', 'boolin_1242', 'itversity_retail_db', 'pg_
↪catalog', 'boolin', 'FUNCTION', None, None, None, None, None, None, 'boolean', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bool', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'boolout_1243', 'itversity_retail_db', 'pg_
↪catalog', 'boolout', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'byteain_1244', 'itversity_retail_db', 'pg_
↪catalog', 'byteain', 'FUNCTION', None, None, None, None, None, None, 'bytea', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'byteaout_31', 'itversity_retail_db', 'pg_
↪catalog', 'byteaout', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
```

(continues on next page)

(continued from previous page)

```
 ('itversity_retail_db', 'pg_catalog', 'charin_1245', 'itversity_retail_db', 'pg_
↪catalog', 'charin', 'FUNCTION', None, None, None, None, None, None, '"char"', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'char', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'charout_33', 'itversity_retail_db', 'pg_
↪catalog', 'charout', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'namein_34', 'itversity_retail_db', 'pg_catalog
↪', 'namein', 'FUNCTION', None, None, None, None, None, None, 'name', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, 'itversity_
↪retail_db', 'pg_catalog', 'name', None, None, None, None, '0', 'EXTERNAL', None,␣
↪None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0, None, None,
↪'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'nameout_35', 'itversity_retail_db', 'pg_
↪catalog', 'nameout', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'int2in_38', 'itversity_retail_db', 'pg_catalog
↪', 'int2in', 'FUNCTION', None, None, None, None, None, None, 'smallint', None, None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, 'itversity_
↪retail_db', 'pg_catalog', 'int2', None, None, None, None, '0', 'EXTERNAL', None,␣
↪None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0, None, None,
↪'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'int2out_39', 'itversity_retail_db', 'pg_
↪catalog', 'int2out', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None)]
```

```
%%sql

SELECT * FROM information_schema.routines
WHERE routine_name ~ 'str'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
36 rows affected.
```

```
[('itversity_retail_db', 'pg_catalog', 'btvarstrequalimage_5050', 'itversity_retail_db
↪', 'pg_catalog', 'btvarstrequalimage', 'FUNCTION', None, None, None, None, None,␣
↪None, 'boolean', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'bool', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'string_to_array_394', 'itversity_retail_db',
↪'pg_catalog', 'string_to_array', 'FUNCTION', None, None, None, None, None, None,
↪'ARRAY', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', '_text', None, None, None, None, '0
↪', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'NO', None,
↪'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'array_to_string_395', 'itversity_retail_db',
↪'pg_catalog', 'array_to_string', 'FUNCTION', None, None, None, None, None, None,
↪'text', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0
↪', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES', None,
↪'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'string_to_array_376', 'itversity_retail_db',
↪'pg_catalog', 'string_to_array', 'FUNCTION', None, None, None, None, None, None,
↪'ARRAY', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', '_text', None, None, None, None, '0
↪', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'NO', None,
↪'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'array_to_string_384', 'itversity_retail_db',
↪'pg_catalog', 'array_to_string', 'FUNCTION', None, None, None, None, None, None,
↪'text', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0
↪', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'NO', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'strpos_868', 'itversity_retail_db', 'pg_
↪catalog', 'strpos', 'FUNCTION', None, None, None, None, None, None, 'integer', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'int4', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'substr_877', 'itversity_retail_db', 'pg_
↪catalog', 'substr', 'FUNCTION', None, None, None, None, None, None, 'text', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'substr_883', 'itversity_retail_db', 'pg_
↪catalog', 'substr', 'FUNCTION', None, None, None, None, None, None, 'text', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
```

(continues on next page)

(continued from previous page)

```
('itversity_retail_db', 'pg_catalog', 'substring_936', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'text', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'substring_937', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'text', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'pg_get_partition_constraintdef_3408',
↪'itversity_retail_db', 'pg_catalog', 'pg_get_partition_constraintdef', 'FUNCTION',␣
↪None, None, None, None, None, None, 'text', None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, 'itversity_retail_db', 'pg_catalog',
↪ 'text', None, None, None, None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL',
↪ 'NO', 'MODIFIES', 'YES', None, 'YES', 0, None, None, 'INVOKER', None, None, None,
↪'NO', None, None, None, 'NO', None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None),
('itversity_retail_db', 'pg_catalog', 'pg_get_constraintdef_1387', 'itversity_retail_
↪db', 'pg_catalog', 'pg_get_constraintdef', 'FUNCTION', None, None, None, None, None,
↪ None, 'text', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'text', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'RI_FKey_restrict_del_1648', 'itversity_retail_
↪db', 'pg_catalog', 'RI_FKey_restrict_del', 'FUNCTION', None, None, None, None, None,
↪ None, 'trigger', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'trigger', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'RI_FKey_restrict_upd_1649', 'itversity_retail_
↪db', 'pg_catalog', 'RI_FKey_restrict_upd', 'FUNCTION', None, None, None, None, None,
↪ None, 'trigger', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'trigger', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'substring_1680', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'bit', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bit', None, None, None, None, '0', 'EXTERNAL',
↪ None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0, None,␣
↪None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None),
```

(continues on next page)

(continued from previous page)

```
('itversity_retail_db', 'pg_catalog', 'substring_1699', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'bit', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bit', None, None, None, None, '0', 'EXTERNAL',
↪ None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0, None,␣
↪None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'string_agg_transfn_3535', 'itversity_retail_db
↪', 'pg_catalog', 'string_agg_transfn', 'FUNCTION', None, None, None, None, None,␣
↪None, 'internal', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'internal', None, None, None,
↪ None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'NO',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'string_agg_finalfn_3536', 'itversity_retail_db
↪', 'pg_catalog', 'string_agg_finalfn', 'FUNCTION', None, None, None, None, None,␣
↪None, 'text', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'text', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'NO',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'string_agg_3538', 'itversity_retail_db', 'pg_
↪catalog', 'string_agg', None, None, None, None, None, None, None, 'text', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'NO', None, 'YES', 0, None,
↪ None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'bytea_string_agg_transfn_3543', 'itversity_
↪retail_db', 'pg_catalog', 'bytea_string_agg_transfn', 'FUNCTION', None, None, None,␣
↪None, None, None, 'internal', None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'internal', None,
↪ None, None, None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES',
↪'MODIFIES', 'NO', None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO',␣
↪None, None, None, 'NO', None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'bytea_string_agg_finalfn_3544', 'itversity_
↪retail_db', 'pg_catalog', 'bytea_string_agg_finalfn', 'FUNCTION', None, None, None,␣
↪None, None, None, 'bytea', None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, 'itversity_retail_db', 'pg_catalog', 'bytea', None,␣
↪None, None, None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES',
↪'MODIFIES', 'NO', None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO',␣
↪None, None, None, 'NO', None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'string_agg_3545', 'itversity_retail_db', 'pg_
↪catalog', 'string_agg', None, None, None, None, None, None, None, 'bytea', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'NO', None, 'YES', 0, None,
↪ None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'substring_2012', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'bytea',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
```

(continues on next page)

(continued from previous page)

```
('itversity_retail_db', 'pg_catalog', 'substring_2013', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'bytea',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'substr_2085', 'itversity_retail_db', 'pg_
↪catalog', 'substr', 'FUNCTION', None, None, None, None, None, None, 'bytea', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'substr_2086', 'itversity_retail_db', 'pg_
↪catalog', 'substr', 'FUNCTION', None, None, None, None, None, None, 'bytea', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'substring_2073', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'text', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'EXTERNAL
↪', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0,␣
↪None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'substring_2074', 'itversity_retail_db', 'pg_
↪catalog', 'substring', 'FUNCTION', None, None, None, None, None, None, 'text', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'text', None, None, None, None, '0', 'SQL',␣
↪None, None, 'SQL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES', 0, None, None,
↪'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'cstring_in_2292', 'itversity_retail_db', 'pg_
↪catalog', 'cstring_in', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'cstring_out_2293', 'itversity_retail_db', 'pg_
↪catalog', 'cstring_out', 'FUNCTION', None, None, None, None, None, None, 'cstring',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
 ('itversity_retail_db', 'pg_catalog', 'cstring_recv_2500', 'itversity_retail_db',
↪'pg_catalog', 'cstring_recv', 'FUNCTION', None, None, None, None, None, None,
↪'cstring', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', 'cstring', None, None, None, None,
↪'0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES', None,
↪'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None),
```

(continues on next page)

(continued from previous page)

```
('itversity_retail_db', 'pg_catalog', 'cstring_send_2501', 'itversity_retail_db',
↪'pg_catalog', 'cstring_send', 'FUNCTION', None, None, None, None, None, None, 'bytea
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, 'itversity_retail_db', 'pg_catalog', 'bytea', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES', None, 'YES',
↪ 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'pg_get_constraintdef_2508', 'itversity_retail_
↪db', 'pg_catalog', 'pg_get_constraintdef', 'FUNCTION', None, None, None, None, None,
↪ None, 'text', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'text', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'NO', 'MODIFIES', 'YES',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'json_strip_nulls_3261', 'itversity_retail_db',
↪ 'pg_catalog', 'json_strip_nulls', 'FUNCTION', None, None, None, None, None, None,
↪'json', None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, 'itversity_retail_db', 'pg_catalog', 'json', None, None, None, None, '0
↪', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None,
↪'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO',␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'strip_3623', 'itversity_retail_db', 'pg_
↪catalog', 'strip', 'FUNCTION', None, None, None, None, None, None, 'tsvector', None,
↪ None, None, None, None, None, None, None, None, None, None, None, None, None,
↪'itversity_retail_db', 'pg_catalog', 'tsvector', None, None, None, None, '0',
↪'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES', None, 'YES
↪', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO', None,␣
↪None, None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None),
('itversity_retail_db', 'pg_catalog', 'jsonb_strip_nulls_3262', 'itversity_retail_db
↪', 'pg_catalog', 'jsonb_strip_nulls', 'FUNCTION', None, None, None, None, None,␣
↪None, 'jsonb', None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, 'itversity_retail_db', 'pg_catalog', 'jsonb', None, None, None,␣
↪None, '0', 'EXTERNAL', None, None, 'INTERNAL', 'GENERAL', 'YES', 'MODIFIES', 'YES',␣
↪None, 'YES', 0, None, None, 'INVOKER', None, None, None, 'NO', None, None, None, 'NO
↪', None, None, None, None, None, None, None, None, None, None, None, None, None,␣
↪None, None, None, None, None, None, None, None, None, None, None)]
```

```
%%sql

SELECT substring('Thomas' from 2 for 3)
```

```
* postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('hom',)]
```

```
%%sql

SELECT substring('Thomas', 2, 3)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('hom',)]
```

6.6.2 String Manipulation Functions

We use string manipulation functions quite extensively. Here are some of the important functions which we typically use.

- Case Conversion - `lower`, `upper`, `initcap`
- Getting size of the column value - `length`
- Extracting Data - `substr` and `split_part`
- Trimming and Padding functions - `trim`, `rtrim`, `ltrim`, `rpad` and `lpad`
- Reversing strings - `reverse`
- Concatenating multiple strings `concat` and `concat_ws`

Case Conversion and Length

Let us understand how to perform case conversion of a string and also get length of a string.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

- Case Conversion Functions - `lower`, `upper`, `initcap`

```
%%sql

SELECT lower('hEllo wOrlD') AS lower_result,
    upper('hEllo wOrlD') AS upper_result,
    initcap('hEllo wOrlD') AS initcap_result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('hello world', 'HELLO WORLD', 'Hello World')]
```

- Getting length - `length`

```
%%sql

SELECT length('hEllo wOrlD') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(11,)]
```

Let us see how to use these functions on top of the table. We will use orders table which was loaded as part of last section.

- order_status for all the orders is in upper case and we will convert every thing to lower case.

```
%%sql

SELECT * FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'COMPLETE'),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'PENDING'),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'CLOSED'),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'CLOSED'),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'COMPLETE'),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'ON_HOLD'),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'COMPLETE'),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'COMPLETE'),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'COMPLETE'),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'CLOSED')]
```

```
%%sql

SELECT order_id, order_date, order_customer_id,
    lower(order_status) AS order_status,
    length(order_status) AS order_status_length
FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'complete', 8),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'pending', 7),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'closed', 6),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'closed', 6),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'complete', 8),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'on_hold', 7),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'complete', 8),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'complete', 8),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'complete', 8),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'closed', 6)]
```

Extracting Data - substr and split_part

Let us understand how to extract data from strings using `substr`/`substring` as well as `split_part`.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

- We can extract sub string from main string using `substr` or `substring` position and length.
- For example, get first 4 characters from date to get year or get last 4 characters from fixed length unique id.
- `substring` have broader options (regular expression) and also can be used with different styles (using keywords such as `FROM`, `FOR`).
- Unlike in other relational databases, we cannot pass negative integers to `substr` or `substring` to get the information from right. We need to use functions like `right` instead.

```
%%sql

SELECT substr('2013-07-25 00:00:00.0', 1, 4) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013',)]
```

```
%%sql

SELECT substring('2013-07-25 00:00:00.0', 1, 4) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013',)]
```

```
%%sql

SELECT substring('2013-07-25 00:00:00.0' FROM 1 FOR 4) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013',)]
```

```
%%sql

SELECT substring('2013-07-25 00:00:00.0', 6, 2) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('07',)]
```

```
%%sql

SELECT substring('2013-07-25 00:00:00.0', 9, 2) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('25',)]
```

```
%%sql

SELECT substring('2013-07-25 00:00:00.0' from 12) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('00:00:00.0',)]
```

```
%%sql

SELECT substr('2013-07-25 00:00:00.0', 12) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('00:00:00.0',)]
```

```
%%sql

SELECT right('123 456 7890', 4) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('7890',)]
```

```
%%sql

SELECT left('123 456 7890', 3) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('123',)]
```

Note: We can also use combination of `substring` and `length` like below to get last 4 digits or characters from a string.

```
%%sql

SELECT substring('123 456 7890' FROM length('123 456 7890') - 4) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(' 7890',)]
```

```
%%sql

SELECT substring('123 456 7890' FROM '....$') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('7890',)]
```

Note: Getting first 3 characters or digits as well as last 4 characters or digits using `substring`. However, this works only when the strings are of fixed length.

```
%%sql

WITH unique_ids AS (
    SELECT '241-80-7115' AS unique_id UNION
    SELECT '694-30-6851' UNION
    SELECT '586-92-5361' UNION
    SELECT '884-65-284' UNION
    SELECT '876-99-585' UNION
    SELECT '831-59-5593' UNION
    SELECT '399-88-3617' UNION
    SELECT '733-17-4217' UNION
    SELECT '873-68-9778' UNION
    SELECT '48'
) SELECT unique_id,
    substring(unique_id FROM 1 FOR 3) AS unique_id_first3,
    substring(unique_id FROM '....$') AS unique_id_last4
FROM unique_ids
ORDER BY unique_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('241-80-7115', '241', '7115'),
 ('399-88-3617', '399', '3617'),
 ('48', '48', None),
 ('586-92-5361', '586', '5361'),
 ('694-30-6851', '694', '6851'),
```

(continues on next page)

(continued from previous page)

```
('733-17-4217', '733', '4217'),
('831-59-5593', '831', '5593'),
('873-68-9778', '873', '9778'),
('876-99-585', '876', '-585'),
('884-65-284', '884', '-284')]
```

- Let us see how we can extract date part from order_date of orders.

```
%%sql

SELECT * FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'COMPLETE'),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'PENDING'),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'CLOSED'),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'CLOSED'),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'COMPLETE'),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'ON_HOLD'),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'COMPLETE'),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'COMPLETE'),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'COMPLETE'),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'CLOSED')]
```

```
%%sql

SELECT order_id,
    substr(order_date::varchar, 1, 10) AS order_date,
    order_customer_id,
    order_status
FROM orders
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, '2013-07-30', 10118, 'COMPLETE'),
 (4068, '2013-08-17', 12293, 'PENDING'),
 (5881, '2013-08-30', 3715, 'CLOSED'),
 (7564, '2013-09-09', 8648, 'CLOSED'),
 (8766, '2013-09-18', 855, 'COMPLETE'),
 (8926, '2013-09-19', 10517, 'ON_HOLD'),
 (9290, '2013-09-21', 11879, 'COMPLETE'),
 (9793, '2013-09-24', 9809, 'COMPLETE'),
 (9816, '2013-09-24', 1753, 'COMPLETE'),
 (14047, '2013-10-20', 6473, 'CLOSED')]
```

Let us understand how to extract the information from the string where there is a delimiter.

- `split_part` can be used to split a string using delimiter and extract the information.
- If there is no data in a given position after splitting, it will be represented as empty string ‘’.

```
%%sql

SELECT split_part('2013-07-25', '-', 1) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013',)]
```

```
%%sql

WITH addresses AS (
    SELECT '593 Fair Oaks Pass, Frankfort, Kentucky, 40618' AS address UNION
    SELECT ', Vancouver, Washington, 98687' UNION
    SELECT '83047 Glacier Hill Circle, Sacramento, California, 94237' UNION
    SELECT '935 Columbus Junction, Cincinnati, Ohio, 45213' UNION
    SELECT '03010 Nevada Crossing, El Paso, Texas, 88579' UNION
    SELECT '9 Dunning Circle, , Arizona, 85271' UNION
    SELECT '96 Fair Oaks Way, Decatur, Illinois, 62525' UNION
    SELECT '999 Caliangt Avenue, Greenville, South Carolina, 29615' UNION
    SELECT '2 Saint Paul Trail, Bridgeport, , 06673' UNION
    SELECT '3 Reindahl Center, Ogden, Utah'
) SELECT split_part(address, ', ', 1) street,
    split_part(address, ', ', 2) city,
    split_part(address, ', ', 3) state,
    split_part(address, ', ', 4) postal_code
FROM addresses
ORDER BY postal_code
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('3 Reindahl Center', 'Ogden', 'Utah', ''),
 ('2 Saint Paul Trail', 'Bridgeport', '', '06673'),
 ('999 Caliangt Avenue', 'Greenville', 'South Carolina', '29615'),
 ('593 Fair Oaks Pass', 'Frankfort', 'Kentucky', '40618'),
 ('935 Columbus Junction', 'Cincinnati', 'Ohio', '45213'),
 ('96 Fair Oaks Way', 'Decatur', 'Illinois', '62525'),
 ('9 Dunning Circle', '', 'Arizona', '85271'),
 ('03010 Nevada Crossing', 'El Paso', 'Texas', '88579'),
 ('83047 Glacier Hill Circle', 'Sacramento', 'California', '94237'),
 ('', 'Vancouver', 'Washington', '98687')]
```

```
%%sql

WITH addresses AS (
    SELECT '593 Fair Oaks Pass, Frankfort, Kentucky, 40618' AS address UNION
    SELECT ', Vancouver, Washington, 98687' UNION
    SELECT '83047 Glacier Hill Circle, Sacramento, California, 94237' UNION
    SELECT '935 Columbus Junction, Cincinnati, Ohio, 45213' UNION
    SELECT '03010 Nevada Crossing, El Paso, Texas, 88579' UNION
    SELECT '9 Dunning Circle, , Arizona, 85271' UNION
    SELECT '96 Fair Oaks Way, Decatur, Illinois, 62525' UNION
    SELECT '999 Caliangt Avenue, Greenville, South Carolina, 29615' UNION
    SELECT '2 Saint Paul Trail, Bridgeport, , 06673' UNION
```

(continues on next page)

(continued from previous page)

```
    SELECT '3 Reindahl Center, Ogden, Utah'
) SELECT split_part(address, ', ', 1) street,
    split_part(address, ', ', 2) city,
    split_part(address, ', ', 3) state,
    split_part(address, ', ', 4) postal_code
FROM addresses
WHERE split_part(address, ', ', 1) = ''
ORDER BY postal_code
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('', 'Vancouver', 'Washington', '98687')]
```

```
%%sql

WITH unique_ids AS (
    SELECT '241-80-7115' AS unique_id UNION
    SELECT '694-30-6851' UNION
    SELECT '586-92-5361' UNION
    SELECT '884-65-284' UNION
    SELECT '876-99-585' UNION
    SELECT '831-59-5593' UNION
    SELECT '399-88-3617' UNION
    SELECT '733-17-4217' UNION
    SELECT '873-68-9778' UNION
    SELECT '480-69-032'
) SELECT unique_id,
    substring(unique_id FROM 1 FOR 3) AS unique_id_first3,
    substring(unique_id FROM '....$') AS unique_id_last4,
    CASE WHEN length(split_part(unique_id, '-', 3)) = 4
        THEN split_part(unique_id, '-', 3)
        ELSE 'Invalid'
    END AS unique_id_last
FROM unique_ids
ORDER BY unique_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('241-80-7115', '241', '7115', '7115'),
 ('399-88-3617', '399', '3617', '3617'),
 ('480-69-032', '480', '-032', 'Invalid'),
 ('586-92-5361', '586', '5361', '5361'),
 ('694-30-6851', '694', '6851', '6851'),
 ('733-17-4217', '733', '4217', '4217'),
 ('831-59-5593', '831', '5593', '5593'),
 ('873-68-9778', '873', '9778', '9778'),
 ('876-99-585', '876', '-585', 'Invalid'),
 ('884-65-284', '884', '-284', 'Invalid')]
```

Using position or strpos

At times we might want to get the position of a substring in a main string. For example, we might want to check whether email ids have @ in them. We can use functions such as `position` or `strpos`.

```
%%sql

SELECT position('@' IN 'it@versity.com'),
    position('@' IN 'itversity.com')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(3, 0)]
```

```
%%sql

SELECT strpos('it@versity.com', '@'),
    strpos('itversity.com', '@')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(3, 0)]
```

```
%%sql

WITH email_ids AS (
    SELECT 'bsellan0@yellowbook.com' AS email_id UNION
    SELECT 'rstelljes1@illinois.edu' UNION
    SELECT 'mmalarkey2@webeden.co.uk' UNION
    SELECT 'emussared3@redcross.org' UNION
    SELECT 'livashin4@bloglovin.com' UNION
    SELECT 'gkeach5@cbc.ca' UNION
    SELECT 'emasham6@xing.com' UNION
    SELECT 'rcobbald7@house.gov' UNION
    SELECT 'rdrohan8@washingtonpost.com' UNION
    SELECT 'aebben9@arstechnica.com'
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.SyntaxError) syntax error at end of input
LINE 12: )
          ^


[SQL: WITH email_ids AS (
    SELECT 'bsellan0@yellowbook.com' AS email_id UNION
    SELECT 'rstelljes1@illinois.edu' UNION
    SELECT 'mmalarkey2@webeden.co.uk' UNION
    SELECT 'emussared3@redcross.org' UNION
    SELECT 'livashin4@bloglovin.com' UNION
    SELECT 'gkeach5@cbc.ca' UNION
    SELECT 'emasham6@xing.com' UNION
    SELECT 'rcobbald7@house.gov' UNION
    SELECT 'rdrohan8@washingtonpost.com' UNION
```

(continues on next page)

(continued from previous page)

```
    SELECT 'aebben9@arstechnica.com'
)]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

Trimming and Padding Functions

Let us understand how to trim or remove leading and/or trailing spaces in a string.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

- `ltrim` is used to remove the spaces on the left side of the string.
- `rtrim` is used to remove the spaces on the right side of the string.
- `trim` is used to remove the spaces on both sides of the string.

```
%%sql

SELECT ltrim('     Hello World') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('Hello World',)]
```

```
%%sql

SELECT rtrim('     Hello World       ') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('     Hello World',)]
```

```
%%sql

SELECT length(trim('     Hello World       ')) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(11,)]
```

```
%%sql

SELECT ltrim('----Hello World----', '-') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('Hello World----',)]
```

```
%%sql

SELECT rtrim('----Hello World----', '-') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('----Hello World',)]
```

```
%%sql

SELECT trim('----Hello World----', '-') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('Hello World',)]
```

Let us understand how to use padding to pad characters to a string.

- Let us assume that there are 3 fields - year, month and date which are of type integer.
- If we have to concatenate all the 3 fields and create a date, we might have to pad month and date with 0.
- `lpad` is used more often than `rpad` especially when we try to build the date from separate columns.

```
%%sql

SELECT 2013 AS year, 7 AS month, 25 AS myDate
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2013, 7, 25)]
```

```
%%sql

SELECT lpad(7::varchar, 2, '0') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('07',)]
```

```
%%sql

SELECT lpad(10::varchar, 2, '0') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('10',)]
```

```
%%sql

SELECT lpad(100::varchar, 2, '0') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('10',)]
```

Reverse and Concatenating multiple strings

Let us understand how to reverse a string as well as concatenate multiple strings.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

- We can use `reverse` to reverse a string.
- We can concatenate multiple strings using `concat` and `concat_ws`.
- `concat_ws` is typically used if we want to have the same string between all the strings that are being concatenated.

```
%%sql

SELECT reverse('Hello World') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('dlroW olleH',)]
```

```
%%sql

SELECT concat('Hello ', 'World') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('Hello World',)]
```

```
%%sql

SELECT concat('Order Status is ', order_status) AS result
FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('Order Status is COMPLETE',),
 ('Order Status is PENDING',),
 ('Order Status is CLOSED',),
 ('Order Status is CLOSED',),
 ('Order Status is COMPLETE',),
 ('Order Status is ON_HOLD',),
 ('Order Status is COMPLETE',),
 ('Order Status is COMPLETE',),
 ('Order Status is COMPLETE',),
 ('Order Status is CLOSED',)]
```

```
%%sql

SELECT * FROM (SELECT 2013 AS year, 7 AS month, 25 AS myDate) q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2013, 7, 25)]
```

```
%%sql

SELECT concat(year, '-', lpad(month::varchar, 2, '0'), '-',
              lpad(myDate::varchar, 2, '0')) AS order_date
FROM
    (SELECT 2013 AS year, 7 AS month, 25 AS myDate) q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013-07-25',)]
```

```
%%sql

SELECT concat_ws('-', year, lpad(month::varchar, 2, '0'),
              lpad(myDate::varchar, 2, '0')) AS order_date
FROM
    (SELECT 2013 AS year, 7 AS month, 25 AS myDate) q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013-07-25',)]
```

String Replacement

Let us go through the details related to string replacement.

- `replace` can be used to replace a sub string with in a string with another string.
- `overlay` can be used to replace a sub string with in a string by position with another string.
- `translate` can be used to replace individual characters with other characters.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT replace('Hello World', 'alo', 'ello') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('Hello World',)]
```

```
%%sql

SELECT overlay('Halo World' PLACING 'ello' FROM 2 FOR 3) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('Hello World',)]
```

```
%%sql

WITH unique_ids AS (
    SELECT '241-80-7115' AS unique_id UNION
    SELECT '694-30-6851' UNION
    SELECT '586-92-5361' UNION
    SELECT '884-65-2844' UNION
    SELECT '876-99-5856' UNION
```

(continues on next page)

(continued from previous page)

```
    SELECT '831-59-5593' UNION
    SELECT '399-88-3617' UNION
    SELECT '733-17-4217' UNION
    SELECT '873-68-9778' UNION
    SELECT '487-21-9802'
) SELECT unique_id,
    replace(unique_id, '-', ' ') AS unique_id_replaced,
    translate(unique_id, '-', ' ') AS unique_id_translated,
    overlay(unique_id PLACING ' ' FROM 4 FOR 1) AS unique_id_overlaid
FROM unique_ids
ORDER BY unique_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('241-80-7115', '241 80 7115', '241 80 7115', '241 80-7115'),
 ('399-88-3617', '399 88 3617', '399 88 3617', '399 88-3617'),
 ('487-21-9802', '487 21 9802', '487 21 9802', '487 21-9802'),
 ('586-92-5361', '586 92 5361', '586 92 5361', '586 92-5361'),
 ('694-30-6851', '694 30 6851', '694 30 6851', '694 30-6851'),
 ('733-17-4217', '733 17 4217', '733 17 4217', '733 17-4217'),
 ('831-59-5593', '831 59 5593', '831 59 5593', '831 59-5593'),
 ('873-68-9778', '873 68 9778', '873 68 9778', '873 68-9778'),
 ('876-99-5856', '876 99 5856', '876 99 5856', '876 99-5856'),
 ('884-65-2844', '884 65 2844', '884 65 2844', '884 65-2844')]
```

```
%%sql

WITH unique_ids AS (
    SELECT '241-80-7115' AS unique_id UNION
    SELECT '694-30:6851' UNION
    SELECT '586-92-5361' UNION
    SELECT '884:65-2844' UNION
    SELECT '876/99-5856' UNION
    SELECT '831-59:5593' UNION
    SELECT '399-88-3617' UNION
    SELECT '733:17-4217' UNION
    SELECT '873:68-9778' UNION
    SELECT '487-21/9802'
) SELECT unique_id,
    replace(replace(unique_id, '-', ' '), ':', ' ') AS unique_id_replaced,
    translate(unique_id, '-:/', '   ') AS unique_id_translated,
    overlay(overlay(unique_id PLACING ' ' FROM 4 FOR 1) PLACING ' ' FROM 7 FOR 1) AS␣
↪unique_id_overlaid
FROM unique_ids
ORDER BY unique_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('241-80-7115', '241 80 7115', '241 80 7115', '241 80 7115'),
 ('399-88-3617', '399 88 3617', '399 88 3617', '399 88 3617'),
 ('487-21/9802', '487 21/9802', '487 21 9802', '487 21 9802'),
 ('586-92-5361', '586 92 5361', '586 92 5361', '586 92 5361'),
 ('694-30:6851', '694 30 6851', '694 30 6851', '694 30 6851'),
```

(continues on next page)

(continued from previous page)

```
 ('733:17-4217', '733 17 4217', '733 17 4217', '733 17 4217'),
 ('831-59:5593', '831 59 5593', '831 59 5593', '831 59 5593'),
 ('873:68-9778', '873 68 9778', '873 68 9778', '873 68 9778'),
 ('876/99-5856', '876/99 5856', '876 99 5856', '876 99 5856'),
 ('884:65-2844', '884 65 2844', '884 65 2844', '884 65 2844')]
```

Note: In case of `translate`, if we do not have characters for replacement, then those will be replaced with empty string. For example, `translate('+86 (238) 954-9649', '+() -', '0')` will result in **0862389549649**.

```
%%sql

SELECT translate('+86 (238) 954-9649', '+() -', '0') AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('0862389549649',)]
```

```
%%sql

WITH phone_numbers AS (
    SELECT '+86 (238) 954-9649' AS phone_number UNION
    SELECT '+420 (331) 900-5807' UNION
    SELECT '+1 (320) 484-4495' UNION
    SELECT '+45 (238) 961-9801' UNION
    SELECT '+51 (123) 545-6543' UNION
    SELECT '+63 (308) 354-2560' UNION
    SELECT '+86 (433) 851-1260' UNION
    SELECT '+63 (332) 705-0319' UNION
    SELECT '+351 (147) 359-3767' UNION
    SELECT '+57 (714) 557-0468'
) SELECT phone_number,
    translate(phone_number, '+() -', '') phone_number_int
FROM phone_numbers
ORDER BY phone_number
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[('+1 (320) 484-4495', '13204844495'),
 ('+351 (147) 359-3767', '3511473593767'),
 ('+420 (331) 900-5807', '4203319005807'),
 ('+45 (238) 961-9801', '452389619801'),
 ('+51 (123) 545-6543', '511235456543'),
 ('+57 (714) 557-0468', '577145570468'),
 ('+63 (308) 354-2560', '633083542560'),
 ('+63 (332) 705-0319', '633327050319'),
 ('+86 (238) 954-9649', '862389549649'),
 ('+86 (433) 851-1260', '864338511260')]
```

6.6.3 Date Manipulation Functions

Let us go through some of the important date manipulation functions.

- Getting Current Date and Timestamp
- Date Arithmetic using `INTERVAL` and `-` operator
- Getting beginning date or time using `date_trunc`
- Extracting information using `to_char` as well as calendar functions.
- Dealing with unix timestamp using `from_unixtime`, `to_unix_timestamp`

Getting Current Date and Timestamp

Let us understand how to get the details about current or today's date as well as current timestamp.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

- `current_date` is the function or operator which will return today's date.
- `current_timestamp` is the function or operator which will return current time up to milliseconds.
- These are not like other functions and do not use () at the end.
- There is a format associated with date and timestamp.
  - Date - `yyyy-MM-dd`
  - Timestamp - `yyyy-MM-dd HH:mm:ss.SSS`
- We can apply all string manipulation functions on date or timestamp once they are typecasted to strings using `varchar`.

```
%%sql

SELECT current_date AS current_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.date(2020, 12, 1),)]
```

```
%%sql

SELECT current_timestamp AS current_timestamp
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 250677, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

Note: Example of applying string manipulation functions on dates. However, it is not a good practice. Postgres provide functions on dates or timestamps for most of the common requirements.

```
%%sql

SELECT substring(current_date::varchar, 1, 4) AS current_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2020',)]
```

Date Arithmetic

Let us understand how to perform arithmetic on dates or timestamps.

- We can add or subtract days or months or years from date or timestamp by using special operator called as `INTERVAL`.
- We can also add or subtract hours, minutes, seconds etc from date or timestamp using `INTERVAL`.
- We can combine multiple criteria in one operation using `INTERVAL`
- We can get difference between 2 dates or timestamps using minus (-) operator.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT current_date + INTERVAL '32 DAYS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2021, 1, 2, 0, 0),)]
```

```
%%sql

SELECT current_date + INTERVAL '730 DAYS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2022, 12, 1, 0, 0),)]
```

```
%%sql

SELECT current_date + INTERVAL '-730 DAYS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2018, 12, 2, 0, 0),)]
```

```
%%sql

SELECT current_date - INTERVAL '730 DAYS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2018, 12, 2, 0, 0),)]
```

```
%%sql

SELECT current_date + INTERVAL '3 MONTHS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2021, 3, 1, 0, 0),)]
```

```
%%sql

SELECT '2019-01-31'::date + INTERVAL '3 MONTHS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2019, 4, 30, 0, 0),)]
```

```
%%sql

SELECT '2019-01-31'::date + INTERVAL '3 MONTHS 3 DAYS 3 HOURS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2019, 5, 3, 3, 0),)]
```

```
%%sql

SELECT current_timestamp + INTERVAL '3 MONTHS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2021, 3, 1, 10, 55, 19, 336241, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT current_timestamp + INTERVAL '10 HOURS' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 20, 55, 19, 343569, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT current_timestamp + INTERVAL '10 MINUTES' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 11, 5, 19, 350628, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT current_timestamp + INTERVAL '10 HOURS 10 MINUTES' AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 21, 5, 19, 357712, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT '2019-03-30'::date - '2017-12-31'::date AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(454,)]
```

```
%%sql

SELECT '2017-12-31'::date - '2019-03-30'::date AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(-454,)]
```

```
%%sql

SELECT current_date - '2019-03-30'::date AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(612,)]
```

```
%%sql

SELECT current_timestamp - '2019-03-30'::date AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.timedelta(612, 39319, 384205),)]
```

Beginning Date or Time - date_trunc

Let us understand how to use `date_trunc` on dates or timestamps and get beginning date or time.

- We can use **MONTH** to get beginning date of the month.
- **YEAR** can be used to get begining date of the year.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT date_trunc('YEAR', current_date) AS year_beginning
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 1, 1, 0, 0, tzinfo=psycopg2.tz.FixedOffsetTimezone(offset=0,
↪ name=None)),)]
```

```
%%sql

SELECT date_trunc('MONTH', current_date) AS month_beginning
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 0, 0, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT date_trunc('WEEK', current_date) AS week_beginning
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 11, 30, 0, 0, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT date_trunc('DAY', current_date) AS day_beginning
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 0, 0, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT date_trunc('HOUR', current_timestamp) AS hour_beginning
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 0, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

Extracting information using to_char

Let us understand how to use `to_char` to extract information from date or timestamp.

Here is how we can get date related information such as year, month, day etc from date or timestamp.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT current_timestamp AS current_timestamp
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 457415, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'yyyy') AS year
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 463947, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '2020')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'yy') AS year
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 470978, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '20')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'MM') AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 477856, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '12')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'dd') AS day_of_month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 485328, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '01')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'DD') AS day_of_month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 492543, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '01')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'DDD') AS day_of_year
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 500876, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '336')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'Mon') AS month_name
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 508738, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'Dec')]
```

```
%%sql
```

(continues on next page)

(continued from previous page)

```
SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'mon') AS month_name
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 516104, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'dec')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'Month') AS month_name
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 524174, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'December ')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'month') AS month_name
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 531890, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'december ')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'day') AS day_name
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 539086, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'tuesday  ')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'DY') AS day_name
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 546707, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'TUE')]
```

Note: When we use `Day` to get the complete name of a day, it will return 9 character string by padding with spaces.

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'Day') AS dayname
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 554521, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'Tuesday  ')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char('2020-11-17'::date, 'Day') AS dayname,
    length(to_char('2020-11-17'::date, 'Day')) AS dayname_length,
    length(trim(to_char('2020-11-17'::date, 'Day'))) AS dayname_trimmed_length
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 562443, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), 'Tuesday  ', 9, 7)]
```

- Here is how we can get time related information such as hour, minute, seconds, milliseconds etc from timestamp.

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'HH') AS hour24
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 569746, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '10')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'hh') AS hour12
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 578703, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '10')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'mm') AS minutes
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 588247, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '12')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'ss') AS seconds
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 595061, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '19')]
```

```
%%sql

SELECT current_timestamp AS current_timestamp,
    to_char(current_timestamp, 'MS') AS millis
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2020, 12, 1, 10, 55, 19, 602141, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)), '602')]
```

- Here is how we can get the information from date or timestamp in the format we require.

```
%%sql

SELECT to_char(current_timestamp, 'yyyyMM') AS current_month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('202012',)]
```

```
%%sql

SELECT to_char(current_timestamp, 'yyyyMMdd') AS current_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('20201201',)]
```

```
%%sql

SELECT to_char(current_timestamp, 'yyyy/MM/dd') AS current_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2020/12/01',)]
```

Extracting information - extract

We can get year, month, day etc from date or timestamp using `extract` function. For almost all these scenarios such as getting year, month, day etc we can use `to_char` as well.

- Let us see the usage of `extract` to get information such as year, quarter, month, week, day, hour etc.
- We can also use `date_part` in place of `extract`. However there is subtle difference between them with respect to the syntax.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT extract(century FROM current_date) AS century
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(21.0,)]
```

```
%%sql

SELECT date_part('century', current_date) AS century
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(21.0,)]
```

```
%%sql

SELECT extract(decade FROM current_date) AS decade
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(202.0,)]
```

```
%%sql

SELECT date_part('decade', current_date) AS century
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(202.0,)]
```

```
%%sql

SELECT extract(year FROM current_date) AS year
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2020.0,)]
```

```
%%sql

SELECT extract(quarter FROM current_date) AS quarter
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(4.0,)]
```

```
%%sql

SELECT extract(month FROM current_date) AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(12.0,)]
```

```
%%sql

SELECT extract(week FROM current_date) AS week
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(49.0,)]
```

```
%%sql

SELECT extract(day FROM current_date) AS day
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1.0,)]
```

```
%%sql

SELECT extract(doy FROM current_date) AS day_of_year
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(336.0,)]
```

```
%%sql

SELECT extract(dow FROM current_date) AS day_of_week
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2.0,)]
```

```
%%sql

SELECT extract(hour FROM current_timestamp) AS hour
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(10.0,)]
```

```
%%sql

SELECT extract(minute FROM current_timestamp) AS minute
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(55.0,)]
```

```
%%sql

SELECT extract(second FROM current_timestamp) AS second
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(19.740129,)]
```

```
%%sql

SELECT extract(milliseconds FROM current_timestamp) AS millis
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(19747.729,)]
```

Dealing with Unix Timestamp

Let us go through the functions that can be used to deal with Unix Timestamp.

- `extract` with `epoch` can be used to convert Unix epoch to regular timestamp. We can also use `date_part`;
- `to_timestamp` can be used to convert timestamp to Unix epoch.
- We can get Unix epoch or Unix timestamp by running `date '+%s'` in Unix/Linux terminal

Let us sww how we can use functions such as `extract` or `to_timestamp` to convert between timestamp and Unix timestamp or epoch.

- We can unix epoch in Unix/Linux terminal using `date '+%s'`

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT extract(epoch FROM current_date) AS date_epoch
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1606780800.0,)]
```

```
%%sql

SELECT date_part('epoch', current_date) AS date_epoch
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1606780800.0,)]
```

```
%%sql

SELECT extract(epoch FROM '2019-04-30 18:18:51'::timestamp) AS unixtime
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1556648331.0,)]
```

```
%%sql

SELECT to_timestamp(1556662731) AS time_from_epoch
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.datetime(2019, 4, 30, 22, 18, 51, tzinfo=psycopg2.tz.
↪FixedOffsetTimezone(offset=0, name=None)),)]
```

```
%%sql

SELECT to_timestamp(1556662731)::date AS time_from_epoch
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.date(2019, 4, 30),)]
```

```
%%sql

SELECT to_char(to_timestamp(1556662731), 'yyyyMM')::int AS yyyyMM_from_epoch
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(201904,)]
```

6.6.4 Overview of Numeric Functions

Here are some of the numeric functions we might use quite often.

- `abs` - always return positive number
- `round` - rounds off to specified precision
- `ceil`, `floor` - always return integer.
- `greatest`
- `sum`, `avg`
- `min`, `max`
- `random`
- `pow`, `sqrt`

Some of the functions highlighted are aggregate functions, eg: `sum`, `avg`, `min`, `max` etc.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT abs(-10.5), abs(10)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(Decimal('10.5'), 10)]
```

```
%%sql

SELECT avg(order_item_subtotal) AS order_revenue_avg FROM order_items
WHERE order_item_order_id = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(193.32666666666668,)]
```

```
%%sql

SELECT order_item_order_id,
    sum(order_item_subtotal) AS order_revenue_sum
FROM order_items
```

(continues on next page)

(continued from previous page)

```
GROUP BY order_item_order_id
ORDER BY order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, 299.98),
 (2, 579.98),
 (4, 699.85),
 (5, 1129.8600000000001),
 (7, 579.9200000000001),
 (8, 729.8399999999999),
 (9, 599.96),
 (10, 651.9200000000001),
 (11, 919.79),
 (12, 1299.8700000000001)]
```

```
%%sql

SELECT
    round(10.58) rnd,
    floor(10.58) flr,
    ceil(10.58) cl
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(Decimal('11'), Decimal('10'), Decimal('11'))]
```

```
%%sql

SELECT
    round(10.48, 1) rnd,
    floor(10.48) flr,
    ceil(10.48) cl
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(Decimal('10.5'), Decimal('10'), Decimal('11'))]
```

```
%%sql

SELECT round(avg(order_item_subtotal)::numeric, 2) AS order_revenue_avg
FROM order_items
WHERE order_item_order_id = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(Decimal('193.33'),)]
```

```
%%sql

SELECT order_item_order_id,
    round(sum(order_item_subtotal)::numeric, 2) AS order_revenue_avg
FROM order_items
GROUP BY order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, Decimal('299.98')),
 (2, Decimal('579.98')),
 (4, Decimal('699.85')),
 (5, Decimal('1129.86')),
 (7, Decimal('579.92')),
 (8, Decimal('729.84')),
 (9, Decimal('599.96')),
 (10, Decimal('651.92')),
 (11, Decimal('919.79')),
 (12, Decimal('1299.87'))]
```

```
%%sql

SELECT greatest(10, 11, 10.5)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(Decimal('11'),)]
```

```
%%sql

SELECT order_item_order_id,
    round(sum(order_item_subtotal)::numeric, 2) AS order_revenue_sum,
    min(order_item_subtotal) AS order_item_subtotal_min,
    max(order_item_subtotal) AS order_item_subtotal_max
FROM order_items
GROUP BY order_item_order_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, Decimal('299.98'), 299.98, 299.98),
 (2, Decimal('579.98'), 129.99, 250.0),
 (4, Decimal('699.85'), 49.98, 299.95),
 (5, Decimal('1129.86'), 99.96, 299.98),
 (7, Decimal('579.92'), 79.95, 299.98),
 (8, Decimal('729.84'), 50.0, 299.95),
 (9, Decimal('599.96'), 199.98, 199.99),
 (10, Decimal('651.92'), 21.99, 199.99),
 (11, Decimal('919.79'), 49.98, 399.96),
 (12, Decimal('1299.87'), 100.0, 499.95)]
```

```
%sql SELECT random()
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(0.03222073158160299,)]
```

```
%sql SELECT (random() * 100)::int + 1
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(98,)]
```

```
%sql SELECT pow(2, 2)::int, sqrt(4)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(4, 2.0)]
```

6.6.5 Data Type Conversion

Let us understand how we can type cast to change the data type of extracted value to its original type.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT '09'::int
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9,)]
```

```
%%sql

SELECT current_date AS current_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.date(2020, 12, 1),)]
```

```
%%sql

SELECT split_part('2020-09-30', '-', 2) AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('09',)]
```

```
%%sql

SELECT split_part('2020-09-30', '-', 2)::int AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9,)]
```

```
%%sql

SELECT to_char('2020-09-30'::date, 'MM') AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('09',)]
```

```
%%sql

SELECT to_char('2020-09-30'::date, 'MM')::int AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9,)]
```

```
%%sql

SELECT to_char(current_date, 'MM')::int AS month
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(12,)]
```

```
%%sql

SELECT cast('0.04000' AS FLOAT) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(0.04,)]
```

```
%%sql

SELECT '0.04000'::float AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(0.04,)]
```

```
%%sql

SELECT cast('09' AS INT) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9,)]
```

```
%%sql

SELECT '09'::int AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(9,)]
```

6.6.6 Handling NULL Values

Let us understand how to handle nulls.

- By default if we try to add or concatenate null to another column or expression or literal, it will return null.
- If we want to replace null with some default value, we can use `coalesce`.
  - Replace commission_pct with 0 if it is null.
- `coalesce` returns first not null value if we pass multiple arguments to it.
- We have a function called as `nullif`. If the first argument is equal to second argument, it returns null. It is typically used when we compare against 2 columns where nulls are also involved.
- You might have seen functions like `nvl`, `nvl2` etc with respect to databases like Oracle. Postgres does not support them.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT 1 + NULL AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(None,)]
```

```
%%sql

SELECT coalesce(1, 0) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%%sql

SELECT coalesce(NULL, NULL, 2, NULL, 3) AS result
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(2,)]
```

```
%sql DROP TABLE IF EXISTS sales
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE IF NOT EXISTS sales(
    sales_person_id INT,
    sales_amount FLOAT,
    commission_pct INT
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO sales VALUES
    (1, 1000, 10),
    (2, 1500, 8),
    (3, 500, NULL),
    (4, 800, 5),
    (5, 250, NULL)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM sales
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 1000.0, 10),
 (2, 1500.0, 8),
 (3, 500.0, None),
 (4, 800.0, 5),
 (5, 250.0, None)]
```

```
%%sql

SELECT s.*,
    round((sales_amount * commission_pct / 100)::numeric, 2) AS incorrect_commission_
↪amount
FROM sales AS s
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 1000.0, 10, Decimal('100.00')),
 (2, 1500.0, 8, Decimal('120.00')),
 (3, 500.0, None, None),
 (4, 800.0, 5, Decimal('40.00')),
 (5, 250.0, None, None)]
```

```
%%sql

SELECT s.*,
    coalesce(commission_pct, 0) AS commission_pct
FROM sales AS s
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 1000.0, 10, 10),
 (2, 1500.0, 8, 8),
 (3, 500.0, None, 0),
 (4, 800.0, 5, 5),
 (5, 250.0, None, 0)]
```

```
%%sql

SELECT s.*,
    round((sales_amount * coalesce(commission_pct, 0) / 100)::numeric, 2) AS␣
↪commission_amount
FROM sales AS s
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 1000.0, 10, Decimal('100.00')),
 (2, 1500.0, 8, Decimal('120.00')),
 (3, 500.0, None, Decimal('0.00')),
 (4, 800.0, 5, Decimal('40.00')),
 (5, 250.0, None, Decimal('0.00'))]
```

```
%%sql

SELECT nullif(1, 0)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(1,)]
```

```
%%sql

SELECT nullif(1, 1)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(None,)]
```

6.6.7 Using CASE and WHEN

At times we might have to select values from multiple columns conditionally.

- We can use `CASE` and `WHEN` for that.
- Let us implement this conditional logic to come up with derived order_status.
  - If order_status is COMPLETE or CLOSED, set COMPLETED
  - If order_status have PENDING in it, then we will say PENDING
  - If order_status have PROCESSING or PAYMENT_REVIEW in it, then we will say PENDING
  - We will set all others as OTHER
- We can also have `ELSE` as part of `CASE` and `WHEN`.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS sales
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE IF NOT EXISTS sales(
    sales_person_id INT,
    sales_amount FLOAT,
    commission_pct INT
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO sales VALUES
    (1, 1000, 10),
    (2, 1500, 8),
    (3, 500, NULL),
    (4, 800, 5),
    (5, 250, NULL)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM sales
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 1000.0, 10),
 (2, 1500.0, 8),
 (3, 500.0, None),
 (4, 800.0, 5),
 (5, 250.0, None)]
```

```
%%sql

SELECT s.*,
    CASE WHEN commission_pct IS NOT NULL
        THEN round((sales_amount * commission_pct / 100)::numeric, 2)
    ELSE 0
    END AS commission_amount
FROM sales s
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
5 rows affected.
```

```
[(1, 1000.0, 10, Decimal('100.00')),
 (2, 1500.0, 8, Decimal('120.00')),
 (3, 500.0, None, Decimal('0')),
 (4, 800.0, 5, Decimal('40.00')),
 (5, 250.0, None, Decimal('0'))]
```

```
%%sql

SELECT DISTINCT order_status FROM orders
ORDER BY order_status
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9 rows affected.
```

```
[('CANCELED',),
 ('CLOSED',),
 ('COMPLETE',),
 ('ON_HOLD',),
 ('PAYMENT_REVIEW',),
 ('PENDING',),
 ('PENDING_PAYMENT',),
 ('PROCESSING',),
 ('SUSPECTED_FRAUD',)]
```

```
%%sql

SELECT o.*,
    CASE WHEN order_status IN ('COMPLETE', 'CLOSED') THEN 'COMPLETED'
    END AS updated_order_status
FROM orders o
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'COMPLETE', 'COMPLETED'),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'PENDING', None),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'CLOSED', 'COMPLETED'),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'CLOSED', 'COMPLETED'),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'COMPLETE', 'COMPLETED'),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'ON_HOLD', None),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'COMPLETE', 'COMPLETED'),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'COMPLETE', 'COMPLETED'),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'COMPLETE', 'COMPLETED'),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'CLOSED', 'COMPLETED')]
```

```
%%sql

SELECT o.*,
    CASE WHEN order_status IN ('COMPLETE', 'CLOSED') THEN 'COMPLETED'
    ELSE order_status
    END AS updated_order_status
FROM orders o
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'COMPLETE', 'COMPLETED'),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'PENDING', 'PENDING'),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'CLOSED', 'COMPLETED'),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'CLOSED', 'COMPLETED'),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'COMPLETE', 'COMPLETED'),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'ON_HOLD', 'ON_HOLD'),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'COMPLETE', 'COMPLETED'),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'COMPLETE', 'COMPLETED'),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'COMPLETE', 'COMPLETED'),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'CLOSED', 'COMPLETED')]
```

```
%%sql

SELECT o.*,
    CASE
        WHEN order_status IN ('COMPLETE', 'CLOSED') THEN 'COMPLETED'
        WHEN order_status ~ 'PENDING' THEN 'PENDING'
        ELSE 'OTHER'
    END AS updated_order_status
FROM orders o
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'COMPLETE', 'COMPLETED'),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'PENDING', 'PENDING'),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'CLOSED', 'COMPLETED'),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'CLOSED', 'COMPLETED'),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'COMPLETE', 'COMPLETED'),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'ON_HOLD', 'OTHER'),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'COMPLETE', 'COMPLETED'),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'COMPLETE', 'COMPLETED'),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'COMPLETE', 'COMPLETED'),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'CLOSED', 'COMPLETED')]
```

```
%%sql

SELECT o.*,
    CASE
        WHEN order_status IN ('COMPLETE', 'CLOSED') THEN 'COMPLETED'
        WHEN order_status LIKE '%PENDING%' OR order_status IN ('PROCESSING', 'PAYMENT_
↪REVIEW')
            THEN 'PENDING'
        ELSE 'OTHER'
    END AS updated_order_status
FROM orders o
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, datetime.datetime(2013, 7, 30, 0, 0), 10118, 'COMPLETE', 'COMPLETED'),
 (4068, datetime.datetime(2013, 8, 17, 0, 0), 12293, 'PENDING', 'PENDING'),
 (5881, datetime.datetime(2013, 8, 30, 0, 0), 3715, 'CLOSED', 'COMPLETED'),
 (7564, datetime.datetime(2013, 9, 9, 0, 0), 8648, 'CLOSED', 'COMPLETED'),
 (8766, datetime.datetime(2013, 9, 18, 0, 0), 855, 'COMPLETE', 'COMPLETED'),
 (8926, datetime.datetime(2013, 9, 19, 0, 0), 10517, 'ON_HOLD', 'OTHER'),
 (9290, datetime.datetime(2013, 9, 21, 0, 0), 11879, 'COMPLETE', 'COMPLETED'),
 (9793, datetime.datetime(2013, 9, 24, 0, 0), 9809, 'COMPLETE', 'COMPLETED'),
 (9816, datetime.datetime(2013, 9, 24, 0, 0), 1753, 'COMPLETE', 'COMPLETED'),
 (14047, datetime.datetime(2013, 10, 20, 0, 0), 6473, 'CLOSED', 'COMPLETED')]
```

```
%%sql

SELECT DISTINCT order_status,
    CASE
        WHEN order_status IN ('COMPLETE', 'CLOSED') THEN 'COMPLETED'
        WHEN order_status LIKE '%PENDING%' OR order_status IN ('PROCESSING', 'PAYMENT_
↪REVIEW')
            THEN 'PENDING'
        ELSE 'OTHER'
    END AS updated_order_status
FROM orders
ORDER BY updated_order_status
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9 rows affected.
```

```
[('CLOSED', 'COMPLETED'),
 ('COMPLETE', 'COMPLETED'),
 ('SUSPECTED_FRAUD', 'OTHER'),
 ('CANCELED', 'OTHER'),
 ('ON_HOLD', 'OTHER'),
 ('PAYMENT_REVIEW', 'PENDING'),
 ('PENDING_PAYMENT', 'PENDING'),
 ('PROCESSING', 'PENDING'),
 ('PENDING', 'PENDING')]
```

6.6.8 Exercises - Pre-Defined Functions

Here are the exercises to ensure our understanding related to Pre-Defined Functions.

- We will use **users** table as well as other tables we got as part of retail database.
- Information will be provided with each exercise.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

DROP TABLE IF EXISTS users
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE users(
    user_id SERIAL PRIMARY KEY,
    user_first_name VARCHAR(30),
    user_last_name VARCHAR(30),
    user_email_id VARCHAR(50),
    user_gender VARCHAR(1),
    user_unique_id VARCHAR(15),
    user_phone_no VARCHAR(20),
    user_dob DATE,
    created_ts TIMESTAMP
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

insert into users (
    user_first_name, user_last_name, user_email_id, user_gender,
    user_unique_id, user_phone_no, user_dob, created_ts
) VALUES
    ('Giuseppe', 'Bode', 'gbode0@imgur.com', 'M', '88833-8759',
     '+86 (764) 443-1967', '1973-05-31', '2018-04-15 12:13:38'),
    ('Lexy', 'Gisbey', 'lgisbey1@mail.ru', 'F', '262501-029',
     '+86 (751) 160-3742', '2003-05-31', '2020-12-29 06:44:09'),
    ('Karel', 'Claringbold', 'kclaringbold2@yale.edu', 'F', '391-33-2823',
     '+62 (445) 471-2682', '1985-11-28', '2018-11-19 00:04:08'),
    ('Marv', 'Tanswill', 'mtanswill3@dedecms.com', 'F', '1195413-80',
     '+62 (497) 736-6802', '1998-05-24', '2018-11-19 16:29:43'),
    ('Gertie', 'Espinoza', 'gespinoza4@nationalgeographic.com', 'M', '471-24-6869',
     '+249 (687) 506-2960', '1997-10-30', '2020-01-25 21:31:10'),
    ('Saleem', 'Danneil', 'sdanneil5@guardian.co.uk', 'F', '192374-933',
     '+63 (810) 321-0331', '1992-03-08', '2020-11-07 19:01:14'),
    ('Rickert', 'O''Shiels', 'roshiels6@wikispaces.com', 'M', '749-27-47-52',
     '+86 (184) 759-3933', '1972-11-01', '2018-03-20 10:53:24'),
    ('Cybil', 'Lissimore', 'clissimore7@pinterest.com', 'M', '461-75-4198',
     '+54 (613) 939-6976', '1978-03-03', '2019-12-09 14:08:30'),
    ('Melita', 'Rimington', 'mrimington8@mozilla.org', 'F', '892-36-676-2',
     '+48 (322) 829-8638', '1995-12-15', '2018-04-03 04:21:33'),
    ('Benetta', 'Nana', 'bnana9@google.com', 'M', '197-54-1646',
     '+420 (934) 611-0020', '1971-12-07', '2018-10-17 21:02:51'),
    ('Gregorius', 'Gullane', 'ggullanea@prnewswire.com', 'F', '232-55-52-58',
     '+62 (780) 859-1578', '1973-09-18', '2020-01-14 23:38:53'),
    ('Una', 'Glayzer', 'uglayzerb@pinterest.com', 'M', '898-84-336-6',
     '+380 (840) 437-3981', '1983-05-26', '2019-09-17 03:24:21'),
    ('Jamie', 'Vosper', 'jvosperc@umich.edu', 'M', '247-95-68-44',
     '+81 (205) 723-1942', '1972-03-18', '2020-07-23 16:39:33'),
    ('Calley', 'Tilson', 'ctilsond@issuu.com', 'F', '415-48-894-3',
     '+229 (698) 777-4904', '1987-06-12', '2020-06-05 12:10:50'),
    ('Peadar', 'Gregorowicz', 'pgregorowicze@omniture.com', 'M', '403-39-5-869',
     '+7 (267) 853-3262', '1996-09-21', '2018-05-29 23:51:31'),
    ('Jeanie', 'Webling', 'jweblingf@booking.com', 'F', '399-83-05-03',
     '+351 (684) 413-0550', '1994-12-27', '2018-02-09 01:31:11'),
    ('Yankee', 'Jelf', 'yjelfg@wufoo.com', 'F', '607-99-0411',
     '+1 (864) 112-7432', '1988-11-13', '2019-09-16 16:09:12'),
    ('Blair', 'Aumerle', 'baumerleh@toplist.cz', 'F', '430-01-578-5',
     '+7 (393) 232-1860', '1979-11-09', '2018-10-28 19:25:35'),
    ('Pavlov', 'Steljes', 'psteljesi@macromedia.com', 'F', '571-09-6181',
     '+598 (877) 881-3236', '1991-06-24', '2020-09-18 05:34:31'),
    ('Darn', 'Hadeke', 'dhadekej@last.fm', 'M', '478-32-02-87',
     '+370 (347) 110-4270', '1984-09-04', '2018-02-10 12:56:00'),
    ('Wendell', 'Spanton', 'wspantonk@de.vu', 'F', null,
     '+84 (301) 762-1316', '1973-07-24', '2018-01-30 01:20:11'),
    ('Carlo', 'Yearby', 'cyearbyl@comcast.net', 'F', null,
     '+55 (288) 623-4067', '1974-11-11', '2018-06-24 03:18:40'),
    ('Sheila', 'Evitts', 'sevittsm@webmd.com', null, '830-40-5287',
     null, '1977-03-01', '2020-07-20 09:59:41'),
```

(continues on next page)

(continued from previous page)

```
    ('Sianna', 'Lowdham', 'slowdhamn@stanford.edu', null, '778-0845',
     null, '1985-12-23', '2018-06-29 02:42:49'),
    ('Phylys', 'Aslie', 'paslieo@qq.com', 'M', '368-44-4478',
     '+86 (765) 152-8654', '1984-03-22', '2019-10-01 01:34:28')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
25 rows affected.
```

```
[]
```

Exercise 1

Get all the number of users created per year.

- Use **users** table for this exercise.
- Output should contain 4 digit year and count.
- Use date specific functions to get the year using created_ts.
- Make sure you define aliases to the columns as **created_year** and **user_count** respectively.
- Data should be sorted in ascending order by **created_year**.
- When you run the query using Jupyter environment, it might have decimals for integers. Hence you can display results even with decimal points.
- Here is the sample output.

created_year	user_count
2018	13
2019	4
2020	8

Exercise 2

Get the day name of the birth days for all the users born in the month of June.

- Use **users** table for this exercise.
- Output should contain user_id, user_dob, user_email_id and user_day_of_birth.
- Use date specific functions to get the month using user_dob.
- **user_day_of_birth** should be full day with first character in upper case such as **Tuesday**
- Data should be sorted by day with in the month of May.

user_id	user_dob	user_email_id	user_day_of_birth
4	1998-05-24	mtanswill3@dedecms.com	Sunday
12	1983-05-26	uglayzerb@pinterest.com	Thursday
1	1973-05-31	gbode0@imgur.com	Thursday
2	2003-05-31	lgisbey1@mail.ru	Saturday

Exercise 3

Get the names and email ids of users added in year 2019.

- Use **users** table for this exercise.
- Output should contain user_id, user_name, user_email_id, created_ts, created_year.
- Use date specific functions to get the year using created_ts.
- **user_name** is a derived column by concatenating user_first_name and user_last_name with space in between.
- **user_name** should have values in upper case.
- Data should be sorted in ascending order by user_name

user_id	user_name	user_email_id	created_ts	created_year
8	CYBIL LISSIMORE	clissimore7@pinterest.com	2019-12-09 14:08:30	2019.0
25	PHYLYS ASLIE	paslieo@qq.com	2019-10-01 01:34:28	2019.0
12	UNA GLAYZER	uglayzerb@pinterest.com	2019-09-17 03:24:21	2019.0
17	YANKEE JELF	yjelfg@wufoo.com	2019-09-16 16:09:12	2019.0

Exercise 4

Get the number of users by gender.

- Use **users** table for this exercise.
- Output should contain gender and user_count.
- For males the output should display **Male** and for females the output should display **Female**.
- If gender is not specified, then it should display **Not Specified**.
- Data should be sorted in descending order by user_count.

user_gender	user_count
Female	13
Male	10
Not Specified	2

Exercise 5

Get last 4 digits of unique ids.

- Use **users** table for this exercise.
- Output should contain user_id, user_unique_id and user_unique_id_last4
- Unique ids are either null or not null.
- Unique ids contain numbers and hyphens and are of different length.
- We need to get last 4 digits discarding hyphens only when the number of digits are at least 9.
- If unique id is null, then you should dispaly **Not Specified**.
- After discarding hyphens, if unique id have less than 9 digits then you should display **Invalid Unique Id**.
- Data should be sorted by user_id. You might see **None** or **null** for those user ids where there is no unique id for **user_unique_id**

user_id	user_unique_id	user_unique_id_last4
1	88833-8759	8759
2	262501-029	1029
3	391-33-2823	2823
4	1195413-80	1380
5	471-24-6869	6869
6	192374-933	4933
7	749-27-47-52	4752
8	461-75-4198	4198
9	892-36-676-2	6762
10	197-54-1646	1646
11	232-55-52-58	5258
12	898-84-336-6	3366
13	247-95-68-44	6844
14	415-48-894-3	8943
15	403-39-5-869	5869
16	399-83-05-03	0503
17	607-99-0411	0411
18	430-01-578-5	5785
19	571-09-6181	6181
20	478-32-02-87	0287
21		Not Specified
22		Not Specified
23	830-40-5287	5287
24	778-0845	Invalid Unique Id
25	368-44-4478	4478

Exercise 6

Get the count of users based up on country code.

- Use users table for this exercise.
- Output should contain country code and count.
- There should be no + in the country code. It should only contain digits.
- Data should be sorted as numbers by country code.
- We should discard user_phone_no with null values.
- Here is the desired output:

country_code	user_count
1	1
7	2
48	1
54	1
55	1
62	3
63	1
81	1
84	1
86	4
229	1
249	1
351	1
370	1
380	1
420	1
598	1

Exercise 7

Let us validate if we have invalid **order_item_subtotal** as part of **order_items** table.

- **order_items** table have 6 fields.
  - order_item_id
  - order_item_order_id
  - order_item_product_id
  - order_item_quantity
  - order_item_subtotal
  - order_item_product_price
- **order_item_subtotal** is nothing but product of **order_item_quantity** and **order_item_product_price**. It means order_item_subtotal is compute by multiplying order_item_quantity and order_item_product_price for each item.
- You need to get the count of order_items where **order_item_subtotal** is not equal to the product of **order_item_quantity** and **order_item_product_price**.
- There can be issues related to rounding off. Make sure it is taken care using appropriate function.
- Output should be 0 as there are no such records.

count
0

Exercise 8

Get number of orders placed on weekdays and weekends in the month of January 2014.

- **orders** have 4 fields
  - order_id
  - order_date
  - order_customer_id
  - order_status
- Use order date to determine the day on which orders are placed.
- Output should contain 2 columns - day_type and order_count.
- **day_type** should have 2 values **Week days** and **Weekend days**.
- Here is the desired output.

day_type	order_count
Weekend days	1505
Week days	4403

6.7 Writing Advanced SQL Queries

As part of this section we will understand how to write queries using some of the advanced features.

- Overview of Views
- Overview of Sub Queries
- CTAS - Create Table As Select
- Advanced DML Operations
- Merging or Upserting Data
- Pivoting Rows into Columns
- Overview of Analytic Functions
- Analytic Functions – Aggregations
- Cumulative Aggregations
- Analytic Functions – Windowing
- Analytic Functions – Ranking
- Getting Top 5 Daily Products
- Exercises - Analytic Functions

6.7.1 Overview of Views

Here are the details related to views.

- View is nothing but a named query. We typically create views for most commonly used queries.
- Unlike tables, views does not physically store the data and when ever we write a query against view it will fetch the data from underlying tables defined as part of the views.
- We can perform DML operations over the tables via views with restrictions (for example, we cannot perform DML operations on views with joins, group by etc).
- Views that can be used to perform DML operations on underlying tables are called as **updatable views**
- Views can be used to provide restricted permissions on tables for DML Operations. However, it is not used these days.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

CREATE OR REPLACE VIEW orders_v
AS
SELECT * FROM orders
```

```
Done.
```

```
[]
```

```
%%sql

CREATE VIEW orders_v
AS
SELECT * FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.DuplicateTable) relation "orders_v" already exists

[SQL: CREATE VIEW orders_v AS
SELECT * FROM orders]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

SELECT * FROM information_schema.tables
WHERE table_name ~ 'orders'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
2 rows affected.
```

```
[('itversity_retail_db', 'public', 'orders', 'BASE TABLE', None, None, None, None,␣
↪None, 'YES', 'NO', None),
 ('itversity_retail_db', 'public', 'orders_v', 'VIEW', None, None, None, None, None,
↪'YES', 'NO', None)]
```

```
%%sql

UPDATE orders_v
SET order_status = lower(order_status)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
68883 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM orders LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(122, datetime.datetime(2013, 7, 26, 0, 0), 2071, 'processing'),
 (123, datetime.datetime(2013, 7, 26, 0, 0), 3695, 'pending_payment'),
 (124, datetime.datetime(2013, 7, 26, 0, 0), 2374, 'complete'),
 (125, datetime.datetime(2013, 7, 26, 0, 0), 4611, 'pending_payment'),
 (126, datetime.datetime(2013, 7, 26, 0, 0), 610, 'complete'),
 (127, datetime.datetime(2013, 7, 26, 0, 0), 5261, 'pending_payment'),
 (128, datetime.datetime(2013, 7, 26, 0, 0), 2772, 'pending_payment'),
 (129, datetime.datetime(2013, 7, 26, 0, 0), 9937, 'closed'),
 (130, datetime.datetime(2013, 7, 26, 0, 0), 7509, 'pending_payment'),
 (131, datetime.datetime(2013, 7, 26, 0, 0), 10072, 'processing')]
```

```
%%sql

UPDATE orders_v
SET order_status = upper(order_status)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
68883 rows affected.
```

```
[]
```

```
%%sql

CREATE OR REPLACE VIEW order_details_v
AS
SELECT * FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT * FROM order_details_v LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 11599, 'CLOSED', 1, 1, 957, 1, 299.98, 299.
↪98),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 2, 2, 1073, 1, 199.
↪99, 199.99),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 3, 2, 502, 5, 250.
↪0, 50.0),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 4, 2, 403, 1, 129.
↪99, 129.99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 5, 4, 897, 2, 49.98, 24.
↪99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 6, 4, 365, 5, 299.95, 59.
↪99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 7, 4, 502, 3, 150.0, 50.0),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 8, 4, 1014, 4, 199.92, 49.
↪98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE', 9, 5, 957, 1, 299.98,␣
↪299.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE', 10, 5, 365, 5, 299.95,␣
↪59.99)]
```

```
%%sql

SELECT count(1) FROM order_details_v
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%%sql

SELECT order_date,
    order_item_product_id,
    round(sum(order_item_subtotal)::numeric, 2) AS revenue
FROM order_details_v
GROUP BY order_date,
    order_item_product_id
ORDER BY order_date,
    revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('10799.46')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('9599.36')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('8499.15')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('7558.74')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('6999.65')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('6397.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('5589.57')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('5100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('2879.28')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'))]
```

```
%%sql

SELECT * FROM order_details_v
WHERE order_id = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[(2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 2, 2, 1073, 1, 199.
↪99, 199.99),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 3, 2, 502, 5, 250.
↪0, 50.0),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 4, 2, 403, 1, 129.
↪99, 129.99)]
```

Note: We cannot directly update data in tables via views when the view is defined with joins. Even operations such as `GROUP BY` or `ORDER BY` etc will make views not updatable by default.

```
%%sql

UPDATE order_details_v
SET
    order_status = 'pending_payment'
WHERE order_id = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.ObjectNotInPrerequisiteState) cannot update view "order_details_v"
DETAIL:  Views that do not select from a single table or view are not automatically␣
↪updatable.
HINT:  To enable updating the view, provide an INSTEAD OF UPDATE trigger or an␣
↪unconditional ON UPDATE DO INSTEAD rule.

[SQL: UPDATE order_details_v SET order_status = 'pending_payment'
WHERE order_id = 2]
(Background on this error at: http://sqlalche.me/e/13/e3q8)
```

6.7.2 Named Queries - Using WITH Clause

Let us understand how to use `WITH` clause to define a named query.

- At times we might have to develop a large query in which same complex logic need to be used multiple times. The query can become cumbersome if you just define the same logic multiple times.
- One of the way to mitigate that issue is by providing the name to the logic using WITH clause.
- We can only use the names provided to named queries as part of the main query which follows the WITH clause.

Note: In case of frequently used complex and large query, we use named queries while defining the views. We will then use view for reporting purposes.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

WITH order_details_nq AS (
    SELECT * FROM orders o
        JOIN order_items oi
            on o.order_id = oi.order_item_order_id
) SELECT * FROM order_details_nq LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, datetime.datetime(2013, 7, 25, 0, 0), 11599, 'CLOSED', 1, 1, 957, 1, 299.98, 299.
↪98),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 2, 2, 1073, 1, 199.
↪99, 199.99),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 3, 2, 502, 5, 250.
↪0, 50.0),
 (2, datetime.datetime(2013, 7, 25, 0, 0), 256, 'PENDING_PAYMENT', 4, 2, 403, 1, 129.
↪99, 129.99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 5, 4, 897, 2, 49.98, 24.
↪99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 6, 4, 365, 5, 299.95, 59.
↪99),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 7, 4, 502, 3, 150.0, 50.0),
 (4, datetime.datetime(2013, 7, 25, 0, 0), 8827, 'CLOSED', 8, 4, 1014, 4, 199.92, 49.
↪98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE', 9, 5, 957, 1, 299.98,␣
↪299.98),
 (5, datetime.datetime(2013, 7, 25, 0, 0), 11318, 'COMPLETE', 10, 5, 365, 5, 299.95,␣
↪59.99)]
```

Error: One cannot use the named queries apart from the query in which it is defined. Following query will fail.

```
%%sql

SELECT * FROM order_details_nq LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedTable) relation "order_details_nq" does not exist
LINE 1: SELECT * FROM order_details_nq LIMIT 10
                      ^

[SQL: SELECT * FROM order_details_nq LIMIT 10]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

WITH order_details_nq AS (
    SELECT * FROM orders o
        JOIN order_items oi
            on o.order_id = oi.order_item_order_id
) SELECT order_date,
    order_item_product_id,
    round(sum(order_item_subtotal)::numeric, 2) AS revenue
FROM order_details_nq
GROUP BY order_date,
    order_item_product_id
ORDER BY order_date,
    revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('10799.46')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('9599.36')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('8499.15')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('7558.74')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('6999.65')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('6397.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('5589.57')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('5100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('2879.28')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'))]
```

```
%%sql

CREATE OR REPLACE VIEW daily_product_revenue_v
AS
WITH order_details_nq AS (
    SELECT * FROM orders o
        JOIN order_items oi
            on o.order_id = oi.order_item_order_id
) SELECT order_date,
    order_item_product_id,
```

(continues on next page)

(continued from previous page)

```
    round(sum(order_item_subtotal)::numeric, 2) AS revenue
FROM order_details_nq
GROUP BY order_date,
    order_item_product_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

SELECT * FROM daily_product_revenue_v
ORDER BY order_date, revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('10799.46')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('9599.36')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('8499.15')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('7558.74')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('6999.65')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('6397.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('5589.57')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('5100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('2879.28')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'))]
```

6.7.3 Overview of Sub Queries

Let us understand details related to Sub Queries. We will also briefly discuss about nested sub queries.

- We can have queries in from clause and such queries are called as sub queries.
- Sub queries are commonly used with queries using analytic functions to filter the data further. We will see details after going through analytic functions as part of this section.
- It is mandatory to have alias for the sub query.
- Sub queries can also be used in `WHERE` clause with `IN` as well as `EXISTS`. As part of the sub query we can have join like conditions between tables in `FROM` clause of the main query and sub query. Such queries are called as **Nested Sub Queries**.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Note: Simplest example for a subquery

```
%%sql

SELECT * FROM (SELECT current_date) AS q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.date(2020, 12, 1),)]
```

Note: Realistic example for a subquery. We will get into details related to this query after covering analytic functions

```
%%sql

SELECT * FROM (
    SELECT nq.*,
        dense_rank() OVER (
            PARTITION BY order_date
            ORDER BY revenue DESC
        ) AS drnk
    FROM (
        SELECT o.order_date,
            oi.order_item_product_id,
            round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
        FROM orders o
            JOIN order_items oi
                ON o.order_id = oi.order_item_order_id
        WHERE o.order_status IN ('COMPLETE', 'CLOSED')
        GROUP BY o.order_date, oi.order_item_product_id
    ) nq
) nq1
WHERE drnk <= 5
ORDER BY order_date, revenue DESC
LIMIT 20
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
20 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 7, 26, 0, 0), 1014, Decimal('4798.08'), 5),
 (datetime.datetime(2013, 7, 27, 0, 0), 1004, Decimal('9599.52'), 1),
 (datetime.datetime(2013, 7, 27, 0, 0), 191, Decimal('5999.40'), 2),
 (datetime.datetime(2013, 7, 27, 0, 0), 957, Decimal('5699.62'), 3),
 (datetime.datetime(2013, 7, 27, 0, 0), 1073, Decimal('5399.73'), 4),
 (datetime.datetime(2013, 7, 27, 0, 0), 365, Decimal('5099.15'), 5),
 (datetime.datetime(2013, 7, 28, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 28, 0, 0), 957, Decimal('5099.66'), 2),
 (datetime.datetime(2013, 7, 28, 0, 0), 365, Decimal('4799.20'), 3),
 (datetime.datetime(2013, 7, 28, 0, 0), 403, Decimal('4419.66'), 4),
 (datetime.datetime(2013, 7, 28, 0, 0), 191, Decimal('4299.57'), 5)]
```

Note: Multiple realistic examples for nested sub queries. You can see example with `IN` as well as `EXISTS` operators.

```
%%sql

SELECT * FROM order_items oi
WHERE oi.order_item_order_id
    NOT IN (
        SELECT order_id FROM orders o
        WHERE o.order_id = oi.order_item_order_id
    )
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT count(1) FROM order_items oi
WHERE oi.order_item_order_id
    IN (
        SELECT order_id FROM orders o
        WHERE o.order_id = oi.order_item_order_id
    )
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

```
%%sql

SELECT * FROM order_items oi
WHERE NOT EXISTS (
        SELECT 1 FROM orders o
        WHERE o.order_id = oi.order_item_order_id
    )
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM order_items oi
WHERE EXISTS (
        SELECT 1 FROM orders o
        WHERE o.order_id = oi.order_item_order_id
    )
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1, 1, 957, 1, 299.98, 299.98),
 (2, 2, 1073, 1, 199.99, 199.99),
 (3, 2, 502, 5, 250.0, 50.0),
 (4, 2, 403, 1, 129.99, 129.99),
 (5, 4, 897, 2, 49.98, 24.99),
 (6, 4, 365, 5, 299.95, 59.99),
 (7, 4, 502, 3, 150.0, 50.0),
 (8, 4, 1014, 4, 199.92, 49.98),
 (9, 5, 957, 1, 299.98, 299.98),
 (10, 5, 365, 5, 299.95, 59.99)]
```

6.7.4 CTAS - Create Table as Select

Let us understand details related to CTAS or Create Table As Select.

- CTAS is primarily used to create tables based on query results.
- Following are some of the use cases for which we typically use CTAS.
  - Taking back up of tables for troubleshooting and debugging performance issues.
  - Reorganizing the tables for performance tuning.
  - Getting query results into a table for data analysis as well as checking data quality.
- We cannot specify column names and data types as part of `CREATE TABLE` clause in CTAS. It will pick the column names from the `SELECT` clause.
- It is a good practice to specify meaningful aliases as part of the `SELECT` clause for derived values.
- Also it is a good practice to explicitly type cast to the desired data type for derived values.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

DROP TABLE IF EXISTS customers_backup
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE customers_backup
AS
SELECT * FROM customers
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
12435 rows affected.
```

```
[]
```

```
%%sql

DROP TABLE IF EXISTS orders_backup
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE orders_backup
AS
SELECT order_id,
    to_char(order_date, 'yyyy')::int AS order_year,
    to_char(order_date, 'MM')::int AS order_month,
    to_char(order_date, 'dd')::int AS order_day_of_month,
    to_char(order_date, 'DDD')::int AS order_day_of_year,
    order_customer_id,
    order_status
FROM orders
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
68883 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM orders_backup LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(1021, 2013, 7, 30, 211, 10118, 'COMPLETE'),
 (4068, 2013, 8, 17, 229, 12293, 'PENDING'),
 (5881, 2013, 8, 30, 242, 3715, 'CLOSED'),
 (7564, 2013, 9, 9, 252, 8648, 'CLOSED'),
 (8766, 2013, 9, 18, 261, 855, 'COMPLETE'),
 (8926, 2013, 9, 19, 262, 10517, 'ON_HOLD'),
 (9290, 2013, 9, 21, 264, 11879, 'COMPLETE'),
 (9793, 2013, 9, 24, 267, 9809, 'COMPLETE'),
 (9816, 2013, 9, 24, 267, 1753, 'COMPLETE'),
 (14047, 2013, 10, 20, 293, 6473, 'CLOSED')]
```

Note: At times we have to create empty table with only structure of the table. We can specify always false condition such as `1 = 2` as part of `WHERE` clause using CTAS.

```
%%sql

DROP TABLE IF EXISTS order_items_empty
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE order_items_empty
AS
SELECT * FROM order_items WHERE 1 = 2
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

```
%%sql

SELECT count(1) FROM order_items_empty
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(0,)]
```

Note: Keeping databases clean is very important. It is a good practice to clean up any temporary tables created for learning or troubleshooting issues.

In this case all the tables created using CTAS are dropped

```
%%sql

DROP TABLE IF EXISTS customers_backup;
DROP TABLE IF EXISTS orders_backup;
DROP TABLE IF EXISTS order_items_empty;
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
Done.
Done.
```

```
[]
```

6.7.5 Advanced DML Operations

As we gain enough knowledge related to writing queries, let us explore some advanced DML Operations.

- We can insert query results into a table using `INSERT` with `SELECT`.
- As long as columns specified for table in `INSERT` statement and columns projected in `SELECT` clause match, it works.
- We can also use query results for `UPDATE` as well as `DELETE`.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Note: Creating customer order metrics table to demonstrate advanced DML Operations. We will also add primary key to this table. We will be storing number of orders placed and revenue generated for each customer in a given month.

```
%%sql

DROP TABLE IF EXISTS customer_order_metrics_mthly
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE customer_order_metrics_mthly (
```

(continues on next page)

(continued from previous page)

```
    customer_id INT,
    order_month CHAR(7),
    order_count INT,
    order_revenue FLOAT
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE customer_order_metrics_mthly
    ADD PRIMARY KEY (order_month, customer_id)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

Note: Here is the query to get monthly customer orders metrics. First we will be inserting customer_id, order_month and order_count into the table.

Warning: If the below query is run multiple times, every time data in both orders and order_items need to be processed. As the data volumes grow the query uses considerable amount of resources. It will be better if we can pre-aggregate the data.

```
%%sql

SELECT o.order_customer_id,
    to_char(o.order_date, 'yyyy-MM') AS order_month,
    count(1) AS order_count,
    round(sum(order_item_subtotal)::numeric, 2) AS order_revenue
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
GROUP BY o.order_customer_id,
    to_char(o.order_date, 'yyyy-MM')
ORDER BY order_month,
    order_count DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(4257, '2013-07', 10, Decimal('2059.75')),
 (5293, '2013-07', 10, Decimal('2781.73')),
 (9103, '2013-07', 9, Decimal('1587.85')),
```

(continues on next page)

(continued from previous page)

```
 (7473, '2013-07', 9, Decimal('1244.90')),
 (2071, '2013-07', 9, Decimal('1629.84')),
 (32, '2013-07', 9, Decimal('2009.75')),
 (488, '2013-07', 9, Decimal('1365.82')),
 (7073, '2013-07', 9, Decimal('1377.83')),
 (8709, '2013-07', 8, Decimal('1349.87')),
 (1498, '2013-07', 8, Decimal('1619.88'))]
```

Warning: Here are the number of records that need to be processed every time. Also it involves expensive join.

```
%%sql

SELECT count(1)
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(172198,)]
```

Note: Let us first insert the data into the table with out revenue. We will update the revenue later as an example for updating using query results.

```
%%sql

INSERT INTO customer_order_metrics_mthly
SELECT o.order_customer_id,
    to_char(o.order_date, 'yyyy-MM') AS order_month,
    count(1) order_count,
    NULL
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
GROUP BY o.order_customer_id,
    to_char(o.order_date, 'yyyy-MM')
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
48059 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM customer_order_metrics_mthly
ORDER BY order_month,
    customer_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(12, '2013-07', 2, None),
 (16, '2013-07', 1, None),
 (17, '2013-07', 2, None),
 (19, '2013-07', 3, None),
 (32, '2013-07', 9, None),
 (45, '2013-07', 4, None),
 (48, '2013-07', 4, None),
 (54, '2013-07', 2, None),
 (58, '2013-07', 4, None),
 (64, '2013-07', 2, None)]
```

Note: Updating order_revenue along with count. This is expensive operation, but we will be running only once.

```
%%sql

UPDATE customer_order_metrics_mthly comd
SET
    (order_count, order_revenue) = (
        SELECT count(1),
            round(sum(order_item_subtotal)::numeric, 2)
        FROM orders o
            JOIN order_items oi
                ON o.order_id = oi.order_item_order_id
        WHERE o.order_customer_id = comd.customer_id
            AND to_char(o.order_date, 'yyyy-MM') = comd.order_month
            AND to_char(o.order_date, 'yyyy-MM') = '2013-08'
            AND comd.order_month = '2013-08'
        GROUP BY o.order_customer_id,
            to_char(o.order_date, 'yyyy-MM')
    )
WHERE EXISTS (
    SELECT 1 FROM orders o
    WHERE o.order_customer_id = comd.customer_id
        AND to_char(o.order_date, 'yyyy-MM') = comd.order_month
        AND to_char(o.order_date, 'yyyy-MM') = '2013-08'
) AND comd.order_month = '2013-08'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3935 rows affected.
```

```
[]
```

Note: As data is pre processed and loaded into the table, queries similar to below ones against **customer_order_metrics_mthly** will run much faster.

We need to process lesser amount of data with out expensive join.

```
%%sql
```

(continues on next page)

(continued from previous page)

```
SELECT * FROM customer_order_metrics_mthly
WHERE order_month = '2013-08'
ORDER BY order_month,
    customer_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(2, '2013-08', 5, 769.82),
 (13, '2013-08', 5, 1065.93),
 (14, '2013-08', 3, 459.97),
 (18, '2013-08', 1, 129.99),
 (20, '2013-08', 2, 739.91),
 (22, '2013-08', 5, 769.96),
 (24, '2013-08', 2, 399.91),
 (25, '2013-08', 1, 129.99),
 (33, '2013-08', 3, 929.92),
 (34, '2013-08', 4, 789.92)]
```

Note: As an example for delete using query, we will delete all the dormant customers from **customers** table. Dormant customers are those customers who never placed any order. For this we will create back up customers table as I do not want to play with customers.

```
%%sql

DROP TABLE IF EXISTS customers_backup
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE customers_backup
AS
SELECT * FROM customers
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
12435 rows affected.
```

```
[]
```

```
%%sql

SELECT count(1) FROM customers_backup c
    LEFT OUTER JOIN orders o
        ON c.customer_id = o.order_customer_id
WHERE o.order_customer_id IS NULL
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(30,)]
```

```
%%sql

SELECT count(1) FROM customers_backup c
WHERE NOT EXISTS (
    SELECT 1 FROM orders o
    WHERE c.customer_id = o.order_customer_id
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(30,)]
```

Note: We need to use nested sub queries as part of the delete with `NOT EXISTS` or `NOT IN` as demonstrated below. We cannot use direct joins as part of the `DELETE`.

```
%%sql

DELETE FROM customers_backup c
WHERE NOT EXISTS (
    SELECT 1 FROM orders o
    WHERE c.customer_id = o.order_customer_id
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[]
```

```
%%sql

SELECT count(1) FROM customers_backup
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(12405,)]
```

```
%%sql

DELETE FROM customers_backup c
WHERE customer_id NOT IN (
    SELECT order_customer_id FROM orders o
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
0 rows affected.
```

```
[]
```

6.7.6 Merging or Upserting Data

At times we need to merge or upsert the data (update existing records and insert new records)

- One of the way to achieve merge or upsert is to develop 2 statements - one to update and other to insert.
- The queries in both the statements (update and insert) should return mutually exclusive results.
- Even though the statements can be executed in any order, updating first and then inserting perform better in most of the cases (as update have to deal with lesser number of records with this approach)
- We can also take care of merge or upsert using `INSERT` with `ON CONFLICT (columns) DO UPDATE`.
- Postgres does not have either `MERGE` or `UPSERT` as part of the SQL syntax.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql DROP TABLE IF EXISTS customer_order_metrics_dly
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE customer_order_metrics_dly (
    customer_id INT,
    order_date DATE,
    order_count INT,
    order_revenue FLOAT
)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE customer_order_metrics_dly
    ADD PRIMARY KEY (customer_id, order_date)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

Note: Let us go through the 2 statement approach. Here we are inserting data for the month of August 2013.

```
%%sql

INSERT INTO customer_order_metrics_dly
SELECT o.order_customer_id,
    o.order_date,
    count(1) order_count,
    NULL
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
WHERE o.order_date BETWEEN '2013-08-01' AND '2013-08-31'
GROUP BY o.order_customer_id,
    o.order_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
4708 rows affected.
```

```
[]
```

Note: Now we want to merge data into the table using 2013 August to 2013 October. As we are using 2 statement approach, first we should update and then we should insert

```
%%sql

UPDATE customer_order_metrics_dly comd
SET
    (order_count, order_revenue) = (
        SELECT count(1),
            round(sum(oi.order_item_subtotal)::numeric, 2)
        FROM orders o
            JOIN order_items oi
                ON o.order_id = oi.order_item_order_id
        WHERE o.order_date BETWEEN '2013-08-01' AND '2013-10-31'
            AND o.order_customer_id = comd.customer_id
            AND o.order_date = comd.order_date
        GROUP BY o.order_customer_id,
            o.order_date
    )
WHERE comd.order_date BETWEEN '2013-08-01' AND '2013-10-31'
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
4708 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM customer_order_metrics_dly
ORDER BY order_date, customer_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(34, datetime.date(2013, 8, 1), 4, 789.92),
 (109, datetime.date(2013, 8, 1), 3, 799.9),
 (174, datetime.date(2013, 8, 1), 5, 654.89),
 (267, datetime.date(2013, 8, 1), 4, 559.97),
 (478, datetime.date(2013, 8, 1), 5, 729.9),
 (553, datetime.date(2013, 8, 1), 2, 399.9),
 (692, datetime.date(2013, 8, 1), 2, 479.92),
 (696, datetime.date(2013, 8, 1), 2, 649.88),
 (800, datetime.date(2013, 8, 1), 5, 609.95),
 (835, datetime.date(2013, 8, 1), 5, 589.9)]
```

```
%%sql

SELECT to_char(order_date, 'yyyy-MM'), count(1) FROM customer_order_metrics_dly
GROUP BY to_char(order_date, 'yyyy-MM')
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[('2013-08', 4708)]
```

```
%%sql

INSERT INTO customer_order_metrics_dly
SELECT o.order_customer_id AS customer_id,
    o.order_date,
    count(1) order_count,
    round(sum(order_item_subtotal)::numeric, 2)
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
WHERE o.order_date BETWEEN '2013-08-01' AND '2013-10-31'
    AND NOT EXISTS (
        SELECT 1 FROM customer_order_metrics_dly codm
        WHERE o.order_customer_id = codm.customer_id
            AND o.order_date = codm.order_date
    )
GROUP BY o.order_customer_id,
    o.order_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9265 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM customer_order_metrics_dly
WHERE order_date::varchar ~ '2013-09'
ORDER BY order_date, customer_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(19, datetime.date(2013, 9, 1), 5, 839.92),
 (95, datetime.date(2013, 9, 1), 5, 969.85),
 (136, datetime.date(2013, 9, 1), 4, 639.94),
 (247, datetime.date(2013, 9, 1), 2, 639.94),
 (383, datetime.date(2013, 9, 1), 5, 729.9),
 (437, datetime.date(2013, 9, 1), 4, 829.97),
 (543, datetime.date(2013, 9, 1), 4, 1489.83),
 (601, datetime.date(2013, 9, 1), 2, 159.99),
 (689, datetime.date(2013, 9, 1), 2, 419.96),
 (842, datetime.date(2013, 9, 1), 4, 954.87)]
```

```
%%sql

SELECT to_char(order_date, 'yyyy-MM'), count(1) FROM customer_order_metrics_dly
GROUP BY to_char(order_date, 'yyyy-MM')
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[('2013-08', 4708), ('2013-10', 4417), ('2013-09', 4848)]
```

Note: Let us see how we can upsert or merge the data using `INSERT` with `ON CONFLICT (columns) DO UPDATE`. We will first insert data for the month of August 2013 and then upsert or merge for the months of August 2013 to October 2013.

```
%sql TRUNCATE TABLE customer_order_metrics_dly
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO customer_order_metrics_dly
```

(continues on next page)

(continued from previous page)

```
SELECT o.order_customer_id,
    o.order_date,
    count(1) order_count,
    NULL
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
WHERE o.order_date BETWEEN '2013-08-01' AND '2013-08-31'
GROUP BY o.order_customer_id,
    o.order_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
4708 rows affected.
```

```
[]
```

Note: We need to have unique or primary key constraint on the columns specified as part of `ON CONFLICT` clause.

```
%%sql

ALTER TABLE customer_order_metrics_dly DROP CONSTRAINT customer_order_metrics_dly_pkey
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

ALTER TABLE customer_order_metrics_dly
    ADD PRIMARY KEY (customer_id, order_date)
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

INSERT INTO customer_order_metrics_dly
SELECT o.order_customer_id,
    o.order_date,
    count(1) order_count,
    round(sum(order_item_subtotal)::numeric, 2) AS order_revenue
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
WHERE o.order_date BETWEEN '2013-08-01' AND '2013-10-31'
GROUP BY o.order_customer_id,
    o.order_date
ON CONFLICT (customer_id, order_date) DO UPDATE SET
```

(continues on next page)

(continued from previous page)

```
    order_count = EXCLUDED.order_count,
    order_revenue = EXCLUDED.order_revenue
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
13973 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM customer_order_metrics_dly
WHERE order_date::varchar ~ '2013-09'
ORDER BY order_date, customer_id
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(19, datetime.date(2013, 9, 1), 5, 839.92),
 (95, datetime.date(2013, 9, 1), 5, 969.85),
 (136, datetime.date(2013, 9, 1), 4, 639.94),
 (247, datetime.date(2013, 9, 1), 2, 639.94),
 (383, datetime.date(2013, 9, 1), 5, 729.9),
 (437, datetime.date(2013, 9, 1), 4, 829.97),
 (543, datetime.date(2013, 9, 1), 4, 1489.83),
 (601, datetime.date(2013, 9, 1), 2, 159.99),
 (689, datetime.date(2013, 9, 1), 2, 419.96),
 (842, datetime.date(2013, 9, 1), 4, 954.87)]
```

```
%%sql

SELECT to_char(order_date, 'yyyy-MM'), count(1) FROM customer_order_metrics_dly
GROUP BY to_char(order_date, 'yyyy-MM')
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
3 rows affected.
```

```
[('2013-08', 4708), ('2013-10', 4417), ('2013-09', 4848)]
```

6.7.7 Pivoting Rows into Columns

Let us understand how we can pivot rows into columns in Postgres.

- Actual results

order_date	order_status	count
2013-07-25 00:00:00	CANCELED	1
2013-07-25 00:00:00	CLOSED	20
2013-07-25 00:00:00	COMPLETE	42
2013-07-25 00:00:00	ON_HOLD	5
2013-07-25 00:00:00	PAYMENT_REVIEW	3
2013-07-25 00:00:00	PENDING	13
2013-07-25 00:00:00	PENDING_PAYMENT	41
2013-07-25 00:00:00	PROCESSING	16
2013-07-25 00:00:00	SUSPECTED_FRAUD	2
2013-07-26 00:00:00	CANCELED	3
2013-07-26 00:00:00	CLOSED	29
2013-07-26 00:00:00	COMPLETE	87
2013-07-26 00:00:00	ON_HOLD	19
2013-07-26 00:00:00	PAYMENT_REVIEW	6
2013-07-26 00:00:00	PENDING	31
2013-07-26 00:00:00	PENDING_PAYMENT	59
2013-07-26 00:00:00	PROCESSING	30
2013-07-26 00:00:00	SUSPECTED_FRAUD	5

- Pivoted results

order_date	CANCELED	CLOSED	COMPLETE	ON_HOLD	PAYMENT_REVIEW	PENDING	PENDING_PAYMENT	PROCESSING	SUSPECTED_FRAUD
2013-07-25	1	20	42	5	3	13	41	16	2
2013-07-26	3	29	87	19	6	31	59	30	5

- We need to use `crosstab` as part of `FROM` clause to pivot the data. We need to pass the main query to `crosstab` function.
- We need to install `tablefunc` as Postgres superuser to expose functions like crosstab - `CREATE EXTENSION tablefunc;`

Note: If you are using environment provided by us, you don't need to install `tablefunc`. If you are using your own environment run this command by logging in as superuser into postgres server to install `tablefunc`.

`CREATE EXTENSION tablefunc;`

However, in some cases you might have to run scripts in postgres. Follow official instructions by searching around.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT order_date,
    order_status,
    count(1)
FROM orders
GROUP BY order_date,
    order_status
ORDER BY order_date,
    order_status
LIMIT 18
```

```
%%sql

SELECT * FROM crosstab(
    'SELECT order_date,
        order_status,
        count(1) AS order_count
    FROM orders
    GROUP BY order_date,
        order_status',
    'SELECT DISTINCT order_status FROM orders ORDER BY 1'
) AS (
    order_date DATE,
    "CANCELED" INT,
    "CLOSED" INT,
    "COMPLETE" INT,
    "ON_HOLD" INT,
    "PAYMENT_REVIEW" INT,
    "PENDING" INT,
    "PENDING_PAYMENT" INT,
    "PROCESSING" INT,
    "SUSPECTED_FRAUD" INT
)
LIMIT 10
```

6.7.8 Overview of Analytic Functions

Let us get an overview of Analytics or Windowing Functions as part of **SQL**.

- Aggregate Functions (`sum`, `min`, `max`, `avg`)
- Window Functions (`lead`, `lag`, `first_value`, `last_value`)
- Rank Functions (`rank`, `dense_rank`, `row_number` etc)
- For all the functions when used as part of Analytic or Windowing functions we use `OVER` clause.
- For aggregate functions we typically use `PARTITION BY`
- For global ranking and windowing functions we can use `ORDER BY sort_column` and for ranking and windowing with in a partition or group we can use `PARTITION BY partition_column ORDER BY sort_column`.
- Here is how the syntax will look like.
  - Aggregate - `func() OVER (PARTITION BY partition_column)`
  - Global Rank - `func() OVER (ORDER BY sort_column DESC)`

  - Rank in a partition - `func() OVER (PARTITION BY partition_column ORDER BY sort_column DESC)`
- We can also get cumulative or moving metrics by adding `ROWS BETWEEN` clause. We will see details later.

Prepare Tables

Let us create couple of tables which will be used for the demonstrations of Windowing and Ranking functions.

- We have **ORDERS** and **ORDER_ITEMS** tables in our retail database.
- Let us take care of computing daily revenue as well as daily product revenue.
- As we will be using same data set several times, let us create the tables to pre compute the data.
- **daily_revenue** will have the **order_date** and **revenue**, where data is aggregated using **order_date** as partition key.
- **daily_product_revenue** will have **order_date**, **order_item_product_id** and **revenue**. In this case data is aggregated using **order_date** and **order_item_product_id** as partition keys.

Note: Let us create table using CTAS to save daily revenue.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

DROP TABLE IF EXISTS daily_revenue
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE daily_revenue
AS
SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
364 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM daily_revenue
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'))]
```

Note: Let us create table using CTAS to save daily product revenue.

```
%%sql

DROP TABLE IF EXISTS daily_product_revenue
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
Done.
```

```
[]
```

```
%%sql

CREATE TABLE daily_product_revenue
AS
SELECT o.order_date,
    oi.order_item_product_id,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date, oi.order_item_product_id
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
9120 rows affected.
```

```
[]
```

```
%%sql

SELECT * FROM daily_product_revenue
ORDER BY order_date, revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'))]
```

6.7.9 Analytic Functions – Aggregations

Let us see how we can perform aggregations with in a partition or group using Windowing/Analytics Functions.

- For simple aggregations where we have to get grouping key and aggregated results we can use **GROUP BY**.
- If we want to get the raw data along with aggregated results, then using **GROUP BY** is not possible or overly complicated.
- Using aggregate functions with **OVER** Clause not only simplifies the process of writing query, but also better with respect to performance.
- Let us take an example of getting employee salary percentage when compared to department salary expense.

Warning: If you are using Jupyter based environment make sure to restart the kernel, as the session might have been already connected with retail database.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
env: DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
%%sql

SELECT employee_id, department_id, salary
FROM employees
ORDER BY department_id, salary
LIMIT 10
```

```
10 rows affected.
```

```
[(200, 10, Decimal('4400.00')),
 (202, 20, Decimal('6000.00')),
 (201, 20, Decimal('13000.00')),
 (119, 30, Decimal('2500.00')),
 (118, 30, Decimal('2600.00')),
 (117, 30, Decimal('2800.00')),
 (116, 30, Decimal('2900.00')),
 (115, 30, Decimal('3100.00')),
 (114, 30, Decimal('11000.00')),
 (203, 40, Decimal('6500.00'))]
```

Note: Let us write the query using `GROUP BY` approach.

```
%%sql

SELECT department_id,
    sum(salary) AS department_salary_expense
FROM employees
GROUP BY department_id
ORDER BY department_id
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
12 rows affected.
```

```
[(10, Decimal('4400.00')),
 (20, Decimal('19000.00')),
 (30, Decimal('24900.00')),
 (40, Decimal('6500.00')),
 (50, Decimal('156400.00')),
 (60, Decimal('28800.00')),
 (70, Decimal('10000.00')),
 (80, Decimal('304500.00')),
 (90, Decimal('58000.00')),
 (100, Decimal('51600.00')),
 (110, Decimal('20300.00')),
 (None, Decimal('7000.00'))]
```

```
%%sql

SELECT e.employee_id, e.department_id, e.salary,
    ae.department_salary_expense,
    ae.avg_salary_expense
FROM employees e JOIN (
    SELECT department_id,
        sum(salary) AS department_salary_expense,
        round(avg(salary)::numeric, 2) AS avg_salary_expense
    FROM employees
    GROUP BY department_id
) ae
ON e.department_id = ae.department_id
ORDER BY department_id, salary
LIMIT 10
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
10 rows affected.
```

```
[(200, 10, Decimal('4400.00'), Decimal('4400.00'), Decimal('4400.00')),
 (202, 20, Decimal('6000.00'), Decimal('19000.00'), Decimal('9500.00')),
 (201, 20, Decimal('13000.00'), Decimal('19000.00'), Decimal('9500.00')),
 (119, 30, Decimal('2500.00'), Decimal('24900.00'), Decimal('4150.00')),
 (118, 30, Decimal('2600.00'), Decimal('24900.00'), Decimal('4150.00')),
 (117, 30, Decimal('2800.00'), Decimal('24900.00'), Decimal('4150.00')),
 (116, 30, Decimal('2900.00'), Decimal('24900.00'), Decimal('4150.00')),
 (115, 30, Decimal('3100.00'), Decimal('24900.00'), Decimal('4150.00')),
 (114, 30, Decimal('11000.00'), Decimal('24900.00'), Decimal('4150.00')),
 (203, 40, Decimal('6500.00'), Decimal('6500.00'), Decimal('6500.00'))]
```

```
%%sql

SELECT e.employee_id, e.department_id, e.salary,
    ae.department_salary_expense,
    ae.avg_salary_expense,
    round(e.salary/ae.department_salary_expense * 100, 2) pct_salary
FROM employees e JOIN (
    SELECT department_id,
        sum(salary) AS department_salary_expense,
        round(avg(salary)::numeric, 2) AS avg_salary_expense
    FROM employees
    GROUP BY department_id
) ae
ON e.department_id = ae.department_id
ORDER BY department_id, salary
LIMIT 10
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
10 rows affected.
```

```
[(200, 10, Decimal('4400.00'), Decimal('4400.00'), Decimal('4400.00'), Decimal('100.00
↪')),
 (202, 20, Decimal('6000.00'), Decimal('19000.00'), Decimal('9500.00'), Decimal('31.58
↪')),
 (201, 20, Decimal('13000.00'), Decimal('19000.00'), Decimal('9500.00'), Decimal('68.
↪42')),
 (119, 30, Decimal('2500.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('10.04
↪')),
 (118, 30, Decimal('2600.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('10.44
↪')),
 (117, 30, Decimal('2800.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('11.24
↪')),
 (116, 30, Decimal('2900.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('11.65
↪')),
 (115, 30, Decimal('3100.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('12.45
↪')),
 (114, 30, Decimal('11000.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('44.
↪18')),
 (203, 40, Decimal('6500.00'), Decimal('6500.00'), Decimal('6500.00'), Decimal('100.00
↪'))]
```

Note: Let us see how we can get it using Analytics/Windowing Functions.

- We can use all standard aggregate functions such as `count`, `sum`, `min`, `max`, `avg` etc.

```
%%sql

SELECT e.employee_id, e.department_id, e.salary,
    sum(e.salary) OVER (
        PARTITION BY e.department_id
    ) AS department_salary_expense
FROM employees e
ORDER BY e.department_id
LIMIT 10
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
10 rows affected.
```

```
[(200, 10, Decimal('4400.00'), Decimal('4400.00')),
 (201, 20, Decimal('13000.00'), Decimal('19000.00')),
 (202, 20, Decimal('6000.00'), Decimal('19000.00')),
 (114, 30, Decimal('11000.00'), Decimal('24900.00')),
 (115, 30, Decimal('3100.00'), Decimal('24900.00')),
 (116, 30, Decimal('2900.00'), Decimal('24900.00')),
 (117, 30, Decimal('2800.00'), Decimal('24900.00')),
 (118, 30, Decimal('2600.00'), Decimal('24900.00')),
 (119, 30, Decimal('2500.00'), Decimal('24900.00')),
 (203, 40, Decimal('6500.00'), Decimal('6500.00'))]
```

```
%%sql

SELECT e.employee_id, e.department_id, e.salary,
    sum(e.salary) OVER (
        PARTITION BY e.department_id
    ) AS department_salary_expense,
    round(e.salary / sum(e.salary) OVER (
        PARTITION BY e.department_id
    ) * 100, 2) AS pct_salary
FROM employees e
ORDER BY e.department_id,
    e.salary
LIMIT 10
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
10 rows affected.
```

```
[(200, 10, Decimal('4400.00'), Decimal('4400.00'), Decimal('100.00')),
 (202, 20, Decimal('6000.00'), Decimal('19000.00'), Decimal('31.58')),
 (201, 20, Decimal('13000.00'), Decimal('19000.00'), Decimal('68.42')),
 (119, 30, Decimal('2500.00'), Decimal('24900.00'), Decimal('10.04')),
 (118, 30, Decimal('2600.00'), Decimal('24900.00'), Decimal('10.44')),
 (117, 30, Decimal('2800.00'), Decimal('24900.00'), Decimal('11.24')),
 (116, 30, Decimal('2900.00'), Decimal('24900.00'), Decimal('11.65')),
 (115, 30, Decimal('3100.00'), Decimal('24900.00'), Decimal('12.45')),
 (114, 30, Decimal('11000.00'), Decimal('24900.00'), Decimal('44.18')),
 (203, 40, Decimal('6500.00'), Decimal('6500.00'), Decimal('100.00'))]
```

```
%%sql

SELECT e.employee_id, e.department_id, e.salary,
    sum(e.salary) OVER (
        PARTITION BY e.department_id
    ) AS sum_sal_expense,
    round(avg(e.salary) OVER (
        PARTITION BY e.department_id
    ), 2) AS avg_sal_expense,
    min(e.salary) OVER (
        PARTITION BY e.department_id
    ) AS min_sal_expense,
    max(e.salary) OVER (
        PARTITION BY e.department_id
    ) AS max_sal_expense,
    count(e.salary) OVER (
        PARTITION BY e.department_id
    ) AS cnt_sal_expense
FROM employees e
ORDER BY e.department_id,
    e.salary
LIMIT 10
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
10 rows affected.
```

```
[(200, 10, Decimal('4400.00'), Decimal('4400.00'), Decimal('4400.00'), Decimal('4400.
↪00'), Decimal('4400.00'), 1),
 (202, 20, Decimal('6000.00'), Decimal('19000.00'), Decimal('9500.00'), Decimal('6000.
↪00'), Decimal('13000.00'), 2),
 (201, 20, Decimal('13000.00'), Decimal('19000.00'), Decimal('9500.00'), Decimal(
↪'6000.00'), Decimal('13000.00'), 2),
 (119, 30, Decimal('2500.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('2500.
↪00'), Decimal('11000.00'), 6),
 (118, 30, Decimal('2600.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('2500.
↪00'), Decimal('11000.00'), 6),
 (117, 30, Decimal('2800.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('2500.
↪00'), Decimal('11000.00'), 6),
 (116, 30, Decimal('2900.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('2500.
↪00'), Decimal('11000.00'), 6),
 (115, 30, Decimal('3100.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal('2500.
↪00'), Decimal('11000.00'), 6),
 (114, 30, Decimal('11000.00'), Decimal('24900.00'), Decimal('4150.00'), Decimal(
↪'2500.00'), Decimal('11000.00'), 6),
 (203, 40, Decimal('6500.00'), Decimal('6500.00'), Decimal('6500.00'), Decimal('6500.
↪00'), Decimal('6500.00'), 1)]
```

Warning: If you are using Jupyter based environment make sure to restart the kernel, as the session might have been already connected with hr database.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT
    order_date,
    order_item_product_id,
    revenue,
    sum(revenue) OVER (PARTITION BY order_date) AS sum_revenue,
    min(revenue) OVER (PARTITION BY order_date) AS min_revenue,
    max(revenue) OVER (PARTITION BY order_date) AS max_revenue
FROM daily_product_revenue
ORDER BY order_date,
    revenue DESC
LIMIT 10
```

```
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), Decimal('31547.23'),
↪ Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), Decimal('31547.23'),
↪ Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), Decimal('31547.23'),
↪ Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), Decimal('31547.23'),␣
↪Decimal('49.98'), Decimal('5599.72'))]
```

6.7.10 Cumulative or Moving Aggregations

Let us understand how we can take care of cumulative or moving aggregations using Analytic Functions.

- When it comes to Windowing or Analytic Functions we can also specify window spec using `ROWS BETWEEN` clause.
- Even when we do not specify window spec, the default window spec is used. For most of the functions the default window spec is `UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`. You also have special clauses such as `CURRENT ROW`.
- Here are some of the examples with respect to `ROWS BETWEEN`.
  - `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`
  - `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`

- `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`
- `ROWS BETWEEN 3 PRECEDING AND CURRENT ROW` - moving aggregations using current record and previous 3 records.
- `ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING` - moving aggregations using current record and following 3 records.
- `ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING` - moving aggregations based up on 7 records (current record, 3 previous records and 3 following records)

* We can leverage `ROWS BETWEEN` for cumulative aggregations or moving aggregations.
* Here is an example of cumulative sum.

Warning: If you are using Jupyter based environment make sure to restart the kernel, as the session might have been already connected with retail database.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
env: DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

Note: Even though it is not mandatory to specify `ORDER BY` as per syntax for cumulative aggregations, it is a must to specify. If not, you will end up getting incorrect results.

```
%%sql

SELECT e.employee_id, e.department_id, e.salary,
    sum(e.salary) OVER (
        PARTITION BY e.department_id
        ORDER BY e.salary
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS sum_sal_expense
FROM employees e
ORDER BY e.department_id, e.salary DESC
LIMIT 10
```

```
10 rows affected.
```

```
[(200, 10, Decimal('4400.00'), Decimal('4400.00')),
 (201, 20, Decimal('13000.00'), Decimal('19000.00')),
 (202, 20, Decimal('6000.00'), Decimal('6000.00')),
 (114, 30, Decimal('11000.00'), Decimal('24900.00')),
 (115, 30, Decimal('3100.00'), Decimal('13900.00')),
 (116, 30, Decimal('2900.00'), Decimal('10800.00')),
 (117, 30, Decimal('2800.00'), Decimal('7900.00')),
 (118, 30, Decimal('2600.00'), Decimal('5100.00')),
 (119, 30, Decimal('2500.00'), Decimal('2500.00')),
 (203, 40, Decimal('6500.00'), Decimal('6500.00'))]
```

Warning: If you are using Jupyter based environment make sure to restart the kernel, as the session might have been already connected with hr database.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Note: Here is the example for cumulative sum for every month using daily_product_revenue in retail database.

```
%%sql

SELECT t.*,
    round(sum(t.revenue) OVER (
        PARTITION BY to_char(order_date, 'yyyy-MM')
        ORDER BY order_date
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ), 2) AS cumulative_daily_revenue
FROM daily_revenue t
ORDER BY to_char(order_date, 'yyyy-MM'),
    order_date
LIMIT 10
```

```
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23'), Decimal('31547.23')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23'), Decimal('86260.46')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48'), Decimal('134671.94')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03'), Decimal('170343.97')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70'), Decimal('224923.67')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29'), Decimal('274252.96')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49'), Decimal('333465.45')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08'), Decimal('49160.08')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58'), Decimal('99848.66')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'), Decimal('143265.40'))]
```

Note: Here are examples for 3 day moving sum as well as average using daily_revenue in retail database.

```
%%sql

SELECT t.*,
    round(sum(t.revenue) OVER (
        ORDER BY order_date
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ), 2) AS moving_3day_revenue
FROM daily_revenue t
```

(continues on next page)

(continued from previous page)

```
ORDER BY order_date
LIMIT 20
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
20 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23'), Decimal('31547.23')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23'), Decimal('86260.46')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48'), Decimal('134671.94')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03'), Decimal('138796.74')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70'), Decimal('138663.21')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29'), Decimal('139581.02')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49'), Decimal('163121.48')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08'), Decimal('157701.86')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58'), Decimal('159061.15')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'), Decimal('143265.40')),
 (datetime.datetime(2013, 8, 4, 0, 0), Decimal('35093.01'), Decimal('129198.33')),
 (datetime.datetime(2013, 8, 5, 0, 0), Decimal('34025.27'), Decimal('112535.02')),
 (datetime.datetime(2013, 8, 6, 0, 0), Decimal('57843.89'), Decimal('126962.17')),
 (datetime.datetime(2013, 8, 7, 0, 0), Decimal('45525.59'), Decimal('137394.75')),
 (datetime.datetime(2013, 8, 8, 0, 0), Decimal('33549.47'), Decimal('136918.95')),
 (datetime.datetime(2013, 8, 9, 0, 0), Decimal('29225.16'), Decimal('108300.22')),
 (datetime.datetime(2013, 8, 10, 0, 0), Decimal('46435.04'), Decimal('109209.67')),
 (datetime.datetime(2013, 8, 11, 0, 0), Decimal('31155.50'), Decimal('106815.70')),
 (datetime.datetime(2013, 8, 12, 0, 0), Decimal('59014.74'), Decimal('136605.28')),
 (datetime.datetime(2013, 8, 13, 0, 0), Decimal('17956.88'), Decimal('108127.12'))]
```

```
%%sql

SELECT t.*,
    round(sum(t.revenue) OVER (
        ORDER BY order_date
        ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING
    ), 2) AS moving_3day_revenue
FROM daily_revenue t
ORDER BY order_date
LIMIT 20
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
20 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23'), Decimal('134671.94')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23'), Decimal('170343.97')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48'), Decimal('224923.67')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03'), Decimal('242705.73')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70'), Decimal('247204.99')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29'), Decimal('247953.59')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49'), Decimal('262970.14')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08'), Decimal('251807.18')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58'), Decimal('237570.90')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'), Decimal('212383.68')),
 (datetime.datetime(2013, 8, 4, 0, 0), Decimal('35093.01'), Decimal('221067.49')),
 (datetime.datetime(2013, 8, 5, 0, 0), Decimal('34025.27'), Decimal('215904.50')),
 (datetime.datetime(2013, 8, 6, 0, 0), Decimal('57843.89'), Decimal('206037.23')),
 (datetime.datetime(2013, 8, 7, 0, 0), Decimal('45525.59'), Decimal('200169.38')),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 8, 8, 0, 0), Decimal('33549.47'), Decimal('212579.15')),
 (datetime.datetime(2013, 8, 9, 0, 0), Decimal('29225.16'), Decimal('185890.76')),
 (datetime.datetime(2013, 8, 10, 0, 0), Decimal('46435.04'), Decimal('199379.91')),
 (datetime.datetime(2013, 8, 11, 0, 0), Decimal('31155.50'), Decimal('183787.32')),
 (datetime.datetime(2013, 8, 12, 0, 0), Decimal('59014.74'), Decimal('196605.61')),
 (datetime.datetime(2013, 8, 13, 0, 0), Decimal('17956.88'), Decimal('199737.25'))]
```

```
%%sql

SELECT t.*,
    round(avg(t.revenue) OVER (
        ORDER BY order_date
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ), 2) AS moving_3day_revenue
FROM daily_revenue t
ORDER BY order_date
LIMIT 20
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
20 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23'), Decimal('31547.23')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23'), Decimal('43130.23')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48'), Decimal('44890.65')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03'), Decimal('46265.58')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70'), Decimal('46221.07')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29'), Decimal('46527.01')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49'), Decimal('54373.83')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08'), Decimal('52567.29')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58'), Decimal('53020.38')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'), Decimal('47755.13')),
 (datetime.datetime(2013, 8, 4, 0, 0), Decimal('35093.01'), Decimal('43066.11')),
 (datetime.datetime(2013, 8, 5, 0, 0), Decimal('34025.27'), Decimal('37511.67')),
 (datetime.datetime(2013, 8, 6, 0, 0), Decimal('57843.89'), Decimal('42320.72')),
 (datetime.datetime(2013, 8, 7, 0, 0), Decimal('45525.59'), Decimal('45798.25')),
 (datetime.datetime(2013, 8, 8, 0, 0), Decimal('33549.47'), Decimal('45639.65')),
 (datetime.datetime(2013, 8, 9, 0, 0), Decimal('29225.16'), Decimal('36100.07')),
 (datetime.datetime(2013, 8, 10, 0, 0), Decimal('46435.04'), Decimal('36403.22')),
 (datetime.datetime(2013, 8, 11, 0, 0), Decimal('31155.50'), Decimal('35605.23')),
 (datetime.datetime(2013, 8, 12, 0, 0), Decimal('59014.74'), Decimal('45535.09')),
 (datetime.datetime(2013, 8, 13, 0, 0), Decimal('17956.88'), Decimal('36042.37'))]
```

6.7.11 Analytic Functions – Windowing

Let us go through the list of Windowing functions supported by Postgres.

- `lead` and `lag`
- `first_value` and `last_value`
- We can either use `ORDER BY sort_column` or `PARTITION BY partition_column ORDER BY sort_column` while using Windowing Functions.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Getting LEAD and LAG values

Let us understand LEAD and LAG functions to get column values from following or prior records.

Note: Here is the example to get values from either immediate prior or following record along with values from curent record. We will get values from prior or following record based on `ORDER BY` within `OVER` Clause.

```
%%sql

SELECT t.*,
    lead(order_date) OVER (ORDER BY order_date DESC) AS prior_date,
    lead(revenue) OVER (ORDER BY order_date DESC) AS prior_revenue,
    lag(order_date) OVER (ORDER BY order_date) AS lag_prior_date,
    lag(revenue) OVER (ORDER BY order_date) AS lag_prior_revenue
FROM daily_revenue AS t
ORDER BY order_date DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2014, 7, 24, 0, 0), Decimal('50885.19'), datetime.datetime(2014,␣
↪7, 23, 0, 0), Decimal('38795.23'), datetime.datetime(2014, 7, 23, 0, 0), Decimal(
↪'38795.23')),
 (datetime.datetime(2014, 7, 23, 0, 0), Decimal('38795.23'), datetime.datetime(2014,␣
↪7, 22, 0, 0), Decimal('36717.24'), datetime.datetime(2014, 7, 22, 0, 0), Decimal(
↪'36717.24')),
 (datetime.datetime(2014, 7, 22, 0, 0), Decimal('36717.24'), datetime.datetime(2014,␣
↪7, 21, 0, 0), Decimal('51427.70'), datetime.datetime(2014, 7, 21, 0, 0), Decimal(
↪'51427.70')),
 (datetime.datetime(2014, 7, 21, 0, 0), Decimal('51427.70'), datetime.datetime(2014,␣
↪7, 20, 0, 0), Decimal('60047.45'), datetime.datetime(2014, 7, 20, 0, 0), Decimal(
↪'60047.45')),
 (datetime.datetime(2014, 7, 20, 0, 0), Decimal('60047.45'), datetime.datetime(2014,␣
↪7, 19, 0, 0), Decimal('38420.99'), datetime.datetime(2014, 7, 19, 0, 0), Decimal(
↪'38420.99')),
 (datetime.datetime(2014, 7, 19, 0, 0), Decimal('38420.99'), datetime.datetime(2014,␣
↪7, 18, 0, 0), Decimal('43856.60'), datetime.datetime(2014, 7, 18, 0, 0), Decimal(
↪'43856.60')),
 (datetime.datetime(2014, 7, 18, 0, 0), Decimal('43856.60'), datetime.datetime(2014,␣
↪7, 17, 0, 0), Decimal('36384.77'), datetime.datetime(2014, 7, 17, 0, 0), Decimal(
↪'36384.77')),
 (datetime.datetime(2014, 7, 17, 0, 0), Decimal('36384.77'), datetime.datetime(2014,␣
↪7, 16, 0, 0), Decimal('43011.92'), datetime.datetime(2014, 7, 16, 0, 0), Decimal(
↪'43011.92')),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2014, 7, 16, 0, 0), Decimal('43011.92'), datetime.datetime(2014,␣
↪7, 15, 0, 0), Decimal('53480.23'), datetime.datetime(2014, 7, 15, 0, 0), Decimal(
↪'53480.23')),
 (datetime.datetime(2014, 7, 15, 0, 0), Decimal('53480.23'), datetime.datetime(2014,␣
↪7, 14, 0, 0), Decimal('29937.52'), datetime.datetime(2014, 7, 14, 0, 0), Decimal(
↪'29937.52'))]
```

Note: Here is the example to get values from either prior or following 7th record along with values from current record.

```
%%sql

SELECT t.*,
    lead(order_date, 7) OVER (ORDER BY order_date DESC) AS prior_date,
    lead(revenue, 7) OVER (ORDER BY order_date DESC) AS prior_revenue
FROM daily_revenue t
ORDER BY order_date DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2014, 7, 24, 0, 0), Decimal('50885.19'), datetime.datetime(2014,␣
↪7, 17, 0, 0), Decimal('36384.77')),
 (datetime.datetime(2014, 7, 23, 0, 0), Decimal('38795.23'), datetime.datetime(2014,␣
↪7, 16, 0, 0), Decimal('43011.92')),
 (datetime.datetime(2014, 7, 22, 0, 0), Decimal('36717.24'), datetime.datetime(2014,␣
↪7, 15, 0, 0), Decimal('53480.23')),
 (datetime.datetime(2014, 7, 21, 0, 0), Decimal('51427.70'), datetime.datetime(2014,␣
↪7, 14, 0, 0), Decimal('29937.52')),
 (datetime.datetime(2014, 7, 20, 0, 0), Decimal('60047.45'), datetime.datetime(2014,␣
↪7, 13, 0, 0), Decimal('40410.99')),
 (datetime.datetime(2014, 7, 19, 0, 0), Decimal('38420.99'), datetime.datetime(2014,␣
↪7, 12, 0, 0), Decimal('38449.77')),
 (datetime.datetime(2014, 7, 18, 0, 0), Decimal('43856.60'), datetime.datetime(2014,␣
↪7, 11, 0, 0), Decimal('29596.32')),
 (datetime.datetime(2014, 7, 17, 0, 0), Decimal('36384.77'), datetime.datetime(2014,␣
↪7, 10, 0, 0), Decimal('47826.02')),
 (datetime.datetime(2014, 7, 16, 0, 0), Decimal('43011.92'), datetime.datetime(2014,␣
↪7, 9, 0, 0), Decimal('36929.91')),
 (datetime.datetime(2014, 7, 15, 0, 0), Decimal('53480.23'), datetime.datetime(2014,␣
↪7, 8, 0, 0), Decimal('50434.81'))]
```

Note: For values related to non existing prior or following record, we will get nulls.

```
%%sql

SELECT t.*,
    lead(order_date, 7) OVER (ORDER BY order_date DESC) AS prior_date,
    lead(revenue, 7) OVER (ORDER BY order_date DESC) AS prior_revenue
FROM daily_revenue t
```

(continues on next page)

(continued from previous page)

```
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23'), None, None),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23'), None, None),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48'), None, None),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03'), None, None),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70'), None, None),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29'), None, None),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49'), None, None),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08'), datetime.datetime(2013, 7,
↪ 25, 0, 0), Decimal('31547.23')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58'), datetime.datetime(2013, 7,
↪ 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'), datetime.datetime(2013, 7,
↪ 27, 0, 0), Decimal('48411.48'))]
```

Note: We can replace nulls by passing relevant values as 3rd argument. However, the data type of the values should be compatible with the columns on which `lead` or `lag` is applied.

```
%%sql

SELECT t.*,
    lead(order_date, 7) OVER (ORDER BY order_date DESC) AS prior_date,
    lead(revenue, 7, 0.0) OVER (ORDER BY order_date DESC) AS prior_revenue
FROM daily_revenue t
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23'), None, Decimal('0.0')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23'), None, Decimal('0.0')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48'), None, Decimal('0.0')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03'), None, Decimal('0.0')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70'), None, Decimal('0.0')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29'), None, Decimal('0.0')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49'), None, Decimal('0.0')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08'), datetime.datetime(2013, 7,
↪ 25, 0, 0), Decimal('31547.23')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58'), datetime.datetime(2013, 7,
↪ 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'), datetime.datetime(2013, 7,
↪ 27, 0, 0), Decimal('48411.48'))]
```

```
%%sql

SELECT * FROM daily_product_revenue
```

(continues on next page)

(continued from previous page)

```
ORDER BY order_date, revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'))]
```

```
%%sql

SELECT t.*,
    LEAD(order_item_product_id) OVER (
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) next_product_id,
    LEAD(revenue) OVER (
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) next_revenue
FROM daily_product_revenue t
ORDER BY order_date, revenue DESC
LIMIT 30
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 191, Decimal('5099.
↪49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 957, Decimal('4499.70
↪')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 365, Decimal('3359.44
↪')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 1073, Decimal('2999.
↪85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 1014, Decimal('2798.
↪88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), 403, Decimal('1949.
↪85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), 502, Decimal('1650.00
↪')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), 627, Decimal('1079.73
↪')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), 226, Decimal('599.99
↪')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), 24, Decimal('319.96
↪')),
```

(continues on next page)

(continued from previous page)

```
(datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96'), 821, Decimal('207.96
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96'), 625, Decimal('199.99
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99'), 705, Decimal('119.99
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99'), 572, Decimal('119.97
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97'), 666, Decimal('109.99
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99'), 725, Decimal('108.00
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00'), 134, Decimal('100.00
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00'), 906, Decimal('99.96
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96'), 828, Decimal('95.97')),
(datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97'), 810, Decimal('79.96')),
(datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96'), 924, Decimal('79.95')),
(datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95'), 926, Decimal('79.95')),
(datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95'), 93, Decimal('74.97')),
(datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97'), 835, Decimal('63.98')),
(datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98'), 897, Decimal('49.98')),
(datetime.datetime(2013, 7, 25, 0, 0), 897, Decimal('49.98'), None, None),
(datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 365, Decimal('7978.
↪67')),
(datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 957, Decimal('6899.54
↪')),
(datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 191, Decimal('6799.32
↪')),
(datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 1014, Decimal('4798.
↪08'))]
```

Getting first and last values

Let us see how we can get first and last value based on the criteria. `min` or `max` can be used to get only the min or max of the metric we are interested in, however we cannot get other attributes of those records.

Here is the example of using first_value.

```
%%sql

SELECT t.*,
    first_value(order_item_product_id) OVER (
        PARTITION BY order_date ORDER BY revenue DESC
    ) first_product_id,
    first_value(revenue) OVER (
        PARTITION BY order_date ORDER BY revenue DESC
    ) first_revenue,
    max(revenue) OVER (
        PARTITION BY order_date
    ) max_revenue
FROM daily_product_revenue t
ORDER BY order_date, revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), 1004, Decimal('5599.
↪72'), Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), 1004, Decimal('5599.72
↪'), Decimal('5599.72'))]
```

Let us see an example with last_value. While using last_value we need to specify **ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING**.

- By default it uses `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.
- The last value with in `UNBOUNDED PRECEDING AND CURRENT ROW` will be current record.
- To get the right value, we have to change the windowing clause to `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

```
%%sql

SELECT t.*,
    last_value(order_item_product_id) OVER (
        PARTITION BY order_date ORDER BY revenue
        ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
    ) last_product_id,
    max(revenue) OVER (
        PARTITION BY order_date
    ) last_revenue
FROM daily_product_revenue AS t
ORDER BY order_date, revenue DESC
LIMIT 30
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1004, Decimal('5599.
↪72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 1004, Decimal('5599.
↪72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 1004, Decimal('5599.
↪72')),
```

(continues on next page)

(continued from previous page)

```
(datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 1004, Decimal('5599.
↪72')),
(datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 1004, Decimal('5599.
↪72')),
(datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), 1004, Decimal('5599.
↪72')),
(datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), 1004, Decimal('5599.
↪72')),
(datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), 1004, Decimal('5599.
↪72')),
(datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), 1004, Decimal('5599.
↪72')),
(datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 25, 0, 0), 897, Decimal('49.98'), 1004, Decimal('5599.72
↪')),
(datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1004, Decimal(
↪'10799.46')),
(datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 1004, Decimal('10799.
↪46')),
(datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 1004, Decimal('10799.
↪46')),
(datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 1004, Decimal('10799.
↪46'))]
```

6.7.12 Analytic Functions – Ranking

Let us see how we can assign ranks using different **rank** functions.

- If we have to assign ranks globally, we just need to specify **ORDER BY**
- If we have to assign ranks with in a key then we need to specify **PARTITION BY** and then **ORDER BY**.
- By default **ORDER BY** will sort the data in ascending order. We can change the order by passing **DESC** after order by.
- We have 3 main functions to assign ranks - `rank`, `dense_rank` and `row_number`. We will see the differences between the 3 in a moment.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Note: Here is an example to assign sparse ranks using daily_product_revenue with in each day based on revenue.

```
%%sql

SELECT t.*,
    rank() OVER (
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) AS rnk
FROM daily_product_revenue t
ORDER BY order_date, revenue DESC
LIMIT 30
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), 6),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), 7),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), 8),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), 9),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), 10),
 (datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96'), 11),
 (datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96'), 12),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99'), 13),
 (datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99'), 14),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97'), 15),
 (datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99'), 16),
 (datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00'), 17),
 (datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00'), 18),
 (datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96'), 19),
 (datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97'), 20),
 (datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96'), 21),
 (datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95'), 22),
 (datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95'), 22),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97'), 24),
 (datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98'), 25),
 (datetime.datetime(2013, 7, 25, 0, 0), 897, Decimal('49.98'), 26),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4)]
```

Note: Here is another example to assign sparse ranks using employees data set with in each department. Make sure to restart kernel as you might have connected to retail database.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
env: DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
%%sql

SELECT employee_id, department_id, salary FROM employees
ORDER BY department_id,
    salary DESC
LIMIT 10
```

```
10 rows affected.
```

```
[(200, 10, Decimal('4400.00')),
 (201, 20, Decimal('13000.00')),
 (202, 20, Decimal('6000.00')),
 (114, 30, Decimal('11000.00')),
 (115, 30, Decimal('3100.00')),
 (116, 30, Decimal('2900.00')),
 (117, 30, Decimal('2800.00')),
 (118, 30, Decimal('2600.00')),
 (119, 30, Decimal('2500.00')),
 (203, 40, Decimal('6500.00'))]
```

```
%%sql

SELECT employee_id, department_id, salary,
    rank() OVER (
```

(continues on next page)

(continued from previous page)

```
        PARTITION BY department_id
        ORDER BY salary DESC
    ) AS rnk
FROM employees
LIMIT 20
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
20 rows affected.
```

```
[(200, 10, Decimal('4400.00'), 1),
 (201, 20, Decimal('13000.00'), 1),
 (202, 20, Decimal('6000.00'), 2),
 (114, 30, Decimal('11000.00'), 1),
 (115, 30, Decimal('3100.00'), 2),
 (116, 30, Decimal('2900.00'), 3),
 (117, 30, Decimal('2800.00'), 4),
 (118, 30, Decimal('2600.00'), 5),
 (119, 30, Decimal('2500.00'), 6),
 (203, 40, Decimal('6500.00'), 1),
 (121, 50, Decimal('8200.00'), 1),
 (120, 50, Decimal('8000.00'), 2),
 (122, 50, Decimal('7900.00'), 3),
 (123, 50, Decimal('6500.00'), 4),
 (124, 50, Decimal('5800.00'), 5),
 (184, 50, Decimal('4200.00'), 6),
 (185, 50, Decimal('4100.00'), 7),
 (192, 50, Decimal('4000.00'), 8),
 (193, 50, Decimal('3900.00'), 9),
 (188, 50, Decimal('3800.00'), 10)]
```

Note: Here is an example to assign dense ranks using employees data set with in each department.

```
%%sql

SELECT employee_id, department_id, salary,
    dense_rank() OVER (
        PARTITION BY department_id
        ORDER BY salary DESC
    ) AS drnk
FROM employees
LIMIT 20
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
20 rows affected.
```

```
[(200, 10, Decimal('4400.00'), 1),
 (201, 20, Decimal('13000.00'), 1),
 (202, 20, Decimal('6000.00'), 2),
 (114, 30, Decimal('11000.00'), 1),
 (115, 30, Decimal('3100.00'), 2),
 (116, 30, Decimal('2900.00'), 3),
 (117, 30, Decimal('2800.00'), 4),
 (118, 30, Decimal('2600.00'), 5),
```

(continues on next page)

(continued from previous page)

```
 (119, 30, Decimal('2500.00'), 6),
 (203, 40, Decimal('6500.00'), 1),
 (121, 50, Decimal('8200.00'), 1),
 (120, 50, Decimal('8000.00'), 2),
 (122, 50, Decimal('7900.00'), 3),
 (123, 50, Decimal('6500.00'), 4),
 (124, 50, Decimal('5800.00'), 5),
 (184, 50, Decimal('4200.00'), 6),
 (185, 50, Decimal('4100.00'), 7),
 (192, 50, Decimal('4000.00'), 8),
 (193, 50, Decimal('3900.00'), 9),
 (188, 50, Decimal('3800.00'), 10)]
```

Note: Here is an example for global rank based on salary. If all the salaries are unique, we can use `LIMIT` but when they are not unique, we have to go with analytic functions.

```
%%sql

SELECT employee_id, department_id, salary,
    rank() OVER (
        ORDER BY salary DESC
    ) AS rnk,
    dense_rank() OVER (
        ORDER BY salary DESC
    ) AS drnk
FROM employees
LIMIT 20
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
20 rows affected.
```

```
[(100, 90, Decimal('24000.00'), 1, 1),
 (101, 90, Decimal('17000.00'), 2, 2),
 (102, 90, Decimal('17000.00'), 2, 2),
 (145, 80, Decimal('14000.00'), 4, 3),
 (146, 80, Decimal('13500.00'), 5, 4),
 (201, 20, Decimal('13000.00'), 6, 5),
 (205, 110, Decimal('12000.00'), 7, 6),
 (147, 80, Decimal('12000.00'), 7, 6),
 (108, 100, Decimal('12000.00'), 7, 6),
 (168, 80, Decimal('11500.00'), 10, 7),
 (148, 80, Decimal('11000.00'), 11, 8),
 (174, 80, Decimal('11000.00'), 11, 8),
 (114, 30, Decimal('11000.00'), 11, 8),
 (149, 80, Decimal('10500.00'), 14, 9),
 (162, 80, Decimal('10500.00'), 14, 9),
 (169, 80, Decimal('10000.00'), 16, 10),
 (204, 70, Decimal('10000.00'), 16, 10),
 (150, 80, Decimal('10000.00'), 16, 10),
 (156, 80, Decimal('10000.00'), 16, 10),
 (170, 80, Decimal('9600.00'), 20, 11)]
```

Let us understand the difference between **rank**, **dense_rank** and **row_number**.

- We can use either of the functions to generate ranks when the rank field does not have duplicates.

- When rank field have duplicates then row_number should not be used as it generate unique number for each record with in the partition.
- **rank** will skip the ranks in between if multiple people get the same rank while **dense_rank** continue with the next number.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
env: DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
%%sql

SELECT
    employee_id,
    department_id,
    salary,
    rank() OVER (
        PARTITION BY department_id
        ORDER BY salary DESC
      ) rnk,
    dense_rank() OVER (
        PARTITION BY department_id
        ORDER BY salary DESC
      ) drnk,
    row_number() OVER (
        PARTITION BY department_id
        ORDER BY salary DESC, employee_id
      ) rn
FROM employees
ORDER BY department_id, salary DESC
LIMIT 50
```

```
 * postgresql://itversity_hr_user:***@localhost:5432/itversity_hr_db
50 rows affected.
```

```
[(200, 10, Decimal('4400.00'), 1, 1, 1),
 (201, 20, Decimal('13000.00'), 1, 1, 1),
 (202, 20, Decimal('6000.00'), 2, 2, 2),
 (114, 30, Decimal('11000.00'), 1, 1, 1),
 (115, 30, Decimal('3100.00'), 2, 2, 2),
 (116, 30, Decimal('2900.00'), 3, 3, 3),
 (117, 30, Decimal('2800.00'), 4, 4, 4),
 (118, 30, Decimal('2600.00'), 5, 5, 5),
 (119, 30, Decimal('2500.00'), 6, 6, 6),
 (203, 40, Decimal('6500.00'), 1, 1, 1),
 (121, 50, Decimal('8200.00'), 1, 1, 1),
 (120, 50, Decimal('8000.00'), 2, 2, 2),
 (122, 50, Decimal('7900.00'), 3, 3, 3),
 (123, 50, Decimal('6500.00'), 4, 4, 4),
```

(continues on next page)

(continued from previous page)

```
(124, 50, Decimal('5800.00'), 5, 5, 5),
(184, 50, Decimal('4200.00'), 6, 6, 6),
(185, 50, Decimal('4100.00'), 7, 7, 7),
(192, 50, Decimal('4000.00'), 8, 8, 8),
(193, 50, Decimal('3900.00'), 9, 9, 9),
(188, 50, Decimal('3800.00'), 10, 10, 10),
(137, 50, Decimal('3600.00'), 11, 11, 11),
(189, 50, Decimal('3600.00'), 11, 11, 12),
(141, 50, Decimal('3500.00'), 13, 12, 13),
(186, 50, Decimal('3400.00'), 14, 13, 14),
(129, 50, Decimal('3300.00'), 15, 14, 15),
(133, 50, Decimal('3300.00'), 15, 14, 16),
(125, 50, Decimal('3200.00'), 17, 15, 17),
(138, 50, Decimal('3200.00'), 17, 15, 18),
(180, 50, Decimal('3200.00'), 17, 15, 19),
(194, 50, Decimal('3200.00'), 17, 15, 20),
(142, 50, Decimal('3100.00'), 21, 16, 21),
(181, 50, Decimal('3100.00'), 21, 16, 22),
(196, 50, Decimal('3100.00'), 21, 16, 23),
(187, 50, Decimal('3000.00'), 24, 17, 24),
(197, 50, Decimal('3000.00'), 24, 17, 25),
(134, 50, Decimal('2900.00'), 26, 18, 26),
(190, 50, Decimal('2900.00'), 26, 18, 27),
(130, 50, Decimal('2800.00'), 28, 19, 28),
(183, 50, Decimal('2800.00'), 28, 19, 29),
(195, 50, Decimal('2800.00'), 28, 19, 30),
(126, 50, Decimal('2700.00'), 31, 20, 31),
(139, 50, Decimal('2700.00'), 31, 20, 32),
(143, 50, Decimal('2600.00'), 33, 21, 33),
(198, 50, Decimal('2600.00'), 33, 21, 34),
(199, 50, Decimal('2600.00'), 33, 21, 35),
(131, 50, Decimal('2500.00'), 36, 22, 36),
(140, 50, Decimal('2500.00'), 36, 22, 37),
(144, 50, Decimal('2500.00'), 36, 22, 38),
(182, 50, Decimal('2500.00'), 36, 22, 39),
(191, 50, Decimal('2500.00'), 36, 22, 40)]
```

Note: Here is another example to with respect to all 3 functions. Make sure to restart kernel as you might have connected to HR database.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT
    t.*,
    rank() OVER (
```

(continues on next page)

(continued from previous page)

```
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) rnk,
    dense_rank() OVER (
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) drnk,
    row_number() OVER (
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) rn
FROM daily_product_revenue AS t
ORDER BY order_date, revenue DESC
LIMIT 30
```

```
30 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1, 1, 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2, 2, 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3, 3, 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4, 4, 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5, 5, 5),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), 6, 6, 6),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), 7, 7, 7),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), 8, 8, 8),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), 9, 9, 9),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), 10, 10, 10),
 (datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96'), 11, 11, 11),
 (datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96'), 12, 12, 12),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99'), 13, 13, 13),
 (datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99'), 14, 14, 14),
 (datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97'), 15, 15, 15),
 (datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99'), 16, 16, 16),
 (datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00'), 17, 17, 17),
 (datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00'), 18, 18, 18),
 (datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96'), 19, 19, 19),
 (datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97'), 20, 20, 20),
 (datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96'), 21, 21, 21),
 (datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95'), 22, 22, 22),
 (datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95'), 22, 22, 23),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97'), 24, 23, 24),
 (datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98'), 25, 24, 25),
 (datetime.datetime(2013, 7, 25, 0, 0), 897, Decimal('49.98'), 26, 25, 26),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1, 1, 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2, 2, 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3, 3, 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4, 4, 4)]
```

6.7.13 Analytic Functions - Filtering

Let us go through the solution for getting top 5 daily products based up on the revenue. In that process we will understand how to apply filtering on top of the derived values using analytic functions.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

Order of execution of SQL

Let us review the order of execution of SQL. First let us review the order of writing the query.

1. **SELECT**
2. **FROM**
3. **JOIN** or **OUTER JOIN** with **ON**
4. **WHERE**
5. **GROUP BY** and optionally **HAVING**
6. **ORDER BY**

Let us come up with a query which will compute daily revenue using COMPLETE or CLOSED orders and also sorted by order_date.

```
%%sql

SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date
ORDER BY o.order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49')),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'))]
```

```
%%sql

SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date
    HAVING round(sum(oi.order_item_subtotal)::numeric, 2) >= 50000
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58')),
 (datetime.datetime(2013, 8, 6, 0, 0), Decimal('57843.89')),
 (datetime.datetime(2013, 8, 12, 0, 0), Decimal('59014.74')),
 (datetime.datetime(2013, 8, 17, 0, 0), Decimal('63226.83')),
 (datetime.datetime(2013, 8, 24, 0, 0), Decimal('52650.15')),
 (datetime.datetime(2013, 9, 5, 0, 0), Decimal('59942.43')),
 (datetime.datetime(2013, 9, 6, 0, 0), Decimal('61976.10'))]
```

However order of execution is typically as follows.

1. **FROM**
2. **JOIN** or **OUTER JOIN** with **ON**
3. **WHERE**
4. **GROUP BY** and optionally **HAVING**
5. **SELECT**
6. **ORDER BY**

As **SELECT** is executed before **ORDER BY** clause, we will not be able to refer the aliases defined in **SELECT** caluse in other clauses except for **ORDER BY** in most of the traditional databases including Postgresql.

Error: This will fail as revenue which is an alias defined in **SELECT** cannot be used in **WHERE**.

```
%%sql

SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
```

(continues on next page)

(continued from previous page)

```
    AND revenue >= 50000
GROUP BY o.order_date
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedColumn) column "revenue" does not exist
LINE 5:     AND revenue >= 50000
                ^

[SQL: SELECT o.order_date, round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND revenue >= 50000
GROUP BY o.order_date
ORDER BY order_date
LIMIT 10]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

Note: This will also fail as we cannot use aggregate functions in `WHERE` clause.

```
%%sql

SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND round(sum(oi.order_item_subtotal)::numeric, 2) >= 50000
GROUP BY o.order_date
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.GroupingError) aggregate functions are not allowed in WHERE
LINE 5:     AND round(sum(oi.order_item_subtotal)::numeric, 2) >= 50...
                      ^

[SQL: SELECT o.order_date, round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    AND round(sum(oi.order_item_subtotal)::numeric, 2) >= 50000
GROUP BY o.order_date
ORDER BY order_date
LIMIT 10]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
```

(continues on next page)

(continued from previous page)

```
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date
ORDER BY order_date,
    revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), Decimal('31547.23')),
 (datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 7, 27, 0, 0), Decimal('48411.48')),
 (datetime.datetime(2013, 7, 28, 0, 0), Decimal('35672.03')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70')),
 (datetime.datetime(2013, 7, 30, 0, 0), Decimal('49329.29')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49')),
 (datetime.datetime(2013, 8, 1, 0, 0), Decimal('49160.08')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58')),
 (datetime.datetime(2013, 8, 3, 0, 0), Decimal('43416.74'))]
```

```
%%sql

SELECT o.order_date,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date
    HAVING round(sum(oi.order_item_subtotal)::numeric, 2) >= 50000
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 26, 0, 0), Decimal('54713.23')),
 (datetime.datetime(2013, 7, 29, 0, 0), Decimal('54579.70')),
 (datetime.datetime(2013, 7, 31, 0, 0), Decimal('59212.49')),
 (datetime.datetime(2013, 8, 2, 0, 0), Decimal('50688.58')),
 (datetime.datetime(2013, 8, 6, 0, 0), Decimal('57843.89')),
 (datetime.datetime(2013, 8, 12, 0, 0), Decimal('59014.74')),
 (datetime.datetime(2013, 8, 17, 0, 0), Decimal('63226.83')),
 (datetime.datetime(2013, 8, 24, 0, 0), Decimal('52650.15')),
 (datetime.datetime(2013, 9, 5, 0, 0), Decimal('59942.43')),
 (datetime.datetime(2013, 9, 6, 0, 0), Decimal('61976.10'))]
```

Error: This one will also fail as we are trying to use alias `drnk` from `SELECT` clause in `WHERE` clause.

```
%%sql
```

(continues on next page)

(continued from previous page)

```
SELECT t.*,
dense_rank() OVER (
  PARTITION BY order_date
  ORDER BY revenue DESC
) AS drnk
FROM daily_product_revenue t
WHERE drnk <= 5
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedColumn) column "drnk" does not exist
LINE 6: WHERE drnk <= 5
              ^

[SQL: SELECT t.*, dense_rank() OVER (
  PARTITION BY order_date
  ORDER BY revenue DESC
) AS drnk
FROM daily_product_revenue t
WHERE drnk <= 5]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

Overview of Sub Queries

Let us recap about Sub Queries.

- We typically have Sub Queries in **FROM** Clause.
- We need to provide alias to the Sub Queries in **FROM** Clause in Postgresql.
- We use sub queries quite often over queries using Analytics/Windowing Functions

```
%%sql

SELECT * FROM (SELECT current_date) AS q
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
1 rows affected.
```

```
[(datetime.date(2020, 12, 1),)]
```

Let us see few more examples with respect to Sub Queries.

```
%%sql

SELECT * FROM (
  SELECT order_date, count(1) AS order_count
  FROM orders
  GROUP BY order_date
) AS q
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 143),
 (datetime.datetime(2013, 7, 26, 0, 0), 269),
 (datetime.datetime(2013, 7, 27, 0, 0), 202),
 (datetime.datetime(2013, 7, 28, 0, 0), 187),
 (datetime.datetime(2013, 7, 29, 0, 0), 253),
 (datetime.datetime(2013, 7, 30, 0, 0), 227),
 (datetime.datetime(2013, 7, 31, 0, 0), 252),
 (datetime.datetime(2013, 8, 1, 0, 0), 246),
 (datetime.datetime(2013, 8, 2, 0, 0), 224),
 (datetime.datetime(2013, 8, 3, 0, 0), 183)]
```

```
%%sql

SELECT * FROM (
  SELECT order_date, count(1) AS order_count
  FROM orders
  GROUP BY order_date
) q
WHERE q.order_count > 150
ORDER BY order_date
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 26, 0, 0), 269),
 (datetime.datetime(2013, 7, 27, 0, 0), 202),
 (datetime.datetime(2013, 7, 28, 0, 0), 187),
 (datetime.datetime(2013, 7, 29, 0, 0), 253),
 (datetime.datetime(2013, 7, 30, 0, 0), 227),
 (datetime.datetime(2013, 7, 31, 0, 0), 252),
 (datetime.datetime(2013, 8, 1, 0, 0), 246),
 (datetime.datetime(2013, 8, 2, 0, 0), 224),
 (datetime.datetime(2013, 8, 3, 0, 0), 183),
 (datetime.datetime(2013, 8, 4, 0, 0), 187)]
```

Note: Above query is an example for sub queries. We can achieve using HAVING clause (no need to have sub query to filter)

Filtering - Analytic Function Results

Let us understand how to filter on top of results of Analytic Functions.

- We can use Analytic Functions only in **SELECT** Clause.
- If we have to filter based on Analytic Function results, then we need to use Sub Queries.
- Once the query is added as subquery, we can apply filter using aliases of the Analytic Functions.

Here is the example where we can filter data based on Analytic Functions.

```
%%sql

SELECT t.*,
```

(continues on next page)

(continued from previous page)

```
dense_rank() OVER (
  PARTITION BY order_date
  ORDER BY revenue DESC
) AS drnk
FROM daily_product_revenue t
WHERE drnk <= 5
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
(psycopg2.errors.UndefinedColumn) column "drnk" does not exist
LINE 6: WHERE drnk <= 5
              ^

[SQL: SELECT t.*, dense_rank() OVER (
  PARTITION BY order_date
  ORDER BY revenue DESC
) AS drnk
FROM daily_product_revenue t
WHERE drnk <= 5]
(Background on this error at: http://sqlalche.me/e/13/f405)
```

```
%%sql

SELECT * FROM (
  SELECT t.*,
    dense_rank() OVER (
      PARTITION BY order_date
      ORDER BY revenue DESC
    ) AS drnk
  FROM daily_product_revenue t
) q
WHERE q.drnk <= 5
ORDER BY q.order_date, q.revenue DESC
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4),
 (datetime.datetime(2013, 7, 26, 0, 0), 1014, Decimal('4798.08'), 5)]
```

6.7.14 Ranking and Filtering - Recap

Let us recap the procedure to get top 5 products by revenue for each day.

- We have our original data in **orders** and **order_items**
- We can pre-compute the data or create a view with the logic to generate **daily product revenue**
- Then, we have to use the view or table or even sub query to compute rank
- Once the ranks are computed, we need to use sub query to filter based up on our requirement.

Let us come up with the query to compute daily product revenue.

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

SELECT o.order_date,
       oi.order_item_product_id,
       round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_item_order_id
WHERE o.order_status IN ('COMPLETE', 'CLOSED')
GROUP BY o.order_date, oi.order_item_product_id
ORDER BY o.order_date, revenue DESC
LIMIT 30
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72')),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49')),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70')),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88')),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85')),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73')),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99')),
 (datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00')),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00')),
 (datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96')),
 (datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95')),
 (datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95')),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97')),
 (datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98')),
 (datetime.datetime(2013, 7, 25, 0, 0), 897, Decimal('49.98')),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46')),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67')),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54')),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'))]
```

Let us compute the rank for each product with in each date using revenue as criteria.

```
%%sql


SELECT nq.*,
    dense_rank() OVER (
        PARTITION BY order_date
        ORDER BY revenue DESC
    ) AS drnk
FROM (
    SELECT o.order_date,
        oi.order_item_product_id,
        round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
    FROM orders o
        JOIN order_items oi
            ON o.order_id = oi.order_item_order_id
    WHERE o.order_status IN ('COMPLETE', 'CLOSED')
    GROUP BY o.order_date, oi.order_item_product_id
) nq
ORDER BY order_date, revenue DESC
LIMIT 30
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
30 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5),
 (datetime.datetime(2013, 7, 25, 0, 0), 1014, Decimal('2798.88'), 6),
 (datetime.datetime(2013, 7, 25, 0, 0), 403, Decimal('1949.85'), 7),
 (datetime.datetime(2013, 7, 25, 0, 0), 502, Decimal('1650.00'), 8),
 (datetime.datetime(2013, 7, 25, 0, 0), 627, Decimal('1079.73'), 9),
 (datetime.datetime(2013, 7, 25, 0, 0), 226, Decimal('599.99'), 10),
 (datetime.datetime(2013, 7, 25, 0, 0), 24, Decimal('319.96'), 11),
 (datetime.datetime(2013, 7, 25, 0, 0), 821, Decimal('207.96'), 12),
 (datetime.datetime(2013, 7, 25, 0, 0), 625, Decimal('199.99'), 13),
 (datetime.datetime(2013, 7, 25, 0, 0), 705, Decimal('119.99'), 14),
 (datetime.datetime(2013, 7, 25, 0, 0), 572, Decimal('119.97'), 15),
 (datetime.datetime(2013, 7, 25, 0, 0), 666, Decimal('109.99'), 16),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 7, 25, 0, 0), 725, Decimal('108.00'), 17),
 (datetime.datetime(2013, 7, 25, 0, 0), 134, Decimal('100.00'), 18),
 (datetime.datetime(2013, 7, 25, 0, 0), 906, Decimal('99.96'), 19),
 (datetime.datetime(2013, 7, 25, 0, 0), 828, Decimal('95.97'), 20),
 (datetime.datetime(2013, 7, 25, 0, 0), 810, Decimal('79.96'), 21),
 (datetime.datetime(2013, 7, 25, 0, 0), 924, Decimal('79.95'), 22),
 (datetime.datetime(2013, 7, 25, 0, 0), 926, Decimal('79.95'), 22),
 (datetime.datetime(2013, 7, 25, 0, 0), 93, Decimal('74.97'), 23),
 (datetime.datetime(2013, 7, 25, 0, 0), 835, Decimal('63.98'), 24),
 (datetime.datetime(2013, 7, 25, 0, 0), 897, Decimal('49.98'), 25),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4)]
```

Now let us see how we can filter the data.

```
%%sql

SELECT * FROM (
    SELECT nq.*,
        dense_rank() OVER (
            PARTITION BY order_date
            ORDER BY revenue DESC
        ) AS drnk
    FROM (
        SELECT o.order_date,
            oi.order_item_product_id,
            round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
        FROM orders o
            JOIN order_items oi
                ON o.order_id = oi.order_item_order_id
        WHERE o.order_status IN ('COMPLETE', 'CLOSED')
        GROUP BY o.order_date, oi.order_item_product_id
    ) nq
) nq1
WHERE drnk <= 5
ORDER BY order_date, revenue DESC
LIMIT 20
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
20 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4),
 (datetime.datetime(2013, 7, 26, 0, 0), 1014, Decimal('4798.08'), 5),
 (datetime.datetime(2013, 7, 27, 0, 0), 1004, Decimal('9599.52'), 1),
 (datetime.datetime(2013, 7, 27, 0, 0), 191, Decimal('5999.40'), 2),
 (datetime.datetime(2013, 7, 27, 0, 0), 957, Decimal('5699.62'), 3),
```

(continues on next page)

(continued from previous page)

```
 (datetime.datetime(2013, 7, 27, 0, 0), 1073, Decimal('5399.73'), 4),
 (datetime.datetime(2013, 7, 27, 0, 0), 365, Decimal('5099.15'), 5),
 (datetime.datetime(2013, 7, 28, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 28, 0, 0), 957, Decimal('5099.66'), 2),
 (datetime.datetime(2013, 7, 28, 0, 0), 365, Decimal('4799.20'), 3),
 (datetime.datetime(2013, 7, 28, 0, 0), 403, Decimal('4419.66'), 4),
 (datetime.datetime(2013, 7, 28, 0, 0), 191, Decimal('4299.57'), 5)]
```

```
%%sql

SELECT * FROM (SELECT dpr.*,
  dense_rank() OVER (
    PARTITION BY order_date
    ORDER BY revenue DESC
  ) AS drnk
FROM daily_product_revenue AS dpr) q
WHERE drnk <= 5
ORDER BY order_date, revenue DESC
LIMIT 20
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
20 rows affected.
```

```
[(datetime.datetime(2013, 7, 25, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 25, 0, 0), 191, Decimal('5099.49'), 2),
 (datetime.datetime(2013, 7, 25, 0, 0), 957, Decimal('4499.70'), 3),
 (datetime.datetime(2013, 7, 25, 0, 0), 365, Decimal('3359.44'), 4),
 (datetime.datetime(2013, 7, 25, 0, 0), 1073, Decimal('2999.85'), 5),
 (datetime.datetime(2013, 7, 26, 0, 0), 1004, Decimal('10799.46'), 1),
 (datetime.datetime(2013, 7, 26, 0, 0), 365, Decimal('7978.67'), 2),
 (datetime.datetime(2013, 7, 26, 0, 0), 957, Decimal('6899.54'), 3),
 (datetime.datetime(2013, 7, 26, 0, 0), 191, Decimal('6799.32'), 4),
 (datetime.datetime(2013, 7, 26, 0, 0), 1014, Decimal('4798.08'), 5),
 (datetime.datetime(2013, 7, 27, 0, 0), 1004, Decimal('9599.52'), 1),
 (datetime.datetime(2013, 7, 27, 0, 0), 191, Decimal('5999.40'), 2),
 (datetime.datetime(2013, 7, 27, 0, 0), 957, Decimal('5699.62'), 3),
 (datetime.datetime(2013, 7, 27, 0, 0), 1073, Decimal('5399.73'), 4),
 (datetime.datetime(2013, 7, 27, 0, 0), 365, Decimal('5099.15'), 5),
 (datetime.datetime(2013, 7, 28, 0, 0), 1004, Decimal('5599.72'), 1),
 (datetime.datetime(2013, 7, 28, 0, 0), 957, Decimal('5099.66'), 2),
 (datetime.datetime(2013, 7, 28, 0, 0), 365, Decimal('4799.20'), 3),
 (datetime.datetime(2013, 7, 28, 0, 0), 403, Decimal('4419.66'), 4),
 (datetime.datetime(2013, 7, 28, 0, 0), 191, Decimal('4299.57'), 5)]
```

6.7.15 Exercises - Analytics Functions

Let us take care of the exercises related to analytics functions. We will be using HR database for the same.

- Get all the employees who is making more than average salary with in each department.
- Get cumulative salary for one of the department along with department name.
- Get top 3 paid employees with in each department by salary (use dense_rank)
- Get top 3 products sold in the month of 2014 January by revenue.

- Get top 3 products in each category sold in the month of 2014 January by revenue.

Prepare HR Database

Here are the steps to prepare HR database.

- Connect to HR DB using `psql` or SQL Workbench. Here is the sample `psql` command.

```
psql -h localhost \
    -p 5432 \
    -d itversity_hr_db \
    -U itversity_hr_user \
    -W
```

- Run scripts to create tables and load the data. You can also drop the tables if they already exists.

```
\i /data/hr_db/drop_tables_pg.sql
\i /data/hr_db/create_tables_pg.sql
\i /data/hr_db/load_tables_pg.sql
```

- Validate to ensure that data is available in the tables by running these queries.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

```
%sql SELECT * FROM employees LIMIT 10
```

```
%%sql

SELECT * FROM departments
ORDER BY manager_id NULLS LAST
LIMIT 10
```

Exercise 1

Get all the employees who is making more than average salary with in each department.

- Use HR database employees and department tables for this problem.
- Compute average salary expense for each department and get those employee details who are making more salary than average salary.
- Make sure average salary expense per department is rounded off to 2 decimals.
- Output should contain employee_id, department_name, salary and avg_salary_expense (derived field).
- Data should be sorted in ascending order by department_id and descending order by salary.

employee_id	department_name	salary	avg_salary_expense
201	Marketing	13000.00	9500.00
114	Purchasing	11000.00	4150.00
121	Shipping	8200.00	3475.56
120	Shipping	8000.00	3475.56

continues on next page

Table 6.1 – continued from previous page

employee_id	department_name	salary	avg_salary_expense
122	Shipping	7900.00	3475.56
123	Shipping	6500.00	3475.56
124	Shipping	5800.00	3475.56
184	Shipping	4200.00	3475.56
185	Shipping	4100.00	3475.56
192	Shipping	4000.00	3475.56
193	Shipping	3900.00	3475.56
188	Shipping	3800.00	3475.56
137	Shipping	3600.00	3475.56
189	Shipping	3600.00	3475.56
141	Shipping	3500.00	3475.56
103	IT	9000.00	5760.00
104	IT	6000.00	5760.00
145	Sales	14000.00	8955.88
146	Sales	13500.00	8955.88
147	Sales	12000.00	8955.88
168	Sales	11500.00	8955.88
148	Sales	11000.00	8955.88
174	Sales	11000.00	8955.88
149	Sales	10500.00	8955.88
162	Sales	10500.00	8955.88
156	Sales	10000.00	8955.88
150	Sales	10000.00	8955.88
169	Sales	10000.00	8955.88
170	Sales	9600.00	8955.88
163	Sales	9500.00	8955.88
151	Sales	9500.00	8955.88
157	Sales	9500.00	8955.88
158	Sales	9000.00	8955.88
152	Sales	9000.00	8955.88
100	Executive	24000.00	19333.33
108	Finance	12000.00	8600.00
109	Finance	9000.00	8600.00
205	Accounting	12000.00	10150.00

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_hr_user:hr_password@localhost:5432/itversity_
↪hr_db
```

Exercise 2

Get cumulative salary with in each department for Finance and IT department along with department name.

- Use HR database employees and department tables for this problem.
- Compute cumulative salary expense for **Finance** as well as **IT** departments with in respective departments.
- Make sure cumulative salary expense per department is rounded off to 2 decimals.
- Output should contain employee_id, department_name, salary and cum_salary_expense (derived field).
- Data should be sorted in ascending order by department_name and then salary.

employee_id	department_name	salary	cum_salary_expense
113	Finance	6900.00	6900.00
111	Finance	7700.00	14600.00
112	Finance	7800.00	22400.00
110	Finance	8200.00	30600.00
109	Finance	9000.00	39600.00
108	Finance	12000.00	51600.00
107	IT	4200.00	4200.00
106	IT	4800.00	9000.00
105	IT	4800.00	13800.00
104	IT	6000.00	19800.00
103	IT	9000.00	28800.00

Exercise 3

Get top 3 paid employees with in each department by salary (use dense_rank)

- Use HR database employees and department tables for this problem.
- Highest paid employee should be ranked first.
- Output should contain employee_id, department_id, department_name, salary and employee_rank (derived field).
- Data should be sorted in ascending order by department_id in ascending order and then salary in descending order.

employee_id	department_id	department_name	salary	employee_rank
200	10	Administration	4400.00	1
201	20	Marketing	13000.00	1
202	20	Marketing	6000.00	2
114	30	Purchasing	11000.00	1
115	30	Purchasing	3100.00	2
116	30	Purchasing	2900.00	3
203	40	Human Resources	6500.00	1
121	50	Shipping	8200.00	1
120	50	Shipping	8000.00	2
122	50	Shipping	7900.00	3
103	60	IT	9000.00	1
104	60	IT	6000.00	2
105	60	IT	4800.00	3
106	60	IT	4800.00	3
204	70	Public Relations	10000.00	1
145	80	Sales	14000.00	1
146	80	Sales	13500.00	2
147	80	Sales	12000.00	3
100	90	Executive	24000.00	1
101	90	Executive	17000.00	2
102	90	Executive	17000.00	2
108	100	Finance	12000.00	1
109	100	Finance	9000.00	2
110	100	Finance	8200.00	3
205	110	Accounting	12000.00	1
206	110	Accounting	8300.00	2

Exercise 4

Get top 3 products sold in the month of 2014 January by revenue.

- Use retail database tables such as orders, order_items and products.
- Consider only those orders which are either in **COMPLETE** or **CLOSED** status.
- Highest revenue generating product should come at top.
- Output should contain product_id, product_name, revenue, product_rank. **revenue** and **product_rank** are derived fields.
- Data should be sorted in descending order by revenue.

product_id	product_name	revenue	product_rank
1004	Field & Stream Sportsman 16 Gun Fire Safe	250787.46	1
365	Perfect Fitness Perfect Rip Deck	151474.75	2
957	Diamondback Women's Serene Classic Comfort Bi	148190.12	3

Exercise 5

Get top 3 products sold in the month of 2014 January under selected categories by revenue. The categories are **Cardio Equipment** and **Strength Training**.

- Use retail database tables such as orders, order_items, products as well as categories.
- Consider only those orders which are either in **COMPLETE** or **CLOSED** status.
- Highest revenue generating product should come at top.
- Output should contain category_id, category_name, product_id, product_name, revenue, product_rank. revenue and product_rank are derived fields.
- Data should be sorted in ascending order by category_id and descending order by revenue.

category_id	category_name	product_id	product_name	revenue	product_rank
9	Cardio Equipment	191	Nike Men's Free 5.0+ Running Shoe	132286.77	1
9	Cardio Equipment	172	Nike Women's Tempo Shorts	870.00	2
10	Strength Training	208	SOLE E35 Elliptical	1999.99	1
10	Strength Training	203	GoPro HERO3+ Black Edition Camera	1199.97	2
10	Strength Training	216	Yakima DoubleDown Ace Hitch Mount 4-Bike Rack	189.00	3

6.8 Query Performance Tuning

As part of this section we will go through basic performance tuning techniques with respect to queries.

- Preparing Database
- Interpreting Explain Plans
- Overview of Cost Based Optimizer
- Performance Tuning using Indexes
- Criteria for indexes
- Criteria for Partitioning
- Writing Queries – Partition Pruning
- Overview of Query Hints

6.8.1 Preparing Database

Let us prepare retail tables to come up with the solution for the problem statement.

- Ensure that we have required database and user for retail data. We might provide the database as part of our labs.

```
psql -U postgres -h localhost -p 5432 -W
```

```
CREATE DATABASE itversity_retail_db;
CREATE USER itversity_retail_user WITH ENCRYPTED PASSWORD 'retail_password';
GRANT ALL ON DATABASE itversity_retail_db TO itversity_retail_user;
```

- Create Tables using the script provided. You can either use `psql` or **SQL Alchemy**.

```
psql -U itversity_retail_user \
  -h localhost \
  -p 5432 \
  -d itversity_retail_db \
  -W

\i /data/retail_db/create_db_tables_pg.sql
```

- Data shall be loaded using the script provided.

```
\i /data/retail_db/load_db_tables_pg.sql
```

- Run queries to validate we have data in all the 6 tables.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%sql SELECT * FROM departments LIMIT 10
```

```
%sql SELECT * FROM categories LIMIT 10
```

```
%sql SELECT * FROM products LIMIT 10
```

```
%sql SELECT * FROM orders LIMIT 10
```

```
%sql SELECT * FROM order_items LIMIT 10
```

```
%sql SELECT * FROM customers LIMIT 10
```

6.8.2 Interpreting Explain Plans

Let us review the below explain plans and understand key terms which will help us in interpreting them.

- Seq Scan
- Index Scan
- Nested Loop

Here are the explain plans for different queries.

- Explain plan for query to get number of orders.

```
EXPLAIN
SELECT count(1) FROM orders;
```

```
                                   QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=1386.04..1386.05 rows=1 width=8)
   ->  Seq Scan on orders  (cost=0.00..1213.83 rows=68883 width=0)
(2 rows)
```

- Explain plan for query to get number of orders by date.

```
EXPLAIN
SELECT order_date, count(1) AS order_count
FROM orders
GROUP BY order_date;
```

```
                                   QUERY PLAN
-------------------------------------------------------------------
 HashAggregate  (cost=1558.24..1561.88 rows=364 width=16)
   Group Key: order_date
   ->  Seq Scan on orders  (cost=0.00..1213.83 rows=68883 width=8)
(3 rows)
```

- Explain plan for query to get order details for a given order id.

```
EXPLAIN
SELECT * FROM orders
WHERE order_id = 2;
```

```
                                       QUERY PLAN
---------------------------------------------------------------------------
 Index Scan using orders_pkey on orders  (cost=0.29..8.31 rows=1 width=26)
   Index Cond: (order_id = 2)
(2 rows)
```

- Explain plan for query to get order and order item details for a given order id.

```
EXPLAIN
SELECT o.*,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = 2;
```

```
                                          QUERY PLAN
----------------------------------------------------------------------------------
 Nested Loop  (cost=0.29..3427.82 rows=4 width=34)
   ->  Index Scan using orders_pkey on orders o  (cost=0.29..8.31 rows=1 width=26)
         Index Cond: (order_id = 2)
   ->  Seq Scan on order_items oi  (cost=0.00..3419.47 rows=4 width=12)
         Filter: (order_item_order_id = 2)
(5 rows)
```

Note: We should understand the order in which the query plans should be interpreted.

- Explain plan for a query with multiple joins

```
EXPLAIN
SELECT
    o.order_date,
    d.department_id,
    d.department_name,
    c.category_name,
    p.product_name,
    round(sum(oi.order_item_subtotal)::numeric, 2) AS revenue
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
    JOIN products p
        ON p.product_id = oi.order_item_product_id
    JOIN categories c
        ON c.category_id = p.product_category_id
    JOIN departments d
        ON d.department_id = c.category_department_id
GROUP BY
    o.order_date,
    d.department_id,
    d.department_name,
    c.category_id,
    c.category_name,
    p.product_id,
    p.product_name
ORDER BY o.order_date,
    revenue DESC;
```

```
                                                                   QUERY PLAN
-------------------------------------------------------------------------------------
↪------------------------------------------------
 Sort  (cost=76368.54..76799.03 rows=172198 width=211)
   Sort Key: o.order_date, (round((sum(oi.order_item_subtotal))::numeric, 2)) DESC
   ->  Finalize GroupAggregate  (cost=25958.31..43735.23 rows=172198 width=211)
         Group Key: o.order_date, d.department_id, c.category_id, p.product_id
         ->  Gather Merge  (cost=25958.31..39886.09 rows=101293 width=187)
               Workers Planned: 1
               ->  Partial GroupAggregate  (cost=24958.30..27490.62 rows=101293␣
↪width=187)
                     Group Key: o.order_date, d.department_id, c.category_id, p.
↪product_id
                     ->  Sort  (cost=24958.30..25211.53 rows=101293 width=187)
```

(continues on next page)

(continued from previous page)

```
                          Sort Key: o.order_date, d.department_id, c.category_id, p.
↪product_id
                          ->  Hash Join  (cost=2495.48..7188.21 rows=101293␣
↪width=187)
                                Hash Cond: (c.category_department_id = d.department_
↪id)
                                ->  Hash Join  (cost=2472.43..6897.32 rows=101293␣
↪width=79)
                                      Hash Cond: (p.product_category_id = c.category_
↪id)
                                      ->  Hash Join  (cost=2470.13..6609.69␣
↪rows=101293 width=63)
                                            Hash Cond: (oi.order_item_product_id = p.
↪product_id)
                                            ->  Hash Join  (cost=2411.87..6284.70␣
↪rows=101293 width=20)
                                                  Hash Cond: (oi.order_item_order_id␣
↪= o.order_id)
                                                  ->  Parallel Seq Scan on order_
↪items oi  (cost=0.00..2279.93 rows=101293 width=16)
                                                  ->  Hash  (cost=1213.83..1213.83␣
↪rows=68883 width=12)
                                                        ->  Seq Scan on orders o ␣
↪(cost=0.00..1213.83 rows=68883 width=12)
                                            ->  Hash  (cost=41.45..41.45 rows=1345␣
↪width=47)
                                                  ->  Seq Scan on products p ␣
↪(cost=0.00..41.45 rows=1345 width=47)
                                      ->  Hash  (cost=1.58..1.58 rows=58 width=20)
                                            ->  Seq Scan on categories c  (cost=0.00.
↪.1.58 rows=58 width=20)
                                ->  Hash  (cost=15.80..15.80 rows=580 width=112)
                                      ->  Seq Scan on departments d  (cost=0.00..15.
↪80 rows=580 width=112)
(27 rows)
```

6.8.3 Overview of Cost Based Optimizer

Let us get an overview of cost based optimizer.

- Databases use cost based optimizer to generate explain plans. In the earlier days, they used to use rule based optimizer.
- For cost based optimizer to generate optimal explain plan, we need to ensure statistics of our data in tables are collected at regular intervals.
- We can analyze tables to collect statistics. Typically DBAs schedule to collect statistics at regular intervals.
- In some cases we might have to compute statistics on the tables that are used in the query which we are trying to tune. The database user need to have permissions to compute statistics.
- Here are some of the basic statistics typically collected.
  - Approximate number of records at table level.
  - Approximate number of unique records at index level.

- When explain plans are generated, these statistics will be used by cost based optimizer to provide us with the most optimal plan for our query.

6.8.4 Performance Tuning using Indexes

Let us understand how we can improve the performance of the query by creating index on order_items.order_item_order_id.

- We have order level details in orders and item level details in order_items.
- When customer want to review their orders, they need details about order_items. In almost all the scenarios in order management system, we prefer to get both order as well as order_items details by passing order_id of pending or outstanding orders.
- Let us review the explain plan for the query with out index on order_items.order_item_order_id.

```
EXPLAIN
SELECT o.*,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = 2;
```

- Develop piece of code to randomly pass 2000 order ids and calculate time.

```
!pip install psycopg2
```

```
import psycopg2
```

```
%%time
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    cursor.execute(query, (1,))
    ctr += 1
cursor.close()
connection.close()
```

- Create index on order_items.order_item_order_id

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

CREATE INDEX order_items_order_id_idx
ON order_items(order_item_order_id);
```

- Run explain plan after creating index on order_items.order_item_order_id

```
EXPLAIN
SELECT o.*,
    oi.order_item_subtotal
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = 2;
```

```
                                              QUERY PLAN
-------------------------------------------------------------------------------------
↪----------------
 Nested Loop  (cost=0.71..16.81 rows=3 width=34)
   ->  Index Scan using orders_pkey on orders o  (cost=0.29..8.31 rows=1 width=26)
         Index Cond: (order_id = 2)
   ->  Index Scan using order_items_order_id_idx on order_items oi  (cost=0.42..8.47␣
↪rows=3 width=12)
         Index Cond: (order_item_order_id = 2)
(5 rows)
```

- Run the code again to see how much time, it get the results for 2000 random orders.

```
import psycopg2
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o JOIN order_items oi
    ON o.order_id = oi.order_item_order_id
WHERE o.order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    order_id = randrange(1, 68883)
    cursor.execute(query, (order_id,))
    ctr += 1
cursor.close()
connection.close()
```

Warning: Keep in mind that having indexes on tables can have negative impact on write operations.

6.8.5 Criteria for Indexing

Let us go through some of the criteria to create indexes on tables.

- Indexes are required to enforce constraints such as Primary Key, Unique etc. Indexes will be automatically created, when we define a column(s) Primary Key or Unique.
- Too many indexes on a given table, can slow down the performance of inserts, updates and deletes on that table. Hence, you need to make sure to strike right balance by creating indexes only when they are required.
- Thorough analysis need to be done about how the queries will hit the table from the application.
- We might have to create indexes on foreign key columns of the child table.
- When we have tables with multiple parents, we need to be due diligent about how the index should be created.
  - Shall we create 2 indexes?
  - Shall we create 1 index with both the columns pointing to 2 tables?
  - If we want to create 1 index with both the columns what should be the order?
- Here are some of the scenarios from the application perspective based upon which we can consider creating indexes.
  - Customer checking all his orders.
    - We need to get the data from orders using customer id and hence we need to add index on **orders.order_customer_id**.
  - Customer checking order details for a given order which include order_item_subtotal as well as product names.
    - We need to join **orders**, **order_items** as well as **products**.
    - **order_items** is child table for both **orders** and **products**.
    - We can create composite index on **order_items.order_item_order_id** and **order_items.order_item_product_id**.
  - Customer care executive to check **all the order details placed by customer using at least first 3 characters of customer's first name**.
    - We can consider creating index on **customers.customer_fname** using upper or lower. You can also consider adding **customer_id** to the index along with customer_fname.
    - Also to get all the order details for a given customer, we have to ensure that there is an index on **orders.order_customer_id**.

```
%load_ext sql
```

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%%sql

DROP INDEX order_items_order_id_idx
```

```
%%sql

SELECT min(customer_id), max(customer_id), count(1)
FROM customers
```

```
import psycopg2
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o
WHERE order_customer_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    customer_id = randrange(10950, 12435)
    cursor.execute(query, (customer_id,))
    ctr += 1
cursor.close()
connection.close()
```

```
%%sql

CREATE INDEX orders_customer_id_idx
ON orders(order_customer_id)
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o
WHERE order_customer_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
```

(continues on next page)

(continued from previous page)

```
    customer_id = randrange(10950, 12435)
    cursor.execute(query, (customer_id,))
    ctr += 1
cursor.close()
connection.close()
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
    JOIN products p
        ON p.product_id = oi.order_item_product_id
WHERE order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    order_id = randrange(1, 68883)
    cursor.execute(query, (order_id,))
    ctr += 1
cursor.close()
connection.close()
```

```
%%sql

CREATE INDEX order_items_oid_pid_idx
ON order_items(order_item_order_id, order_item_product_id);
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
```

(continues on next page)

(continued from previous page)

```
    JOIN products p
        ON p.product_id = oi.order_item_product_id
WHERE order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
        break
    order_id = randrange(1, 68883)
    cursor.execute(query, (order_id,))
    ctr += 1
cursor.close()
connection.close()
```

Note: As our products table only have handful of records there will not be significant difference in performance between the 2 approaches.

- Index on order_items.order_item_order_id
- Index on order_items.order_item_order_id, order_items.order_item_product_id

Howeever if you create index using product id as driving field then the performance will not be as good as above 2 approaches.

```
%%sql

DROP INDEX order_items_oid_pid_idx
```

```
%%sql

CREATE INDEX order_items_pid_oid_idx
ON order_items(order_item_product_id, order_item_order_id);
```

```
%%time

from random import randrange
connection = psycopg2.connect(
    host='localhost',
    port='5432',
    database='itversity_retail_db',
    user='itversity_retail_user',
    password='retail_password'
)
cursor = connection.cursor()
query = '''SELECT count(1)
FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_item_order_id
    JOIN products p
        ON p.product_id = oi.order_item_product_id
WHERE order_id = %s
'''
ctr = 0
while True:
    if ctr == 2000:
```

(continues on next page)

(continued from previous page)

```
        break
    order_id = randrange(1, 68883)
    cursor.execute(query, (order_id,))
    ctr += 1
cursor.close()
connection.close()
```

Note: Here are the indexes to tune the performance of comparing with at least first 3 characters of customer first name.

```
%%sql

DROP INDEX IF EXISTS orders_customer_id_idx
```

```
%%sql

DROP INDEX IF EXISTS customers_customer_fname_idx
```

- Explain plan for query with out indexes.

```
EXPLAIN
SELECT *
FROM orders o JOIN customers c
    ON o.order_customer_id = c.customer_id
WHERE upper(c.customer_fname) = upper('mar');
```

```
                               QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (cost=42.38..1437.09 rows=40 width=99)
   Hash Cond: (o.order_customer_id = c.customer_id)
   ->  Seq Scan on orders o  (cost=0.00..1213.83 rows=68883 width=26)
   ->  Hash  (cost=42.29..42.29 rows=7 width=73)
         ->  Seq Scan on customers c  (cost=0.00..42.29 rows=7 width=73)
               Filter: (upper((customer_fname)::text) = 'MAR'::text)
(6 rows)
```

```
%%sql

CREATE INDEX customers_customer_fname_idx
ON customers(upper(customer_fname))
```

```
%%sql

CREATE INDEX orders_customer_id_idx
ON orders(order_customer_id)
```

- Explain plan for query with indexes. Check the cost, it is significantly low when compared to the plan generated for the same query with out indexes.

```
EXPLAIN
SELECT *
FROM orders o JOIN customers c
```

(continues on next page)

(continued from previous page)

```
    ON o.order_customer_id = c.customer_id
WHERE upper(c.customer_fname) = upper('mar');
```

```
                                                QUERY PLAN
-------------------------------------------------------------------------------------
↪-----------
 Nested Loop  (cost=8.67..204.43 rows=40 width=99)
   ->  Bitmap Heap Scan on customers c  (cost=4.33..18.58 rows=7 width=73)
         Recheck Cond: (upper((customer_fname)::text) = 'MAR'::text)
         ->  Bitmap Index Scan on customers_customer_fname_idx  (cost=0.00..4.33␣
↪rows=7 width=0)
               Index Cond: (upper((customer_fname)::text) = 'MAR'::text)
   ->  Bitmap Heap Scan on orders o  (cost=4.34..26.49 rows=6 width=26)
         Recheck Cond: (order_customer_id = c.customer_id)
         ->  Bitmap Index Scan on orders_customer_id_idx  (cost=0.00..4.34 rows=6␣
↪width=0)
               Index Cond: (order_customer_id = c.customer_id)
(9 rows)
```

6.8.6 Criteria for Partitioning

Let us understand how we can leverage partitioning to fine tune the performance.

- Partitioning is another key strategy to boost the performance of the queries.
- It is extensively used as key performance tuning strategy as part of tables created to support reporting requirements.
- Even in transactional systems, we can leverage partitioning as one of the performance tuning technique while dealing with large tables.
- For application log tables, we might want to discard all the irrelevant data after specific time period. If partitioning is used, we can detach and/or drop the paritions quickly.
- Over a period of time most of the orders will be in **CLOSED** status. We can partition table using list parititioning to ensure that all the **CLOSED** orders are moved to another partition. It can improve the performance for the activity related to active orders.
- In case of reporting databases, we might partition the transaction tables at daily level so that we can easily filter and process data to pre-aggregate and store in the reporting data marts.
- Most of the tables in ODS or Data Lake will be timestamped and partitioned at daily or monthly level so that we can remove or archive old partitions easily

6.8.7 Writing Queries – Partition Pruning

Let us understand how to write queries by leveraging partitioing.

- Make sure to include a condition on partitioned column.
- Equal condition will yield better results.
- Queries with condition on partition key will result in partition pruning. The data from the other partitions will be fully ignored.
- As partition pruning will result in lesser I/O, the overall performance of such queries will improve drastically.

6.8.8 Overview of Query Hints

Let us get an overview of query hints.

- We can specify hint using /*+ HINT */ as part of the query.
- Make sure there are no typos in the hint.
- If there are typos or there no indexes specified as part of hint, they will be ignored.
- In case of complex queries, CBO might use incorrect index or inappropriate join.
- As an expert if we are sure that, the query should be using a particular index or right join, then we can force the optimizer to choose such index or join type leveraging hint.

6.8.9 Exercises - Tuning Queries

As part of this exercise, you need to prepare data set, go through the explain plan and come up with right indexes to tune the performance.

- As of now customer email id in customers table contain same value (**XXXXXXXXX**).
- Let us update customer_email_id.
  - Use initial (first character) of customer_fname
  - Use full string of customer_lname
  - Use row_number by grouping or partitioning the data by first character of customer_fname and full customer_lname then sort it by customer_id.
  - Make sure row_number is at least 3 digits, if not pad with 0 and concatenate to email id. Here are the examples
  - Also make sure email ids are in upper case. |customer_id|customer_fname|customer_lname|rank|customer_email| |————|—————–|—————–|—-|——————| |11591|Ann|Alexander|1|AALEXANDER001@SOME.COM| |12031|Ashley|Benitez|1|ABENITEZ001@SOME.COM| |11298|Anthony|Best|1|ABEST001@SOME.COM| |11304|Alexander|Campbell|1|ACAMPBELL001@SOME.COM| |11956|Alan|Campos|1|ACAMPOS001@SOME.COM| |12075|Aaron|Carr|1|ACARR001@SOME.COM| |12416|Aaron|Cline|1|ACLINE001@SOME.COM| |10967|Alexander|Cunningham|1|ACUNNINGHAM001@SOME.COM| |12216|Ann|Deleon|1|ADELEON001@SOME.COM |11192|Andrew|Dickson|1|ADICKSON001@SOME.COM|
- Let us assume that customer care will try to search for customer details using at least first 4 characters.
- Generate explain plan for the query.
- Create unique index on customer_email.
- Generate explain plan again and review the differences.

```
%env DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
env: DATABASE_URL=postgresql://itversity_retail_user:retail_password@localhost:5432/
↪itversity_retail_db
```

```
%load_ext sql
```

```
%%sql

SELECT q.*,
    upper(concat(substring(customer_fname, 1, 1), customer_lname, lpad(rnk::varchar,␣
↪3, '0'), '@SOME.COM')) AS customer_email
FROM (
    SELECT customer_id,
        customer_fname,
        customer_lname,
        rank() OVER (
            PARTITION BY substring(customer_fname, 1, 1), customer_lname
            ORDER BY customer_id
        ) AS rnk
    FROM customers
) q
ORDER BY customer_email
LIMIT 10
```

```
 * postgresql://itversity_retail_user:***@localhost:5432/itversity_retail_db
10 rows affected.
```

```
[(11591, 'Ann', 'Alexander', 1, 'AALEXANDER001@SOME.COM'),
 (12031, 'Ashley', 'Benitez', 1, 'ABENITEZ001@SOME.COM'),
 (11298, 'Anthony', 'Best', 1, 'ABEST001@SOME.COM'),
 (11304, 'Alexander', 'Campbell', 1, 'ACAMPBELL001@SOME.COM'),
 (11956, 'Alan', 'Campos', 1, 'ACAMPOS001@SOME.COM'),
 (12075, 'Aaron', 'Carr', 1, 'ACARR001@SOME.COM'),
 (12416, 'Aaron', 'Cline', 1, 'ACLINE001@SOME.COM'),
 (10967, 'Alexander', 'Cunningham', 1, 'ACUNNINGHAM001@SOME.COM'),
 (12216, 'Ann', 'Deleon', 1, 'ADELEON001@SOME.COM'),
 (11192, 'Andrew', 'Dickson', 1, 'ADICKSON001@SOME.COM')]
```