dataHandoff

January 13, 2022

1 Import Dependencies

[180]:
```python
import pandas as pd
from pathlib import Path
import psycopg2
from psycopg2 import OperationalError
from dotenv import load_dotenv
import os
from IPython.display import IFrame
```

Load Postegres Password from `.env` File
To download PostgreSQL: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

Postgres password saved in file `.env`:
`db_password = 'THE_POSTGRES_PASSWORD_HERE'`

[183]:
```python
load_dotenv()
db_password = os.getenv("db_password")
```

[184]:
```python
# db_password
```

[185]:
```python
def create_connection(db_name, db_user, db_password, db_host, db_port):
    connection = None
    try:
        connection = psycopg2.connect(
            database=db_name,
            user=db_user,
            password=db_password,
            host=db_host,
            port=db_port,
        )
        # print("Connection to PostgreSQL DB successful")
    except OperationalError as e:
        print(f"The error '{e}' occurred")
    return connection
```

[186]:
```
connection = create_connection(
    "postgres", "postgres", db_password, "127.0.0.1", "5432"
)
```

[187]:
```
def create_database(connection, query):
    connection.autocommit = True
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        print("Query executed successfully")
    except OperationalError as e:
        print(f"The error '{e}' occurred")
```

[188]:
```
create_database_query = "CREATE DATABASE validate_health_db" # make sure to use␣
 ↪lower case
```

[189]:
```
try:
    create_database(connection, create_database_query)
except:
    print('This database already exists.')
```

```
This database already exists.
```

[190]:
```
def connection():
    return create_connection(
    "validate_health_db", "postgres", db_password, "127.0.0.1", "5432"
)
connection = connection()
```

[191]:
```
def execute_query(connection, query):
    connection.autocommit = True
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        print("Query executed successfully")
    except OperationalError as e:
        print(f"The error '{e}' occurred")
```

Create patient_visit Table

[192]:
```
create_patient_visit_table = """
CREATE TABLE IF NOT EXISTS patient_visit (
    "social_security_num" INT8,
    "patient_first_name" CHAR(80),
    "patient_last_name" CHAR(80),
    "patient_age" INT8,
    "visit_date" DATE,
    "procedure_code" CHAR(5),
    "doctor_name" CHAR(200),
    "charge_for_visit" MONEY
)
"""
```

[193]:
```
# try:
execute_query(connection, create_patient_visit_table)
# except:
    # print('This table already exists.')
```

```
Query executed successfully
```

Import .csv Data Into course_details Table

[194]:
```
csv_dataset = Path(str(Path.cwd()) + '/' + 'data_handoff_data.csv')
print(str(csv_dataset))
```

```
C:\Users\mchar\Downloads\Guild\guild_proj\data_handoff_data.csv
```

[195]:
```
csv_dataset = """
COPY patient_visit
FROM '{}'
DELIMITER ','
CSV HEADER;
""".format(str(csv_dataset))
```

[196]:
```
execute_query(connection, csv_dataset)
# If an error gets thrown change the permissions to 'Everyone' ->'Full control'␣
 ↪in properties of the file (if using Windows).
# Use chmod if using macOS or Linux distro.
```

```
Query executed successfully
```

[197]:
```
def execute_read_query(connection, query):
    result = None
    incrementor = 0
    try:
        cursor = connection.cursor()
        cursor.execute(query)
        result =cursor.fetchall()

        if len(result) < 10:
            for x in result:
                print(x)
        else:
            while incrementor < 15:
                print(result[incrementor])
                incrementor += 1
    except OperationalError as e:
        print(f"The error '{e}' occurred")
```

[198]:
```
# next_query = """
# DROP DATABASE IF EXISTS  validate_health_db;
# """

# execute_read_query(connection, next_query)
```

[199]:
```
# next_query = """
# DROP TABLE IF EXISTS patient_visit;
# """

# execute_query(connection, next_query)
```

[ ]:
```
# next_query = """
# DROP TABLE IF EXISTS patient_spending;
# """

# execute_query(connection, next_query)
```

[ ]:
```
# Generating the patient_spending Tabel
```

[287]:
```
create_patient_spending_table = """
CREATE TABLE patient_spending AS
SELECT
         ntile(3) OVER (ORDER BY social_security_num) || social_security_num AS␣
 ↪deident_patient_id,
     visit_month_yyyymm,
     charge_per_month
FROM (
        SELECT
        RIGHT(CAST("social_security_num" AS CHAR(7)), 6) AS social_security_num,
        to_char(visit_date,'YYYYMM') as visit_month_yyyymm,
        SUM("charge_for_visit") AS "charge_per_month"
        FROM patient_visit
        GROUP BY social_security_num, visit_month_yyyymm
        ) AS temp_table
ORDER BY  visit_month_yyyymm ASC;
"""

try:
    execute_query(connection, create_patient_spending_table)
    raise Exception
except Exception:
    next_query = """
    DROP TABLE IF EXISTS patient_spending;
    """
    execute_query(connection, next_query)
    execute_query(connection, create_patient_spending_table)
```

```
Query executed successfully
Query executed successfully
```

[288]:
```
check_patient_spending_table = """
SELECT *
FROM patient_spending;
"""
execute_read_query(connection, check_patient_spending_table)
```

```
('2513519', '202104', '$23.00')
('2513519', '202104', '$23.00')
('3513520', '202104', '$75.00')
('1513518', '202104', '$24.00')
('3513520', '202104', '$75.00')
('1513518', '202104', '$63.00')
('3513520', '202104', '$72.00')
('3513520', '202104', '$63.00')
('3513520', '202104', '$43.00')
('2513519', '202104', '$23.00')
('2513519', '202104', '$72.00')
('2513519', '202105', '$34.00')
('3513520', '202105', '$63.00')
('3513520', '202105', '$24.00')
('1513518', '202105', '$23.00')
```

[ ]:
```

```

[71]:
```
connection.close()
```

[ ]:
```

```

2. Under your methodology of generating field `deident_patient_id`, what's the maximum number of patient identifiers that could be generated if the field has to be an integer data type? What if the data type is char(7)? How did you come up with that number?

This is a question having to do with the rule of products in combinatorics. If the `deident_patient_id` must be an `int(7)` data type we know that the first digit "must be one of 3 values (1, 2, 3)", then the next 6 digits must be one of 10 values (0,1,2,3,4,5,6,7,8,9). Therefore we have:

[323]:
```
maximum_number_of_patient_identifiers = 3 * 10 * 10 * 10 * 10 * 10 * 10
maximum_number_of_patient_identifiers
```

[323]:
```
3000000
```

or equivalently:

[324]:
```
set_of_first_digit_outcomes = 3 ** 1
set_of_remaining_digit_outcomes = 10 ** 6
```

```
maximum_number_of_patient_identifiers = set_of_first_digit_outcomes *␣
↪set_of_remaining_digit_outcomes
maximum_number_of_patient_identifiers
```

[324]: 3000000

If the data type is type char(7), it depends on by the database character set according the the PostgreSQL 9.5 Documentation. UTF=8 seems to be the default database character set and [ASCII] is the the most common subset of UTF-8. Most sources cite that ASCII contains 128 chars ([0,127]), MS's documentation for T-SQL and char datatypes corresponds with this., the same rule of products applies then. Therefore, we have:

[325]:
```
set_of_first_digit_outcomes = 3 ** 1
set_of_remaining_digit_outcomes = 128 ** 6
maximum_number_of_patient_identifiers = set_of_first_digit_outcomes *␣
↪set_of_remaining_digit_outcomes
maximum_number_of_patient_identifiers
```

[325]: 13194139533312

*Note: the actual figure can be lower than this, because some SQL databases (depending on which version of SQL we are using) do now allow the use of certain characters.

[ ]:
```

```

3. Next we need to take the name and email address from table patient_contact_info and combine it with the doctor name from table patient_visit, so that we can generate table announce_doctor. This is needed so that we can send appropriate emails to the patients. However, since the two tables come from completely different sources, it's quite possible that the patient names don't match up exactly. How would you go about producing the most accurate merge possible?

[ ]:
```
# create_patient_contact_info_table = """
# CREATE TABLE patient_spending AS
# SELECT
#     patient_full_name,
#     email_address,
#     doctor_name
#     FROM patient_contact_info
#     INNER JOIN patient_visit
#      --Assuming that there are no first or last names less than 3 chars. Also␣
↪may beed to be seperate select &/or use Like & Having.
#           ON patient_visit(LEFT(patient_contact_info."patient_first_name"), 3)␣
↪|| RIGHT(patient_contact_info."patient_last_name"), 3)).patient_contact_info␣
↪=
#      patient_visit(LEFT(patient_visit."patient_full_name"), 3) ||␣
↪RIGHT(patient_visit."patient_full_name"), 3))
#  """
```