PostgreSQL

In [1]:

```
import os
import sys
import time
import psycopg2
import numpy as np
import pandas as pd
import matplotlib
from setvis.membership import *
from utils import (generate_data)
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from IPython.display import clear_output
```

In [2]:

```
def format_row(row):
    if row >= 1000000:
        return f"{row / 1000000}m"
    elif row >= 1000:
        return f"{row / 1000}k"
    else:
        return f"{row}"
# constants
PM = 'planned missing'
GM = 'general missing'
SET = 'sets'
PATTERNS = [SET, GM, PM]
SETVIS = 'setvis'
GM_ROW = 10000
GM_COL = 10

########################## 100M case #######
# 1e3 rows with 10 cols                    #
# takes 1.6 seconds, size of df max 0.38MB #
# 100M will take 100k times more           #
# 1.6 * 1e5 = 1.6 * 1e5 / 60 / 60          #
# 44.44 hrs                                #
# size will be 0.38 * 1e5                  #
# 38000MB = 38GB in memory & ~ disk
# 30M rows died after ~6hrs
# 15M rows took
############################################
# so 50M rows should be about 14GB
ROWS = [int(15e6)]
COLS = [GM_COL]

# sets  : 10000      700
# Time to generate & populate data: 4.47
# 0.03 secs
# general missing   : 10000      700
# row is too big: size 11232, maximum size 8160
```

In [3]:

```
def get_connection():
    conn = psycopg2.connect(
        host="localhost",
        port="5432",
        user="postgres",
        password="postgres",
        dbname="postgres"
    )
    return conn
```

In [4]:

```
# populate the instance with a table called `setvis`
# Connect to the database
def populate_psql_with_df(df, conn):
    cursor = conn.cursor()

    table_name = 'setvis'

    # drop the table if it already exists
    cursor.execute(f"DROP TABLE IF EXISTS {table_name}")
    conn.commit()

    # Get the column names
    columns = df.columns.tolist()
    # escape @ in column names
    columns = [c.replace('@', '') for c in columns]
    # Create a string with the column names and types
    columns_str = ','.join([f'{col} CHAR(50)' for col in columns])

    # Create the table in the database
    cursor.execute(f'CREATE TABLE {table_name} (key SERIAL PRIMARY KE

    # Insert the data from the dataframe into the table
    for i, row in df.iterrows():
        cursor.execute(f"INSERT INTO {table_name} ({','.join(columns)
        row.tolist())

    # Commit the changes
    conn.commit()
    cursor.close()
```

In [5]:

```
# Do setvis eval
def postgresql_intersections(conn):
    data = Membership.from_postgres(
        conn,
        "setvis",
        "key",
    )
    return data
#     print(m_pg.intersections().head())
```

In [6]:

```
rams = []
times = []

def evaluate(df, row, col, pattern):
    """Evaluates the performance of the "Membership.from_postgres" fu
    for a given PostgreSQL relation populated using rows and columns.
    """
    # 1. capture time
    t = time.time()
    data = postgresql_intersections(conn)
    t = time.time() - t
    times.append((row, col, t, pattern))
    print(f'{t:.2f} secs')
    # 2. capture memory
    m = (sys.getsizeof(data._intersection_id_to_columns) +
         sys.getsizeof(data._intersection_id_to_records)
        )

    rams.append((row, col, m, pattern))
```

Main

In [7]:

```
# populate & read
conn = get_connection()
start_time = time.time()
for c in COLS:
    for r in ROWS:
        for p in PATTERNS:
            num_int = None
            if p == GM:
                num_int = c * 2 - 1

            print(p, "\t:", r, "\t", c)
            t = time.time()
            df = generate_data(p, r, c, num_int)
            print(f"Size of df: {sys.getsizeof(df)/1024/1024:.3f}MB"
            print(f"Time to generate data: {time.time()-t:.2f} secs"
            t = time.time()
            populate_psql_with_df(df, conn)
            print(f"Time to populate psql: {time.time()-t:.2f} secs"
            evaluate(df, r, c, p)

# Close the connection
conn.close()
# clear_output()
print(f"Done. Total time ({time.time() - start_time:.2f}s). Runs:",
      len(times))
```

```
sets    : 15000000       10
Size of df: 829.697MB
Time to generate data: 5.48 secs
Time to populate psql: 2340.47 secs

/home/layik/code/setvis/setvis/membership.py:870: UserWarning: pand
as only supports SQLAlchemy connectable (engine/connection) or data
base string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects
are not tested. Please consider using SQLAlchemy.
  intersection_id_to_columns = pd.read_sql_query(
/home/layik/code/setvis/setvis/membership.py:888: UserWarning: pand
as only supports SQLAlchemy connectable (engine/connection) or data
base string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects
are not tested. Please consider using SQLAlchemy.
  intersection_id_to_records = pd.read_sql_query(

22.63 secs
general missing         : 15000000       10
Size of df: 5722.046MB
Time to generate data: 131.81 secs
Time to populate psql: 3285.33 secs

/home/layik/code/setvis/setvis/membership.py:870: UserWarning: pand
as only supports SQLAlchemy connectable (engine/connection) or data
base string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects
are not tested. Please consider using SQLAlchemy.
  intersection_id_to_columns = pd.read_sql_query(
/home/layik/code/setvis/setvis/membership.py:888: UserWarning: pand
as only supports SQLAlchemy connectable (engine/connection) or data
base string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects
are not tested. Please consider using SQLAlchemy.
  intersection_id_to_records = pd.read_sql_query(

37.86 secs
```

```
planned missing         : 15000000       10
Size of df: 5722.046MB
Time to generate data: 187.43 secs
Time to populate psql: 3296.49 secs
```

```
/home/layik/code/setvis/setvis/membership.py:870: UserWarning: pand
as only supports SQLAlchemy connectable (engine/connection) or data
base string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects
are not tested. Please consider using SQLAlchemy.
  intersection_id_to_columns = pd.read_sql_query(
/home/layik/code/setvis/setvis/membership.py:888: UserWarning: pand
as only supports SQLAlchemy connectable (engine/connection) or data
base string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects
are not tested. Please consider using SQLAlchemy.
  intersection_id_to_records = pd.read_sql_query(
```

```
34.41 secs
Done. Total time (9341.91s). Runs: 3
```

In [ ]:

```
# cleanup psql
conn = get_connection()
cursor = conn.cursor()

table_name = 'setvis'
# drop the table if it already exists
cursor.execute(f"DROP TABLE IF EXISTS {table_name}")
conn.commit()
conn.close()
```

In [8]:

```
times_df = pd.DataFrame([
    {
        "colxrow": r*c,
        "seconds": t,
        "pattern": p
    } for r, c, t, p in times
])
mem_df = pd.DataFrame([
    {
        "colxrow": r*c,
        "memory": round(m/1024/1024,5),
        "pattern": p
    } for r, c, m, p in rams
])
```

In [ ]:

```
# using seaborn
import seaborn as sns

def plot_df(d, time = True, x= 'colxrow', y = 'seconds', line = True
    palette = {PM: "green", GM: "blue", SET: "red"}
    # switching between time and memory
    LT = 4
    ylabel = "Time (seconds)"
    xlabel = "Number of cells"
    if not time:
        ylabel = "Memory (MB)"
        LT = 1

    plt.figure(figsize=(4, 4))
    # format axis function
    def format_ax(ax):
        xticks = np.arange(d[x].min(), d[x].max(), d[x].max()/10)
        ax.set_xticks(xticks)
        xticks = [format_row(x) for x in xticks]
        ax.set_xticklabels(xticks, rotation=30)
        ax.set_xlabel(xlabel, fontsize=12)
        ax.set_ylabel(ylabel, fontsize=12)

    g = sns.relplot(d, x=x, y=y, hue='pattern', palette=palette,
                    kind='line', height = 3, aspect = 1)
    for ax in g.axes.flat:
        format_ax(ax)
        g.add_legend()

    plt.tight_layout()
    plt.show()
```

In [9]:

```
# write them to csv file
file = '-'.join(map(str, COLS)) + 'X' + '-'.join(map(str, ROWS))
times_df.to_csv(file + "-psql-times.csv", index=False)
mem_df.to_csv(file + "-psql-mems.csv", index=False)
```

In [ ]:

```
# times_sql = pd.read_csv(file + "-times.csv")
# mem_sql = pd.read_csv(file + "-mems.csv")
```

In [ ]:

```
# plot_df(times_sql)
# plot_df(mem_sql, time = False, y = 'memory')
```