Databases - Gruppe F

Christoffer, Jonathan, Mark, Oscar

DB Assignment 01 - Report

Introduction

In our first project for the database course, we were assigned the task of constructing a database in Microsoft SQL Server, using five provided datasets. The project involved several key steps:

1. Data Preparation: We began by analyzing and cleaning the datasets to ensure their quality and relevance for our objectives.

2. Question Formulation: We identified ten questions to serve as the foundation for our data exploration and analysis.

3. Data Ingestion: Utilizing Python, we developed scripts to efficiently import the cleaned data into the database.

4. Query Development: With the data in place, we crafted SQL queries to systematically address the formulated questions, extracting meaningful insights.

5. Optimization and Maintenance: The final phase focused on optimizing query performance and establishing maintenance practices to ensure the database's long-term efficiency and reliability.

This approach allowed us to not only apply theoretical knowledge in a practical setting but also to hone our skills in database design, data manipulation, and SQL programming.

Useful links

Github:
https://github.com/Christoffer-Nielsen-Cph/DB_Soft

Bacpac (Can be downloaded and imported through SSMS):
https://github.com/Christoffer-Nielsen-Cph/DB_Soft/blob/master/DB_Soft/SQL_Scripts/DB_Soft.bacpac

Christoffer, Jonathan, Mark, Oscar

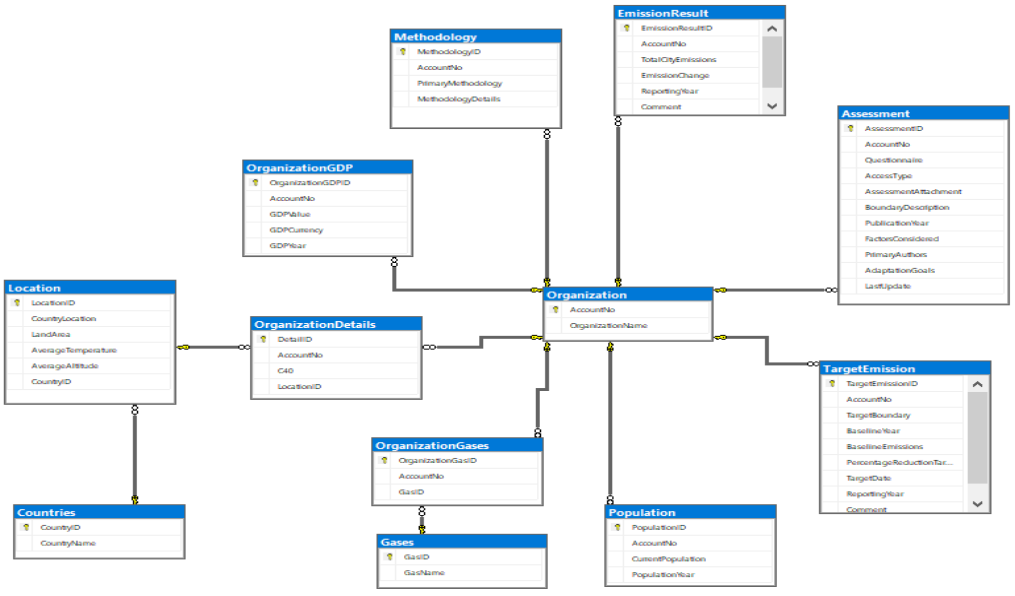
Data exploration

In the initial phase of our data exploration, we faced a significant challenge upon receiving the datasets in the form of five CSV files. These files presented challenges due to inconsistent column names across the datasets, which contained values that were essentially the same. Such inconsistencies threatened not only the process of importing data but also the eventual consistency and reliability of the data within our intended database system.

However, by thoroughly examining the data we quickly realized that there were certain patterns. By recognizing these patterns, we determined a more nuanced approach was necessary. We decided to divide the original five CSV files into twelve distinct files. This decision was made to ensure that each file directly corresponded to an individual table in our database.

This restructuring allowed us to align data fields with our database tables more accurately, which created a smoother and more reliable importing of the data. Through this reorganization, we laid a strong foundation for our database, ensuring that it would be well-equipped for the assignment, and for future projects.

Database structure



Our database schema, as illustrated in the diagram above, shows the comprehensive structure and design of our database. The central table within our schema is “Organization”, which serves as the core for all our relational connections. Our approach to structuring the database was methodical, ensuring that we thoroughly applied the principles of normalization to reduce data redundancy and improve data integrity.

We have normalized our database to meet at least the third normal form (3NF). This is demonstrated by the clear division of responsibilities among our tables, with each table dedicated to a specific type of data relevant to its particular context.

Data cleaning & ingesting

To clean the data and ingest it into the database, we used Python with Pandas and Pyodbc. An example of this is how we loaded the data from '2023_Cities_Climate_Risk_and_Vulnerability_Assessments_20240207.csv' into the database table called 'Assessments' (*See appendix 1*):

First, we loaded the CSV as a dataframe and then converted the column types where needed, such as “Last Update,” which were converted to a datetime object. Since the database does not support NaN values, we replaced them with empty strings or 0 to ensure successful data insertion.

Lastly, we used Pyodbc to insert the data into the database using insert statements while also remembering the order of the different tables that rely on foreign keys to maintain data integrity and preserve the connections between tables.

Database maintenance

For database maintenance, we have developed a stored procedure which is scheduled for execution on a weekly basis using SQL Server Agent. This procedure is designed to defragment the entire database, thereby ensuring optimized utilization of storage resources, quicker query execution and enhanced index performance (*See appendix 2*).

Additionally, multiple backup copies of the database have been established as a precautionary measure. This strategy is implemented to safeguard against potential data loss, ensuring data integrity and continuity in the event of unforeseen database compromise.

Working with the database

In order for us to work with the database we formulated 10 questions to help us explore and understand our data.

1. **List all organizations with their respective country names.**

The SQL procedure [dbo].[SP_OrganizationWithCountries] is designed to retrieve a unique list of organizations along with the names of the countries they are associated with. When executed, this procedure performs a query that involves multiple joins. (*This is the practice we used for the rest of the questions)*

```
CREATE PROCEDURE [dbo].[SP_OrganizationWithCountries]
AS
BEGIN

SELECT DISTINCT O.OrganizationName, C.CountryName
FROM Organization O
JOIN OrganizationDetails OD ON O.AccountNo = OD.AccountNo
JOIN Location L ON OD.LocationID = L.LocationID
JOIN Countries C ON L.CountryID = C.CountryID;

END
```

2. **Get an organization's total emission for a specific year.**

```
CREATE PROCEDURE [dbo].[SP_OrgEmissionForYear]
@year nvarchar (20),
@organizationName nvarchar(50)
AS
BEGIN

SELECT
    o.OrganizationName,
    er.ReportingYear,
    er.TotalCityEmissions
FROM
    dbo.EmissionResult er
INNER JOIN
    dbo.Organization o ON er.AccountNo = o.AccountNo
WHERE
    er.ReportingYear = @year
    AND o.OrganizationName = @organizationName

END
```

3. Find organizations that use a specific gas.

```
CREATE PROCEDURE [dbo].[SP_OrgSpecificGas]
@gas nvarchar (20)
AS
BEGIN

SELECT O.OrganizationName
FROM Organization O
JOIN OrganizationGases OG ON O.AccountNo = OG.AccountNo
JOIN Gases G ON OG.GasID = G.GasID
WHERE G.GasName = @gas;

END
```

4. List the population details of organizations for a specific year.

```
CREATE PROCEDURE [dbo].[SP_PopulationDetailsYear]
@year nvarchar (20)
AS
BEGIN

SELECT O.OrganizationName, P.CurrentPopulation
FROM Organization O
JOIN Population P ON O.AccountNo = P.AccountNo
WHERE P.PopulationYear = @year;

END
```

5. Show organizations with their GDP in a specific year.

```
CREATE PROCEDURE [dbo].[SP_OrganizationWithGDP]
@year nvarchar (20)
AS
BEGIN

SELECT O.OrganizationName, OG.GDPValue, OG.GDPCurrency
FROM Organization O
JOIN OrganizationGDP OG ON O.AccountNo = OG.AccountNo
WHERE OG.GDPYear = @year;

END
```

6. Display the emission reduction targets for each organization.

```
CREATE PROCEDURE [dbo].[SP_OrganizationsWithEmissions]
AS
BEGIN

SELECT O.OrganizationName, TE.PercentageReductionTarget,
TE.TargetDate
FROM Organization O
JOIN TargetEmission TE ON O.AccountNo = TE.AccountNo;

END
```

7. List all organizations that are a C40 city.

```
CREATE PROCEDURE [dbo].[SP_OrganizationC40]
AS
BEGIN

SELECT DISTINCT O.OrganizationName
FROM Organization O
JOIN OrganizationDetails OD ON O.AccountNo = OD.AccountNo
WHERE  OD.C40 = 1;

END
```

8. List all gases used by a specific organization.

```
CREATE PROCEDURE [dbo].[SP_OrganizationGases]
@organizationName nvarchar(50)
AS

BEGIN
SELECT G.GasName
FROM Gases G
JOIN OrganizationGases OG ON G.GasID = OG.GasID
JOIN Organization O ON OG.AccountNo = O.AccountNo
WHERE O.OrganizationName = @organizationName;

END
```

9. Get Details of the methodology used by organizations.

```
CREATE PROCEDURE [dbo].[SP_GetOrganizationMethodologies]
    @organizationName NVARCHAR(100)
AS
BEGIN
    SELECT O.OrganizationName,
M.PrimaryMethodology,M.MethodologyDetails
    FROM Organization O
    JOIN Methodology M ON O.AccountNo = M.AccountNo
    WHERE O.OrganizationName = @organizationName;
END
```

10. List countries with their total land area and average temperature.

```
CREATE PROCEDURE [dbo].[SP_GetCountriesLandAreaAndTemperature]
AS
BEGIN
    SELECT C.CountryName, SUM(L.LandArea) AS TotalLandArea,
AVG(L.AverageTemperature) AS AverageTemperature
    FROM Countries C
    JOIN Location L ON C.CountryID = L.CountryID
    GROUP BY C.CountryName;
END
```

We have also created a backend system using MVC C# ASP.NET Core Web App - but we created our controller as a controller API given the timeframe we had. In the future we might turn it into a fully developed web application. Below is an example of how our backend code looks like. This also means that we can call out endpoints in Postman and get the results out in Json.

```
namespace DB_Soft.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class OrganizationController : ControllerBase
    {
        private readonly OrganizationService _organizationService;
        private readonly IConfiguration _configuration;

        public OrganizationController(OrganizationService organizationService,  IConfiguration configuration)
        {
            _organizationService = organizationService;
            _configuration = configuration;
        }


        [HttpGet("Emissions")]
        public JsonResult GetEmissions(string reportingYear, string   organizationName)
        {
            if (string.IsNullOrWhiteSpace(organizationName))
            {
                return new JsonResult(new { error = "Organization name is required." }) { StatusCode = 400 };
            }

            DataTable table = new DataTable();
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");

            using (SqlConnection sqlConnection = new SqlConnection(sqlDataSource))
            {
                sqlConnection.Open();
                using (SqlCommand sqlCommand = new SqlCommand("SP_OrgEmissionForYear", sqlConnection))
                {
                    sqlCommand.CommandType = CommandType.StoredProcedure;
                    sqlCommand.Parameters.AddWithValue("@year", reportingYear);
                    sqlCommand.Parameters.AddWithValue("@organizationName", organizationName);

                    using (SqlDataReader sqlReader = sqlCommand.ExecuteReader())
                    {
                        table.Load(sqlReader);
                    }
                }
            }
```

Scaling the database (ACID/CAP)

While developing our MSSQL database, we considered ACID properties and the CAP theorem for data integrity, consistency, and system reliability. Our design follows ACID principles for reliable transaction processing, maintaining atomicity, consistency, isolation, and durability.

Regarding the CAP theorem, our database inherently prioritizes Consistency and Availability since it is a SQL server. Not prioritizing Partition tolerance does have potential trade-offs for future scaling needs by limiting horizontal scalability.

Test of operations

Our approach to testing database operations was straightforward, reflecting our current level of experience with testing methodologies. Initially, we developed SQL queries as the foundation for our operations. These queries were manually tested to ensure they executed without errors and returned accurate results. Following the successful validation of these queries, we transformed them into stored procedures. The final step involved executing these stored procedures to verify their functionality and consistency in producing the expected outcomes.

Database tuning and optimization

Following the establishment of our database and the implementation of stored procedures to address our set of ten questions, we proceeded to explore avenues for tuning and optimization. Given the relatively modest size of the datasets at our disposal, we initially anticipated that indexing our tables might not yield substantial benefits. Nevertheless, our coursework introduced us to the SQL Server Profiler, a tool that, on Windows platforms, allows users to select from various tracing templates, including one dedicated to tuning. Utilizing this template, we initiated a trace and executed all our stored procedures, subsequently saving the trace as a data file. This file was then analyzed using the Database Engine Tuning Advisor within SQL Server Management Studio (SSMS), a utility designed to identify optimal indexing opportunities and quantify potential improvements in performance.

Below are some pictures that shows the result:

Estimated improvement: 11%

Object Name	Recommendation	Target of Recommendation
[dbo].[EmissionResult]	create	_dta_index_EmissionResult_7_1221579390__K5_K2_3
[dbo].[Methodology]	create	_dta_index_Methodology_7_1317579732__K2_3_4

```
CREATE NONCLUSTERED INDEX [Index_EmissionResult] ON [dbo].[EmissionResult]
(
    [ReportingYear] ASC,
    [AccountNo] ASC
)
INCLUDE([TotalCityEmissions]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]

CREATE NONCLUSTERED INDEX [Index_Methodology] ON [dbo].[Methodology]
(
    [AccountNo] ASC
)
INCLUDE([PrimaryMethodology],[MethodologyDetails]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
```

The analysis advised creating indexes on the EmissionResult and Methodology tables, citing our stored procedures' reliance on ReportingYear and AccountNo as variables. This focus aimed at these specific areas was determined to be where the most intensive processing occurred in our queries. Acting on this recommendation, we implemented the proposed indexes, recognizing them as crucial for improving our database's operational efficiency.

Conclusion & Recommendations

This report covered our journey of building a database from scratch for our database course project. We started with cleaning up the five datasets, formulated questions that were required in the assignment, and then imported the cleaned data into Microsoft SQL Server. By developing SQL queries, we were able to answer the questions that we made and dive deep into the data, optimizing the performance and setting up some maintenance routines to keep the database running smoothly.
One of the main challenges we faced was dealing with inconsistent data, which we overcame by reorganizing the csv datasets. This not only made our data more reliable but also taught us the importance of structure and quality in databases.

Future recommendations:
For future database projects, we would probably create indexes and statistics for better performance. Also we could set up more SQL Server agents to run stored procedures for maintenance like creating backups etc.

Appendix

1. *Python script for data cleaning & ingesting:*

```
import pandas as pd
import pyodbc

conn_str = (
    r'DRIVER={ODBC Driver 17 for SQL Server};'
    r'SERVER=localhost;'
    r'DATABASE=DB_Soft;'
    r'Trusted_Connection=yes;'
)

df = pd.read_csv("2023_Cities_Climate_Risk_and_Vulnerability_Assessments_20240207.csv")

# Convert 'Last Update" to datetime
df['Last update'] = pd.to_datetime(df['Last update'], format='%m/%d/%Y %I:%M:%S %p')
# Replace NaN values with an empty string
df = df.fillna({
    'Questionnaire': '',
    'Access': '',
    'Assessment attachment and/or direct link': '',
    'Boundary of assessment relative to jurisdiction boundary': '',
    'Factors considered in assessment': '',
    'Primary author(s) of assessment': '',
    'Does the city have adaptation goal(s) and/or an adaptation plan?': '',
    'Last update': pd.Timestamp.now(),
    'Organization Number': 0,
    'Year of publication or approval': 0
})

with pyodbc.connect(conn_str) as conn:
    with conn.cursor() as cursor:
        for index, row in df.iterrows():
            # Check if AccountNo exists in the Organization table
            check_query = "SELECT COUNT(*) FROM Organization WHERE AccountNo = ?"
            cursor.execute(check_query, row["Organization Number"])
            result = cursor.fetchone()
            if result is not None:
                count = result[0]
            else:
                count = 0

            if count == 0:
                # Find Country
                find_city_query = "SELECT CountryID FROM Countries WHERE CountryName =
?"
                cursor.execute(find_city_query, row["Country/Area"])
                result = cursor.fetchone()
                if result is not None:
                    countryID = result[0]
                else:
                    print("CountryID not found in Countries table")
                    continue
```

```
                # Create a new organization
                insert_org_query = """
                INSERT INTO Organization (AccountNo, OrganizationName, CountryID)
                VALUES (?, ?, ?);
                """
                cursor.execute(insert_org_query, (
                    row["Organization Number"],
                    row["Organization Name"],
                    countryID
                ))

            sqlStatement = """
            INSERT INTO Assessment
            (AccountNo, Questionnaire, AccessType, AssessmentAttachment,
BoundaryDescription, PublicationYear, FactorsConsidered, PrimaryAuthors,
AdaptationGoals, LastUpdate)
            VALUES
            (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
            """
            cursor.execute(sqlStatement, (
                row["Organization Number"],
                row["Questionnaire"],
                row["Access"],
                row["Assessment attachment and/or direct link"],
                row["Boundary of assessment relative to jurisdiction boundary"],
                row["Year of publication or approval"],
                row["Factors considered in assessment"],
                row["Primary author(s) of assessment"],
                row["Does the city have adaptation goal(s) and/or an adaptation plan?"],
                row["Last update"]
            ))


conn.commit()
conn.close()
```

2. *Stored procedure that handles rebuilding and reorganizing of indexes*

```
CREATE PROCEDURE SP_OptimizeDatabaseIndexes
AS
BEGIN
    SET NOCOUNT ON;

    CREATE TABLE #FragmentationInfo
    (
        object_id INT,
        index_id INT,
        partition_number INT,
        fragmentation FLOAT
    );
```

```
    INSERT INTO #FragmentationInfo(object_id, index_id, partition_number, fragmentation)
    SELECT
        f.object_id,
        f.index_id,
        f.partition_number,
        f.avg_fragmentation_in_percent
    FROM
        sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED') AS f
    WHERE
        f.avg_fragmentation_in_percent > 5 AND
        f.index_id > 0;

    DECLARE curIndexes CURSOR FOR
        SELECT object_id, index_id, partition_number, fragmentation
        FROM #FragmentationInfo;

    OPEN curIndexes;

    DECLARE @ObjectId INT, @IndexId INT, @PartitionNumber INT, @Fragmentation FLOAT;
    DECLARE @SqlCmd NVARCHAR(MAX);

    FETCH NEXT FROM curIndexes INTO @ObjectId, @IndexId, @PartitionNumber,
@Fragmentation;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @Fragmentation BETWEEN 5 AND 30
        BEGIN
            SET @SqlCmd = 'ALTER INDEX ' + QUOTENAME(INDEX_NAME(@IndexId, @ObjectId)) +
                          ' ON ' + QUOTENAME(OBJECT_SCHEMA_NAME(@ObjectId)) + '.' +
                          QUOTENAME(OBJECT_NAME(@ObjectId)) + ' REORGANIZE;';
            EXEC sp_executesql @SqlCmd;
        END
        ELSE IF @Fragmentation > 30
        BEGIN
            SET @SqlCmd = 'ALTER INDEX ' + QUOTENAME(INDEX_NAME(@IndexId, @ObjectId)) +
                          ' ON ' + QUOTENAME(OBJECT_SCHEMA_NAME(@ObjectId)) + '.' +
                          QUOTENAME(OBJECT_NAME(@ObjectId)) + ' REBUILD WITH (ONLINE =
ON);';
            EXEC sp_executesql @SqlCmd;
        END

        FETCH NEXT FROM curIndexes INTO @ObjectId, @IndexId, @PartitionNumber,
@Fragmentation;
    END;

    CLOSE curIndexes;
    DEALLOCATE curIndexes;

    DROP TABLE #FragmentationInfo;
END;
```