CS 5300 Database Systems
Programming Project 01

Project Team Members:
Hemanth Sirish Kumar Sai Mandava - 12613159
Sivani Rayala - 12607415
Bhargavi Gunji - 12607182
Nivitha Maddela - 12613507

Objective: To develop a program that takes a dataset (relation) and functional dependencies as input, normalizes the relations based on the provided functional dependencies, produces SQL queries to generate the normalized database tables, and optionally determines the highest normal form of the input table.

Given input files:
- exampleInputTable.csv
- dependency_parser.txt
- mvd_dependencies.txt

This project contains the following code files:
- input_parser.py
- sql_table_creator.py
- normalization_procedures.py
- main.py

This project was done and executed in google colab where each file is a cell.

Each code file has it's own functional specification as follows:

1. input_parser.py (This file is for parsing the input)

Function Definitions:

a. check_for_comma(column_data)

Purpose:
To determine if any values in a given DataFrame column contain a comma.
Parameters:
column_data: A pandas Series representing a single column of a DataFrame.
Returns:
A boolean value. True if any value in the column contains a comma, otherwise False.
Usage:

This function is intended to be used internally within the script to identify columns that contain comma-separated values.

b. input_parser(data_file)

Purpose:
To parse a DataFrame, splitting any values that contain commas into lists and trimming whitespace from each element within those lists.
Parameters:
data_file: A pandas DataFrame with string-compatible data that needs to be parsed.
Process:
The function converts all values in the DataFrame to strings. This is done to ensure that the operations that follow, which are specific to string data, can be performed without type errors.
It identifies columns that contain comma-separated values using the check_for_comma function. This is accomplished through a list comprehension that iterates over the DataFrame's columns.
It then iterates over each column identified to have comma-separated values. For each column, it:
Splits the string on each comma.
Applies a lambda function that strips whitespace from each element in the resulting list.
Returns:
The modified pandas DataFrame where all comma-separated values have been split into lists and trimmed of whitespace.
Usage:
This function is intended to be called with a DataFrame as an argument when there is a need to process string data that may contain comma-separated values. It is especially useful in preparing data for further analysis or processing, where such comma-separated values need to be handled individually.

2. normalization_procedures.py (This file is used for all the normalizations)

Utility Functions:

- **is_list_or_set(item)**: Checks if the input item is a list or set.
- **is_superkey(relation, left_hand_side)**: Determines whether the left_hand_side attributes form a superkey in the given relation.
- **powerset(s)**: Generates the powerset of a given set s.
- **project(data, attributes)**: Used within other functions to project a set of attributes from data.
- **is_lossless(df, df1, df2)**: Checks if joining df1 and df2 on common columns is lossless.

Normal Form Check Functions:

- **check_1nf(relation)**: Checks if a relation is in 1NF.
- **check_2nf(primary_key, dependencies)**: Checks if the dependencies satisfy 2NF for a given primary_key.
- **check_3nf(relations, dependencies)**: Checks if each relation in relations is in 3NF based on dependencies.
- **check_bcnf(relations, primary_key, dependencies)**: Checks if each relation is in BCNF.
- **check_4nf(relations, mvd_dependencies)**: Checks if each relation is in 4NF considering multivalued dependencies.
- **check_5nf(relations)**: Checks if each relation is in 5NF and prompts the user for candidate keys.

Normal Form Enforcement Functions:

- **bcnf_decomposition(relation, dependencies)**: Decomposes a relation into BCNF using the given dependencies.
- **validate_first_nf(relation)**: Adjusts a relation to conform to 1NF.
- **validate_second_nf(relation, primary_key, dependencies)**: Decomposes relation into 2NF.
- **validate_third_nf(relations, primary_key, dependencies)**: Decomposes each relation in relations into 3NF.
- **validate_bc_nf(relations, primary_key, dependencies)**: Enforces BCNF on each relation in relations.
- **validate_fourth_nf(relations, mvd_dependencies)**: Decomposes relations into 4NF considering mvd_dependencies.
- **decompose_5nf(dataframe, candidate_keys)**: Decomposes a dataframe into 5NF based on candidate_keys.
- **validate_fifth_nf(relations, primary_key, dependencies)**: Decomposes relations into 5NF.

3. sql_table_creator.py (This file is used for output generator)

Functions Overview:

convert_dtype_to_sql(data_type): Converts pandas data types to corresponding SQL data types.

generate_1nf(pkeys, relation): Generates a SQL command for creating a table that adheres to the First Normal Form (1NF).

generate_2nf_3nf(assoc_relations): Generates SQL commands for creating tables that adhere to the Second (2NF) and Third Normal Forms (3NF).

generate_bcnf_4nf_5nf(relation_sets): Generates SQL commands for creating tables that adhere to the Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF).

Function Details:

convert_dtype_to_sql(data_type)

Purpose: Converts a data type from a pandas DataFrame to a SQL-compatible data type.

Arguments:

data_type: The data type from a pandas DataFrame to be converted.

Returns: A string representing the SQL data type.

Supported Conversions:

int to INT

float to FLOAT

object to VARCHAR(255)

datetime to DATETIME

Any other type to TEXT

generate_1nf(primary_keys, df):

Purpose: Generates and prints a SQL CREATE TABLE statement for a table structured in First Normal Form.

Arguments:

pkeys: A list of primary key column names.

relation: An object that has columns and dtypes attributes (presumably a pandas DataFrame).

Behavior:

Constructs a table name by concatenating primary key column names with "_relation".

Iterates over the columns in the relation, applying convert_dtype_to_sql to the column data types.

Sets the primary key(s) using the columns listed in pkeys.

Removes the trailing comma and newline from the SQL command string before closing the SQL statement.

generate_2nf_3nf(relations):

Purpose: Generates and prints SQL CREATE TABLE statements for tables structured in Second and Third Normal Forms.

Arguments:

assoc_relations: A dictionary where keys are tuples of primary key column names and values are objects with columns and dtypes attributes (presumably pandas DataFrames).

Behavior:

Similar to generate_1nf but iterates over a dictionary of relations.

generate_bcnf_4nf_5nf(relation_sets)

Purpose: Generates and prints SQL CREATE TABLE statements for tables structured in

generate_bcnf_4nf_5nf(relations):

Arguments:

relation_sets: A list of objects each with a columns and dtypes attribute (presumably a list of pandas DataFrames).

Behavior:

Similar to generate_1nf, but assumes the first column in each relation is the primary key.

4. **main.py** (This is the main file for reading CSV data and performing normalization operations)

The script reads a CSV file and a dependencies file, processes the data, and normalizes the data up to a specified normal form as per user input. The supported normal forms include 1NF (First Normal Form), 2NF (Second Normal Form), 3NF (Third Normal Form), BCNF (Boyce-Codd Normal Form), 4NF (Fourth Normal Form), and 5NF (Fifth Normal Form).

Dependencies:
The script imports the following Python modules:

pandas: A powerful data manipulation and analysis library for Python.
csv: A module for reading and writing CSV files.
normalization_procedures: A custom module, presumably containing functions that validate various normal forms.
input_parser: A custom module, likely used to parse and process the input CSV data.
sql_table_creator: A custom module that seems to contain functions to generate SQL queries for creating tables in various normal forms.

Main Functionality:
CSV Data Input:
Reads a CSV file using pandas into a DataFrame (input_file).
Prints the content of the input table.
Dependency Parsing:
Reads a file that contains functional dependencies and parses it into a dependencies dictionary.
User Input:
Prompts the user to select the target normal form to achieve.
Asks if the user wants to find the highest normal form of the input table.
Accepts the primary key values from the user.
Multi-valued Dependency Parsing (If applicable):
If the target normal form is 4NF or 5NF, and not BCNF, it reads another file containing multi-valued dependencies.
Parses the multi-valued dependencies and stores them in a mvd_dependencies dictionary.
Normalization Procedures:
The script checks the input data against the selected normal form target and executes normalization procedures using the normalization_procedures module.
For each normal form, the script:
Validates if the data meets the requirements of the normal form.
Prints messages if the data is already normalized.

Generates and prints the SQL queries needed to decompose the tables into the target normal form using the sql_table_creator module.

Print Highest Achieved Normal Form:

If the user opted to find the highest normal form, the script will print out the highest normal form that the input data has been normalized to.

➔ Execution flow:

The execution flow of the script can be summarized in the following steps:

Import Dependencies:

- The script begins by importing the necessary Python libraries and modules.

Read Input Files:

- It reads the CSV data into a pandas DataFrame.
- It reads the functional dependencies from a text file and stores them in a dictionary.

Display Initial Data:

- The script prints out the read CSV data and the parsed dependencies for the user to review.

User Interaction:

- It prompts the user to specify the highest normal form to be achieved.
- It asks the user whether to determine the highest normal form the input table can reach.
- It collects the primary key(s) from the user input.

Parse Multi-valued Dependencies (If Needed):

- If targeting 4NF or 5NF (excluding BCNF), it reads and parses a file containing multi-valued dependencies.

Data Parsing:

- The script calls a function from input_parser to process the input DataFrame.

Normalization Steps:

- The script performs a series of conditional checks based on the target normal form selected by the user.
- For each normal form (1NF to 5NF and BCNF), the script:
    - Calls the appropriate validation function from normalization_procedures to check if the data meets the criteria for that normal form.
    - Updates the highest normal form achieved if the data meets the criteria.
    - If the data is already normalized to a specific normal form, it prints a message stating this.
    - If the user has chosen to normalize the data to a specific normal form, it prints the SQL queries for creating tables in that normal form, generated by sql_table_creator.

Final Output:

- If the user wanted to know the highest normal form possible for the input data, the script prints out the result.
- Throughout the process, the script also prints out messages and generated SQL queries based on the user's choices.