CIS 425

Project 2 – Database integration

Hassan Mehdi, Matthew London

11/14/2018

Logic –

We first ensured that one application could call both databases with a single query. To achieve this, we linked one of our projects to the others database, and created a new table fill field. An issue we ran into was that we had used different names for our tables, even though the information in them was the same. To get around this, when a SQL query was made, the database that didn't have that table went into an if statement that simply speaking said "if the table that was fetched had [identifying data column name] in it, then run the same SQL query but with [relevant table name] on the other database, and change its column names to match the first. This allowed us to have two data table objects filled with relevant data, and matching column names. To be clear, .NET acquires a DataTable data-structure, along with its subsidiaries DataRow and DataColumn. These data encapsulations where used continuously throughout the project – we aren't sure this integration module would have been possible without them.

After gathering a general understanding of database manipulation in .NET architecture, we decided to tackle the next challenging requirement, which was to implement the global schema query module. For strictly presenting the data to be output, custom .NET DataTable classes were built to outline the Global schema of G-Cars, G-Customers, and G-Rentals. Hardcoding these tables made it easy in the future to manipulate data back to the user. Next required parsing the global schema query to the core databases. The original query string is processed using regular expressions to create two new custom queries – one for the London database, and one for the Mehdi Database.

After the finally fabricating the correct tools to create custom queries for each database, we had to next develop the DataTable Merge function. This part turned out being extremely tedious, frustrating, patience-testing, and most importantly, the most rewarding classwork thus far. Even though its current state shares a resemblance to spaghetti code, it provides a robust merge function. What led to this section being very tedious would be the constant need to reformat temporary DataTables in order to produce the correct data structure with harmonious datatypes and column names from both the London and Mehdi Database. Nearly every single attribute in the entire schema had disjoint datatypes, which were eventually sorted through and organized.

After producing the adequate DataTable from the joint query, the subsequent DataTable is placed in a DataGridView element and presented to the user. There are two figures below, Figure 1a mimics a context level diagram and represents the Integration Module base model; while Figure 1b represents a "diagram 0" view of the Integration Module.

Figure 1a

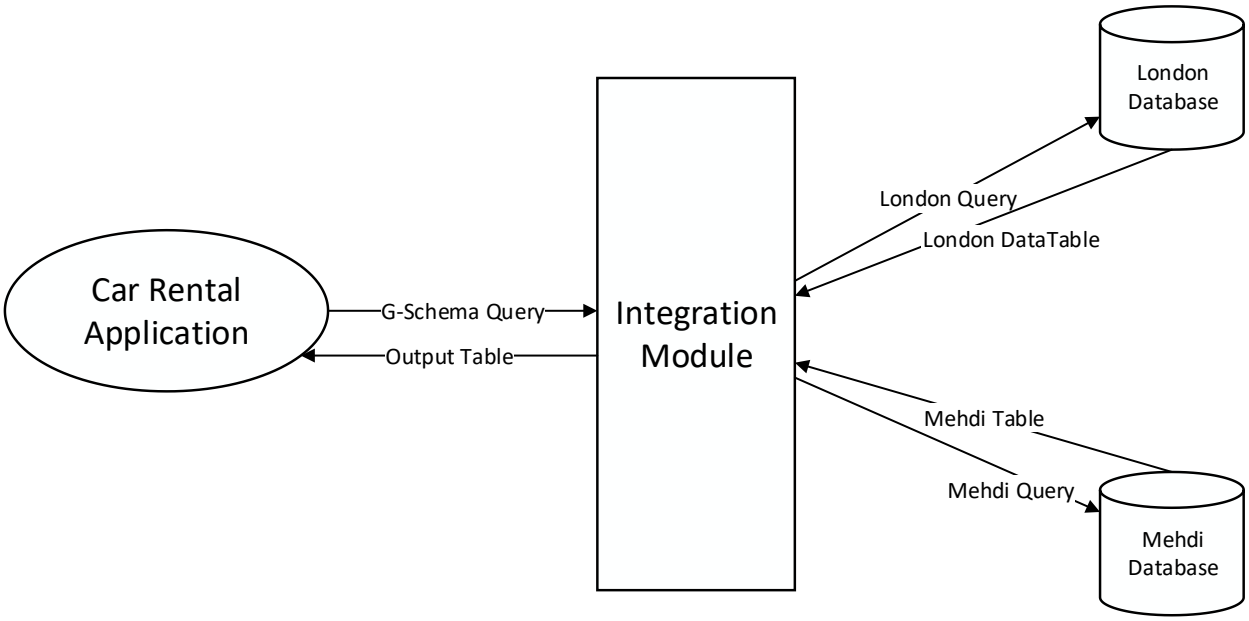
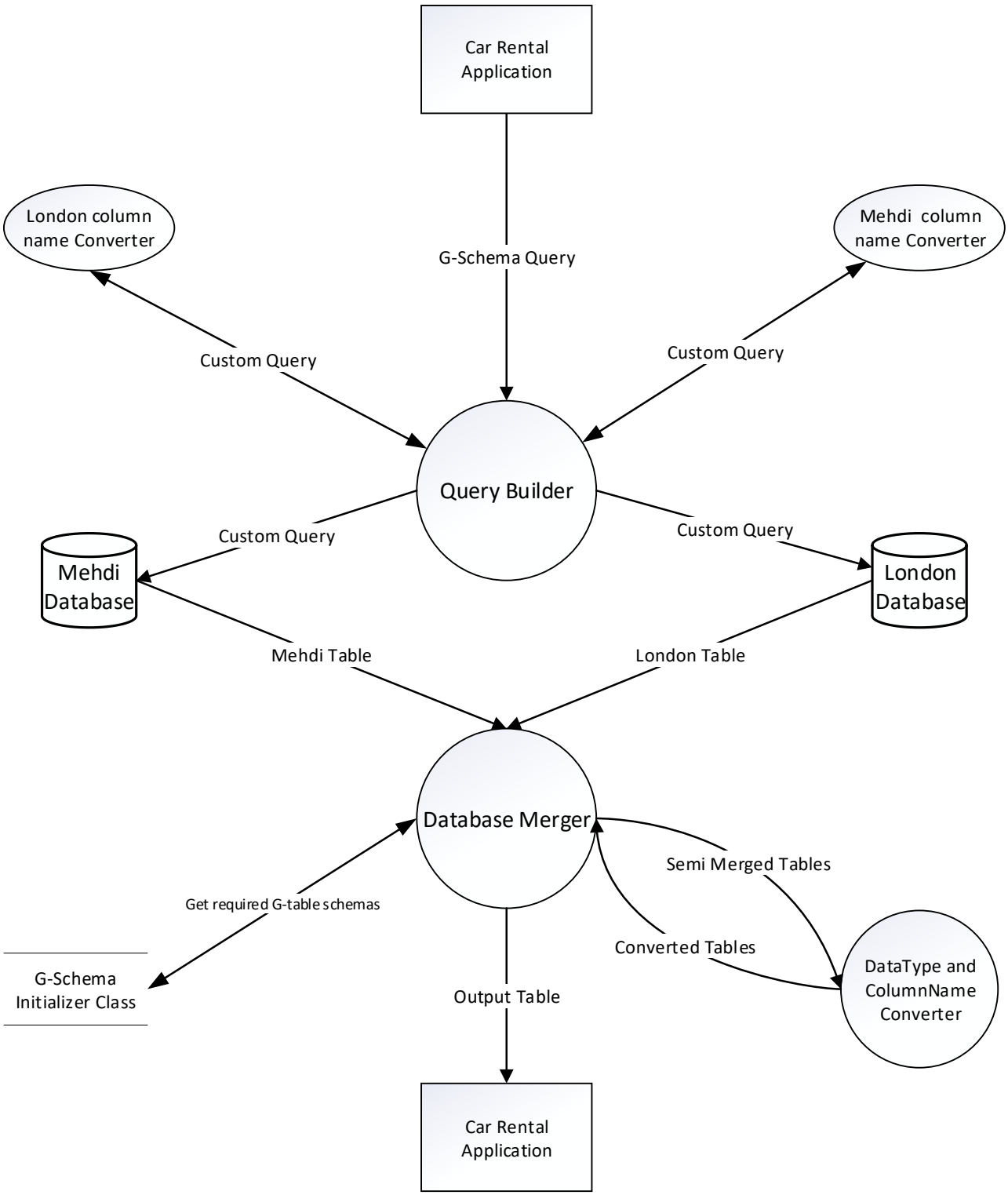

London
Database
London Query
London DataTable
Car Rental
Application
G-Schema Query
Output Table
Integration
Module
Mehdi Table
Mehdi Query
Mehdi
Database

Figure 1b


Car Rental Application
G-Schema Query
London column name Converter
Mehdi column name Converter
Custom Query
Custom Query
Query Builder
Custom Query
Custom Query
Mehdi Database
London Database
Mehdi Table
London Table
Database Merger
Semi Merged Tables
Get required G-table schemas
Converted Tables
G-Schema Initializer Class
Output Table
DataType and ColumnName Converter
Car Rental Application

Changes to the original Databases –

The only change is minimal and should not result in penalization. In Mehdi's original database, there was the mistake of not including the DateOfBirth column from the original project 1. Without this addition, g-customers would have lost data in the joined query, so it was critical that we apply it.

Adding 3rd Database –

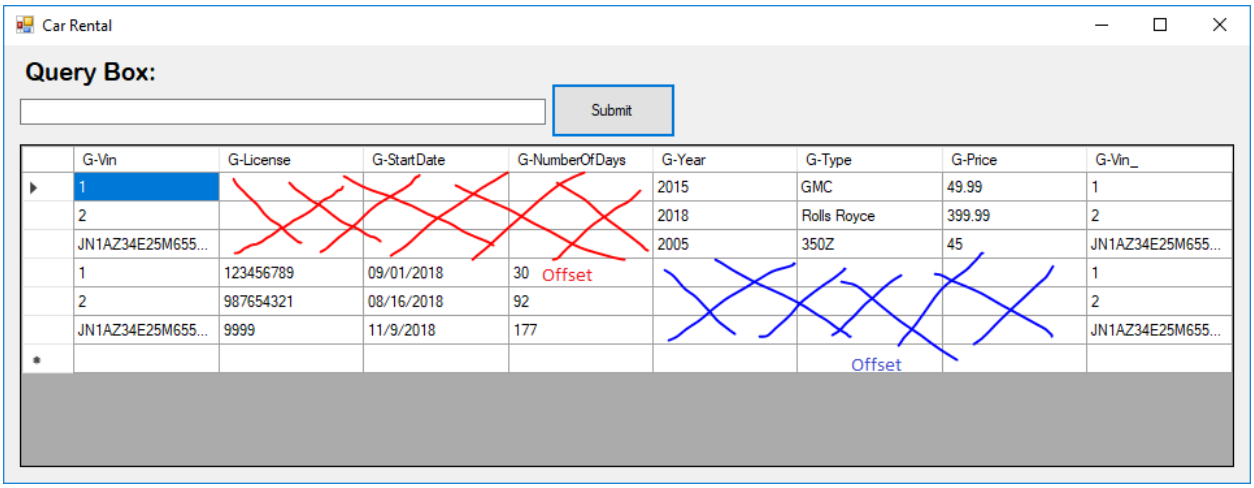
Would need to implement strict column formatting additions for any new columns not accounted for previously. Such column formatting characteristics involve column names, and datatypes – the two largest obstacles in database merge integration. If you notice our last names are close in proximity alphabetically, so we ended up having the same assigned database schema. This simply means there are some hardcoded values that must be added to query builder and merger when modifying column names if we are to add more databases. Otherwise, we took a focused approach in building the output table ***dynamically*** and should handle new information handling without too much issue. To elaborate, the final output table's columns were added in iterations depending on what tables were being looked through.

Rough Test Plan –

Input	Expected Output	Status
Basic select all from a table – Select * from g-customers	All database tuples from both databases populated in Global Schema	Pass
Basic select all from a table – Select * from g-cars	All database tuples from both databases populated in Global Schema	Pass
Basic select all from a table – Select * from g-rentals	All database tuples from both databases populated in Global Schema	Pass
Select partial columns from a table – Select g-vin, g-type from g-cars	All database tuples from both databases populated in Global Schema, but only g-vin and g-type are populated	Pass
Select partial columns including a database tuple attribute that doesn't exist in either internal Databases – Select g-vin, g-color from g-cars	All tuples populated, but only values from g-vin are present (because no g-color values exist)	Pass
Select one column that does not exist in either database – Select g-discount from g-rentals	The empty table	Pass

Select all columns on a join between g-cars and g-rentals on g-vin – select * from g-cars inner join g-rentals on g-cars.g-vin = g-rentals.g-vin **OR** select * from g-cars, g-rentals where g-cars.g-vin=g-rentals.g-vin	A populated table with columns from both tables, an extra vin column, and all data merged into one row	Partial pass/fail Failure is due to the last constraint: Not all rows are merged, rather the correct tuples are offset. (note: no extraneous tuples are present, see figure Test1 below)
Select all columns on a join between g-customers and g-rentals on g-license – select * from g-customers, g-rentals where g-customers.g-license=g-rentals.g-license	A populated table with columns from both tables, an extra license column, and all data merged into one row	Partial pass/fail See above row
Select from g-license on condition- Select * from g-customers where g-license=9999	Only one tuple returned, which comes from Matt's database	pass
Select from g-license on condition with incorrect data format- Select * from g-customers where g-license='9999'	Message box appears for each database query saying the query is flawed	pass

Figure Test1



Note: The lack of screenshots is due to the eventual demo, so screenshots would be redundant. We plan to execute the rough test plan for our demo.

Contributions –

Hassan Mehdi – wrote code for integration, wrote report

Matthew London – wrote code for integration, wrote report, testing