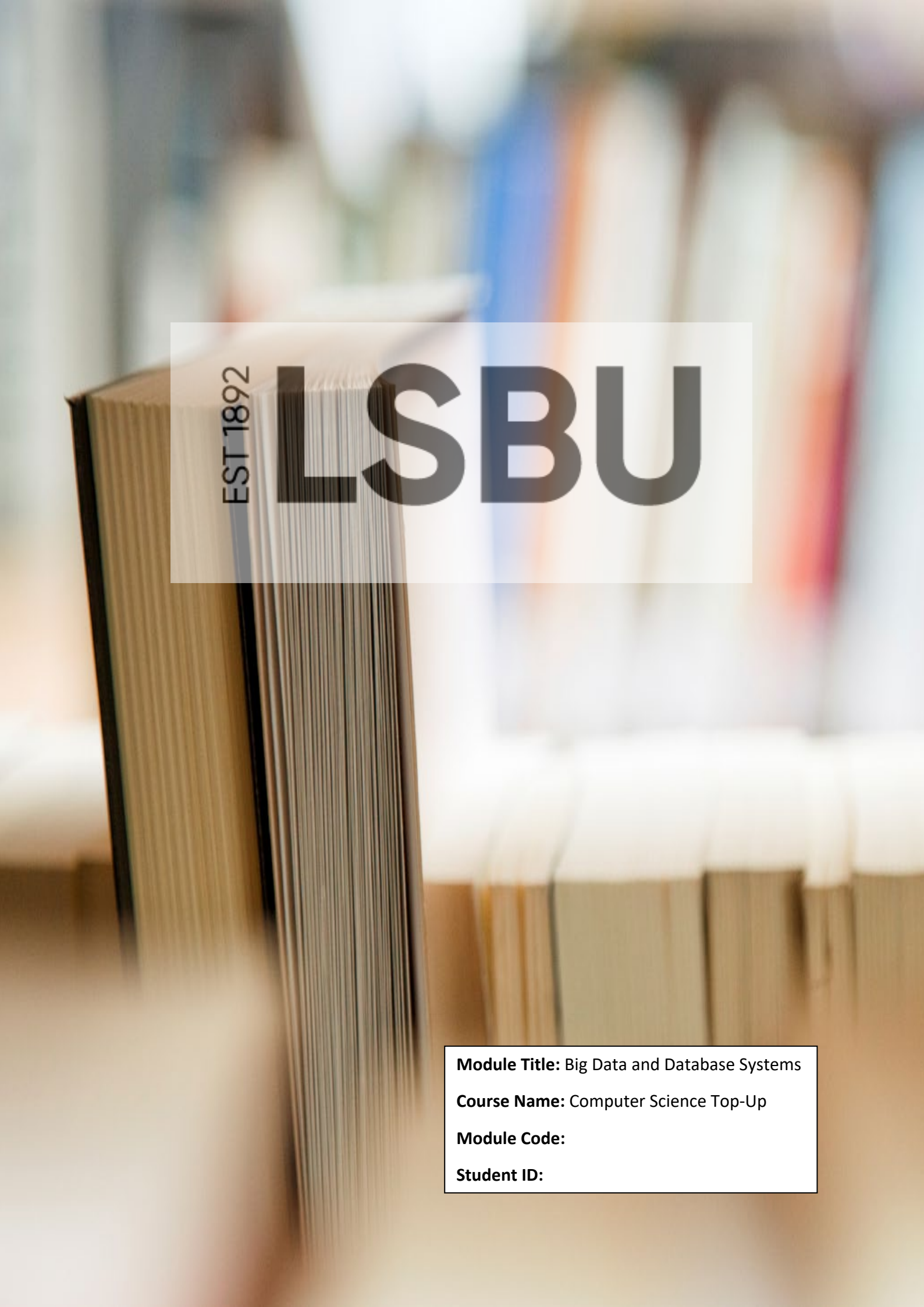

Module Title: Big Data and Database Systems

Course Name: Computer Science Top-Up

Module Code:

Student ID:

Table of Contents

1. Abstract

This work focuses on the design of a database system for LSBU Smart Home, which automates the management of clients, buildings, and IoT sensors. Through the implementation of queries and stored procedures, as shown in the code figures, the system allows for efficient management of device inventory and the assignment of specialized personnel for installations. The trg_PreventDoubleBooking trigger prevents double staffing, and queries allow for generating reports on installations performed within a specific period, as seen in the queries for details of installations performed in a given month. The billing process is also facilitated by incorporating new payment methods. This system optimizes LSBU Smart Home's operations, ensuring resource availability and the proper integration of devices in each installation.

2. Introduction

In the new era of technology, the continuous development of information systems is essential for businesses to keep growing, especially in sectors like the Internet of Things (IoT). Advances in smart sensors have made it possible for homes to improve security, efficiency, and comfort for people. In this context, LSBU Smart Home specializes in the design, maintenance, and automation of smart home systems by implementing the most advanced IoT sensors.

The aim of this assignment is to design a prototype information system for LSBU Smart Home to optimize designs, teamwork, client management, inventory management, and control of IoT components. It also focuses on collecting data from installed IoT sensors in order to support better real-time decision-making.

A comprehensive database design has been proposed to efficiently store and manage data, linking it directly to the operational processes of LSBU Smart Home, as described in the Case Study. Some of these implementations require the integration of IoT sensors with controllers, strong communication with a central database, and the potential use of cloud platforms to support flexible payment methods such as PayPal and Google Pay.

3. Entity Relationship Diagram

3.1 Objectives of the model

The objective of this model is to build an operational system for LSBU Smart Home that includes managing clients, buildings, personalized IoT sensor designs, installations, inventory control with multiple suppliers, staff allocation, and the collection and analysis of smart home data.

3.2 Key entities and relationships

The initial entities considered in the initiation of this project were the following:

- Client
- Building
- Design
- DesignComponent
- Sensor
- Controller
- Staff
- InstallationTeam
- Supplier
- StockItem
- StockDevice
- SensorReading
- CentralSensorLog
- Invoice
- Payment

These entities served as the basis for developing the LSBU Smart Home database system, based on the functional and business requirements provided in Case Study C. However, while working on the physical database using SQL Server, it was necessary to adjust and expand the relational model to better reflect real-world scenarios.

As a result, the following tables were added:

- *ClientBuilding*, to allow multiple clients to be linked to the same building
- *TeamMember*, to enable flexible staff assignment across multiple teams
- *SensorControllerCompatibility*, to register compatibility between specific sensors and controllers

- *StockCheckLog*, to track inventory changes and maintain stock accuracy over time

These changes are observed in *Figure 1*, which shows the updated Entity Relationship Diagram (ERD) that includes 20 normalized tables. The ERD follows the IE Crow's Foot notation, used to represent *one-to-one*, *one-to-many*, and *many-to-many* relationships. All primary and foreign keys were implemented and verified through constraints in SQL Server, ensuring that the structure is relationally sound and accurately aligned with the business model (Dybka, 2016).

During the creation of the LSBU Smart Home database, I made several changes to the relational model in order to meet the requirements set in Case Study C. Initially, the model only included a few basic entities like *Client, Building, Sensor, Controller*, and *Invoice*. But as the system developed and was tested through SQL scripts, it became clear that more tables were needed to support the necessary relationships and business rules.

- Since several clients can share a single building, I added the ClientBuilding table
- Each building can have its own unique design, whether implemented by LSBU or not
- Sensor and controller compatibility needed to be recorded, so I added *SensorControllerCompatibility*
- Installation teams required flexible staff management, which led to the creation of *TeamMember*
- Periodic sensor readings needed to be sent securely to LSBU's central database, which required *CentralSensorLog*
- A proper inventory control system with support for multiple suppliers needed to be implemented using *StockItem, StockDevice*, and *StockCheckLog*

Based on these needs, I modified the *ClientBuilding* table to manage shared buildings, created *DesignComponent* to capture specific components used within each design, and added *TeamMember* to allow flexible team management for installations. The *SensorControllerCompatibility* table was added to track which controllers work with specific sensors, and *CentralSensorLog* was introduced to transmit sensor data to the LSBU data centre securely.

Finally, I included *Invoice* and *Payment* tables to fulfil the requirement of allowing future integration of multiple payment methods, including PayPal and Google Pay.

Once again in *Figure 1*, the final Entity Relationship Diagram (ERD) shows the complete data model containing 20 normalized tables, all of which follow Third Normal Form (3NF) and, in some cases, Boyce-Codd Normal Form (BCNF) (Visual Paradigm, 2019). The ERD demonstrates proper use of primary and foreign keys, and the data integrity has been enforced through SQL constraints. The tables have also been populated with realistic and logically connected test data that match the structure and logic described in the Case Study.

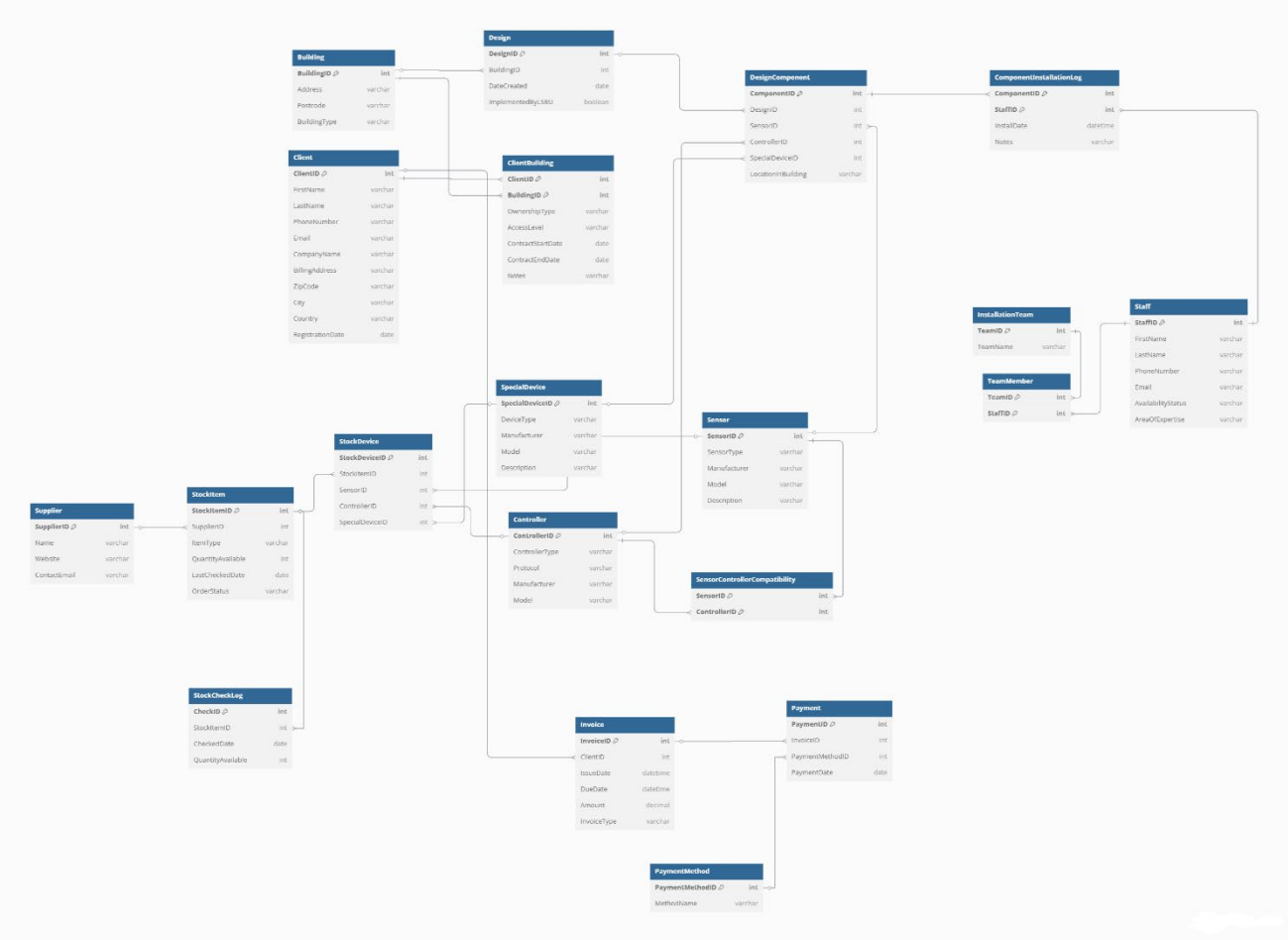

Figure 1.ERD

As well, *Figure 2*, shows the ERD table with the relation between the groups:

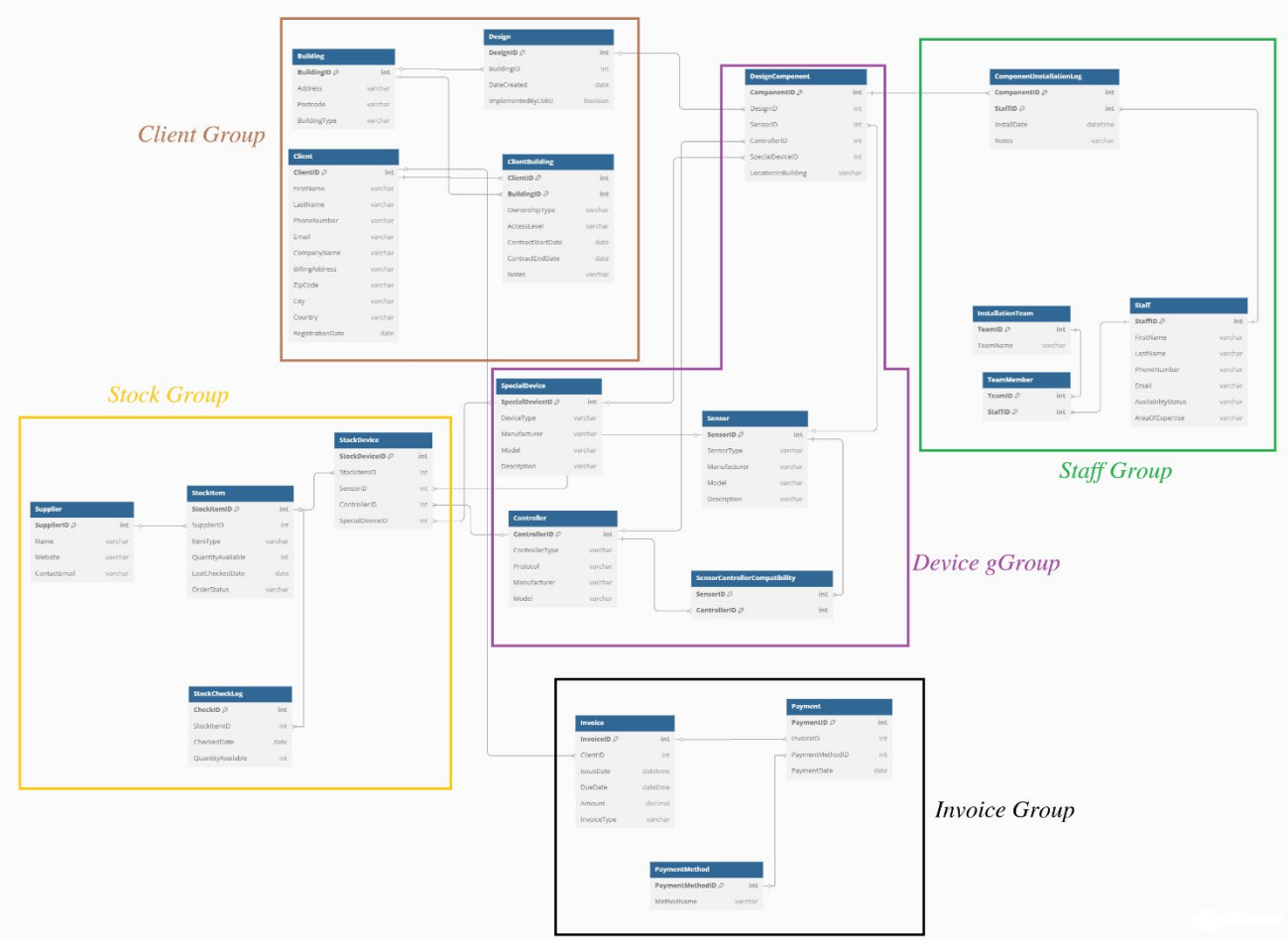

Figure 2. ERD by Groups

4. Functional Dependency statements

Functional dependency describes how a column in a table relates to other columns. It means that one attribute's value can determine another's value. This concept is important to maintain integrity and consistency in a database. Functional dependencies are also a key part of database normalization, which helps avoid data duplication and maintain accuracy (Staff, 2024).

Normalization is a design process used to reduce redundancy and prevent database anomalies. It ensures that data is logically organized across related tables. The process is guided by several normal forms (Microsoft, 2024), each with specific rules.

First Normal Form (1NF):
All attributes must be atomic. Each column contains only one value and a primary key uniquely identifies each row. This helps structure data clearly and eliminates duplicate values in columns (Microsoft, 2024).

Second Normal Form (2NF):
Once a table is in 1NF, 2NF checks that every non-key attribute depends on the entire primary key. This prevents partial dependencies, which occur when a non-key attribute depends on only part of a composite key. In 2NF, all attributes describe the entity represented by the full primary key (Microsoft, 2024).

Third Normal Form (3NF):
After a table reaches 2NF, the goal is to optimize its structure. In this stage, transitive dependencies must be removed. This means no non-key attribute should depend on another non-key attribute. Every non-key value should depend directly on the primary key. This helps maintain a clean, logical structure without unnecessary or indirect relationships (Microsoft, 2024).

In *Table 1,* the 20 tables that make up the relational model are presented, along with their corresponding attributes and functional dependencies. As shown, each table has a clearly defined primary key, and all non-key attributes depend solely on that primary key, which confirms compliance with the Second Normal Form (2NF).

For example: InvoiceID → ClientID, IssueDate, DueDate, Amount, InvoiceType in the *Invoice* table.

Additionally, it can be confirmed that there are no transitive dependencies since no non-key attribute determines another non-key attribute within the same table. All attributes depend directly and exclusively on their respective primary key, as seen in the *Sensor* table:

SensorID → SensorType, Manufacturer, Model, Description.

Finally, it is important to highlight that all functional dependencies originate from primary keys to non-key attributes, fulfilling the requirements of the Third Normal Form (3NF). The relationships have been correctly normalized, and bridge tables such as *ClientBuilding* and *TeamMember* use composite keys properly to represent many-to-many relationships, reinforcing the integrity and consistency of the relational model design.

Table 1. Functional Dependency

TABLE	Functional Dependency
Client	ClientID → FirstName, LastName, PhoneNumber, Email, CompanyName, BillingAddress, ZipCode, City, Country, RegistrationDate
Building	BuildingID → Address, Postcode, BuildingType
ClientBuilding	(ClientID, BuildingID) → OwnershipType, AccessLevel, ContractStartDate, ContractEndDate, Notes
Sensor	SensorID → SensorType, Manufacturer, Model, Description
Controller	ControllerID → ControllerType, Protocol, Manufacturer, Model
SpecialDevice	SpecialDeviceID → DeviceType, Manufacturer, Model, Description
Design	DesignID → BuildingID, DateCreated, ImplementedByLSBU
DesignComponent	ComponentID → DesignID, SensorID, ControllerID, SpecialDeviceID, LocationInBuilding
SensorControllerCompatibility	(SensorID, ControllerID) → Compatibility
Supplier	SupplierID → Name, Website, ContactEmail
StockItem	StockItemID → SupplierID, ItemType, QuantityAvailable, LastCheckedDate, OrderStatus
StockDevice	StockDeviceID → StockItemID, SensorID, ControllerID, SpecialDeviceID
StockCheckLog	CheckID → StockItemID, CheckedDate, QuantityAvailable
Staff	StaffID → FirstName, LastName, PhoneNumber, Email, AvailabilityStatus, AreaOfExpertise
InstallationTeam	TeamID → TeamName
TeamMember	(TeamID, StaffID) → RoleInTeam
ComponentInstallationLog	(ComponentID, StaffID) → InstallDate, Notes
Invoice	InvoiceID → ClientID, IssueDate, DueDate, Amount, InvoiceType
PaymentMethod	PaymentMethodID → MethodName
Payment	PaymentID → InvoiceID, PaymentMethodID, PaymentDate

5. Transforming design to implementation

In this section, I will demonstrate through screenshots the creation of the 20 tables of this database, as well as the insertion of values into each table using Microsoft SQL Server.

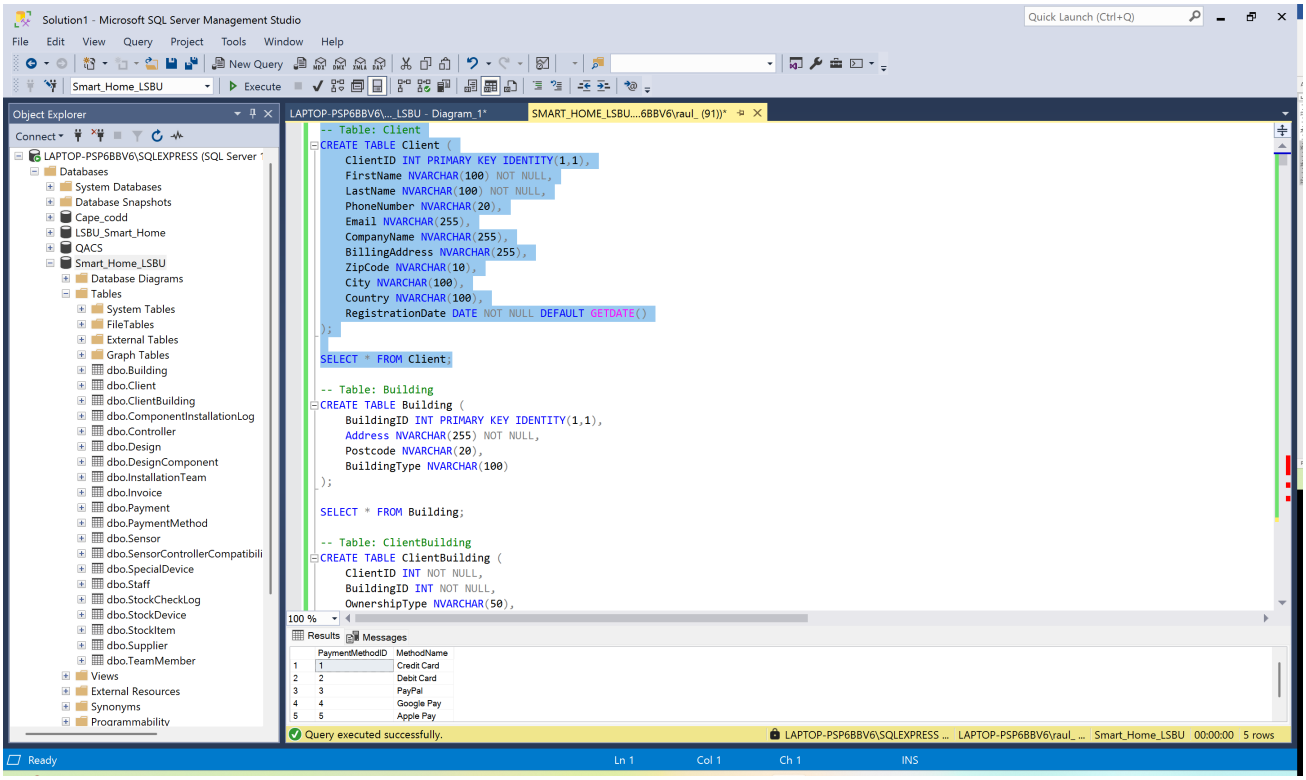

Figure 3. Create Client

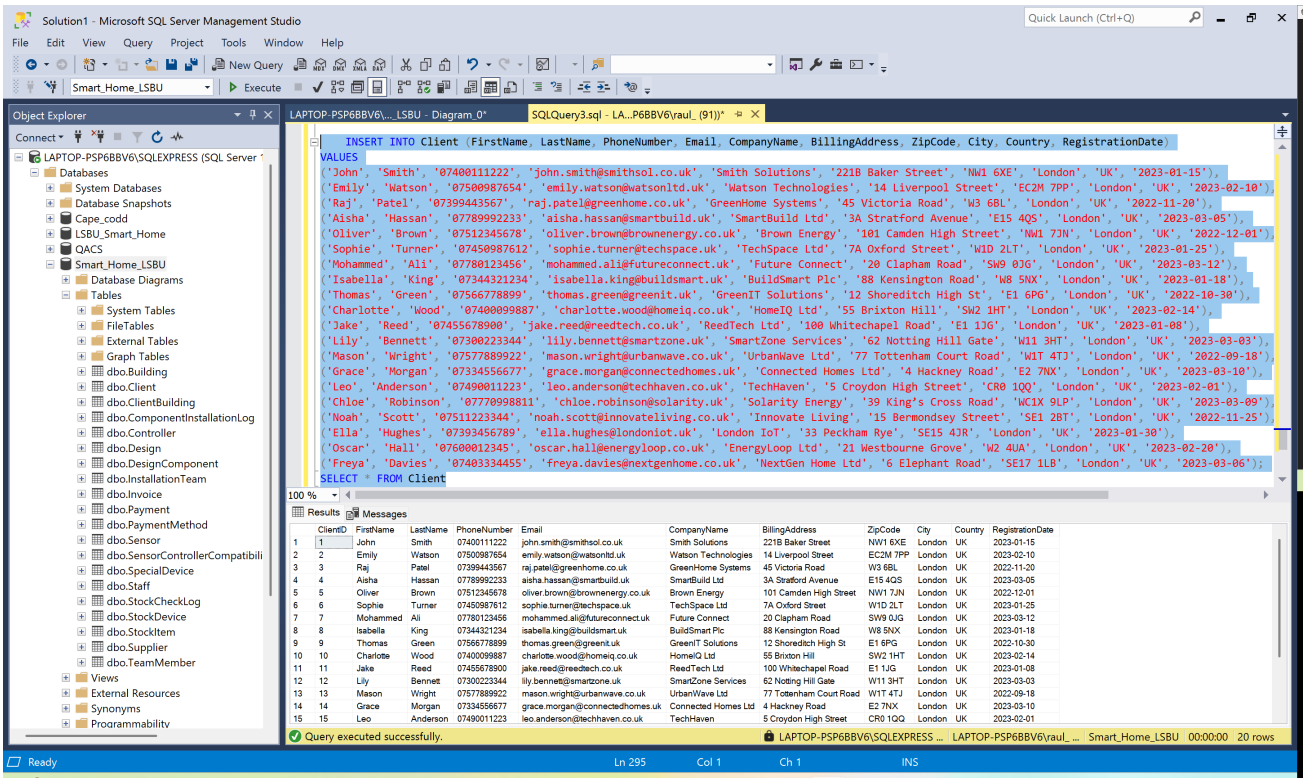

Figure 4.Insert Client Values

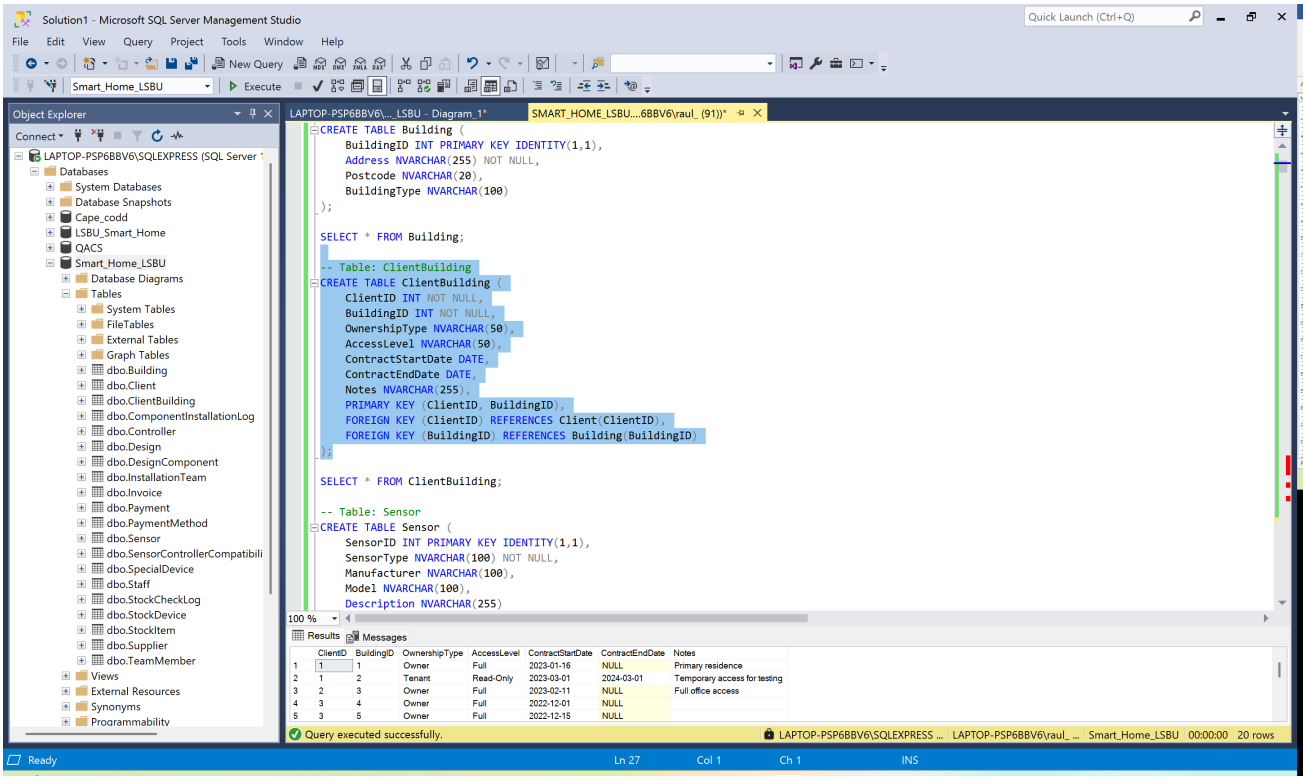

Figure 5.Create Building

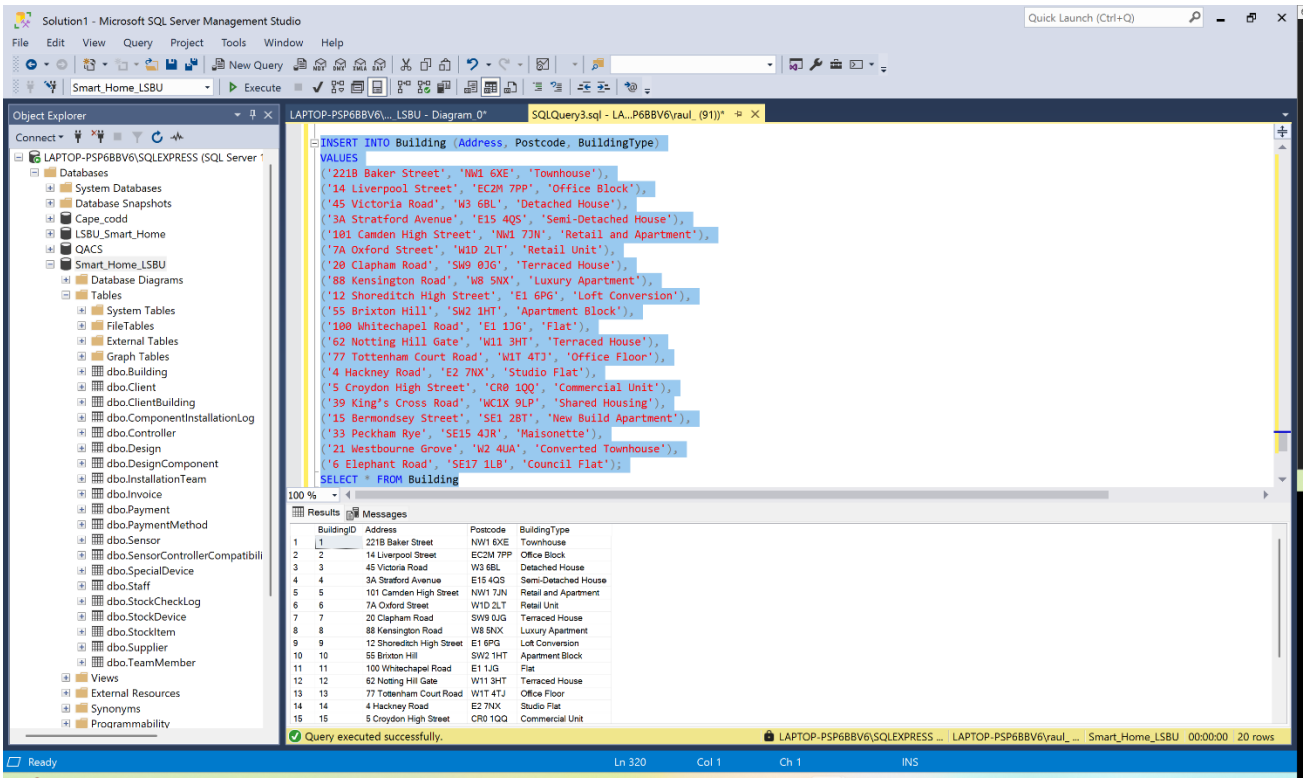

Figure 6. Insert Building Values

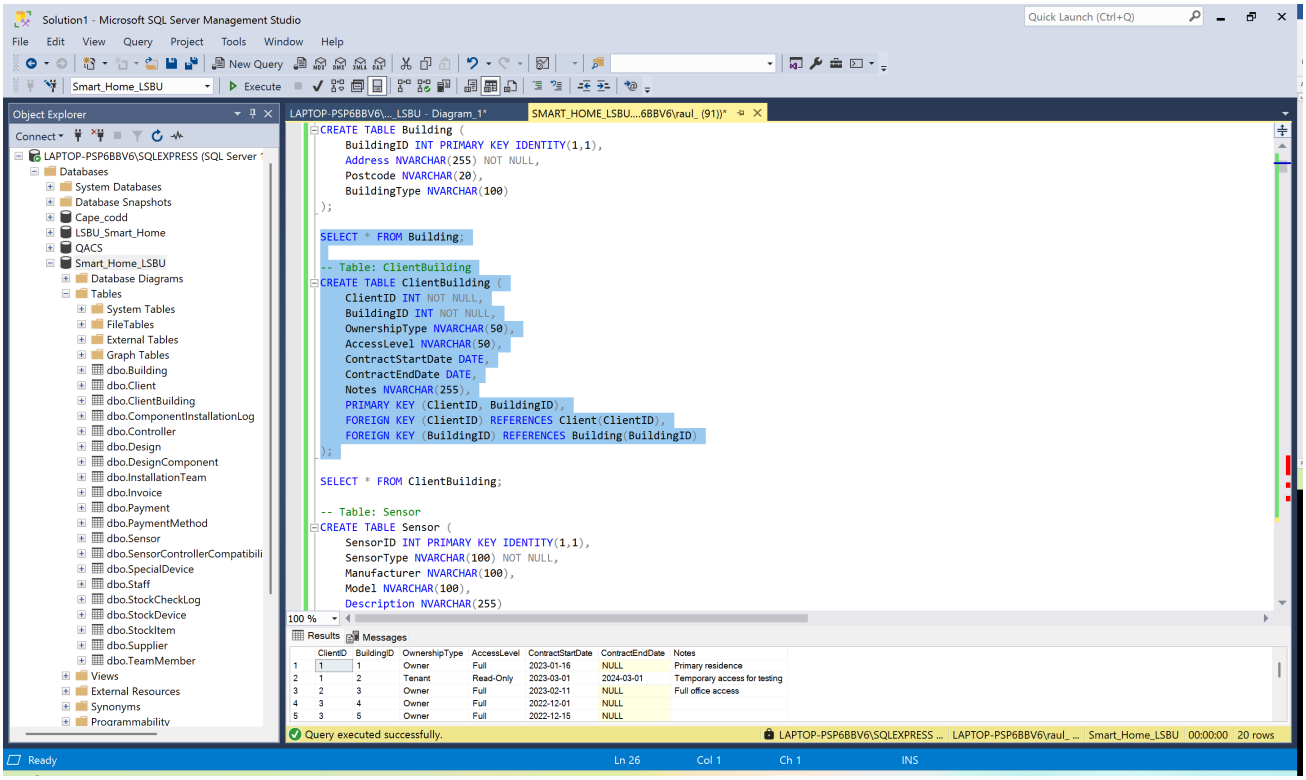

Figure 7. Create ClientBuilding

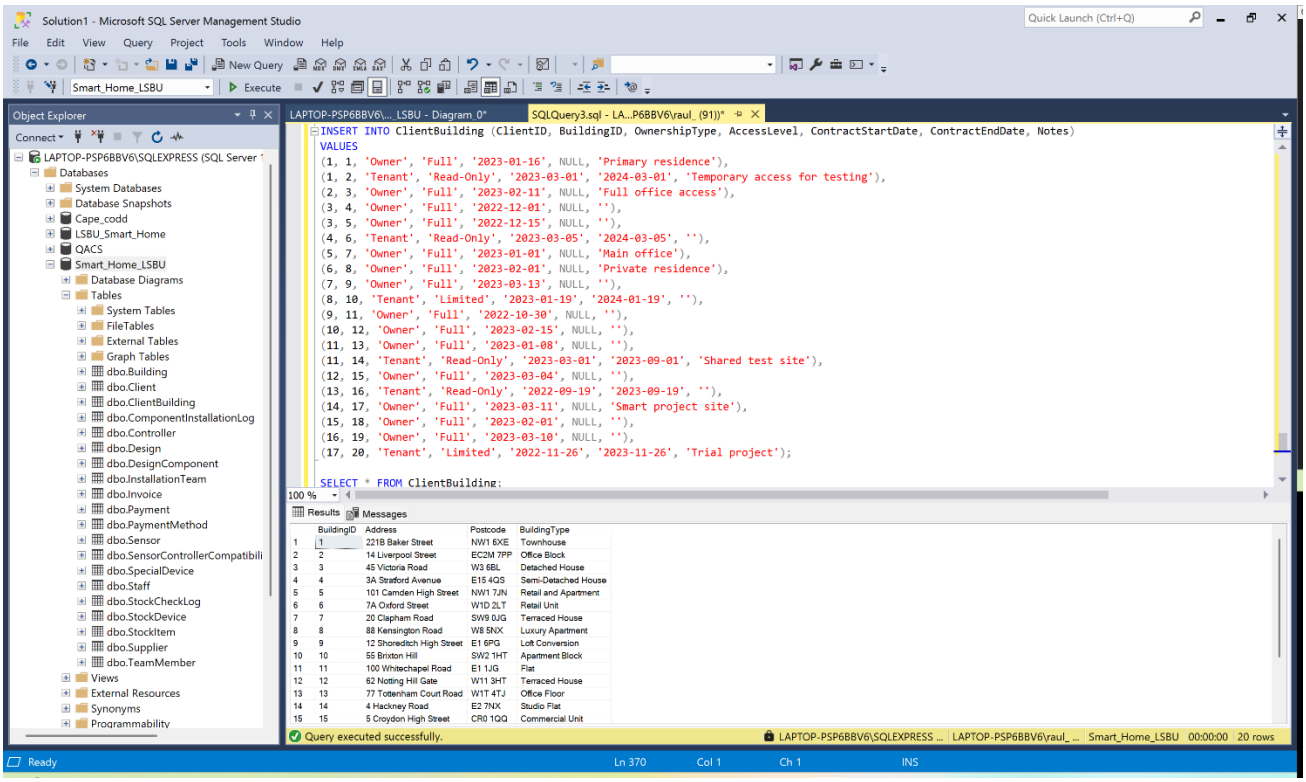

Figure 8. Insert ClientBuilding Values

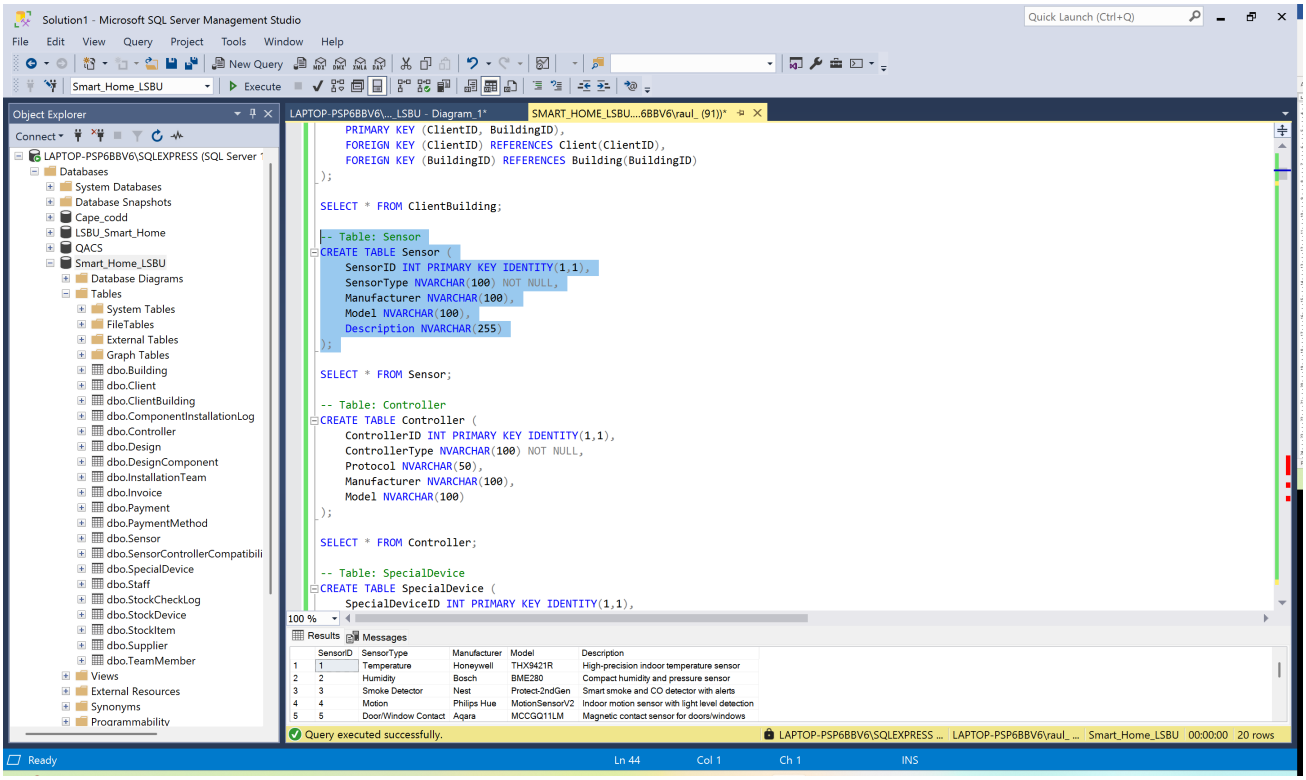

Figure 9. Create Sensor

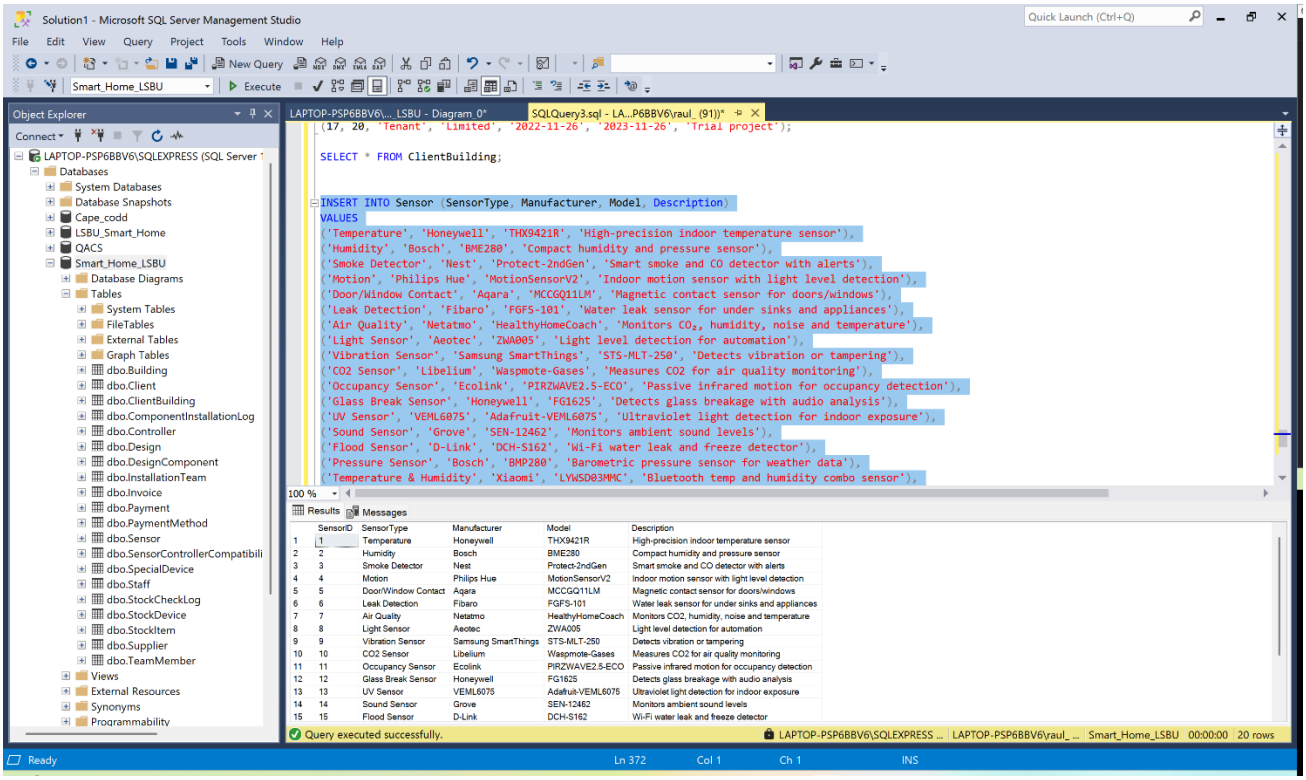

Figure 10. Insert Sensor Values

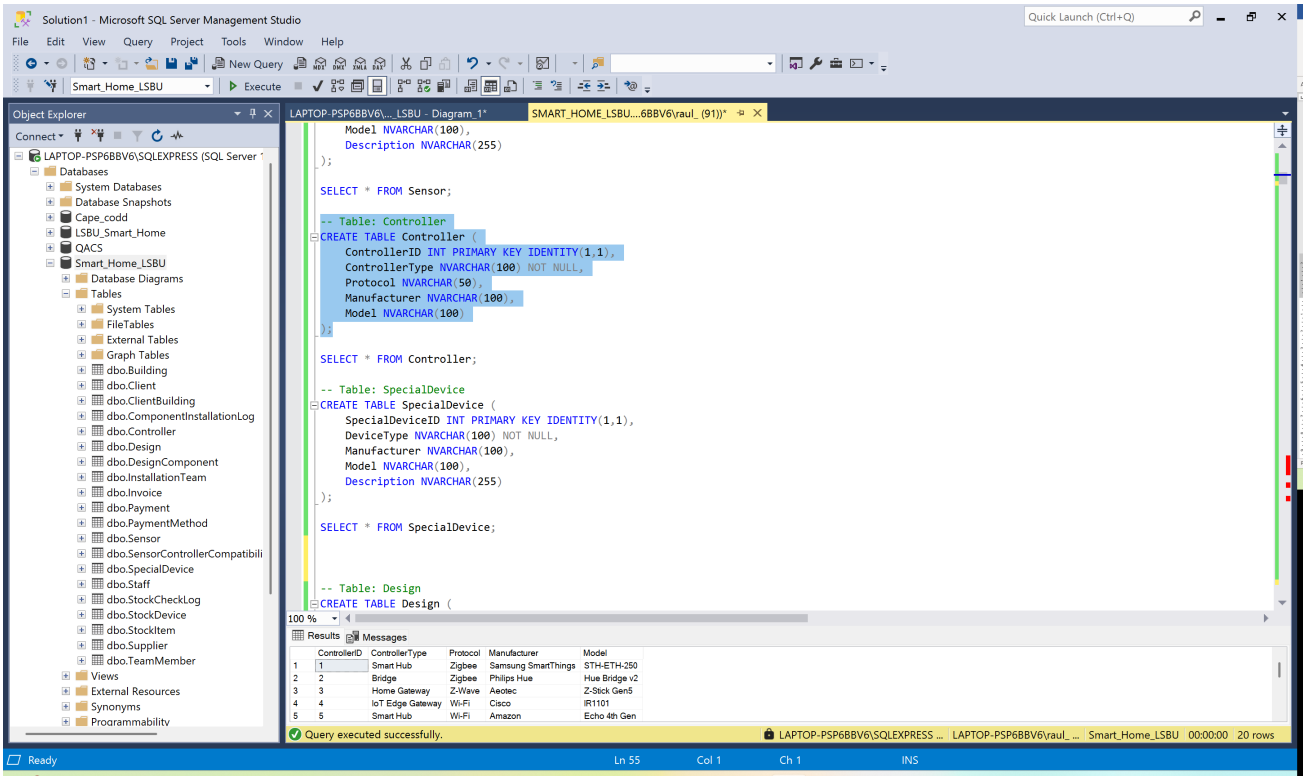

Figure 11.Create Controller

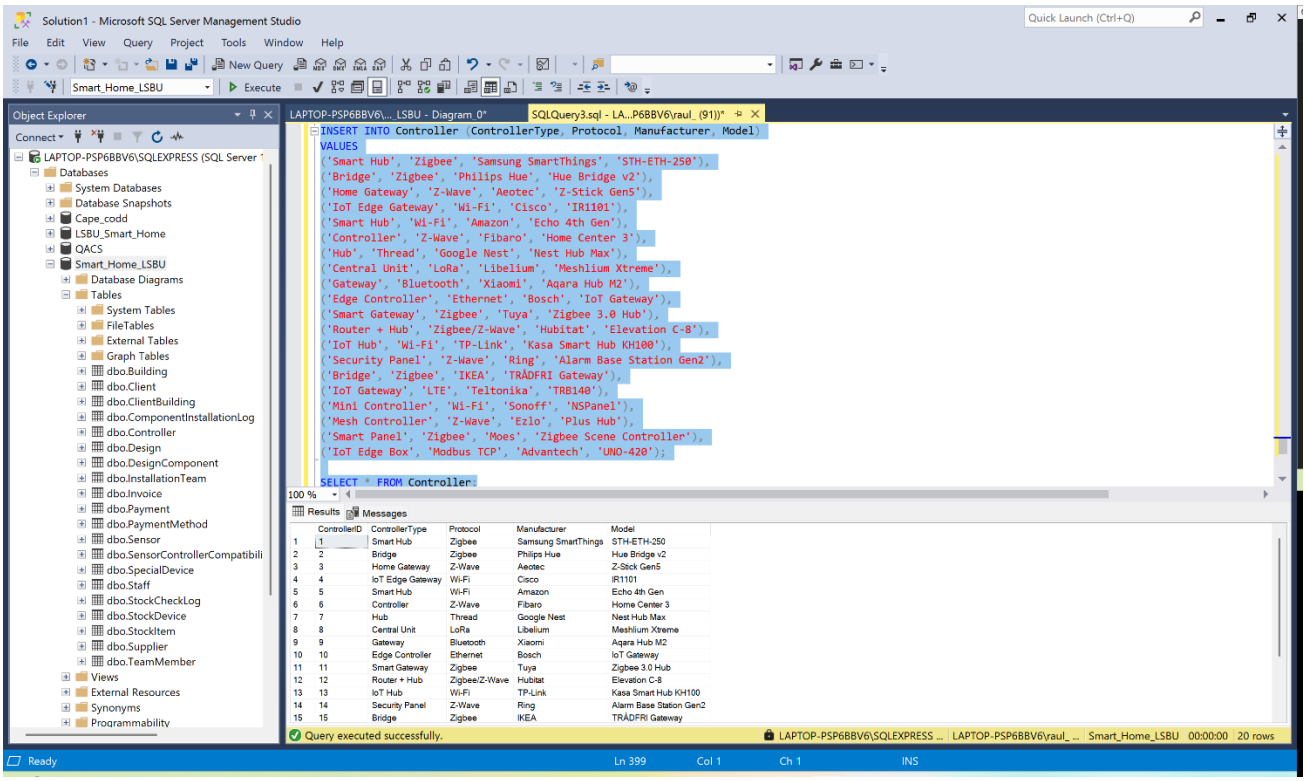

Figure 12.Insert Controller Values

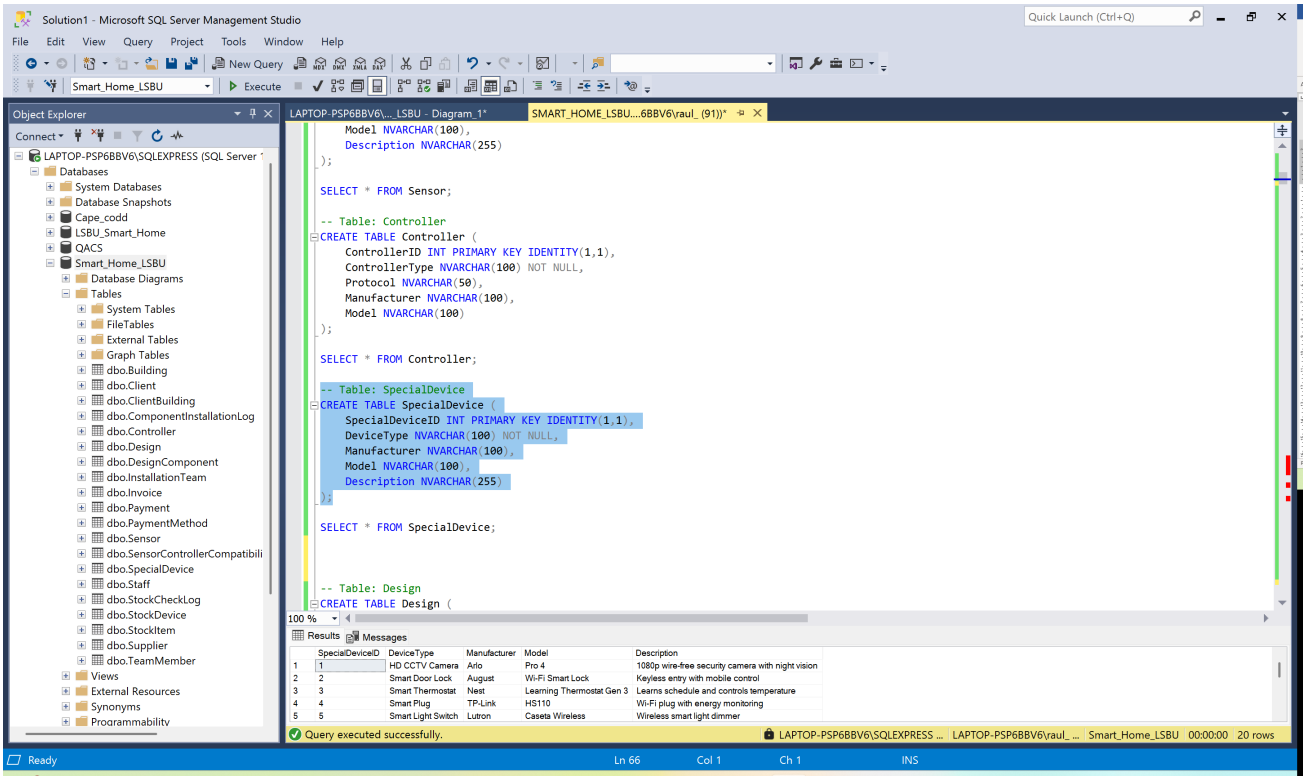

Figure 13. Create SpecialDevice

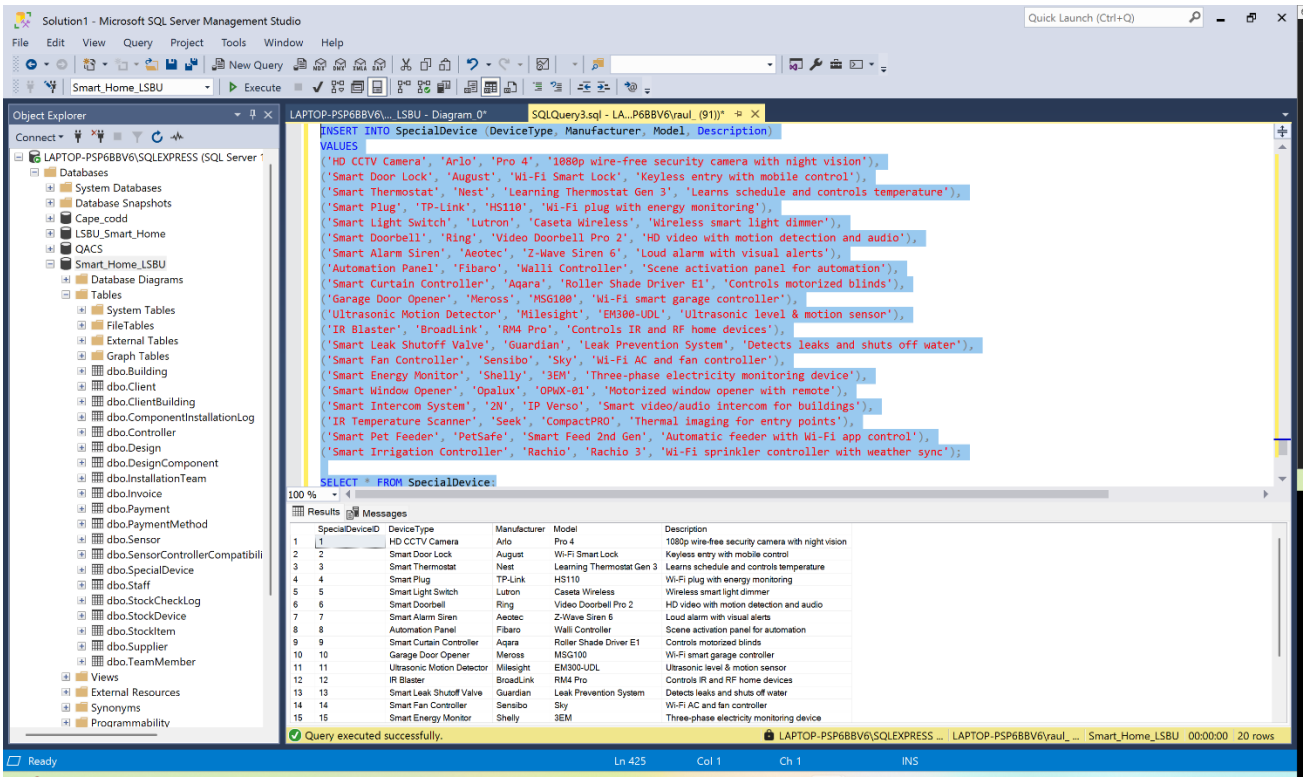

Figure 14. Insert SpecialDevice Values

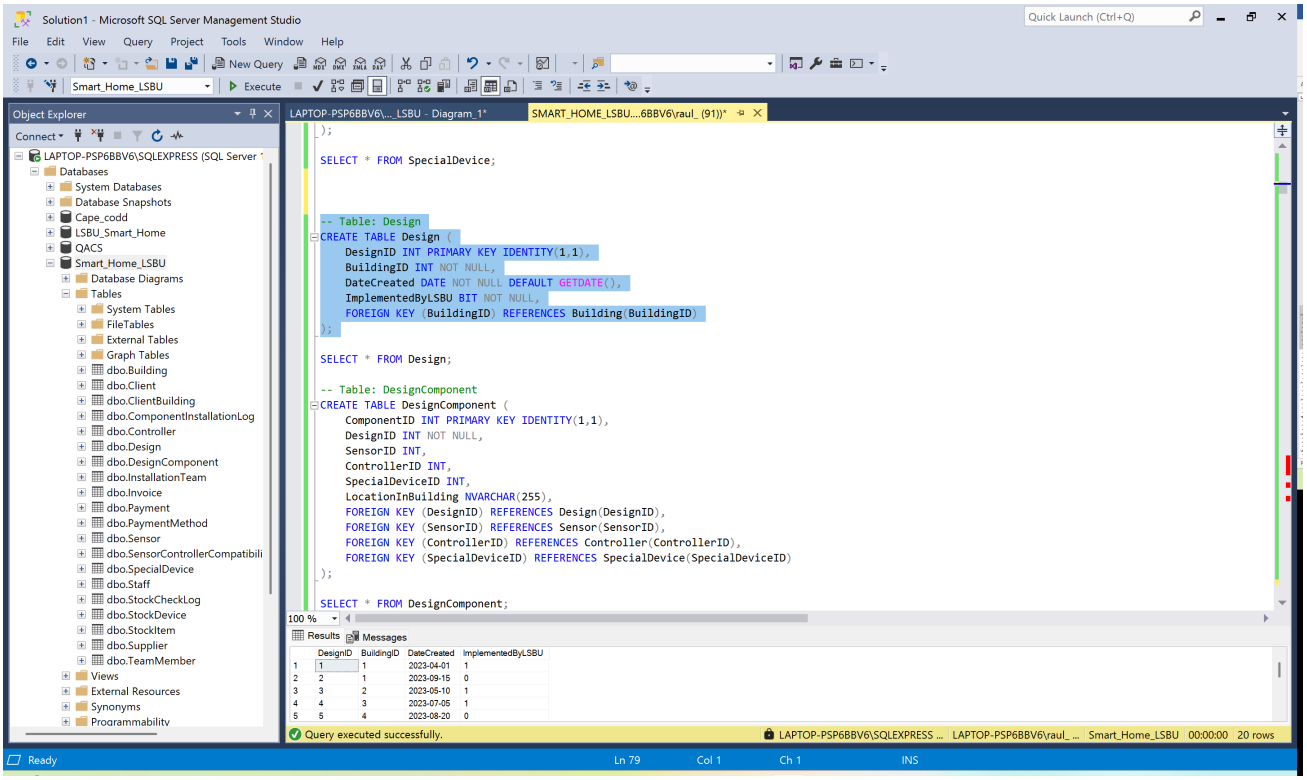

Figure 15. Create Design

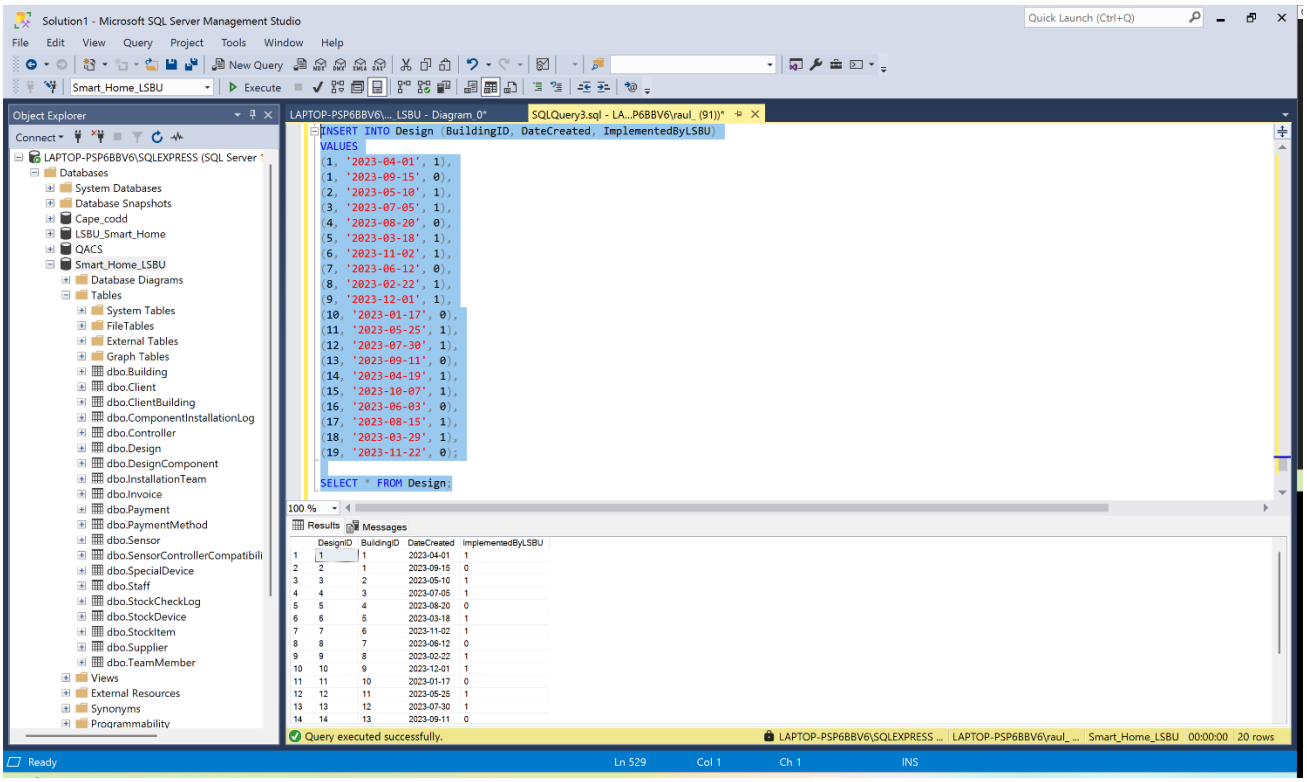

Figure 16. Insert Design Values

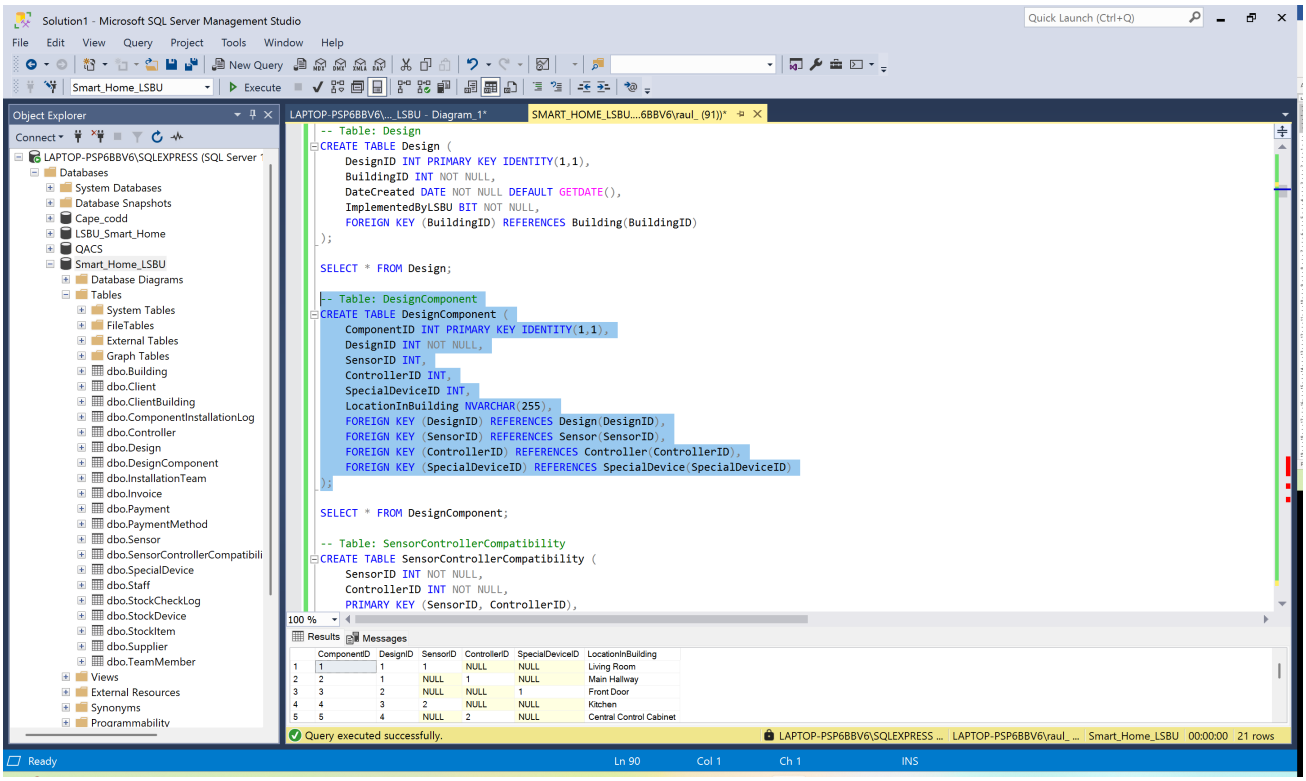

Figure 17. Create DesignComponent

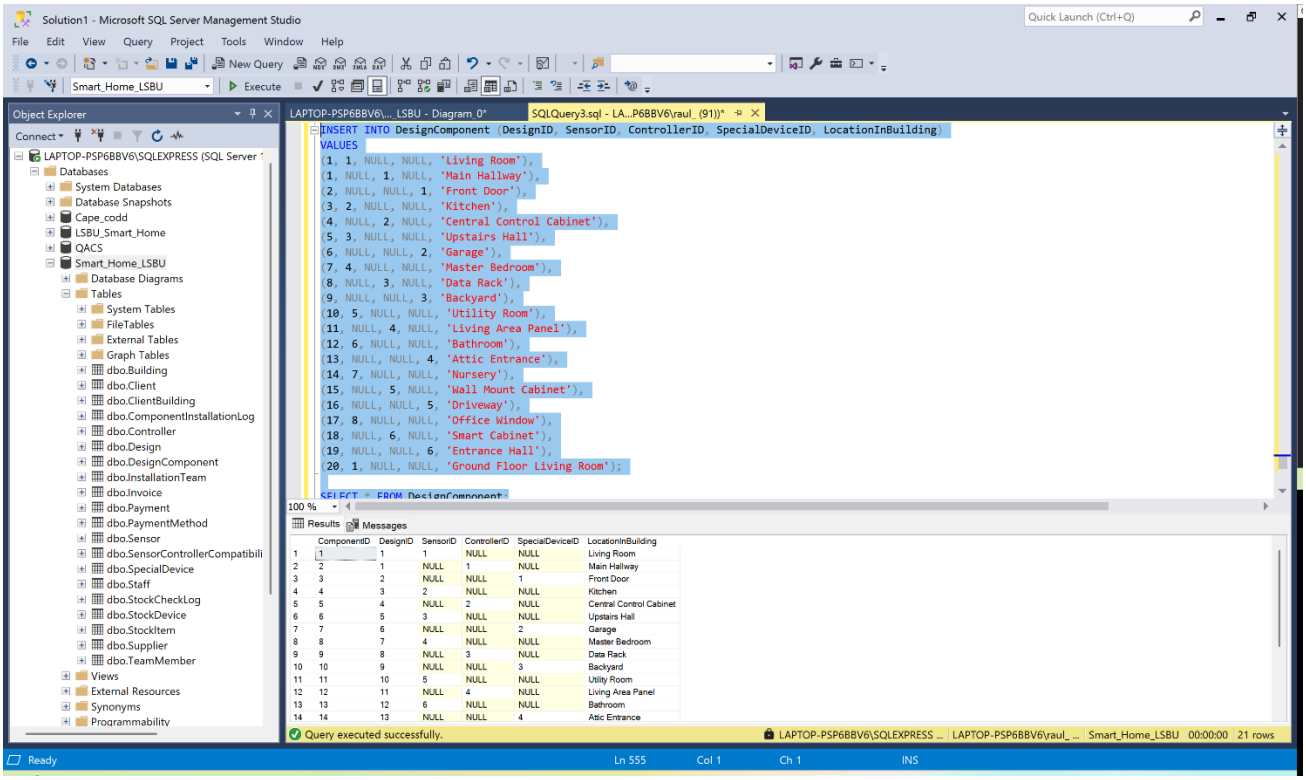

Figure 18. Insert DesignComponent Values

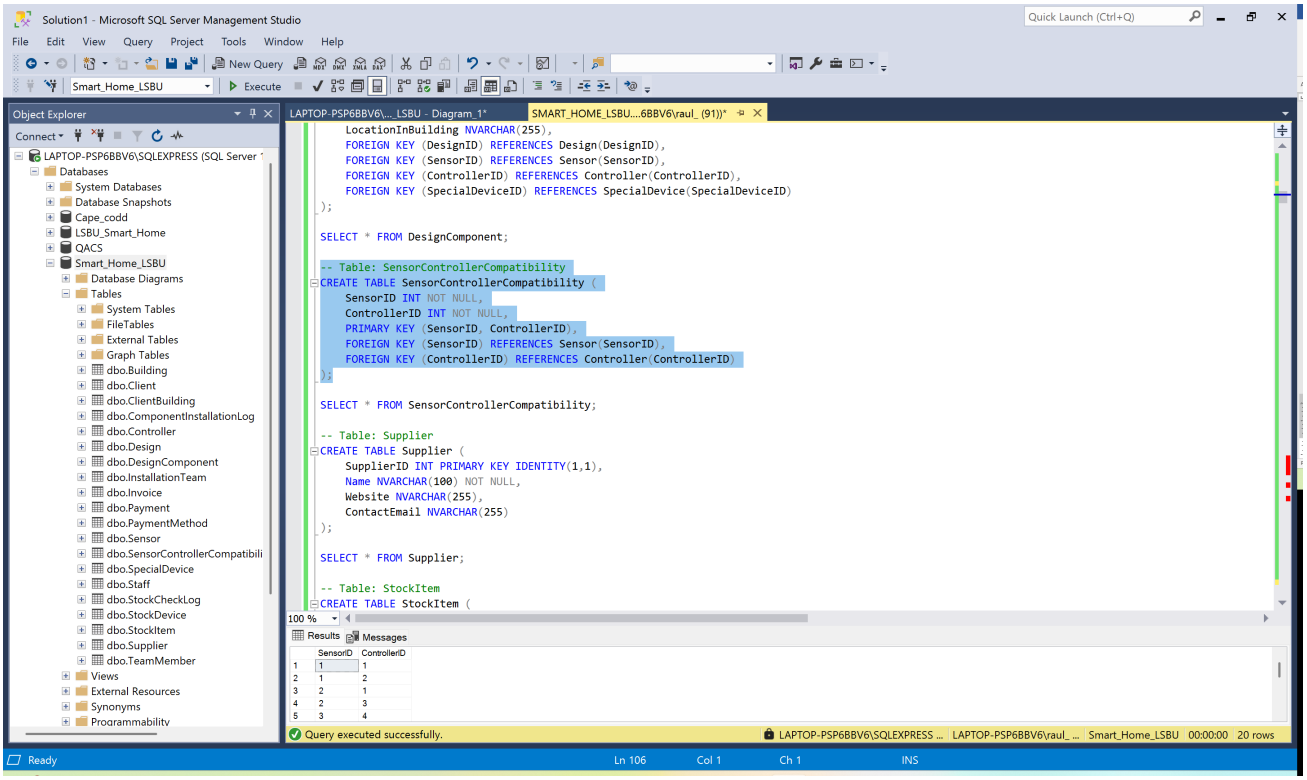

Figure 19. Create SensorControllerCompatibility

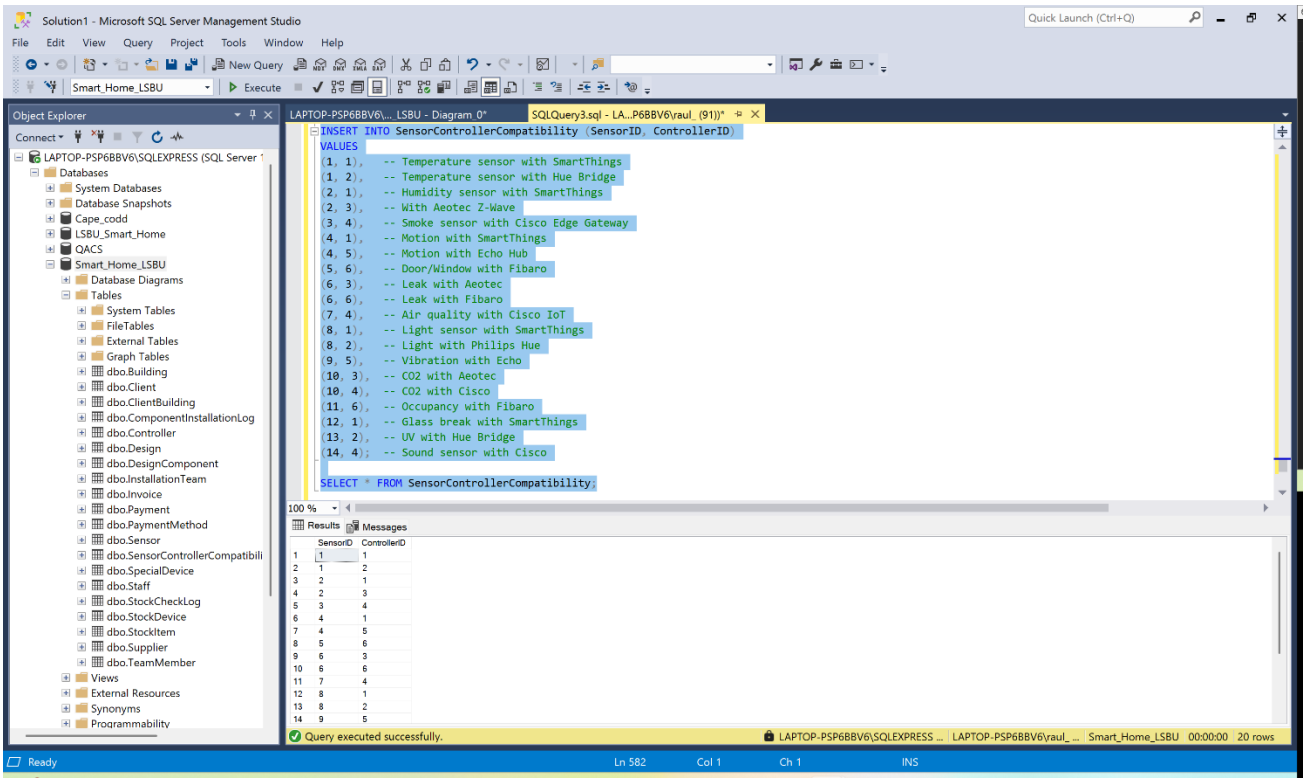

Figure 20. Insert SensorControllerCompatibility Values

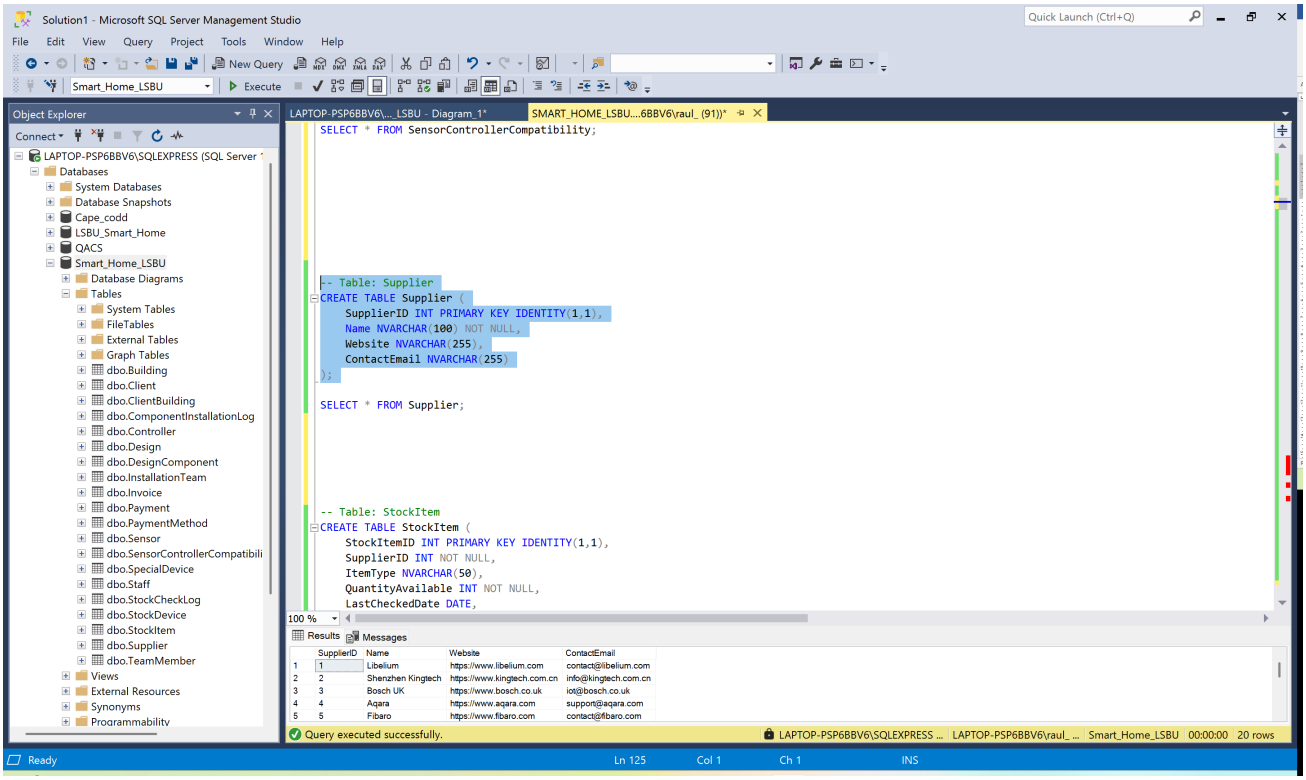

Figure 21. Create Supplier

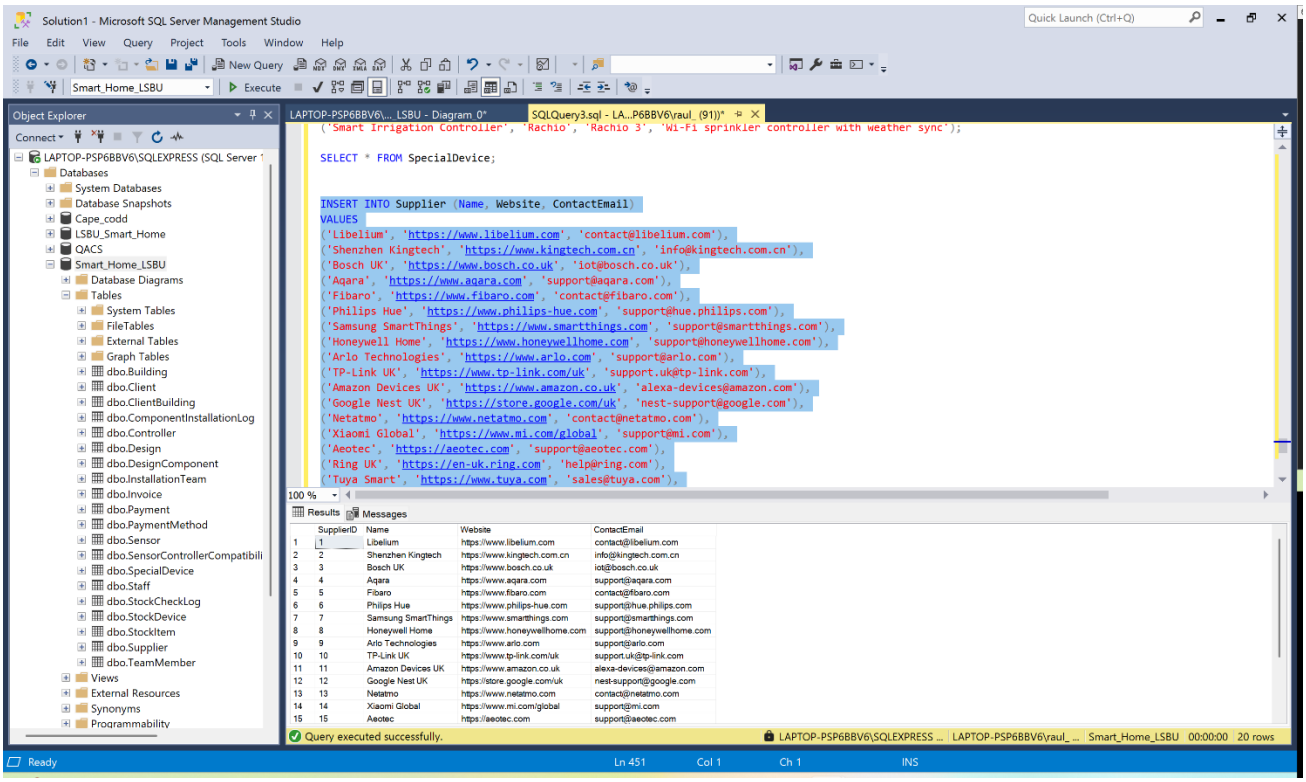

Figure 22. Insert Supplier Values

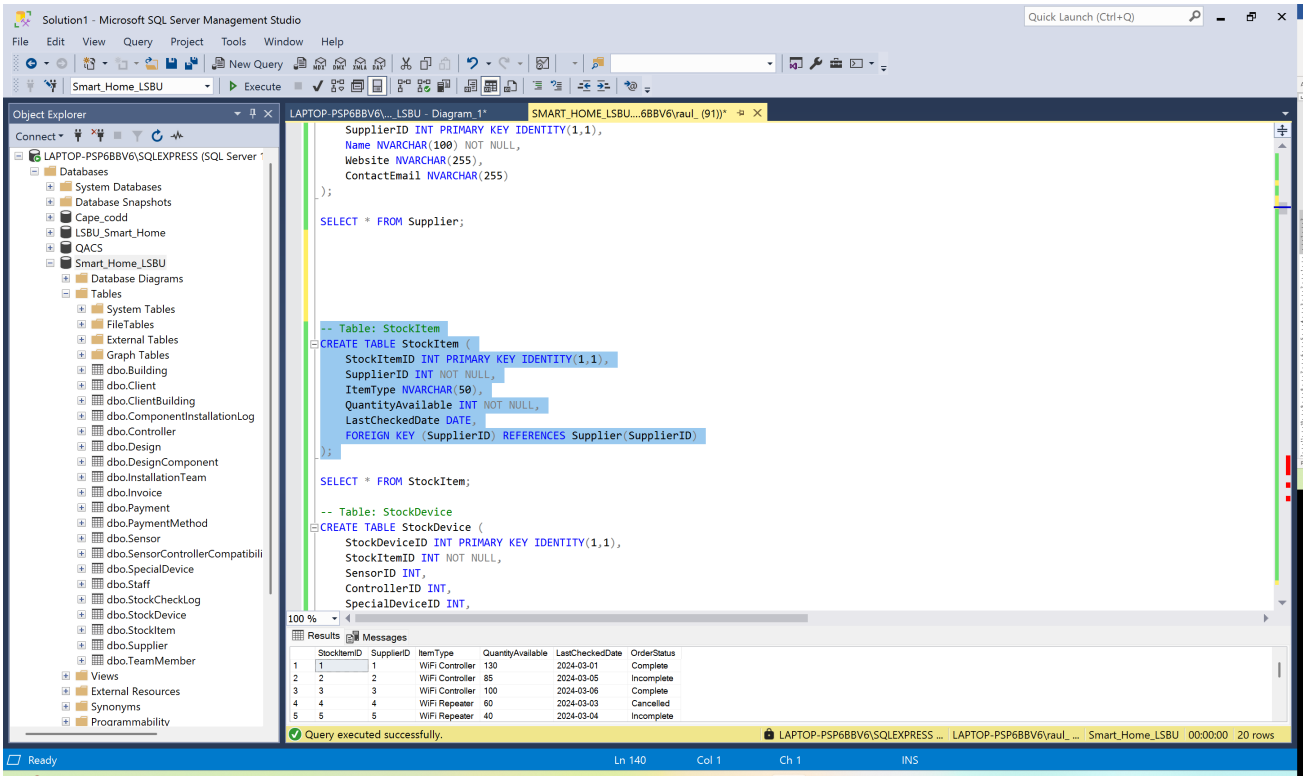

Figure 23. Create StockItem

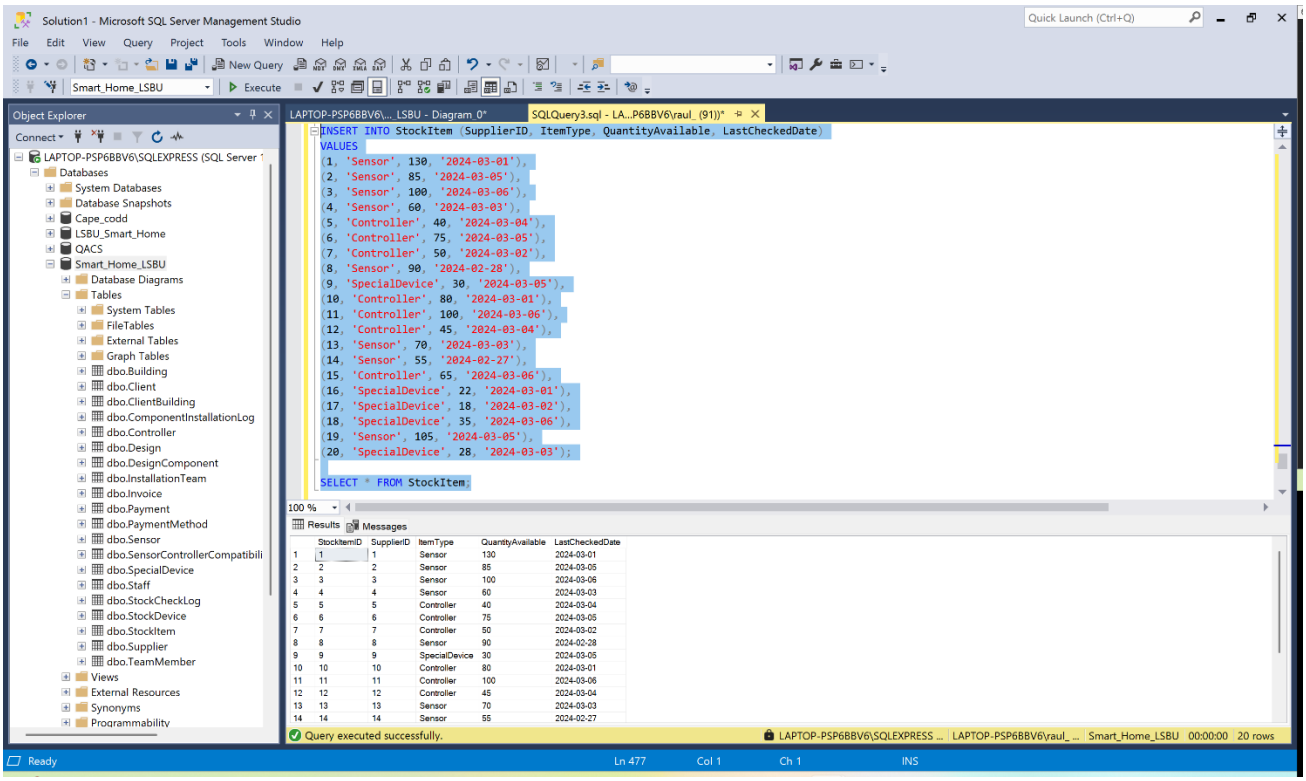

Figure 24. Insert Stockitem Values

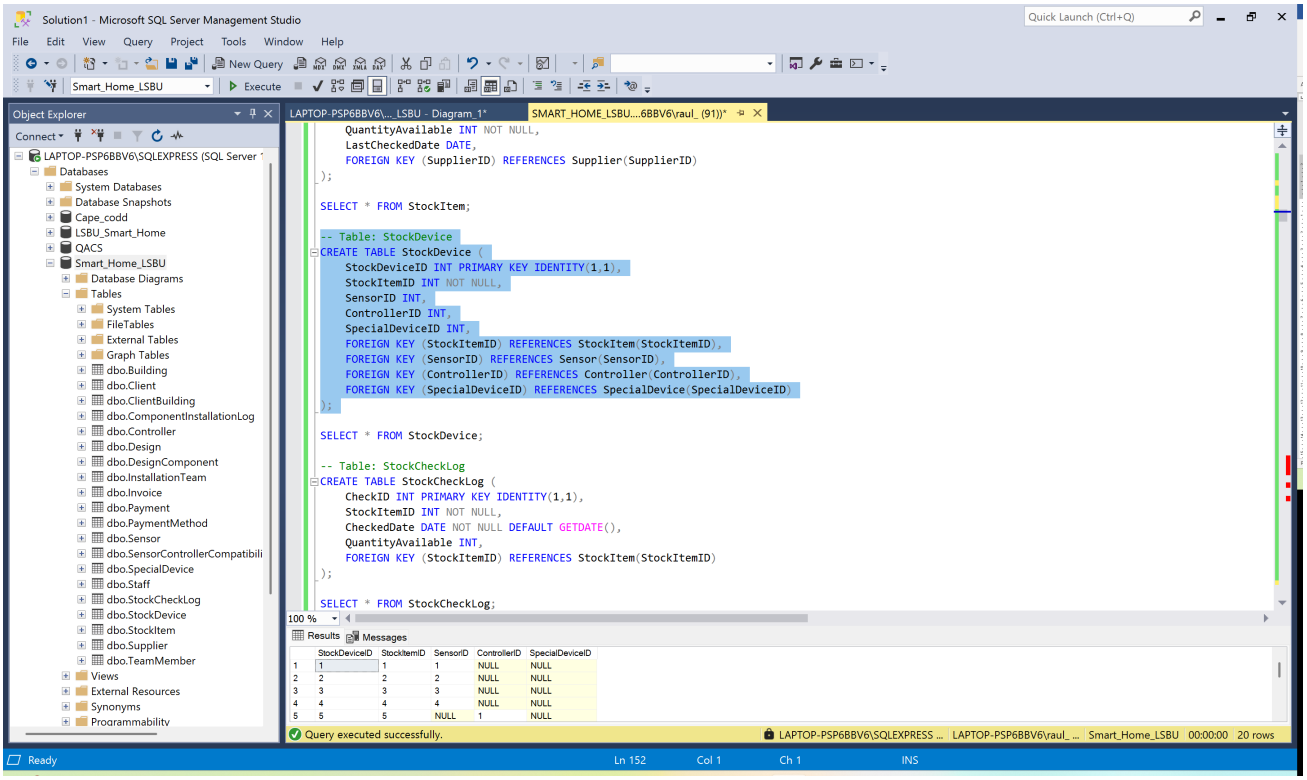

Figure 25. Create StockDevice

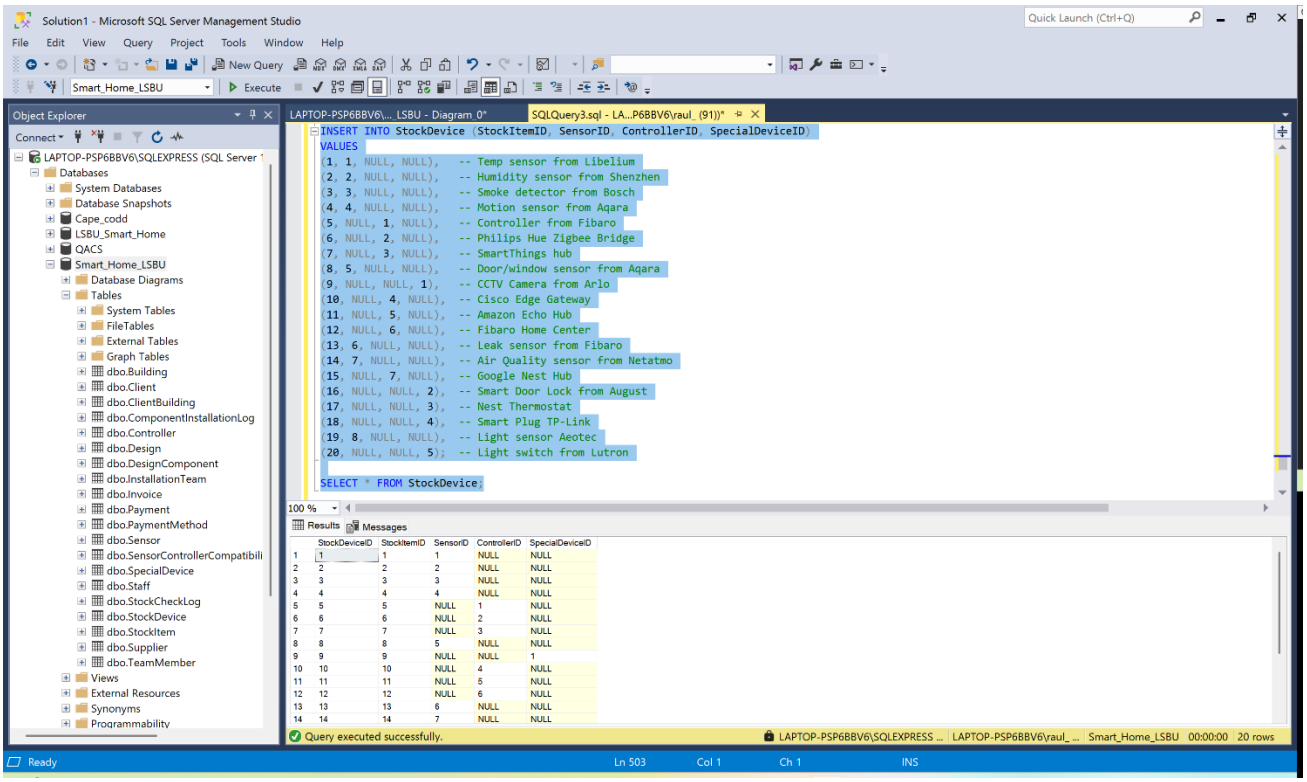

Figure 26. Insert StockDevice Values

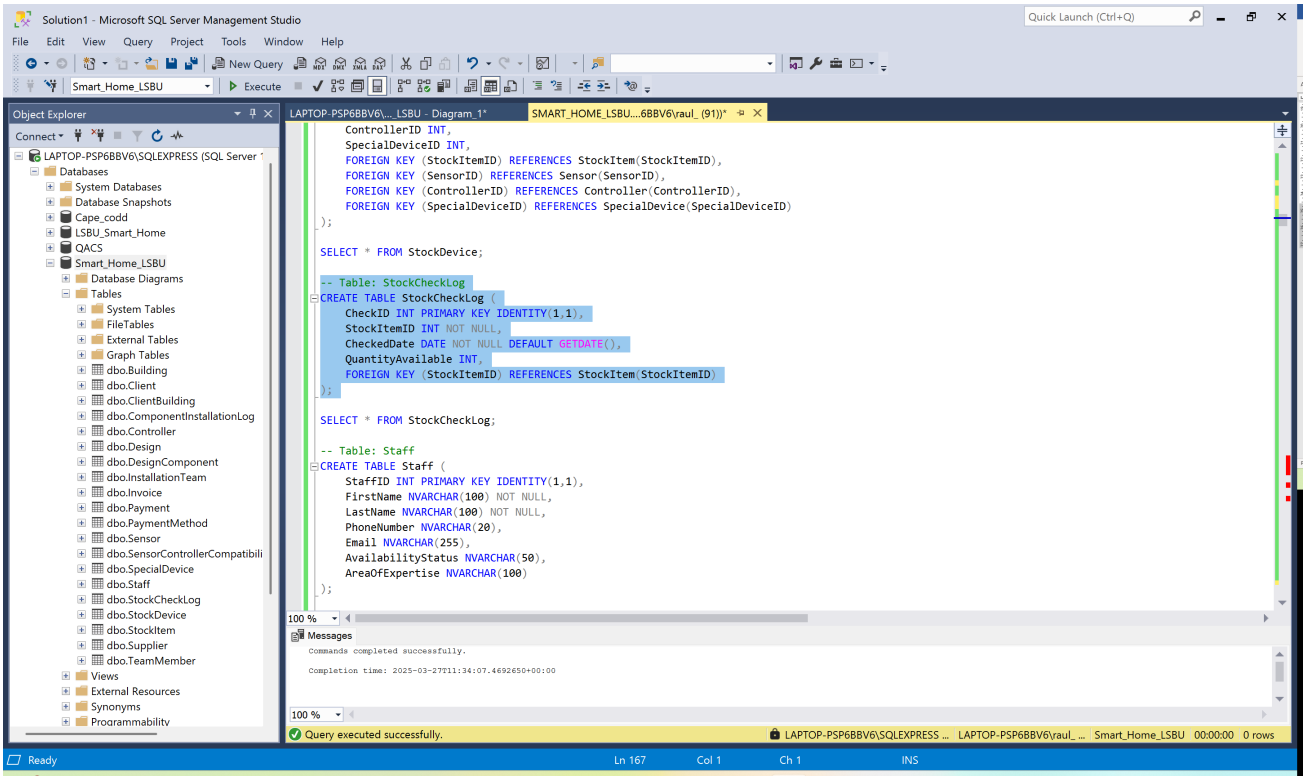

Figure 27. Create StockCheckLog

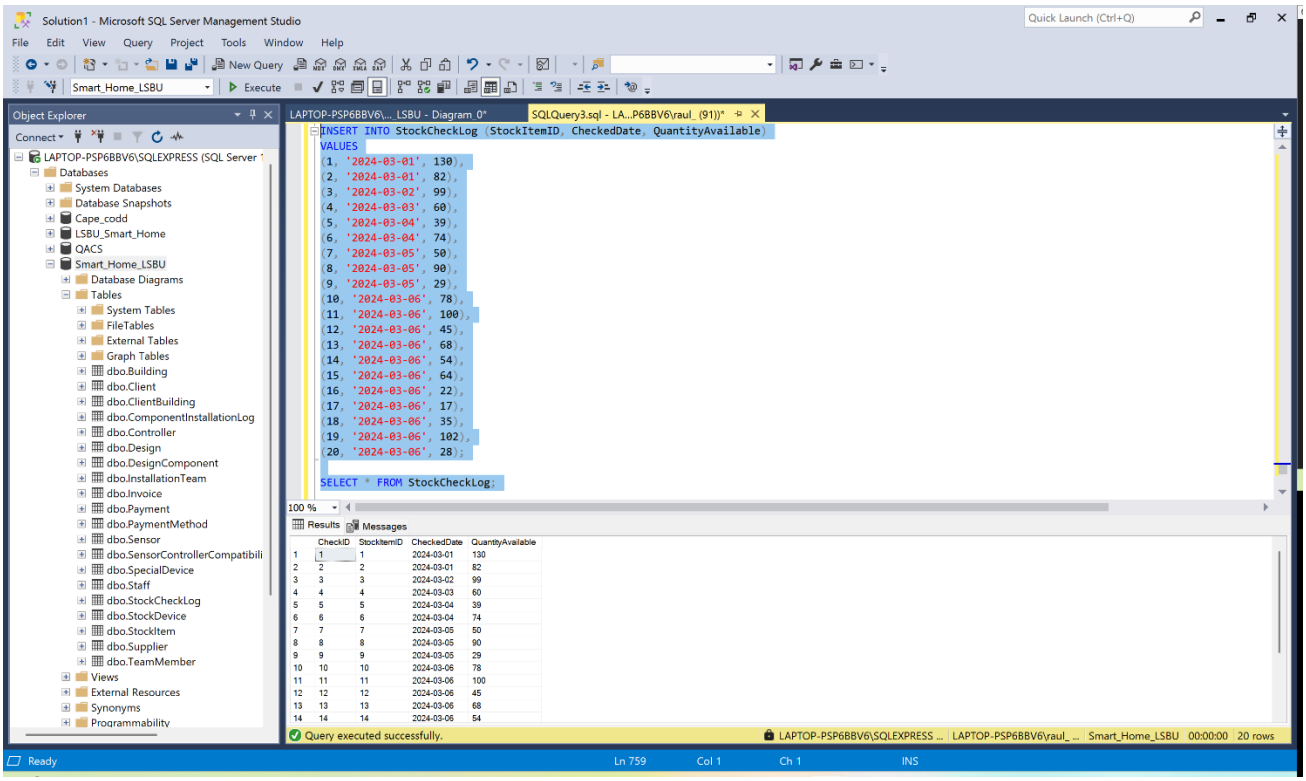

Figure 28. Insert StockCheckLog Values

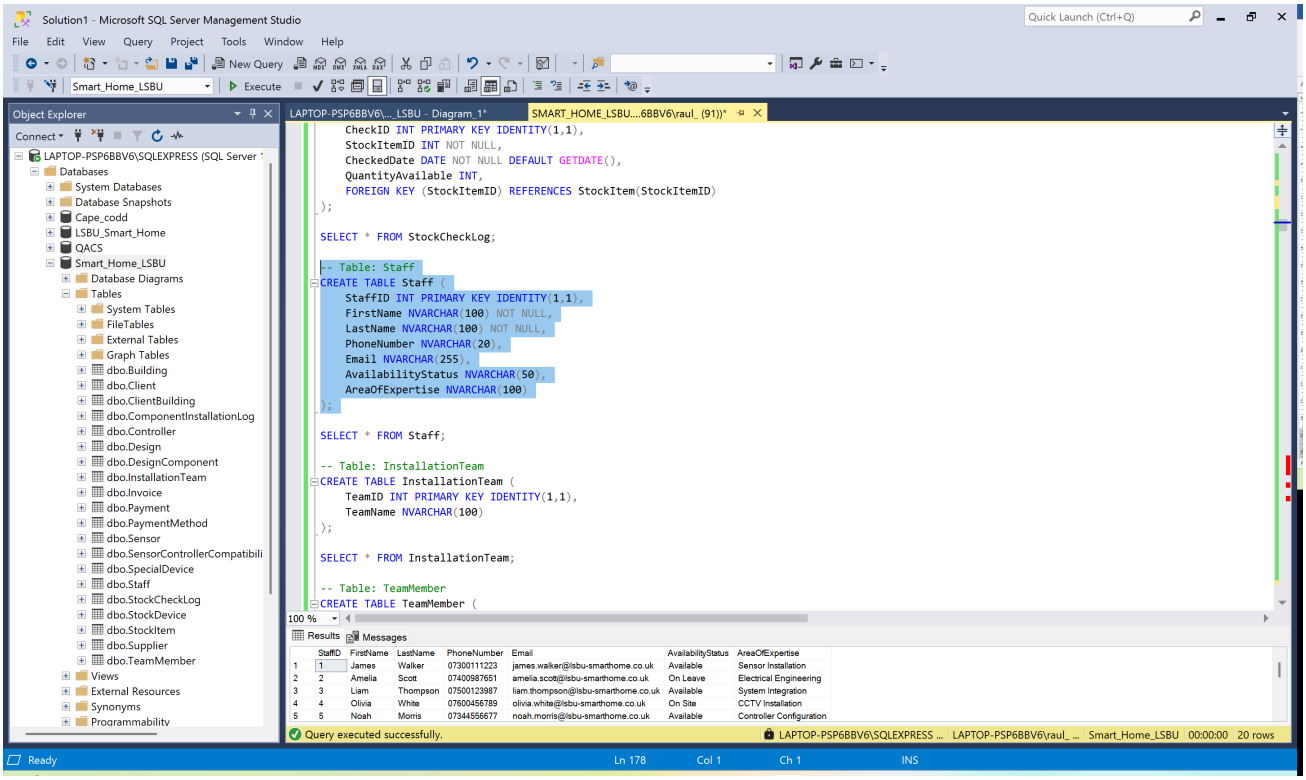

Figure 29. Create Staff

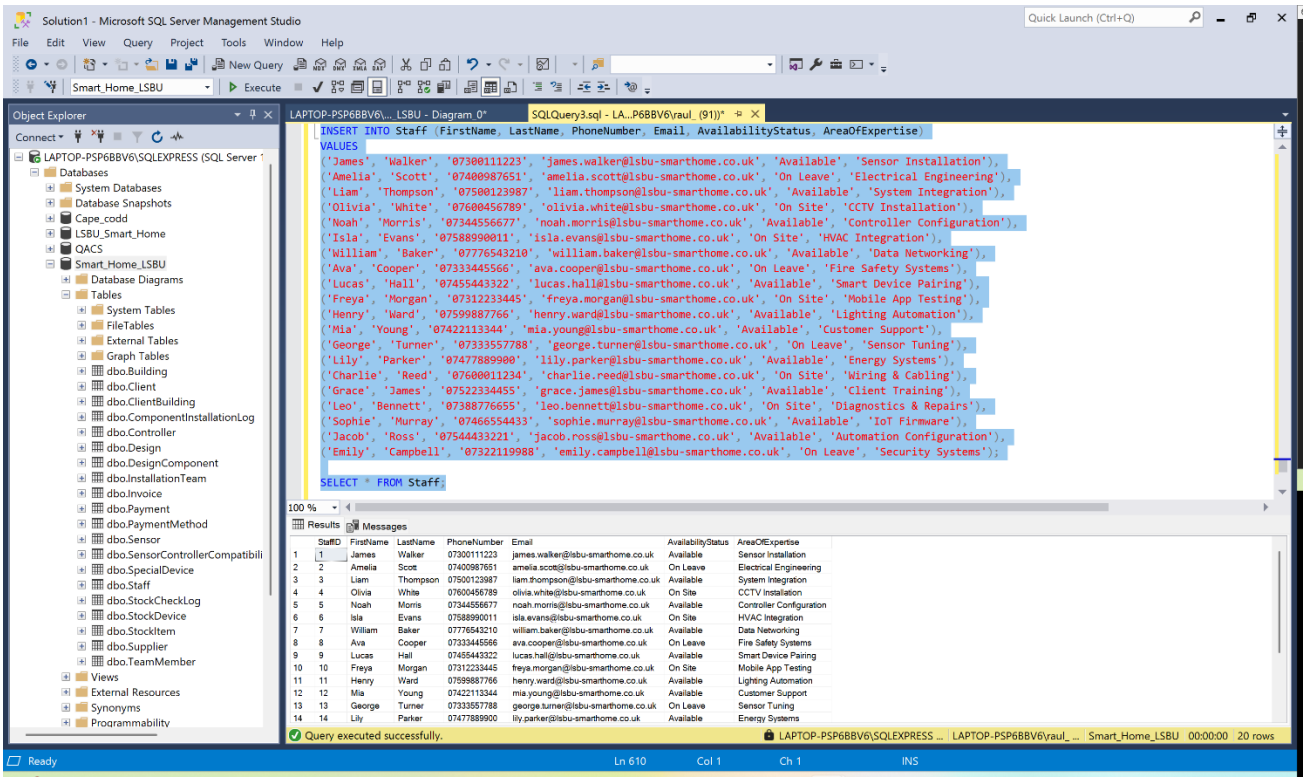

Figure 30. Insert Staff Values

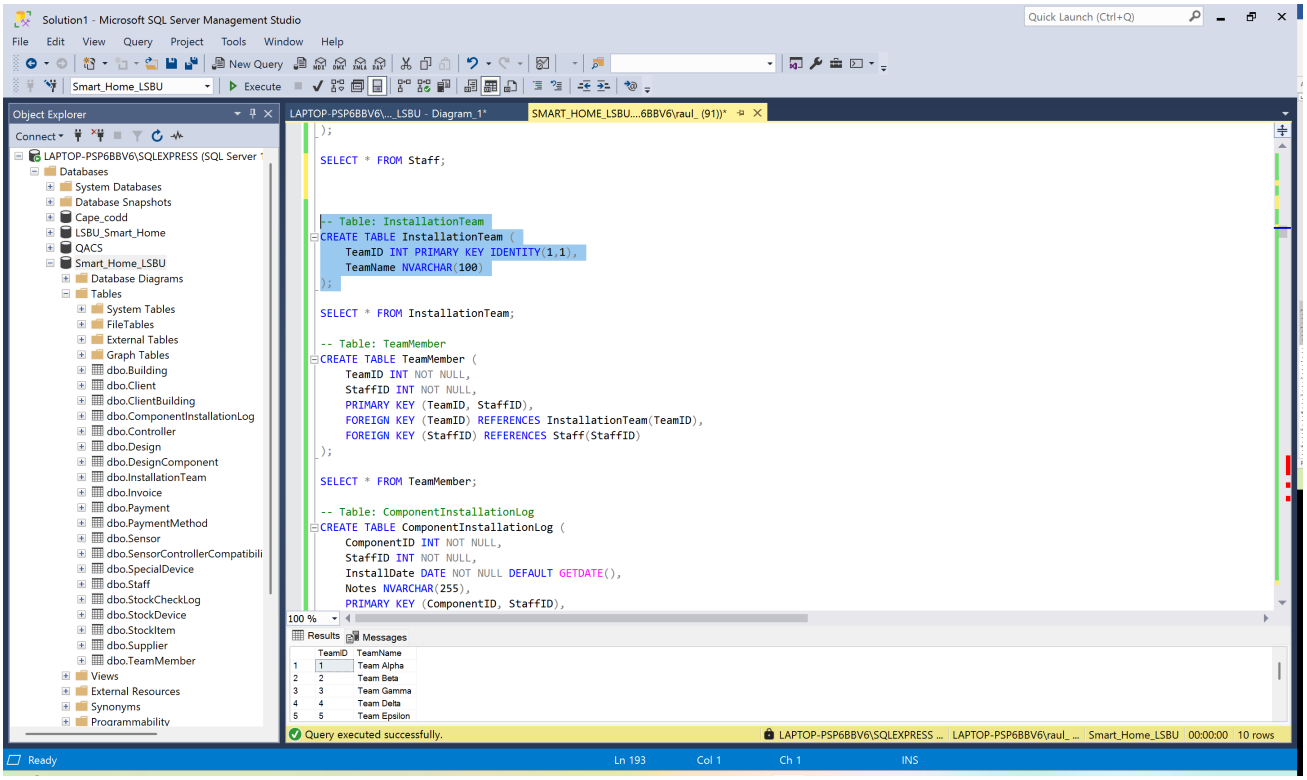

Figure 31. Create InstalationTeam

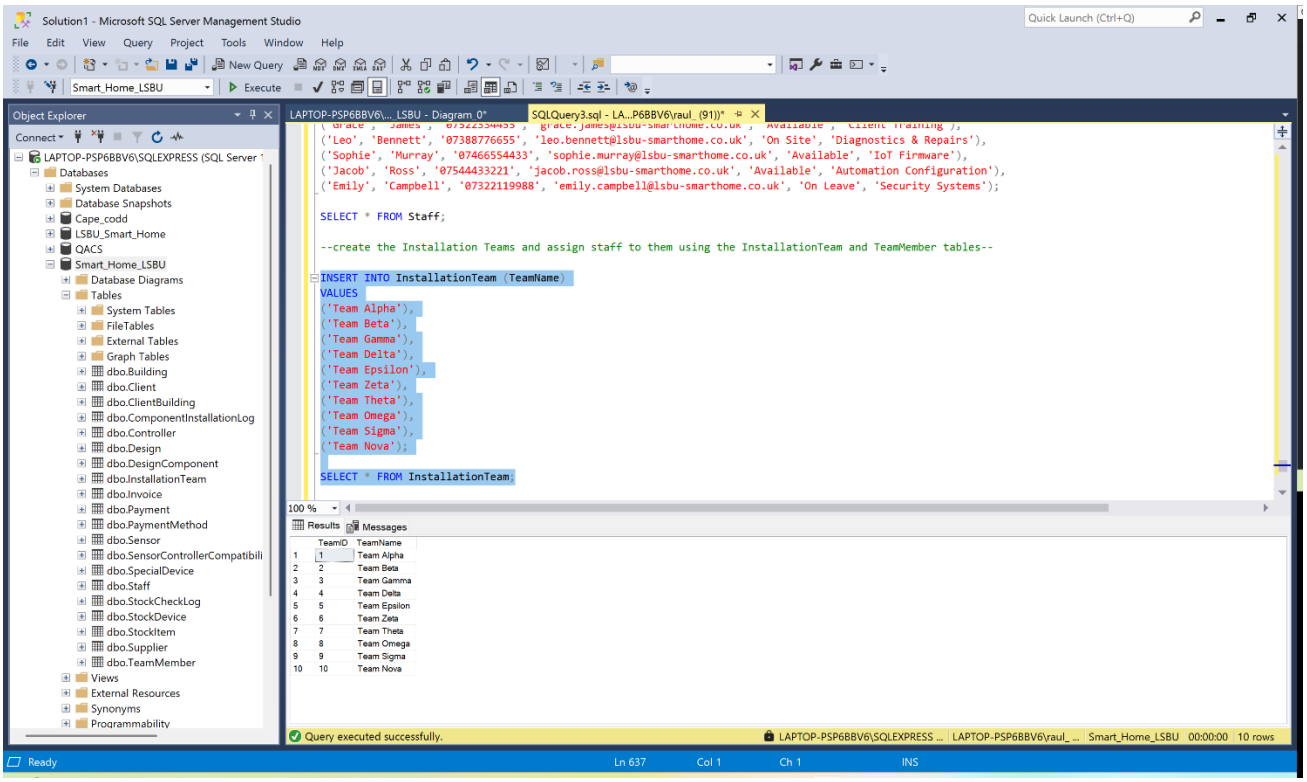

Figure 32. Insert InstalationTeam Values

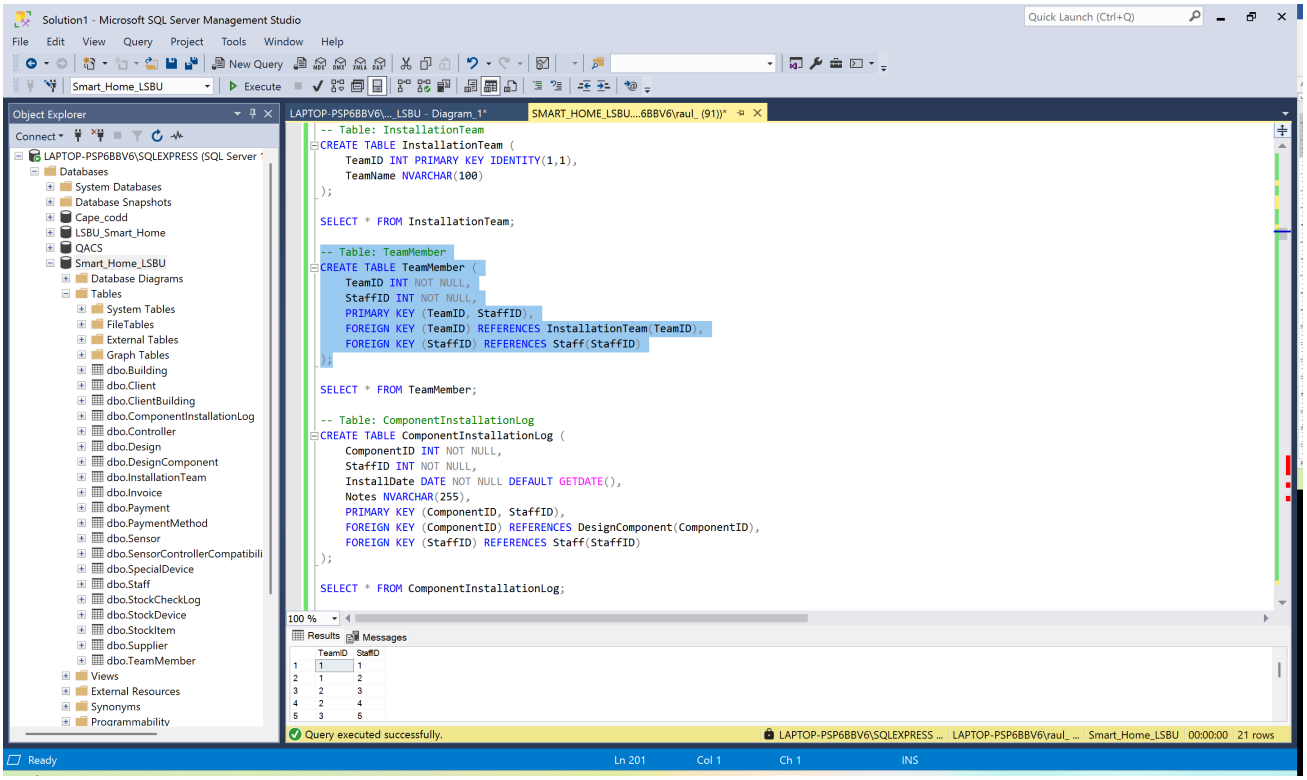

Figure 33. Create TeamMember

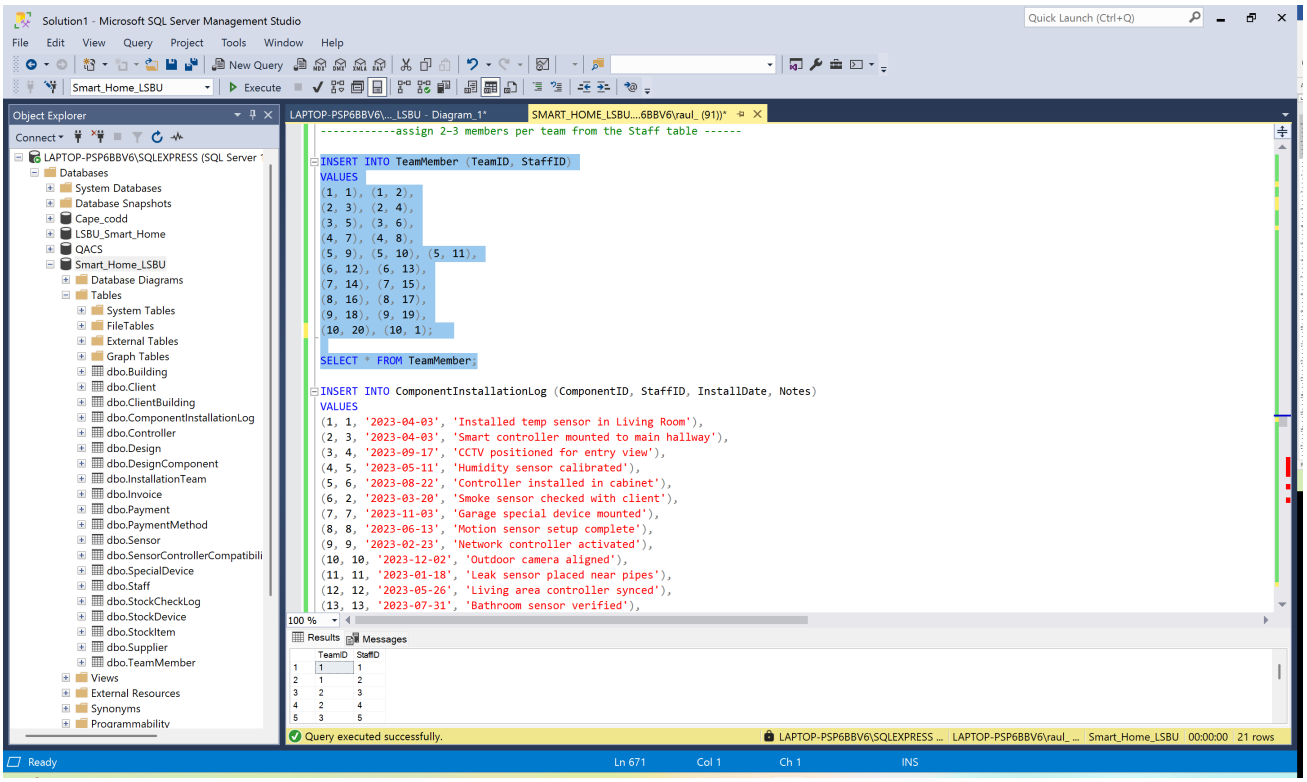

Figure 34. Insert TeamMember Values

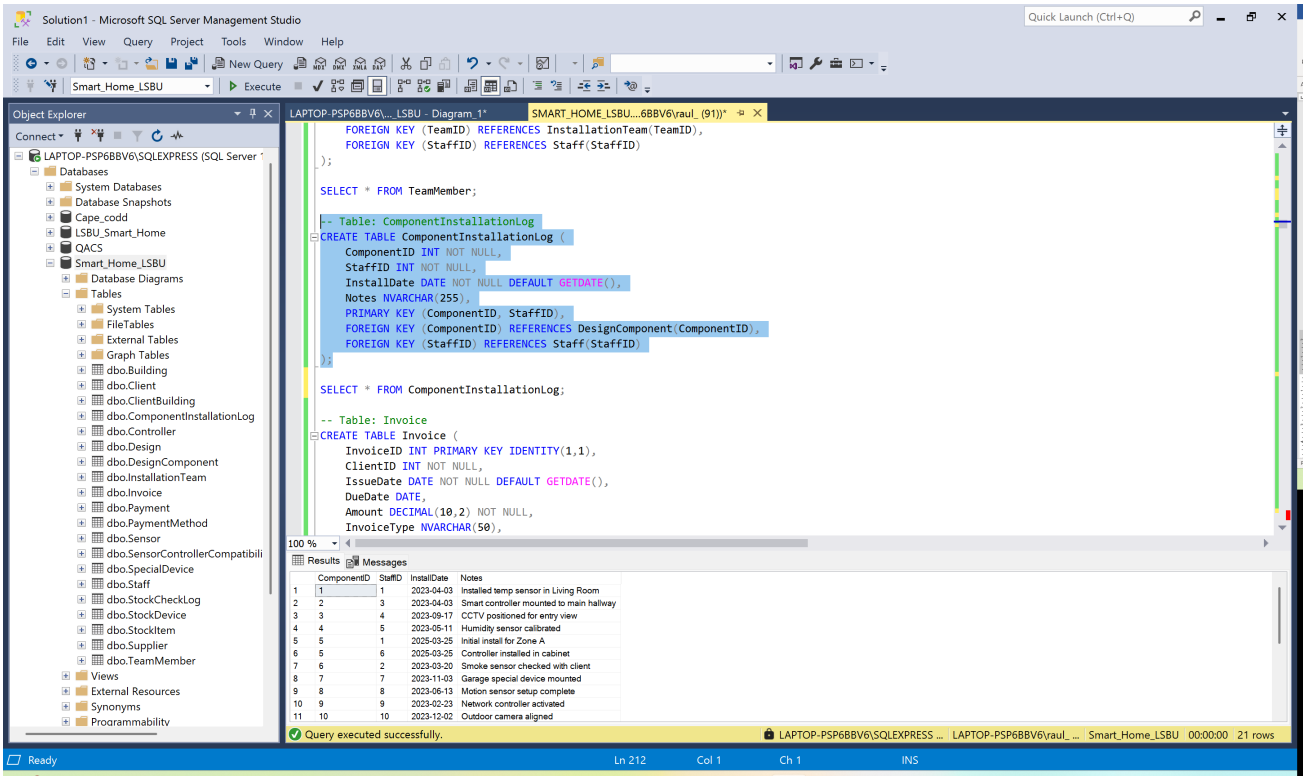

Figure 35. Create ComponentInstallationLog

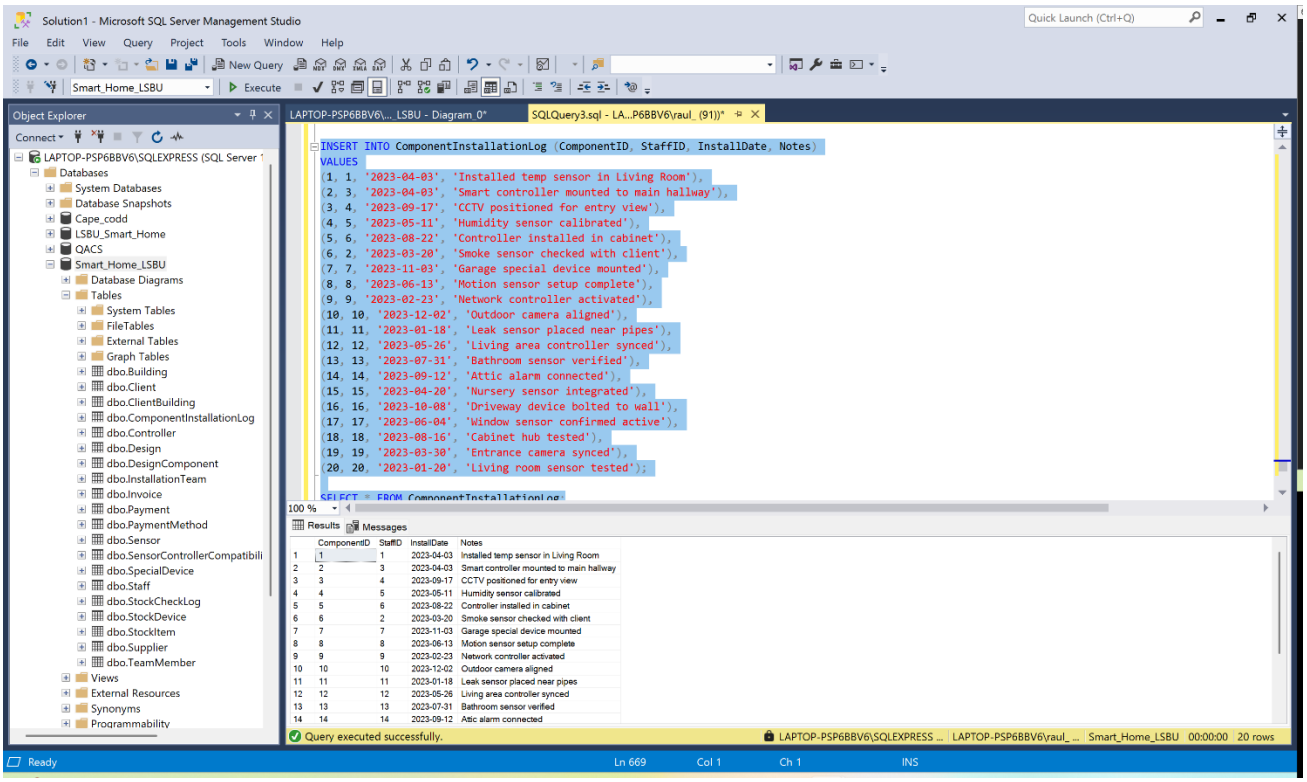

Figure 36. Insert ComponentInstallationLog Values

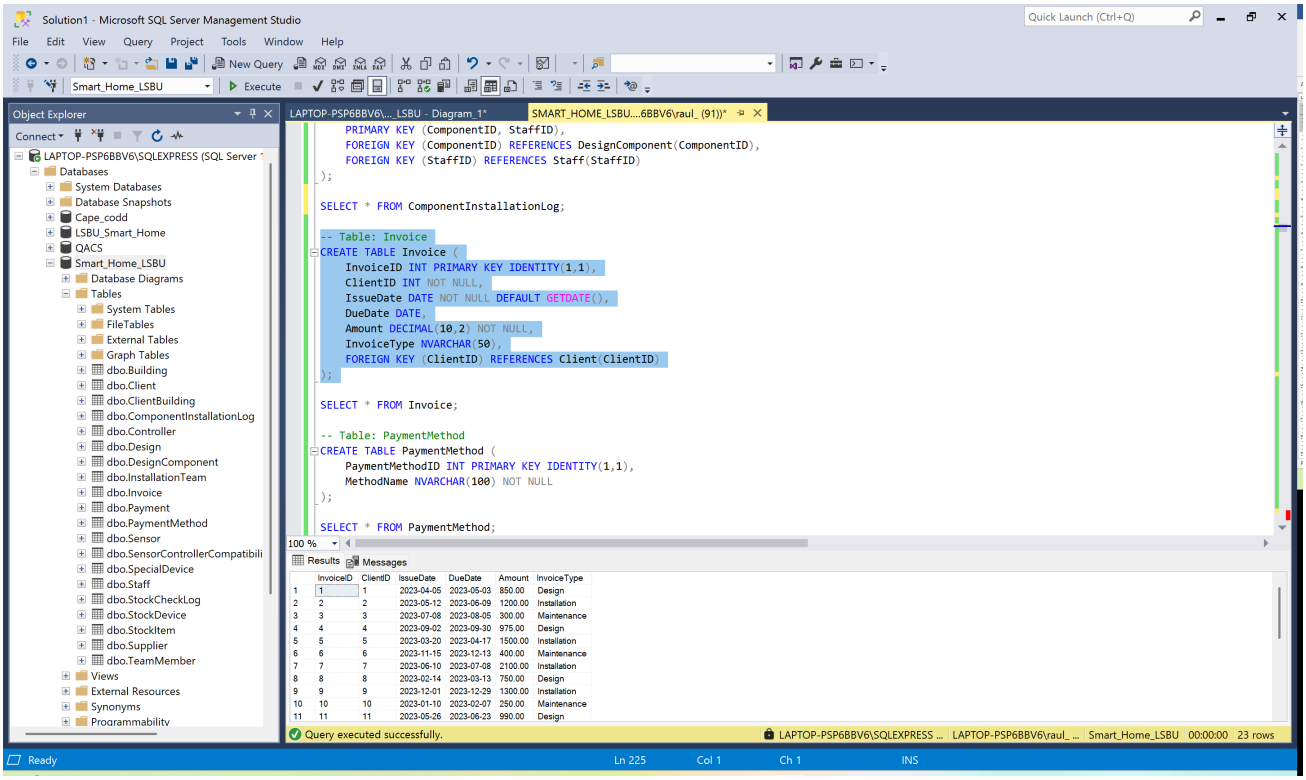

Figure 37. create Invoice

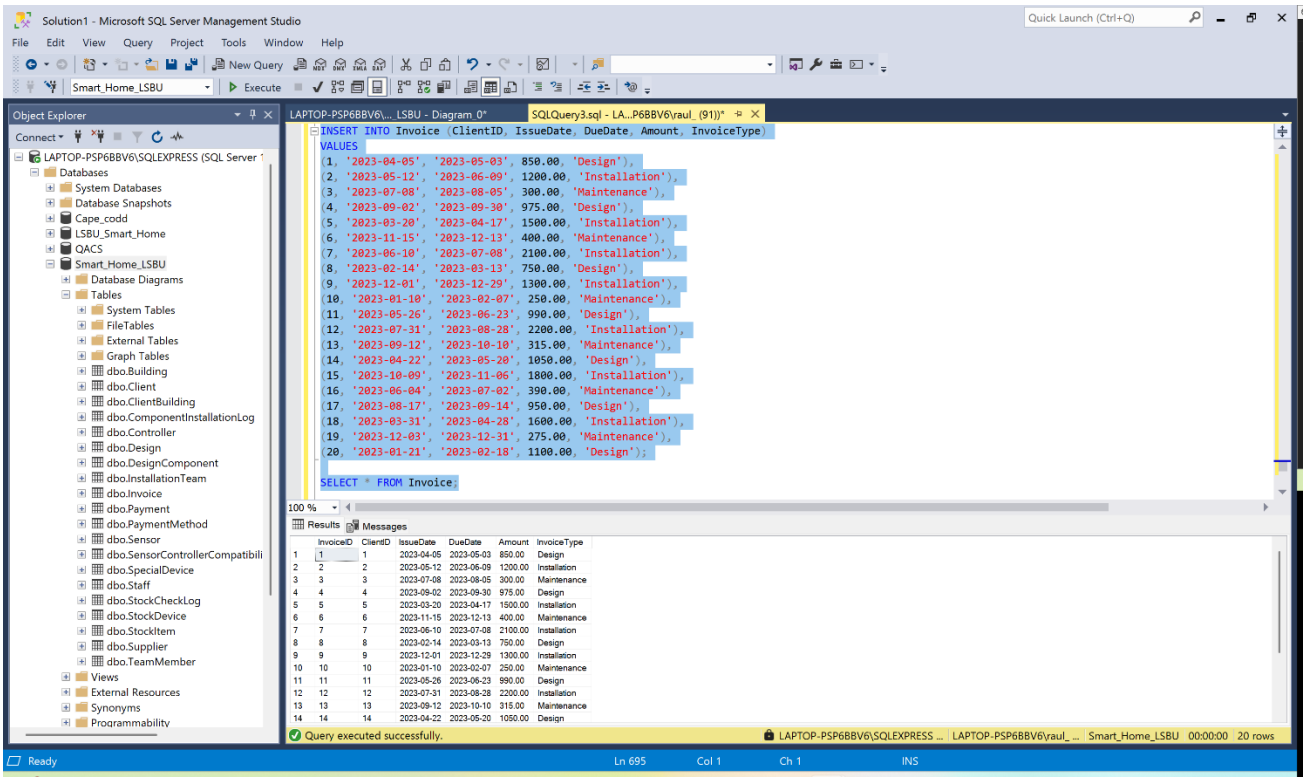

Figure 38. Insert Invoice Values

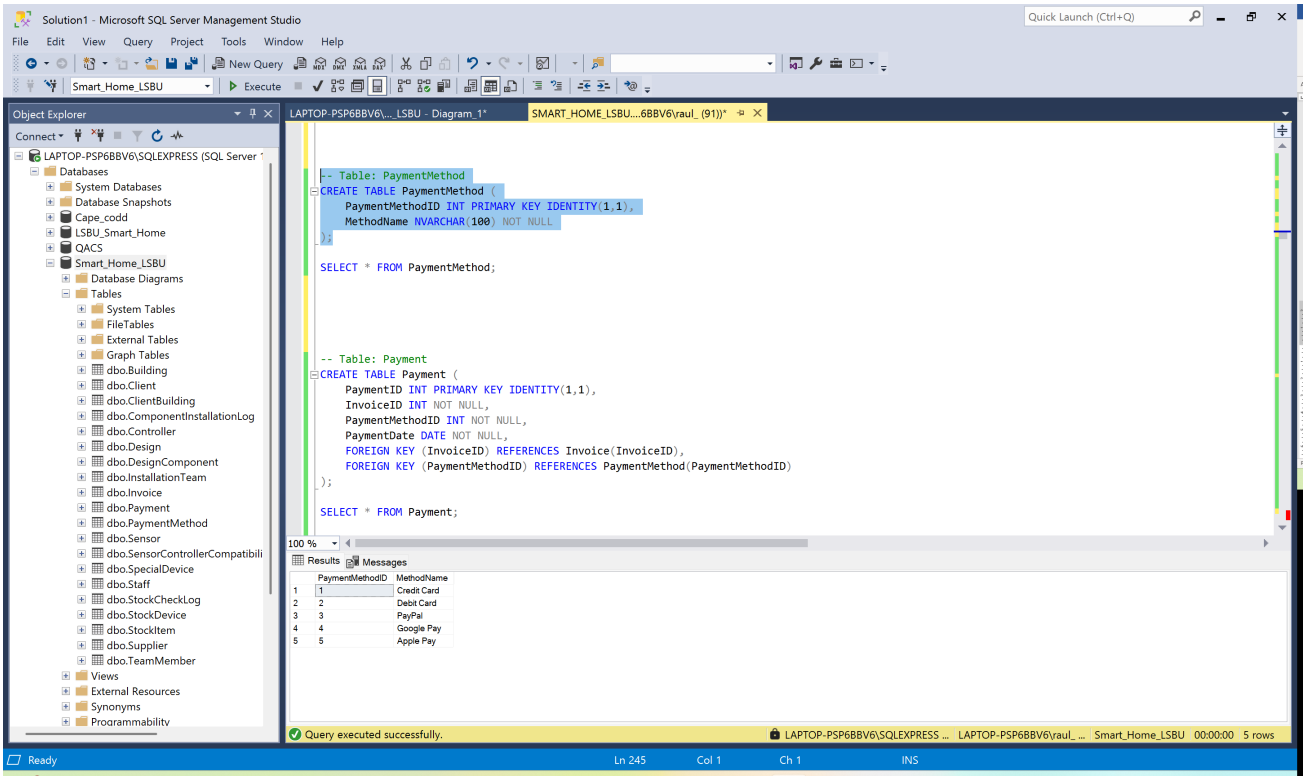

Figure 39. Create PaymentMethod

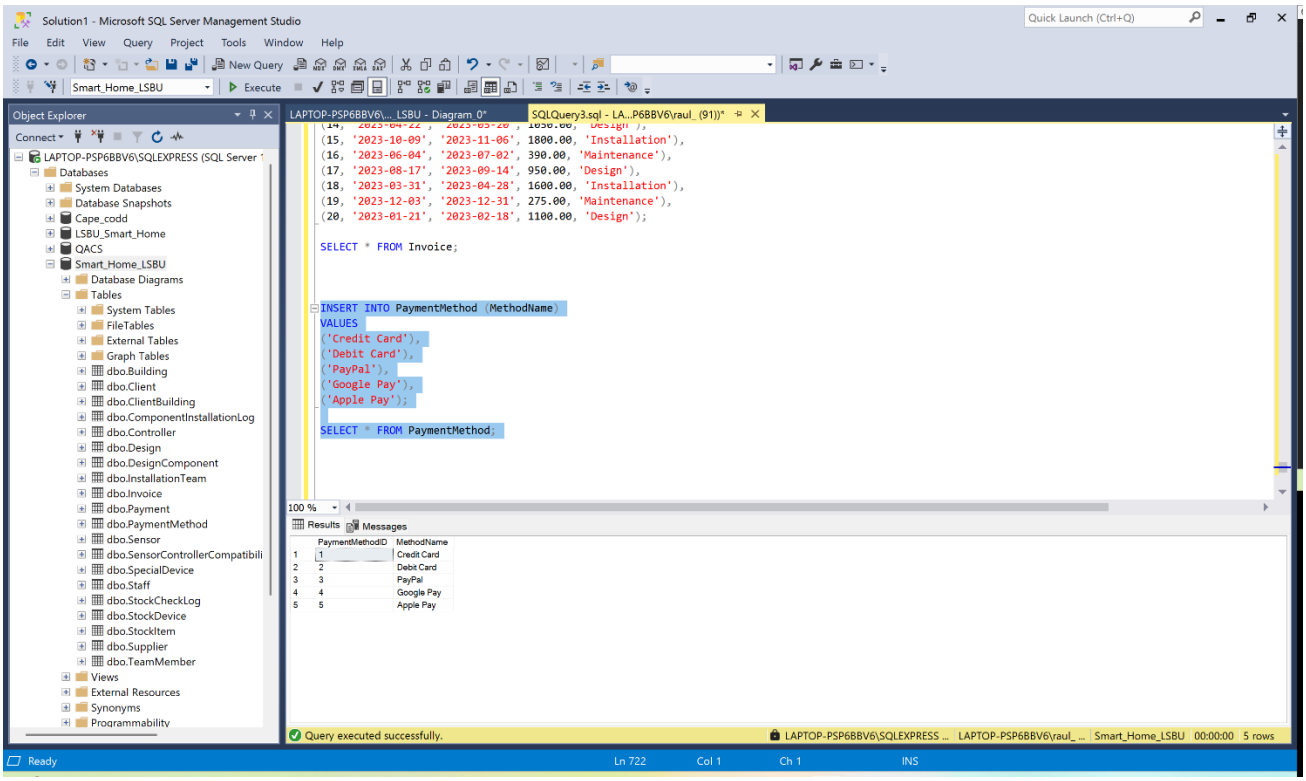

Figure 40. Insert PaymentMethod Values

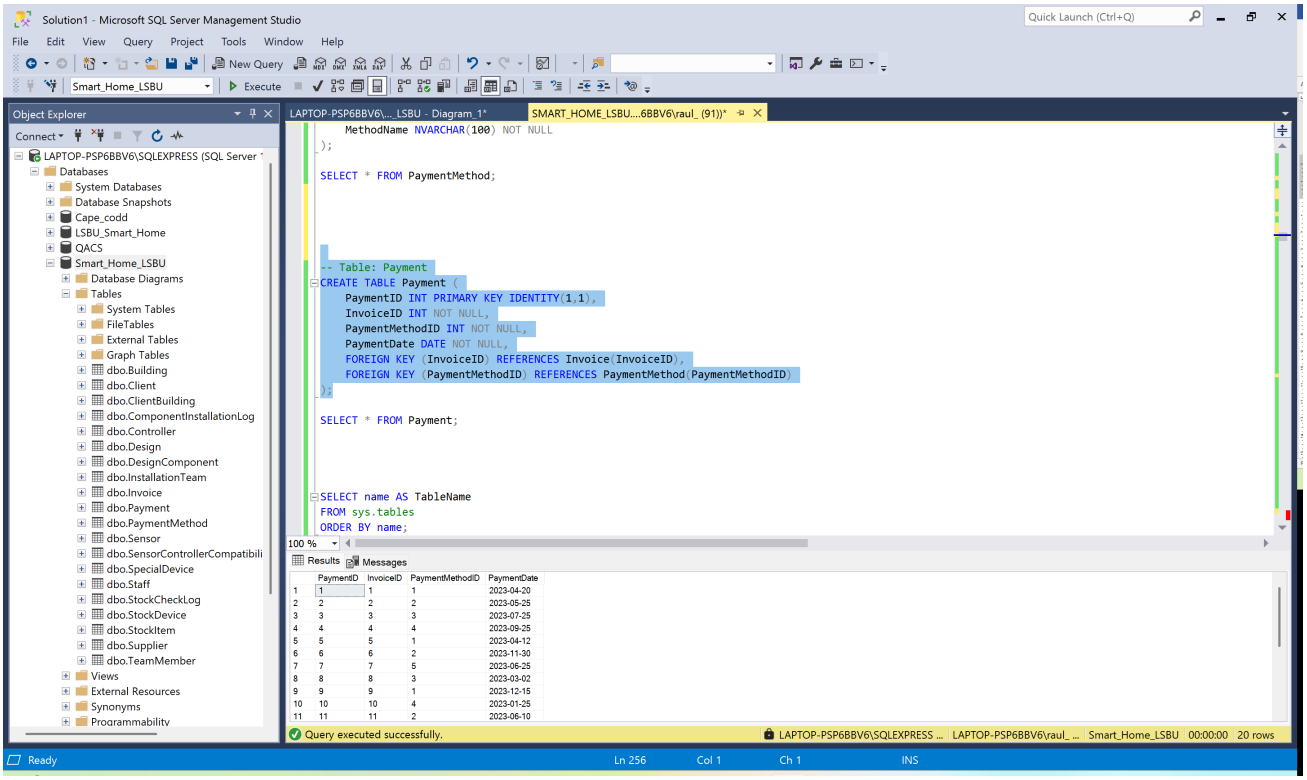

Figure 41. Create Payment

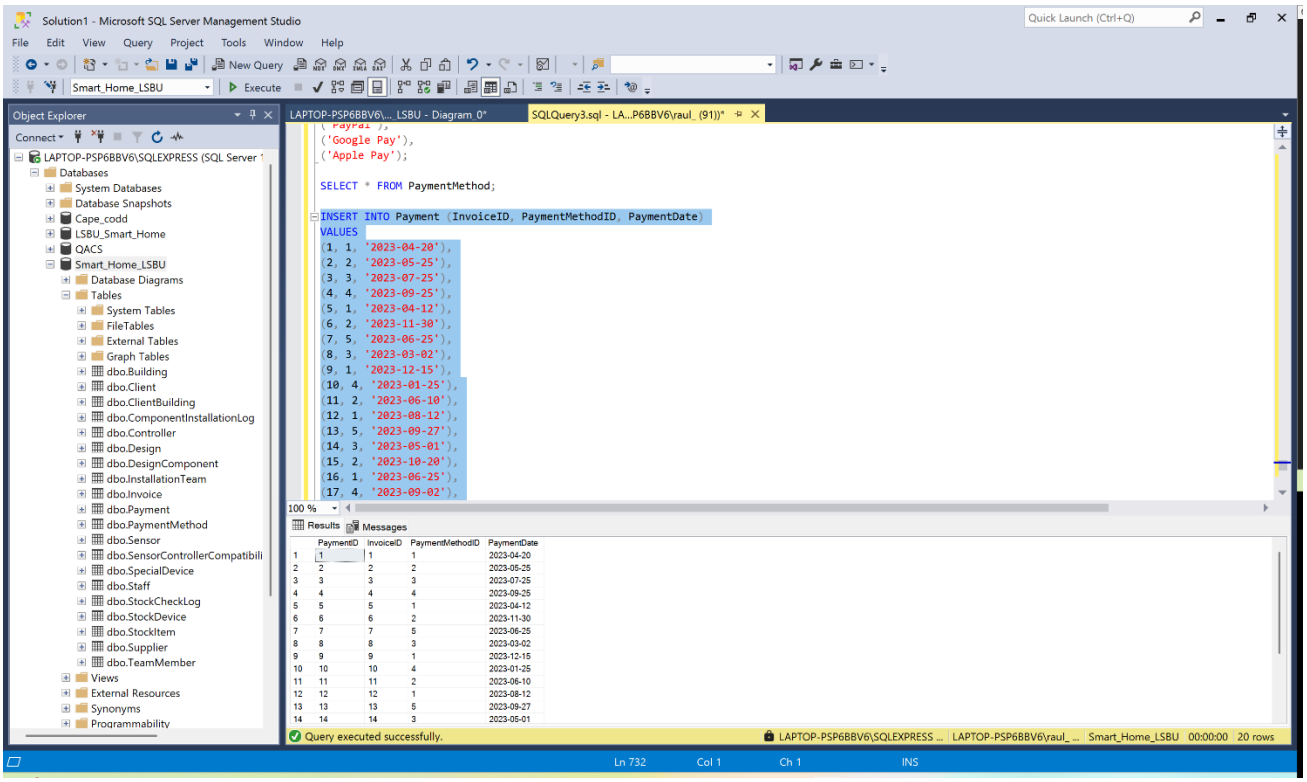

Figure 42. Insert Payment Values

The problem with the ClientBuilding table was that the ClientID values didn't match those in the Client table. This meant that when trying to join the two tables, no data appeared because there were no matching IDs. Once we deleted the incorrect records and reinserted them using valid ClientIDs, the join worked correctly, and the relationships were restored. This can be observed in *Figure 43*.

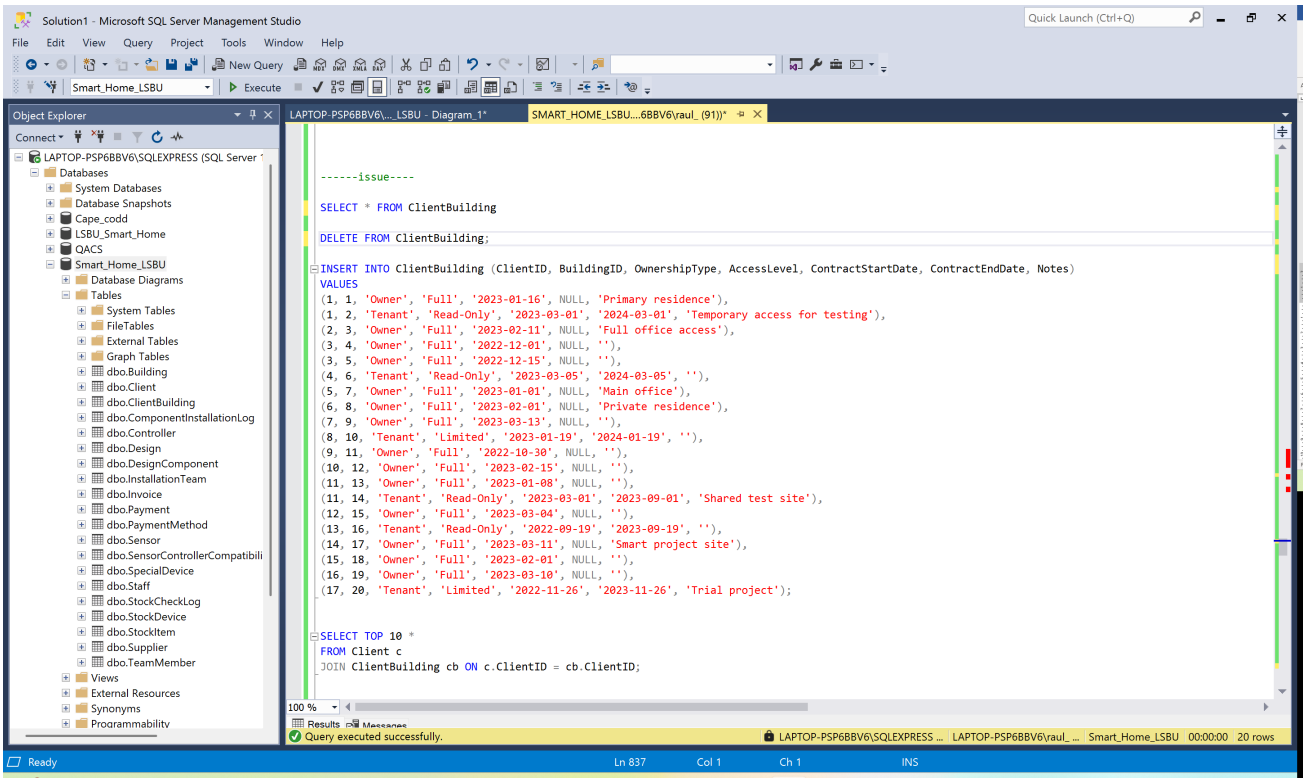

Figure 43. ClientBuilding Correction

6. Queries using Structured Query Language – SQL

In this section, I will demonstrate, using screenshots and T-SQL, the questions required in Phase 4.

Query 4. a

Write a query to show details of clients, their property location(s) and the total value of Smart Home designs installed, specifically for those clients who have made the most expensive and the least expensive total values.

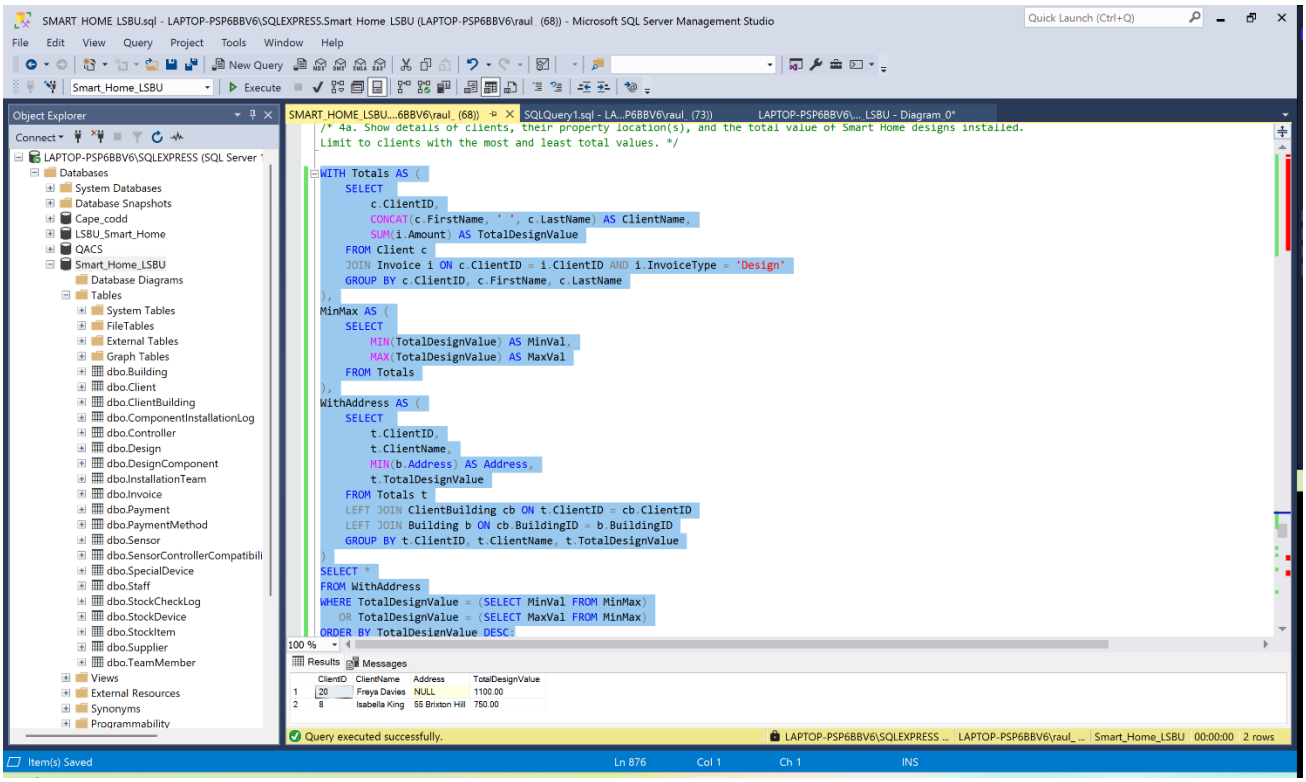

Figure 44. Query 4. A

As seen in *Figure 44*, the first part of the *ClientDesignTotals* code creates a temporary table that returns the client's ClientID, first name, last name, address, and the total sum of the "Design" invoices. The second part of the code calculates the minimum and maximum *TotalDesignValue* for all clients to identify who spent the most and who spent the least. Finally, the SELECT statement shows us only the clients whose total spending matches the

minimum or maximum value from the entire list. The result shown at the bottom of the code responds to Query 4.a, which displays the client details. The final result indicates that Jake Reed is the client with the highest spending on his Smart Home designs (£1980) and Isabella King is the one who spent the most on her designs (£750).

Query 4. b

Write a query to calculate individual totals for the number of complete, incomplete and cancelled WiFi product orders from each of its suppliers.

To resolve this query, I had to modify the values in the StockItem table, as it originally lacked an OrderStatus field. As shown in *Figure 45*, the implementation code updates the OrderStatus field in the StockItem table, assigning corresponding values to the Wi-Fi products based on their order status. The OrderStatus values were assigned as 'Complete' for certain items, 'Incomplete' for others, and 'Cancelled' for cancelled orders.

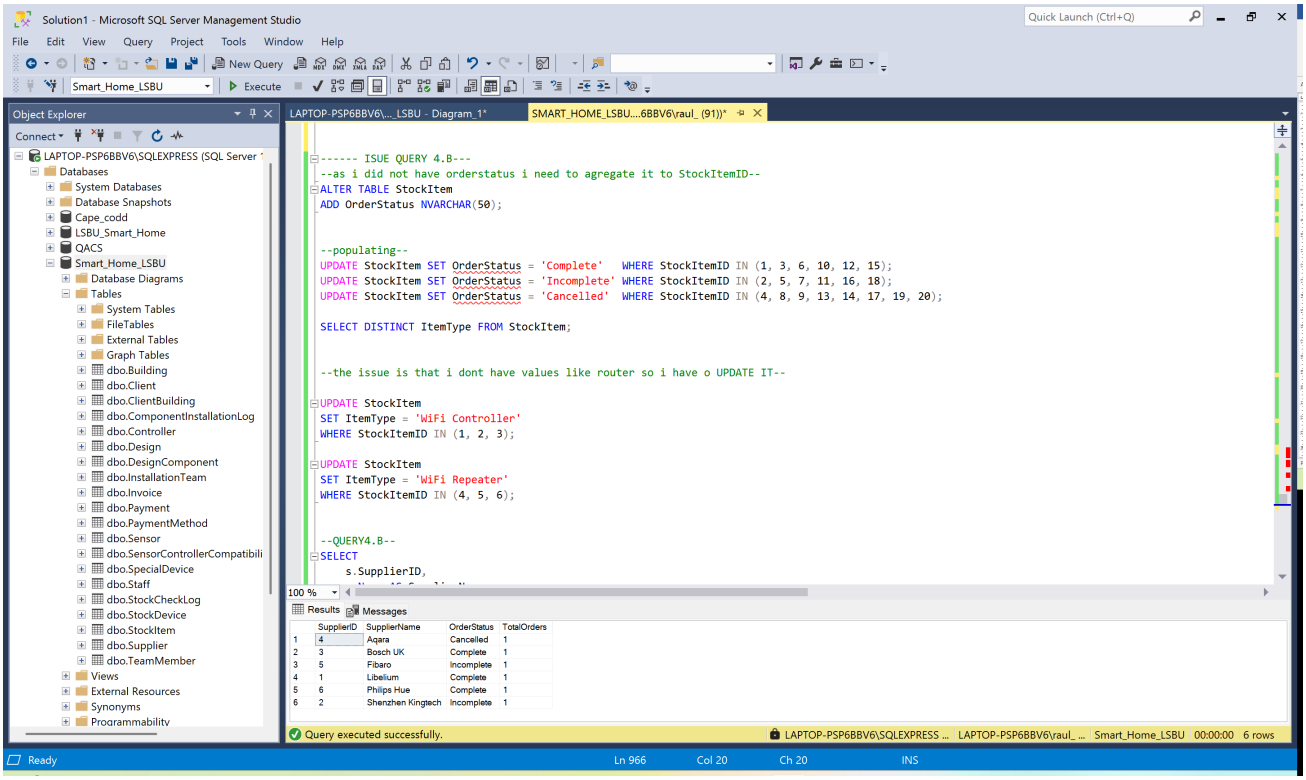

Figure 45. Query 4. B issue

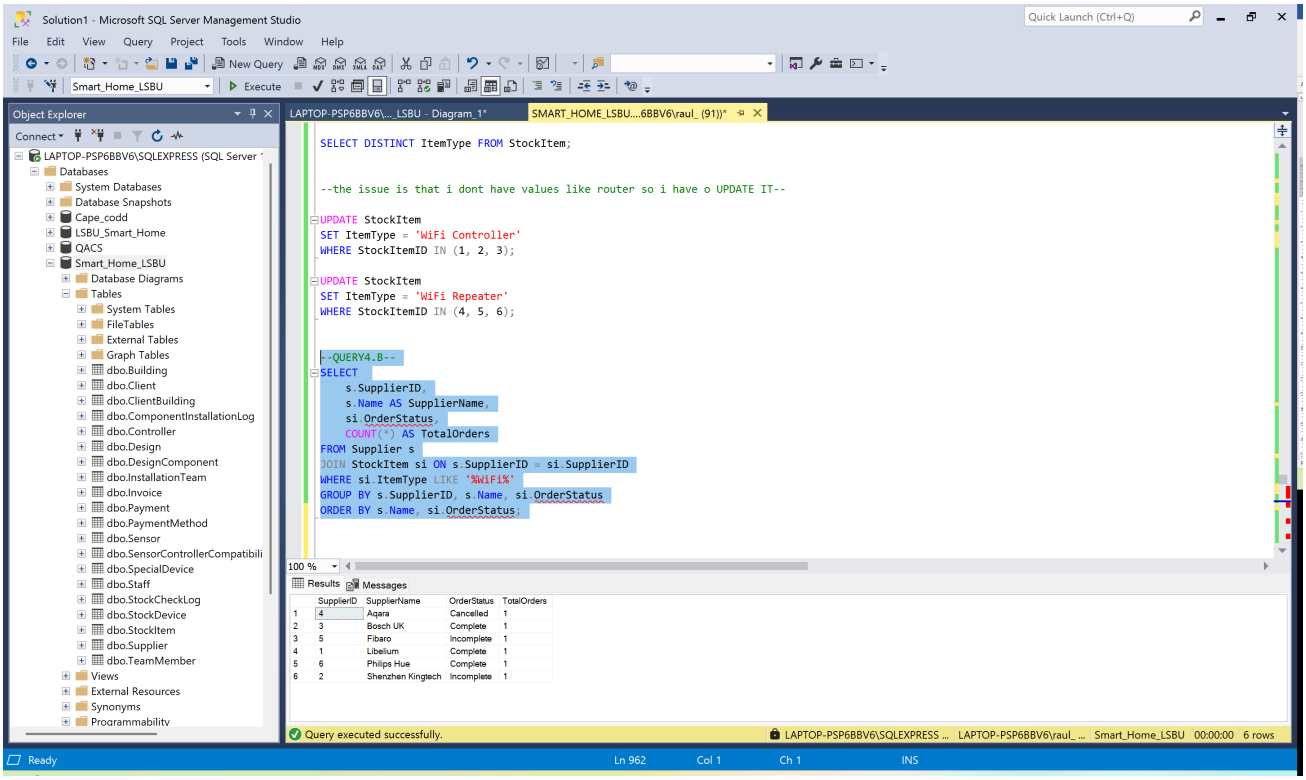

Figure 46. Query 4.b corrected

Figure 46 shows the result of query 4.b. The query calculates individual totals for each Wi-Fi product supplier, grouping them by their *OrderStatus* (whether the order is complete, incomplete, or cancelled). This enables the analysis of each supplier's performance and the monitoring of order status based on their current status.

The results show that each supplier has at least one Wi-Fi order, and these are categorized by status. For example, Bosch UK, Libelium, and Philips Hue have full orders, while Fibaro and Shenzhen Kingtech have incomplete orders. This helps LSBU monitor supplier performance and identify those struggling to fulfil orders.

Query 4. c

Write a query to find the details of specialist staff who are available for booking onto installation jobs this week.

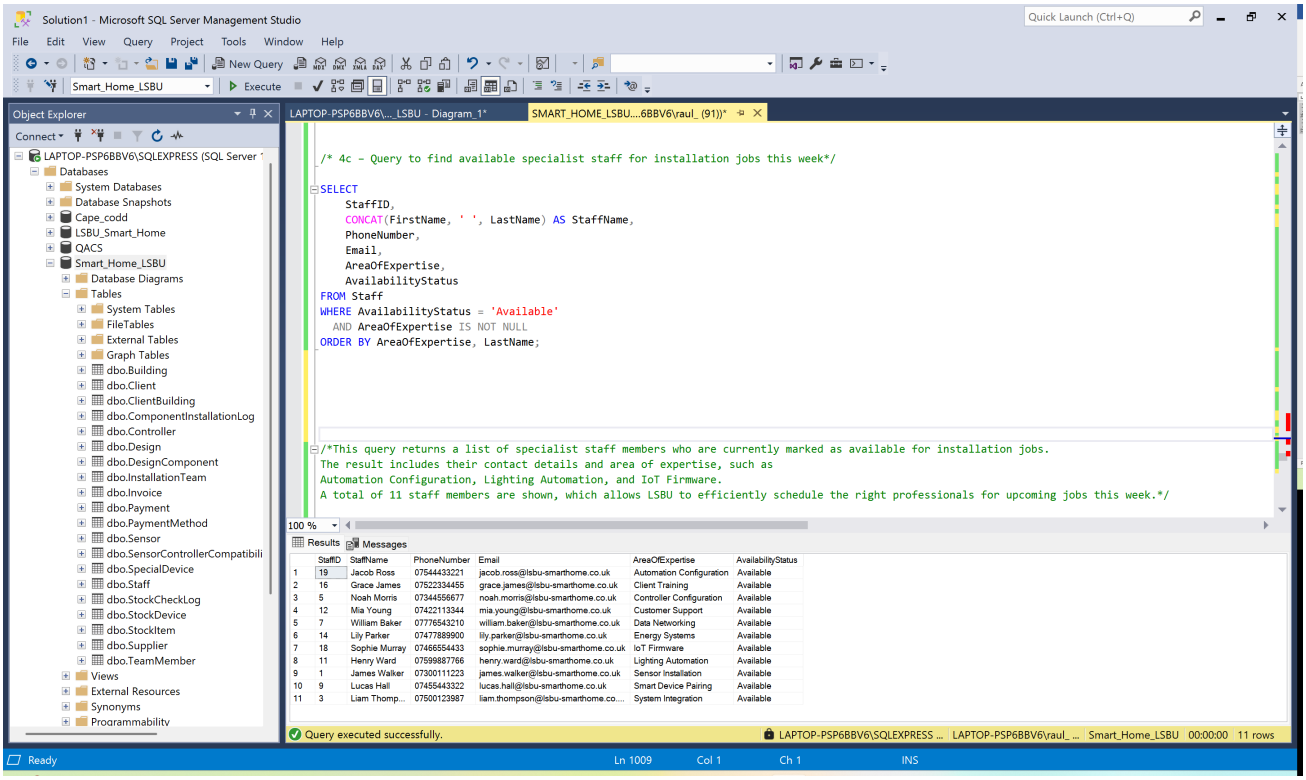

Figure 47. Query 4.c

To resolve query 4.c, as shown in *Figure 47*, the first line of code selects the details of the specialized personnel available this week to be assigned to installation work. The information obtained includes the personnel ID, name, phone number, technical skill, and availability status. This is achieved through a filter with WHERE AvailabilityStatus = 'Available', which ensures that only those staff members who are free to be assigned to installation tasks are returned.

The result identifies 11 available technicians, each with a relevant specialization for Smart Home work, such as sensor automation, controller configuration, smart device support, among others. This allows LSBU to efficiently organize and schedule the appropriate personnel for the installations scheduled for that week.

Query 4. d

Implement using triggers and/or stored procedures automatic prevention of double-booking of specialist staff during the installation booking process.

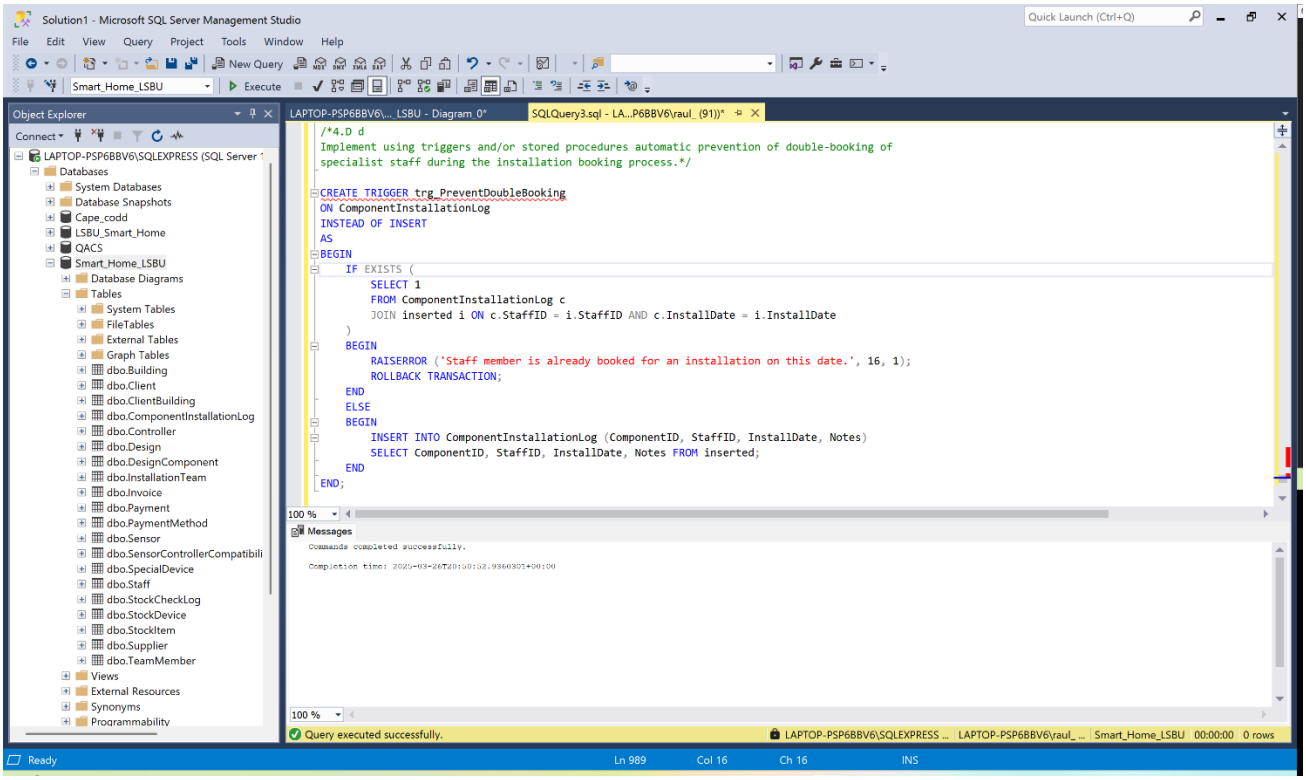

Figure 48. Trigger Creation

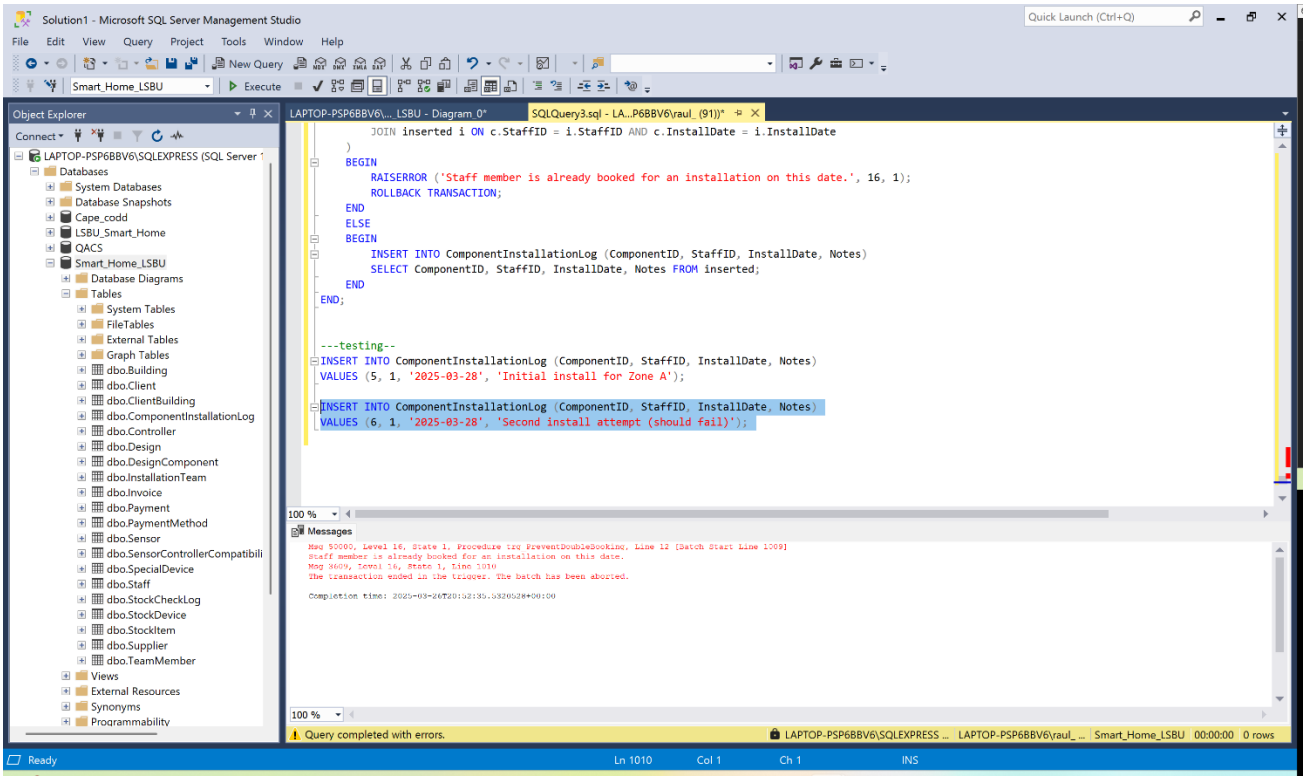

Figure 49. Trigger testing 1

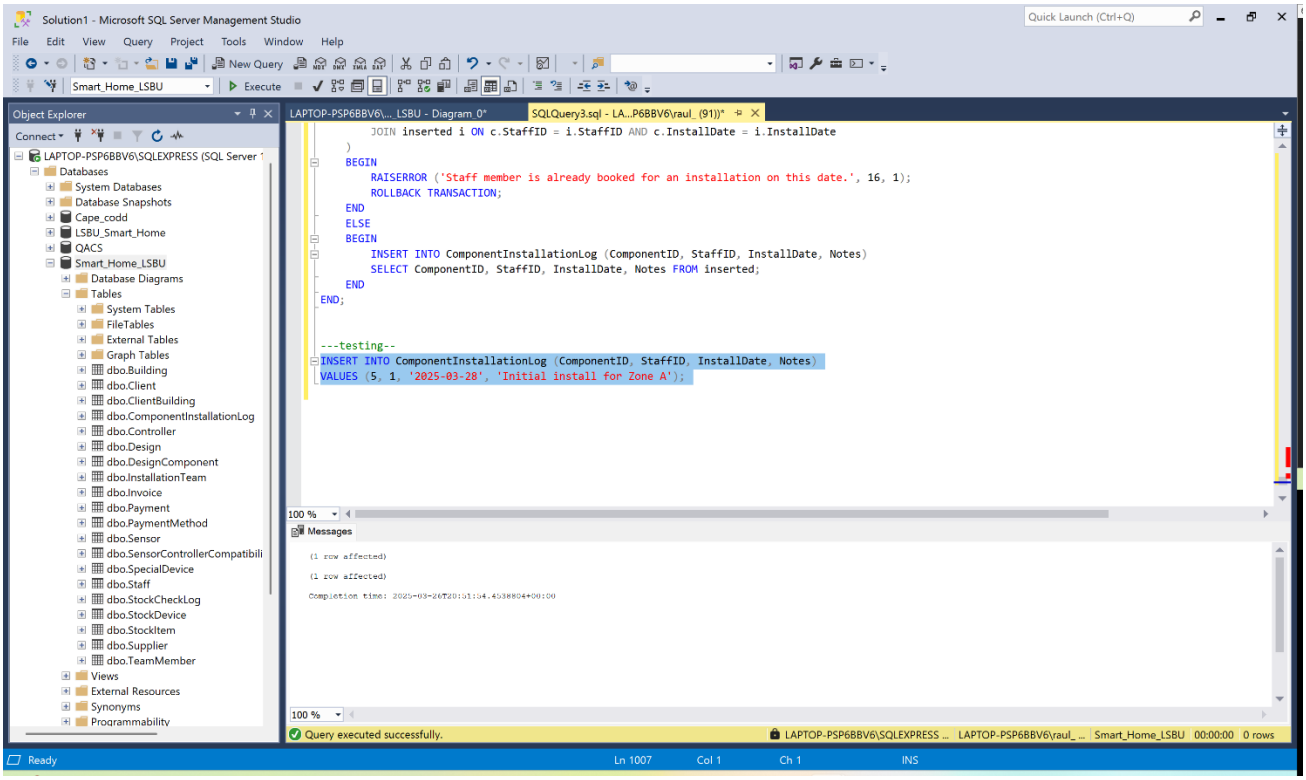

Figure 50. Trigger testing 2

As shown in *Figure 48*, a trigger called *trg_PreventDoubleBooking* was implemented to automatically prevent the double assignment of specialized staff to the same team during the installation process. The trigger fires on each insert attempt into the ComponentInstallationLog table and checks whether the staff member is already assigned to another team on the same installation date. If a duplicate assignment is detected, an error is thrown, and the transaction is rolled back to ensure there are no scheduling conflicts. Otherwise, the trigger allows the new data to be inserted without issue as shown in *Figure 50* and *Figure 51*.

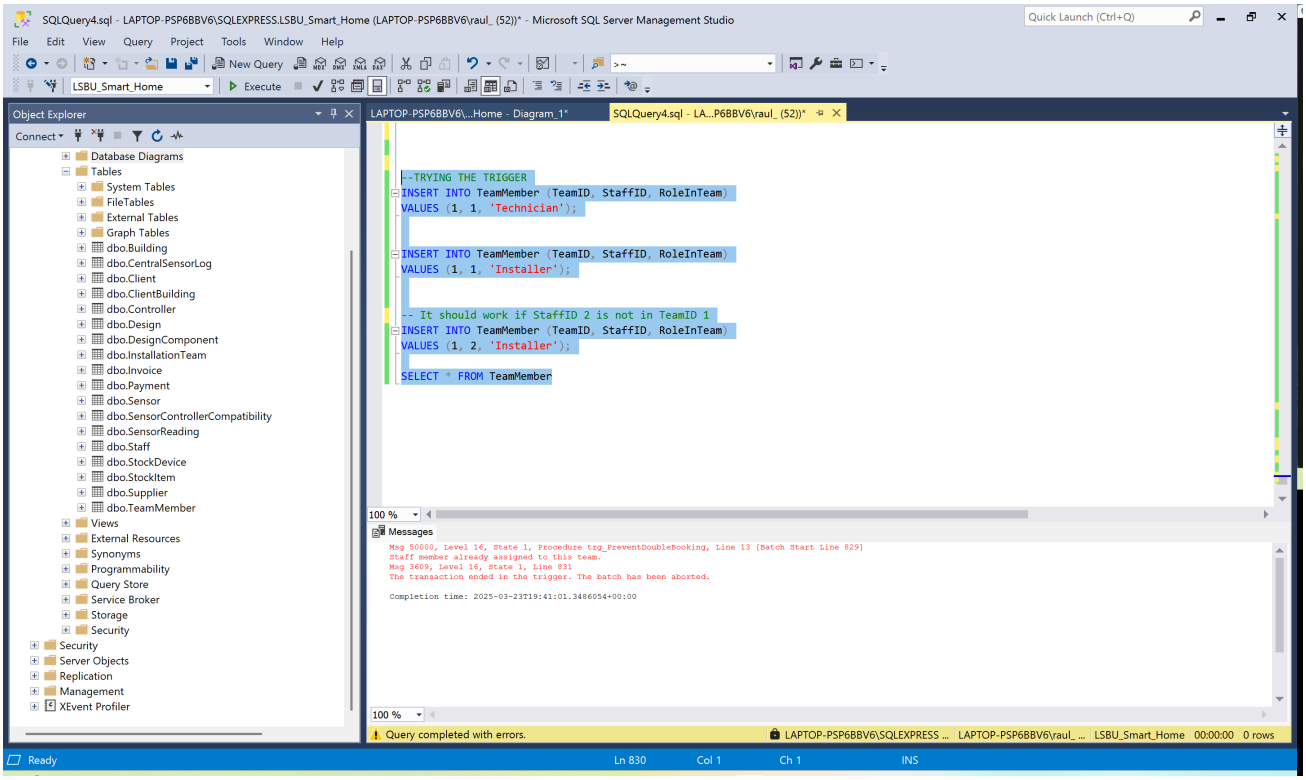

Figure 51. Trying Trigger

Query 4. e

Write a Stored Procedure that can generate a complete costed device list and total cost for any installation (s) in a specified time related period. The output must also include client and device details. The procedure should be able to accept appropriate parameter values to enable dynamic search by week, month or quarter (3 months) Include appropriate attributes and totals in your report.

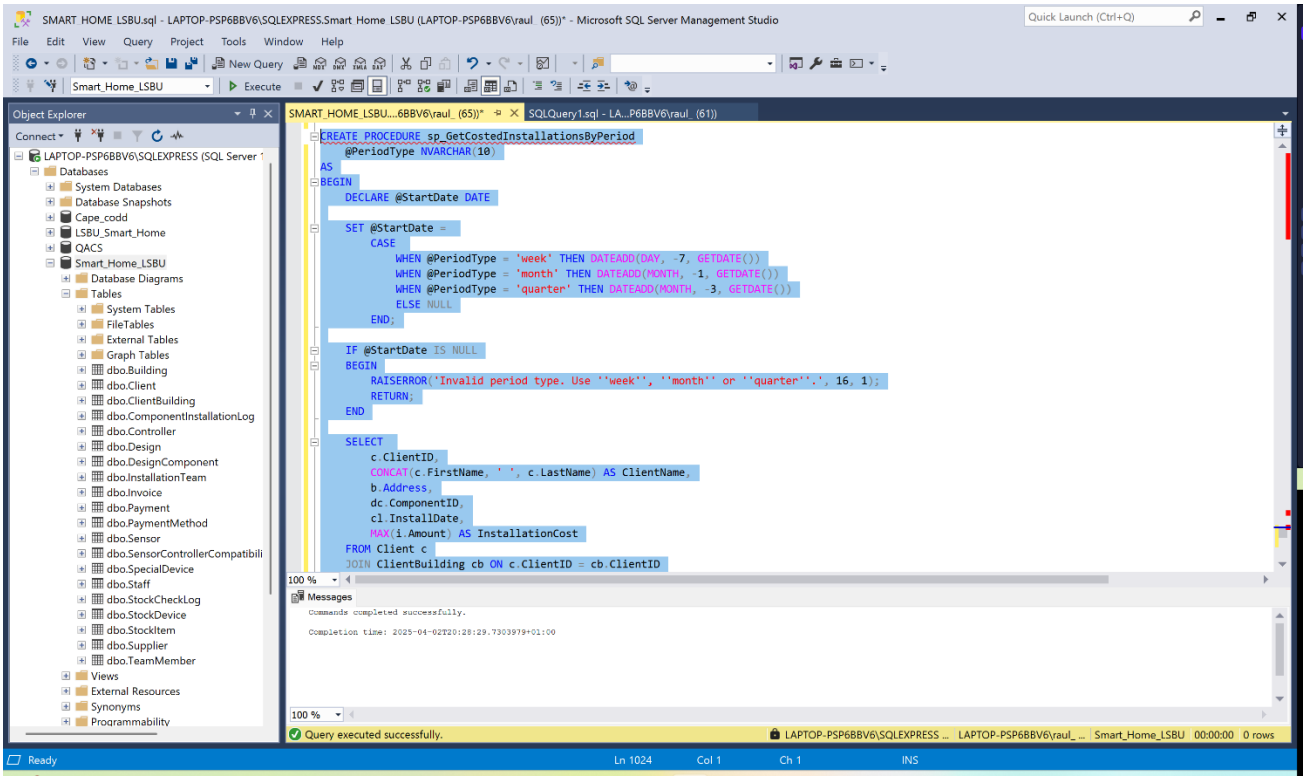

Figure 52. Procedure Creation

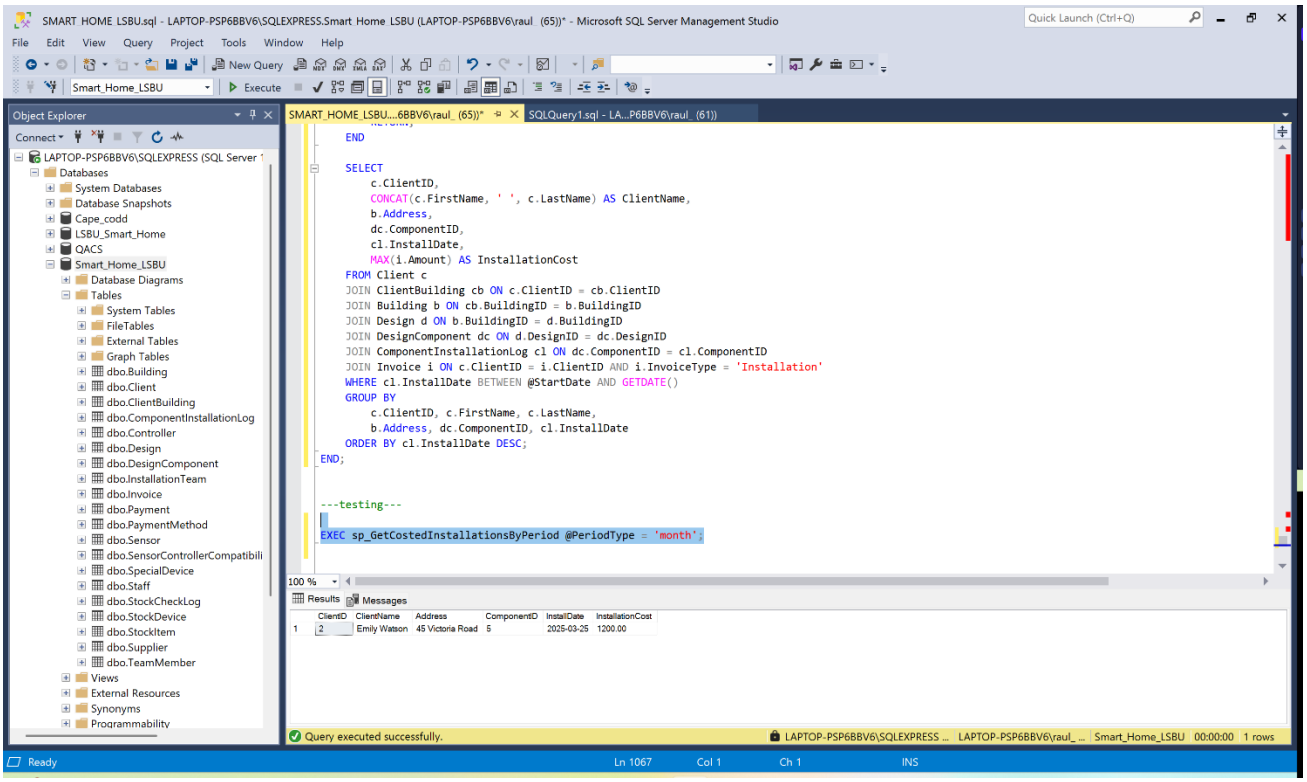

Figure 53. Procedure result

The code in *Figure 52* shows a stored procedure called sp_GetCostedInstallationsByPeriod, which generates a detailed report listing devices and the total cost of installations within a specified time period. This procedure accepts a @periodType parameter, allowing dynamic searches by week, month, or quarter. Depending on the value of this parameter, the start date is adjusted, and the corresponding data is filtered.

Running this procedure returns complete details of the customers, the devices installed, the total installation cost, and the exact dates of installation. This is clearly shown in the results in *Figure 53*.

The code provides a clear and detailed overview of the installations completed during the selected period, making it easier for LSBU to manage the costs associated with Smart Home installations.

7. Data Visualization

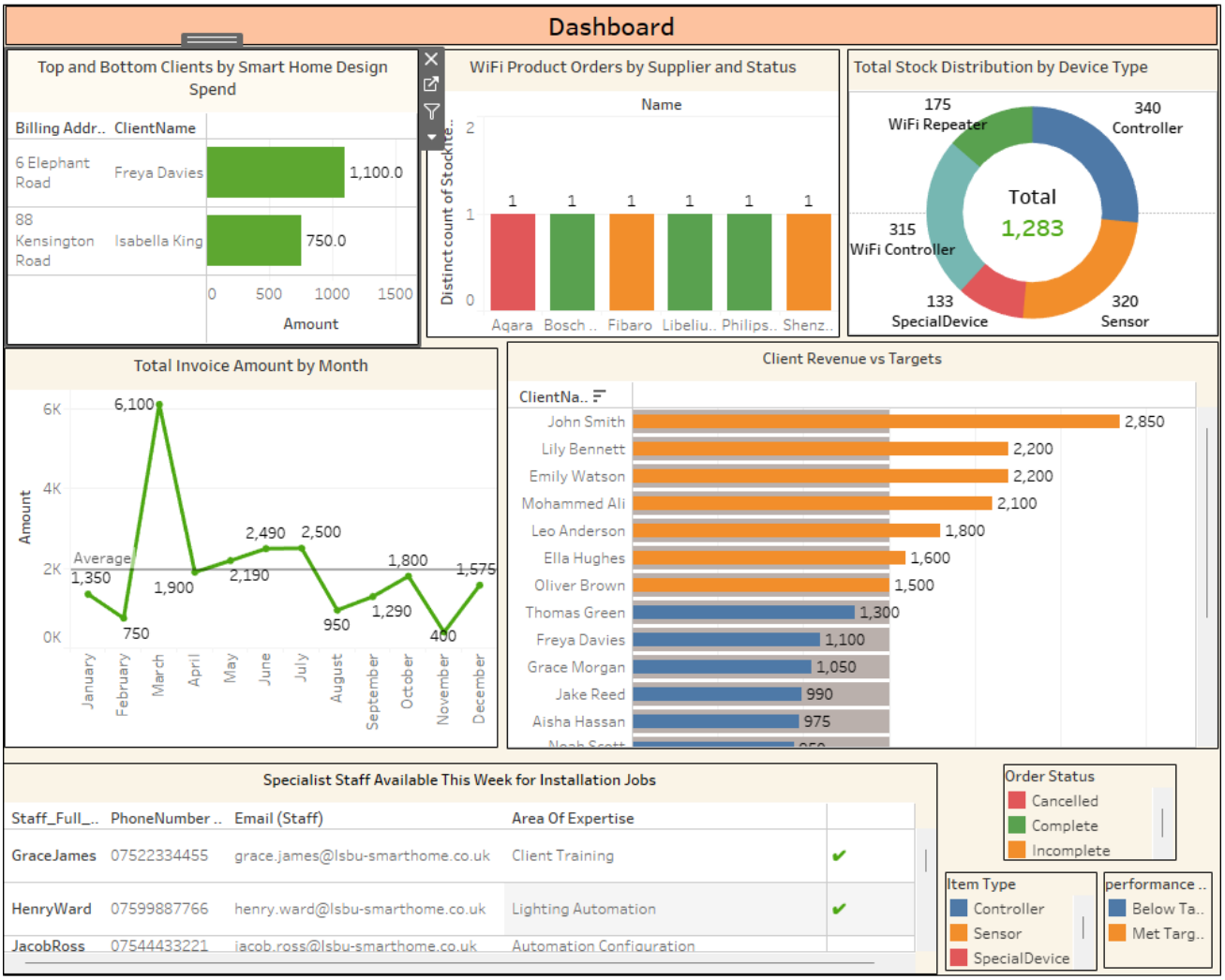

Figure 54. Dashboard

In *Figure 54*, we can observe the graphical representation of the Dashboard. The first top left graph, named “Top and Bottom clients by Smart Home Design Spend” is the answer for **Query 4. a**. The graph titled “Wi-Fi Product Order by Supplier and Status” represents **Query 4. b**. The graph titled “Specialist Staff Available This Week for Installation Jobs” represents **Query 4.c**.

The next graphs were created randomly by me. The graph titled “Total stock distribution by Device Type” represents the sum of the quantity available for each device, illustrating how stock is distributed. This graph aims to provide information on the current inventory levels of various smart home devices.

The following graph, titled “Total Invoice Amount by Month,” illustrates the monthly revenue generated. The line chart tracks total invoice amounts every month, highlighting revenue trends throughout the year. Furthermore, the dotted trendline might show average performance.

The las graph with name “Client Revenue vs Targets” compares the actual revenue per client against a predefined revenue target (1500). Each client’s actual revenue (coloured bar) is compared to a standard target (background bar). Colour indicates whether they met or missed the target. Useful for performance tracking and client segmentation.

The dashboard can be accessed on the following GitHub repository:

https://raulcimpe.github.io/Dashboard/

8. Video Demonstration

Microsoft:

https://stulsbuac-my.sharepoint.com/:v:

Google Drive:

https://drive.google

9. References

Dybka, P. (2016). *Crow's Foot Notation*. [online] Vertabelo Data Modeler. Available at: https://vertabelo.com/blog/crow-s-foot-notation/.

Visual Paradigm (2019). *What is Entity Relationship Diagram (ERD)?* [online] Visual-paradigm.com. Available at: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/.

Staff, F. (2024). *Understanding Functional Dependencies in Software Engineering*. [online] As We May Think — products & tools for thought. Available at: https://fibery.io/blog/product-management/functional-dependencies-explained/.

Microsoft (2024). *Database normalization description*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description.

10. Appendix

Originality & Use of Generative AI Statement

I understand that using the work or ideas of others, including AI-generated content, without proper acknowledgement is considered academic misconduct.
I confirm that this submission is my own original work, and that all sources and references have been included in line with LSBU Academic Regulations.

DECLARATION OF AI USE:
I **DID** use Generative AI technology in the development, writing, or editing of this assignment.

1. Tools used and purpose

I used ChatGPT mainly for idea generation, technical clarification, and general writing support. It helped me brainstorm database structure, refine explanations of certain concepts (like normalization and functional dependencies), and polish grammar in a few parts. Additionally, Grammarly was used for checking English grammar and improving sentence structure, especially when translating from Spanish to English since English is not my first language.

2. Sections supported by AI

AI tools supported parts of:

- Clarifying technical concepts (ERD, functional dependencies)
- Some of the SQL query explanations
- Minor parts of the writing and formatting

All technical implementation, data modelling, and database creation was done manually by me.

3. Example prompts used

- “Explain how a trigger can prevent staff double booking.”
- “How can I describe a normalized ERD in simple academic language?”
- “Translate this sentence from Spanish to English while keeping technical accuracy’”

4. Reflection on using AI

Using AI helped speed up the writing process and provided quick support when I needed to check grammar or describe technical parts more clearly. While helpful, I still made sure to review and edit everything based on my own understanding of the case study and project. It was a tool to support my work.

By including this statement in my coursework submission, I attest that the information provided in this Originality and Use of Generative AI Statement is accurate and complete to the best of my knowledge. I understand that providing false information is a violation of the LSBU Academic Regulations and may result in academic and/or disciplinary consequences.