Student Name: Quang Dong Nguyen

Student ID: 20744696

Class: Database Design and Development INFS2001

Assignment 2

# Question 1: Selected Additional Exercise

## 1/Complete Question alpha and gamma in the additional Exercises for practical 8

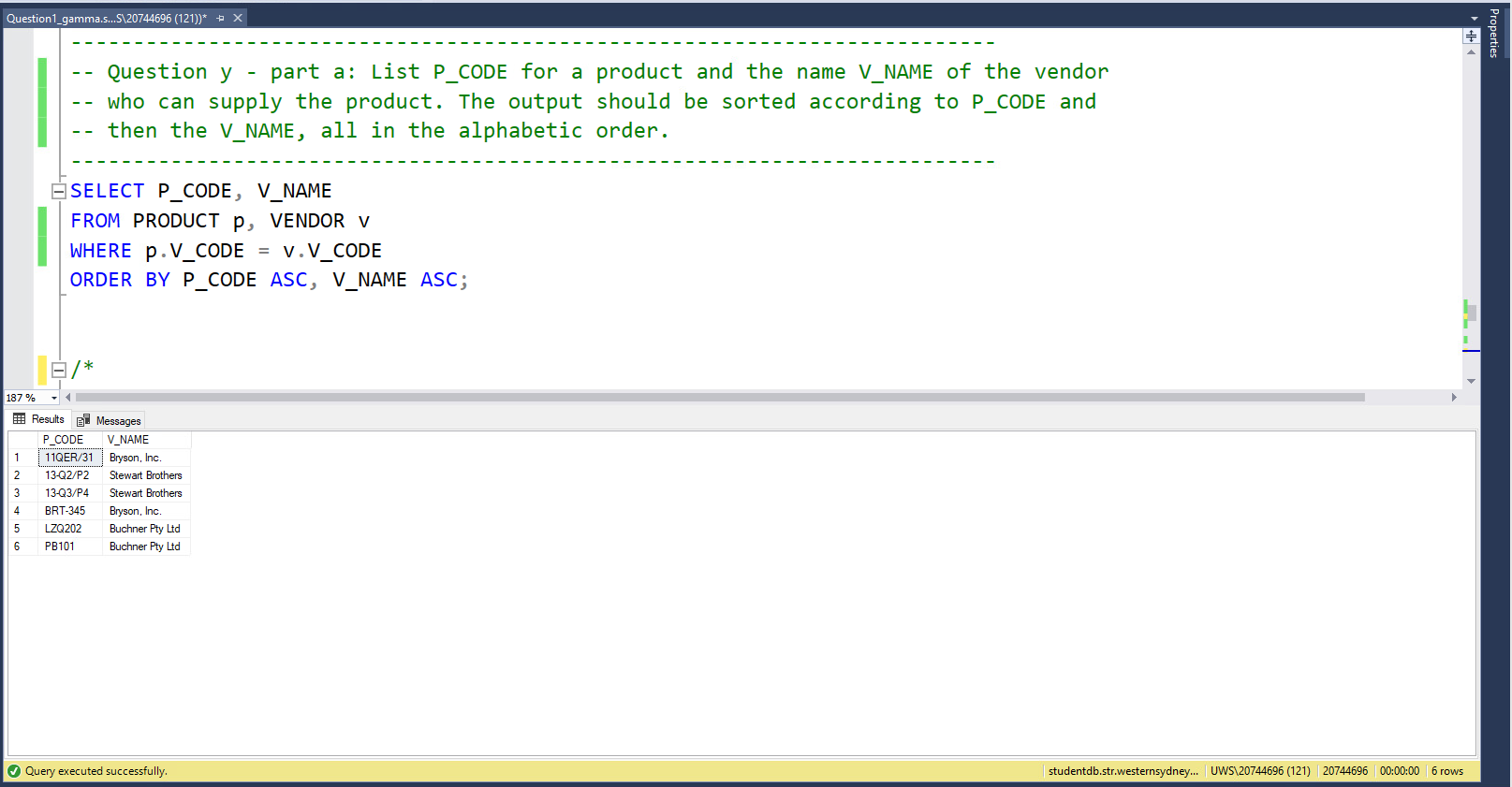
### Question alpha:

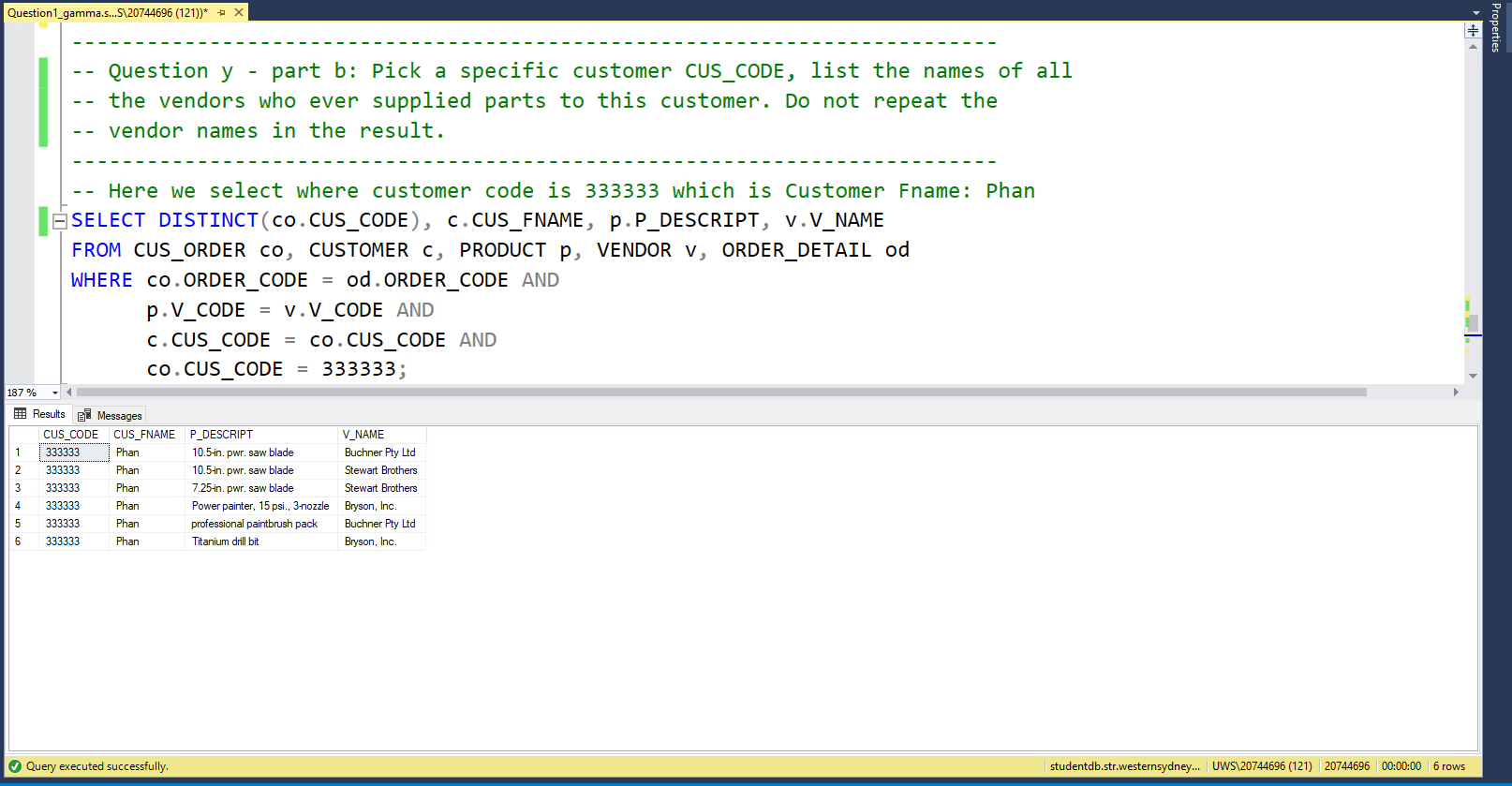
Data anomaly occurs when alternation of records in the redundant data are not made successfully. Deletion anomalies occur as when deleting records from the redundant data results in the accidental loss of data.

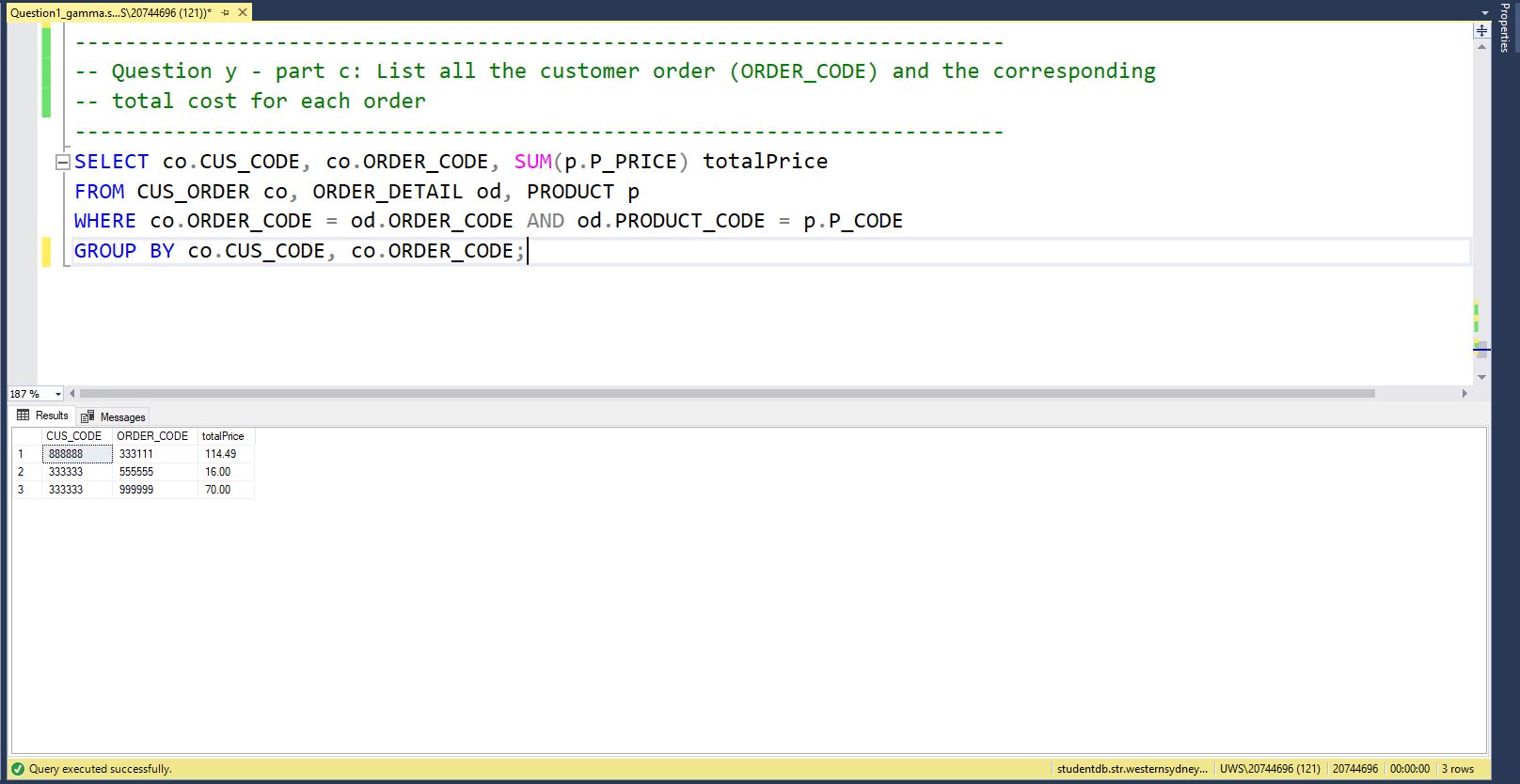
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StoreID** | **StoreName** | **StorePostcode** | **SalePerson** | **Number** |
| 001 | Aukland | 2145 | Mary Hatson | 043-222-435 |
| 001 | Aukland | 2145 | John Dean | 043-261-413 |
| 003 | Pokart | 2133 | Garp Fisherman | 042-260-222 |
| 004 | Aokmond | 2000 | Skylar White | 041-250-222 |
| 004 | Aokmond | 2000 | Walter White | 041-250-221 |
| 002 | SpringField | 2010 | Felina | 041-250-099 |

For instance: In the 3rd row, deleting Garp Fisherman and his number out of the store as he is planning to retire will result in losing information about the existence of Pokart store, including its store id, store name and as well as the store post code.

### Question gamma:







## 2/Complete Question theta in the additional exercises for Practical 9

### Question theta:

Let says we have a table in 1NF that records House Ids, their address and people who live their

**1NF Table:**

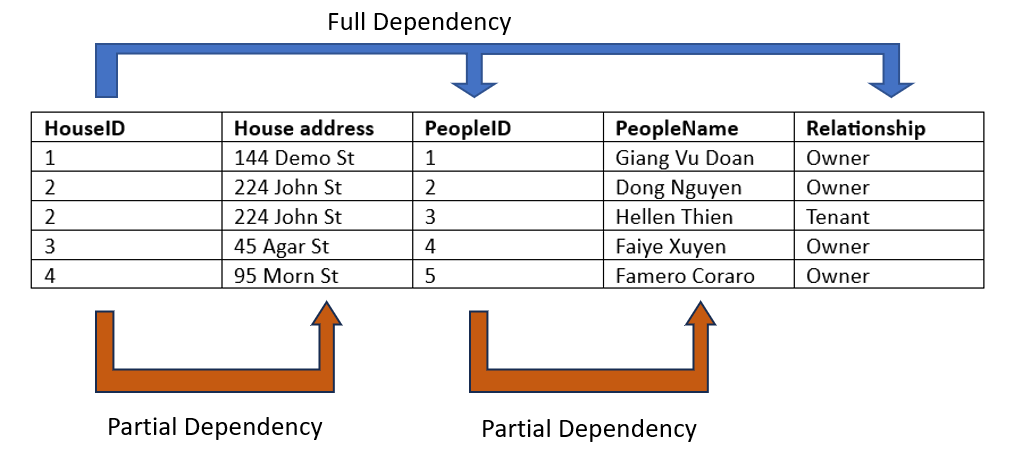
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HouseID** | **House address** | **PeopleID** | **PeopleName** | **Relationship** |
| 1 | 144 Demo St | 1 | Giang Vu Doan | Owner |
| 2 | 224 John St | 2 | Dong Nguyen | Owner |
| 2 | 224 John St | 3 | Hellen Thien | Tenant |
| 3 | 45 Agar St | 4 | Faiye Xuyen | Owner |
| 4 | 95 Morn St | 5 | Famero Coraro | Owner |

**HouseID, PeopleID** -> Relationship

**HouseID** -> House Address

**PeopleID** -> People Name

**Diagrams with Functional Dependency, attributes and values:**



Hence to transform 1NF table to 2NF table, we must remove partial dependencies exist in the table.

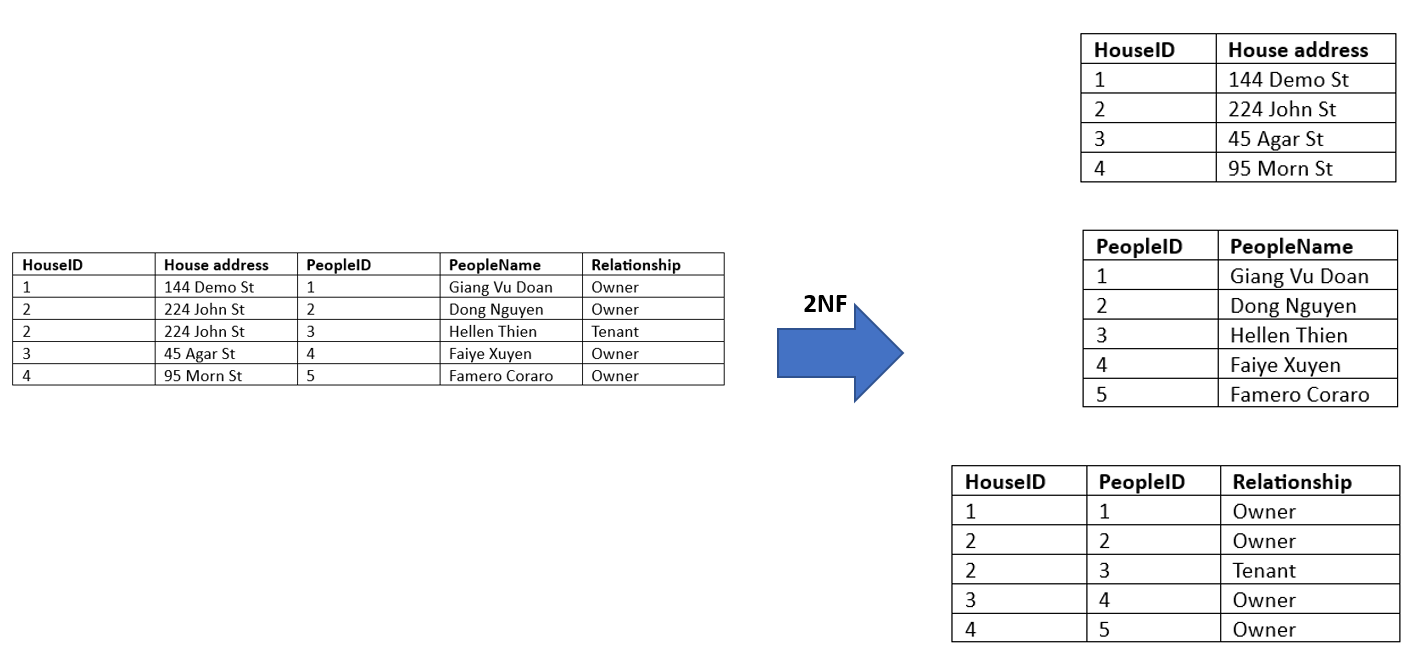


Figure 1: Data Normalisation from 1NF to 2NF

## 3/Complete Question alpha and Beta in the additional Exercises for Practical 10

### Question alpha:

**Determine Functional Dependency in each of the table:**

2NF:

**TrainID** -> TrainNumber, BranchNo, BranchOperator

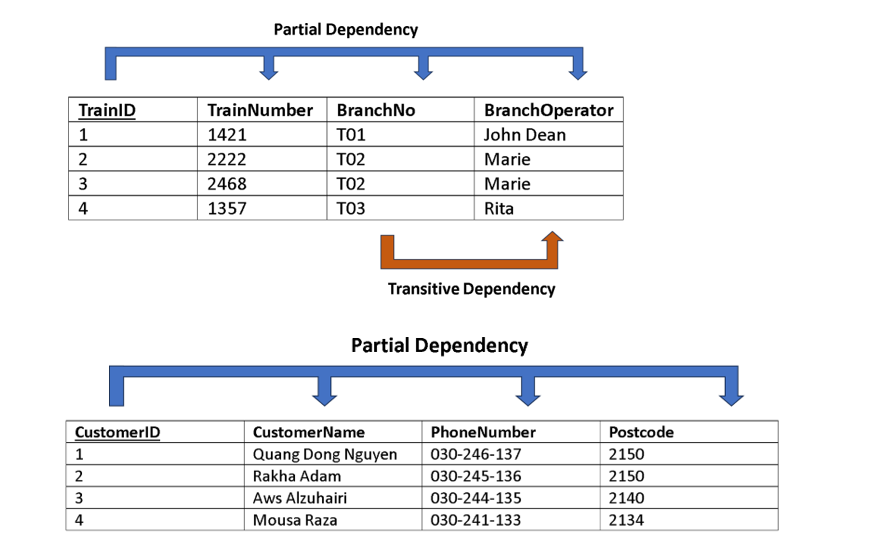
**CustomerID** -> CustomerName, PhoneNumber, Postcode

**Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **TrainID** | **TrainNumber** | **BranchNo** | **BranchOperator** |
| 1 | 1421 | T01 | John Dean |
| 2 | 2222 | T02 | Marie |
| 3 | 2468 | T02 | Marie |
| 4 | 1357 | T03 | Rita |

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **PhoneNumber** | **Postcode** |
| 1 | Quang Dong Nguyen | 030-246-137 | 2150 |
| 2 | Rakha Adam | 030-245-136 | 2150 |
| 3 | Aws Alzuhairi | 030-244-135 | 2140 |
| 4 | Mousa Raza | 030-241-133 | 2134 |

**Diagrams with Functional Dependency, attributes and values:**



Hence to make data normalised into 3NF we eliminate any transitive dependencies by splitting tables into smaller unit tables within the data, as they could cause some anomalies like insertion, deletion and update anomalies. Hence the table that contains transitive dependency will be splitted into 2 tables as shown in in the Diagram below:

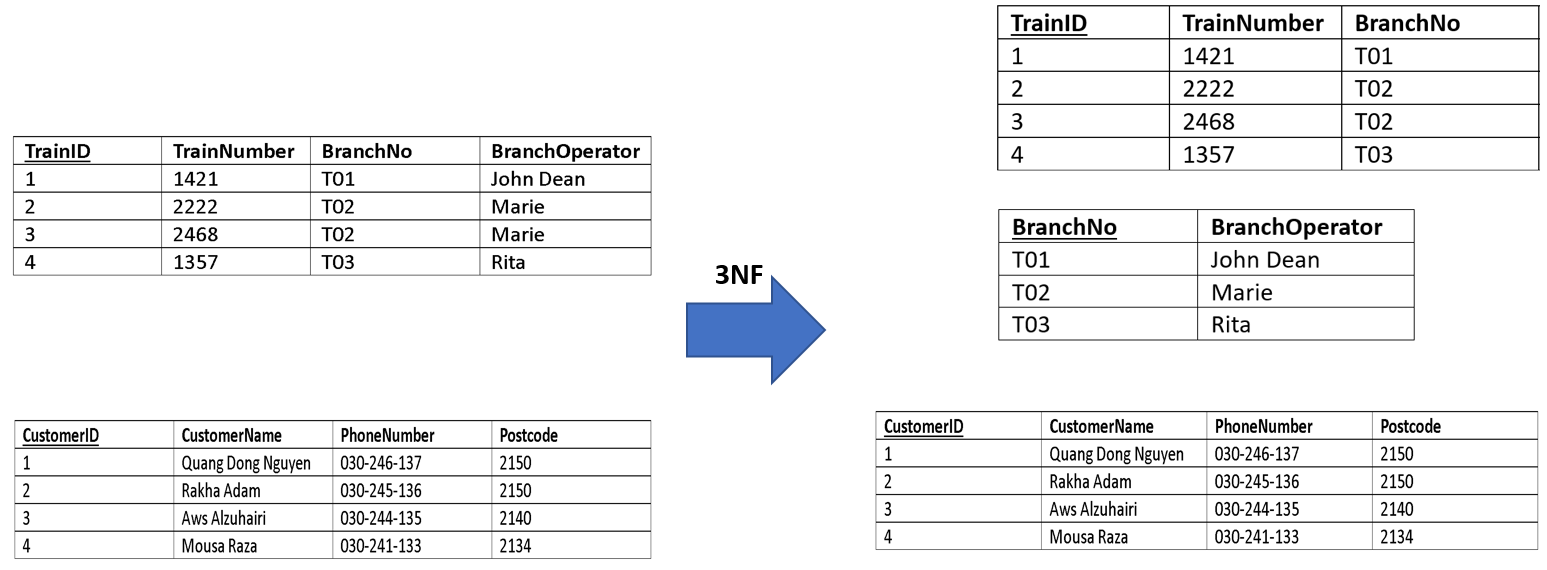


Figure 2: The Process of Data Normalisation from 2NF to 3NF

Hence the final tables are:

|  |  |  |
| --- | --- | --- |
| **TrainID** | **TrainNumber** | **BranchNo** |
| 1 | 1421 | T01 |
| 2 | 2222 | T02 |
| 3 | 2468 | T02 |
| 4 | 1357 | T03 |

|  |  |
| --- | --- |
| **BranchNo** | **BranchOperator** |
| T01 | John Dean |
| T02 | Marie |
| T03 | Rita |

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **PhoneNumber** | **Postcode** |
| 1 | Quang Dong Nguyen | 030-246-137 | 2150 |
| 2 | Rakha Adam | 030-245-136 | 2150 |
| 3 | Aws Alzuhairi | 030-244-135 | 2140 |
| 4 | Mousa Raza | 030-241-133 | 2134 |

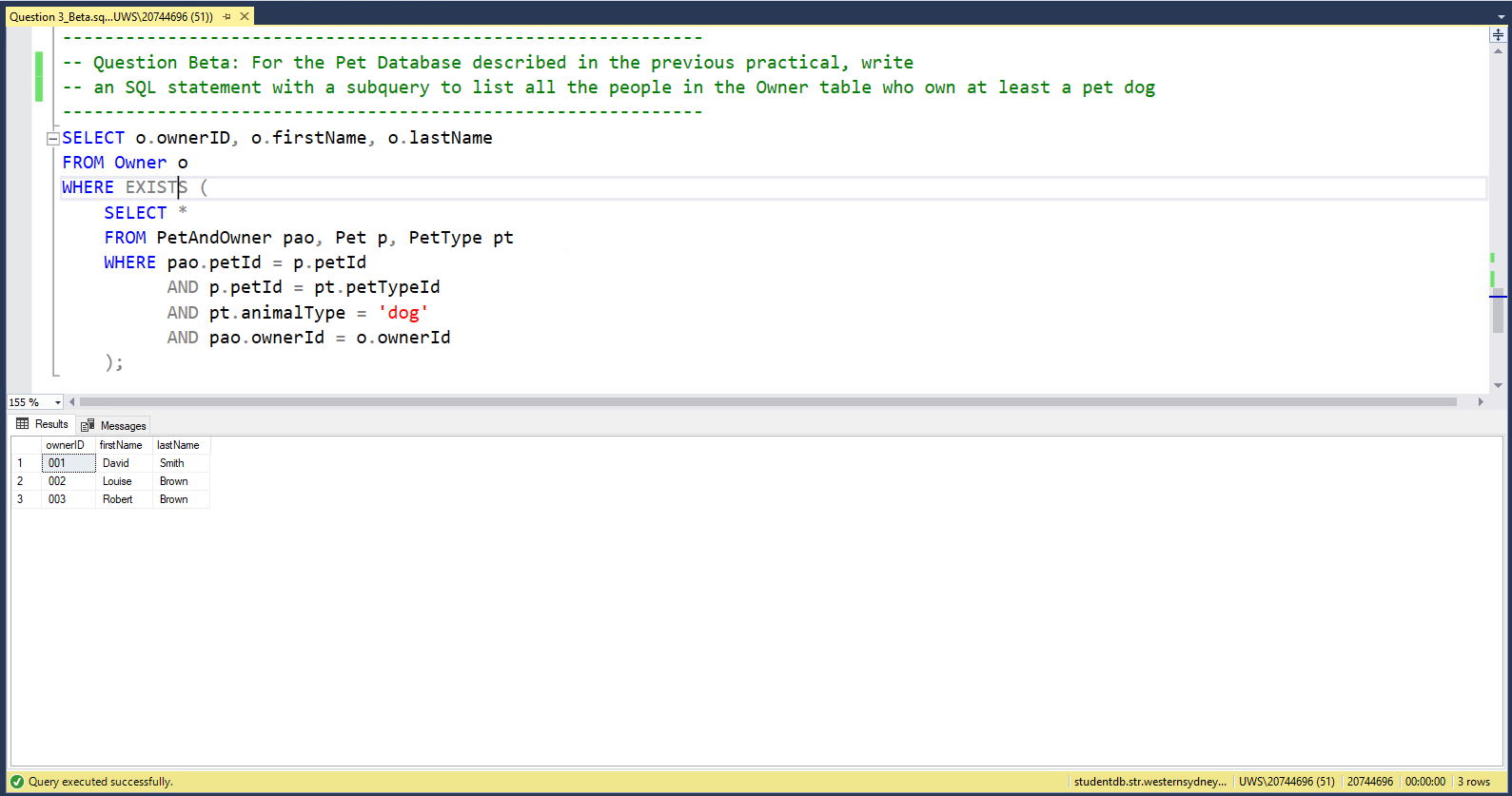
3NF:

**TrainID** -> TrainNumber, BranchNo

**BranchNo** -> BranchOperator

**CustomerID** -> CustomerName, PhoneNumber, Postcode

### Question Beta:



## 4/ Complete Questions alpha and Beta in the additional Exercises for Practical 11

### Question alpha:

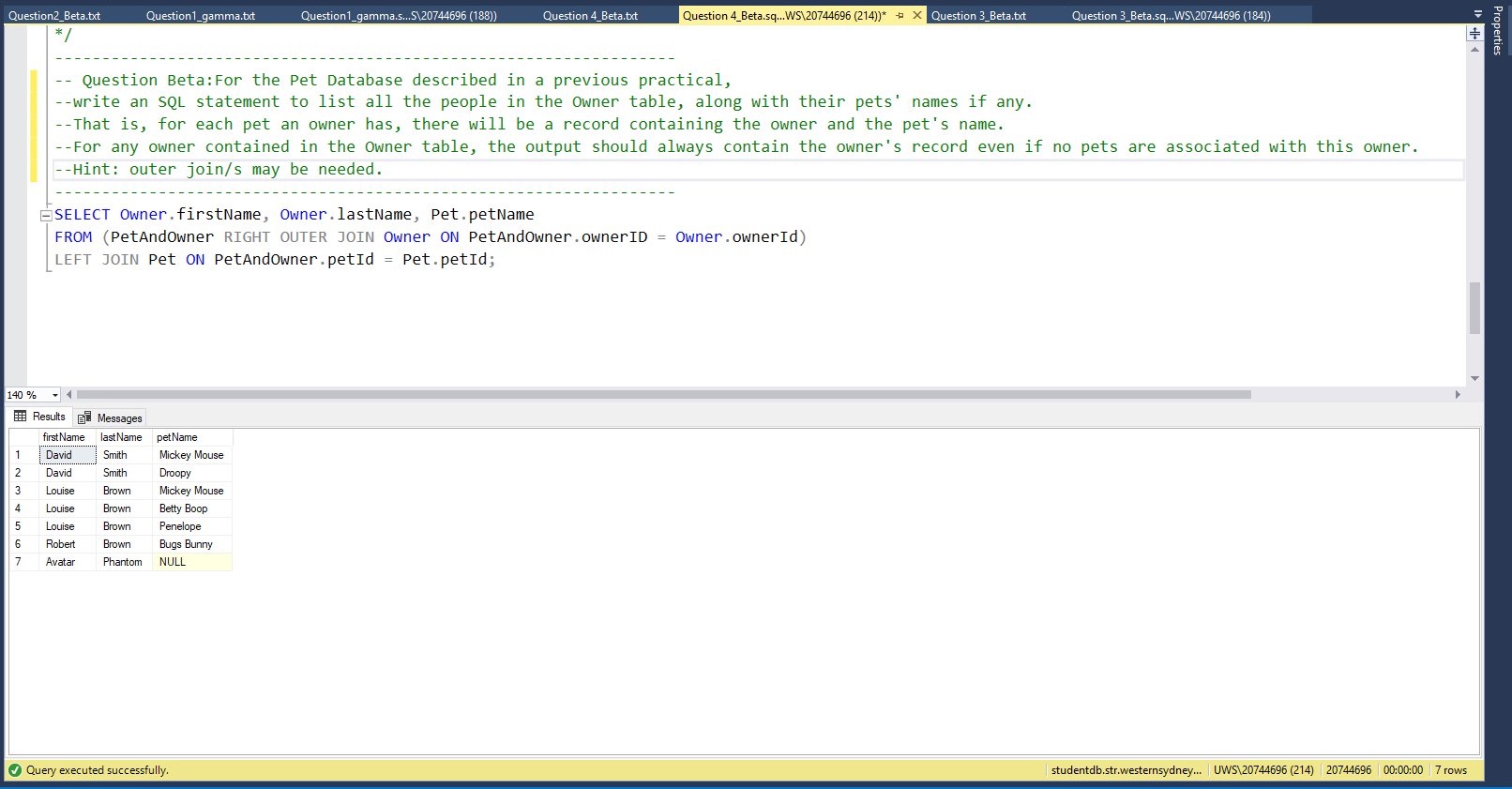
In the context of database transactions, all transactions must display Atomicity, Consistency, Isolation and Durability (often abbreviated to ACID). In short, these are the fundamental steps toward of a successful transaction. Each of these properties means:

* Atomicity: All operations of transactions must be completed as a whole, or none will be executed. Atomicity helps to protect the integrity of the database and to prevent corruption in transactions.
* Consistency: Ensures the outcomes are yielded with consistent results from the concurrent execution of multiple transactions. Consistent results refer to the absolute adaptability of results to all the business rules, database rules and constraints. And if it fails to converge the adaptation, meaning the data drive into an illegal state during the transaction, then the whole transaction will be discarded.
* Isolation: Ensures all transactions execute independently of one another in an isolated environment, meaning that transactions can be executed simultaneously, and no data should have an impact on another. This isolated environment enables both transactions guaranteed to reach completion while not interfering with each other.
* Durability: Indicates the permanent state of data’s consistency. In another word, when a transaction is completed, the database reaches a consistent state, meaning changes in data are permanently recorded in the database and it will not be lost because of any events of disruption such as system’s failure, power outages, etc.

An example to demonstrate the property of the letter “I” in ACID:

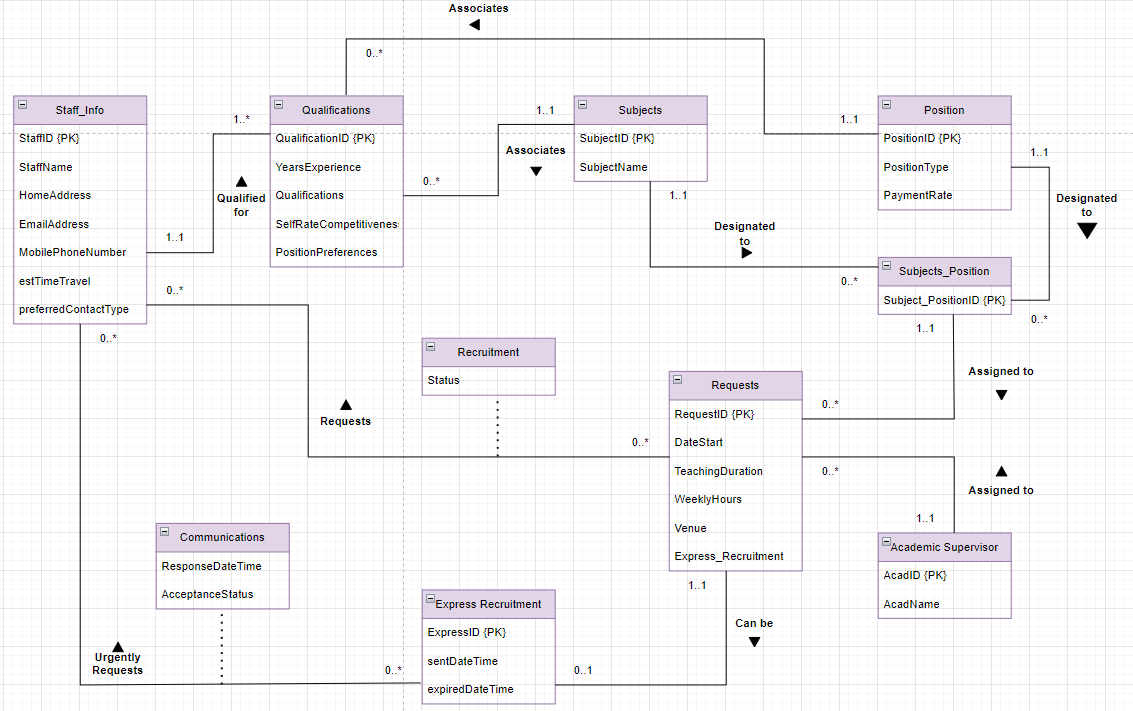
A banking application where two concurrent transactions occur at the same time. Transaction 1 has a monetary transfer of 500$ from account 1 to account 2, while the second transaction shows a withdraw of 100$ from account 1. Based on the Isolation properties of database, it is guaranteed to succeed both transactions. However, it will go as follows: The first transaction is proceeded if there is no systematic errors or logical error happened in the database system, then it will be completed successfully and followed on by the second transaction. Since the data required is based on the first transaction, there will be no second stage of transaction if the first transaction meets an error. In other word, the process on the second state will begin after the operation on the first state of transaction is finished. Therefore, if the second transaction is completed without errors, then the total operation succeeds; Otherwise, the whole transaction operation fails to complete and there will be no transaction is made.

### Question beta:



# Question 2:

**ER Diagram (Entity-Relationship Diagram) of the recruitment system of school:**



From the ERD Diagram, we have the following entities:

* **Staff\_Info:** This entity shows the general information of each staff members
* **Staff\_Subjects:** Shows the qualifications, years experience, self-rate competitiveness and preferred Positions for each subjects
* **Subjects**: Reviews available Subjects.
* **Position**: Reviews available position offered by school.
* **Subjects\_Position:** Tells positions for each subject and pay rate based on the position of staff
* **Requests**: Stores available requests to be sent to staff
* **Academic Supervisor:** Stores information of available Academic Supervisor
* **Express\_Recruitment:** Stores urgent requests that needs to be reviewed.

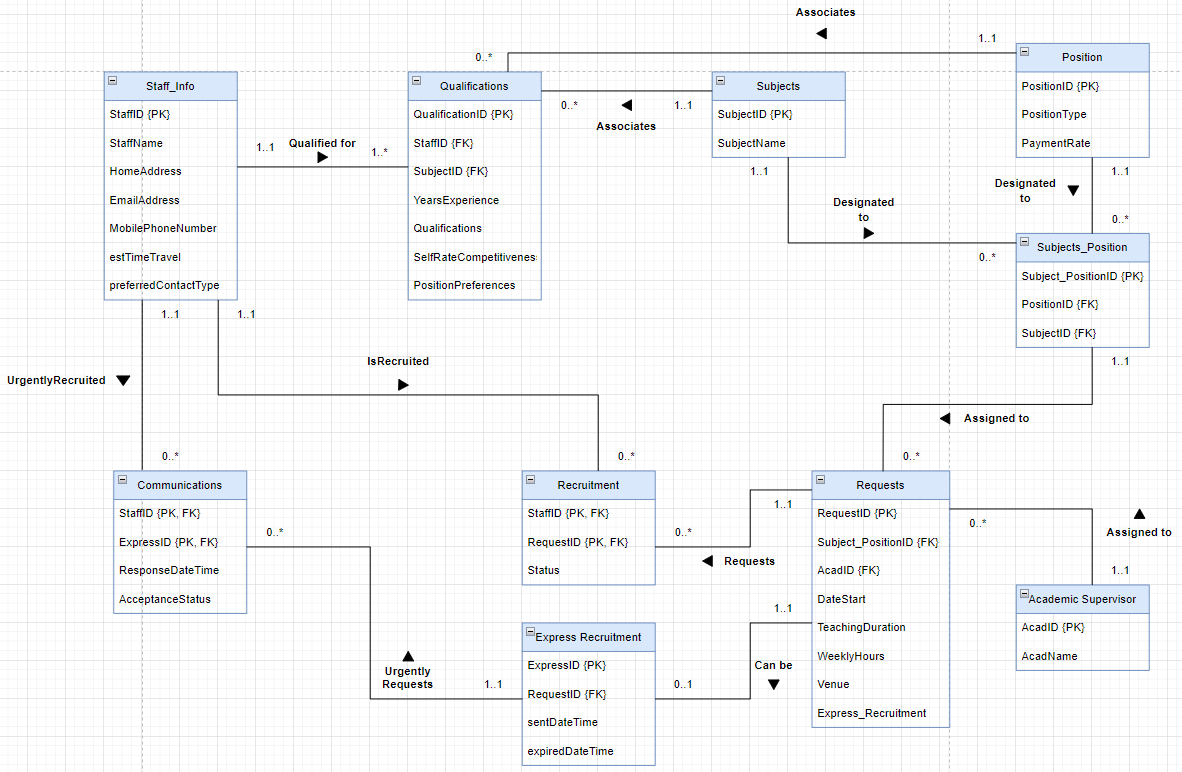
With relationships:

* Recruitment: Tells which staff is assigned to which requests.
* Communication: Tells which staff is assigned to which requests based on the response time of conversation between the staffs and the schools.

To improve conceptual design of the database and business rule. I modified some participation multiplicities:

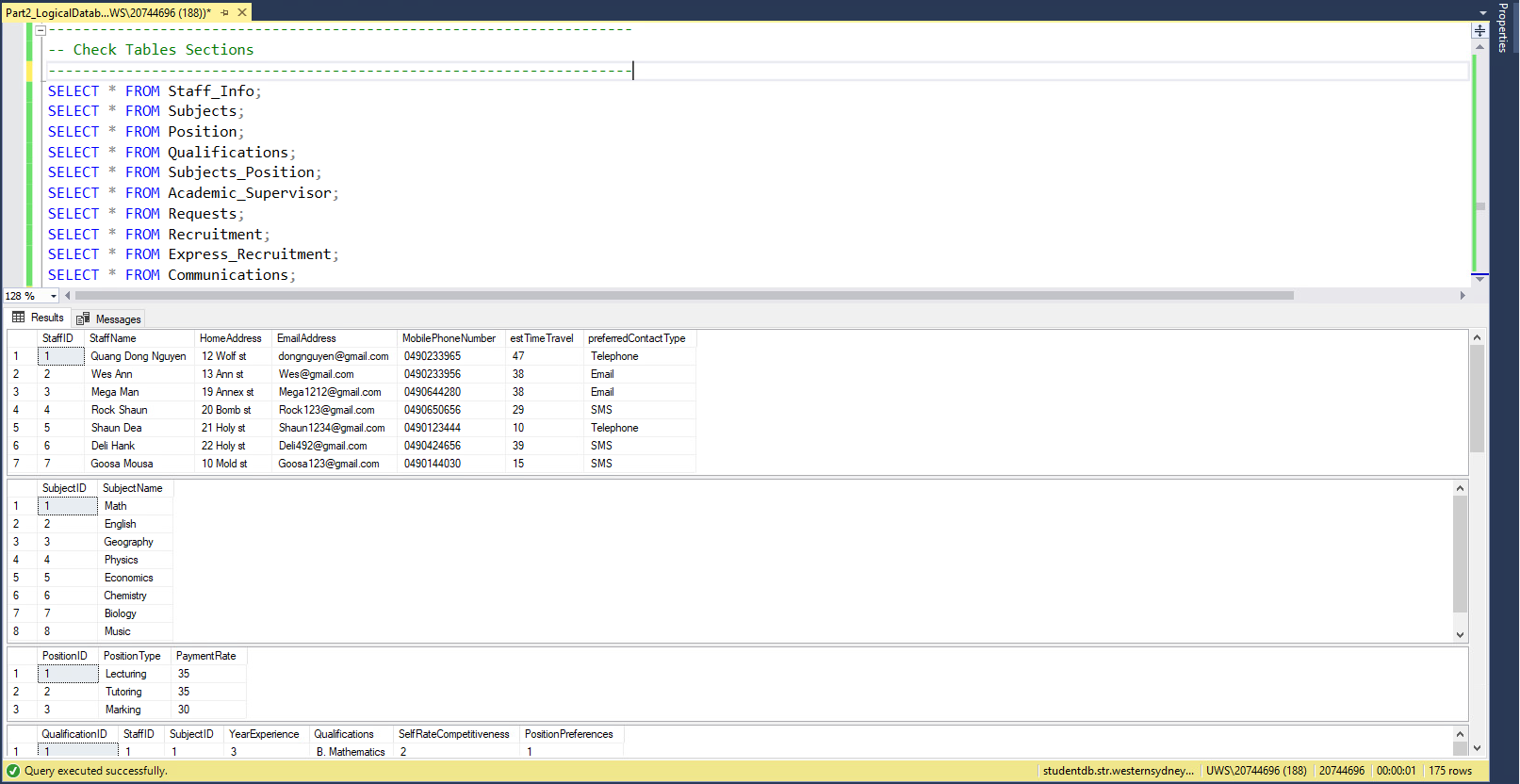
* Between Staff\_Info and Qualifications (1-to-many): Each staff should have at least 1 or many qualifications, since hired staffs should have some basic qualifications before getting hired by the school. And each qualification belongs to only one Casual Staff.
* Between Subjects – Subjects\_Position and Position – Subjects\_Position (1-to-many): Each Subjects and position should be designated to at least 1 Subjects\_position as every subjects positions have a payment rate. And
* Between Staff\_Info and Requests (many-to-many relationships): Each casual staff could apply for zero or many requests and each request could be sent to zero or many people. This is the same for between the Staff\_Info and Express\_Recruitment entities.

**GR Diagram (Global Relation Diagram) of the recruitment system of school:**

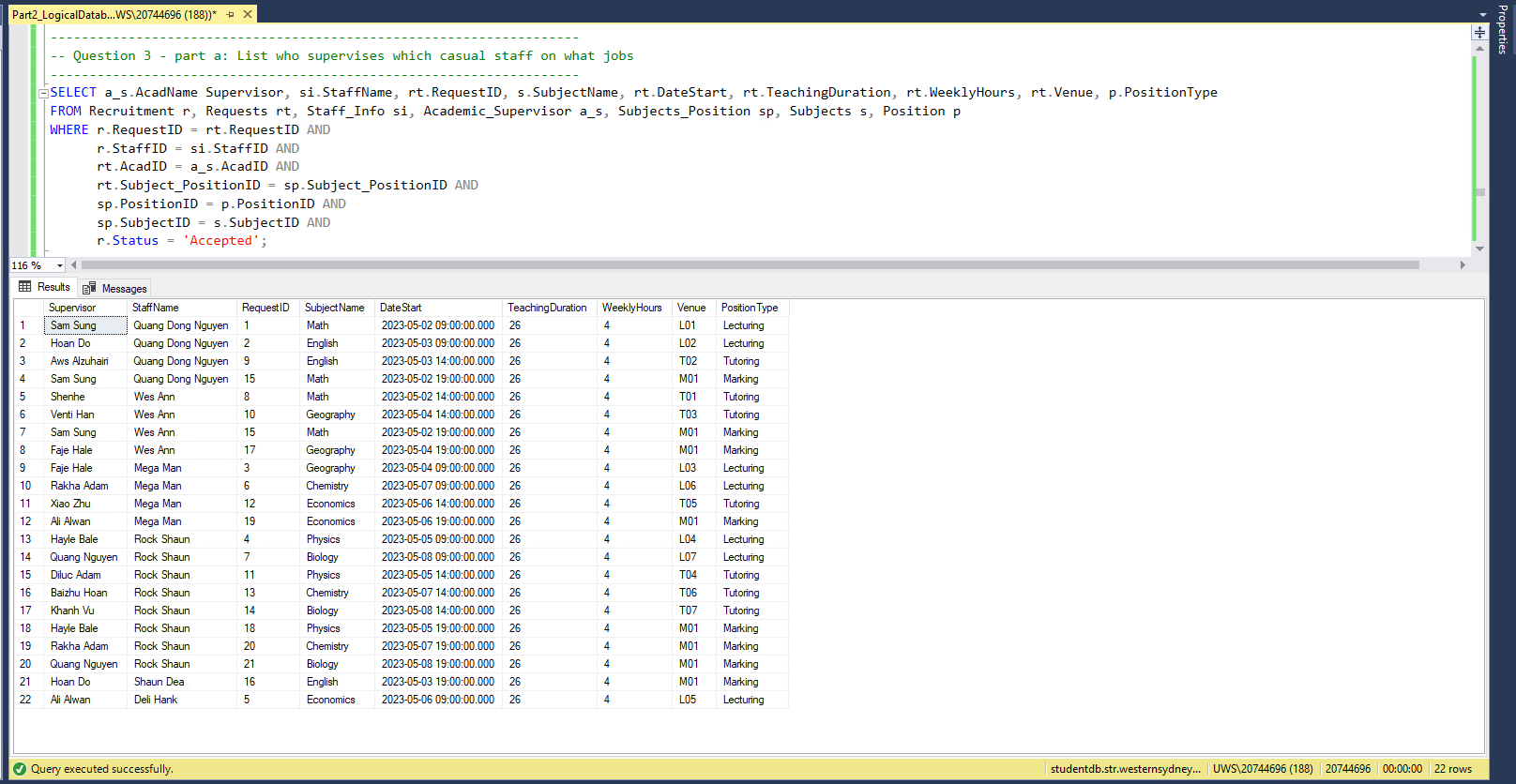


# Question 3

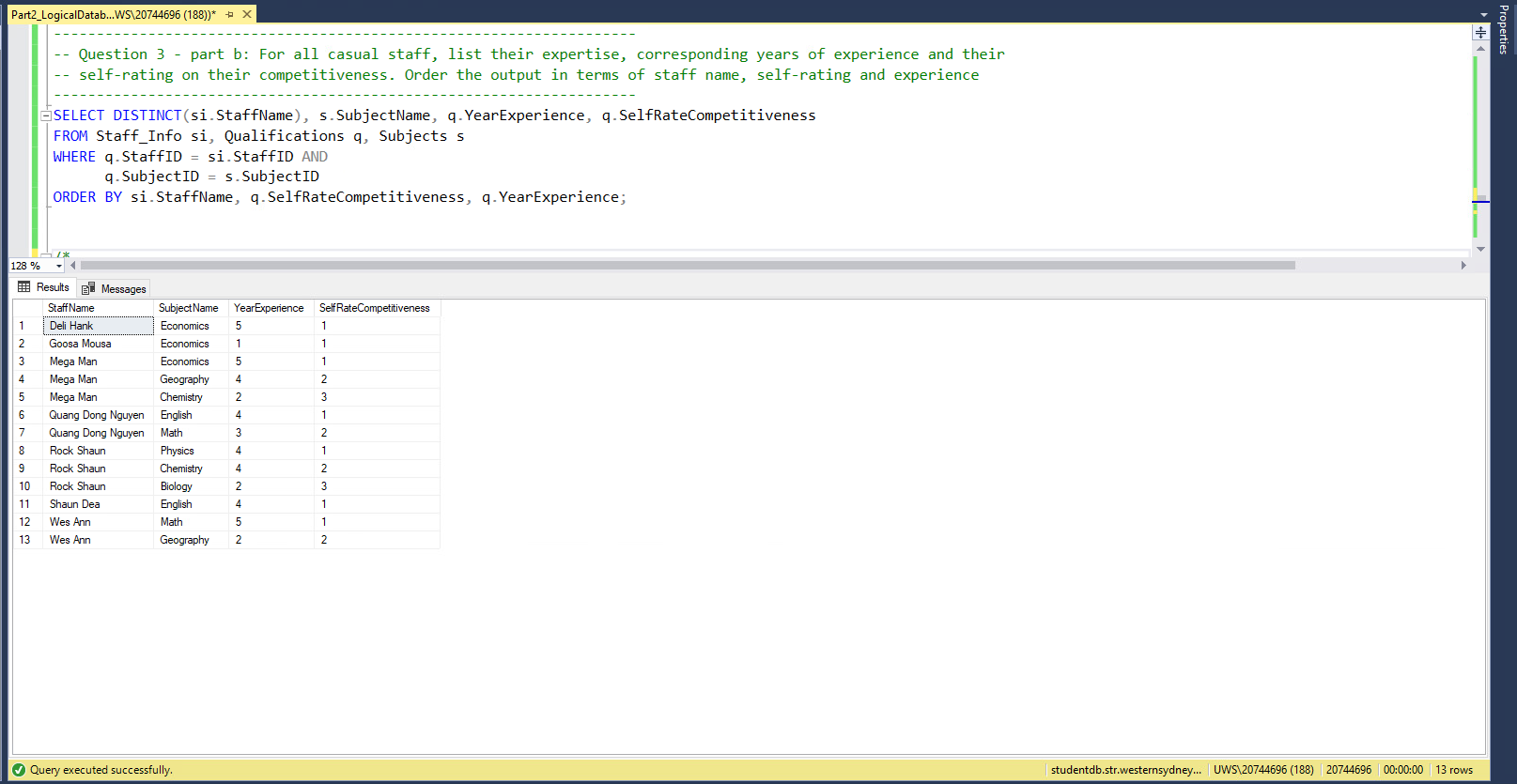
## Question I:



## Question ii. Part a:

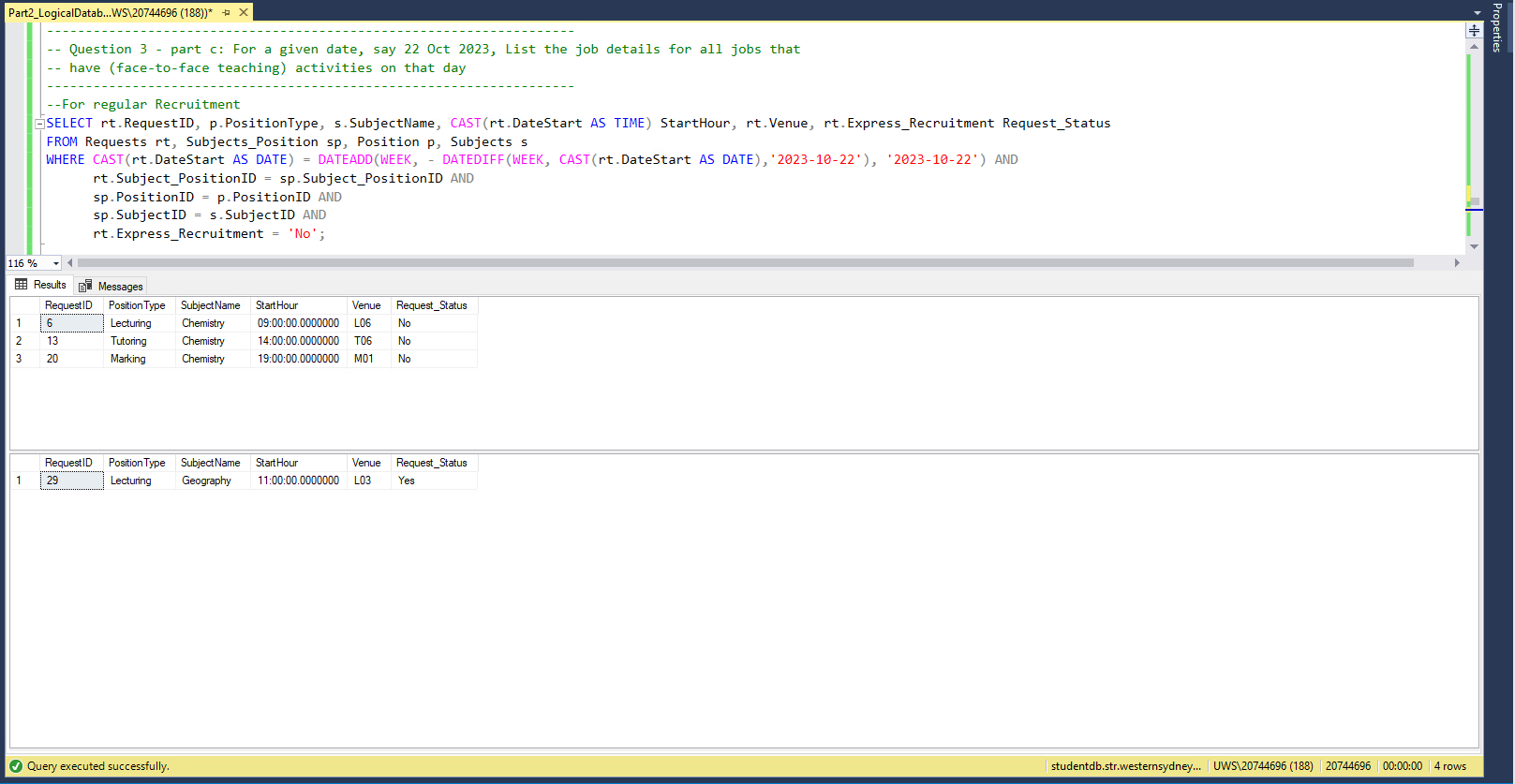


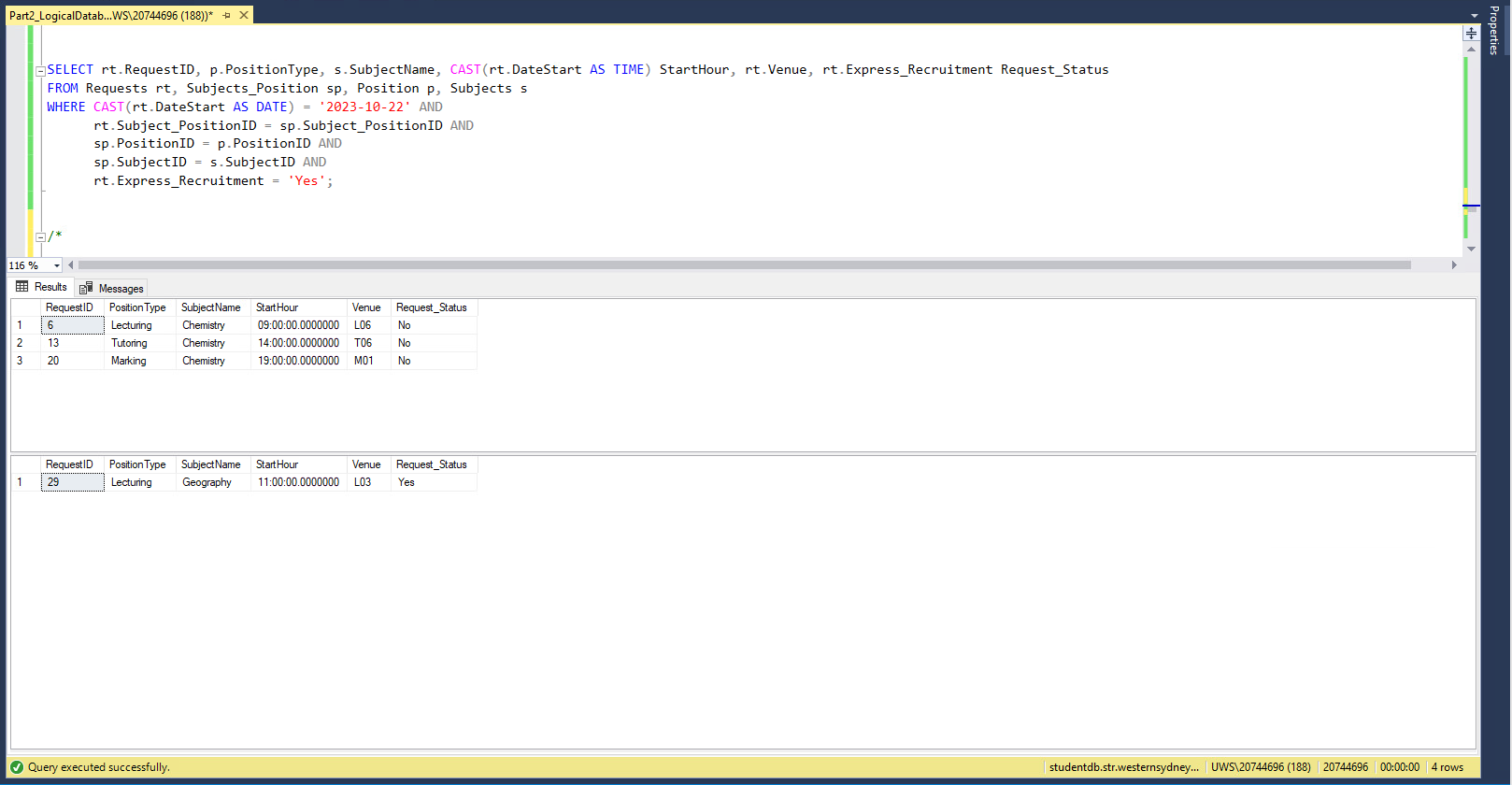
## Question ii. Part b:



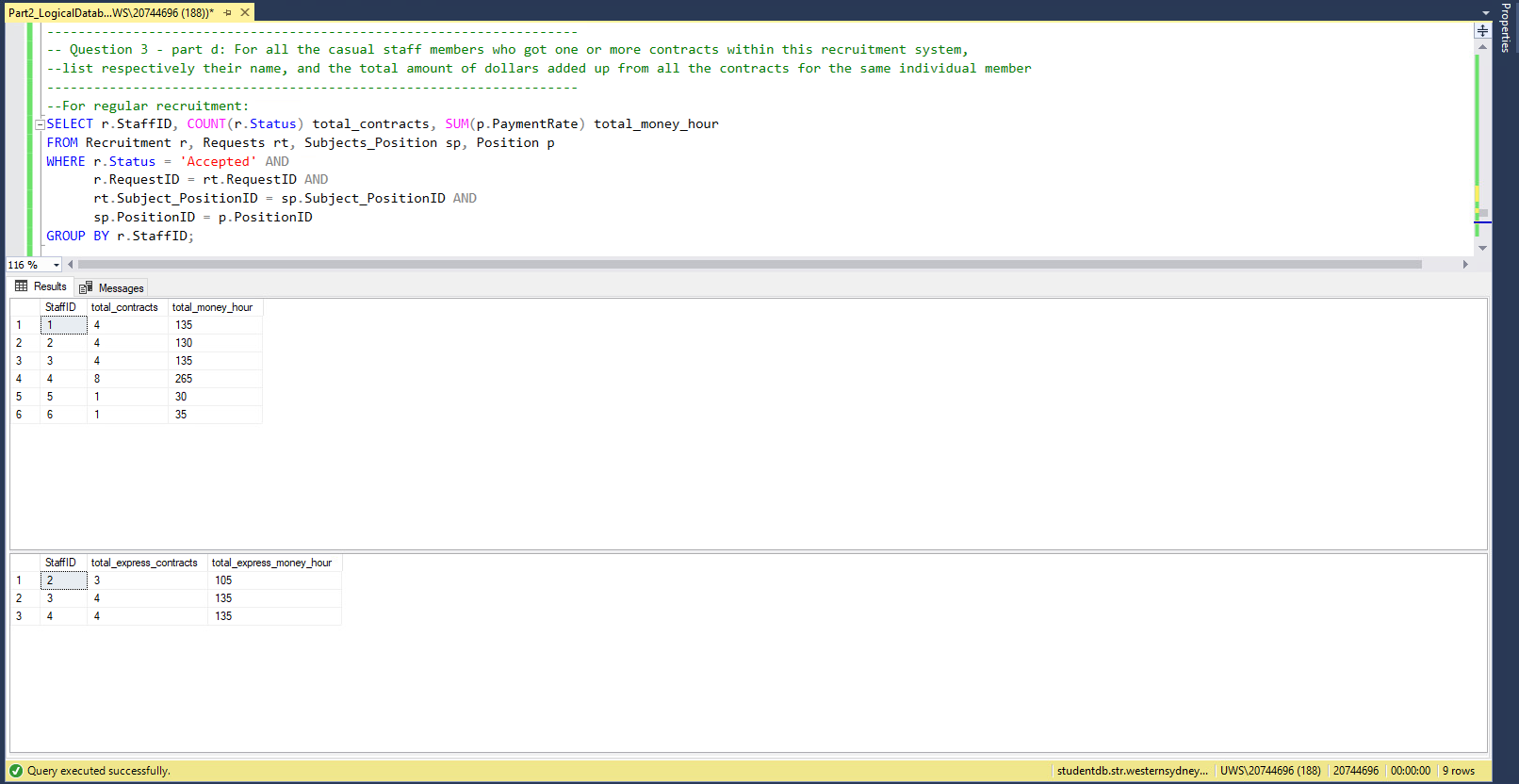
## Question ii. Part c:

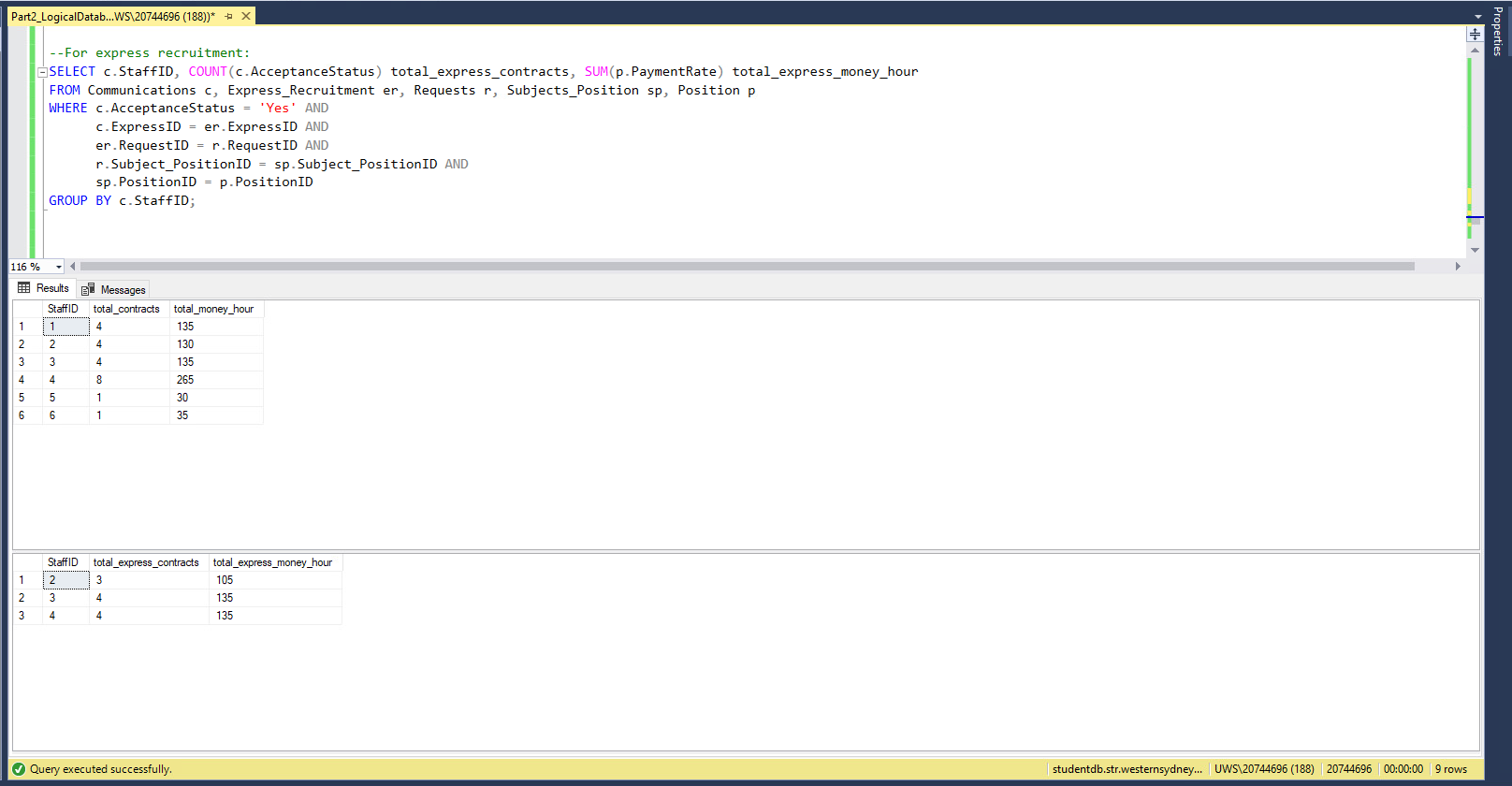
As my database system has two distinctive tables for the recruitment systems (Recruitment for Regular Recruitment and Express\_Recruitment for Express Recruitment). I might use more than one SQL commands to answer the question



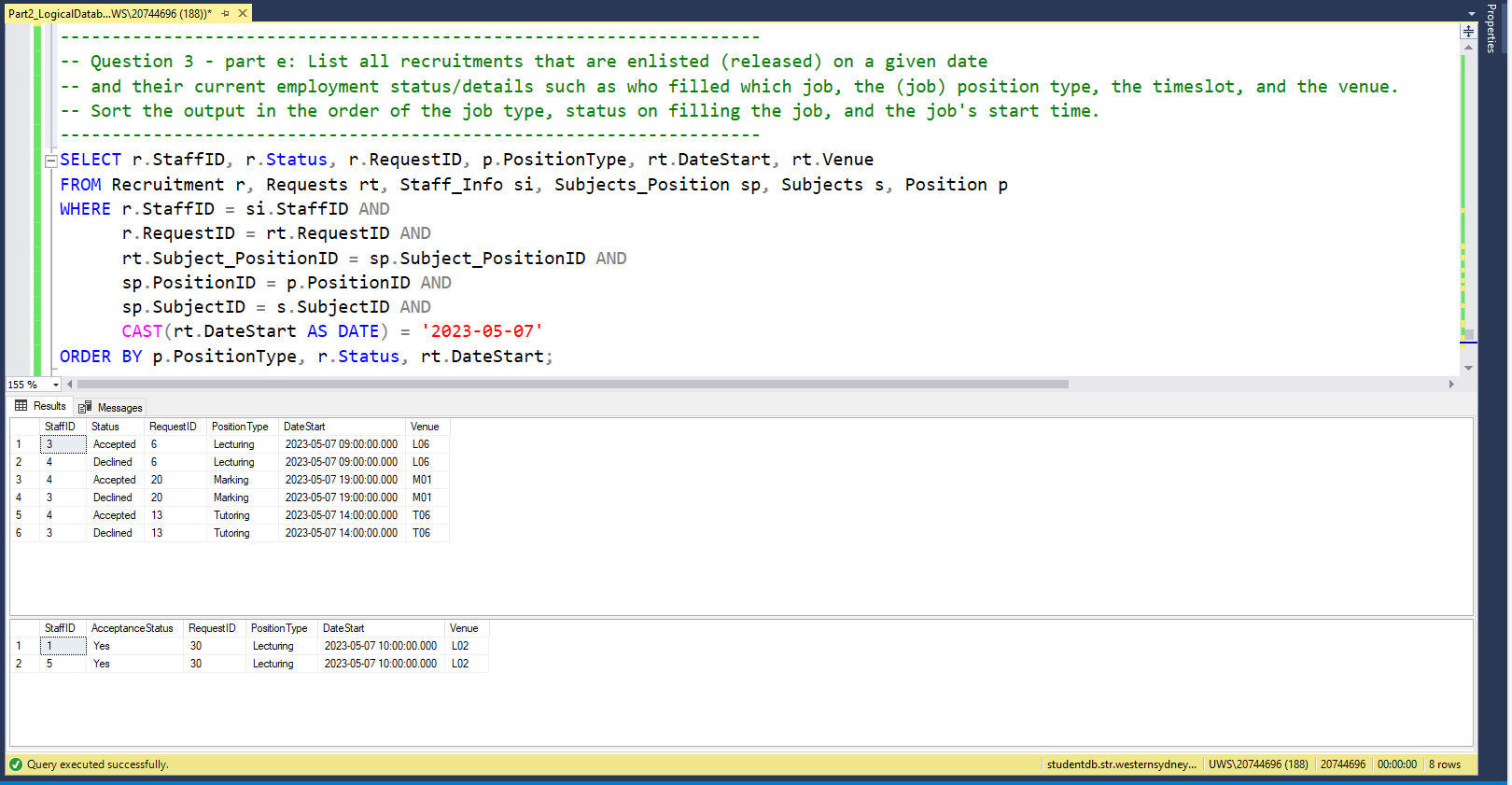


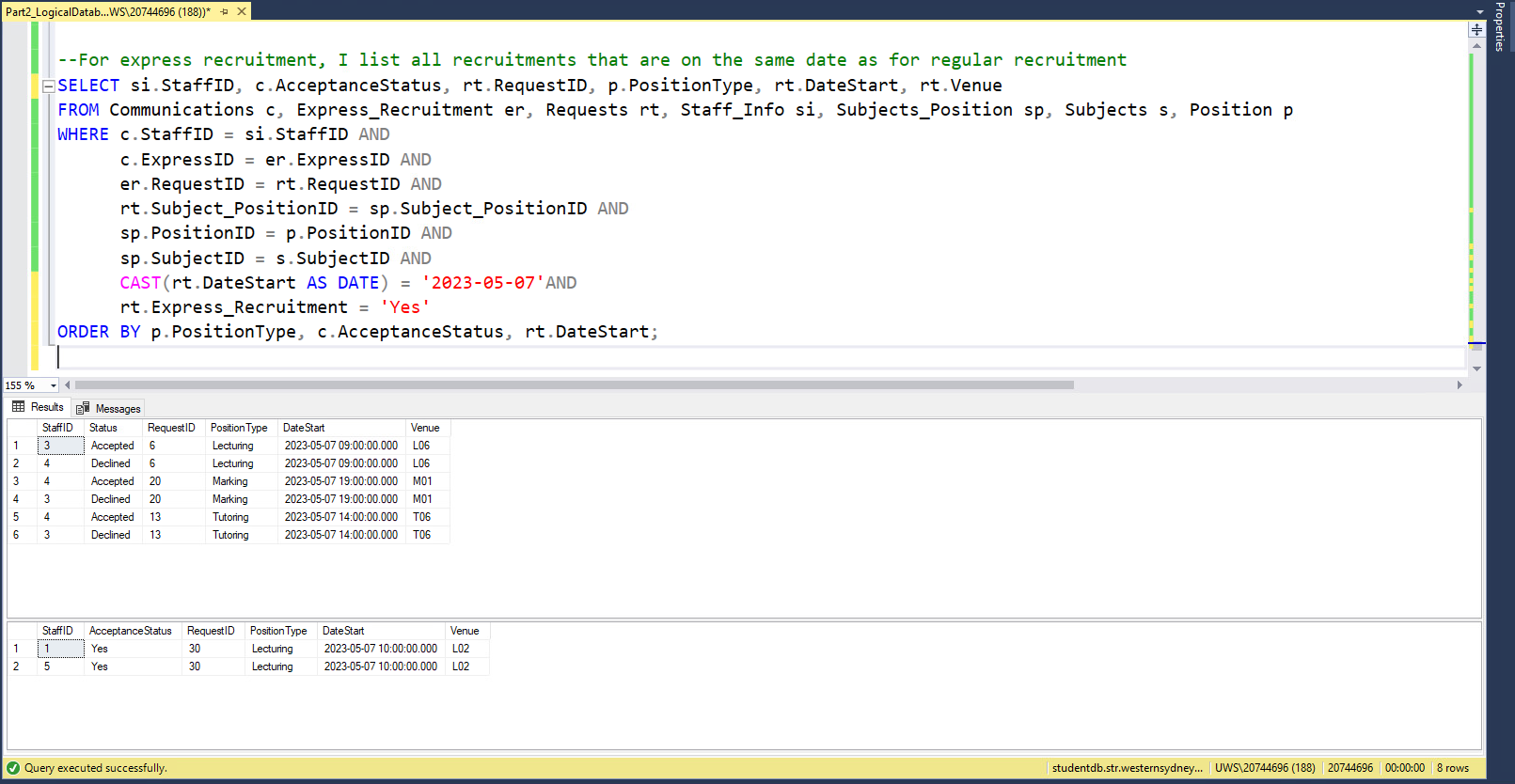
## Question ii. Part d:





## Question ii. Part e:

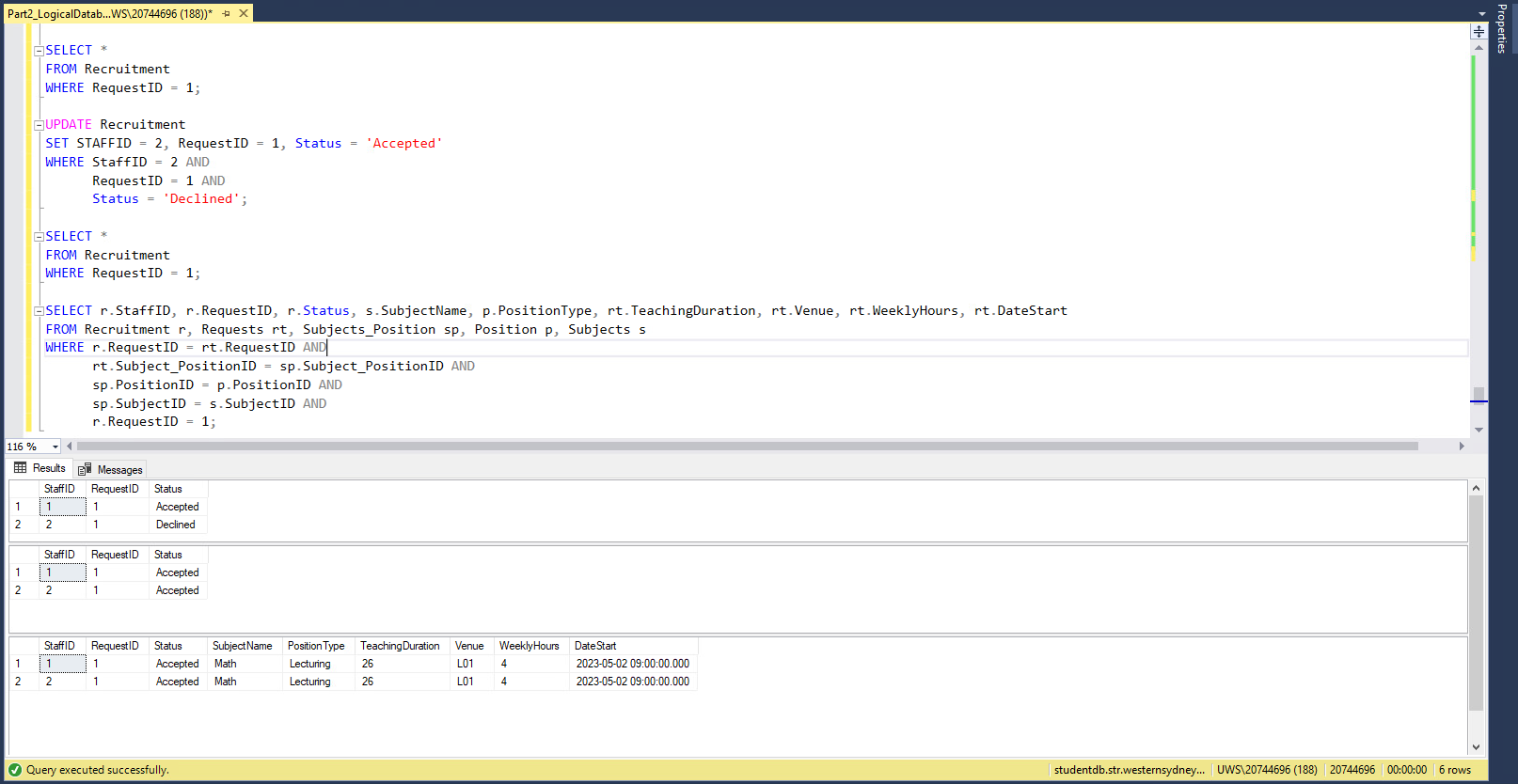




## Question iii

In my Recruitment System, there is a flaw in designing the relationship between the Staff\_Info table and the Recruitment table. It is mentioned that between them there is a many-to-many cardinality relationship, which means a staff can be offered by multiple requests and a request can be sent to multiple staffs. This, however, created a difficult problem in the system to sort who will be assigned to which jobs on which days. Since in my recruitment system, date and times are specified clearly and they are fixed terms, meaning they are constant throughout the term. And because of that, some requests sent to Staffs will have similar days and time, so it tells that many staffs will be offered for the same requests and at the same time. This problem causes a major disruption in data input in. Every result inputted in this table must be manually checked before any value is inserted. For example, in the table Recruitment, Request #1 and #2 is sent to both Staff #1 and Staff #2. Staff #1 is accepted while Staff #2 declined because I have decided so when I created the table. However, in the real-world scenario, both Staffs could have accepted. This would cause conflicts in the final timetable as both staffs will be on the same time, at the same venue and same position. Hence, the database manager eventually would have manually decided who to take on that job for a casual position. That is the situation where I am at. Adding primary keys, foreign keys or even trying different SQL input would not resolve the case completely.

Using SQL as an example:



As shown in the figure above, before the update, there are 2 requests being sent to staff #1 and #2, one Staff declined and the other accepted. However, after the update of values to represent the actual representation of data in the real world, there are two staffs that are accepted to be the lecturer of Math for the next semester. This data does not accurately depict the real situation where two people should not be teaching at the same time. Hence, this database design is not enough to ensure all the data integrity within the business context.