**VIETNAM NATIONAL UNIVERSITY – HCM**

**INTERNATIONAL UNIVERSITY**

**DEPARTMENT OF COMPUTER SICENCE & ENGINEER**

SEMESTER 2 (2022-2023)

**IT079IU**

**PRINCIPAL OF DATABASE PROJECT**

**Topic ID: 12**

**Topic name: E-commerce portal for used furniture sales.**

*May 2023*

**Group 9:**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Full name - Student name** | **Student ID** | **Contribution (%)** |
| 1 | PHAM VU TUYET ANH | ITDSIU21073 | 20 |
| 2 | NGUYEN TRI TIN | ITDSIU21123 | 20 |
| 3 | BUI PHUONG THANH | ITITIU21311 | 20 |
| 4 | LE THI THUY NGA | ITDSIU21024 | 20 |
| 5 | DANG THAI SON | ITDSIU21115 | 20 |

**Table of Contents**

[I. INTRODUCTION 5](#_Toc133997410)

[1 Background 5](#_Toc133997411)

[2 Summary idea 5](#_Toc133997412)

[3 The goal of topic 5](#_Toc133997413)

[II. THEORETICAL BASIS 6](#_Toc133997414)

[1 ERD- EERD (Entity - Relationship Diagram - Enhanced entity-relationship model) 6](#_Toc133997415)

[1.1 Definition and Purpose 6](#_Toc133997416)

[1.2 Component of ERD 7](#_Toc133997417)

[1.3 Way to design ERD - EERD 10](#_Toc133997418)

[**2** Database normalization 11](#_Toc133997419)

[2.1 Definition. 11](#_Toc133997420)

[2.2 Process 11](#_Toc133997421)

[**3** Convert ERD model to relational model. 11](#_Toc133997422)

[**4** SQL server 12](#_Toc133997423)

[4.1 Basic SQL 12](#_Toc133997424)

[4.2 Data type 14](#_Toc133997425)

[4.3 Constraint: 16](#_Toc133997426)

[5 Querying database 18](#_Toc133997427)

[6 Connecting Java and SQL server database 27](#_Toc133997428)

[6.1 Database 27](#_Toc133997429)

[6.2 Java 27](#_Toc133997430)

[III. RELATIVE WORKS. 30](#_Toc133997431)

[1 Analyst requirement of topic 30](#_Toc133997432)

[2 Design ERD 30](#_Toc133997433)

[3 Convert ERD model to relational model. 37](#_Toc133997434)

[4 Normalize 37](#_Toc133997435)

[5 Create database in SQL server. 38](#_Toc133997436)

[6 Collect & Insert data. 40](#_Toc133997437)

[7 Querying database 42](#_Toc133997438)

[8 Create View 53](#_Toc133997439)

[9 Connect SQL server with Java. 54](#_Toc133997440)

[10 DEMO 58](#_Toc133997441)

[IV. CONCLUSION 61](#_Toc133997442)

[1 Summary 61](#_Toc133997443)

[2 What we learn after project 61](#_Toc133997444)

[3 Completed work & Future works. 61](#_Toc133997445)

[V. REFERENCES 63](#_Toc133997446)

**Table of Figures**

**Figure II‑1 Example of ERD** 7

**Figure II‑2 Example of EERD** 8

**Figure II‑3 Notation in ERD & EERD** 10

**Figure II‑4 Example of Specialization** 11

**Figure II‑5 Example of Generalization** 11

**Figure II‑6 String Data Types** 15

**Figure II‑7 Date and Time Data Types** 15

**Figure II‑8 Numeric Data Types** 16

**Figure II‑9 Numeric Data Types** 16

**Figure II‑10 Other Data Types** 17

**Figure II‑11 Source code to connect** 30

**Figure III‑1 ERD Diagram** 37

**Figure III‑2 Relational Schema** 37

**Figure III‑3 Database Diagram** 38

**Figure III‑4 Admin code** 42

**Figure III‑5 Admin values** 42

**Figure III‑6 Customer code** 42

**Figure III‑7 Customer values** 42

**Figure III‑8 Employees code** 42

**Figure III‑9 Employees values** 43

**Figure III‑10 Display Project file and Libraries** 55

**Figure III‑11 Design Form** 56

**Figure III‑12 Rename Text Field** 56

**Figure III‑13 Rename Button** 57

**Figure III‑14 Rename Text Table** 57

**Figure III‑15 Main class** 58

**Figure III‑16 Interface form to Query** 58

**Figure III‑17 Source code to connect & query** 59

**Figure III‑18 Query Demo 1** 59

**Figure III‑19 Query Demo 2** 60

**Figure III‑20 Query Demo 3** 60

**Figure III‑21 Query Demo 4** 61

**Figure III‑22 Query Demo 5** 61

**Table of Tables**

**Table II.1 SQL description and syntax** 13

**Table II.2 Constraint description and syntax** 17

**Table II.3 Query syntax** 19

**Table III.1 Description of ERD** 31

**Table III.2 Define relationship and map cardinality.** 36

**Project implementation plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Timeline** | **Task** | **Responsibility** | **Note** |
| 1 | 10/02 – 14/02 | Analyze the issue | All of members |  |
| 2 | 15/02 – 18/02 | Create ERD | Tuyet Anh | Partner:  Thai Son |
| 3 | 15/02 – 18/02 | Convert ERD to Relational model | Tuyet Anh |  |
| 4 | 15/02 – 18/02 | Normalize | Tuyet Anh |  |
| 4 | 18/02 – 20/02 | Feedback & Change ERD | All of members |  |
| 5 | 21/02 – 26/02 | Create Database in SQL | Thai Son |  |
| 6 | 26/02 – 27/02 | Feedback & Change | All of members |  |
| 7 | 28/02 – 4/3 | Collect & Insert data | Tri Tin |  |
| 8 | 4/3 – 2/5 | Connect Java to SQL -> Design form | Thuy Nga & Phuong Thanh | Partner: Tuyet Anh |
| 9 | 14/3 – 20/4 | Write Report | All of members |  |
| 10 | 20/04 – 27/04 | Prepare file presentation | All of members |  |
| 11 | 27/04 – 3/5 | Preview + Demo | All of members |  |

# INTRODUCTION

## Background

Furniture has always played an important role in every space. Not only does it beautify the area, but also provides comfort and is functional that fits its owner's needs. There is a variety of furniture available in the market today consisting of different sizes, shapes, colors, etc. However, for some people, investing in new quality furniture products is an expensive affair, that is why nowadays more and more people start taking second-hand furniture into consideration. Buying used products can be tricker than new ones, they require the buyers to have a certain experience in checking and examining the item's condition. Moreover, since it is based on the item’s condition, the prices can be varied and unpredictable. Understanding the problems with used furniture and the future potential business, our group project about creating an E-commerce platform has been developed aimed to help enhance both buyer and seller experience with second-hand products.

## Summary idea

E-commerce portal for used furniture sales. This project is for a startup that is acquiring used furniture from users at a price, refurbishing it and selling it off at margin. The result of the project can help company to store and manage the total of second-hand furniture that is currently being sold/ bought, the total customer of the company, the employee, sales revenue from transactions, …

## The goal of topic

Our team got the chance to study SQL and use the programming abilities they had acquired by building a database model of a furniture company. Using the information that has been taught in ways like:

* Analyze data requirements.
* Define The entities, attributes, relationship,..
* Insert data.
* Data query
* Connect database and interface design by using java.

Also, the creation of a system for the liquidation and sale of secondhand furniture will inevitably satisfy the demand and address the demand. The final goal of this project is to render this system efficient by enhancing the appraisal system, customer experience, global sourcing, and, most crucially, market share expansion to draw in more transactions.

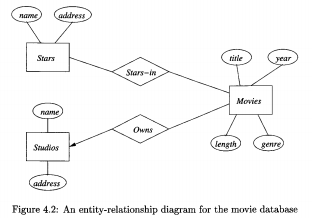
# **THEORETICAL BASIS**

## ERD- EERD (Entity - Relationship Diagram - Enhanced entity-relationship model)

### Definition and Purpose

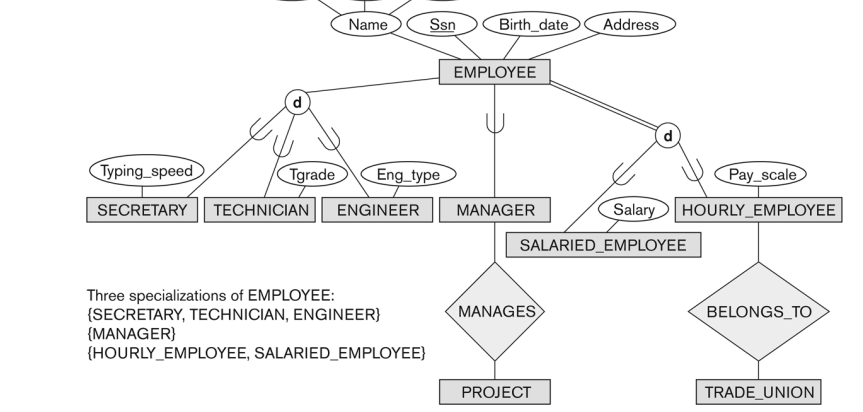
1. **ERD**

An Entity - Relationship Diagram, ERD, is a visual form of relational database that represents entity sets, attributes and relationships. Rectangles, rhombuses, ovals, and connecting lines are just a few of the notations used by ERD to represent the union of organizations, their connections, and their characteristics, which we will show later. By enabling the definition of an enterprise schema, which reflects the general logical structure of a database, ERD makes database design easier. Moreover, it can be used to transfer the meanings and relationships of real-world businesses onto a mental framework. One of ERD examples:



**Figure II‑1 Example of ERD**

1. **EERD**

EER diagrams are essentially an expanded form of entity-relationship (ER) diagrams. EER models are useful tools for creating high-level model databases. By carefully examining the qualities and limitations, you may plan databases more completely thanks to their improved features.

**Figure II‑2 Example of EERD**

### Component of ERD

1. **Entity - Entity set.**

An entity is a “thing” or “object” in the real word that is distinguishable from all other objects. For example, each employee in the company is an entity, each car in the garbage is an entity. Entity set is a collection of entities of the same type that share the same properties or attributes. Notation of the entity set is rectangle. For example, an entity set name as “student” collects all students in class.

There are two main types of entities: Strong entity set, and weak entity set. Alongside weak entity set: a weak entity set is one whose existence is dependent on another entity set to exist, called its identifying entity set. Weak entity sets do not have a primary key, it is identified by discriminator attributes. Therefore, all weak entities depend on identifying entity set. Weak entity set is depicted via a double rectangle and has an identifying relationship. Strong entity sets are the inverse; they can be differentiated by the main key and are not entirely reliant on other entities.

1. **Attribute**

Attributes are properties of the entities in the entity set presented as oval. For each attribute, there is a set of permitted values, called the domain, or value set, of that attribute. An example for this, a student has some properties like student\_name, student\_id, day of birth, ... There are many types of attributes:

Normal attribute or simple attribute: “the attributes have been simple; that is, they have not been divided into subparts”. Primary attribute or identifying attribute that used to identify each entity in the entity set. Therefore, there are no situations that have duplicate values in the primary attribute. Besides that, the primary key can refer to another entity set, foreign attribute, which helps to identify relationships between entity sets. However, foreign attributes need not be unique, because a lot of records can share the same value.

Multivalued attributes can include many values in a record. For example, customers have phone attributes, and one customer can have many phone numbers, therefore, phone attributes in each customer can include one or many phone numbers.

Different from single attribute, composite attribute can divide into some subclass. For instance, a Full name attribute can be divided into 3 single attributes: First name, Middle name, Last name. Derived attributes do not exist in the physical database, its value made by other attributes. Furthermore, attributes can exist on relationship that is useful to attach an attribute to a relationship.

1. **Relationship and Identifying relationship**.

Relationships are connections among two or more entity sets. A relationship set is a set of relationships of the same type. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship… In general, a weak entity set must have a total participation in its identifying relationship set, and the relationship is many-to-one toward the identifying entity set.

1. **Cardinality**

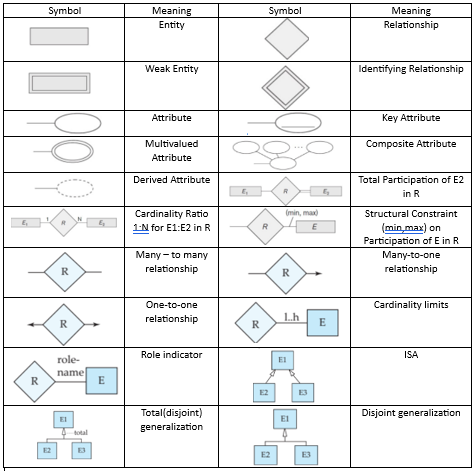
One-to-one. An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One-to-many. An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.

Many-to-one. An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.

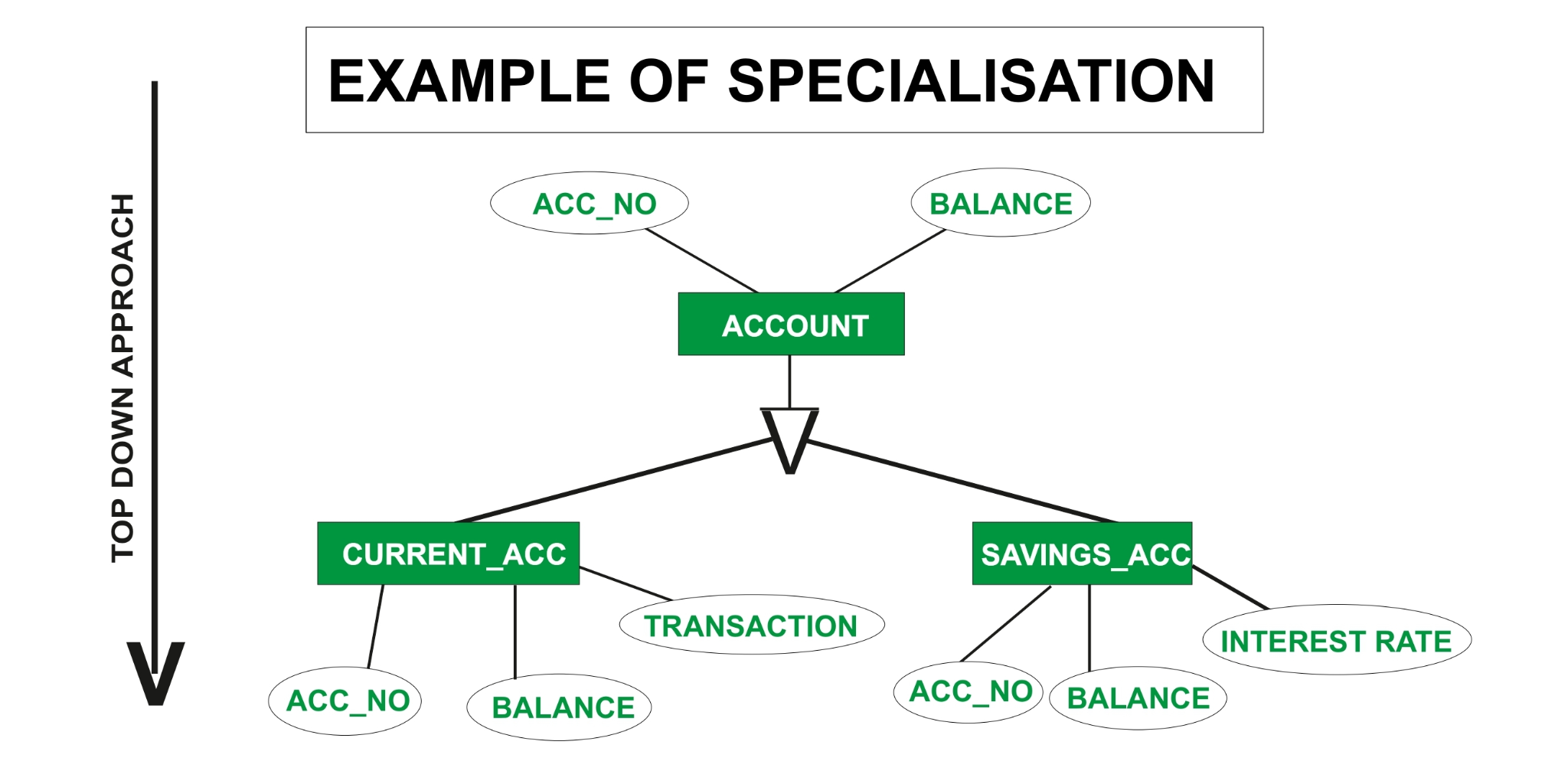
Many-to-many. An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*.

1. **Notation.**



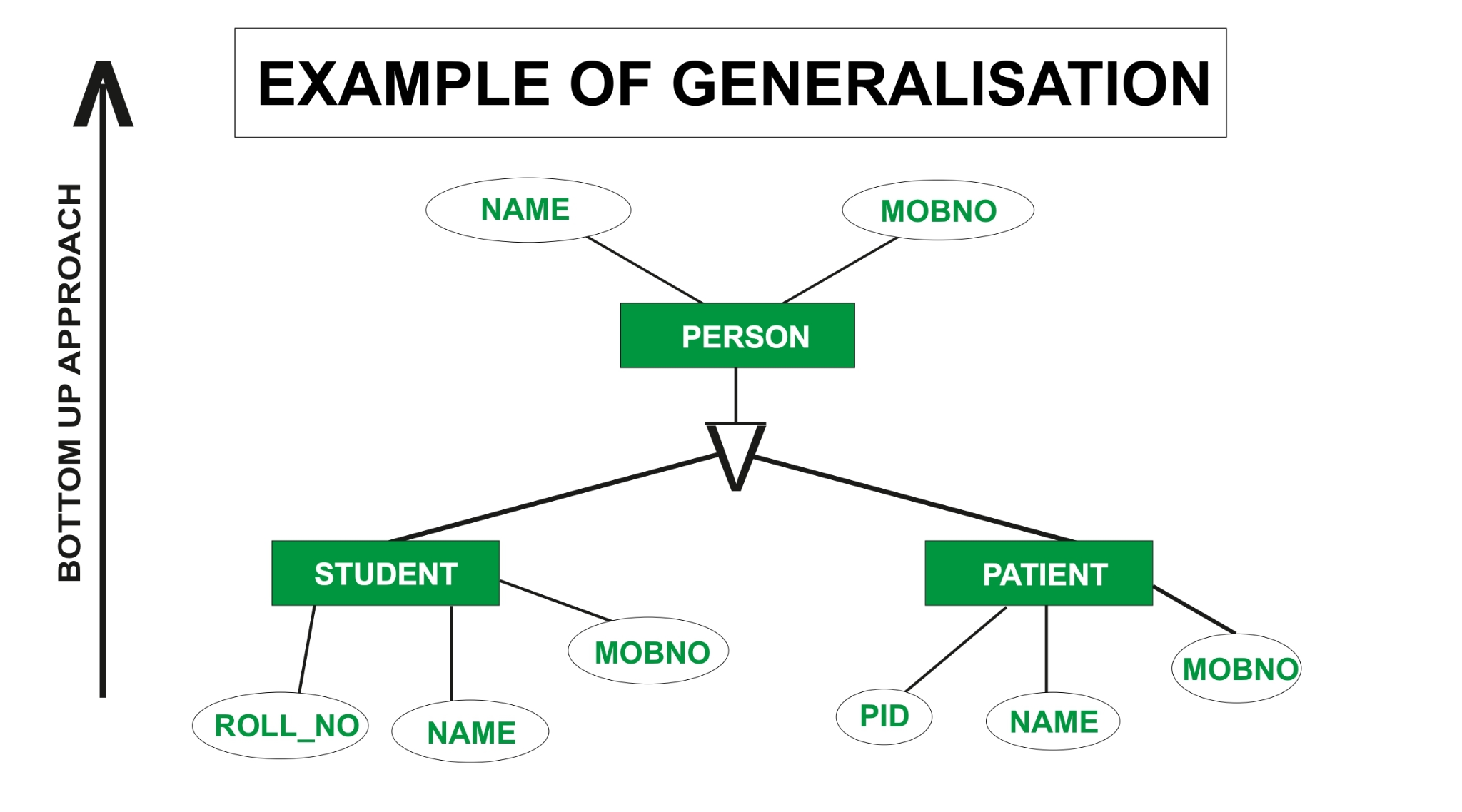
**Figure II‑3 Notation in ERD & EERD**

1. **Specialization and Generalization**

**Specialization:** Specialization is the process of defining a set of subclasses of an entity type; this entity type is referred to as the super-class of the specialization. The collection of subclasses that make up a specialization are determined by one or more characteristics that separate the entities in the super-class apart. Clearly, the top-down specialization technique divides a higher-level entity into numerous specialized lower-level entities. 

**Figure II‑4 Example of Specialization**

**Generalization:** We may imagine a reverse abstraction process in which we conceal the distinctions between numerous entity types, find their common properties, and generalize them into a single super-class, of which the original entity types are special subclasses. Exactly, generalization maps or group types or classes to more abstract or combined ones. Notice that the generalization process can be viewed as being functionally the inverse of the specialization process.



**Figure II‑5 Example of Generalization**

### Way to design ERD - EERD

* Step 1: After analyzing the requirement, we can list the number of entities that you have to use. Alongside entities, we already can identify attributes to accept requirements of a topic.
* Step 2: Connecting all entities together by relationship.
* Step 3: Identifying and mapping attributes to entities, relationships.
* Step 4: Determine the value range for each attribute.
* Step 5: Determine key attribute for each entity.
* Step 6: Identify cardinality for each relationship: One - many, many - many, one - one whose step is called Cardinality identification.
* Step 7: If ERD has generalized/specialized constraints, please identify them. It means hierarchical design of EERD.

## Database normalization

### Definition.

Normalization of data may be thought of as a process of examining the provided relation schemas based on their FDs and primary keys. It may be viewed as a "filtering" or "purification" process to improve the quality of the design over time. An unsatisfactory relation schema that fails to satisfy the normal form test is divided into smaller relation schemas that each include a subset of the attributes and pass the test that the original relation failed to satisfy. The purpose of normalization is (1) make the schema informative (2) Minimize information anomalies (3) Avoid modification anomalies (4) Disallow spurious tuples.

### Process

1. **1NF – First Normal Form:**

Any attribute in a tuple must have a single value from the domain of that attribute, and the domain of an attribute must only include atomic values.

1. **2NF – Second Normal Form.**

If a relation is in first normal form and every non-key characteristic is completely functionally dependent on the primary key, it is said to be in second normal form.

1. **3NF – Third Normal Form**

A table design is said to be in 3NF if table must be in 2NF and transitive functional dependency of non-prime attribute on any super key should be removed.

## Convert ERD model to relational model.

Step 1: Convert Strong entity sets: create a corresponding relation that includes all the simple attributes (includes simple attributes of composite relations). Choose one of the key attributes as primary. If composite, the simple attributes together form the primary key. Any remaining key attributes are kept as secondary unique keys (these will be useful for physical tuning with reference to indexing analysis)

Step 2: Convert Weak entity sets: create a corresponding relation that includes all the simple attributes. Add as a foreign key all the primary key attribute(s) in the entity corresponding to the owner entity type. The primary key is the combination of all the primary key attributes from the owner and the partial key of the weak entity, if any

Step 3: Convert mapping binary 1 - to -1. Choose one relation as S (total participation). Add to S all the simple attributes of a relationship. Add as a foreign key in S the primary key attributes of T.

Step 4: Binary 1 - to - N. Choose S as the type at the N - side of the relationship. Add as a foreign key to S all the primary key attributes of T.

Step 5: Binary M to N: Create new relationship relation. Add as foreign keys the primary keys of both relations, which combination forms the primary key of relationship relation and add any attribute on this relation.

Step 6: Convert multivalued attributes. Create a new relation S. Add as foreign keys the primary keys of the corresponding relation. Add the attribute to S, the combination of all attributes in S forms the primary keys.

Step 7: Specialization/Generalization

* Multiple relations – if there are subclass and super-class, then usually works with assumes unique id at parent. If subclass only, the relation should be used for disjoint.
* Single relation with one type of attribute and only for disjoint, we can result in many NULLs. On the other hand, relation with multiple type attributes, we could be disjoint.

## SQL server

### Basic SQL

**Table II.1 SQL description and syntax**

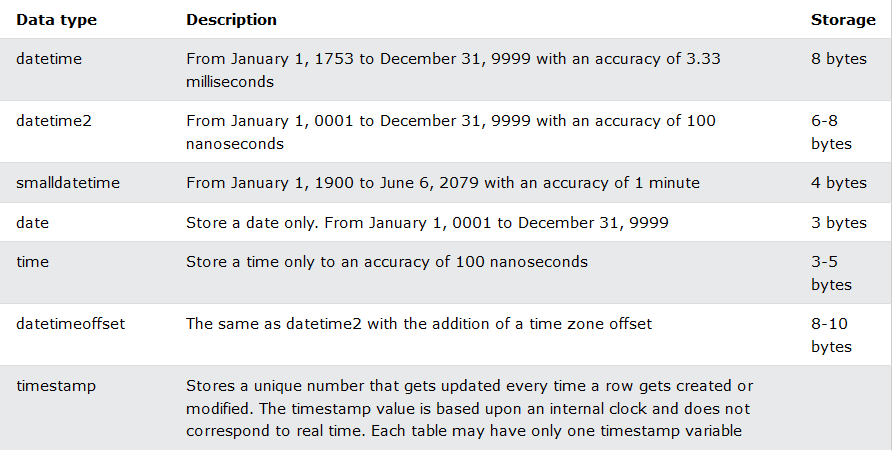
|  |  |  |
| --- | --- | --- |
|  | **Description** | **Syntax** |
| **DATA DEFINITION LANGUAGE (DDL)** | | |
| **CREATE** | Create tables or databases. While creating a table, we specify table names, column names, datatype of each column, and column sizes. If there are any integrity constraints or key constraints of the table like PRIMARY KEY, UNIQUE, NOT NULL, etc. we must specify them while creating it along with the column names and details. | CREATE DATABASE databasename;  CREATE TABLE tablename  (column1 datatype, colum2 datatype,...); |
| **ALTER** | Change the structure of an existing database table. Using this command, we can perform operations like adding a new column, removing any column, adding or removing integrity constraints, or changing the data type of the existing column on the existing table. ALTER command is used to modify the existing structure of the table. | ALTER TABLE table\_name ; |
| **DROP** | Delete an existing table completely from the database. This command deletes the table records along with their structure. | DROP TABLE table\_name |
| **DATA MANIPULATION LANGUAGE (DML)** | | |
| **INSERT** | Insert data into the rows of a table. | INSERT INTO table\_name  (column1, column2,..column\_n)  VALUES(value1, value2,..value\_n); |
| **UPDATE** | It allows us to update or modify the existing data in tables. It changes the data from one or more records in a table. | UPDATE table\_name  SET col\_1 = value\_1, col\_2 = value\_2,..  WHERE condition; |
| **DELETE** | It allows us to remove single or multiple records from the database tables depending upon the condition we specify in the WHERE clause. | DELETE FROM table\_name WHERE condition; |

### Data type

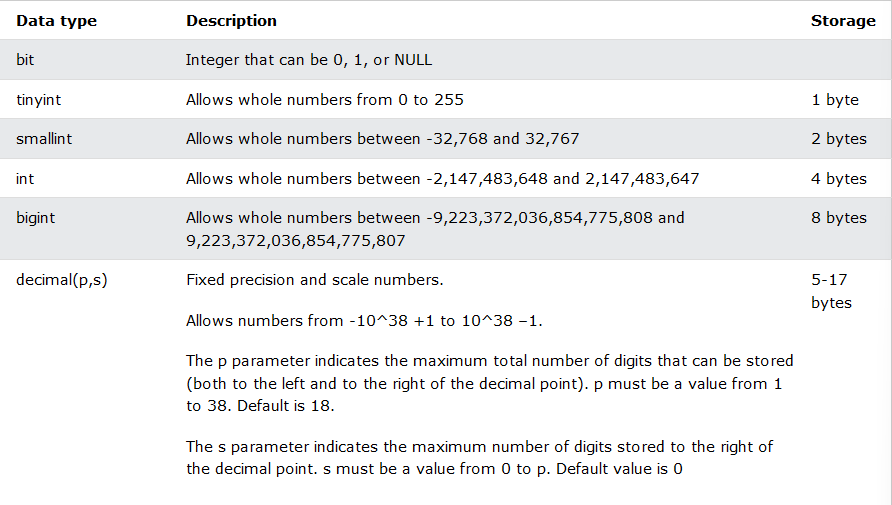
**Table

Description automatically generated**

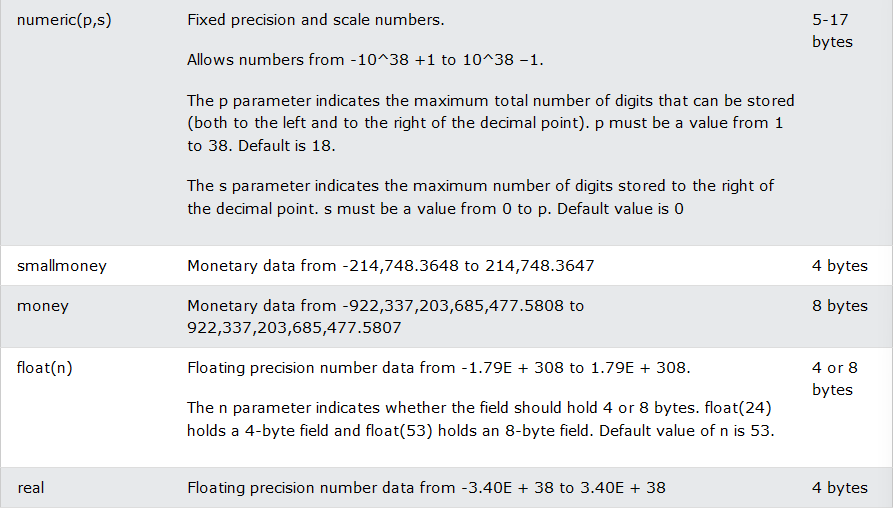
**Figure II‑6 String Data Types**



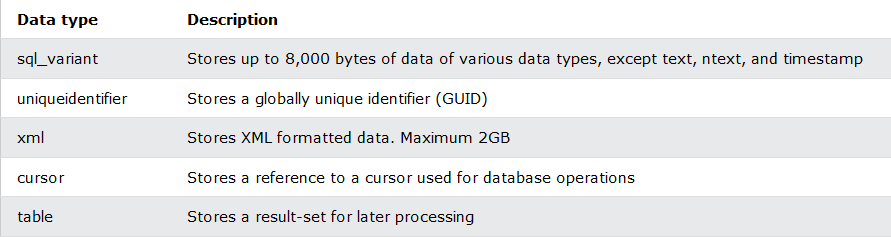
**Figure II‑7 Date and Time Data Types**



**Figure II‑8 Numeric Data Types**



**Figure II‑9 Numeric Data Types**



**Figure II‑10 Other Data Types**

### Constraint:

**Table II.2 Constraint description and syntax**

|  |  |  |
| --- | --- | --- |
| **Constraint** | **Description** | **Syntax** |
| **NOT NULL** | Ensures that a column cannot have a NULL value. | On create table:  CREATE TABLE Persons (  ID int NOT NULL,  LastName varchar(255) NOT NULL,  FirstName varchar(255) NOT NULL,  Age int );  On alter table:  ALTER TABLE Persons  ALTER COLUMN Age int NOT NULL; |
| **UNIQUE** | Ensures that all values in a column are different. | On create table:  CREATE TABLE Persons(  ID int NOT NULL,  LastName varchar(255) UNIQUE);  Another way  ALTER TABLE Persons  ADDCONSTRAINT uniquename UNIQUE(column\_name); |
| **PRIMARY KEY** | A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table. | On create table:  CREATE TABLE Person (  ID int PRIMARY KEY,  Name varchar(255) NOT NULL);  On alter table:  ALTER TABLE Persons  ADD PRIMARY KEY (ID); |
| **FOREIGN KEY** | Prevents actions that would destroy links between tables | On create table:  CREATE TABLE Person (  ID int PRIMARY KEY,  Name varchar(255) NOT NULL  OrderID FOREIGN KEY REFERENCES Order(OrderID));  On alter table:  ALTER TABLE Person  ADDFOREIGNKEY(OrderID)REFERENCES Order(OrderID) |
| **CHECK** | Ensures that the values in a column satisfies a specific condition. | On create table:  CREATE TABLE Person (  ID int PRIMARY KEY,  Name varchar(255) NOT NULL  Age int CHECK (Age >= 18));  On alter table:  ALTER TABLE Persons  ADD CHECK( Age >=18); |
| **DEFAULT** | Sets a default value for a column if no value is specified. | On create table:  CREATE TABLE Person (  ID int PRIMARY KEY,  Name varchar(255) NOT NULL  City varchar(255) DEFAULT ‘Sandnes’);  On alter table:  ALTER TABLE Persons  ADD CONSTRAINT df\_City DEFAULT ‘Sadnes’ FOR City; |
| **CREATE INDEX** | Used to create and retrieve data from the database very quickly | CREATE INDEX index\_name  ON tablename (column1, column2,...); |

## Querying database

**Table II.3 Query syntax**

|  |  |  |
| --- | --- | --- |
|  | **Description** | **Syntax + Example** |
| **SELECT** | Fetch the data from a database table which returns this data in the form of a result table | SELECT column1,..column\_n FROM table\_name;  Example:  SELECT ID, NAME FROM CUSTOMER |
| **WHERE** | Clause specifies a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table | SELECT column1,...column\_n FROM tablename  WHERE[condition];  Example  SELECT ID, Name FROM Customer  WHERE Salary>2000; |
| **GROUP BY** | To group similar data into groups, the SELECT statement and the SQL GROUP BY clause are used | SELECT column1,column2  FROM table\_name  WHERE [ conditions ]  GROUP BY column1, column2  Example  SELECT NAME, SUM(SALARY)  FROM CUSTOMERS  GROUP BY NAME; |
| **HAVING** | Condition for Aggregate | SELECT column1, column2 FROM table1, table2  WHERE [ conditions ]  GROUP BY column1, column2  HAVING [ conditions ]  ORDER BY column1, column2;  Example  SELECT ID, NAME, AGE, ADDRESS, SALARY  FROM CUSTOMERS  GROUP BY age  HAVING COUNT(age) >= 2; |
| **ORDER BY** | Data can be sorted in ascending or descending order based on one or more columns using the SQL ORDER BY clause. | SELECT column-list FROM table\_name  [WHERE condition]  [ORDER BY column1, column2, .. column] [ASC | DESC];  Example:  SELECT \* FROM CUSTOMERS  ORDER BY NAME, SALARY; |
| **TOP** | The TOP N number or X percent of records in a table can be retrieved using the SQL TOP clause. | SELECT TOP number|percent columnname  FROM tablename  Example  SELECT TOP 3 \* FROM CUSTOMERS; |
| **DISTINCT** | It makes more sense to only retrieve the unique records for such records rather than retrieving duplicate records. | SELECT DISTINCT column1,..columnN  FROM tablename;  Example:  SELECT DISTINCT SALARY  FROM CUSTOMERS  ORDER BY SALARY |
| **BETWEEN…AND** | The BETWEEN operator chooses values from a predetermined range. The values could be text, numbers, or dates. | SELECT column\_name(s)  FROM table\_name  WHERE column\_name BETWEEN value1 AND value2;  Example:  SELECT Name FROM Customers  WHERE Age BETWEEN 10 AND 20; |
| **LIKE** | The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.  The percent sign (%): one or many characters  The underscore (\_): single number/character | SELECT \* FROM CUSTOMERS  WHERE SALARY LIKE '200%'; |
| **SUM** | SQL SUM function is used to find out the sum of a field in various records | SELECT SUM(*column\_name*)  FROM *table\_name*  WHERE *condition*;  Example:  Calculate total of all the dialy\_typing\_pages, then you can do so by using the following command  SELECT SUM(daily\_typing\_pages)  FROM employee\_tbl; |
| **MIN** | SQL MIN function is used to find out the record with minimum value among a record set. | Syntax:  SELECT MIN(*column\_name*)  FROM *table\_name*  WHERE *condition*;  Example:  SELECT MIN(daily\_typing\_pages)  FROM employee\_tbl; |
| **MAX** | SQL MAX function is used to find out the record with maximum value among a record set. | Syntax:  SELECT MAX(*column\_name*)  FROM *table\_name*  WHERE *condition*;  Example:  SELECT MAX(daily\_typing\_pages)  FROM employee\_tbl; |
| **AVG** | SQL AVG function is used to find out the average of a field in various records. | Syntax:  SELECT AVG(*column\_name*) FROM *table\_name*  WHERE *condition*;  Example:  SELECT AVG(daily\_typing\_pages)  FROM employee\_tbl; |
| **COUNT** | SQL COUNT function is the simplest function and very useful in counting the number of records, which are expected to be returned by a SELECT statement. | Syntax:  SELECT COUNT(*column\_name*)FROM *table\_name*  WHERE *condition*;  Example  SELECT COUNT(\*) FROM employee\_tbl ; |
| **PROJECT (∏)** | Project operation is done by Projection Operator which is represented by "pi"(∏). It is used to.  retrieve certain attributes(columns) from the table. It is also known as vertical partitioning as it separates the table vertically. It is also a unary operator. | Notation: ∏ a(r)  Where ∏ is used to represent PROJECTION r is used to represent RELATION a is the attribute list  Example:  Notation: ∏ NAMEՅSTUDENTՆ |
| **CARTESIAN PRODUCT Cross-product (X)** | The Cartesian product is denoted by the "X" symbol. Let's say we have two relations R and S. Cartesian product will combine every tuple(row) from R with all the tuples from S. It is a binary operator as it requires two operands. | Notation: R X S  R is the first relation  S is the second relation  Example: make a combination of the two relations STUDENT and EMPLOYEE.  Notation: STUDENT X EMPLOYEE |
| **UNION (∪)** | Union operation is done by Union Operator which is represented by "union"(∪). It is the same as the union operator from set theory, i.e., it selects all tuples from both relations but with the exception that for the union of two relations/tables both relations must have the same set of Attributes. It is a binary operator as it requires two operands. | Notation: R ∪ S  R is the first relation, S is the second relation  Notice: Two relation must have same schema |
| **DIFFERENCE (-)** | Set Difference as its name indicates is the difference between two relations ՅR՟SՆ. It is denoted by a "Hyphen"(-) and it returns all the tuples(rows) which are in relation R but not in relation S. It is also a binary operator. | Notation: R - S  R is the first relation  S is the second relation  Notice: Two relation must have same schema |
| **THETA JOIN (θ)** | Theta Join combines two relations using a condition. This condition is represented by the symbol "theta"(θ). Here conditions can be inequality conditions such as >,<,>=,<=, etc. | Notation: R ⋈θ S  R is the first relation, S is the second relation  θ is the condition  Notation:EMPLOYEE⋈EMPLOYEE.EXPERIENCE>=DEPARTMENT.MIN\_EXPERIENCE DEPARTMENT |
| **EQUIVALENT JOIN** | Equivalent Join is a special case of theta join where the condition can only contain equality(=) comparisons. A non-equijoin is the inverse of an equivalent join, which occurs when you join on a condition other than "=". | Notation: R ⋈ a=b S  R is the first relation  S is the second relation  a=b is the condition  Example: we would like to join EMPLOYEE and DEPARTMENT relation where E\_NO from  EMPLOYEE ֠ E\_NO from DEPARTMENT.  Notation: `EMPLOYEE ⋈ EMPLOYEE.E\_NO ֠ DEPARTMENT.E\_NO DEPARTMENT |
| **NATURAL JOIN (⋈)** | A comparison operator is not used in a natural join. It does not concatenate like a Cartesian product. A Natural Join can be performed only if two relations share at least one common attribute. Furthermore, the attributes must share the same name and domain. Natural join operates on matching attributes where the values of the attributes in both relations are the same and removes the duplicate ones. Preferably Natural Join is performed on the foreign key. | Notation : R ⋈ S  R is the first relation  S is the second relation  Example: EMPLOYEE ⋈ DEPARTMENT |
| **INTERSECTION (∩)** | Intersection operation is done by Intersection Operator which is represented by "intersection" (∩). It is the same as the intersection operator from set theory, i.e., it selects all the tuples which are present in both relations. It is a binary operator as it requires two operands. Also, it eliminates duplicates. | Notation : R ∩ S  R is the first relation ,S is the second relation  Example: Suppose we want the names which are present in STUDENT as well as in EMPLOYEE relation, Relations we used in Basic Operations.  Notation: ∏NAME(STUDENT)∩∏NAME(EMPLOYEE) |
| **DIVISION (÷)** | Division Operation is represented by "division"(÷ or /) operator and is used in queries that involve keywords "every", "all", etc. | Notation: R(X,Y)/S(Y)  R is the first relation from which data is retrieved.  S is the second relation that will help to retrieve the data.  X and Y are the attributes/columns present in relation. We can have multiple attributes in relation, but keep in mind that attributes of S must be a proper subset of attributes of R. For each corresponding value of Y, the above notation will return us the value of X from tuple ֜X,Y֝ which exists everywhere. |

## Connecting Java and SQL server database

### Database

* **Create a login user.**
* Login to SQL Server with Authentication is set to Windows Authentication.
* From the Object Explorer window, select Security and then Logins. Right click is on sa and click Properties.
* From the General page, assign a password for user sa.
* Switch to the Status page and select Enabled in the Login section. Click OK to finish.
* Restart the database server by right clicking the server’s name in the Object Explorer window and click Restart.
* Select Yes to perform the action.

### Java

**a. Create a new project.**

* Downloading Apache NetBeans IDE.
* Open NetBeans and create a new project by selecting “File” from the menu, then “New Project.”
* Select “Java” and “Java Application” and then click “Next”.
* Give your project a name, and then click “Finish.”

**b. Create a form.**

* Right click on the package which has the same name as the project, select **New**, and then select **JFrame Form**.
* Input name the JFrame Form, i.e.: form, then click **Finish**.

**c. Adding more libraries.**

* In the Properties window, select Libraries and click on the Add Library button.
* Select Microsoft SQL Server JDBC Driver from the list of available libraries and click on Add Library.
* Right-click on the project folder again and select New -> Other.
* In the New File dialog box, select JDBC -> JDBC Connection and click Next.
* Enter a name for the connection and select the Microsoft SQL Server JDBC Driver from the lists of drivers.
* Enter the connection details, including the server name, database name, username, and password.
* Test the connection by clicking on the Test Connection button.

**d. User Interface Design**

* In the “New JFrame Form” dialog, give your form a name and select any other desired options. Click “Finish”.
* The form designer will open, allowing you to drag and drop components onto the form. The available components can be found in the “Palette” window.
* To add a component to the form, drag it from the “Palette” window onto the form. You can modify the properties of the components in the “Properties” window.

**e. Display the form.**

* Once you have added all of the desired components, you can save the form by selecting “File” -> “Save” from the menu.
* To add functionality to the form, you can double-click on the components to generate an event handler.
* Once you have finished designing your form and adding functionality, you can run the project by selecting “Run” -> “Run Project” from the menu.

**f. Connecting to the database.**

* Double click on button in the form to create an on click event for the button.
* In this dialog, enter the code for the on click event of the button.

2.1: Check if the user hasn’t input a query, display an error message and return control tp the main form.

2.2: Database connection string

2.3: Fetch the column information for the table

2.4: Obtain the results of the query

2.5: Display the result into Text Area

**Figure II‑11 Source code to connect**

**g. Execution result**: Input a query and click run.

# RELATIVE WORKS.

## Analyst requirement of topic

1. **User (Customer) is able to:**

* Sign up on the site and set up their profile.
* Search for their required products and checkout to the payment page.
* View/add/remove content from shopping cart.
* Once payment is successful, the expected time of product arrival must be communicated to the user via emailas well as displayed on the order page.
* Also, they can sell used furniture to the company. Users must be able to upload pictures of the items they intend to sell.

1. **Company is able to**

* Buys used furniture from users at a discount, refurbishes it, and sells it at a profit.
* Based on the item’s images of customers, company admins must be able to decide whether to buy or not.
* Admin can control and give task to employees and see all product.
* Employees can sell/add/delete/update product.
* Employees can process customer orders

## Design ERD

We define the following entities and characteristics based on the subject requirement, as shown in the table.

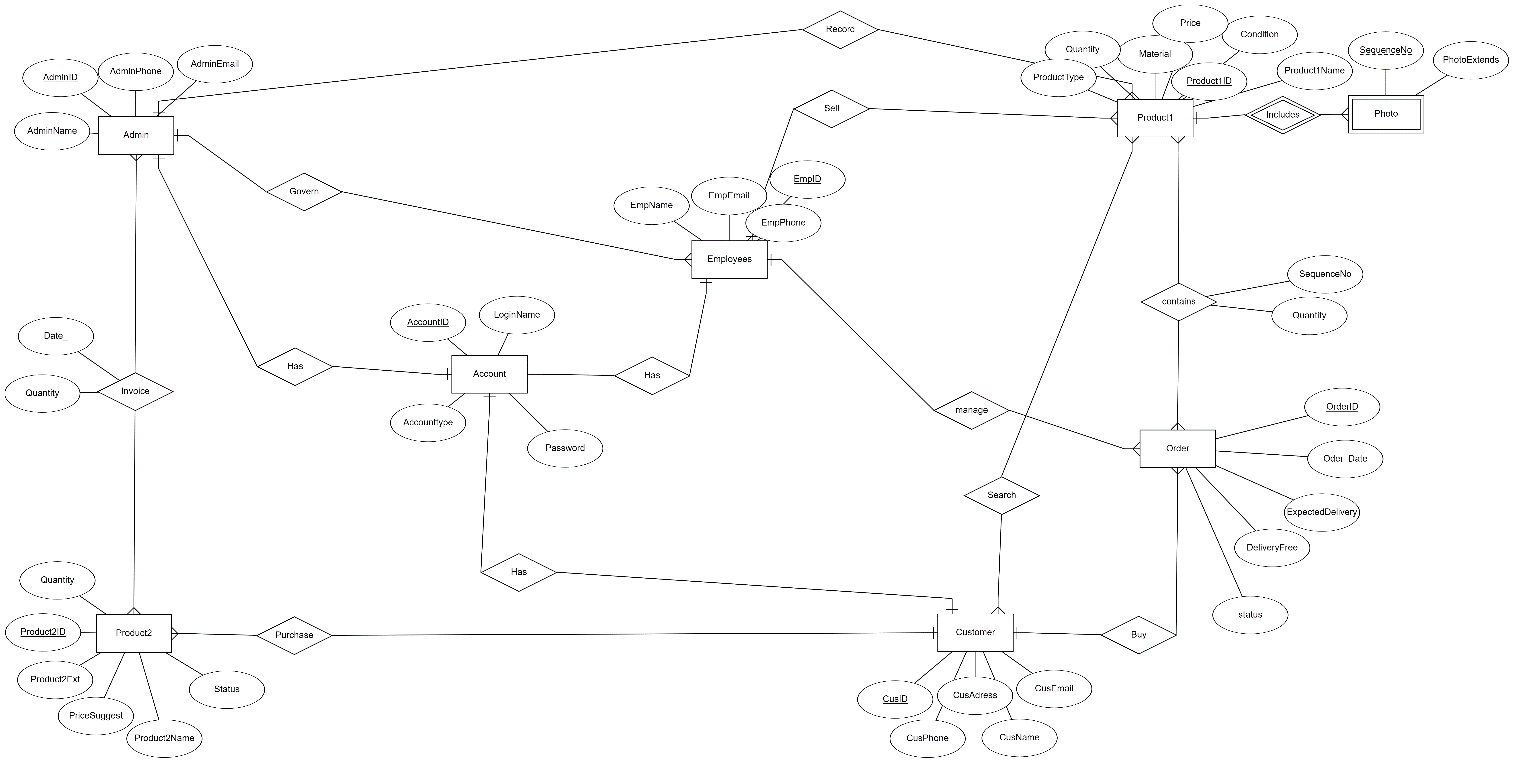
**Table III.1 Description of ERD**

|  |  |  |
| --- | --- | --- |
| **Entity** | **Meaning** | **Attribute** |
| **Admins** | The entity sets have a relationship with three other entity sets. Specifically, administrators can record the clothing department, having the ability to decide whether to buy second-hand products from customers. In addition, the administrator will be the person who directly assigns tasks and controls the work situation of employees. | **AdminID :** There are many managers in the same website, so separating them requires separate ID. |
| **AdminEmail:** Every owner’s email address is needed to communicate with them professionally and it will help staff connect them with any information that needs to be passed on. |
| **AdminName:** Their name is recorded here. |
| **AdminPhone:** In the event of an emergency or any mishap, employees need to contact their manager |
| **Employees**  **(Staff).** | This entity provides a list of employees provided with a unique ID to log into their own accounts to systematically perform their duties. This entity directly sells or updates/ posts/deletes clothing, so it has a link to the actual clothing | **EmpID**: Each employee has been assigned a unique ID to allow them to have access to a specific section, this information is stored in this property. |
| **EmpName:** Their name is recorded here. |
| **EmpEmail**: To pass on any information or to communicate with employees about business matters. |
| **EmpPhone:** It is necessary to record their phone number when emergencies at work |
| **Customers** | Customers have a big impact on the product because they are the key stakeholders in the company. Customers are those who purchase goods from retailers, generating revenue for them. Customers have the option to look for and purchase things based on their needs. | **CusID:** Each customer needs a unique ID to distinguish customers from each other. |
| **CusName:** Their name is recorded here. |
| **CusPhone**: Customer’s phone number required for delivery |
| **CusAddress:** Address provided required for delivery |
| **CusEmail:** This attribute includes customer’s email to provide for us expected time of delivery |
| **Account** | This entity sets contains login information of all three entity sets: Admin, Employees, Customer | **AccountID**: Unique ID distinguishes accounts from each other. |
| **LoginName**: Username |
| **Password** |
| **Accounttype**:Admin/Customer/Employee |
| **Product 1** | Products are for sale on the web. If a customer has any requirement, then this entity will further assist the customer to check according to customer request. It needs sundry properties. | **Product1ID:** Each product has a unique ID to distinguish it |
| **Product1Name**: Production’s name |
| **Condition** (%) |
| **Quantity**: The number of productions of each type |
| **ProductType**: The property divides clothing data into distinct groups. |
| **Material**: Product material |
| **Price**: Price of product |
| **Photo** | Each product type also has its own image so we say production 1 entity set has a connection with photo entity set | **SequenceNo**: Since each item may contain multiple images, this will be the order for that image. |
| **PhotoExtends**: Store product images. |
| **Product2** | Customers with administrator clearance can opt to sell things to the firm. As a result, this entity is related to both the customer and admin | **Product2ID:** Each product has a unique ID to distinguish it |
| **Product2Name**: Production’s name |
| **Product2Ext**: Images of store products |
| **PriceSuggest:** The seller's desired price. |
| **Quantity:** The number of productions |
| **Order\_** |  | OrderID: Each order has a unique ID to distinguish it |
|  | Oder\_Date: This attribute store the day when customer buy product |
|  | ExpectedDelivery: The expected time of product arrival must be communicated to the customer |
|  | DeliveryFee |

**Table III.2 Define relationship and map cardinality.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Relationship** | **Attribute on relationship** | **Entity sets in relationship** | **Cardinality** |
| Record |  | Admin & Production1 | One to Many |
| Govern |  | Admin & Employees | One to Many |
| Invoice | Date\_, Quantity, PriceProduct, | Admin & Production2 | Many to Many |
| Purchase |  | Customer & Production2 | One to Many |
| Contains(OrderDetails) | Sequence No,Quantity | Customer & Production1 | Many to Many |
| Search |  | Customer & Production1 | Many to Many |
| Sell |  | Employees & Production1 | One to Many |
| Includes |  | Production1 & Photo | One to Many |
| Has |  | Admin & Account | One to One |
| Has |  | Customer & Account | One to One |
| Has |  | Employees & Account | One to One |
| Buy |  | Order & Customer | Many to One |
|  |  |  |  |

According to Entities, Attributes, Relationship, we design ERD.

Diagram

Description automatically generated

**Figure III‑1 ERD Diagram**

**Figure III‑2 Relational Schema**

Diagram, schematic

Description automatically generated

**Figure III‑3 Database Diagram**

## Convert ERD model to relational model.

* + - Admin\_(AdminID, AdminPhone, AdminEmail, AdminName, AccountID)
    - Product1(Product1ID, ProductType, Product1Name, Quantity, Condition, Material, Price, AdminID, EmpID)
    - Photo( Product1ID, SequenceNo, PhotoExtends)
    - Order\_(OrderID, Oder\_Date, Status ExpectedDelivery, DeliveryFree, CusID, EmpID)
    - OrderDetails (OrderID, Product1ID, SequenceNo, Quantity)
    - Search( CusID, Product1ID)
    - Customers(CusID, CusName, CusEmail, CusPhone, CusAdress, AccountID)
    - Product2(Product2ID, Product2Ext, Product2Name, Quantity, PriceSuggest,Status, CusID)
    - Invoice(AdminID, Product2ID, Date\_, Quantity)
    - Account(AccountID, LoginName, Password, Accounttype)
    - Employees(EmpID, EmpName, EmpEmail, EmpPhone, AdminID, AccountID)

## Normalize

**1NF:** A relational database that has all attributes in a tuple must have a single value from the domain of that attribute, and the domain of an attribute only include atomic values. Therefore, database is already in the first normal form.

**2NF**: A relational database is in first normal form and every non-key characteristic is completely functionally dependent on the primary key. Therefore, database is already in the second normal form.

**3NF:** A relational database is in 2NF and non-transitive functional dependency of non-prime attributes on any super key. Therefore, database is already in the third normal form.

## Create database in SQL server.

CREATE DATABASE FinalSourceDatabase;

Use FinalSourceDatabase;

CREATE TABLE ACCOUNT(

LoginName varchar (12) NOT NULL,

Password varchar (20) NOT NULL,

Accounttype varchar (12) NOT NULL,

AccountID char (6) PRIMARY KEY);

CREATE TABLE Admin\_ (

AdminEmail varchar (30) NOT NULL,

AdminPhone varchar (12) NOT NULL,

AdminName varchar (50) NOT NULL,

AdminID char (6) PRIMARY KEY,

AccountID char (6) FOREIGN KEY REFERENCES ACCOUNT(AccountID) NOT NULL);

CREATE TABLE Customers (

CusID char(6) PRIMARY KEY ,

CusName nvarchar(50) NOT NULL,

CusAddress varchar(255)NOT NULL,

CusPhone varchar(12) NOT NULL,

CusEmail varchar(255) NOT NULL,

AccountID char(6) FOREIGN KEY REFERENCES ACCOUNT(AccountID) NOT NULL);

CREATE TABLE Employees (

EmpEmail varchar(30) NOT NULL,

EmpID char(6) PRIMARY KEY,

EmpName nvarchar(50) NOT NULL,

EmpPhone varchar(12) NOT NULL,

AdminID char(6) FOREIGN KEY REFERENCES Admin\_(AdminID) NOT NULL,

AccountID char(6) FOREIGN KEY REFERENCES ACCOUNT(AccountID) NOT NULL);

CREATE TABLE Product1(

Product1ID char(8) PRIMARY KEY,

Product1Name varchar(100) NOT NULL,

ProductType varchar (50) NOT NULL,

Quantity smallint NOT NULL,

Material varchar(100) NOT NULL,

Price money NOT NULL,

Condition decimal(5,2) NOT NULL,

EmpID char(6) FOREIGN KEY REFERENCES Employees(EmpID) NOT NULL,

AdminID char(6) FOREIGN KEY REFERENCES Admin\_(AdminID)) NOT NULL;

CREATE TABLE photo(

Product1ID char(8) FOREIGN KEY REFERENCES Product1(Product1ID),

SequenceNo tinyint NOT NULL,

PhotoExtends varbinary(max)NOT NULL,

PRIMARY KEY ( Product1ID, SequenceNo));

CREATE TABLE Product2(

Product2ID varchar(8) PRIMARY KEY NOT NULL,

CusID char(6) FOREIGN KEY REFERENCES Customers(CusID),

Product2Name varchar(100) NOT NULL,

PriceSuggest money NOT NULL,

Status varchar(50) NULL,

Quantity smallint NOT NULL,

Product2Ext varbinary(max) NOT NULL);

CREATE TABLE Invoice(

AdminID char(6) FOREIGN KEY REFERENCES Admin\_(AdminID),

Date\_ date NOT NULL,

Product2ID varchar(8) FOREIGN KEY REFERENCES Product2(Product2ID),

Quantity tinyint NOT NULL,

PRIMARY KEY(AdminID, Product2ID));

CREATE TABLE Order\_(

Order\_Date date NOT NULL,

ExpectedDelivery date NULL,

Quantity smallint NOT NULL,

CusID char(6) FOREIGN KEY REFERENCES Customers(CusID) ,

Product1ID char(8) FOREIGN KEY REFERENCES Product1(Product1ID),

DeliveryFee money NOT NULL,

OrderID varchar(6) Primary Key,

EmpID char(6) NULL FOREIGN KEY REFERENCES Employees(EmpID),

;

CREATE TABLE OrderDetails(

SequenceNo char(20) NOT NULL,

Quantity smallint() NOT NULL,

OrderID varchar(6) NOT NULL FOREIGN KEY REFERENCES Order\_(OrderID),

Product1ID char(8) NOT NULL FOREIGN KEY REFERENCES Product1(Product1ID),

Primary key(OrderID, Product1ID),

;

CREATE TABLE Search(

CusID char(6) FOREIGN KEY REFERENCES Customers(CusID),

Product1ID char(8) FOREIGN KEY REFERENCES Product1(Product1ID),

PRIMARY KEY (CusID, Product1ID));

## Collect & Insert data.

Data used in the database is made randomly by using Python and related databases from Kaggle since collecting data for an e-commercial site was not possible due to privacy. Also notice that because the data was randomly created, duplicated and illogical data was unavoidable.

The primary goal of using Python was to create random records that follow the “string” form of INSERT INTO (value1) VALUES function in SQL Server. For example, below is a record from Admin\_ relation:

Use Database

INSERT INTO Admin\_ VALUES

(N'manhtoan203@gmail.com', '0949382892', 'Toan Manh', 'Admin1', 'Ad1'),

(N'tyhoang195@gmail.com', '0945682122', 'Hoang Ty', 'Admin2', 'Ad2'),

(N'nganguyen093@gmail.com', '0923682555', 'Nguyen Hoang Nga', 'Admin3', 'Ad3'),

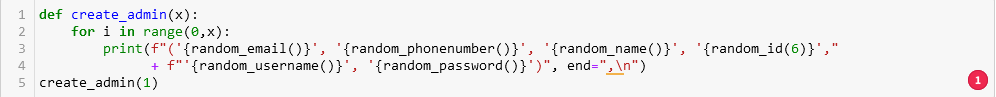
(N'thigung909@gmail.com', '0923682868', 'Tran Thi Gung', 'Admin4','Ad4'),

(N'vancau123@gmail.com', '0926682999', 'Doan Van Cau', 'Admin5', 'Ad5');

Most of the work was done by utilizing built-in functions and libraries in Python along with some custom functions which were made to fit with the data demand. Databases from different sources were also used to create records for attributes like Customers name, Address, phone number, and even pictures of Products (link to those datasets can be seen at reference part).

To have a better insight of how these random codes work, Admin, Customers, and Employee will be taken as examples:

Admin:

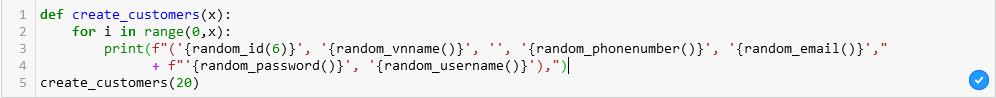


**Figure III‑4 Admin code**

Return: 

**Figure III‑5 Admin values**

Customers:



**Figure III‑6 Customer code**

Return:

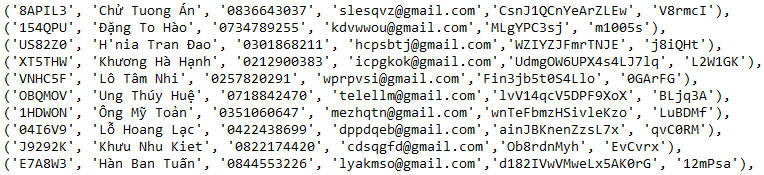


**Figure III‑7 Customer values**

Employees:



**Figure III‑8 Employees code**

Return: 

**Figure III‑9 Employees values**

The inserted value followed the order of attributes in their corresponding relation, this prevented extra syntax from INSERT INTO and provided better modification if necessary for future adjustments. Some data might seem illogical as mentioned before, so they were modified again to better suit making queries.

All the datasets from Kaggle will be listed in the References, and the Python code is also included in the file package.

## Querying database

* + 1. Select all column of table ‘photo’?
* SQL: Select\*From photo
* RA: πProduct1ID, SequenceNo, PhotoExtends photo
* Tree Notation:



* + 1. Find a list of employees managed by Admin 2?
    - SQL:

SELECT \* FROM Employees

WHERE AdminID= 'Admin2';

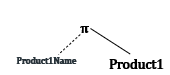
* + - RA: σAdminID='Admin2' (Employees)
    - Tree Notation:



* + 1. Find name of product currently on sale?
    - SQL:

SELECT DISTINCT Product1Name FROM Product1

* + - RA: π product1name Product1
    - Tree Notation:



* + 1. Find items on sale that are made of MDF ?
    - SQL:

SELECT \* FROM Product1

WHERE Material = 'MDF';

* + - RA: σ Material = 'MDF'(Product1)
    - Tree Notation:

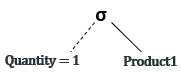


* + 1. Find items on sale that have quantity equals to one?
    - SQL:

SELECT \* FROM Product1

WHERE Quantity = 1;

* + - RA: σ Quantity = '1' (Product1)
    - Tree Notation:

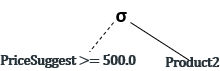


* + 1. Product that Customer list for sale with price-suggest greater than 500,00 ?
    - SQL:

SELECT \* FROM product2

WHERE PriceSuggest >=500.00;

* + - RA: σ PriceSuggest >=500.0 (Product2)
    - Tree Notation:



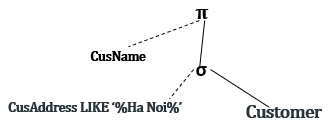
* + 1. Find all name of customers who live in HaNoi?
    - SQL:

SELECT CusName

FROM Customers

WHERE CusAddress LIKE '%Ha Noi%';

* + - RA: π CusName(σ CusAddress LIKE '%Ha Noi%' (Customers))
    - Tree Notation



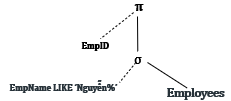
**Customers**

* + 1. Information gathering on employees with the last name ‘Nguyen’?
* SQL:

SELECT EmpID FROM Employees

WHERE EmpName LIKE 'Nguyen%';

* RA: π EmpID(σEmpName LIKE ' Nguyen %'(Employees))
* Tree Notation:



* + 1. Locate the highest and lowest prices for each category of products?
* SQL:

SELECT Material, MAX(Price) as maxprice, MIN(Price) as minprice  
FROM Product1  
GROUP BY Material

* RA: γ Material, MAX (price), MIN (price) → maxprice, minprice Product1
* Text, letter

  Description automatically generatedTree Notation:
  + 1. The total number of items that the firm now sells?
    - SQL:

SELECT COUNT(Quantity) AS TotalProfuct

FROM Product1;

* + - RA: G count(Quantity) → TotalProduct (product1)
    - Tree Notation:

A picture containing diagram

Description automatically generated

* + 1. Count how many items with a quantity higher than three are available for purchase?
    - SQL:

SELECT COUNT (\*) AS countproduct2

FROM Product2

WHERE Quantity>3;

* + - RA: G Count (\*) → countproduct2(σ Quantity > 3)(Product2)
    - Tree Notation:

Chart, radar chart

Description automatically generated

* + 1. Find all employees whose name begins with "Sùng%"?
    - SQL:

SELECT CusName FROM Customers

WHERE CusName LIKE 'Sùng%';

* + - RA: π CusName(σ(CusName LIKE 'Sùng%')(Customers))
    - Tree notation:

Chart, radar chart

Description automatically generated

* + 1. Total value of existing products?
* SQL: SELECT SUM(Price) AS TotalProductPrice FROM Product1;
* RA: G SUM (Price) → TotalProductPrice (Product1)
* Tree Notation:

Diagram, timeline

Description automatically generated

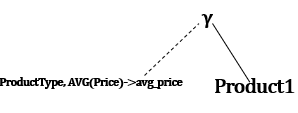
* + 1. Find average product value follow by each product type?
    - SQL:

SELECT ProductType, AVG(Price) AS avg\_price

FROM Product1

GROUP BY ProductType;

* + - RA: γ ProductType, AVG (price) → avg\_price(Product1)
    - Tree Notation:



* + 1. Find products with prices from 400000.00 to 8000000.00?
    - SQL:

SELECT Product1Name, Price, Condition FROM Product1

WHERE Price BETWEEN 4000000.00 AND 8000000.00;

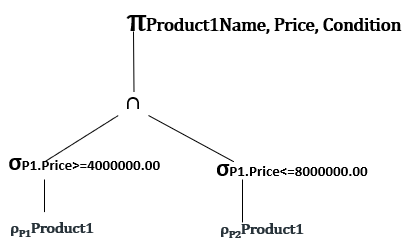
* + - RA:

ρP1Product1

ρP2Product1

(π Product1Name, Price, Condition (σ P1.Price >= 4000000.00 (P1)) **∩** (π P2.Product1Name, P2.Price,P2.Condition (σ P2.Price <= 8000000.00 (P2))

* + - Tree Notation:



* + 1. Find out customers who haven't purchased any products yet?
    - SQL:

Select C.CusID, C.CusName

From Customers as C

Where CusID NOT IN (Select O.CusID

From Order\_ as O)

* + - RA:

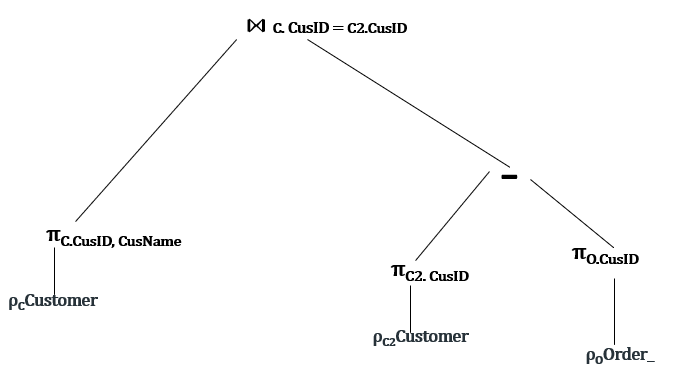
ρCCustomer

ρC2Customer

ρOOrder\_

πC.CusID,Cusname(C)**⋈** (πC2.CusID(C2)-πO.CusID(O))

* + - Tree Notation:



* + 1. Find the highest and lowest order value?
    - SQL:

SELECT MAX((table1.Price\*table1.Quantity)+table2.DeliveryFee) as max, MIN((table1.Price\*table1.Quantity)+ table2.DeliveryFee) as min FROM (SELECT P.Price,O.Quantity, O.OrderID FROM OrderDetails as O, Product1 as P Where O.Product1ID = P.Product1ID) as table1, (SELECT Order\_.OrderID, DeliveryFee From Order\_ ) as table2 WHERE table1.OrderID = table2.OrderID ;

* + - RA:

ρPProduct1

ρOOrderDetails

πP.Price, O.Quantity, O.OrderID(O⋈ P) as table1

πDeliveryFee, OrderID(Order\_) as table2

GMAX((table1.Price \*table1.Quantity)+table2.DeliveryFee) , MIN ((table1.Price \*table1.Quantity)+table2.DeliveryFee)  Max, Min(table1⋈ table2)

* + - Tree Notation:

Diagram

Description automatically generated

* + 1. Find the product with the lowest condition value ?
* SQL:

SELECT \* FROM Product1

WHERE Condition =

(SELECT MIN(Condition) FROM Product1);

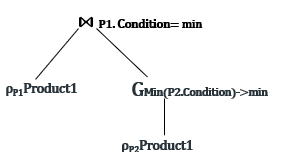
* RA:

ρP1Product1

ρP2Product1

P1⋈ P1.Condition = min(GMin(P2.Condition)->min(P2))

* Tree Notation:



* + 1. The most expensive item on the web?
* SQL:

SELECT \* FROM Product1

WHERE Price =

(SELECT MAX(Price) FROM Product1);

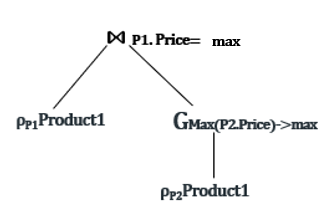
* RA:

ρP1Product1 (Rename)

ρP2Product1 (Rename)

P1⋈ P1.Price = max(GMax(P2.Price)->max(P2))

* Tree Notation:



* + 1. How many customers have an address in Ho Chi Minh City?

SQL:

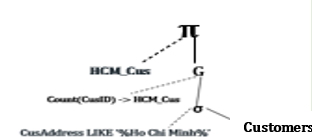
SELECT COUNT(CusID) AS HCM\_Cus

FROM Customers

WHERE CusAddress LIKE '%Ho Chi Minh City%';

RA: GCount(CusID)->HCM\_Cus(σ CusAddress LIKE '%Ho Chi Minh City%' (Customers))

Tree Notation:



* + 1. Sort by descending order of product of customers sell with products with a price-suggest greater than 1000000.00?
* SQL:

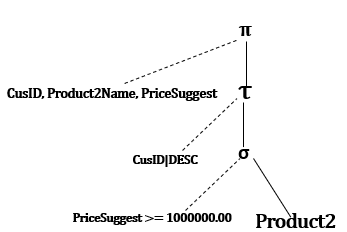
SELECT CusID, Product2Name, PriceSuggest

FROM Product2

WHERE PriceSuggest >=1000000.00

ORDER BY CusID DESC;

* RA: π CusId, Product2Name, PriceSuggest (τ Cusid |DESC(σ PriceSuggest >= 1000000.0 (Product2)))
* Tree Notation:



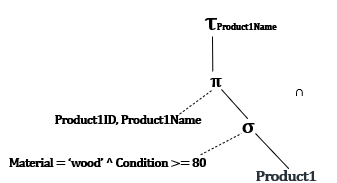
* + 1. Product1ID and Name with Wood material and condition 80 or more Sort by product name?
* SQL:

SELECT Product1ID, Product1Name

FROM Product1 WHERE Material= 'wood' AND Condition >= 80

ORDER BY Product1Name;

* RA: π Product1ID, Product1Name(τ Product1Name(σ Material= 'wood'^Condition >= 80 (Product1))
* Tree Notation:



* + 1. Find customers with the most sales for the company?

SQL:

Select P.CusID, Count(Distinct CusID) as count

From Product2 as P, Invoice as I

Where P.Product2ID = I.Product2ID

Group by P.CusID

Order by count DESC;

RA: τ count|DESC (γ CusID, count-distinct (CusID)->count (Product2 **⋈** Invoice))

Tree Notation:

Diagram

Description automatically generated

* + 1. The maximum and minimum price of each material, with the requirement that the minimum price be at least 900?
* SQL:

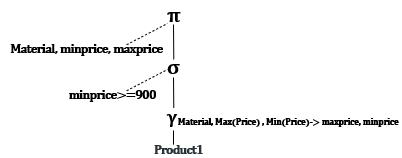
SELECT Material, MAX(Price) as maxprice, MIN(Price) as minprice

FROM Product1

GROUP BY Material

HAVING MIN(Price) >=900

* RA: σ minprice >= 900.00 (γ material, MIN (price), MAX (price)->minprice, maxprice Product1)
* Tree Notation:



* + 1. Find out products that aren't currently on sale.
* SQL:

Select Product1ID, Product1Name

From Product1

Where Product1ID NOT IN (Select O.Product1ID From OrderDetails AS O)

* RA:

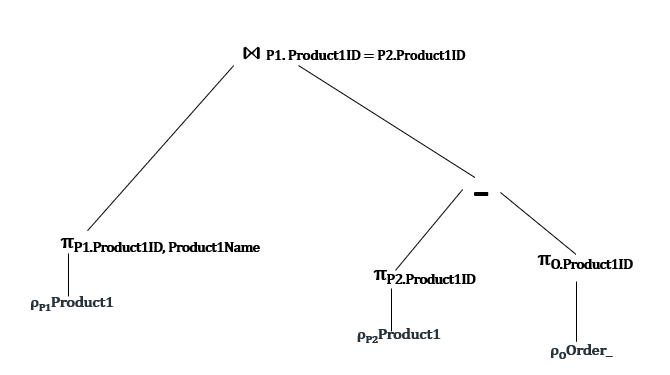
ρP1Product1

ρP2Product1

ρOOrderDetails

πP1.Product1ID, Product1Name(P1)**⋈** (πP2.Product1ID(P2)-πO.Product1ID(O))

* Tree Notation:



* + 1. Monthly income statistics in 2022, as well as the number of orders every month?
* SQL:

SELECT Month, SUM(Sum) as From (SELECT Month,(table1.Price\*table1.Quantity)+table2.DeliveryFee as Sum FROM (SELECT P.Price,O.Quantity, O.OrderID FROM OrderDetails as O, Product1 as P Where O.Product1ID = P.Product1ID) as table1, (SELECT Month(Order\_Date) as Month, Order\_.OrderID, DeliveryFee From Order\_ Where Year(Order\_Date)='2022' ) as table2 WHERE table1.OrderID = table2.OrderID) as table3 group by Month;

* RA:

ρPProduct1

ρOOrder\_

πP.Price, O.Quantity, O.OrderID(O⋈ P) as table1

πMonth(Order\_Date) as Month, DeliveryFee, OrderID(σ Year(Order\_Date) = ‘2022’ Order\_) as table2

γ Month, SUM((table1.Price \*table1.Quantity)+table2.DeliveryFee)->Sum (table1⋈ table2)

* Diagram

  Description automatically generatedTree Notation:
  + 1. Find out the name of the employee who sells the most orders in 2022?
* SQL:

Select E.EmpID, E.EmpName From Employees as E

INNER JOIN

(SELECT TOP 3 table2.EmpID, SUM ((table1.Price\*table1.Quantity)+ table2.DeliveryFee) as Fee FROM (SELECT P.Price,O.Quantity, O.OrderID FROM OrderDetails as O, Product1 as P Where O.Product1ID = P.Product1ID) as table1, (SELECT EmpID, Order\_.OrderID, DeliveryFee From Order\_ Where Year(Order\_Date) = ‘2022’) as table2 WHERE table1.OrderID = table2.OrderID Group by table2.EmpID Order by Fee DESC) as table3 ON E.EmpID = table3.EmpID;

* RA:

ρEEmployees

ρPProduct1

ρOOrder\_

πP.Price, O.Quantity, O.OrderID(O⋈ P) as table1

πEmpID,DeliveryFee, OrderID(σ Year(Order\_Date) = ‘2022’ Order\_) as table2

τ top3Fee (γ Month, SUM((table1.Price \*table1.Quantity)+table2.DeliveryFee)->Fee (table1⋈ table2)) as table3

πEmpName,E.EmpID(E)⋈E.EmpID = table3.EmpID table3

* Tree Notation:

Diagram

Description automatically generated

* + 1. Find revenue per employee?
* SQL:

SELECT \* FROM (SELECT table2.EmpID, SUM((table1.Price\*table1.Quantity) + table2.DeliveryFee) as Sum, Count(DISTINCT table2.EmpID) as TotalOrder FROM (SELECT P.Price,O.Quantity, O.OrderID FROM OrderDetails as O, Product1 as P Where O.Product1ID = P.Product1ID) as table1, (SELECT EmpID, OrderID, DeliveryFee From Order\_) as table2 WHERE table1.OrderID = table2.OrderID Group by table2.EmpID) as table3

LEFT JOIN

(SELECT E.EmpID, AdminID, EmpName From Employees as E) as table4

ON table3.EmpID = table4.EmpID;

* RA:

ρEEmployees

ρPProduct1

ρOOrder\_

πP.Price, O.Quantity, O.OrderID(O⋈ P) as table1

πEmpID,DeliveryFee, OrderID(Order\_) as table2

γ table2.EmpID, SUM((table1.Price \*table1.Quantity)+table2.DeliveryFee)->Sum (table1⋈ table2)) as table3

table3 table3.EmpID = E.EmpID (πEmpName,E.EmpID(E))

* Tree Notation:

Diagram

Description automatically generated

* + 1. Find three names of customers who purchased the most expensive bills in 2022.
* SQL:

SELECT table4.CusID, CusName, Sum FROM (SELECT TOP 3 table2.CusID,SUM((table1.Price\*table1.Quantity)+table2.DeliveryFee) as Sum FROM (SELECT P.Price,O.Quantity, O.OrderID FROM OrderDetails as O, Product1 as P Where O.Product1ID = P.Product1ID) as table1, (SELECT CusID, Order\_.OrderID, DeliveryFee From Order\_) as table2 WHERE table1.OrderID = table2.OrderID Group by table2.CusID Order by Sum DESC) as table3

LEFT JOIN

(SELECT C.CusID, CusName From Customers as C) as table4

ON table3.CusID = table4.CusID ;

* RA:

ρCCustomers

ρPProduct1

ρOOrder\_

πP.Price, O.Quantity, O.OrderID(O⋈ P) as table1

πCusID,DeliveryFee, OrderID(σ Year(Order\_Date) = ‘2022’ Order\_) as table2

τ top3Sum (γ CusID, SUM((table1.Price \*table1.Quantity)+table2.DeliveryFee)->Sum (table1⋈ table2)) as table3

πCusID,CusName(C) C.CusID = table3.CusID table3

* Tree Notation:

Diagram

Description automatically generated

## Create View

Make a view that contains ID, Name, Email of Customer.

CREATE VIEW VWCustomers

AS

SELECT CusID, CusName, CusEmail

FROM Customers;

Create View of Orders.

CREATE VIEW VWOrder\_2022

AS

SELECT table4.CusID, CusName,CusPhone, CusAddress, table3.OrderID, Sum FROM

(SELECT table2.CusID,table2.OrderID,SUM((table1.Price\*table1.Quantity) +table2.DeliveryFee) as Sum FROM (SELECT P.Price,O.Quantity, O.OrderID FROM OrderDetails as O, Product1 as P Where O.Product1ID = P.Product1ID) as table1, ( SELECT CusID, OrderID,Order\_Date, DeliveryFee From Order\_) as table2 WHERE table1.OrderID = table2.OrderID Group by table2.OrderID, table2.CusID) as table3

LEFT JOIN

(SELECT C.CusID, CusName, CusPhone, CusAddress

From Customers as C) as table4

ON table3.CusID = table4.CusID ;

;

CREATE View of information of account in system.

CREATE VIEW VWAccount

AS

SELECT \* FROM ACCOUNT

View of all name of customers who live in HaNoi?

CREATE VIEW Customer\_Hanoi

AS

SELECT CusName

FROM Customers

WHERE CusAddress LIKE '%Ha Noi%';

Create View that sort by descending order of product code customers sell with products with a price-suggest greater than 1000000.00?

CREATE VIEW Product2\_mostcustomers

AS

SELECT TOP 3 CusID, Product2Name, PriceSuggest

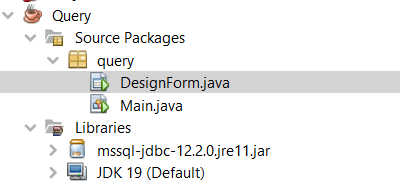
FROM Product2

WHERE PriceSuggest >=1000000.00

ORDER BY CusID DESC;

## Connect SQL server with Java.

1. Create a login user in SQL Sever
2. Create Query Project
3. Downloading JDBC and adding Library (jdbc-12.2.0.jre11.jar) into Apache NetBeans.



**Figure III‑10 Display Project file and Libraries**

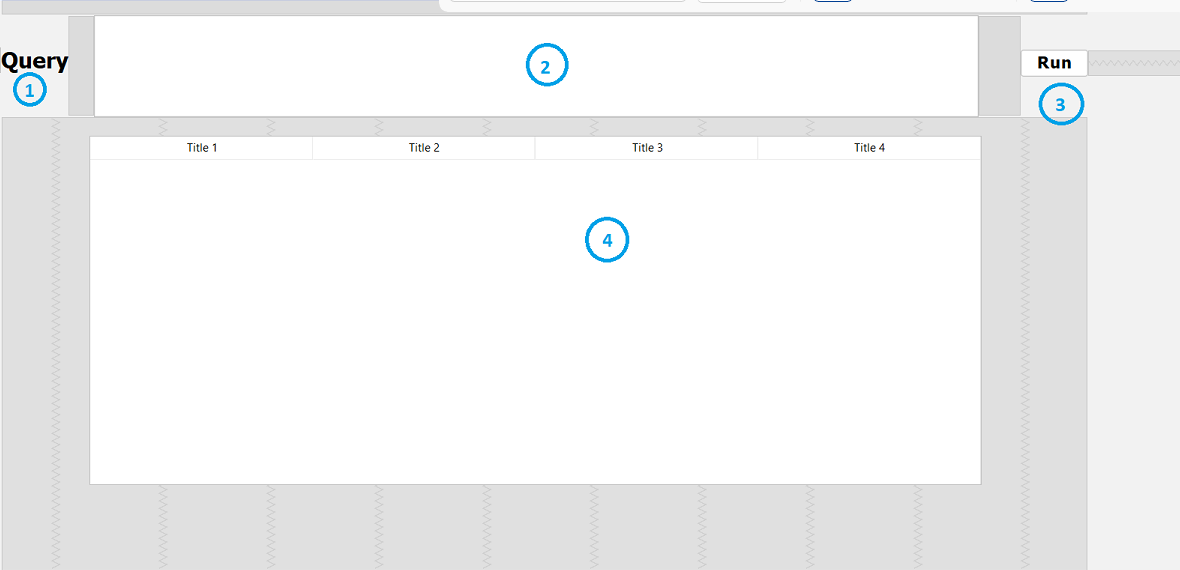
1. Connecting to SQL server then getting localhost, username, password to connect with Java in NetBeans
2. Designing a user interface form by using NetBeans common controls:

1. Label

2. Text Field

3. Button

4. Text Table



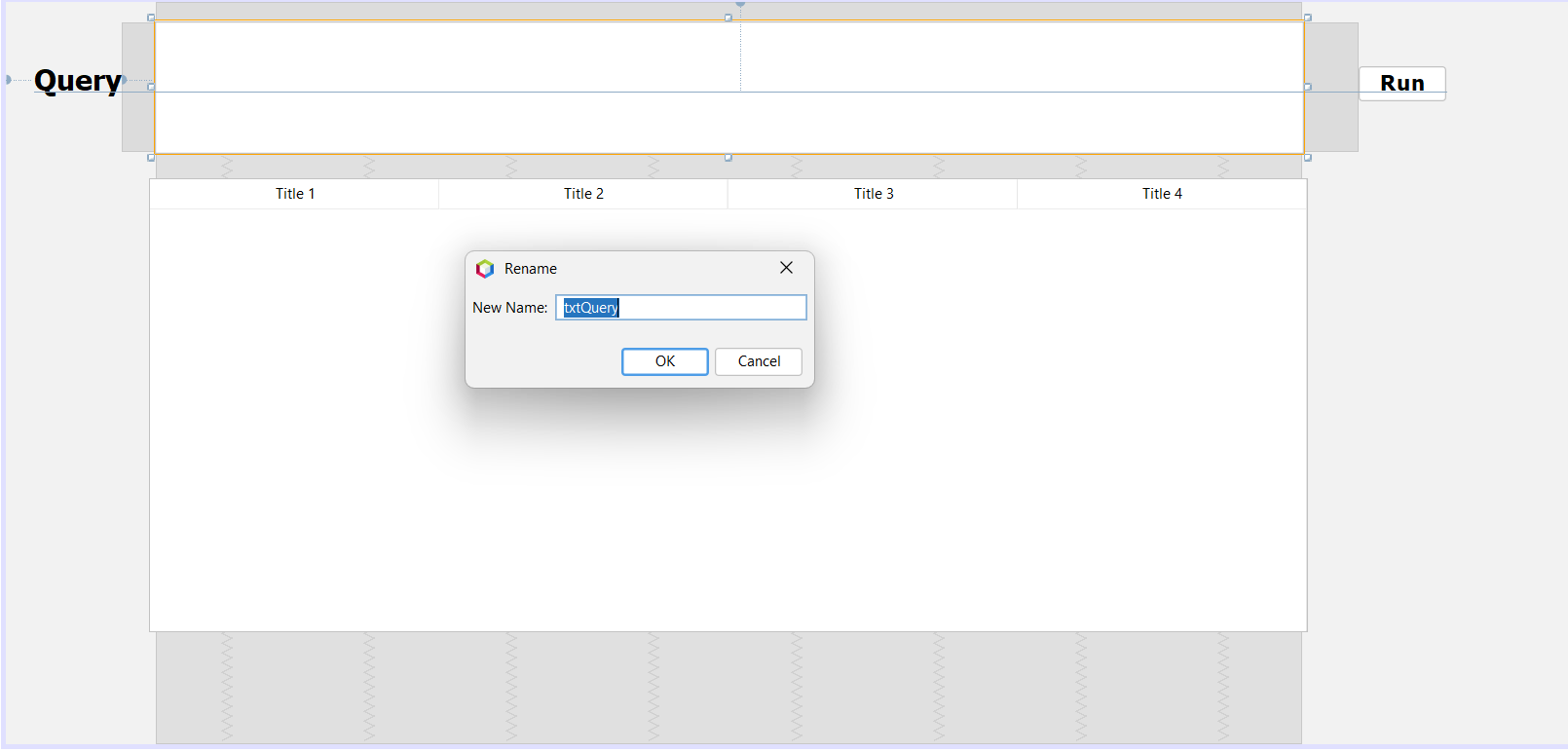
**Figure III‑11 Design Form**

1. Change control’s Variable Names to suitable. Right click the control, select Properties. From the Code tab, modify the field Variable Name:

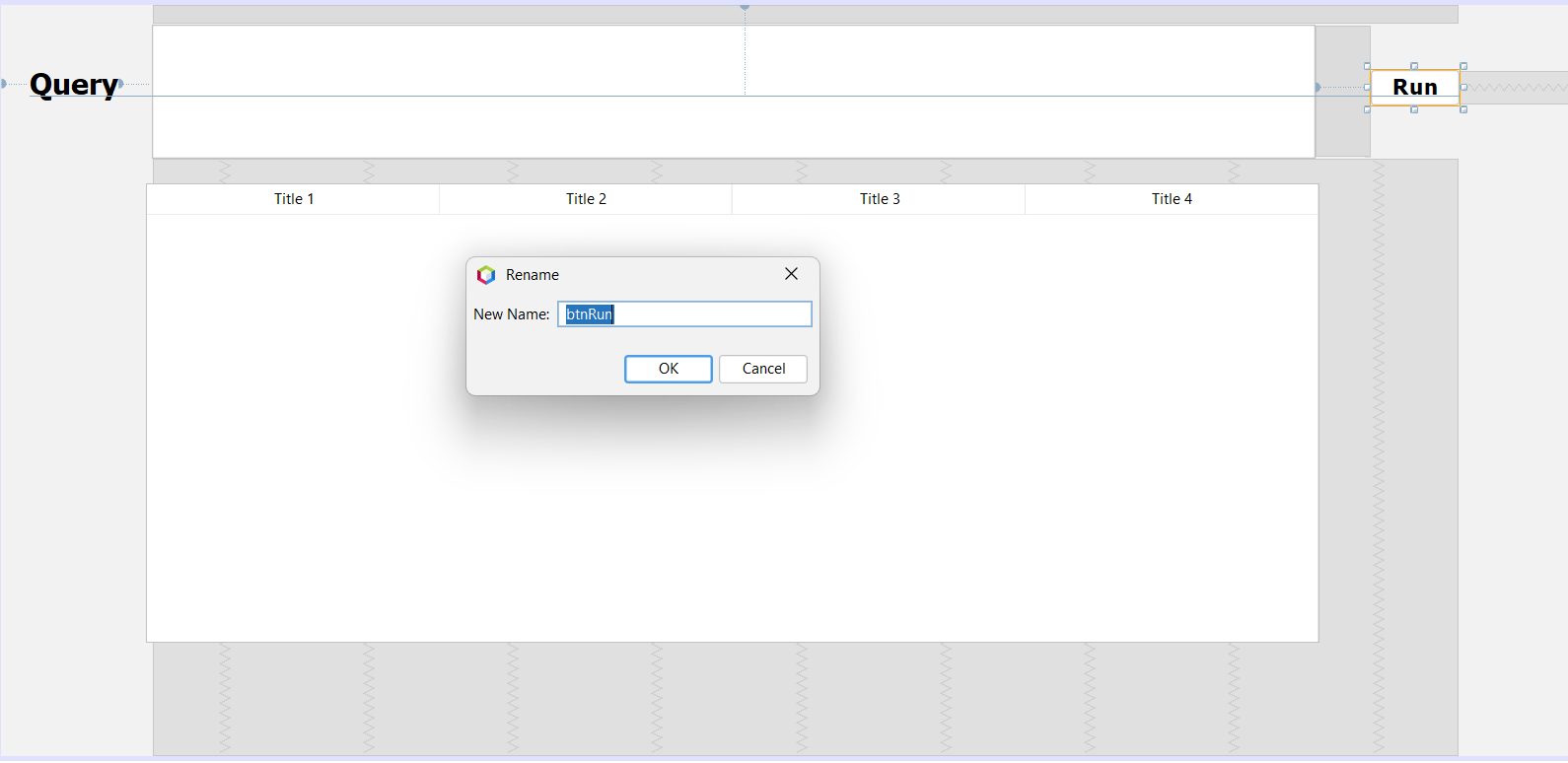
1. Text Field: txtQuery

2. Button: btnRun

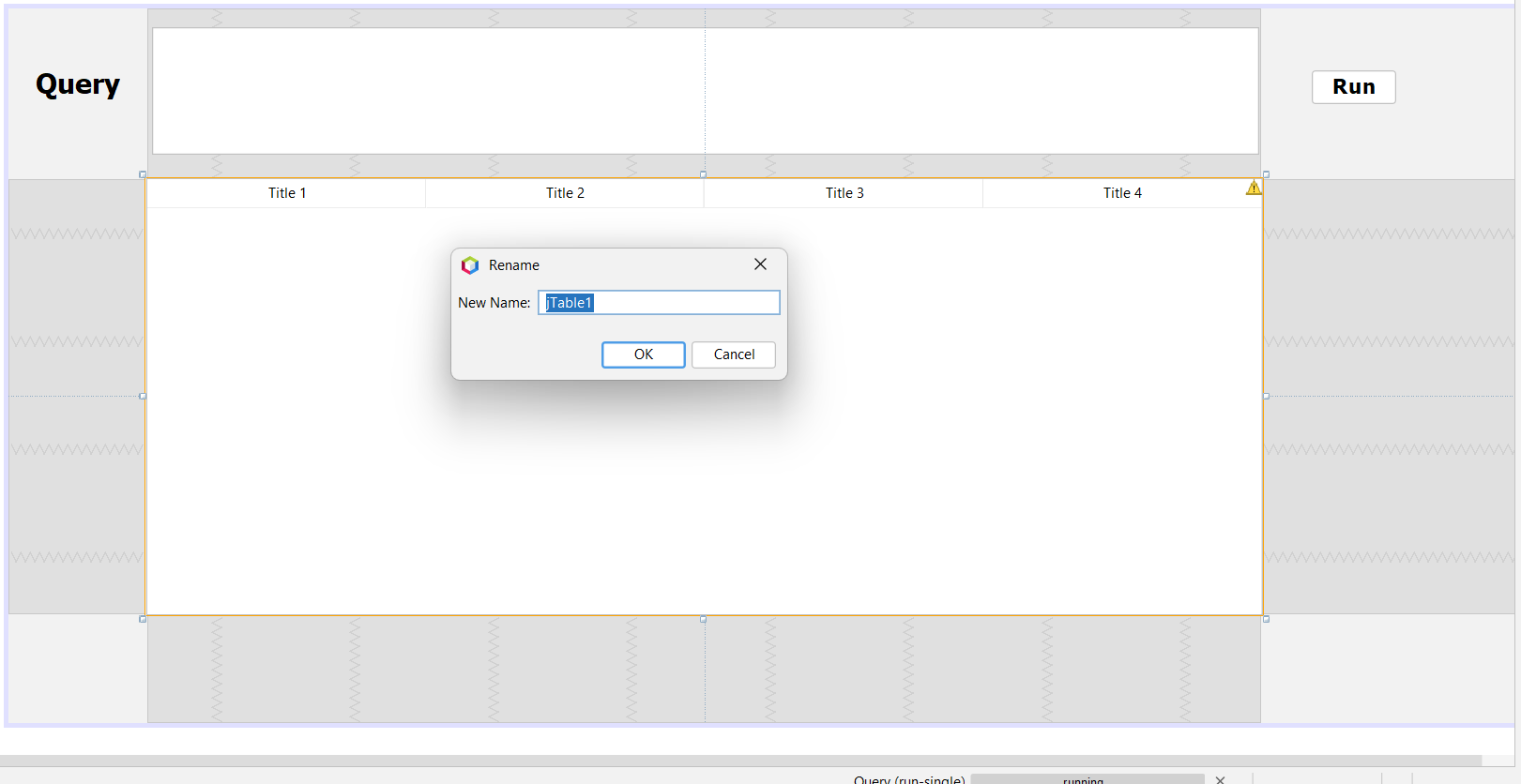
3. Text Table: jTable1



**Figure III‑12 Rename Text Field**



**Figure III‑13 Rename Button**



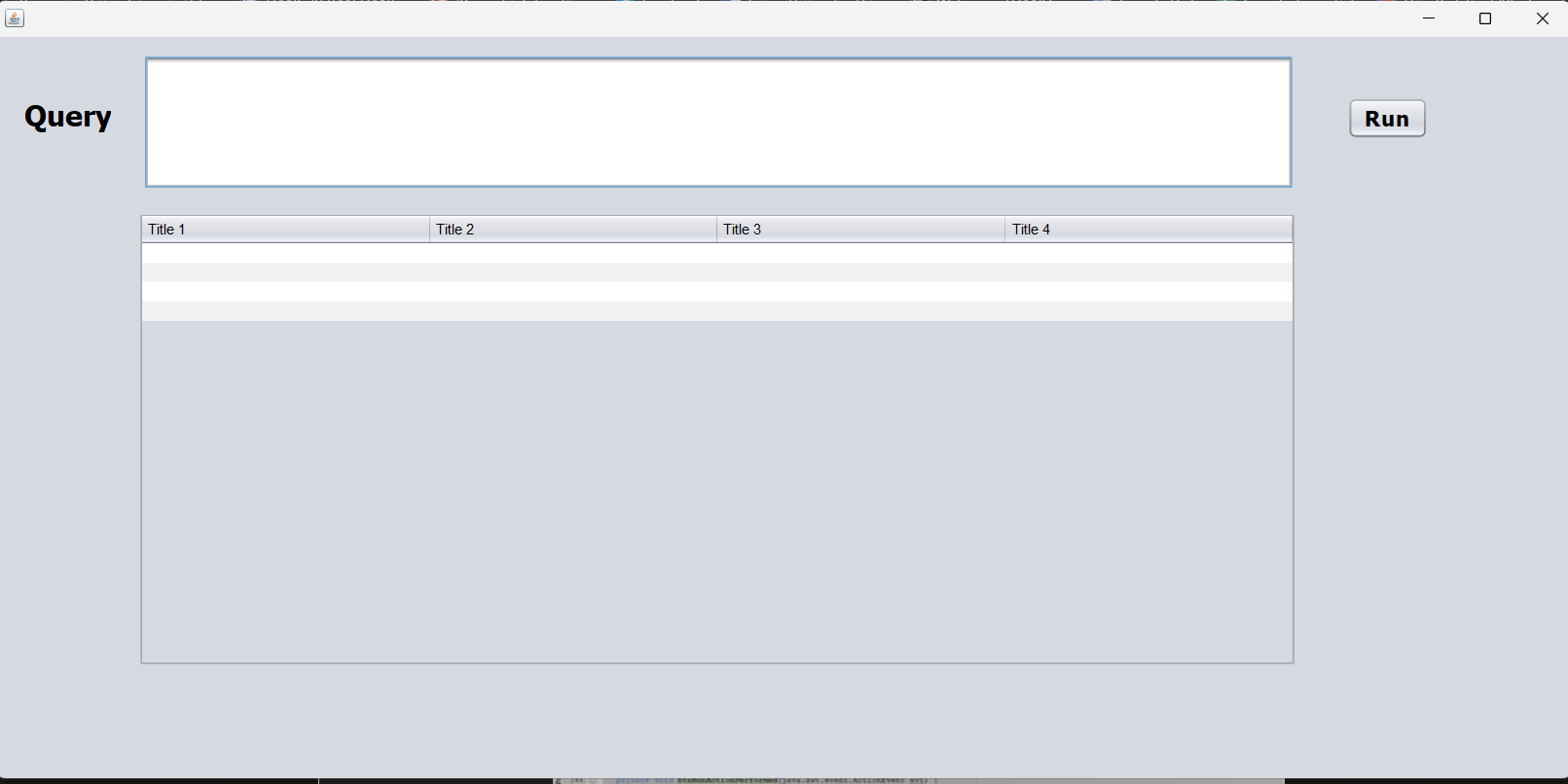
**Figure III‑14 Rename Text Table**

1. Select and the file Main.java, from the main function, enter code to display the newly created form.



**Figure III‑15 Main class**

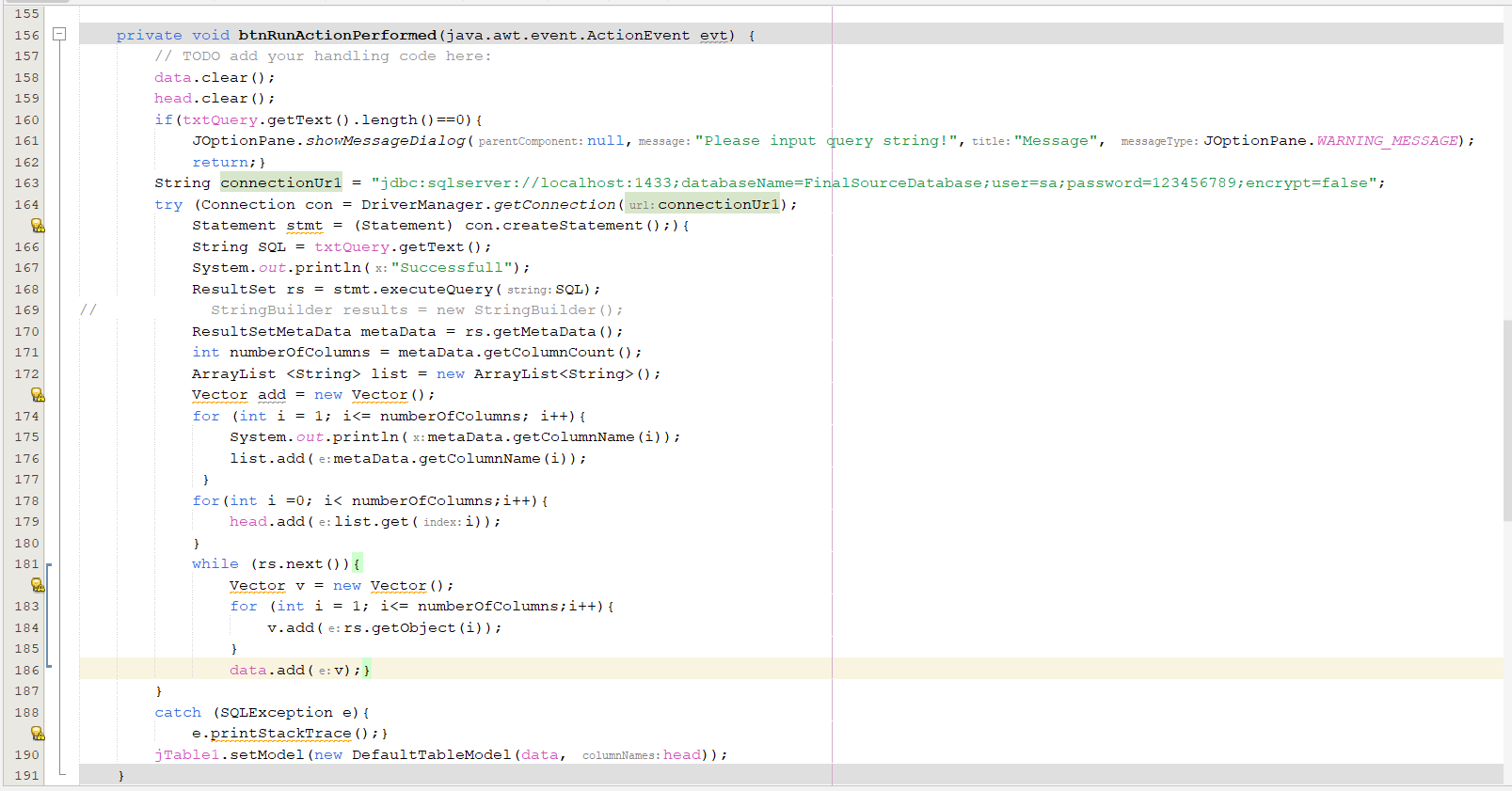
1. Run Project Query and output:



**Figure III‑16 Interface form to Query**

1. Connecting the Database:

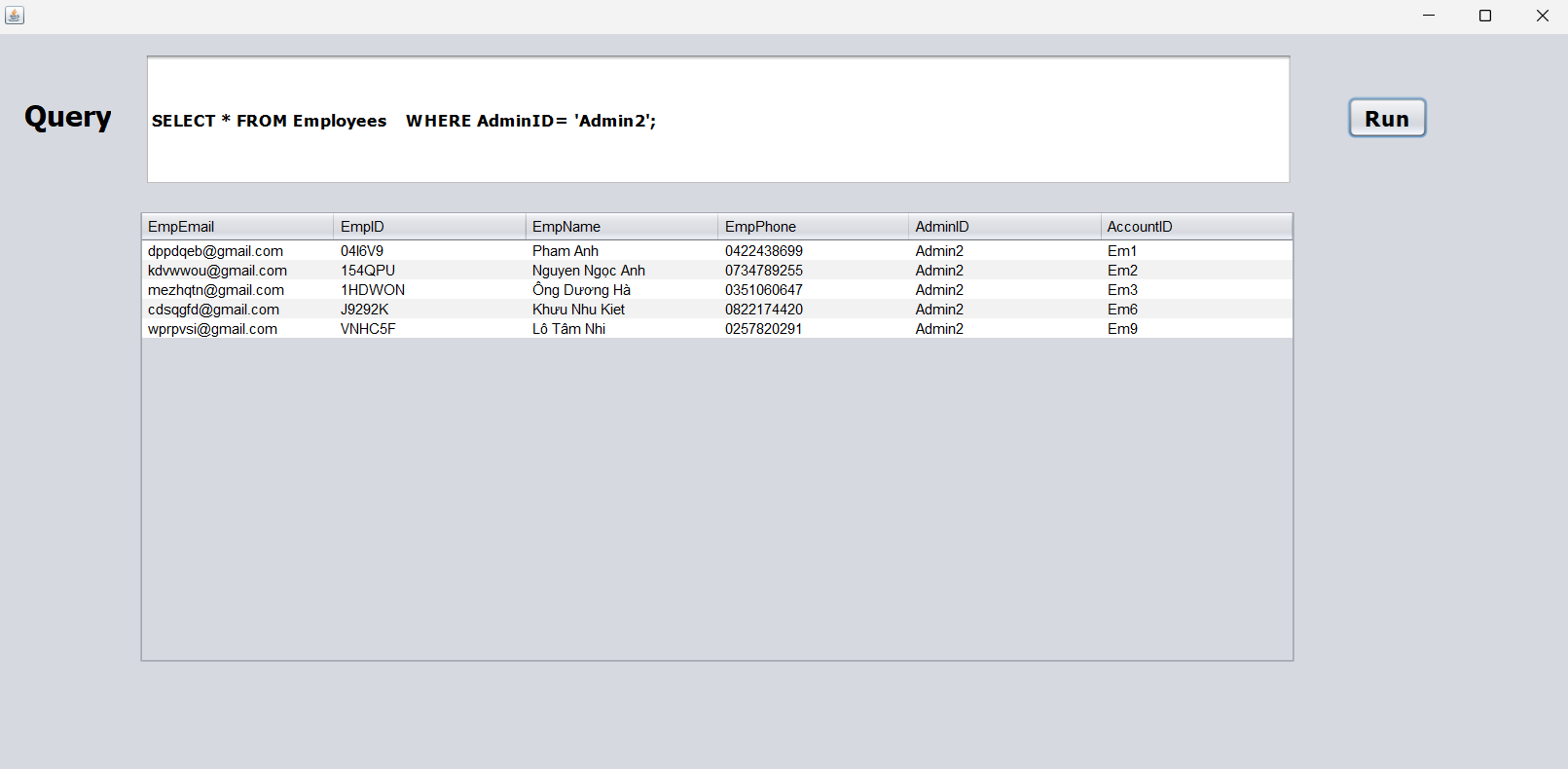
* Double click on button in the form to create an on click event for the button.
* Entering the code on this



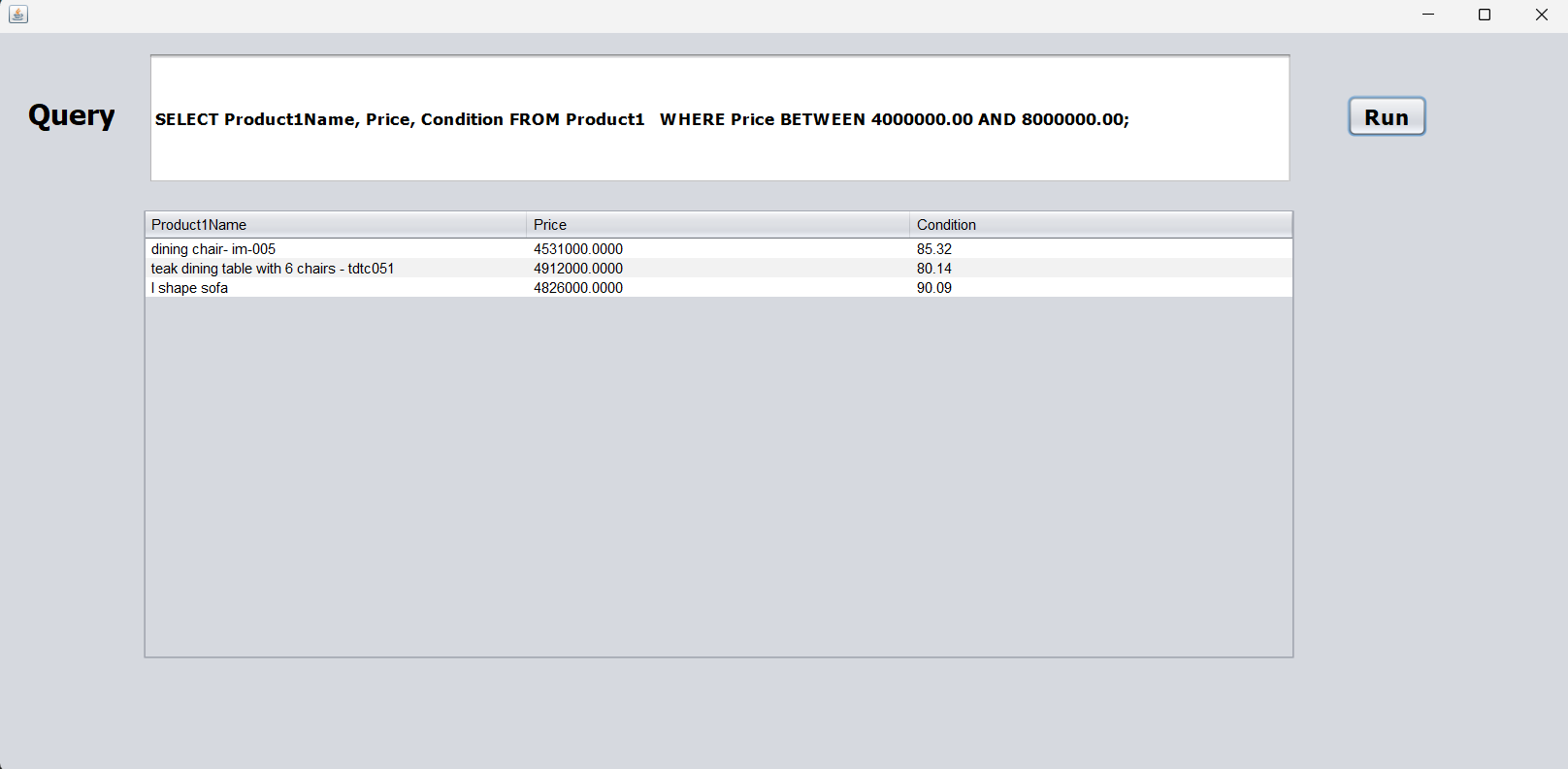
**Figure III‑17 Source code to connect & query**

1. Execution result.

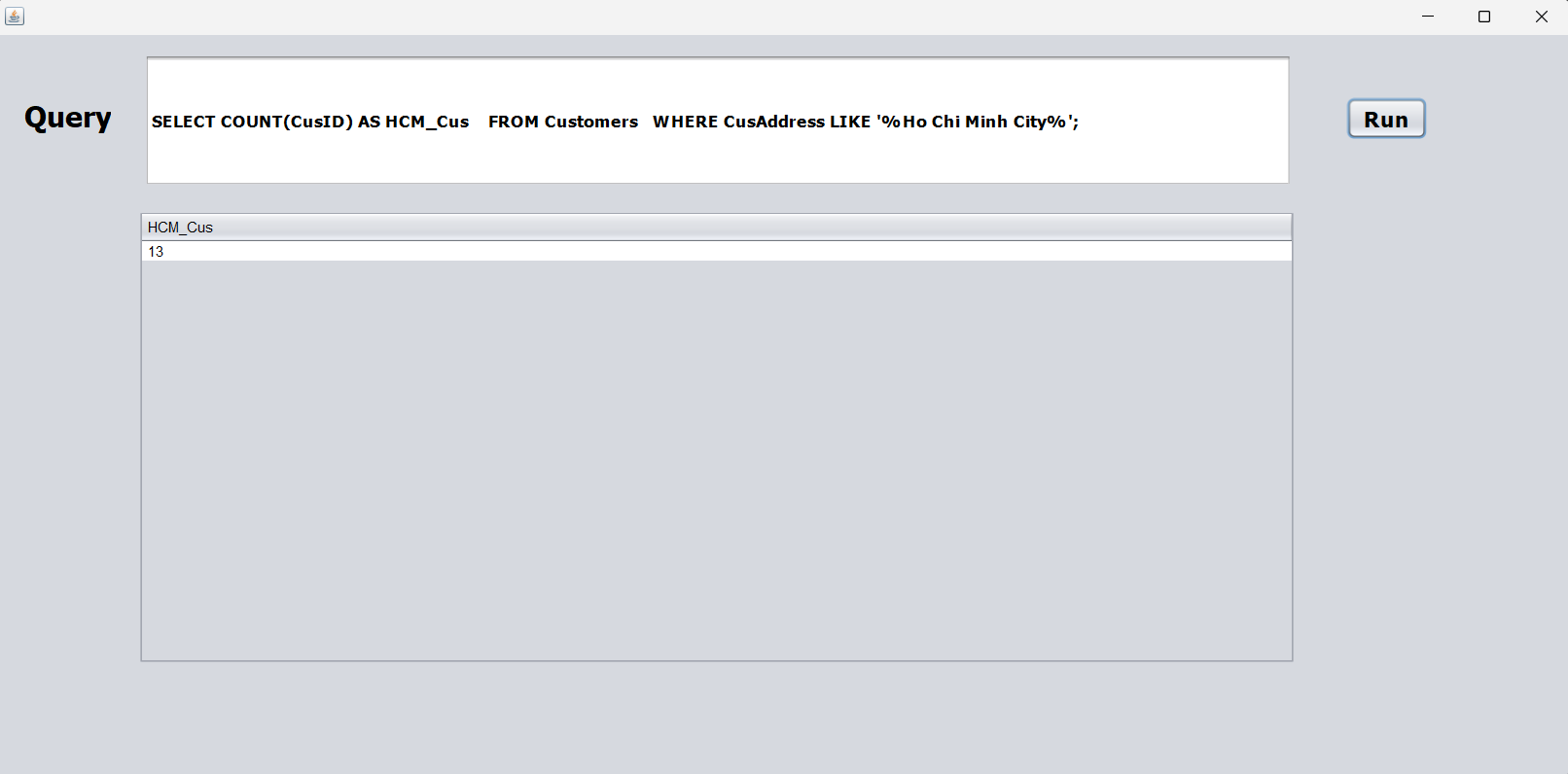
## DEMO



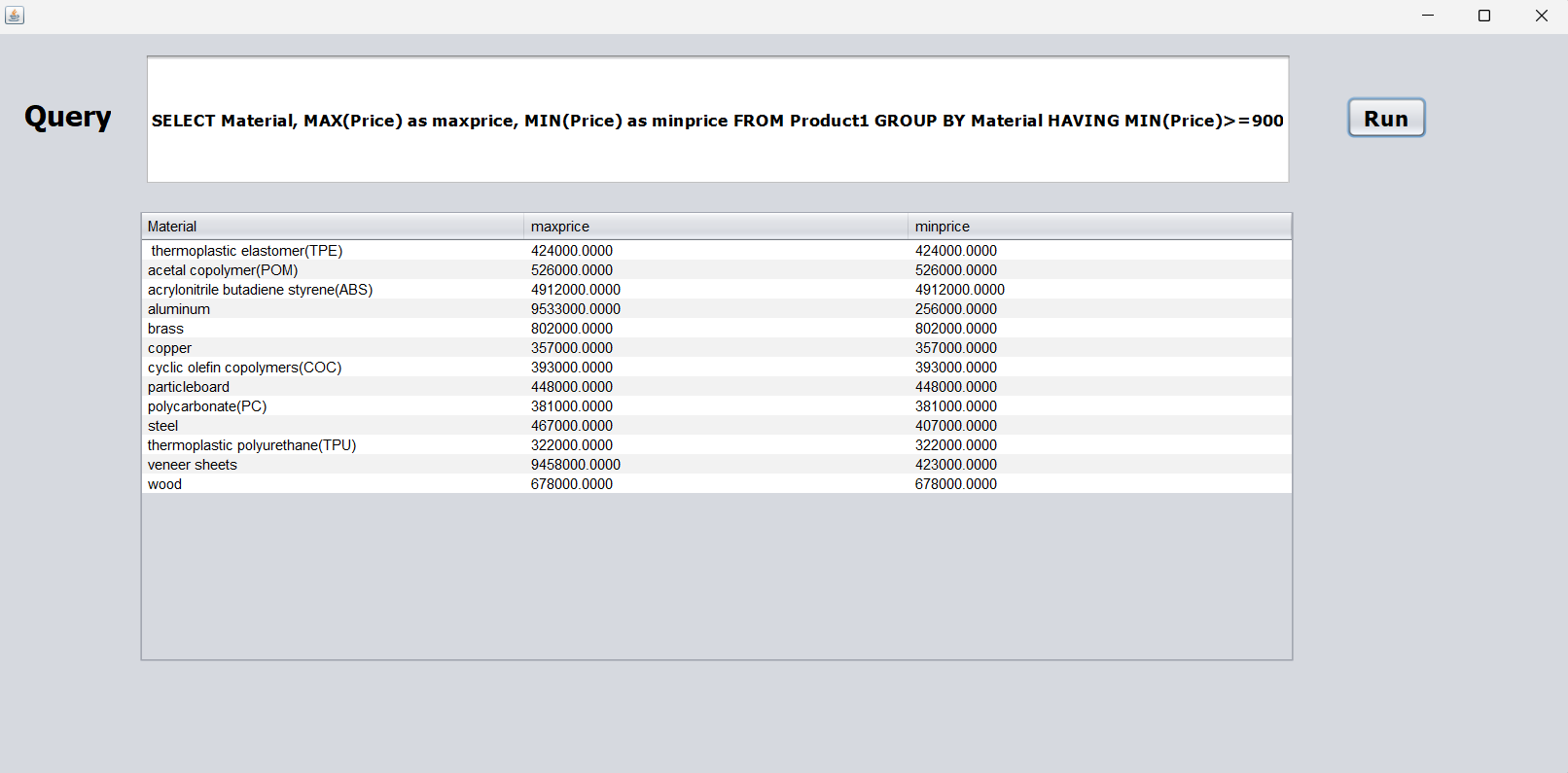
**Figure III‑18 Query Demo 1**



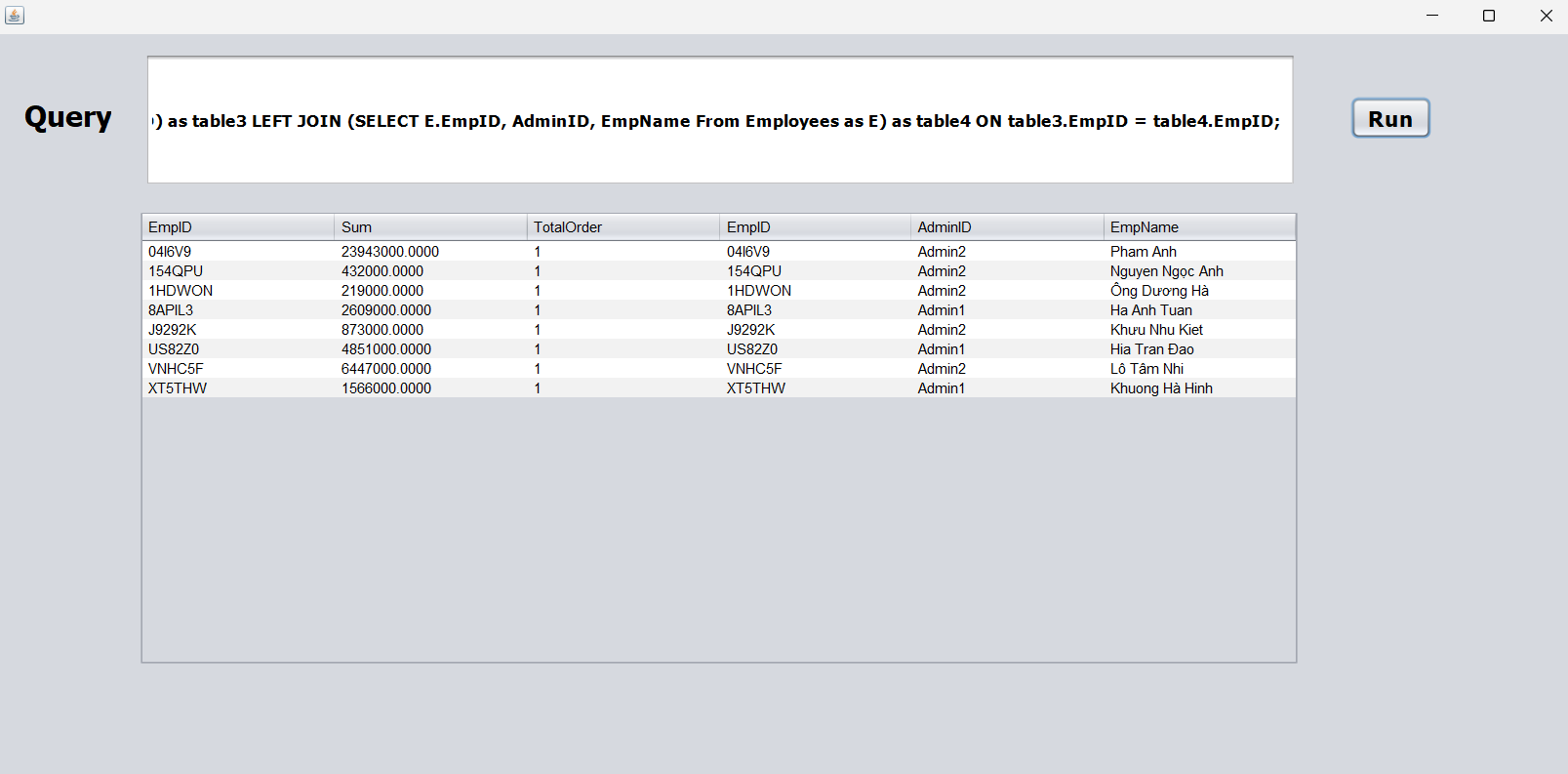
**Figure III‑19 Query Demo 2**



**Figure III‑20 Query Demo 3**



**Figure III‑21 Query Demo 4**



**Figure III‑22 Query Demo 5**

# CONCLUSION

## Summary

Our team utilized structural-driven analysis techniques to create and execute this website project. The outcomes we achieved include:

* Theoretical:
* Establishing the project’s objectives.
* Drawing the Entity Relationship Diagram (ERD) and relational model.
* Analyzing the problem and applying learned methodologies.
* Creating a database.
* Creating interfaces to manage data storage and updates.
* Program:
* Using SQL Server to create databases.
* Using Java in Apache NetBeans IDE (or IntelliJ) to build interfaces and connecting databases from SQL server.
* The system has been tested and meets all user requirements.
* However, our system is not yet highly professional. The program interface is not very attractive and needs improvement to look more professional.

## What we learn after project

The members of our group have gained a lot of knowledge after finishing this project, including:

* Relational model and Algebra SQL.
* Create a database ER.
* Insert data to a database.
* Make a query Server SQL
* Connect Java Display form and SQL Server
* Additionally, each group member will benefit from this collaboration by:
* How to lay out a whole report's structure

For each person, create a particular implementation strategy with a deadline.

* Analyze to post, ask for a topic, ask for information.

## Completed work & Future works.

* Understanding how to use SQL Server effectively.
* Understand how to develop forms with NetBeans.
* Understand how to add a library, log in as a user, and connect to SQL Server using Java.
* Recognize the fundamentals of creating a sales organization of any size.
* Encountering issues when the Project is being implemented.
* Utilize the best option that is appropriate for the group's workload.

The analysis and design of the Demo have essentially been accomplished quite extensively due to time constraints and a lack of practical expertise, but it is not possible to fully detail all facets of the problem. Create a database, but only for typical cases; more cases are required for this project to function more effectively and completely.

In the event that something goes wrong, our group will finish this Demo more thoroughly, making it simpler to query the data and present the results.

# REFERENCES

1. Vietnamese Person Name - by YenTrang - updated in 2021: <https://www.kaggle.com/datasets/minhnguynphc/vietnamese-person-name>
2. Furniture Images Dataset -by Lasal Jayawardena - updated in 2022: <https://www.kaggle.com/datasets/lasaljaywardena/furniture-images-dataset>
3. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *ISE Database System Concepts*.
4. Hoffer, J. A., Ramesh, V., & Topi, H. (2016). *Modern Database Management*. Pearson.
5. Elmasri, R., & Navathe, S. B. (2006). *Fundamentals of Database Systems, 5th Edition*.
6. Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems*.
7. Elmasri, R., & Navathe, S. (2011). *Fundamentals of Database Systems*. Addison Wesley Longman.
8. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database Systems: The Complete Book*. Prentice Hall.