**-**

**VIETNAM NATIONAL UNIVERSITY – HCM**

**INTERNATIONAL UNIVERSITY**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEER**

SEMESTER 1 (2023-2024)

**IT079IU**

**PRINCIPLE OF DATABASE MANAGEMENT PROJECT**

**Topic ID: 1**

**Topic name: E-commerce Platform**

DECEMBER 2023

**GROUP 8**

| **No** | **Full Name/ Student Name** | **Student ID** |
| --- | --- | --- |
| 01 | Dinh Thi Thanh Hang | ITDSIU21083 |
| 02 | Dinh Vu Ngoc Linh | ITDSIU21095 |
| 03 | Le Thi Tuyet My | ITDSIU21005 |
| 04 | Trinh Le Bich Hang | ITDSIU21084 |
| 05 | Nguyen Mai Phuong | ITDSIU20105 |
| 06 | Nguyen Huu Thuy | ITCSIU22141 |

**TABLE OF CONTENTS**

**CHAPTER 1: INTRODUCTION**

[**1. Background**](#_heading=h.6yqpluv4oppq) **10**

[**2. Executive Summary**](#_heading=h.xlgi3thb3ssf) **10**

[**3. The Purposes**](#_heading=h.v6hu4y3o9447) **10**

**CHAPTER 2: THEORETICAL IMPLICATION**

[**1. ERD-EERD (Entity Relationship Diagram - Enhanced Entity Relationship Diagram)**](#_heading=h.u3gr87hvm7cp) **11**

[1.1. Abstract](#_heading=h.90sg1apkayle) 11

[a. ERD](#_heading=h.mgq4h4nzo5rr) 11

[b. EERD](#_heading=h.eg7dtutbzar) 11

[1.2. Component of ERD](#_heading=h.8jbxayu0ej1x) 12

[a. Entities](#_heading=h.4x4jh2dz3gqa) 12

[b. Attributes](#_heading=h.bnhmf5qt3cfy) 12

[c. Relationship](#_heading=h.58acrxmflyc2) 13

[d. Cardinality](#_heading=h.vcr4edqxedgu) 13

[e. Notation](#_heading=h.sjrpazka1ydm) 15

[f. Specialization and Generalization](#_heading=h.flboz6354o4k) 16

[1.3. ERD - EERD application](#_heading=h.spw6oi8kqzrv) 16

[**2. Database normalization**](#_heading=h.f4b78wohlku6) **17**

[**3. Convert ERD to Relational Model**](#_heading=h.7y2njbl6pne0) **18**

[**4. SQL Server**](#_heading=h.cob352bbc80e) **19**

[4.1. Basic of SQL](#_heading=h.8ygvndomhfjj) 19

[4.2. Data type](#_heading=h.4loojyw4v5sw) 21

[4.3. Integrity Constraints](#_heading=h.n2zfkryk2au2) 25

[**5. Basic of Database**](#_heading=h.5aflol12itty) **26**

[5.1. Key in Databases](#_heading=h.paewecjj7ooz) 26

[5.2. Querying Database](#_heading=h.swq3wif2w3p4) 27

[5.3. Operators](#_heading=h.hk378yc3khv7) 30

[**6. Connecting Java and SQL server database**](#_heading=h.pagmf4j8r871) **32**

[6.1. Database guide](#_heading=h.qe6mufew3khj) 32

[6.2. Connect Java to database](#_heading=h.oasv4jo6idc) 33

**CHAPTER 3: RELATIVE WORKS**

[**1. Analyst requirement of topic**](#_heading=h.rwv38l9ku0tb) **36**

[1.1. User (Customers) can:](#_heading=h.e6of5vgwjjum) 36

[1.2. User (Sellers) can:](#_heading=h.uwgbaus9le9t) 36

[1.3. Admin can:](#_heading=h.nx06cpi5lg7q) 36

[**2. Design ERD**](#_heading=h.sqpamey185yi) **36**

[**3. Convert ERD to Relational Model**](#_heading=h.qktuizpee6qa) **41**

[**4. Normalization**](#_heading=h.3b51kguq4j8e) **42**

[**5. Create database**](#_heading=h.pguyoz4m94ly) **43**

[**6. Collect & insert data**](#_heading=h.seddvqa1742r) **45**

[**7. Querying database**](#_heading=h.osu89hbrwshm) **49**

[**8. Create view**](#_heading=h.rvxttec6dfbh) **54**

[**9. Connect Sql to server with Java**](#_heading=h.oz47cp2w9yov) **55**

[**10. Demo**](#_heading=h.rd6ifotg31f5) **58**

**CHAPTER 4: CONCLUSION**

[**1. Summary**](#_heading=h.o0mpqbf8npp8) **60**

[**2. What we learned**](#_heading=h.9elw1xm7j6cz) **60**

[**3. Completed works & future works**](#_heading=h.10qcpuwqr7fz) **61**

**CHAPTER 5: REFERENCES**

**TABLE OF FIGURES**

**CHAPTER 2:**

**Figure II1.1 Example of ERD...........................................................................................9**

**Figure II1.2.1 One-to-One relationship..........................................................................11**

**Figure II1.2.2 One-to-Many relationship.......................................................................12**

**Figure II1.2.3 Many-to-One relationship.......................................................................12**

**Figure II1.2.4 Many-to-Many relationship....................................................................13**

**Figure II1.2.5 Notation (1)...............................................................................................13**

**Figure II1.2.6 Notation (2)...............................................................................................14**

**Figure II4.1 Exact Numerics Data Types.......................................................................20**

**Figure II4.2 Approximate Numerics Data Types..........................................................20**

**Figure II4.3 Date and Time Data Types.........................................................................20**

**Figure II4.4 Character Strings Data Types...................................................................21**

**Figure II4.5 Unicode Character Data Types.................................................................21**

**Figure II4.6 Binary Data Types......................................................................................22**

**Figure II4.7 Miscellaneous Data Types..........................................................................22**

**Figure II6.1 Steps to check connecting errors...............................................................33**

**CHAPTER 3:**

**Figure III2.1 Entities Identification...............................................................................35**

**Figure III2.2 Relationship Identification.......................................................................35**

**Figure III2.3 Mapping attributes to entities, relationships..........................................36**

**Figure III2.4 Value range for each attribute.................................................................37**

**Figure III2.5 Key attribute for Entities ........................................................................38**

**Figure III2.6 Cardinality identification.........................................................................38**

**Figure III2.7 ERD Diagram ...........................................................................................39**

**Figure III3.1 Entities.......................................................................................................39**

**Figure III3.2 Binary 1-to-N.............................................................................................40**

**Figure III3.2 Binary M-to-N...........................................................................................40**

**Figure III9.1 Library mssql-jdbc...................................................................................53**

**Figure III9.2 Display the form.......................................................................................54**

**Figure III9.3 Code for connecting to Java.....................................................................55**

**TABLE OF TABLES**

**CHAPTER 2:**

**Table II4.1 Data Definition Language (DDL)...............................................................18**

**Table II4.2 Data Manipulation Language (DML)........................................................19**

**Table II4.3 Commonly used Integrity Constraints.......................................................23**

**Table II5.1 Commonly used Query................................................................................25**

**Table II5.2 Commonly used Operators.........................................................................28**

**RESEARCH PLAN**

| **No** | **Timeline** | **Task** | **Responsibility** |
| --- | --- | --- | --- |
| 1 | 5/11-8/11/2023 | Analyze the requirements | All of Members |
| 2 | 15/11-18/11/2023 | Create ERD | Thanh Hang |
| 3 | 19/11-22/11/2023 | Convert ERD To Relational Model | Thanh Hang |
| 4 | 22/11-24/11/2023 | Normalization | Thanh Hang |
| 5 | 27/11/2023 | Feedback & Change | All members |
| 6 | 28/11/2023 | Create Database in SQL | Huu Thuy, Mai Phuong |
| 7 | 30/11-31/11/2023 | Feedback & Change | All of Member |
| 8 | 31/11-1/12/2023 | Collect & Insert Data | Huu Thuy, Mai Phuong |
| 9 | 2/12-5/12/2023 | Connect Java to SQL | Tuyet My, Ngoc Linh, Bich Hang |
| 10 | 5/12-10/12/2023 | Write Report | All members |
| 11 | 10/12-12/12/2023 | Demo | All members |
| 12 | 12/12-15/12/2023 | Presentation Preparation | All members |

CHAPTER 1: INTRODUCTION

# Background

Transactions like selling and buying products have always been a great need for both sellers and buyers in many places. However, purchasing things through traditional commerce platforms does not cater to many customers. For example, there are better fits than the traditional market for people who live far away from the commercial hub. As a result, an e-commerce platform is required by both the sellers and the buyers for a more personalized shopping experience, which is available for a wide range of customers and sellers, and a simplified purchasing process. Understanding the necessity of this topic, our team decided to create an e-commerce platform that satisfies all the following requirements.

# Executive Summary

This project aims to create an e-commerce platform in which customers can register a new account with their name, and password, and login into that account and search for a specific item in the product list, purchase one or multiple items. Besides that, the system will also take in the bank account name, and the bank account of customers, which enables buyers to purchase things in both cash and credit. Furthermore, each customer will have a unique user ID generated by the system to set them apart from other users. This project will also display the invoice of the order which includes the ordered items, quantity of each, subtotal, delivery fee, total, and transaction date.

# The Purposes

Throughout the project, our team got to have hands-on experience with ERD-EERD and relational Schema and got to know how to create, and manage a database through SQL and Java. The final goal of this project is to create a more personalized, fast, and convenient e-commerce platform, which shall bring out a better customer experience, and a chance for sellers to take part in e-commerce with a low start-up cost.

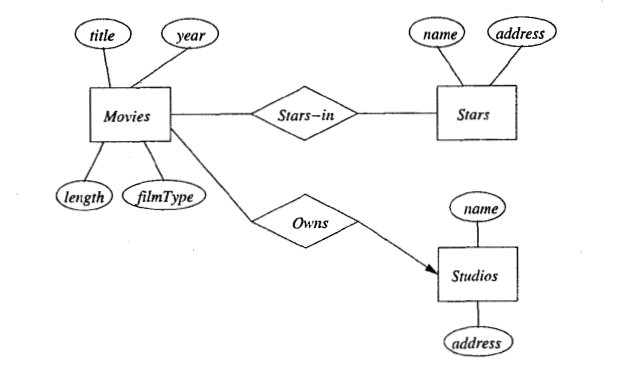
CHAPTER 2: THEORETICAL IMPLICATION

# ERD-EERD (Entity Relationship Diagram - Enhanced Entity Relationship Diagram)

## Abstract

### ERD

An entity relationship diagram (ERD) shows the relationships and characteristics of the entities in a system visually. Additionally, ERDs provide a concise and clear approach (such as shapes and notations) for displaying a database's structure, which facilitates stakeholders' understanding of the data model—developers, designers, and business users, among others. The normalization process, which involves structuring data to reduce redundancy and enhance data integrity, is another procedure that ERDs help with.



#### *Figure II1.1 Example of ERD*

### EERD

#### An EERD is an extension of the traditional ERD that adds new concepts and features to represent more complex relationships and limitations. It makes use of a number of ideas that are closely connected to object-oriented programming and design, such as entity classification and specialization/generalization (inheritance). Additionally, participation restrictions—which outline the minimum and maximum number of entities required to participate in a relationship—are supported by EERDs in addition to other constraints.

## Component of ERD

### Entities

#### Entities are real-world concepts or items that can have data about them. They are the primary "things" in the database and are usually nouns. An ERD typically uses rectangles to represent entities (double rectangles for Weak Entities). Every entity has its name labeled on it.

#### There are two types of entities: Strong Entity and Weak Entity.

#### A strong entity has a main key and can define itself without the participation of other entities.

#### A weak entity lacks a main key and is dependent on other entities to exist. This implies that it may have the main key of another entity as a foreign key.

### Attributes

#### Attributes are properties or characteristics of entities. Every set of data that may be included in a single entity is described by them. Attributes are depicted as ovals and are connected to their respective entities by lines. Each attribute is labeled with its name. For example, in the Student entity, these are attributes named: student\_id, student\_name, and student\_grade.

#### An attribute in the ERD can be classified into various types:

#### Simple attributes: Simple attributes cannot be subdivided any further.

#### Composite attribute: Composite attributes are made up of two or more simple attributes.

#### Multivalued attribute: Multivalued attributes can have more than one value.

#### Derived attribute: Derived attributes are based on other attributes and are not stored directly in the database.

#### Key attribute: An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes.

### Relationship

#### A relationship in a relational database is the association of two or more tables based on shared fields. Creating relations between tables is a way to organize data and maintain data integrity. Types of relationships include one-to-one, one-to-many, many-to-one, and many-to-many relationships. We use the diamonds in ERD to represent the relationship between two strong entities. The relation between one strong and one weak entity is represented by a double diamond.

### Cardinality

#### Cardinality indicates the number of entities with which another entity can be related through a relationship set. The mapping cardinality for a binary relationship set must be one of the following types.[1].

#### One-to-One (1:1): One instance of one entity is connected with exactly one instance of another entity in a one-to-one relationship, and vice versa. The notation often includes the number "1" near both ends of the relationship line.

#### 

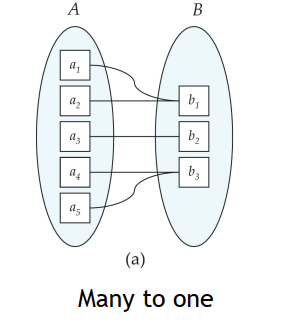
#### *Figure II1.2.1 One-to-One relationship*

#### One-to-Many(1:N): One instance of one entity is associated with zero or more instances of another entity in a one-to-many connection. Each instance of the second entity, on the other hand, is associated with exactly one instance of the first entity. "1" is near the entity on the "one" side, while "M" is near the entity on the "many" side.

#### 

#### *Figure II1.2.2 One-to-Many relationship*

#### Many-to-One (N:1): Many instances of one entity are associated with exactly one instance of another entity in a many-to-one connection. "M" (or an infinity symbol) is placed near the entity on the "many" side, and "1" is placed near the entity on the "one" side.



#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### *Figure II1.2.3 Many-to-One relationship*

#### Many-to-Many (M: N): Many instances of one entity can be associated with many instances of another entity in a many-to-many relationship, and vice versa. Near both ends of the relationship line, the notation contains "M" .

#### 

#### *Figure II1.2.4 Many-to-Many relationship*

### Notation

#### 

#### *Figure II1.2.5 Notation (1)*

#### 

#### 

#### *Figure II1.2.6 Notation (2)*

### Specialization and Generalization

#### Specialization: To further simplify things, specialization involves breaking things down into smaller components. Another way to put it is that with specialization, an entity is broken down into smaller entities according to its attributes. Inheritance occurs in Specialization as well.

#### Generalization: It operates according to the bottom-up approach. Lower level functions are merged in generalization to create higher level functions, or entities. To create advanced level entities, this process is repeated one more time.

## ERD - EERD application

1) Identify Entities: Identify the main entities in the system you are modeling. An entity is a real-world object or concept, sometimes, it can also be a noun.

2) Define Attributes: Identify and list the attributes (properties) that describe each entity. These characteristics must be atomic and directly tied to the entity.

3) Identify Relationships: Determine the relationships between different entities. Relationships can be one-to-one, one-to-many, or many-to-many.

4) Determine Cardinality: Define the cardinality of each relationship, specifying the maximum number of occurrences of one entity that can be associated with the other.

5) Identify Primary Keys: Choose a primary key for each entity. The primary key uniquely identifies each record in the entity.

6) Implement Foreign Keys: Identify relationships that involve foreign keys. A foreign key is a field in one table that refers to the primary key in another table.

7) Draw Initial ERD: Draw the fundamental ERD on paper or with a diagramming tool using graphical symbols (rectangles for entities, diamonds for relationships, ovals for attributes).

8) Normalize Entities: Check for attribute redundancy and normalize entities to reduce data duplication. This entails arranging data to reduce repetition and reliance.

9) Refine Relationships: Refine relationships based on input and make necessary modifications. Ensure that relationships appropriately indicate how entities are linked.

10) Add Additional Details (Optional): Include other features like restrictions, business rules, and annotations to provide a more complete knowledge of the database structure.

11). Review and Validate: Validate the ERD's accuracy and completeness with stakeholders and subject matter experts.

12) Document the ERD: In a clear and straightforward manner, document the entities, attributes, relationships, and any extra details. This documentation is intended to be used as a reference for database implementation.

# Database normalization

Data normalization is the process of examining given relation schemas based on their FDs and primary keys to achieve the desirable property of minimizing redundancy and minimizing insertion, deletion, and update anomalies. [2]

**a. 1NF – First Normal Form:**

It was designed to disallow the use of multivalued attributes, composite attributes, and their combinations. It specifies that the domain of an attribute must only contain atomic (simple, indivisible) values and that the value of every attribute in a tuple must be a single value from that attribute's domain.[3]

**b. 2NF – Second Normal Form:**

If a relation is in the 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.

**c. 3NF – Third Normal Form:**

A relational model R is in 3NF if each of its nonprime attributes fits both of the following conditions:

* It is entirely functionally dependent on every key of R.
* It is non-transitive, reliant on every R key.

# Convert ERD to Relational Model

**Stage 1:** Mapping Entity

Make a table for each entity, and the attributes of entities should be converted into table fields with the appropriate data types. Make a primary key declaration for each entity.

**Stage 2**: Mapping Weak Entity Sets

Make a table for the weak entity set. Add all of its attributes as fields to the table. Include the primary key for identifying the entity set. And, declare all foreign key constraints..

**Stage 3:** Mapping Relationship

* Case 1: Mapping binary relationship 1-to-1(1:1)

Use the primary key of one table as a foreign key in the other.

* Case 2: Mapping binary relationship 1-to-Many and Many-to-one(1:N)/(N:1)

The entity on the "many" side of the connection can only be connected with one entity on the "one" side; we will choose to capture the relationship by storing the "one" side table's primary key in the "many" side table.

* Case 3: Mapping binary relationship Many-to-Many(M:N)

In the event of a many-to-many relationship, a join table comprising details about the two entities' relationship will be created. According to that version, the primary key is the primary key shared by the two entities involved in the relationship. Each record in the join table represents a combination of related records from the original two entities

**Stage 4:** Mapping Hierarchical Entities

* Multiple relations: If there are subclasses and superclasses, the method typically assumes that each parent has a unique id. The relation should be utilized for disjoint if subclass only.
* Many NULLs can result from a single relation with a single type of attribute, and only when the relation is disjoint. Conversely, we might be disjoint in a relation with many types of attributes.

# SQL Server

## Basic of SQL

Data Definition Language (DDL) is a subset of SQL commands that is used to define and manage structures in a database. The primary Data Definition Language (DDL) is shown in ***Table II4.1***.

Data Manipulation Language (DML) is a subset of SQL commands that is used for accessing and manipulating data in a database. The primary Data Manipulation Language (DML) is shown in ***Table II4.2***.

***Table II4.1 DATA DEFINITION LANGUAGE (DDL)***

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| **CREATE** | Create a new SQL database, a new table in a database, a view of a table, or other objects in the database. | CREATE DATABASE *database\_name*;  CREATE TABLE *table\_name* (  *column\_1 datatype*,  …  *column\_N datatype*,  PRIMARY KEY (one or more columns)  ); |
| **ALTER** | Modifies an existing database object, for instance, by changing a table’s structure, adding or removing columns, modifying a column’s data type, or adjusting integrity constraints.. | ALTER TABLE *table\_name*  -- ADD column  ADD *colume\_name datatype*;  -- DROP column  DROP COLUMN *column\_name*;  -- RENAME column  RENAME COLUMN *old\_name* to *new\_name*;  -- ALTER datatype  ALTER COLUMN *column\_name datatype*; |
| **DROP** | Drops an existing SQL database, deletes an entire table, a view of a table, or other objects in the database. | DROP DATABASE *database\_name*;  DROP TABLE *table\_name*;  DROP VIEW *view\_name*; |

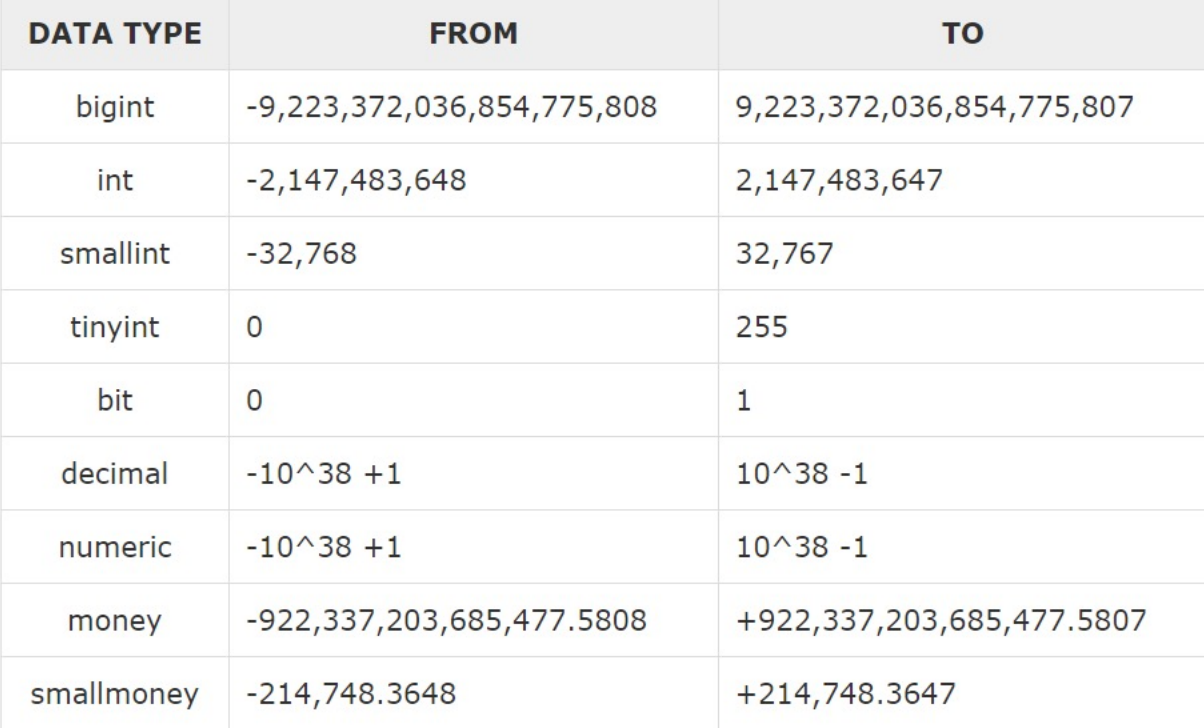
***Table II4.2 DATA MANIPULATION LANGUAGE (DML)***

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| **SELECT** | Retrieves certain records from one or more tables. | SELECT *column1*, *column2*, …  FROM *table\_name*; |
| **INSERT** | Creates a record. It is used to add new rows of data to a table in the database. | INSERT INTO *table\_name* (*column1*, *column2*,...)  VALUES (*value1*, *value2*, *value3*,...); |
| **UPDATE** | Modifies records. Update existing data within a table. | UPDATE *table\_name*  SET *column1* = *value1*, *column2* = *value2*, …  WHERE *condition*; |
| **DELETE** | Delete one or more records from the database tables based on the condition specified in the WHERE clause. | DELETE FROM *table\_name*  WHERE *condition*; |

## Data type

In SQL, every column, local variable, expression, and parameter is associated with a distinct data type. A data type is a property that delineates the kind of data an object can accommodate, such as integer data, character data, monetary data, date and time data, binary strings, etc.

Data types in SQL are organized into the following categories: Exact numerics (***Figure II4.1***), Approximate numerics (***Figure II4.2***), Date and time (***Figure II4.3*** ), Character strings (***Figure II4.4*** ), Unicode character strings (***Figure II4.5*** ), Binary (***Figure II4.6*** ), Miscellaneous (***Figure II4.7*** ).

****

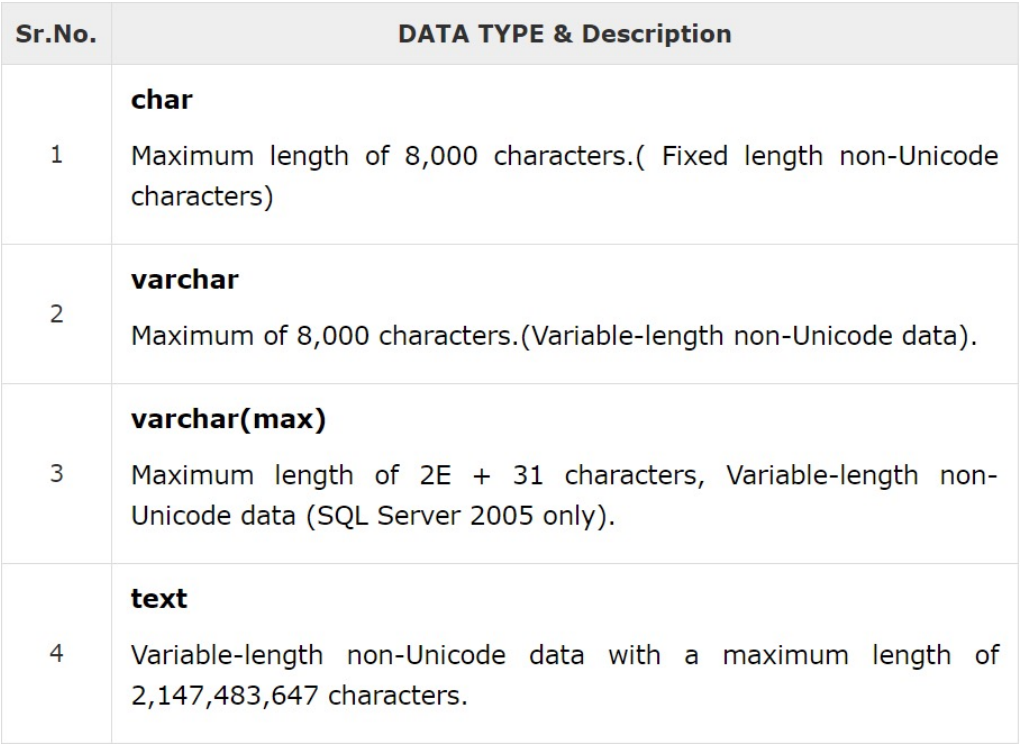
***Figure II4.1 Exact Numerics Data Types***

******

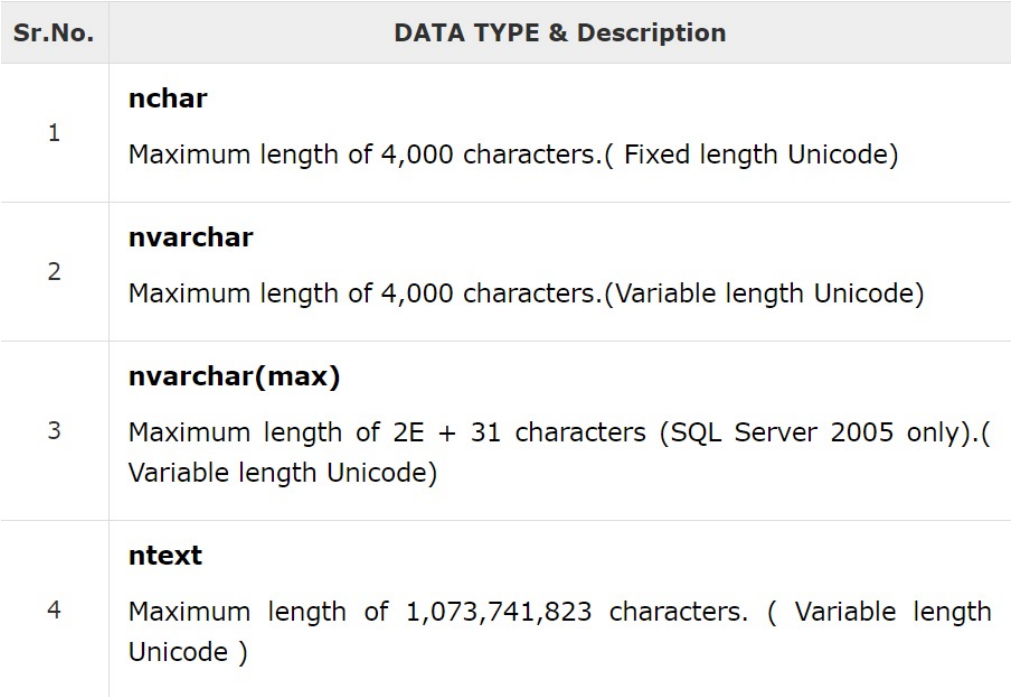
***Figure II4.2 Approximate Numerics Data Types***

******

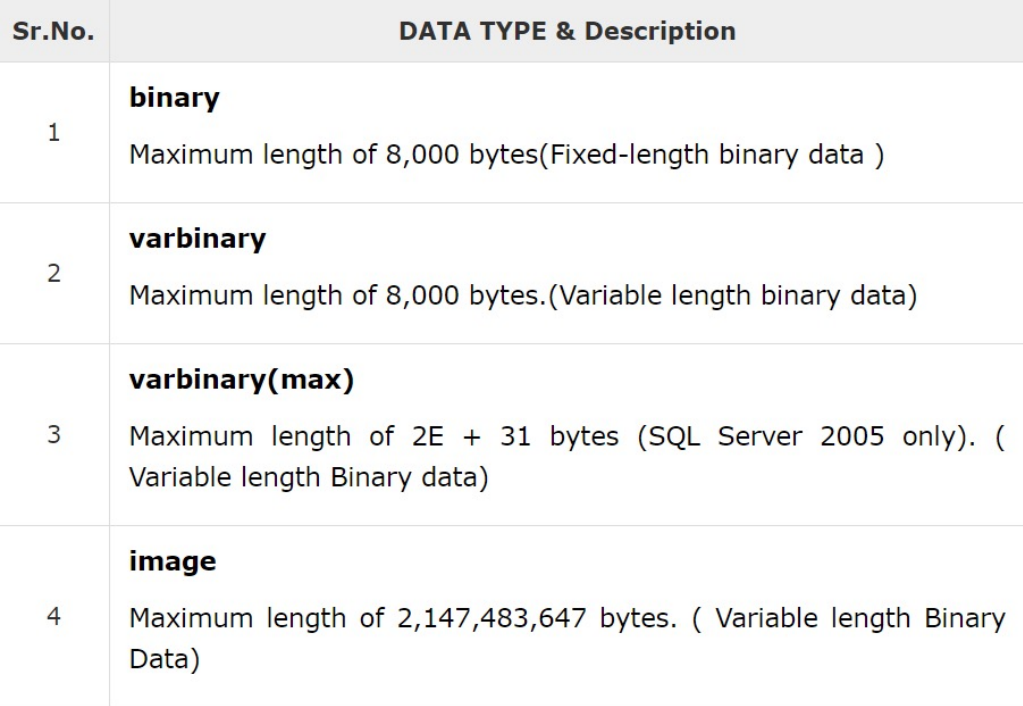
***Figure II4.3 Date and Time Data Types***

******

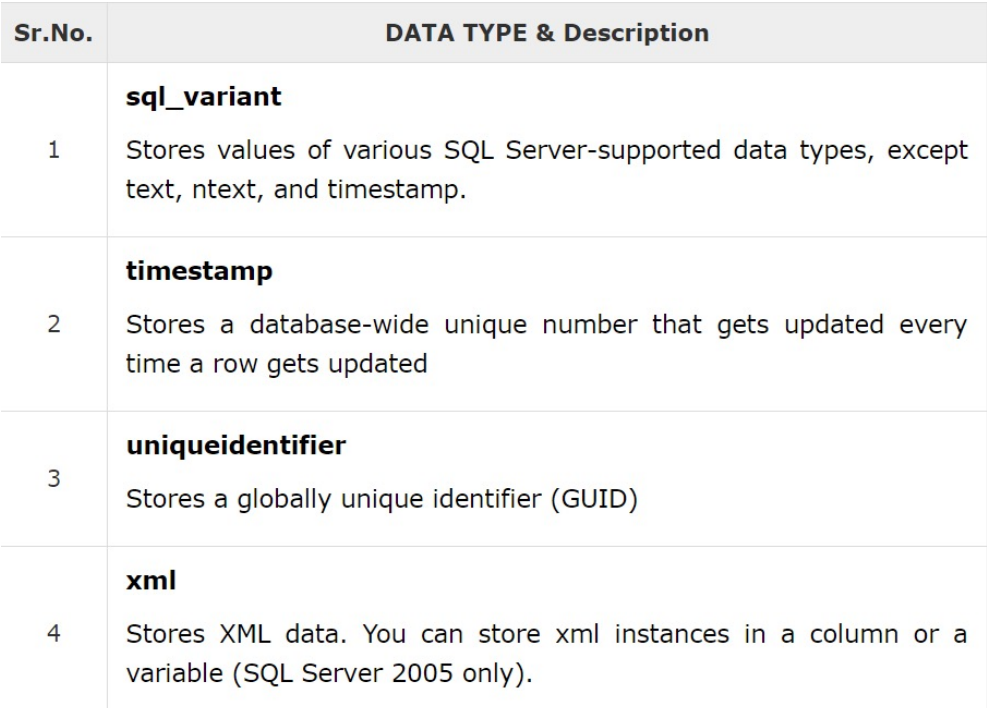
***Figure II4.4 Character Strings Data Types***

******

***Figure II4.5 Unicode Character Data Types***

******

***Figure II4.6 Binary Data Types***

******

***Figure II4.7 Miscellaneous Data Types***

## Integrity Constraints

In SQL, a constraint serves as a guideline enforced on the data within a table to maintain its accuracy and consistency. These constraints can be implemented either at the column level or at the table level. If any action on the data violates the constraint, the action is immediately halted.

A variety of commonly used constraints in SQL are shown in ***TABLE II4.3***.

***Table II4.3 Commonly used Integrity Constraints***

| **Constraint** | **Description** | **Syntax** |
| --- | --- | --- |
| **NOT NULL** | Ensures that a column cannot have a NULL value. | CREATE TABLE *table\_name* (  *column1 datatype* NOT NULL,  *column2 datatype* NOT NULL,  …  ); |
| **UNIQUE** | Ensures that all values in a column are different. | CREATE TABLE *table\_name* (  *column1 datatype* UNIQUE,  *column2 datatype* UNIQUE,  …  ); |
| **DEFAULT** | Provides a default value for a column when none is specified. | CREATE TABLE *table\_name* (  *column1 datatype* DEFAULT,  *column2 datatype* DEFAULT,  …  ); |
| **PRIMARY KEY** | Uniquely identifies each row or record in a table. | CREATE TABLE *table\_name* (  *column1 datatype* PRIMARY KEY,  *column2 datatype*,  …  ); |
| **FOREIGN KEY** | Uniquely identifies a row in any of the given tables. | CREATE TABLE *table\_name2* (  *column1 datatype*,  *column2 datatype*,  …  FOREIGN KEY (*column1*)  REFERENCES *table\_name1* (*column1*)  ); |
| **CHECK** | Ensures that all the values in a column satisfies certain conditions. | CREATE TABLE *table\_name* (  *column1 datatype*,  *column2 datatype*,  *column3 datatype* CHECK (*condition*),  …  ); |
| **INDEX** | Used to create and retrieve data from the database very quickly | CREATE INDEX *index\_name*  ON *table\_name* (*column1*, *column2*,...); |

# Basic of Database

## Key in Databases

**Primary Key**: This is a unique identifier for each record in a table. An entity can contain multiple keys, but the most suitable one becomes the primary key.

**Candidate Key**: This is an attribute or set of attributes that can uniquely identify a tuple. Except for the primary key, the remaining attributes are considered candidate keys.

**Super Key**: This is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

**Foreign Key**: These are the columns of a table used to point to the primary key of another table.

**Alternate Key**: There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key.

## Querying Database

A variety of commonly used queries in SQL are shown in ***TABLE II5.1***.

***Table II5.1 Commonly used Query***

| **Query** | **Description** | **Syntax** |
| --- | --- | --- |
| **SELECT**  [1] | Utilized to retrieve data from a database. The retrieved data is then stored in what is known as a result-set table. | SELECT *column1*, *column2*, …  FROM *table\_name*; |
| **WHERE**  [1] | Used to filter records. It is used to extract only those records that fulfill a specified condition. | SELECT *column1, column2,* ...  FROM *table\_name*  WHERE *condition*; |
| **SELECT DISTINCT**  [1] | Used to retrieve only unique values from a database. It’s particularly beneficial when there’s a need to remove any duplicate entries that may exist in specific columns or tables. | SELECT DISTINCT column1, column2, ...  FROM table\_name; |
| **GROUP BY** [1] | Used to group rows that have the same values in specified columns into aggregated data. It’s often used with aggregate functions like COUNT, MAX, MIN, SUM, AVG to perform calculations on each group of rows. | SELECT *column1, column2, aggregate\_function(column3)*  FROM *table\_name*  GROUP BY *column1*, *column2*; |
| **HAVING** [1] | Used to filter the results of a GROUP BY operation. It’s similar to the WHERE clause, but operates on grouped data produced by a GROUP BY clause. | SELECT *column1, aggregate\_function(column2)*  FROM *table\_name*  GROUP BY *column1*  HAVING *condition;* |
| **ORDER BY** [1] | Ensures that all the values in a column satisfies certain conditions. | SELECT *column1, column2, ..., columnN*  FROM *table\_name*  ORDER BY *column1* [ASC**|**DESC],  *column2* [ASC**|**DESC]*, ..., columnN* [ASC**|**DESC]; |
| **INDEX** [1] | Used to create and retrieve data from the database very quickly. | CREATE INDEX *index\_name*  ON *table\_name* (*column1*, *column2*,...); |
| **TOP** [1] | Utilized to restrict the quantity of records yielded by a query. This clause becomes especially beneficial when working with extensive tables containing thousands of records, where fetching all records could potentially degrade performance. | SELECT TOP *number***|***percent* *column\_name(s)*  FROM *table\_name*  WHERE *condition*; |
| **AND/OR** [1] | Utilized to refine data and obtain accurate results based on several conditions. These operators are employed within the WHERE clause of a SELECT, INSERT, UPDATE, or DELETE statement. | SELECT *column1, column2, ..., columnN*  FROM *table\_name*  WHERE *condition1* {AND**|**OR} *condition2* |
| **IN** [1] | To specify multiple values in a WHERE clause. It serves as a shorthand for multiple OR conditions. | SELECT *column1, column2, ..., columnN*  FROM *table\_name*  WHERE *column\_name* IN *(value\_1,... value\_N);* |
| **BETWEEN** [1] | Utilized to pick out values that fall within a specified range. These values can be numerical, textual, or dates. The BETWEEN operator is inclusive, signifying that the start and end values are encompassed within the range. | SELECT *column1, column2, ..., columnN*  FROM *table\_name*  WHERE *column\_name* BETWEEN *value\_1* AND value\_2; |
| **LIKE** [1] | Used to search for a specific pattern in a column of a table. It can be combined with the WHERE clause to filter the rows that match the pattern. | SELECT *column1, column2, ..., columnN*  FROM *table\_name*  WHERE *column\_name* LIKE { *PATTERN* }; |
| **COUNT** [1] | It is an aggregate function that returns the number of rows that match a specified condition | SELECT COUNT (*column\_name*)  FROM *table\_name*  WHERE *condition*; |
| **MAX** [1] | Used to perform aggregate functions that return the maximum value in a column or a set of values. | SELECT MAX(*column\_name*)  FROM *table\_name*  WHERE *condition*; |
| **MIN** [1] | Used to perform aggregate functions that return the minimum value in a column or a set of values. | SELECT MIN(*column\_name*)  FROM *table\_name*  WHERE *condition*; |
| **SUM** [1] | Used to perform aggregate functions that return the sum of all or distinct values in a column or a set of values. | SELECT SUM(*column\_name*)  FROM *table\_name*  WHERE *condition*; |
| **AVG** [1] | Used to perform aggregate functions that return the average of a column or a set of values. | SELECT AVG(*column\_name*)  FROM *table\_name*  WHERE *condition*; |

## Operators

A variety of commonly used operators are shown in ***TABLE II5.2***.

***Table II5.2 Commonly used Operators***

| **Operator** | **Description** | **Notation** |
| --- | --- | --- |
| **PROJECT (∏)** | The PROJECT operation, symbolized by "pi" (∏), is utilized in relational algebra to extract specific attributes (or columns) from a table. This operation is often referred to as vertical partitioning due to its function of segregating the table vertically. It's noteworthy that the PROJECT operation is a unary operator. | **∏ a(r)**  Node:  Where ∏ is used to represent PROJECTION; r is used to represent RELATION a is the attribute list. |
| **Cross- product (X)** | The Cartesian product, represented by the symbol "X", is a concept in relational algebra. Given two relations, R and S, the Cartesian product operation pairs each tuple (or row) from R with every tuple from S. This operation is classified as a binary operator due to its requirement for two operands. | **R X S**  Node:  R is the first relation  S is the second relation |
| **UNION (∪)** | The Union operation, symbolized by "union" (∪), is a process in relational algebra that mirrors the union operation in set theory. It selects all tuples from both relations, however, it requires that the two relations or tables involved have an identical set of attributes. This operation is considered a binary operator because it necessitates two operands. | **R ∪ S**  Node:  R is the first relation, S is the second relation  Notice: Two relation must have same schema |
| **DIFFERENCE (-)** | The Set Difference operation, represented by a "Hyphen" (-), is a process in relational algebra that finds the difference between two relations, denoted as R-S. It yields all the tuples (or rows) that are present in relation R but not in relation S. This operation is classified as a binary operator because it requires two operands. | **R - S**  Node:  R is the first relation  S is the second relation  Notice: Two relation must have same schema |
| **THETA JOIN (θ)** | The Theta Join operation in relational algebra merges two relations based on a specific condition, symbolized by "theta" (θ). This condition can encompass inequality expressions such as greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). | **R ⋈θ S**  Node:  R is the first relation, S is the second relation  θ is the condition |
| **EQUIVALENT JOIN** | An Equi-Join is a specific type of Theta Join where the condition exclusively involves equality (=) comparisons. Conversely, a Non-Equi-Join is the opposite of an Equi-Join, occurring when the join condition consists of comparisons other than equality (=). | **R ⋈ a=b S**  Node:  R is the first relation  S is the second relation  a=b is the condition |
| **NATURAL JOIN (⋈)** | A Natural Join in relational algebra does not employ a comparison operator and avoids concatenation akin to a Cartesian product. It can only be executed if there is at least one common attribute between the two relations, with the same name and domain. The operation is carried out on matching attributes where the attribute values in both relations are identical, eliminating duplicates. Ideally, a Natural Join is performed on a foreign key. | **R ⋈ S**  Node:  R is the first relation  S is the second relation |
| **INTERSECTION (∩)** | The Intersection operation, denoted by "intersection" (∩), is a process in relational algebra that mirrors the intersection operation in set theory. It selects all tuples that are common to both relations. This operation is classified as a binary operator because it requires two operands. Additionally, it removes any duplicate tuples. | **R ∩ S**  Node:  R is the first relation  S is the second relation |
| **DIVISION (÷)** | The Division operation, denoted by "division" (÷ or /), is utilized in relational algebra for queries that involve terms such as "every", "all", and so on. | **R(X,Y)/S(Y)**  Node:  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. |

# Connecting Java and SQL server database

## Database guide

* **Create a login user:** To create a login user and enable SQL Server Authentication, follow these steps:

1. Open SQL Server and ensure that Authentication is set to Windows Authentication.
2. In the Object Explorer window, navigate to Security and then to Logins. Right-click on 'sa' and select Properties.
3. In the General page, assign a password for the 'sa' user.
4. Navigate to the Status page and select 'Enabled' in the Login section. Click OK to apply the changes.
5. To restart the database server, right-click the server’s name in the Object Explorer window and select Restart.
6. Confirm the action by selecting Yes.

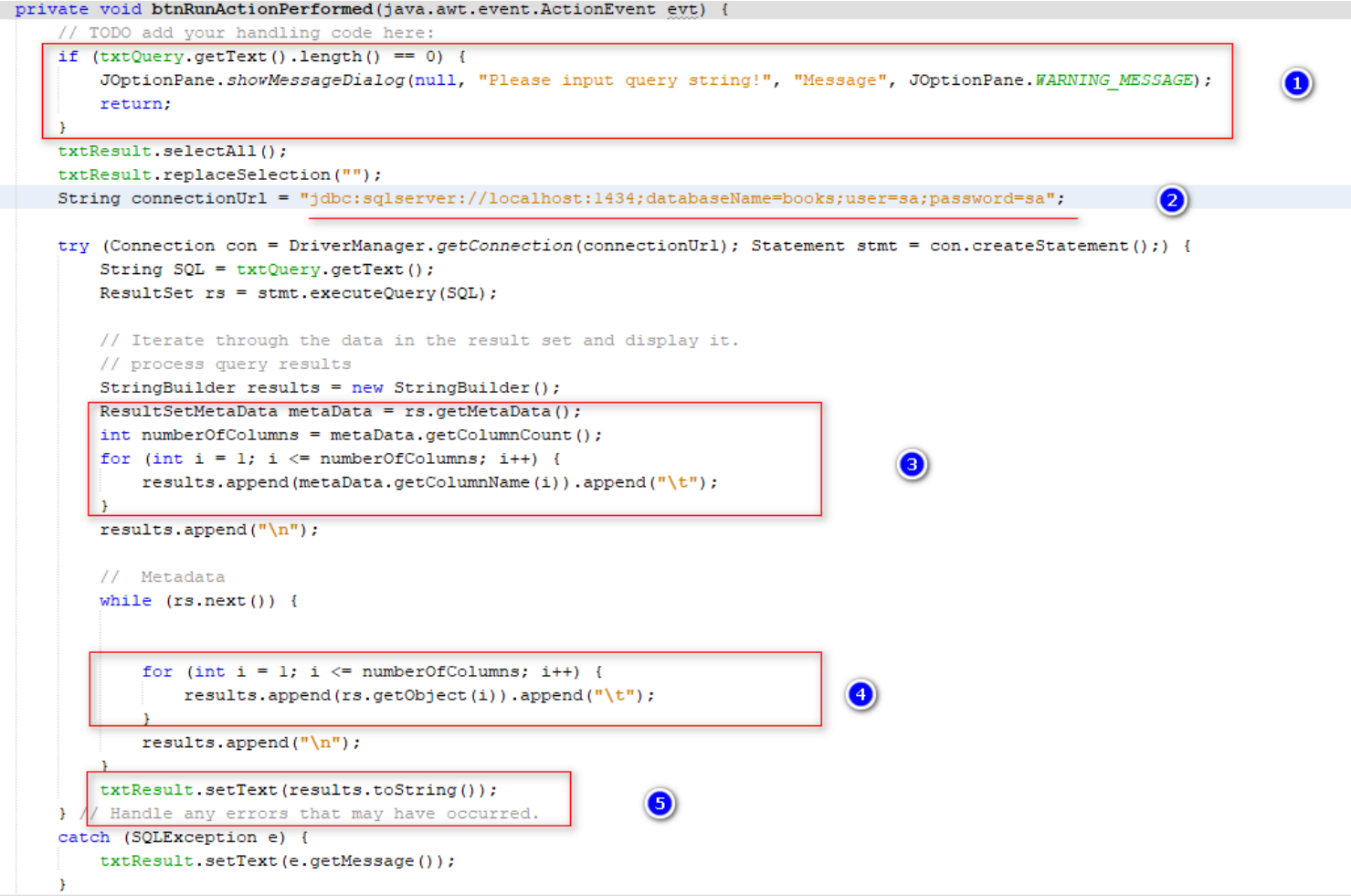
* **Create a database:** To create a database with SQL Server Authentication, follow these steps:

1. Click on “New Query” and write a script to create a database, including the necessary tables and columns.
2. Insert some data into the database and click “Execute” to run the script. A success message will appear in the Messages window.
3. Right-click on “Databases” and select “New Database.”
4. Set the Database name to match the database you created in the script.

## Connect Java to database

* **Create a new project**
  1. Notice that we use Apache NetBeans IDE, to create a new project, follow these steps:
  2. Download and install Apache NetBeans IDE.
  3. Launch NetBeans and initiate a new project by navigating to “File” in the menu, then selecting “New Project.”
  4. Choose “Java” followed by “Java Application,” then click “Next.”
  5. Assign a name to your project and click “Finish.”
* **Create a form**
  1. Right-click on the package which has the same name as the project, select “New”, and then select “JFrame Form”.
  2. Input the name the “JFrame Form”, i.e.: form, then click “Finish’.
* **Adding more libraries**
  1. You need to download the Microsoft JDBC Driver for SQL Server that suits your Java runtime.
  2. Right-click on “Libraries”, and select “Add JAR/Folder”.
  3. Select the file library you have downloaded and click “Open”.
* **User Interface Design**
  1. Right-click in the package, select ”New”, then click the “New JFrame Form” dialog, give your form a name and select any other desired options. Click “Finish”.
  2. The form designer will appear, enabling you to add components to the form via drag-and-drop. These components can be found in the “Palette” window.
  3. To incorporate a component into the form, drag it from the “Palette” window and drop it onto the form. You can adjust the properties of the components in the “Properties” window.
* **Display the form**
  1. After adding all the necessary components, save the form by navigating to “File” -> “Save” in the menu.
  2. Once the form design and functionality are complete, execute the project by selecting “Run” -> “Run Project” from the menu.
* **Connecting to the database**

1. Double-click on the button in the form to create an on-click event for the button.
2. In this dialog, enter the code for the on-click event of the button.
3. Check the following steps as shown in ***Figure II6.1*** :
   1. Check if the user hasn’t input a query, display an error message, and return control to the main form.
   2. Database connection string
   3. Fetch the column information for the table.
   4. Obtain the results of the query.
   5. Display the results in the Text Area.



***Figure II6.1 Steps to check connecting errors***

CHAPTER 3: RELATIVE WORKS

# Analyst requirement of topic

## User (Customers) can:

* Sign up/ Sign in on the application.
* Search for desired products and proceed to the payment page.
* Manage the shopping cart by viewing, adding, or removing items.
* Upon successful payment, the estimated product delivery time should be communicated to the user on the order page.

## User (Sellers) can:

* Sign up/ Sign in on the application.
* Search for desired products and proceed to the payment page.
* Manage the shopping cart by viewing, adding, or removing items.
* Add/ Remove products to sell.
* Process orders.

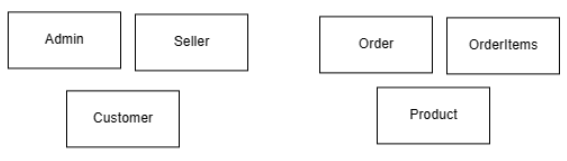
## Admin can:

* Sign up/ Sign in on the application.
* Manage Users including Customers and Sellers.
* Manage orders such as approve orders.

# Design ERD

**Step 1: Entity Identification**

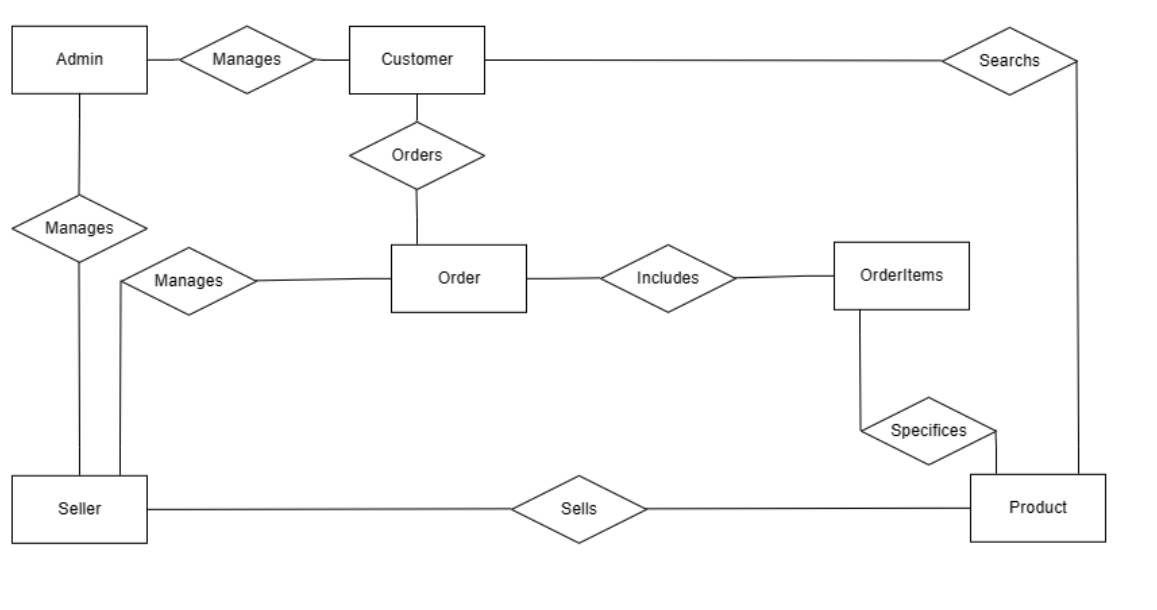
Identify the main entities of the e-commerce platform.



***Figure III2.1 Entities Identification***

**Step 2: Relationship Identification**

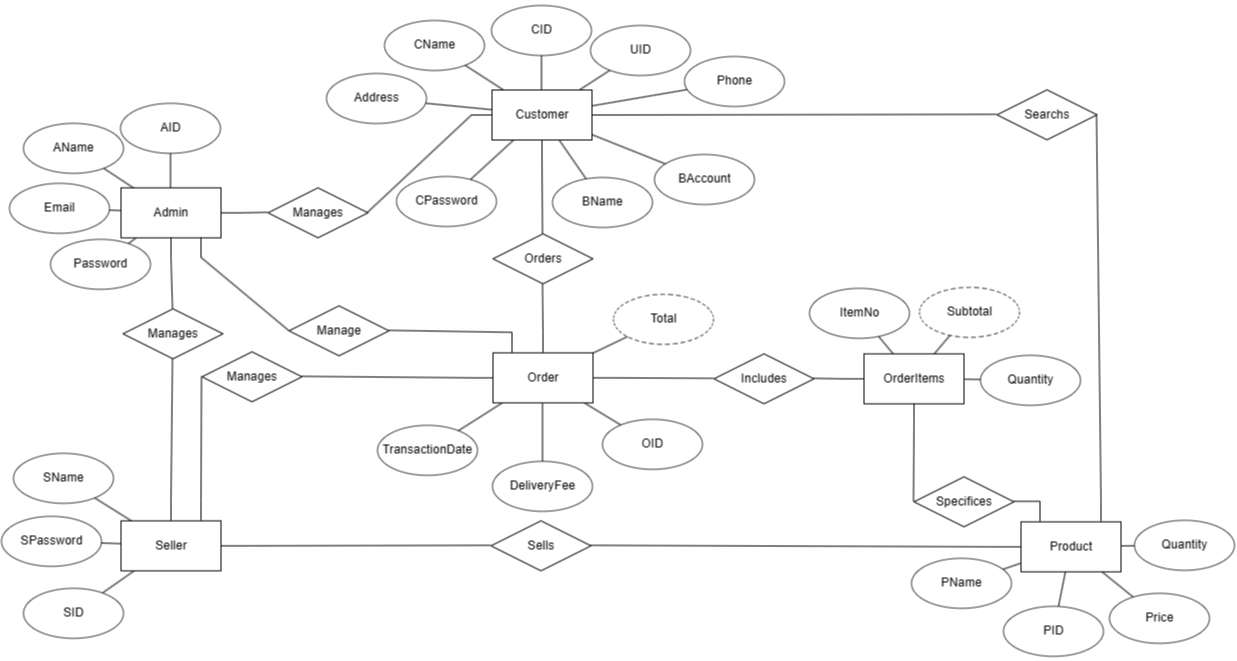
Determine the relationships between the identified entities.



***Figure III2.2 Relationship Identification***

**Step 3: Identifying and mapping attributes to entities, relationships**

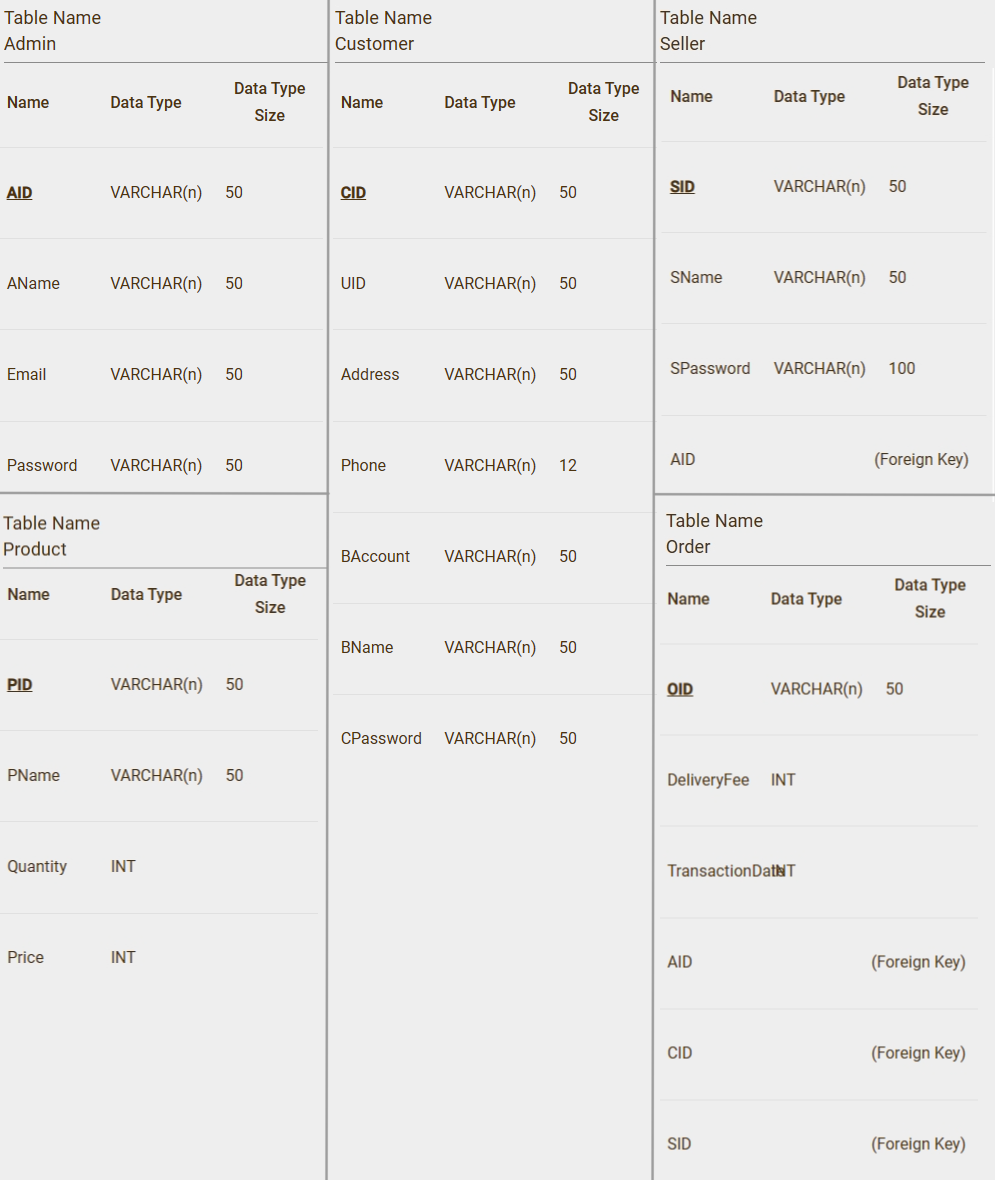
Map the identified attributes to their respective entities or relationships.



***Figure III2.3 Mapping attributes to entities, relationships***

**Step 4: Determine the value range for each attribute**

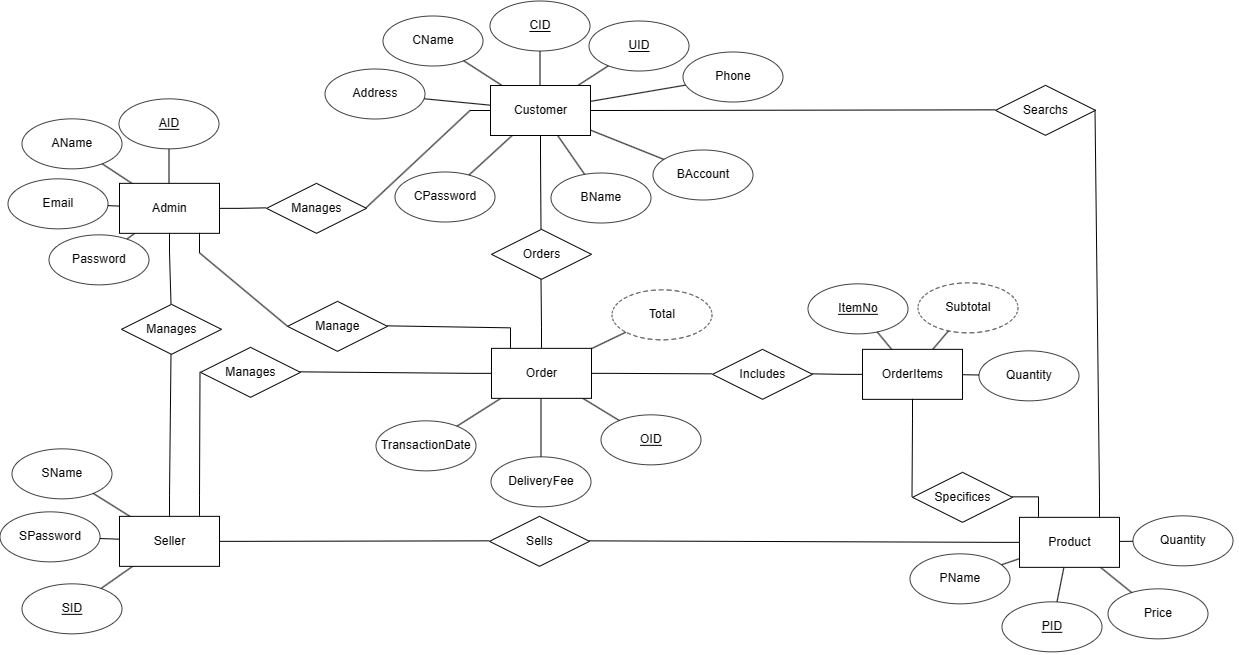
Define the permissible values for each attribute.



***Figure III2.4 Value range for each attribute***

**Step 5: Determine key attribute for each entity**

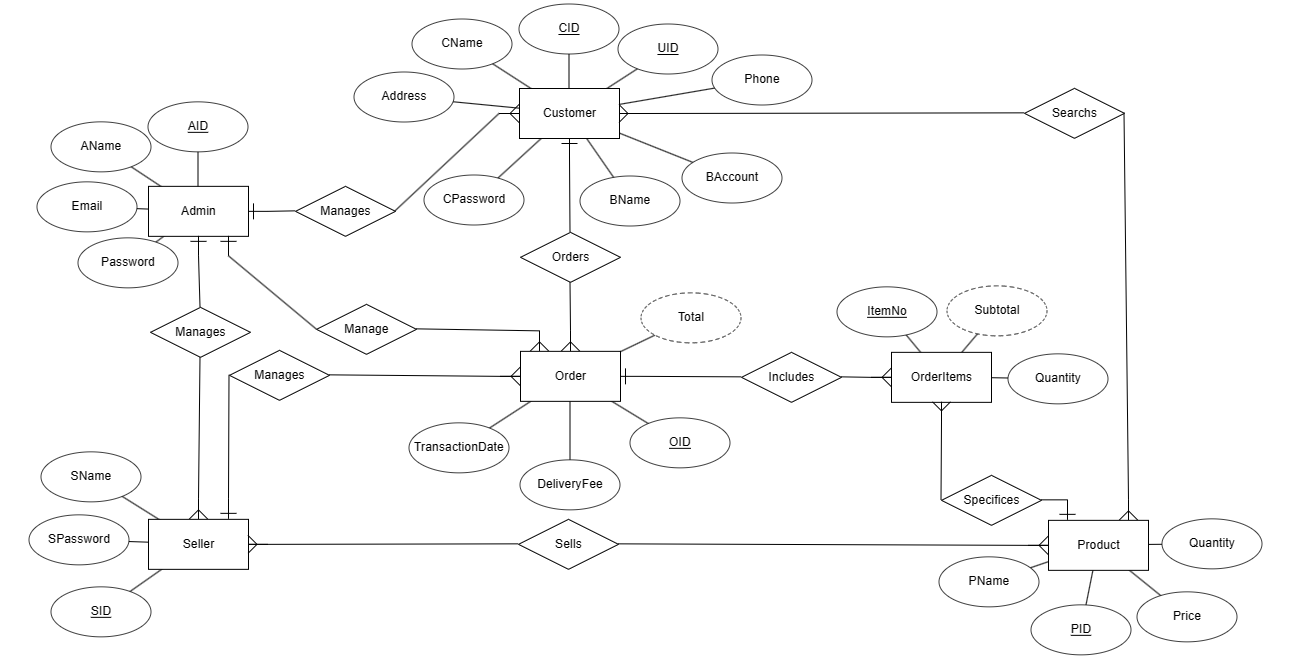
Identify the key attribute for each entity. Each key attribute has an underline below it.



***Figure III2.5 Key attribute for Entities***

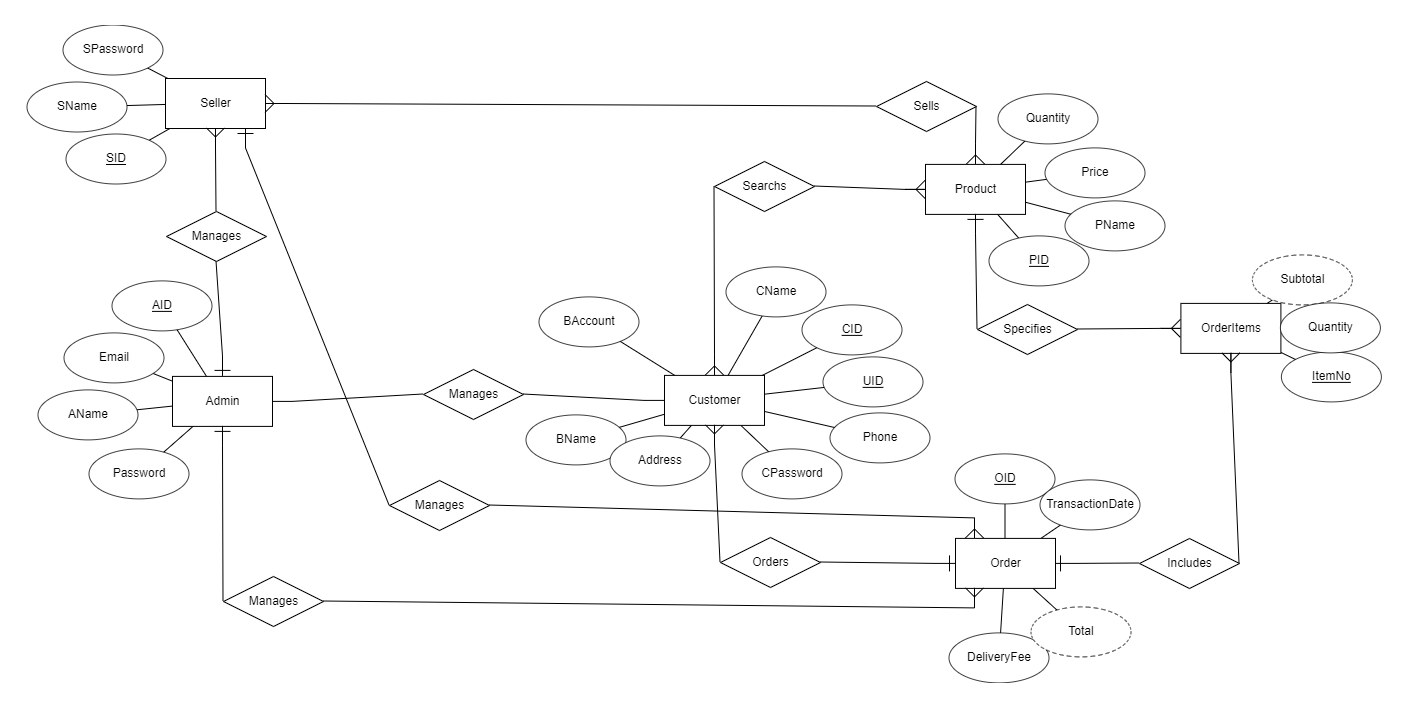
**Step 6: Cardinality identification**

Determine the cardinality of relationships.



***Figure III2.6 Cardinality identification***

**Step 7: Hierarchical design (generalized/specialized) constraints**

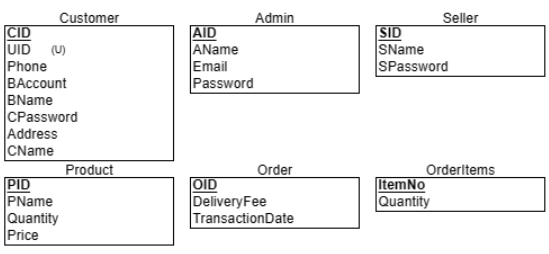
****

***Figure III2.7 ERD Diagram***

# Convert ERD to Relational Model

**Step 1: Entity Types**

Create a matching relation with all the basic attributes, then choose one of the key attributes as the primary keys. The remaining key attributes are secondary unique keys.



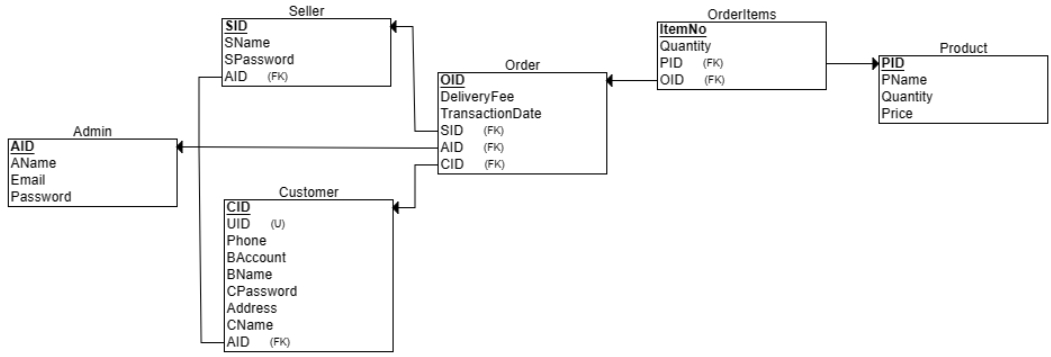
***Figure III3.1 Entity***

**Step 2: Binary 1-to-1**

This step is skipped because there is no 1-to-1 relationship in the ERD model.

**Step 3: Binary 1-to-N**

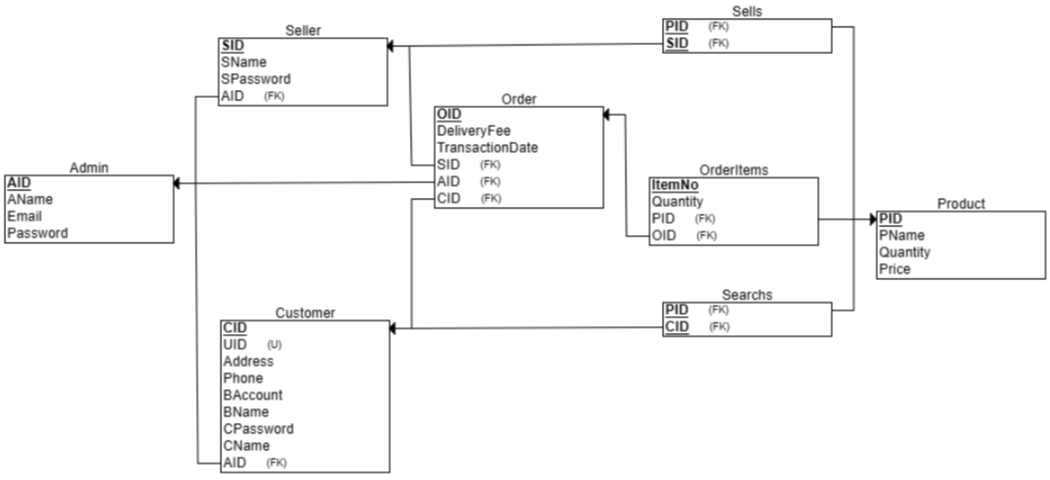
Add as a foreign key in one relation the primary key attributes of the other.



***Figure III3.2 Binary 1-to-N***

**Step 4: Binary M-to-N**

Create a new relation for many-to-many relationships to store the data and add the primary keys of both relations as foreign keys.



***Figure III3.3 Binary M-to-N***

**Step 5: Multivalued Attributes**

Due to the lack of a multivalued type, proceed without executing this step in the ERD model.

# Normalization

The initial model already complies with the First Normal Form (NF1) by enforcing the following actions:

* Each cell may never contain more than one value. For example, the table Admin has AID, AName, Email, Password, and each value has its own column.
* Each row must be unique. The system-generated primary keys (AID, CID, SID, PID, OID, ItemNo) satisfy this requirement by default.
* There are no repeating columns in the same entity that are duplicated. For instance, UID and CID in the Customer entity have different first letters to separate from each other.

The Second Normal Form (2NF) is satisfied by eliminating partial dependencies. OrderItems separate from Order to make its own table, so each column depends on the whole primary key.

Transitioning to the Third Normal Form (3NF) involves further refinement, particularly in addressing transitive dependencies. In the Order table, for instance, the removal of Quantity to a separate OrderItems table ensures that non-key attributes are dependent only on the primary key.

# Create database

| CREATE DATABASE **EcommerceData**  USE [**EcommerceData**] |
| --- |

An overview of all database entities and their attributes

| **Table** | **Attributes** |
| --- | --- |
| CREATE TABLE [dbo].[**Admin**] | [**AID**] [nvarchar](50) NOT NULL,  [**AName**] [nvarchar](50) NOT NULL,  [**Email**] [nvarchar](100) NOT NULL,  [**Password**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**AID**]) |
| CREATE TABLE [dbo].[**Customer**] | [CID] [nvarchar](50) NOT NULL,  [UID] [nvarchar](50) NOT NULL,  [**Address**] [nvarchar](50) NOT NULL,  [**Phone**] [nvarchar](12) NOT NULL,  [**BAccount**] [nvarchar](50) NOT NULL,  [**BName**] [nvarchar](50) NOT NULL,  [**CPassword**] [nvarchar](50) NOT NULL,  [**CName**] [nvarchar](50) NOT NULL,  [**AID**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**CID**]),  FOREIGN KEY ([**AID**]) REFERENCES [Admin]([**AID**]),  UNIQUE ([**UID**]) |
| CREATE TABLE [dbo].[**Seller**] | [**SID**] [nvarchar](50) NOT NULL,  [**SName**] [nvarchar](50) NOT NULL,  [**SPassword**] [nvarchar](100) NOT NULL,  [**AID**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**SID**]),  FOREIGN KEY ([**AID**]) REFERENCES [Admin]([**AID**]) |
| CREATE TABLE [dbo].[**Product**] | [**PID**] [nvarchar](50) NOT NULL,  [**PName**] [nvarchar](50) NOT NULL,  [**Quantity**] [int] NOT NULL,  [**Price**] [int] NOT NULL,  PRIMARY KEY ([**PID**]) |
| CREATE TABLE [dbo].[**Sells**] | [**PID**] [nvarchar](50) NOT NULL,  [**SID**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**PID**], [**SID**]),  FOREIGN KEY ([**PID**]) REFERENCES [Product]([**PID**]),  FOREIGN KEY ([**SID**]) REFERENCES [Seller]([**SID**]) |
| CREATE TABLE [dbo].[**Order**] | [**OID**] [nvarchar](50) NOT NULL,  [**DeliveryFee**] [int] NOT NULL,  [**TransactionDate**] [int] NOT NULL,  [**AID**] [nvarchar](50) NOT NULL,  [**CID**] [nvarchar](50) NOT NULL,  [**SID**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**OID**]),  FOREIGN KEY ([**CID**]) REFERENCES [Customer]([**CID**]),  FOREIGN KEY ([**SID**]) REFERENCES [Seller]([**SID**]),  FOREIGN KEY ([**AID**]) REFERENCES [Admin]([**AID**]) |
| CREATE TABLE [dbo].[**OrderItems**] | [**ItemNo**] [int] NOT NULL,  [**Quantity**] [int] NOT NULL,  [**PID**] [nvarchar](50) NOT NULL,  [**OID**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**ItemNo**]),  FOREIGN KEY ([**PID**]) REFERENCES [**Product**]([**PID**]),  FOREIGN KEY ([**OID**]) REFERENCES [Order]([**OID**]) |
| CREATE TABLE [dbo].[**Searchs**] | [**CID**] [nvarchar](50) NOT NULL,  [**PID**] [nvarchar](50) NOT NULL,  PRIMARY KEY ([**CID**], [**PID**]),  FOREIGN KEY ([**CID**]) REFERENCES [**Customer**]([**CID**]),  FOREIGN KEY ([**PID**]) REFERENCES [Product]([**PID**]) |

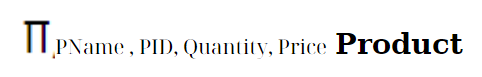
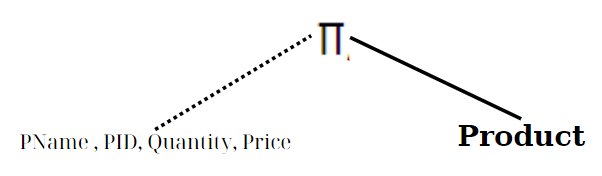
# Collect & insert data

In general, the data insertion comes from a self-generated source for testing purposes, as in **Admin**, **Customer**, **Seller**, **Product**, **Sells**, **Order**, **OrderItems**, and **Searchs**.

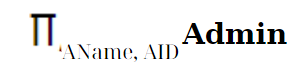
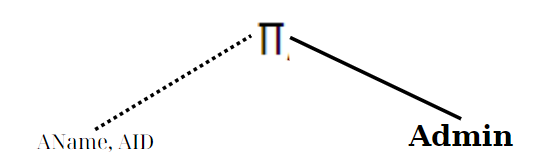
| **Table** | **Data insertion** |
| --- | --- |
| **Admin** | INSERT INTO [dbo].[**Admin**] ([**AID**], [**AName**], [**Email**], [**Password**]) VALUES  ('A01', 'PDM Project', 'PDMProject@email.com','login1'); |
| **Customer** | INSERT INTO [dbo].[**Customer**] ([**CName**], [**CID**], [**UID**], [**Address**], [**Phone**], [**BAccount**], [**BName**], [**CPassword**], [**AID**]) VALUES  ('John Doe', 'C01', 'U01', '123 Main St, Anytown, USA', '1234567890', '11110000', 'Bank A', 'pass1', 'A01'),  ('Jane Smith', 'C02', 'U02', '456 Oak Ave, Smallville, USA', '2345678901', '22220000', 'Bank B', 'pass2', 'A01'),  ('Jim Brown', 'C03', 'U03', '789 Pine Rd, Lakeview, USA', '3456789012', '33330000', 'Bank C', 'pass3', 'A01'),  ('Jill Johnson', 'C04', 'U04', '321 Elm St, Metropolis, USA', '4567890123', '44440000', 'Bank D', 'pass4', 'A01'),  ('Jack Davis', 'C05', 'U05', '654 Maple Ave, Pleasantville, USA', '5678901234', '55550000', 'Bank E', 'pass5', 'A01'),  ('Alice Williams', 'C06', 'U06', '987 Cedar Rd, Riverview, USA', '6789012345', '66660000', 'Bank F', 'pass6', 'A01'),  ('Bob Martin', 'C07', 'U07', '147 Willow St, Hilltop, USA', '7890123456', '77770000', 'Bank G', 'pass7', 'A01'),  ('Charlie Thompson', 'C08', 'U08', '258 Poplar Ave, Seaside, USA', '8901234567', '88880000', 'Bank H', 'pass8', 'A01'),  ('Diana Garcia', 'C09', 'U09', '369 Birch Rd, Forestview, USA', '9012345678', '99990000', 'Bank I', 'pass9', 'A01'),  ('Ethan Martinez', 'C10', 'U10', '963 Spruce St, Valleyview, USA', '1234567890', '00001111', 'Bank J', 'pass10', 'A01'); |
| **Seller** | INSERT INTO [dbo].[**Seller**] ([**SID**], [**SName**], [**SPassword**], [**AID**]) VALUES  ('S01', 'Oliver Wilson', 'password1', 'A01'),  ('S02', 'Sophia Taylor', 'password2', 'A01'),  ('S03', 'Mason Moore', 'password3', 'A01'),  ('S04', 'Ava Anderson', 'password4', 'A01'),  ('S05', 'Liam Thomas', 'password5', 'A01'),  ('S06', 'Mia Jackson', 'password6', 'A01'),  ('S07', 'Lucas Martin', 'password7', 'A01'),  ('S08', 'Emma Thompson', 'password8', 'A01'),  ('S09', 'Noah Garcia', 'password9', 'A01'),  ('S10', 'Isabella Martinez', 'password10', 'A01'); |
| **Product** | INSERT INTO [dbo].[**Product**] ([**PID**], [**PName**], [**Quantity**], [**Price**]) VALUES  ('P100001', 'Apple', 50, 15),  ('P100002', 'Banana', 100, 30),  ('P100003', 'Cherry', 200, 45),  ('P100004', 'Date', 150, 25),  ('P100005', 'Elderberry', 75, 40),  ('P100006', 'Fig', 60, 35),  ('P100007', 'Grape', 120, 20),  ('P100008', 'Honeydew', 80, 50),  ('P100009', 'Iceberg Lettuce', 90, 10),  ('P100010', 'Jackfruit', 70, 45),  ('P100011', 'Kiwi', 110, 30),  ('P100012', 'Lemon', 130, 40),  ('P100013', 'Mango', 140, 15),  ('P100014', 'Nectarine', 150, 35),  ('P100015', 'Orange', 160, 20); |
| **Sells** | INSERT INTO [dbo].[**Order**] ([**OID**], [**DeliveryFee**], [**TransactionDate**], [**AID**], [**CID**], [**SID**]) VALUES  ('O10001', 10, 20231130, 'A01', 'C02', 'S07'),  ('O10002', 15, 20231130, 'A01', 'C03', 'S06'),  ('O10003', 20, 20231130, 'A01', 'C04', 'S06'),  ('O10004', 25, 20231130, 'A01', 'C05', 'S01'),  ('O10005', 30, 20231130, 'A01', 'C06', 'S03'),  ('O10006', 35, 20231130, 'A01', 'C07', 'S02'),  ('O10007', 40, 20231130, 'A01', 'C08', 'S09'),  ('O10008', 45, 20231130, 'A01', 'C02', 'S08'),  ('O10009', 50, 20231130, 'A01', 'C03', 'S07'),  ('O10010', 55, 20231130, 'A01', 'C04', 'S06'),  ('O10011', 60, 20231130, 'A01', 'C05', 'S01'),  ('O10012', 65, 20231130, 'A01', 'C06', 'S05'),  ('O10013', 70, 20231130, 'A01', 'C09', 'S04'),  ('O10014', 75, 20231130, 'A01', 'C10', 'S03'),  ('O10015', 80, 20231130, 'A01', 'C02', 'S02'),  ('O10016', 85, 20231130, 'A01', 'C03', 'S10'),  ('O10017', 90, 20231130, 'A01', 'C04', 'S09'),  ('O10018', 95, 20231130, 'A01', 'C07', 'S08'),  ('O10019', 100, 20231130, 'A01', 'C08', 'S07'),  ('O10020', 105, 20231130, 'A01', 'C03', 'S06'),  ('O10021', 110, 20231130, 'A01', 'C02', 'S05'),  ('O10022', 115, 20231130, 'A01', 'C01', 'S04'),  ('O10023', 120, 20231130, 'A01', 'C09', 'S03'),  ('O10024', 125, 20231130, 'A01', 'C10', 'S02'),  ('O10025', 130, 20231130, 'A01', 'C02', 'S01'); |
| **Order** | INSERT INTO [dbo].[**OrderItems**] ([**ItemNo**], [**Quantity**], [**PID**], [**OID**]) VALUES  (1001, 150, 3, 'P100001', 'O10001'),  (1002, 300, 6, 'P100002', 'O10002'),  (1003, 450, 9, 'P100003', 'O10003'),  (1004, 600, 12, 'P100004', 'O10004'),  (1005, 750, 15, 'P100005', 'O10005'),  (1006, 900, 18, 'P100006', 'O10006'),  (1007, 1050, 21, 'P100007', 'O10007'),  (1008, 1200, 24, 'P100008', 'O10008'),  (1009, 1350, 27, 'P100009', 'O10009'),  (1010, 1500, 30, 'P100010', 'O10010'),  (1011, 150, 3, 'P100011', 'O10011'),  (1012, 300, 6, 'P100012', 'O10012'),  (1013, 450, 9, 'P100013', 'O10013'),  (1014, 600, 12, 'P100014', 'O10014'),  (1015, 750, 15, 'P100015', 'O10015'),  (1016, 900, 18, 'P100006', 'O10016'),  (1017, 1050, 21, 'P100007', 'O10017'),  (1018, 1200, 24, 'P100008', 'O10018'),  (1019, 1350, 27, 'P100009', 'O10019'),  (1020, 1500, 30, 'P100010', 'O10020'),  (1021, 900, 18, 'P100006', 'O10021'),  (1022, 1050, 21, 'P100007', 'O10022'),  (1023, 1200, 24, 'P100008', 'O10023'),  (1024, 1350, 27, 'P100009', 'O10024'),  (1025, 1500, 30, 'P100010', 'O10025'); |
| **OrderItems** | INSERT INTO [dbo].[**Searchs**] ([**CID**], [**PID**]) VALUES  ('C01', 'P100001'),  ('C02', 'P100002'),  ('C03', 'P100003'),  ('C04', 'P100004'),  ('C05', 'P100005'),  ('C06', 'P100006'),  ('C07', 'P100007'),  ('C08', 'P100008'),  ('C09', 'P100009'),  ('C10', 'P100010'),  ('C10', 'P100001'),  ('C08', 'P100002'),  ('C06', 'P100003'),  ('C04', 'P100004'),  ('C02', 'P100005'),  ('C01', 'P100006'),  ('C03', 'P100007'),  ('C07', 'P100008'),  ('C05', 'P100009'),  ('C09', 'P100010'); |
| **Searchs** | INSERT INTO [dbo].[**Sells**] ([**SID**], [**PID**]) VALUES  ('S01', 'P100001'),  ('S02', 'P100002'),  ('S03', 'P100003'),  ('S04', 'P100004'),  ('S05', 'P100005'),  ('S06', 'P100006'),  ('S07', 'P100007'),  ('S08', 'P100008'),  ('S09', 'P100009'),  ('S10', 'P100010'),  ('S02', 'P100001'),  ('S04', 'P100002'),  ('S06', 'P100003'),  ('S08', 'P100004'),  ('S10', 'P100005'),  ('S01', 'P100006'),  ('S03', 'P100007'),  ('S07', 'P100008'),  ('S07', 'P100009'),  ('S09', 'P100010'); |

# Querying database

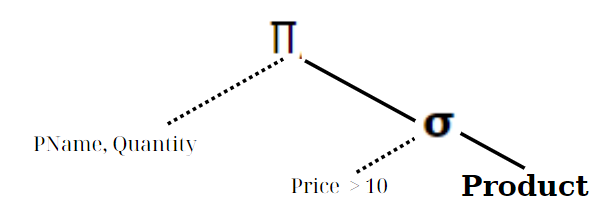
1. **Find all attribute in product**

* SQL: SELECT \* FROM Product
* RA: 
* Tree expression:

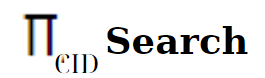
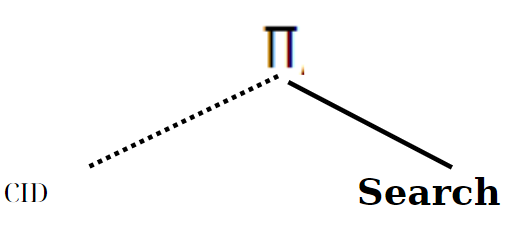
1. **Get all column Admin's name and Id**

* SQL: SELECT AName, AID FROM Admin;
* RA:
* Tree Expression:

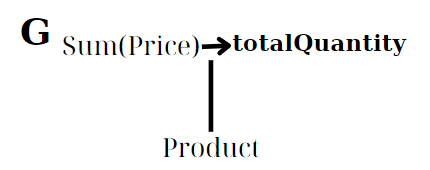
1. **Get product's name and its quantity which price more than 10 dolar**

* SQL: SELECT PName, Quantity FROM Product WHERE Price > 10
* RA: 
* Tree expression

1. **Get customer Id of customer already ordered**

* SQL: SELECT DISTINCT Cid FROM Search
* RA:
* Tree expression:

1. **Get the total quantity of all the product is available**

* SQL: SELECT Sum(Quantity) as totalQuantity FROM Product
* RA : 
* Tree expression:

1. **Find Name and Price of product which price is more than 20 all customer have search and its Customer Name**

* SQL:

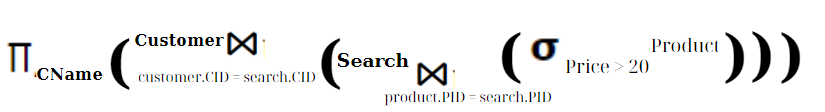
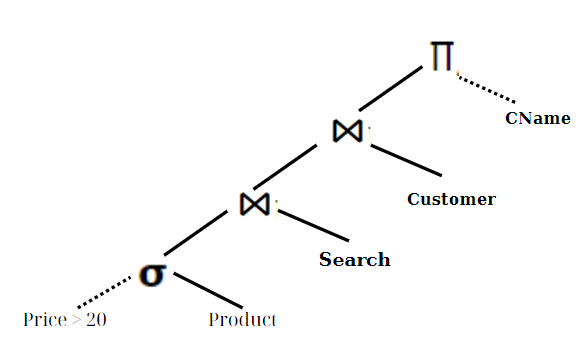
SELECT c.CName

FROM product p

INNER JOIN Searchs s ON p.PID = s.PID

INNER JOIN Customer c ON c.CID = s.CID

WHERE p.Price > 20;

* RA:
* Tree expression

1. **Find customer name, Phone, Bank Account and Bank Name of the customer who have j in their name and order by the descending of Bank Name**

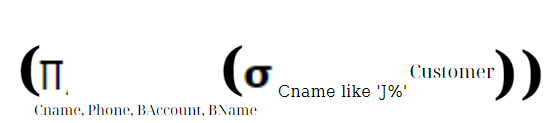
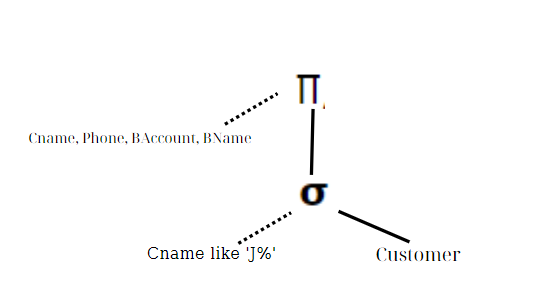
* SQL:

SELECT Cname, Phone, BAccount, BName

FROM customer

WHERE Cname like 'J%'

ORDER BY Bname DESC;

* RA:
* Tree expression

1. **Query the maximum and minimum of a product whose price more than 20**

* SQL:

SELECT max(quantity)

FROM product

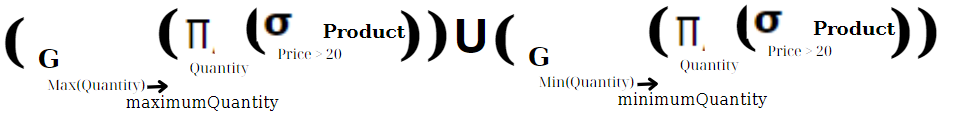
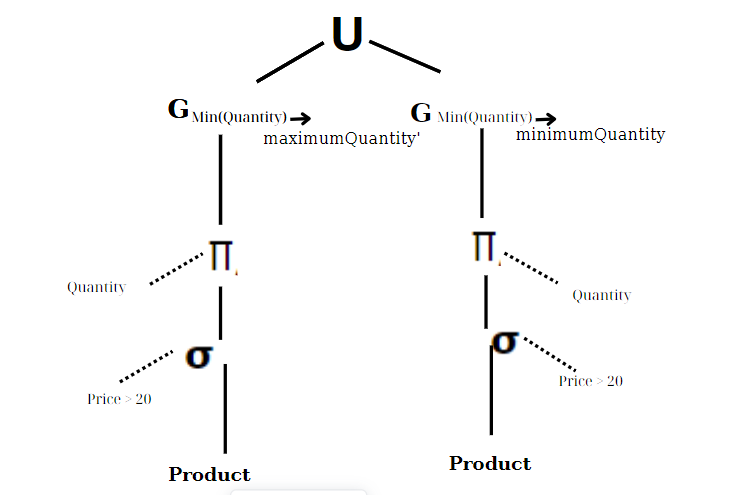
WHERE price > 20

UNION

SELECT min(quantity)

FROM product

WHERE price > 20;

* RA:
* Tree expression
* 

# Create view

1. **Create view of product name and quantity**

CREATE VIEW productView AS

SELECT PName, quantity FROM product ;

1. **Create view of customer whose name start with ‘J’**

CREATE VIEW customerView AS

SELECT \*

FROM Customer

WHERE CName like 'J%';

1. **Create view of the name and id of the customer who have searched about the product whose id is ‘C01’**

CREATE VIEW customerSearchView AS

SELECT c.CId, c.Cname

FROM Customer c, Searchs s

WHERE s.PID = 'C01';

1. **Create view of the total number of customer**

CREATE VIEW customerTotalView AS

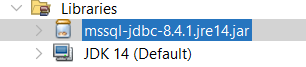
SELECT Count(cid)

FROM customer

# Connect Sql to server with Java

1. Create login server
2. Download and instal IDE for Java Development (In our project, we use Apache Netbean IDE)
3. Create Java project for Querying Database
4. Create Java Form for querying
5. Add more library

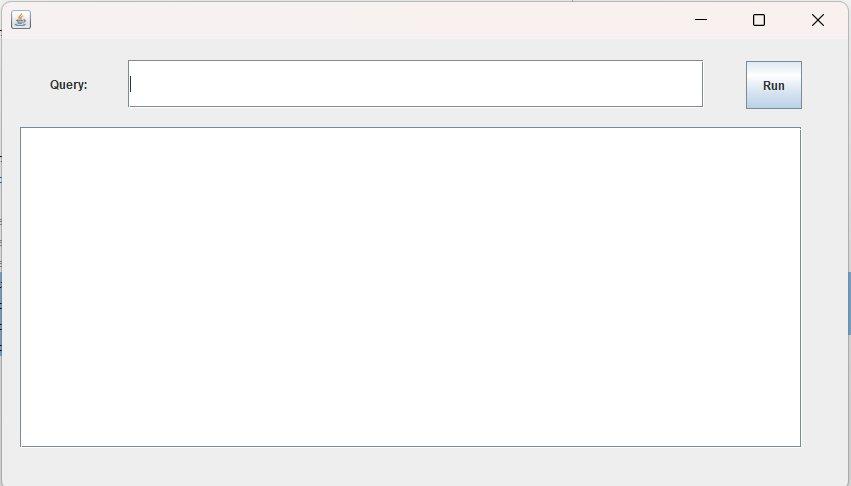
* Download the library **mssql-jdbs-8.4.1.jre14.jar** and select add library into Jar/Folder



#### *Figure III9.1. Library mssql-jdbc*

1. Design the user interface and display the form

* Label
* Text Field
* Button
* Text Area

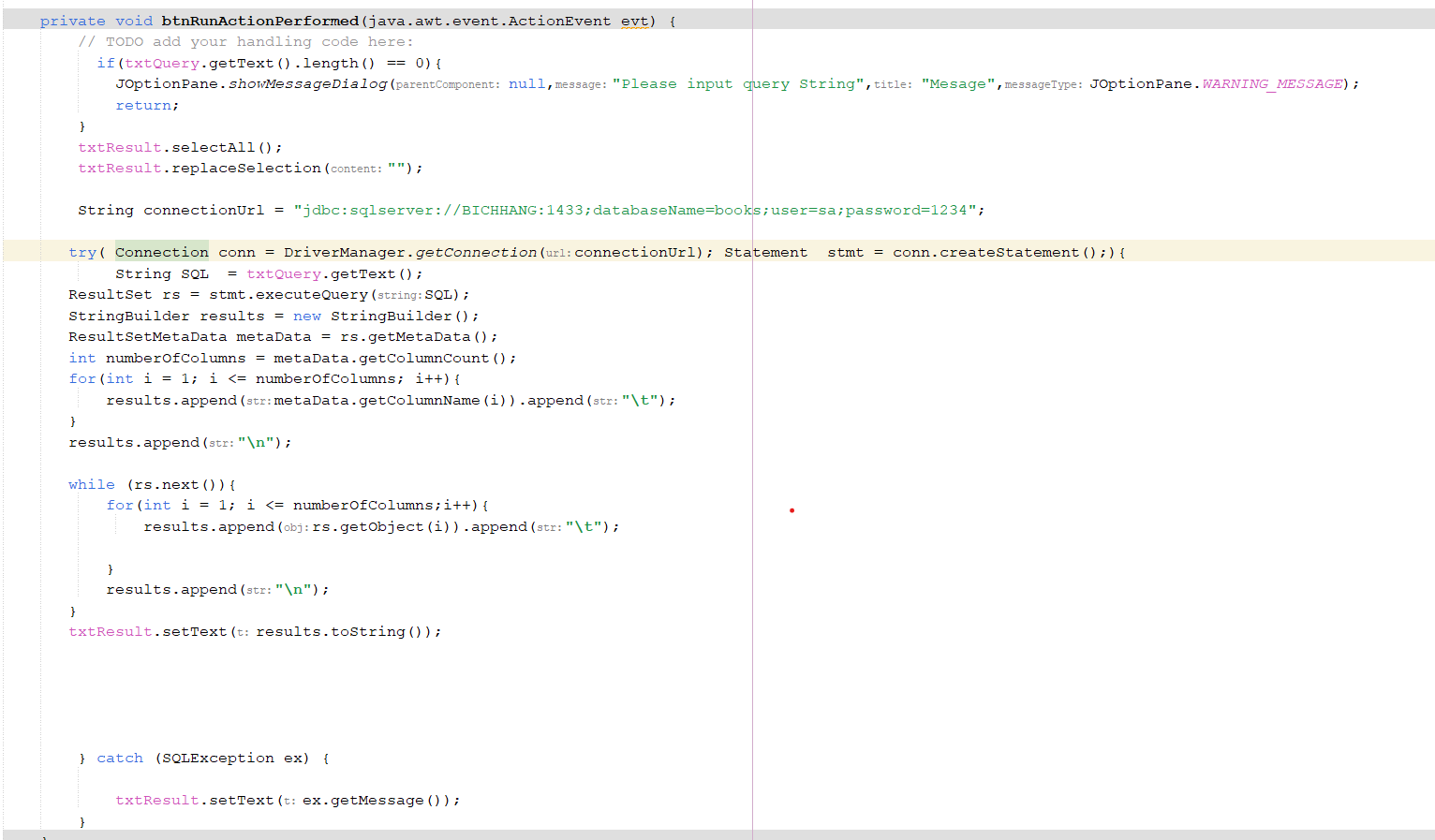
****

#### *Figure III.9.2. Display the form*

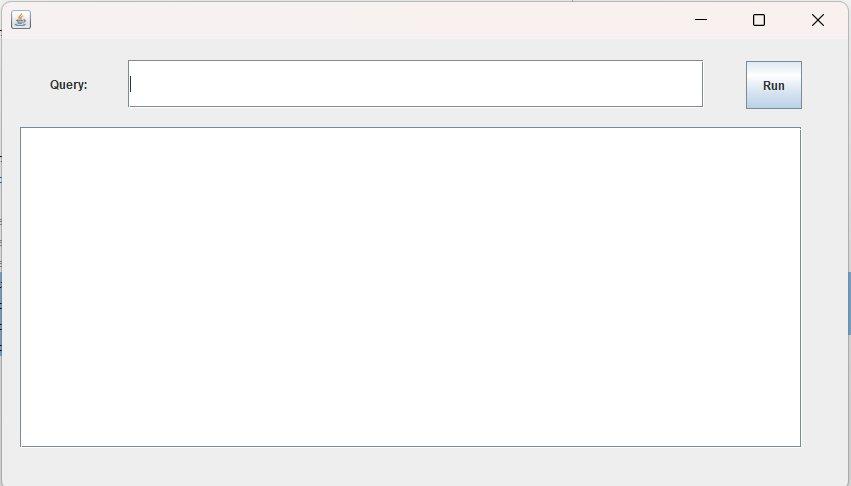
1. Connecting to database

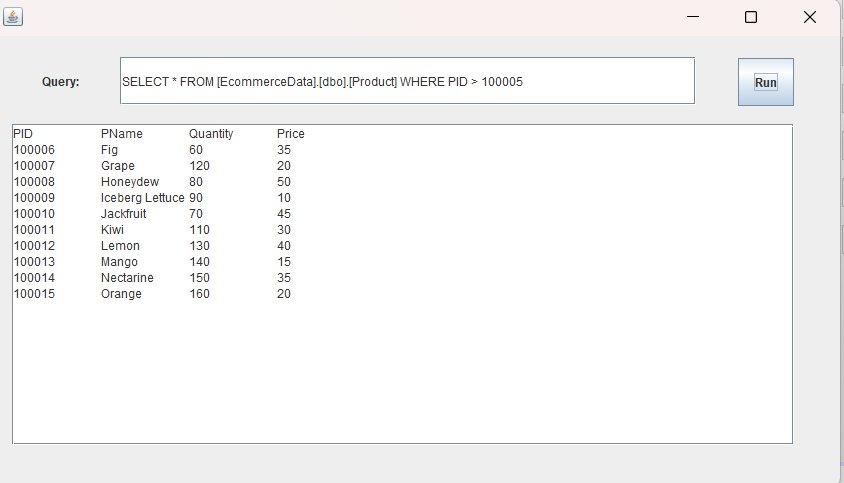
* Enter the code for the on click event of the button

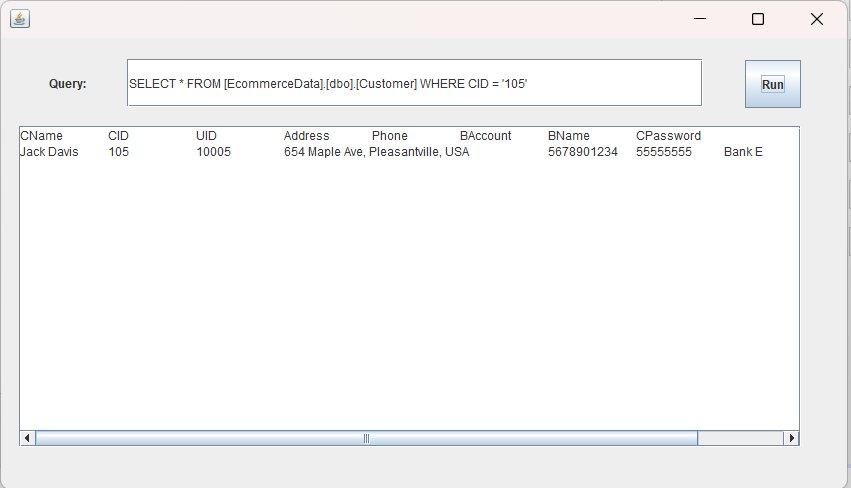
1. If the user has not entered a query, an error message will be displayed.
2. Database connection string
3. Retrieve the table's column information
4. Display the result after querying on the text Area

***Figure III9.3 Code for connecting to Java***

# Demo

****





CHAPTER 4: CONCLUSION

# Summary

Here are the key accomplishments that our team developed and carried out this application project using structural-driven analytical methods:

* Theoretical:

1. Defining the project’s goals.
2. Constructing the Entity Relationship Diagram (ERD) and the relational model.
3. Analyzing the problem and implementing the methodologies learned from the course.
4. Utilizing Query for database creation.
5. Designing interfaces for managing data storage and updates.

* Promaming:

1. Utilization of SQL Server for database creation and management.
2. Application of Java within the Apache NetBeans IDE for interface development.
3. Establishment of connections from SQL Server databases to Java form interfaces.

# What we learned

Through the execution of this project, our team has been able to apply the theoretical principles learned from the "Principle of Database Management" course. The primary lessons learned from this project:

* Applied the theoretical principles learned from the course to the project:

1. Understanding the relational model and SQL Algebra.
2. Establishing an ER database.
3. Executing queries in SQL Server.
4. Inputting data into a database.
5. Linking Java Display form with SQL Server.

* Group’s Collaborative:

1. Ensuring clear and effective communication within the group.
2. Distributing tasks evenly among members to enhance efficiency.
3. Learning from diverse opinions within the team and resolving disagreements constructively, leading to improved solutions and harmonious teamwork.
4. Recognizing the importance of planning, task prioritization, and efficient work practices for effective time management.
5. Acquiring the skill to write a report right structure and formal

# Completed works & future works

* Our team has created an E-commercial database and Java forms including sign-in/sign-out, selling products for sellers, searching to buy products for both sellers and customers, and managing sellers and customers for admins.
* The analysis and design of the Demo have been largely completed, despite time limitations and a lack of hands-on experience.
* The application will be updated and improved in the future.

CHAPTER 5: REFERENCES

| [1] | Ramesh Venkataraman, Heikki Topi, Jeff Hoffer, Modern Database Management 12th edition, published by Pearson Education, 2016. |
| --- | --- |
| [2] | S. N. Ramez Elmasri, Fundamentals of Database Systems 7th Edition, published by Pearson, 2015. |
| [3] | Jeffrey D. Ullman, J. Widom, Hector Garcia-Molina, Database Systems: The Complete Book 2nd Edition, published by Pearson, 2008. |