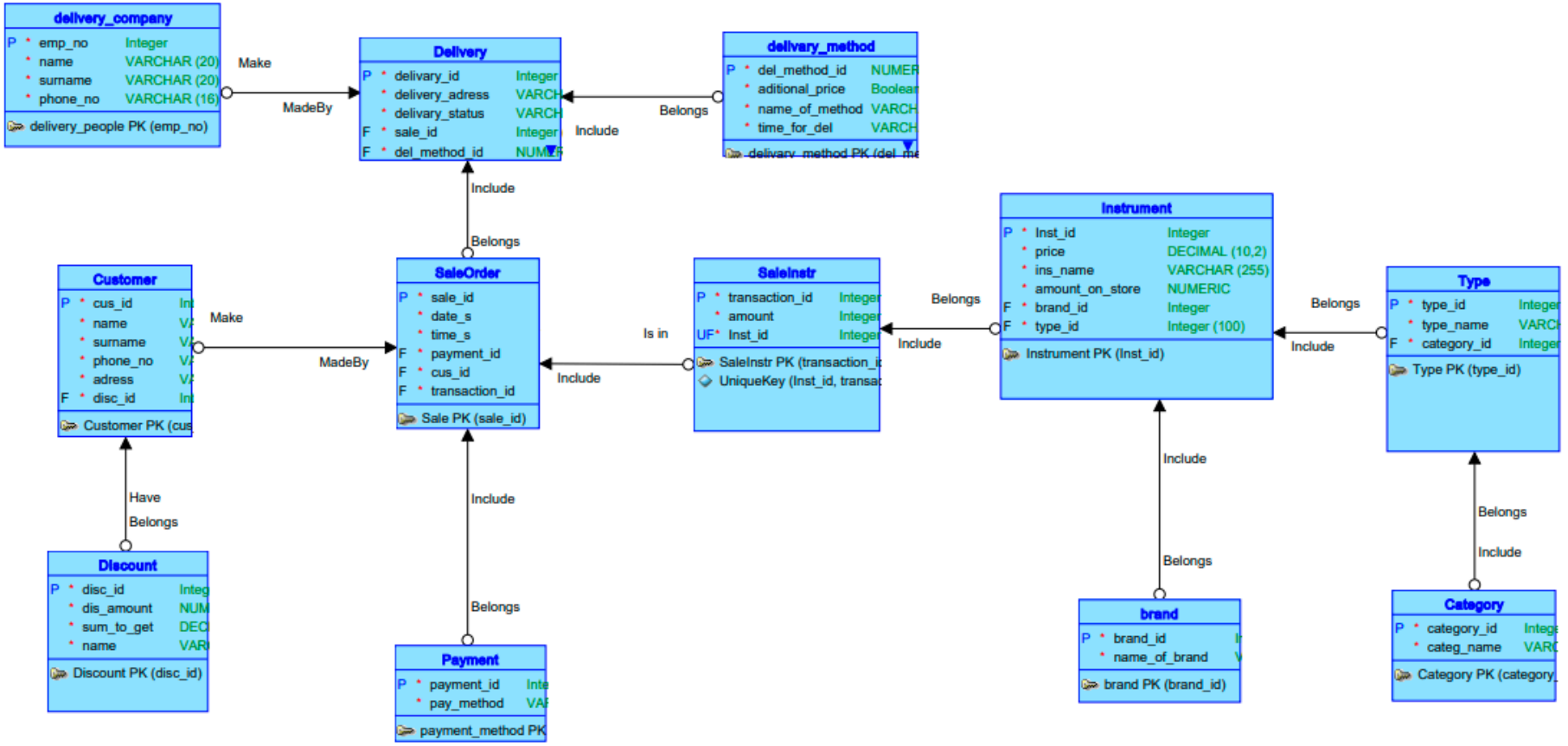
DataBase Project

“Music shop”

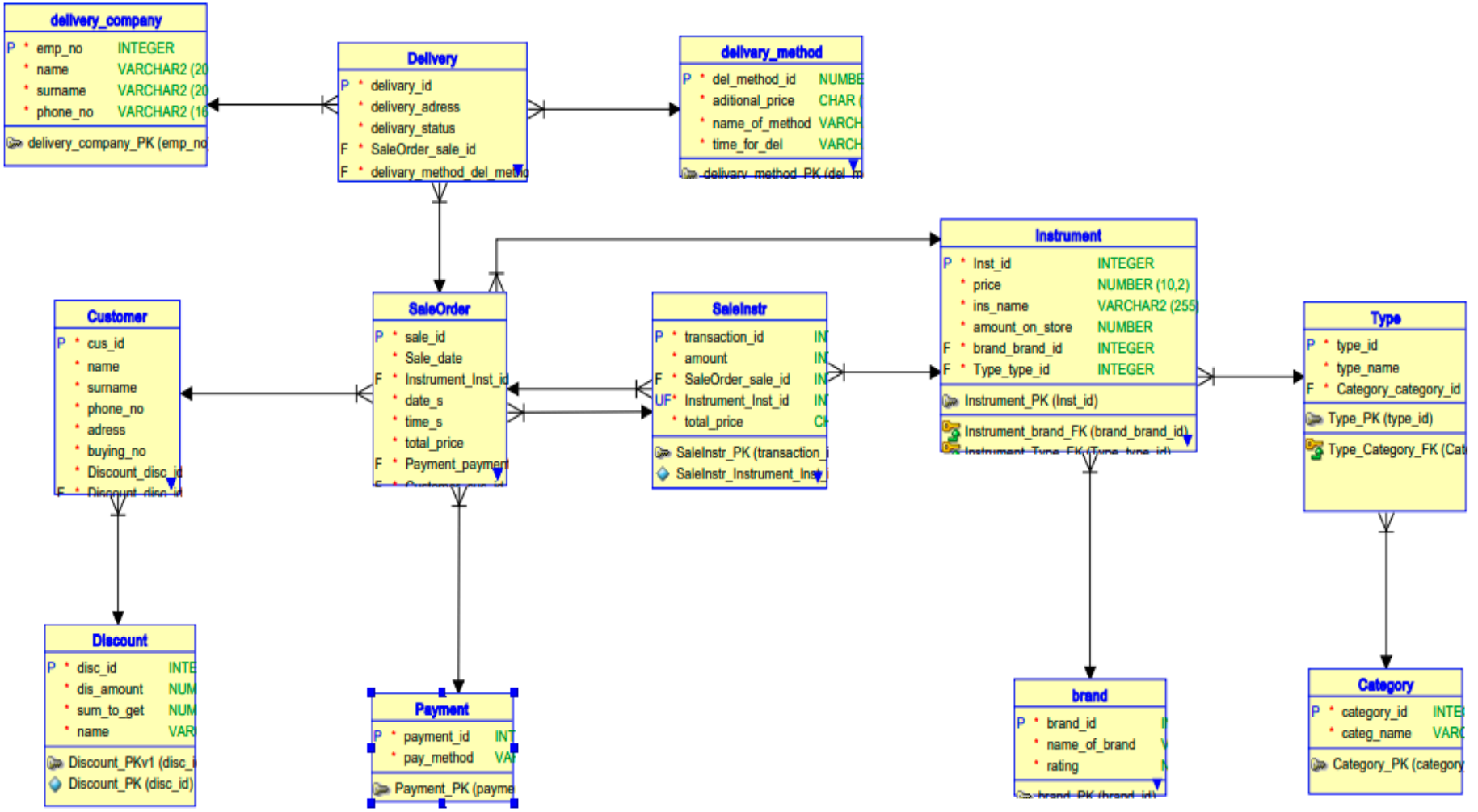
Made by student
2IiE-SP
Volodymyr Chuchka

Part 1 : Logical and Relational models

Logical model:



Relational model :



Logical Model:

The logical model of our music shop database includes several key entities and their relationships, ensuring the representation of the shop's operations. It encompasses entities such as Customer, Discount, SaleOrder, Payment, Instrument, Brand, Type, Category, Delivery, DeliveryMethod, and DeliveryCompany. The model defines how these entities interact with each other, capturing essential details like customer information, sales orders, payments, instruments, brands, types, categories, and delivery details. This model ensures data integrity and efficient data management through the use of primary and foreign keys, along with defined relationships.

Relational Model:

The relational model translates the logical model into a structured format that can be implemented in a relational database system. It includes detailed tables for each entity with defined attributes and relationships. The tables include primary keys to uniquely identify records and foreign keys to establish relationships between tables. The model ensures normalization to minimize redundancy and maintain data integrity. This structured format supports the operations of the music shop by allowing efficient data retrieval, insertion, updating, and deletion.

Key Components:

- *Entities:*

    - *Customer, Discount, SaleOrder, Payment, Instrument, Brand, Type, Category, Delivery, DeliveryMethod, DeliveryCompany*
- ***Relationships:***
    - *One-to-Many and Many-to-One relationships ensure data integrity and efficient management.*
- ***Primary and Foreign Keys****:*
    - *Used to uniquely identify records and establish relationships.*
- ***Normalization:***
    - *Ensures minimal redundancy and efficient data management.*

This comprehensive model supports the music shop's operations, ensuring robust data management and integrity.

Explanation of the Model:

Our database model for the music shop application is designed to comprehensively cover all aspects of the shop's operations, from customer interactions to sales and deliveries.

Logical Model: The logical model serves as an abstract representation of the data and its relationships. It includes entities such as:

- *Customer: Stores customer details like name, phone number, address, and any associated discounts.*
- *Discount: Details the discounts available, including the discount amount and conditions.*
- *SaleOrder: Records each sale, including the date, time, customer, payment method, and the items sold.*
- *Payment: Information about the payment methods and transactions.*
- *Instrument: Details of the instruments available in the shop, including price, brand, type, and quantity in stock.*
- *Brand, Type, Category: These entities categorize instruments into different brands, types, and categories.*
- *Delivery, DeliveryMethod, DeliveryCompany: Capture information related to the delivery of instruments, the methods used, and the companies handling deliveries.*

These entities are interconnected, ensuring that every sale can be traced back to the customer who made the purchase, the instruments sold, the payment method used, and the delivery details if applicable.

Relational Model: *The relational model translates this logical structure into a database schema, with tables for each entity. Each table includes:*

- ***Attributes****: Specific details relevant to each entity, such as names, IDs, dates, and prices.*

- ***Primary Keys****: Unique identifiers for each record in a table, ensuring that each record can be uniquely identified.*
- ***Foreign Keys****: Keys that link records in one table to records in another, establishing relationships between tables.*

For example, the SaleOrder table has foreign keys linking it to the Customer table (to know who made the purchase), the Payment table (to know how it was paid for), and the Instrument table (to know what was purchased).

The relational model ensures that data is organized in a way that minimizes redundancy (through normalization) and maintains data integrity. By using primary and foreign keys, the model supports efficient data operations, allowing for quick retrieval, updates, and deletions.

Overall, this structured and detailed model allows the music shop to manage its operations effectively, ensuring that all data is interconnected and easily accessible.

Part 2: Description of the Views and Triggers

Views

`

CustomerTotalSpending View:

This view aggregates the total spending of each customer. It joins the Customer table with the SaleOrder table on the customer ID and sums up the total price of all sales orders made by each customer. The result includes the customer's name, surname, ID, and the total amount they have spend

```
CREATE VIEW CustomerTotalSpending AS
SELECT c.name_ AS customer_name,
       c.surname AS customer_surname,
       c.cus_id AS customer_id,
       SUM(so.total_price) AS total_spent
FROM Customer c
JOIN SaleOrder so ON c.cus_id = so.cus_id
GROUP BY c.cus_id;
```

InstrumentSalesRanking View:

This view ranks instruments based on their total sales. It joins the Instrument table with the SaleInstr table on the instrument ID, summing the total amount sold for each instrument. Instruments are ranked in descending order of total sales amount. The result includes the instrument ID, name, total amount sold, and its sales rank.

```sql
CREATE OR REPLACE VIEW InstrumentSalesRanking AS
SELECT
    i.Inst_id,
    i.ins_name,
    COALESCE(SUM(s.amount), 0) AS total_amount_sold,
    RANK() OVER (ORDER BY COALESCE(SUM(s.amount), 0) DESC) AS sales_rank
FROM
    Instrument i
LEFT JOIN
    SaleInstr s ON i.Inst_id = s.Inst_id
GROUP BY
    i.Inst_id,
    i.ins_name;
```

Triggers:

Trigger: before_insert_saleinstr

This trigger ensures that before a new record is inserted into the SaleInstr table, the total price is calculated based on the price of the instrument and the amount being purchased. It retrieves the instrument price from the Instrument table and sets the total_price field of the new record accordingly.

```
CREATE TRIGGER before_insert_saleinstr
BEFORE INSERT ON SaleInstr
FOR EACH ROW
BEGIN
    DECLARE instrument_price DECIMAL(10,2);

    -- Get the price of the instrument
    SELECT price INTO instrument_price FROM Instrument WHERE Inst_id = NEW.Inst_id;

    -- Calculate the total price
    SET NEW.total_price = NEW.amount * instrument_price;
END //
```

Trigger: before_insert_saleinstr_update_inventory

This trigger operates before inserting a new record into the SaleInstr table. It checks the current stock of the instrument to ensure there is enough inventory to fulfill the order. If there is insufficient stock, it signals an error. If there is enough stock, it updates the inventory by subtracting the purchased amount and calculates the total price for the new record.

```
DELIMITER //

CREATE TRIGGER before_insert_saleinstr_update_inventory
BEFORE INSERT ON SaleInstr
FOR EACH ROW
BEGIN
    DECLARE available_amount INTEGER;

    -- Get the current amount on store for the instrument
    SELECT amount_on_store INTO available_amount FROM Instrument WHERE Inst_id = NEW.Inst_id;

    -- Check if there is enough stock
    IF available_amount < NEW.amount THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Not enough stock for this instrument';
    ELSE
        -- Subtract the amount being purchased from the stock
        UPDATE Instrument
        SET amount_on_store = amount_on_store - NEW.amount
        WHERE Inst_id = NEW.Inst_id;

        -- Calculate the total price
        SET NEW.total_price = NEW.amount * (SELECT price FROM Instrument WHERE Inst_id = NEW.Inst_id);
    END IF;
END //

DELIMITER ;
```

Trigger: before_insert_update_total_price

This trigger is executed before a new record is inserted into the SaleOrder table. It calculates the total price of the sale by summing the total prices of all instruments in the sale. It also applies a discount based on the customer's discount level. The discount amount is retrieved from the Discount table, and the final total price is calculated by applying this discount.

```
DELIMITER //

CREATE TRIGGER before_insert_update_total_price
BEFORE INSERT ON SaleOrder
FOR EACH ROW
BEGIN
    DECLARE total DOUBLE(10,2);
    DECLARE discountr DOUBLE(3,3);
    DECLARE customer_disc_id INTEGER; -- Declare variable here

    -- Calculate total price from SaleInstr table for the given tran_id
    SELECT SUM(total_price) INTO total FROM SaleInstr WHERE tran_id = NEW.tran_id;

    -- Get disc_id from Customer table for the given cus_id
    SELECT disc_id INTO customer_disc_id FROM Customer WHERE cus_id = NEW.cus_id;

    -- Get discount amount from Discount table based on the disc_id obtained from the Customer table
    SELECT dis_amount INTO discountr FROM Discount WHERE disc_id = customer_disc_id;

    -- Calculate final total price after discount
    IF discountr IS NOT NULL THEN
        SET NEW.total_price = total - (total * discountr);
    ELSE
        SET NEW.total_price = total;
    END IF;
END //

DELIMITER ;
```

Trigger: update_customer_discount

This trigger is executed after a new record is inserted into the SaleOrder table. It calculates the total spending of the customer across all sales. Based on the total amount spent, it determines the appropriate discount level from the Discount table and updates the customer's discount level in the Customer table accordingly.

```
CREATE TRIGGER update_customer_discount
AFTER INSERT ON SaleOrder
FOR EACH ROW
BEGIN
    DECLARE total_spent DECIMAL(10, 2);
    DECLARE new_disc_id INTEGER;

    -- Calculate the total spending of the customer
    SELECT SUM(total_price) INTO total_spent
    FROM SaleOrder
    WHERE cus_id = NEW.cus_id;

    -- Determine the appropriate discount level
    IF total_spent >= (
        SELECT MAX(sum_to_get)
        FROM Discount
    ) THEN
        SET new_disc_id = (
            SELECT disc_id
            FROM Discount
            WHERE sum_to_get = (
                SELECT MAX(sum_to_get)
                FROM Discount
            )
        );
    ELSE
        SET new_disc_id = (
            SELECT disc_id
            FROM Discount
            WHERE sum_to_get <= total_spent
            ORDER BY sum_to_get DESC
            LIMIT 1
        );
    END IF;

    -- Update the Customer table with the new discount level
    UPDATE Customer
    SET disc_id = new_disc_id
    WHERE cus_id = NEW.cus_id;
END //
```

Part 3: Application

a# Importing all needed libraries

```
from sqlalchemy import create_engine, Column, Integer, String, Numeric, Date, Time, DateTime, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker
import sqlalchemy
import re
import pandas as pd
from sqlalchemy import func
from datetime import datetime
import random
```

b# Defining connection to database and defining classes and views for every table in our database for easier usage

```
engine = create_engine('mysql+pymysql://root@localhost/musicshop')
Base = sqlalchemy.orm.declarative_base()
Session = sessionmaker(bind=engine)
session = Session()

#Database tables as Python classes
class Brand(Base):
    __tablename__ = 'brand'

    brand_id = Column(Integer, primary_key=True)
    name_of_brand = Column(String(20), nullable=False)
    rating = Column(Numeric(20), nullable=False)
```

....

c# Defining class person with private variables(also getters and setters) for storing info about current user

```
class Person:
    def __init__(self):
        self.__name = ""
        self.__surname = ""
        self.__phone_no = ""
        self.__address = ""
        self.__disc_id = 1
        self.__ccus_id = None
```

d# Creating function that shows our Rank of sales view

```
def show_instrument_sales_view():
    results = session.query(InstrumentSalesView).all()

    if results:
        print(f"{'Instrument ID':<15}{'Instrument Name':<30}{'Total Amount Sold':<20}{'Sales Rank':<10}")
        print("="*75)
        for row in results:
            print(f"{row.Inst_id:<15}{row.ins_name:<30}{row.total_amount_sold:<20}{row.sales_rank:<10}")
    else:
        print("No data available in the view.")
```

e#Creating helping functions

```python
def validate_input(prompt, pattern, example):
    while True:
        value = input(prompt)
        if re.match(pattern, value):
            return value
        else:
            print(f"Invalid input. Example of valid input: {example}")
```

Making a loop for input data with pattern for it and if data isn't matching to it returns example message and again starting the loop

```python
def fetch_cus_id(name, surname, phone_no, address):
    # Query the database to find the customer with matching details
    customer = session.query(Customer).filter_by(
        name_=name, surname=surname, phone_no=phone_no, address=address).first()

    if customer:
        return customer.cus_id  # Return the cus_id if a customer is found
    else:
        return None  # Return None if no matching customer is found
```

Returning cus_id by it name,surname,phone_no and address

```python
def insert_sale_instr(tran_id, inst_id, amount):
    instrument = session.query(Instrument).filter_by(Inst_id=inst_id).first()
    if instrument:
        total_price = instrument.price * amount
        sale_instr = SaleInstr(tran_id=tran_id, Inst_id=inst_id, amount=amount, total_price=total_price)
        session.add(sale_instr)
        session.commit()
        return total_price
    else:
        print(f"Instrument with ID {inst_id} not found.")
        return None

def get_instrument_id(name):
    instrument = session.query(Instrument).filter_by(ins_name=name).first()
    if instrument:
        print(instrument.Inst_id)
        return instrument.Inst_id
    else:
        print(f"Instrument {name} not found.")
        return None

def show_total_price(tran_id):
    total = session.query(func.sum(SaleInstr.total_price)).filter_by(tran_id=tran_id).scalar()
    print(f"Total price for transaction : {total}")
```

insert_sale_instr - adding new row to SaleInstr table
get_instrument__id - returning the id of instrument by its name
show_total_price - returning total_sum of transaction by knowing tran_id

f#Creating function to show all instrument which are on shop

```
def show_instruments():
    # Define the SQL query to fetch the instruments data
    query = """
    SELECT ins_name AS 'Instrument Name', amount_on_store AS 'Amount on Store', price AS 'Price'
    FROM Instrument
    WHERE amount_on_store > 0;
    """

    # Execute the query and fetch the data into a pandas DataFrame
    with engine.connect() as connection:
        df = pd.read_sql(query, connection)

    # Display the DataFrame
    print(df)
```

g# Registration function

```python
# Registration function
def register_customer():
    name = validate_input("Enter your name: ", r'^[A-Za-z]+$', "John")
    person.set_name(name)
    surname = validate_input("Enter your surname: ", r'^[A-Za-z]+$', "Doe")
    person.set_surname(surname)
    phone_no = validate_input("Enter your phone number (e.g., 123-456-7890): ", r'^\d{3}-\d{3}-\d{4}$', "123-456-7890")
    person.set_phone_no(phone_no)
    address = validate_input("Enter your address: ", r'^[A-Za-z0-9\s,]+$', "123 Main St")
    person.set_phone_no(phone_no)

    new_customer = Customer(name_=name, surname=surname, phone_no=phone_no, address=address, disc_id=1)

    session.add(new_customer)
    session.commit()
    ccus_id = fetch_cus_id(name, surname, phone_no, address)
    if ccus_id is None:
        print("Customer not found in the database.")
        return None

    person.set_cus_id(ccus_id)
    print("Registration successful!")
```

Registering current user and stores it in object of Person class for future using

```python
def show_discount_card_info():
    # Get the list of available customer IDs from the Customer table
    available_cus_ids = [cus.cus_id for cus in session.query(Customer).all()]

    # Construct the pattern for validating customer IDs
    pattern = '^(' + '|'.join(str(cus_id) for cus_id in available_cus_ids) + ')$'

    # Get the customer ID using validate_input with the constructed pattern
    cus_id = validate_input("Enter the customer ID: ", pattern, "Invalid customer ID. Please enter a valid customer ID.")

    # Query the Customer table to retrieve the customer's discount ID
    customer = session.query(Customer).filter_by(cus_id=cus_id).first()

    # Check if the customer exists
    if customer:
        # Get the discount ID for the customer
        discount_id = customer.disc_id

        # Query the Discount table to retrieve the discount card information for the customer's discount ID
        discount_card = session.query(Discount).filter_by(disc_id=discount_id).first()

    # Check if the discount card information exists for the provided customer ID
    if discount_card:
        # Display the discount card information
        print("Discount Card Information:")
        print(f"Customer ID: {cus_id}")
        print(f"Discount Percentage: {discount_card.dis_amount * 100} %")
        print(f"Name of a card: {discount_card.name_}")
    else:
        print("Discount card information not found for the provided customer ID.")
```

Checking all available cus_id d adding it as pattern to valid_input taking as input cus_id and return discount card info for person

g# Creating order function

Description: This function is making whole process of making order from ordering exact instrument to writing your delivery_address. It is using a lot of helping functions.

```
def create_new_transaction():
    tran_id = session.query(func.max(SaleInstr.tran_id)).scalar()
    tran_id = 1 if tran_id is None else tran_id + 1
    total_price = 0

    while True:
        name = validate_input("Enter the instrument name: ", r'^[A-Za-z\s]+$', "Invalid input. Please enter a valid instrument name.")
        inst_id = get_instrument_id(name)
        if not inst_id:
            continue
        amount = int(validate_input("Enter the amount: ", r'^\d+$', "Invalid input. Please enter a valid amount."))
        total_price += insert_sale_instr(tran_id, inst_id, amount)

        show_total_price(tran_id)

        choice = validate_input("Choose an option: 'Resign and comeback to menu', 'Order more', 'Continue order process': ", r'^(Resign and comeback to menu|Order more|Contin
        if choice == 'Resign and comeback to menu':
            session.query(SaleInstr).filter_by(tran_id=tran_id).delete()
            session.commit()
            print("Transaction cancelled.")
            break
        elif choice == 'Order more':
            continue
        elif choice == 'Continue order process':
            payment_methods = session.query(Payment).all()
            print("Payment methods:")
            for method in payment_methods:
                print(method.pay_method)
            chosen_method = validate_input("Choose a payment method : ", r'^(Bank Transfer|PayPal||Credit Card)$', "Invalid input. Please choose a valid payment method.")
            payment_id = session.query(Payment.payment_id).filter(Payment.pay_method == chosen_method).scalar()
            if not payment_id:
                print("Payment method not found.")
                continue
            ccus_id = person.get_cus_id()  # Implement this function to fetch emp_no

            # Get current date and time
            current_date = datetime.now().date()
            current_time = datetime.now().time()
```

```python
# Create a new SaleOrder instance and insert it into the database
new_order = SaleOrder(tran_id=tran_id, date_s=current_date, time_s=current_time, total_price=total_price, payment_id=payment_id, cus_id=ccus_id)
session.add(new_order)
session.commit()

delivery_methods = session.query(DeliveryMethod).all()
print("Delivery methods:")
for method in delivery_methods:
    print(f"Name: {method.name_}, Time for delivery: {method.time_for_del}, Additional Price: {method.aditional_price}")
chosen_method_name = validate_input("Choose a delivery method: ", r'^(Standard|Express|Inpost)$', "Invalid input. Please choose a valid delivery method.")
chosen_method = session.query(DeliveryMethod).filter(DeliveryMethod.name_ == chosen_method_name).first()
if chosen_method:
    del_method_id = chosen_method.del_method_id
else:
    print("Delivery method not found.")
    continue
sale_order = session.query(SaleOrder).filter_by(tran_id=tran_id).first()
if not sale_order:
    print("SaleOrder not found.")
    return

# Take a random emp_no from all DeliveryCompany
delivery_companies = session.query(DeliveryCompany).all()
if not delivery_companies:
    print("No delivery companies found.")
    return
random_delivery_company = random.choice(delivery_companies)
emp_no = random_delivery_company.emp_no

# Get del_method_id from the previously chosen delivery method
if not del_method_id:
    print("Delivery method not found.")
    return

# Ask for delivery address
delivery_adress = validate_input("Enter delivery address: ", r'^[A-Za-z0-9\s,]+$', "123 Main St")
```

```
# Ask for delivery address
delivery_adress = validate_input("Enter delivery address: ", r'^[A-Za-z0-9\s,]+$', "123 Main St")

# Insert new row into the Delivery table
new_delivery = Delivery(delivery_adres=delivery_adress, delivery_status='registered', sale_id=sale_order.sale_id, del_method_id=del_method_id, emp_no=emp_no)
session.add(new_delivery)
session.commit()

print('Your order is succesfuly regestrated')

break
```

g#Print menu and main functions

```python
def print_menu():
    print("Welcome to the Music Shop!")
    print("1. Show list of instruments")
    print("2. Make an order")
    print("3. Show current discount card")
    print("4. Show instrument ranking")
    print("5. Exit")

def main():
    print("Hi,first of all complete registration:)")
    register_customer()
    while True:
        print_menu()
        choice = validate_input("Enter your choice: ", r'^(1|2|3|4|5)$', 'Choose option from 1- 5')
        if choice == "1":
            show_instruments()
        elif choice == "2":
            create_new_transaction()
            print("Making an order...")
        elif choice == "3":
            print("Showing current discount card...")
            show_discount_card_info()
        elif choice == "4":
            print("Showing instrument ranking...")
            show_instrument_sales_view()
        elif choice == "5":
            print("Exiting...")
            break

if __name__ == "__main__":
    main()
```

Part 4: Application Example of usage

1)Making registration

```
Hi,first of all complete registration:)
Enter your name: Volodymyr
Enter your surname: Chuchka
Enter your phone number (e.g., 123-456-7890): 888-645-8419
Enter your address: ul. Szafrana 8
Invalid input. Example of valid input: 123 Main St
Enter your address: ul Szafrana 8
Registration successful!
Welcome to the Music Shop!
1. Show list of instruments
2. Make an order
3. Show current discount card
4. Show instrument ranking
5. Exit
Enter your choice:
```

17	17	Volodymyr	Chuchka	888-645-8419	ul Szafrana 8	3

2)Looking on list of instruments

```
                Instrument Name  Amount on Store    Price
0  Yamaha Acoustic Guitar                 5   799.99
1  Fender Electric Guitar                 6   899.99
2            Roland Drums                 4   499.99
3            Yamaha Flute                 9   299.99
4            Bach Trumpet                 4   899.99
5         Moog Synthesizer                1  1199.99
6          Buffet Clarinet                5   599.99
7            Deering Banjo                5   799.99
```

3)making an order

```
Enter the instrument name: Deering banjo
9
Enter the amount: 2
Total price for transaction : 1599.98
Choose an option: 'Resign and comeback to menu', 'Order more', 'Continue order process': Continue order process
Payment methods:
Credit Card
PayPal
Bank Transfer
Choose a payment method : PayPal
Delivery methods:
Name: Standard, Time for delivery: 3-5 days, Additional Price: 5.0
Name: Express, Time for delivery: 1-2 days, Additional Price: 10.0
Name: Inpost, Time for delivery: 3 days, Additional Price: 15.0
Choose a delivery method: Inpost
Enter delivery address: ul Szafrana 8
Your order is succesfuly regestrated
Making an order...
```

22	**15**	2	**9**	1 599,98	**sale ints**

14	**14**	15	2024-06-19	11:15:11	1 599,98	2	17	**sale order**

10	ul Szafrana 8	registered	14	3	3	**delivery**

Part 5: DataBaseCreator script:

```
CREATE OR replace TABLE brand (
    brand_id INTEGER NOT NULL AUTO_INCREMENT,
    name_of_brand VARCHAR(20) NOT NULL,
    rating NUMERIC(20) NOT NULL,
    PRIMARY KEY (brand_id)
);

CREATE OR replace TABLE Category (
    category_id INTEGER NOT NULL AUTO_INCREMENT,
    categ_name VARCHAR(20) NOT NULL,
    PRIMARY KEY (category_id)
);



CREATE OR replace TABLE delivery_method (
    del_method_id INTEGER NOT NULL AUTO_INCREMENT,
    aditional_price DOUBLE(10,2) NOT NULL,
    name_ VARCHAR(10) NOT NULL,
    time_for_del VARCHAR(10) NOT NULL,
    PRIMARY KEY (del_method_id)
);

CREATE OR Replace TABLE Payment (
    payment_id INTEGER NOT NULL AUTO_INCREMENT,
    pay_method VARCHAR(100) NOT NULL,
    PRIMARY KEY (payment_id)
);

CREATE OR replace TABLE delivery_company (
    emp_no INTEGER NOT NULL AUTO_INCREMENT,
    name_ VARCHAR(20) NOT NULL,
    surname VARCHAR(20) NOT NULL,
    phone_no VARCHAR(16) NOT NULL,
    PRIMARY KEY (emp_no)
```

```
);

CREATE OR replace TABLE Discount (
    disc_id INTEGER NOT NULL AUTO_INCREMENT,
    dis_amount DOUBLE(3,3) NOT NULL,
    sum_to_get DOUBLE(10,2) NOT NULL,
    name_ VARCHAR(100) NOT NULL,
    PRIMARY KEY (disc_id)
);

CREATE OR replace TABLE Customer (
    cus_id INTEGER NOT NULL AUTO_INCREMENT,
    name_ VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    phone_no VARCHAR(100) NOT NULL,
    address VARCHAR(100) NOT NULL,
    disc_id INTEGER NOT NULL,
    PRIMARY KEY (cus_id),
    FOREIGN KEY (disc_id) REFERENCES discount(disc_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE OR replace TABLE Type_ (
    type_id INTEGER NOT NULL AUTO_INCREMENT,
    type_name VARCHAR(100) NOT NULL,
    category_id INTEGER NOT NULL,
    PRIMARY KEY (type_id),
    FOREIGN KEY (category_id) REFERENCES Category(category_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE OR replace TABLE Instrument (
    Inst_id INTEGER NOT NULL AUTO_INCREMENT,
    price DECIMAL(10,2) NOT NULL,
    ins_name VARCHAR(255) NOT NULL,
    amount_on_store INTEGER(5) NOT NULL,
    brand_id INTEGER NOT NULL,
    type_id INTEGER NOT NULL,
    PRIMARY KEY (Inst_id, ins_name),
    FOREIGN KEY (brand_id) REFERENCES brand(brand_id) ON DELETE NO ACTION ON UPDATE NO ACTION,
```

```
    FOREIGN KEY (type_id) REFERENCES Type_(type_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE OR replace TABLE SaleInstr (
    tran_id INTEGER NOT NULL Auto_increment ,
    amount INTEGER NOT NULL,
    Inst_id INTEGER NOT NULL,
    total_price DOUBLE(10,2) NOT NULL,
    PRIMARY KEY (tran_id,Inst_id),
    FOREIGN KEY (Inst_id) REFERENCES Instrument(Inst_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE OR replace TABLE SaleOrder (
    sale_id INTEGER NOT NULL AUTO_INCREMENT,
    tran_id INTEGER NOT NULL,
    date_s DATE NOT NULL,
    time_s TIME NOT NULL,
    total_price DOUBLE(10,2) ,
    payment_id INTEGER NOT NULL,
    cus_id INTEGER NOT NULL,
    PRIMARY KEY (sale_id),
    FOREIGN KEY (tran_id) REFERENCES SaleInstr(tran_id) ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY (payment_id) REFERENCES Payment(payment_id) ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY (cus_id) REFERENCES Customer(cus_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);



CREATE OR replace TABLE Delivery (
    delivary_id INTEGER NOT NULL AUTO_INCREMENT,
    delivery_adres VARCHAR(40) NOT NULL,
    delivery_status VARCHAR(10) NOT NULL,
    sale_id INTEGER NOT NULL,
    del_method_id INTEGER NOT NULL,
    emp_no INTEGER NOT NULL,
    PRIMARY KEY (delivary_id),
    FOREIGN KEY (del_method_id) REFERENCES delivery_method(del_method_id) ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY (emp_no) REFERENCES delivery_company(emp_no) ON DELETE NO ACTION ON UPDATE NO ACTION,
```

```
    FOREIGN KEY (sale_id) REFERENCES SaleOrder(sale_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

ALTER TABLE SaleOrder ADD CONSTRAINT unique_tran_id UNIQUE (tran_id);
```