ДОДАТОК А КОД МОУДЛЯ

Register.java

package org.vitapasser.photocopypoint.Model;

import org.vitapasser.photocopypoint.Exception.NotExistTypeException;

import java.util.List;

public class Register {

private Order order;

private final TypeList typeList;

private final PickUpStation pickUpStation;

private final TicketList ticketList;

public Register(TypeList typeList, PickUpStation pickUpStation, TicketList ticketList) {

this.typeList = typeList;

this.pickUpStation = pickUpStation;

this.ticketList = ticketList;

}

public void newOrder() {

this.order = Order.create();

}

public List<TypeItem> addType(String name, int count) {

order.addType(name, typeList, count);

return order.getTypeItems();

}

public Term getTerm() {

return order.getTerm();

}

public Money getPrice() {

return order.getPrice();

}

public Money makePayment(Money countPayMoney,

String fullName,

String phoneNumber){

return order.makePayment(countPayMoney,

fullName,

phoneNumber,

pickUpStation,

ticketList);

}

public List<Type> getTypes(String likeNameTickets) throws NotExistTypeException {

return typeList.getTypes(likeNameTickets);

}

public List<TypeItem> getOrderTypes() {

return order.getTypeItems();

}

public List<Ticket> getAllTickets() {

return ticketList.getAllTickets();

}

}

Money.java

package org.vitapasser.photocopypoint.Model;

import java.util.Objects;

public record Money(Double value, String unit) {

public Money sum(Money money) {

try {

if (!Objects.equals(unit, money.unit())){

throw new Exception("Money units don't match");

}

return new Money(value + money.value(), unit);

} catch (Exception e) {

throw new RuntimeException(e);

}

}

@Override

public String toString() {

return String.format("%.2f", value) + " " + unit;

}

}

Operator.java

package org.vitapasser.photocopypoint.Model;

public record Operator(Long id, String name) {}

Order.java

package org.vitapasser.photocopypoint.Model;

import java.util.ArrayList;

import java.util.List;

public class Order {

private List<TypeItem> typeItems;

private Term term;

private Money price;

private String fullName;

private String phoneNumber;

private Operator operator;

private Order() {}

public static Order create() {

Order order = new Order();

order.typeItems = new ArrayList<>();

return order;

}

public void addType(String name, TypeList typeList, int count) {

Type type = typeList.getType(name);

typeItems.add(new TypeItem(type, count));

}

public Term getTerm() {

term = typeItems.stream()

.map(TypeItem::getTerm)

.reduce(Term::sum)

.orElseThrow();

return term;

}

public Money getPrice() {

price = typeItems.stream()

.map(TypeItem::getMoney)

.reduce(Money::sum)

.orElseThrow();

return price;

}

public Money makePayment(Money countPayMoney,

String fullName,

String phoneNumber,

PickUpStation pickUpStation,

TicketList ticketList) {

this.fullName = fullName;

this.phoneNumber = phoneNumber;

operator = pickUpStation.getOperator();

OrderIDAndOddMany OrderIDAndOddMany = pickUpStation.fixSale(this, countPayMoney);

assert OrderIDAndOddMany != null;

Money change = OrderIDAndOddMany.oddMoney();

ticketList.createTicket(OrderIDAndOddMany.orderID());

return change;

}

/\*

\* Якщо потрібно буде отримати данні замовлення для інших програмних потреб.

\* \*/

public List<TypeItem> getTypeItems() {

return typeItems;

}

public String getFullName() {

return fullName;

}

public String getPhoneNumber() {

return phoneNumber;

}

public Operator getOperator() {

return operator;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public void setOperator(Operator operator) {

this.operator = operator;

}

}

OrderIDAndOddMoney.java

package org.vitapasser.photocopypoint.Model;

public record OrderIDAndOddMany(long orderID, Money oddMoney) {}

PickUpStation.java

package org.vitapasser.photocopypoint.Model;

import java.sql.\*;

import java.util.Objects;

public record PickUpStation(Connection connectionToDataBase, String address) {

/\*

\* Через запрос у базу даних отримуються вільний оператор

\* та повератється методом. (return null - є заглушка)

\*/

public Operator getOperator() {

try {

Statement statement = connectionToDataBase.createStatement();

ResultSet sqlResult = statement.executeQuery(

"SELECT PhotocopyPoint.Contact\_info.id, PhotocopyPoint.Contact\_info.full\_name\n" +

"From PhotocopyPoint.Contact\_info \n" +

"inner join PhotocopyPoint.Staff on PhotocopyPoint.Staff.contact\_info\_id " +

"= PhotocopyPoint.Contact\_info.id \n" +

"inner join PhotocopyPoint.PickUpStationStaff on " +

"PhotocopyPoint.PickUpStationStaff.staff\_id = PhotocopyPoint.Staff.id\n" +

"left join PhotocopyPoint.OrderStaff on " +

"PhotocopyPoint.OrderStaff.staff\_id = ALL(" +

"Select PhotocopyPoint.Staff.id from PhotocopyPoint.Staff " +

"inner join PhotocopyPoint.PickUpStation " +

"on PhotocopyPoint.PickUpStation.id = " +

"PhotocopyPoint.PickUpStationStaff.pickUpStation\_id " +

"WHERE PhotocopyPoint.PickUpStation.address = \""+address+"\" )\n" +

"left join PhotocopyPoint.`Order` on PhotocopyPoint.`Order`.id = " +

"PhotocopyPoint.OrderStaff.order\_id\n" +

"left join PhotocopyPoint.TypeServiceOrder " +

"on PhotocopyPoint.TypeServiceOrder.id = PhotocopyPoint.`Order`.`id`\n" +

"where (PhotocopyPoint.TypeServiceOrder.term < NOW() or " +

"(PhotocopyPoint.TypeServiceOrder.term is null)) LIMIT 1;");

sqlResult.next();

return new Operator(sqlResult.getLong("id"),

sqlResult.getString("full\_name"));

} catch (Exception e) {

System.out.println(e.getMessage());

}

return null;

}

/\*

\* Через запрос у базу даних отправляється замовлення.

\* Також метод рахує здачу. (return null - є заглушка)

\*/

public OrderIDAndOddMany fixSale(Order order, Money countPayMoney) {

try {

long idMoney = insertMoney(order);

long idContactInfo = insertContactInfo(order);

long idClient = insertClient(idContactInfo);

long idOrder = insertOrder(idMoney, idClient);

order.getTypeItems().forEach(type -> {

try {

insertTypeServiceOrder(idOrder, type, order.getTerm());

} catch (SQLException e) {

throw new RuntimeException(e);

}

});

insertOrderStaff(idOrder, order.getOperator());

Money oddMoney = null;

Money orderPrice = order.getPrice();

if (Objects.equals(orderPrice.unit(), countPayMoney.unit()))

oddMoney = new Money(countPayMoney.value() - orderPrice.value(),

orderPrice.unit());

return new OrderIDAndOddMany(idOrder, oddMoney);

} catch (Exception e) {

System.out.println(e.getMessage());

e.printStackTrace();

}

return null;

}

private void insertOrderStaff(long idOrder, Operator operator) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.OrderStaff(order\_id, staff\_id, specialization) " +

"VALUES (?, ?, 'Оператор принтеру');\n");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setLong(1, idOrder);

statement.setLong(2, operator.id());

checkAffectRows(statement.executeUpdate());

getId(statement);

}

private void insertTypeServiceOrder(long idOrder, TypeItem type, Term term) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.TypeServiceOrder(order\_id, term, type\_service\_id, " +

"count) VALUES (?, ADDTIME(NOW(), SEC\_TO\_TIME(?)), ?, ?);");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setLong(1, idOrder);

//statement.setLong(2, type.);

statement.setInt(2, term.value());

statement.setLong(3, type.type().id());

statement.setInt(4, type.count());

checkAffectRows(statement.executeUpdate());

getId(statement);

}

private long insertOrder(long idMoney, long idClient) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.`Order`(money\_id, client\_id) " +

"VALUES (?, ?);\n");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setLong(1, idMoney);

statement.setLong(2, idClient);

checkAffectRows(statement.executeUpdate());

return getId(statement);

}

private long insertClient(long id) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.Client(station\_name, contact\_info\_id) " +

"VALUES (?, ?);\n");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setString(1, address);

statement.setLong(2, id);

checkAffectRows(statement.executeUpdate());

return getId(statement);

}

private long insertContactInfo(Order order) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.Contact\_info(full\_name, phone\_number) " +

"VALUES (?, ?);\n");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setString(1, order.getFullName());

statement.setString(2, order.getPhoneNumber());

checkAffectRows(statement.executeUpdate());

return getId(statement);

}

private long insertMoney(Order order) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.Money(count, unit) VALUES (?, ?);");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setDouble(1, order.getPrice().value());

statement.setString(2, order.getPrice().unit());

checkAffectRows(statement.executeUpdate());

return getId(statement);

}

private static long getId(PreparedStatement statement) throws SQLException {

try (ResultSet generatedKeys = statement.getGeneratedKeys()) {

if (generatedKeys.next()) {

return generatedKeys.getLong(1);

}

else {

throw new SQLException("Creating failed, no ID obtained.");

}

}

}

private static void checkAffectRows(int affectedRows) throws SQLException {

if (affectedRows == 0) {

throw new SQLException("Creating failed, no rows affected.");

}

}

}

Term.java

package org.vitapasser.photocopypoint.Model;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

public record Term(Integer value) {

public Term sum(Term term) {

return new Term(this.value() + term.value());

}

public LocalTime toLocalTime(){

return LocalTime.ofSecondOfDay(value);

}

@Override

public String toString() {

LocalTime time = toLocalTime();

return time.format(DateTimeFormatter.ofPattern("HH:mm:ss"));

}

}

Ticket.java

package org.vitapasser.photocopypoint.Model;

import java.util.Objects;

public final class Ticket {

private long id;

private long orderId;

private String name;

private boolean isUsed;

private String fullName;

private String phoneNumber;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public long getOrderId() {

return orderId;

}

public void setOrderId(long orderId) {

this.orderId = orderId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public boolean getIsUsed() {

return isUsed;

}

public void setIsUsed(boolean used) {

isUsed = used;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public Ticket(long id,

long orderId,

String name,

boolean isUsed,

String fullName,

String phoneNumber) {

this.id = id;

this.orderId = orderId;

this.name = name;

this.isUsed = isUsed;

this.fullName = fullName;

this.phoneNumber = phoneNumber;

}

@Override

public boolean equals(Object obj) {

if (obj == this) return true;

if (obj == null || obj.getClass() != this.getClass()) return false;

var that = (Ticket) obj;

return this.id == that.id &&

this.orderId == that.orderId &&

Objects.equals(this.name, that.name) &&

this.isUsed == that.isUsed &&

Objects.equals(this.fullName, that.fullName) &&

Objects.equals(this.phoneNumber, that.phoneNumber);

}

@Override

public int hashCode() {

return Objects.hash(id, orderId, name, isUsed, fullName, phoneNumber);

}

@Override

public String toString() {

return "Ticket[" +

"id=" + id + ", " +

"orderId=" + orderId + ", " +

"name=" + name + ", " +

"isUsed=" + isUsed + ", " +

"fullName=" + fullName + ", " +

"phoneNumber=" + phoneNumber + ']';

}

}

TicketList.java

package org.vitapasser.photocopypoint.Model;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public record TicketList(Connection connectionToDataBase) {

public List<Ticket> getAllTickets() {

try {

String sql = """

SELECT PT.id,

PT.order\_id,

GROUP\_CONCAT(TS.name SEPARATOR ', ') AS all\_names,

PT.isUsed,

Ci.full\_name,

Ci.phone\_number

FROM PhotocopyPoint.Ticket PT

INNER JOIN PhotocopyPoint.`Order` PO ON PO.id = PT.order\_id

INNER JOIN PhotocopyPoint.Client C ON PO.client\_id = C.id

INNER JOIN PhotocopyPoint.Contact\_info Ci ON C.contact\_info\_id = Ci.id

INNER JOIN PhotocopyPoint.TypeServiceOrder TSO ON PO.id = TSO.order\_id

INNER JOIN PhotocopyPoint.TypeService TS ON TSO.type\_service\_id = TS.id

GROUP BY PT.id, PT.order\_id, PT.isUsed, Ci.full\_name, Ci.phone\_number

ORDER BY PT.id ASC;

""";

PreparedStatement statement = connectionToDataBase.prepareStatement(sql);

statement.executeQuery();

return getTickets(statement);

} catch (Exception e) {

System.out.println(e.getMessage());

}

return null;

}

private static List<Ticket> getTickets(PreparedStatement statement) throws SQLException {

ResultSet resultSet = statement.getResultSet();

List<Ticket> tickets = new ArrayList<>();

while(resultSet.next()){

tickets.add(new Ticket(resultSet.getLong("id"),

resultSet.getLong("order\_id"),

resultSet.getString("all\_names"),

resultSet.getBoolean("isUsed"),

resultSet.getString("full\_name"),

resultSet.getString("phone\_number")));

}

return tickets;

}

public long createTicket(long idOrder) {

long ticketId = 0;

try {

ticketId = insertTicket(idOrder);

return ticketId;

} catch (SQLException e) {

System.out.println(e.getMessage());

e.printStackTrace();

}

return ticketId;

}

private long insertTicket(long idOrder) throws SQLException {

StringBuilder sql;

PreparedStatement statement;

sql = new StringBuilder("INSERT INTO `PhotocopyPoint`.Ticket(order\_id) VALUES (?);\n");

statement = connectionToDataBase.prepareStatement(sql.toString(), Statement.RETURN\_GENERATED\_KEYS);

statement.setLong(1, idOrder);

checkAffectRows(statement.executeUpdate());

return getId(statement);

}

private static long getId(PreparedStatement statement) throws SQLException {

try (ResultSet generatedKeys = statement.getGeneratedKeys()) {

if (generatedKeys.next()) {

return generatedKeys.getLong(1);

}

else {

throw new SQLException("Creating failed, no ID obtained.");

}

}

}

private static void checkAffectRows(int affectedRows) throws SQLException {

if (affectedRows == 0) {

throw new SQLException("Creating failed, no rows affected.");

}

}

}

Type.java

package org.vitapasser.photocopypoint.Model;

import java.time.LocalDateTime;

public record Type(Long id, String name, String info, Term term, Money money, LocalDateTime create\_time) {

}

TypeItem.java

package org.vitapasser.photocopypoint.Model;

public record TypeItem(Type type, int count) {

public Money getMoney() {

return new Money(type.money().value() \* count, type.money().unit());

}

public Term getTerm() {

return new Term(type.term().value() \* count);

}

}

TypeList.java

package org.vitapasser.photocopypoint.Model;

import org.vitapasser.photocopypoint.Exception.NotExistTypeException;

import org.vitapasser.photocopypoint.Util.Mysql;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class TypeList {

private final Connection connectionToDataBase;

public TypeList(Connection connectionToDataBase) {

this.connectionToDataBase = connectionToDataBase;

}

public List<Type> getTypes(String likeNameTypes) throws NotExistTypeException {

List<Type> types = new ArrayList<>();

try {

String sql = """

SELECT TS.name,

TS.id,

TS.info,

TS.term,

PM.count,

PM.unit,

TS.create\_time

FROM PhotocopyPoint.TypeService TS

INNER JOIN (

SELECT name, MAX(create\_time) AS max\_create\_time

FROM PhotocopyPoint.TypeService

GROUP BY name

) AS latest\_services

ON TS.name = latest\_services.name AND TS.create\_time = latest\_services.max\_create\_time

INNER JOIN PhotocopyPoint.Price PP ON TS.price\_id = PP.id

INNER JOIN PhotocopyPoint.Money PM ON PP.money\_id = PM.id

WHERE TS.name LIKE CONCAT('%', ?, '%')

ORDER BY TS.id DESC;

""";

PreparedStatement statement = connectionToDataBase.prepareStatement(sql);

statement.setString(1, likeNameTypes);

statement.executeQuery();

ResultSet resultSet = statement.getResultSet();

while(resultSet.next()){

types.add(new Type(resultSet.getLong("id"),

resultSet.getString("name"),

resultSet.getString("info"),

new Term(resultSet.getTime("term").toLocalTime().toSecondOfDay()),

new Money(resultSet.getDouble("count"), resultSet.getString("unit")),

Mysql.dbDateTimeToLocalDateTime(resultSet.getString("create\_time"))));

}

} catch (Exception e) {

System.out.println(e.getMessage());

}

if (types.isEmpty()) {

throw new NotExistTypeException("Помилкова назва послуги!");

}

return types;

}

public Type getType(String typeName) {

try {

String sql = """

SELECT TS.id,

TS.name,

TS.info,

TS.term,

PM.count,

PM.unit,

TS.create\_time

From PhotocopyPoint.Money PM

inner join PhotocopyPoint.Price PP on PM.id = PP.money\_id

inner join PhotocopyPoint.TypeService TS on PP.id = TS.price\_id

where TS.name like CONCAT('%', ?, '%')

ORDER BY TS.create\_time DESC LIMIT 1;""";

PreparedStatement statement = connectionToDataBase.prepareStatement(sql);

statement.setString(1, typeName);

statement.executeQuery();

ResultSet resultSet = statement.getResultSet();

resultSet.next();

return new Type(resultSet.getLong("id"),

resultSet.getString("name"),

resultSet.getString("info"),

new Term(resultSet.getTime("term").toLocalTime().toSecondOfDay()),

new Money(resultSet.getDouble("count"), resultSet.getString("unit")),

Mysql.dbDateTimeToLocalDateTime(resultSet.getString("create\_time")));

} catch (Exception e) {

System.out.println(e.getMessage());

}

return null;

}

}

NotExistTypeException.java

package org.vitapasser.photocopypoint.Exception;

public class NotExistTypeException extends Exception{

public NotExistTypeException(String message) {

super(message);

}

}

Mysql.java

package org.vitapasser.photocopypoint.Util;

import java.io.IOException;

import java.io.InputStream;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.time.LocalDateTime;

import java.util.Properties;

public class Mysql {

public static LocalDateTime dbDateTimeToLocalDateTime(String dateTime){

String dateTimeForParse = dateTime.replace(' ', 'T');

return LocalDateTime.parse(dateTimeForParse);

}

public static Connection getConnection() throws SQLException, IOException {

Properties props = new Properties();

try(InputStream in = Files.newInputStream(Paths.get("Configuration/database.properties"))){

props.load(in);

}

String url = props.getProperty("url");

String username = props.getProperty("username");

String password = props.getProperty("password");

return DriverManager.getConnection(url, username, password);

}

}