KSP JAVA 2025

OBJECT PERSISTENCE 2

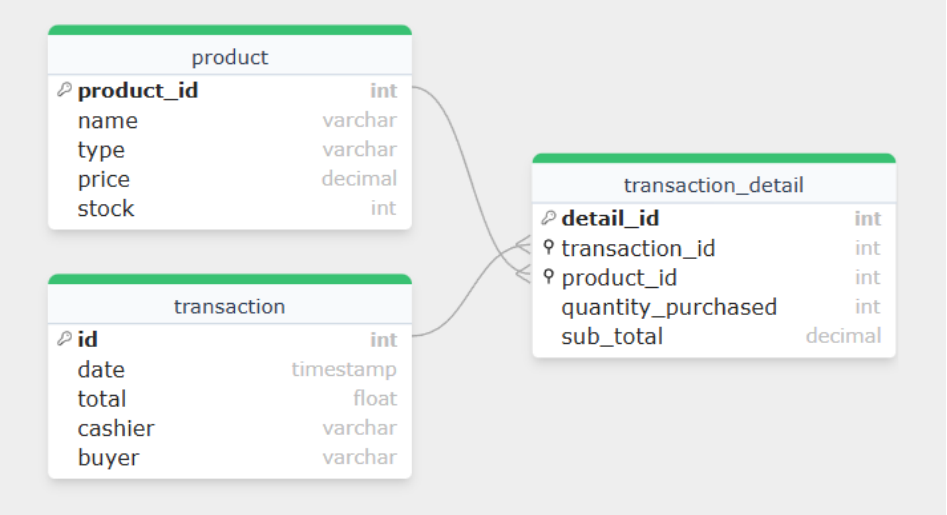
BACKEND

Kevin Philips Tanamas
Tok Se Ka

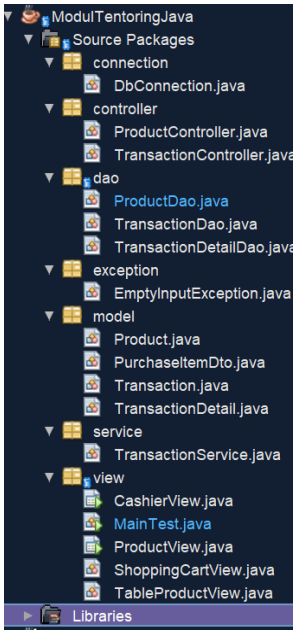
Object Persistence 2 - Backend

Pada modul backend kali ini, adalah lanjutan dari modul OP1. Kita akan membuat menu **Cashier** untuk Warkop PBO yang sudah kita buat sebelumnya. Modul ini mencakup dua aspek utama dalam pengelolaan data di aplikasi Java:

- Operasi **JOIN** → Menggabungkan data dari beberapa tabel.
- Operasi **Transaksional** → Memastikan bahwa seluruh query yang tergabung dalam satu transaksi berjalan dengan benar dan konsisten.

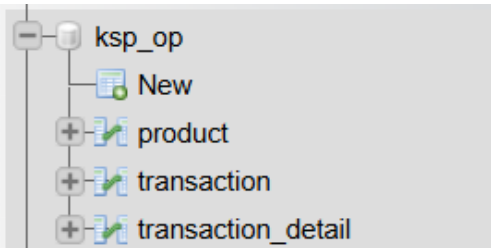


Struktur file akan **sama** dengan struktur file di OP1, untuk memastikan projek kalian memiliki struktur yang sama, berikut ini adalah screenshot struktur file yang digunakan pada modul OP2 :

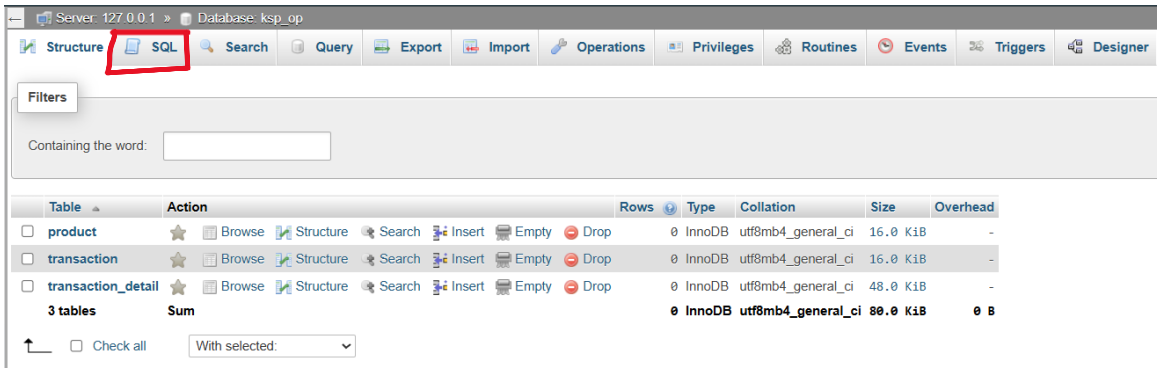


Sebelumnya, silahkan lakukan alter table dulu pada database melalui xampp kalian, tujuannya supaya semua kolom id memiliki tipe data integer, dengan mengikuti langkah-langkah berikut:

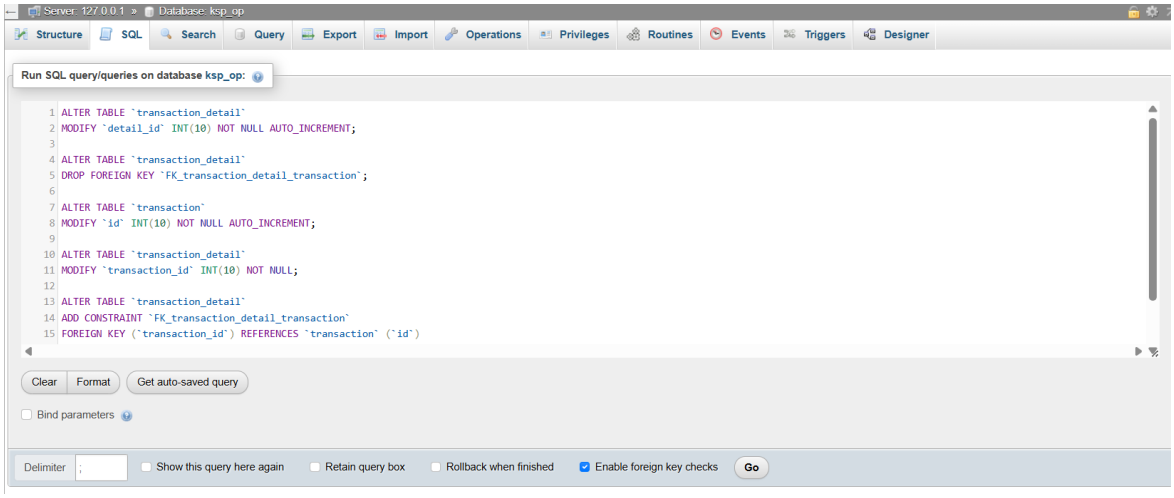
1. Silahkan buka **phpmyadmin** kalian melalui xampp, kemudian **klik** database **ksp_op**



2. Kemudian akan muncul tampilan seperti ini disebelah kanan list database kalian, klik tombol **sql** untuk menjalankan query



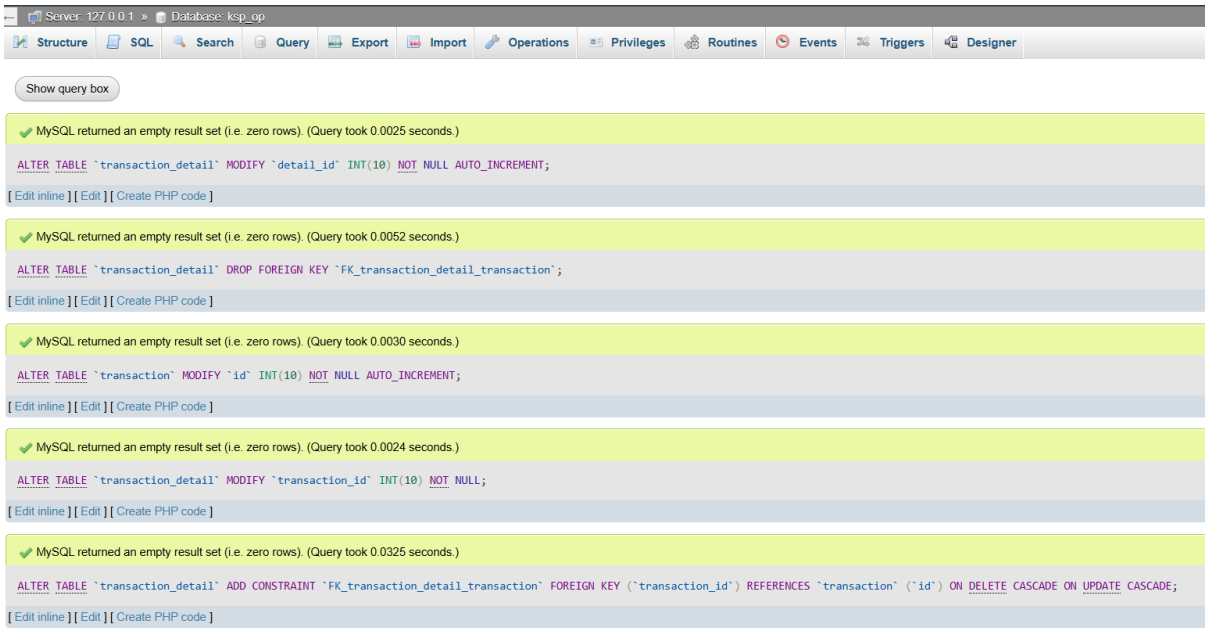
3. Masukan script query berikut ke dalam field inputan sql, kemudian klik **go** untuk menjalankan script (script tersedia dibawah, tinggal dicopy saja)




```
ALTER TABLE `transaction_detail`
MODIFY `detail_id` INT(10) NOT NULL AUTO_INCREMENT;
ALTER TABLE `transaction_detail`
```

```
DROP FOREIGN KEY `FK_transaction_detail_transaction`;
ALTER TABLE `transaction`
MODIFY `id` INT(10) NOT NULL AUTO_INCREMENT;
ALTER TABLE `transaction_detail`
MODIFY `transaction_id` INT(10) NOT NULL;
ALTER TABLE `transaction_detail`
ADD CONSTRAINT `FK_transaction_detail_transaction`
FOREIGN KEY (`transaction_id`) REFERENCES `transaction` (`id`)
ON DELETE CASCADE ON UPDATE CASCADE;
```

4. Setelah muncul tampilan seperti ini, berarti database teman-teman sudah siap untuk dilakukan pengkodean



Selanjutnya, pada bagian ini, kita akan membuat query untuk bagian transaksional, terutama untuk menu **cashier** dari proyek kita.

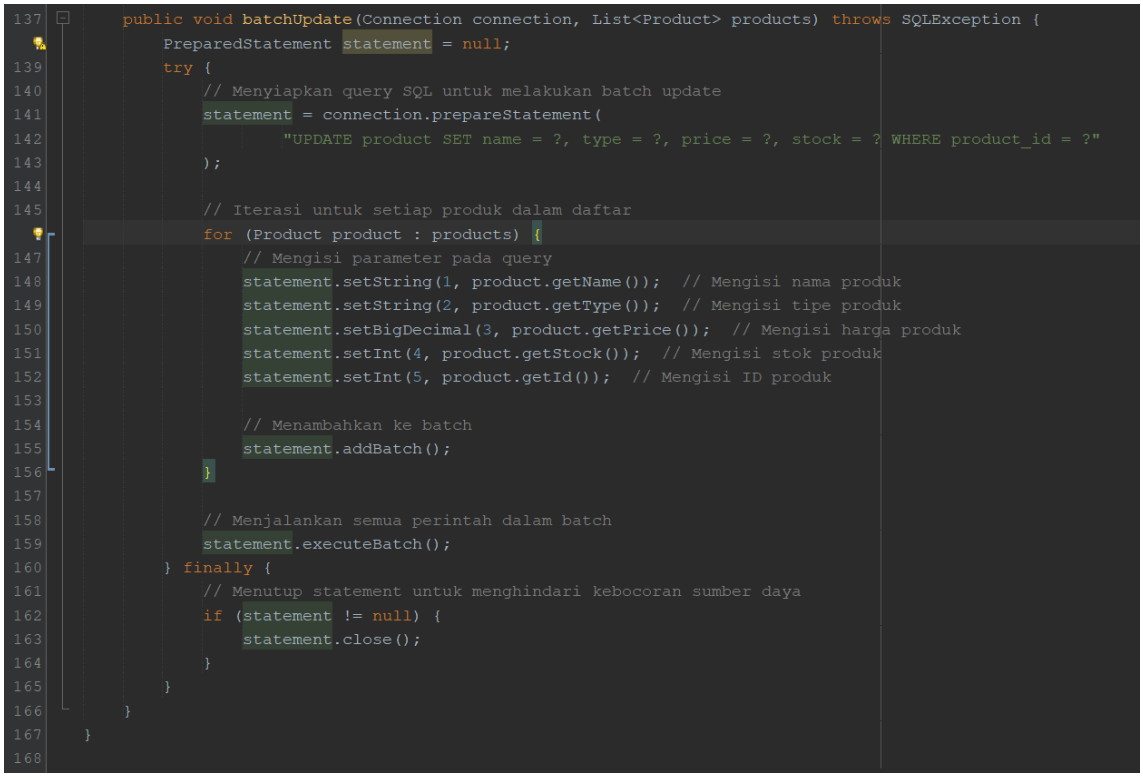
1. Pertama-tama, kita akan melanjutkan dengan membuat model terlebih dahulu. Buatlah dua kelas, yaitu **Transaction.java** dan **TransactionDetail.java**, yang masing-masing merepresentasikan tabel *transaction* dan *transaction_detail*. Pada kelas Transaction, kita akan menetapkan nilai *dummy* untuk *user* dan *buyer* di dalam konstruktor sebagai langkah awal.
    a. **Transaction.java** : copas saja kode di file di Transaction_model.txt
    b. **TransactionDetail.java** : copas saja kode di file di TransactionDetail_model.txt
2. Selain model yang merepresentasikan tabel, kita juga memiliki model DTO (Data Transfer Object), yaitu **PurchaseItemDTO**, yang berfungsi untuk memfasilitasi transfer data antar

layer atau lapisan dalam aplikasi. Cara kerja **PurchaseItemDTO** adalah ketika suatu item dipilih dari menu, objek item tersebut beserta jumlah totalnya akan dikemas dalam bentuk instance **PurchaseItemDTO**. Berikut adalah contoh pengkodean **PurchaseItemDTO** (bisa copas saja kode di file PurchaseItemDto.txt ke PurchaseItemDto.java)

Contohnya: "Nasi Goreng (beserta detail atributnya), total = 5".

```java
public class PurchaseItemDto {
    private Product product;
    private Integer quantityPurchased;

    public PurchaseItemDto(Product product, Integer quantityPurchased) {
        this.product = product;
        this.quantityPurchased = quantityPurchased;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public Integer getQuantityPurchased() {
        return quantityPurchased;
    }

    public void setQuantityPurchased(Integer quantityPurchased) {
        this.quantityPurchased = quantityPurchased;
    }

}

```

3. Pada **ProductDao.java**, tambahkan metode baru dibagian paling bawah seperti berikut:



```java
    public void batchUpdate(Connection connection, List<Product> products) throws SQLException {
        PreparedStatement statement = null;
        try {
            // Menyiapkan query SQL untuk melakukan batch update
            statement = connection.prepareStatement(
                    "UPDATE product SET name = ?, type = ?, price = ?, stock = ? WHERE product_id = ?"
            );

            // Iterasi untuk setiap produk dalam daftar
            for (Product product : products) {
                // Mengisi parameter pada query
                statement.setString(1, product.getName());  // Mengisi nama produk
                statement.setString(2, product.getType());  // Mengisi tipe produk
                statement.setBigDecimal(3, product.getPrice());  // Mengisi harga produk
                statement.setInt(4, product.getStock());  // Mengisi stok produk
                statement.setInt(5, product.getId());  // Mengisi ID produk

                // Menambahkan ke batch
                statement.addBatch();
            }

            // Menjalankan semua perintah dalam batch
            statement.executeBatch();
        } finally {
            // Menutup statement untuk menghindari kebocoran sumber daya
            if (statement != null) {
                statement.close();
            }
        }
    }
}

```

Metode ini berfungsi untuk melakukan update secara **batch**. Dengan menggunakan **batch**, kita dapat meningkatkan efisiensi eksekusi query karena beberapa perintah update akan dijalankan sekaligus dalam satu proses, mengurangi overhead komunikasi dengan database.

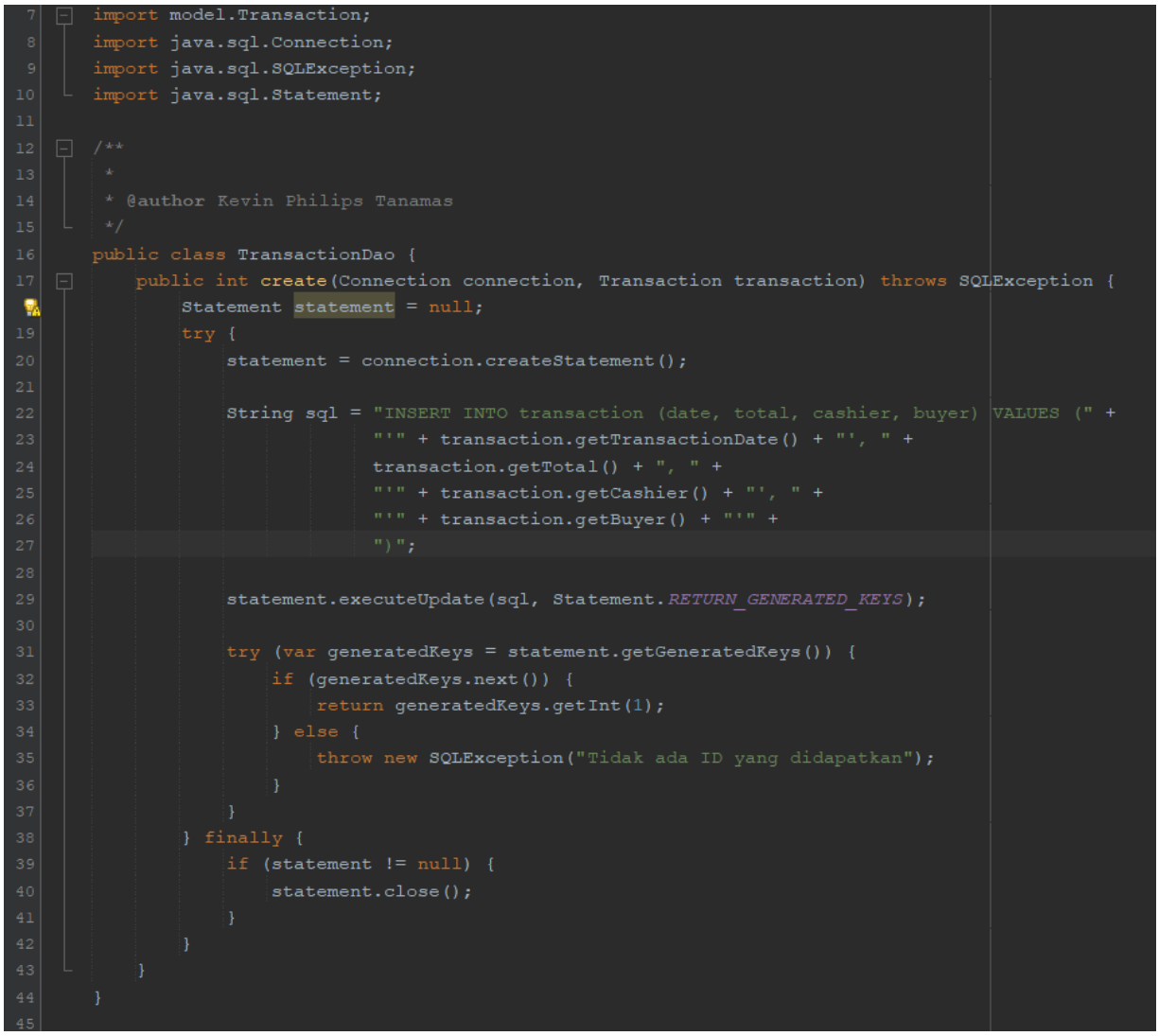
4. Silakan copas kode berikut ke dalam file **TransactionDetailDao.java** (kode ada di TransactionDetailDao.txt).

```java
package dao;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.List;
import model.TransactionDetail;
import java.sql.Statement;
import java.util.ArrayList;
/**
 *
 * @author Kevin Philips Tanamas
 */
public class TransactionDetailDao {
    public void create(Connection connection, List<TransactionDetail> transactionDetails) throws SQLException {
        Statement statement = null;
        List<Integer> generatedIds = new ArrayList<>();
        try {
            statement = connection.createStatement();

            for (TransactionDetail transactionDetail : transactionDetails) {
                String sql = "INSERT INTO transaction_detail (transaction_id, product_id, quantity_purchased, sub_total) VALUES (" +
                             transactionDetail.getTransactionId() + ", " +
                             transactionDetail.getProductId() + ", " +
                             transactionDetail.getQuantityPurchased() + ", " +
                             transactionDetail.getSubTotal() +
                             ")";

                statement.executeUpdate(sql, Statement.RETURN_GENERATED_KEYS);

                // Ambil id yang baru dibuat setelah tiap insert
                try (var generatedKeys = statement.getGeneratedKeys()) {
                    if (generatedKeys.next()) {
                        generatedIds.add(generatedKeys.getInt(1));
                    } else {
                        throw new SQLException("Creating transaction detail failed, no ID obtained.");
                    }
                }
```

```java
                    statement.executeUpdate(sql, Statement.RETURN_GENERATED_KEYS);

                    // Ambil id yang baru dibuat setelah tiap insert
                    try (var generatedKeys = statement.getGeneratedKeys()) {
                        if (generatedKeys.next()) {
                            generatedIds.add(generatedKeys.getInt(1));
                        } else {
                            throw new SQLException("Creating transaction detail failed, no ID obtained.");
                        }
                    }
                }
            } finally {
                if (statement != null) {
                    statement.close();
                }
            }
        }
    }

```

Kode ini bertujuan untuk menyisipkan data ke tabel transaction_detail. Dengan metode ini, beberapa data dapat dikirim dan dieksekusi sekaligus, sehingga mempercepat proses penyisipan data dibandingkan dengan menambahkannya satu per satu.

5. Silakan copas kode berikut ke dalam file **TransactionDao.java** (kode ada di TransactionDao.txt).



```java
import model.Transaction;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

/**
 *
 * @author Kevin Philips Tanamas
 */
public class TransactionDao {
    public int create(Connection connection, Transaction transaction) throws SQLException {
        Statement statement = null;
        try {
            statement = connection.createStatement();

            String sql = "INSERT INTO transaction (date, total, cashier, buyer) VALUES (" +
                         "'" + transaction.getTransactionDate() + "', " +
                         transaction.getTotal() + ", " +
                         "'" + transaction.getCashier() + "', " +
                         "'" + transaction.getBuyer() + "'" +
                         ")";

            statement.executeUpdate(sql, Statement.RETURN_GENERATED_KEYS);

            try (var generatedKeys = statement.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    return generatedKeys.getInt(1);
                } else {
                    throw new SQLException("Tidak ada ID yang didapatkan");
                }
            }
        } finally {
            if (statement != null) {
                statement.close();
            }
        }
    }
}
```

Kode di atas adalah kelas **TransactionDao** yang berfungsi untuk menyimpan data transaksi ke database menggunakan **SQL INSERT**. Method create menerima objek Transaction dan koneksi database, lalu membangun dan mengeksekusi query untuk memasukkan data transaksi ke tabel transaction, serta mengembalikan ID yang dihasilkan.

6. Buatlah sebuah package/folder bernama **service** yang akan digunakan untuk menangani logika bisnis terkait transaksi. **Service** ini bertugas **mengelola seluruh alur proses pembelian**, mulai dari **penambahan data** transaksi hingga **pengurangan stok produk**. Karena proses ini melibatkan banyak tabel, pemisahan logika bisnis di lapisan service diperlukan untuk menjaga agar DAO hanya berfungsi sebagai lapisan akses data untuk masing-masing tabel, sedangkan logika kompleks dikelola di lapisan service. (Kode ada di TransactionService.txt)

```java
package service;

import connection.DbConnection;
import dao.ProductDao;
import dao.TransactionDao;
import dao.TransactionDetailDao;
import java.math.BigDecimal;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import model.PurchaseItemDto;
import model.Product;
import model.Transaction;
import model.TransactionDetail;

public class TransactionService {
    private final TransactionDao transactionDao = new TransactionDao();
    private final TransactionDetailDao transactionDetailDao = new TransactionDetailDao();
    private final ProductDao productDao = new ProductDao();

    public String productPurchase(List<PurchaseItemDto> purchasedItems) {
        try {
            Timestamp transactionDatetime = new Timestamp(System.currentTimeMillis());

            DbConnection dbConnection = new DbConnection();
            dbConnection.makeConnection();
            DbConnection.CON.setAutoCommit(false);

            BigDecimal total = hitungTotalTransaksi(purchasedItems);
            System.out.println("Total Transaksi: " + total);

            Transaction transaction = new Transaction();
            transaction.setTransactionDate(transactionDatetime);
            transaction.setTotal(total);
            transaction.setBuyer("buyer-dummy");
            transaction.setCashier("cashier-dummy");

            int transactionId = transactionDao.create(DbConnection.CON, transaction);

            List<TransactionDetail> transactionDetails = new ArrayList<>();
            List<Product> products = new ArrayList<>();
```

```
            List<TransactionDetail> transactionDetails = new ArrayList<>();
            List<Product> products = new ArrayList<>();

            for (PurchaseItemDto item : purchasedItems) {
                TransactionDetail transactionDetail = new TransactionDetail();
                transactionDetail.setTransactionId(transactionId);
                transactionDetail.setProductId(item.getProduct().getId());
                transactionDetail.setQuantityPurchased(item.getQuantityPurchased());
                transactionDetail.setSubTotal(hitungSubTotalTransaksi(item));
                transactionDetails.add(transactionDetail);

                Product product = item.getProduct();
                product.setStock(product.getStock() - item.getQuantityPurchased());
                products.add(product);
            }
            transactionDetailDao.create(DbConnection.CON, transactionDetails);
            productDao.batchUpdate(DbConnection.CON, products);
            DbConnection.CON.commit();
            DbConnection.CON.close();
            return String.valueOf(total);
        } catch (SQLException ex) {
            ex.printStackTrace();
            try {
                if (DbConnection.CON != null) {
                    DbConnection.CON.rollback();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return null;
        }
    }

    public BigDecimal hitungTotalTransaksi(List<PurchaseItemDto> purchasedItems) {
        BigDecimal total = BigDecimal.ZERO;
        for (PurchaseItemDto item : purchasedItems) {
            total = total.add(hitungSubTotalTransaksi(item));
        }
        return total;
    }

    public BigDecimal hitungSubTotalTransaksi(PurchaseItemDto purchasedItem) {
        BigDecimal price = purchasedItem.getProduct().getPrice();
        BigDecimal quantity = new BigDecimal(purchasedItem.getQuantityPurchased());
        return price.multiply(quantity);
    }
}
```

Kode **TransactionService** ini menangani proses pembelian produk, mulai dari menghitung total transaksi, membuat data transaksi baru, mencatat detail pembelian tiap produk, hingga memperbarui stok produk di database. Dalam method productPurchase, data transaksi dan detail transaksi disimpan secara batch, dan perubahan hanya dikonfirmasi ke database setelah semua operasi berhasil, dengan menggunakan mekanisme commit dan rollback untuk menjaga

konsistensi data jika terjadi error. Selain itu, terdapat method tambahan untuk menghitung total dan subtotal transaksi berdasarkan daftar produk yang dibeli.

Ringkasan fungsi setiap method pada **TransactionService**:

Method	Fungsinya
productPurchase(List<PurchaseItemDto> purchasedItems)	Memproses transaksi pembelian produk, menyimpan data transaksi dan memperbarui stok.
hitungTotalTransaksi(List<PurchaseItemDto> purchasedItems)	Menghitung total transaksi dengan menjumlahkan semua subtotal item yang dibeli.
hitungSubTotalTransaksi(PurchaseItemDto purchasedItem)	Menghitung subtotal dari satu item (harga * jumlah).

7. Controller berfungsi sebagai penghubung antara **presentation layer** dan **logic layer**. Pada contoh ini, **TransactionController** menerima permintaan pembelian produk dan meneruskannya ke service layer (dalam hal ini TransactionService) untuk diproses lebih lanjut. Dengan demikian, controller bertindak sebagai penerima input dan pengarah alur logika bisnis yang terjadi pada service. (Kode ada di TransactionController.txt).

```java
package controller;

import java.util.List;
import model.PurchaseItemDto;
import service.TransactionService;

/**
 *
 * @author Kevin Philips Tanamas
 */
public class TransactionController {
    private final TransactionService transactionService = new TransactionService();

    public String create(List<PurchaseItemDto> items){
        return transactionService.productPurchase(items);
    }
}
```

8. Selamat teman-teman sudah menyelesaikan pengkodean backend!, silahkan melanjutkan ke pengkodean **view**.

--- Selamat Belajar, GBU ---