Order Management

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

COORDONATOR PROIECT:DORIN MOLDOVAN

STUDENT: Rus Mihai-Tudorel

GRUPA : 30221

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

CUPRINS

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

I. Specificație

- Cerința

Obiectivul acestei teme este implementarea unei aplicatii OrderManagement pentru procesarea comenzilor unor clienti pentru un depozit. Pentru stocarea si tinerea in evidenta a produselor, clientilor si a comenzilor s-a creat data de baze relationala warehouse MySQL. In ceea ce priveste structurarea codului java, acesta este impartit in mai multe packages (architectural layers) pentru o mai buna intelegere a codului si pentru lizibilitate. Aceste packages sunt :

• Main - contine clasa MainClass, clasa principala a programului

• connection - contine clasa ConnectionFactory, folosita pentru realizarea cu succes a conexiunii dintre proiectul java si baza de date MySQL

• model - contine entitatile Client, Product si My_order, avand aceleasi campuri ca si tabelele din baza de date si set-erele si get-erele necesare

• bussines - contine clasele ClientBussines, ProductBussines si My_orderBussines care explica logica comportamentelor fiecarei tabele

• data - contine clasele ClientData, ProductData si My_orderData care au implementate acele comportamente, creand queries si executandu-le

- Analiza problemei

Problema se împarte în mai mulți pași după cum ne și sugerează cerința. Atunci cand adaugam o noua comanda in baza de date, in acelasi timp trebuie actualizat si stocul produsul respectiv. Din acest motiv, s-a creat un trigger cu numele update_quantity, care scade din cantitatea curenta a produsului respectiv cantitatea ceruta de catre client. Acest trigger se "activeaza" pe tabela my_order atunci cand urmeaza sa se efectueze o operatie de INSERT pe aceasta tabela (BEFORE INSERT). Cu alte cuvinte, stocul se actualizeaza inainte de a se efectua propriu-zis inserarea unei noi comenzi in baza de date. A fost tratat cazul in care utilizator vrea sa introduca in baza de date acelasi client de mai multe ori. Aceasta eroare a fost evitata folosind sintaxa "INSERT IGNORE", care "sare peste" clientii pre-existenti in baza de date.

II. Proiectarea și implementarea

Aplicatia este impartita in mai multe "straturi", fiecare "strat" are un scop specific si apeleaza functii ale "strarturilor" inferioare celui appellant

. • "Stratul" MainClass - contine parser-ul de care se foloseste pentru a prelua informatiile comenzilor care trebuie sa fie executate cu succes de catre aplicatie (in acest "strat" se gaseste clasa principala a aplicatiei, clasa MainClass)

• "Stratul" connection - contine clasa "ConnectionFactory" care se ocupa de configurarea conexiunii dintre aceasta aplicatie java cu baza de date warehouse

• "Stratul" model - contine clasele Client, Product si My_order, clase mapate la tabelele client, product si my_order a bazei de date

• "Stratul" data (Data Access Object) - contine clasele Client Data, Product Data, My_order Data si Warehouse Data, clase care contin comenzi SQL de executat (queries) INSERT, DELETE, SELECT si legatura dintre acestea si baza de date Warehouse.

• "Stratul" bussines (Business Logic Layer) - contine clasele care incapsuleaza logica aplicatiei (clasele Client Bussines, Product Bussines si My_order Bussines).

Package Main

-clasa MainClass

• implementeaza parser-ul, citirea din fisier se realizeaza precum citirea din consola, deoarece in metoda public static void main(String[] args) s-a implementat aceasta functionalitate si numele fisierului este transmis prin args[0]

• folosind variabilele locale pdf_id si report_id tinem cont de numerele rapoartelor si bonurilor .pdf care vor trebui sa fie create (pentru a numi aceste fisiere de iesire in conformitate cu cerinta), daca acea comanda transmisa cere acest lucru

Package connection

-clasa ConnectionFactory

• prin campurile acestei clase, s-au setat path-ul catre baza de date warehouse, user-ul, parola de acces, url-ul bazei de date si driver-ul acesteia

• exista metoda de creere a unei conexiuni private createConnection(), de "preluat" o conexiune existenta private Connection getConnection(), de inchis o conexiune existenta private static void close(Connection connection), de inchis un Statement private static void close(Statement statement) si de inchis un ResultSet private static void close(ResultSet resultSet).

Package model

-clasa Client

• are campurile String name si String adress

• are sett-ere si get-ere

clasa Product

• are campurile String name, int quantity si double price

• are sett-ere si get-ere

• pentru get-erele campurilor int quantity si double price, am creat variabile temporare de tip Integer, respectiv Double pentru a transmite valorile acestea in format String, apeland functiile public String toString() predefinite de catre aceste tipuri.

-clasa My_order

• are campurile int id, String client_name, String product_name si int product_quantity

• are sett-ere si get-ere

• pentru get-erele campurilor int id si int product_quantity, am creat variabile temporare de tip Integer pentru a transmite valorile acestea in format String, apeland functiile public String toString() predefinite de catre acest tip.

Package bussines

-clasa Client Bussinees

• are campul privat data de tip ClientDATA

• are metoda public void insertClient(Client client), care apeleaza metoda din package-ul data de inserare a unui client in baza de date warehouse si in acelasi timp verifica daca aceasta s-a apelat cu success

• are metoda public void deleteClient(String name, String adress), care apeleaza metoda din package-ul data de stergere a unui client din baza de date warehouse si in acelasi timp verifica daca aceasta s-a apelat cu success

- clasa Product BUSSINES

• are campul privat data de tip ProductDATA

• are metoda public void insertProduct (Product product), care apeleaza metoda din package-ul data de inserare a unui produs in baza de date warehouse si in acelasi timp verifica daca aceasta s-a apelat cu succes

• are metoda public void deleteProduct (String name), care apeleaza metoda din package-ul data de stergere a unui produs din baza de date warehouse si in acelasi timp verifica daca aceasta s-a apelat cu succes

-clasa My_order BUSSINES

• are campul privat data de tip My_orderDATA

• are metoda public void insertMy_order (My_order my_order), care apeleaza metoda din package-ul data de inserare a unei comenzi in baza de date warehouse si in acelasi timp verifica daca aceasta s-a apelat cu succes

Package data

-clasa Client DATA

• are campurile private Connection connection, private Statement statement, private PreparedStatement preparedStatement, private ResultSet resultSet si private ResultSetMetaData resultSetMD prin intermediul carora se vor stabili conexiunile necesare, se vor crea query-urile specifice fiecarei metode si se vor efectua aceste query-uri, in acelasi timp verificandu-se daca acest lucru se intampla cu succes

• are constructor fara parametri si fara instructiuni

• are metoda public void insertClient(Client client), metoda care este apelata din packageul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de datele transmise prin parametru pentru a respecta sintaxa unei comenzi corecte de inserare a unui nou client in tabela client din baza de date warehouse

• are metoda public void deleteClient(String name, String adress), metoda care este apelata din package-ul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de datele transmise prin parametru pentru a respecta sintaxa unei comenzi corecte de stergere a unui client din tabela client din baza de date warehouse

• are metoda public PdfPTable selectClients(), metoda care este apelata din package-ul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de tabela din care dorim sa vedem toate informatiile existente pentru a respecta sintaxa unei comenzi corecte de selectare a tuturor clientilor in tabela client din baza de date warehouse

• se returneaza un obiect de tip PdfPTable in care sunt stocate toate aceste informatii rezultate in urma comenzii efectuate, sub forma unui tabel pentru a urma sa fie afisate intr-un fisier de tip .pdf .

-clasa Product DATA

• are campurile private Connection connection, private Statement statement, private PreparedStatement preparedStatement, private ResultSet resultSet si private ResultSetMetaData resultSetMD prin intermediul carora se vor stabili conexiunile necesare, se vor crea query-urile specifice fiecarei metode si se vor efectua aceste query-uri, in acelasi timp verificandu-se daca acest lucru se intampla cu success

• are constructor fara parametri si fara instructiuni

• are metoda public void insertProduct(Product product), metoda care este apelata din package-ul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de datele transmise prin parametru pentru a respecta sintaxa unei comenzi corecte de inserare a unui nou produs in tabela product din baza de date warehouse

• are metoda public void deleteProduct (String name, String adress), metoda care este apelata din package-ul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de datele transmise prin parametru pentru a respecta sintaxa unei comenzi corecte de stergere a unui client din tabela product din baza de date warehouse

• are metoda public PdfPTable selectProducts(), metoda care este apelata din package-ul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de tabela din care dorim sa vedem toate informatiile existente pentru a respecta sintaxa unei comenzi corecte de selectare a tuturor clientilor in tabela product din baza de date warehouse

• se returneaza un obiect de tip PdfPTable in care sunt stocate toate aceste informatii rezultate in urma comenzii efectuate, sub forma unui tabel pentru a urma sa fie afisate intr-un fisier de tip .pdf

-clasa My_order DATA

• are campurile private Connection connection, private Statement statement, private PreparedStatement preparedStatement, private ResultSet resultSet si private ResultSetMetaData resultSetMD prin intermediul carora se vor stabili conexiunile necesare, se vor crea query-urile specifice fiecarei metode si se vor efectua aceste query-uri, in acelasi timp verificandu-se daca acest lucru se intampla cu succes

• are constructor fara parametri si fara instructiuni

• are metoda public void insertMy_order(My_order my_order), metoda care este apelata din package-ul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de datele transmise prin parametru pentru a respecta sintaxa unei comenzi corecte de inserare a unui nou client in tabela my_order din baza de date warehouse

• are metoda public PdfPTable selectMy_order (), metoda care este apelata din packageul Bussines; in aceasta metoda se construieste efectiv query-ul in functie de tabela din care dorim sa vedem toate informatiile existente pentru a respecta sintaxa unei comenzi corecte de selectare a tuturor clientilor in tabela my_order din baza de date warehouse

• se returneaza un obiect de tip PdfPTable in care sunt stocate toate aceste informatii rezultate in urma comenzii efectuate, sub forma unui tabel pentru a urma sa fie afisate intr-un fisier de tip .pdf .

-clasa Warehouse DATA

• are campurile private Connection connection si private Statement statement prin intermediul carora se vor stabili conexiunile necesare, se vor crea query-urile specifice fiecarei metode si se vor efectua aceste query-uri, in acelasi timp verificandu-se daca acest lucru se intampla cu succes

• are constructor fara parametri si fara instructiuni

• are metoda public void createDatabase() care executa query-ul necesar pentru a crea o baza de date daca aceasta nu exista deja, iar mai apoi verifica daca acest lucru a fost efectuat cu success.

III. Lista de clase

Pentru a respecta principiile fundamentale ale OOP, programul conține mai multe clase:

- Client
- Order
- Product
- Client Bussines
- Order Bussines
- Product Bussines
- Client Data
- Order Data
- Product Data
- Warehouse Data
- Connection Factory
- MainClass

IV. Justificarea soluției alese

Am optat pentru această soluție de implementare deoarece mi s-a părut o modalitate atât ușoară, cât și eficientă pentru implementarea aplicatiei. Folosirea stărilor intermediare prin care trece programul ne ajută atât la simplificarea soluției, cât și la implementarea eficientă a aplicatiei.

V. Utilizarea și rezultatele

Rezultatele efectuarii tuturor comenzilor din fisierul de intrare vor fi transmise prin intermediul unor obiecte de tip PdfPTable (obiecte existente datorita incorporarii dependentei fata de libraria externa itextpdf) si asezate in tabele in asa fel incat informatiile sa fie usor de interpretat. Comanda transmisa de catre utilizator "Report client" , "Report product" sau "Report order" va rezulta in crearea unui fisier .pdf cu identificatorul reportx, unde x este inlocuit cu valoare varibailei locale pdf_id din metoda public static void main(String[] args) care tine evidenta a numarului de ordine al fisierului .pdf de tip raport creat pana in momentul respectiv. Comanda transmisa de catre utilizator "Order: ..." va rezulta in in crearea unui fisier .pdf cu identificatorul orderx (pe langa dubla functionalitate de a insera comanda precizata in baza de date warehouse), unde x este inlocuit cu valoare variabilei locale order_id din metoda public static void main(String[] args) care tine evidenta a numarului de ordine al fisierului .pdf de tip comanda creat pana la momentul respectiv.

VI. Posibilități de dezvoltare ulterioară

Consider ca, o dezvoltare ulterioara ar putea fii aceea de a adauga si o simulare in timp real a cozilor,nu in timp masurat in secunde ci, simularea cat mai realistica, cu aparitia clientilor la aceleasi momente de timp,cu intercalarea timpului,practic o aplicatie de simulare apropiata cu realitatea.

VII. Concluzii

Concluziile includ faptul că o planificare bine gândită a claselor ușurează lucrurile și economisește timp. Odată cu creșterea dificultății în proiecte, trebuie abordată problema cu o perspectivă inteligentă și nu doar să sari imediat la scrierea codului. Astfel, începând cu diagramele UML mi-am dat timp să reflectez pentru abordarea adecvată a implementării programului dorit.

Pentru a le include in .jar aplicatiei, a fost necesara exportarea de tip Runnable JAR, care a oferit aceasta optiune. In urma realizarii acesti proiect, am invatat cum sa stabilesc o conexiune intre un proiect java realizat in IDE Eclipse si o baza de date relationala MySQL prin intermediul pattern-ului Singleton, structurarea arhitecturala pe layer-e a codului aplicatiei si interactionarea dintre ele, executarea query-urilor din interiorul aplicatiei. De asemenea, miam reamintit de anumite comenzi folosite pentru crearea de tabele si trigger-e in MySQL.

Îmbunătățirile care ar putea fi realizate în cadrul proiectului includ o ușoară modificare, astfel încât calculatorul acceptă intrări float, care ar putea fi ușor realizate prin modificarea matcherelor din funcție care transformă intrarea utilizatorului șirului în obiectul respectiv. Într-o anumită măsură...

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

VIII. Bibliografie

http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

http://www.javacodegeeks.com/2013/01/java-thread-pool-example-usingexecutors-and-threadpoolexecutor.html

https://stackoverflow.com/questions/19154202/data-access-object-dao-in-javahttp://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3.pdf