MANAGER DE COMENZI
DOCUMENTATIA TEMEI

Student: Bivolaru Alexandru
Grupa: 30225
Profesor laborator: Moldovan Dorin

CUPRINS

Cuvant inainte!

- Conexiunea MySQL Workbench a fost efectuata cu ajutorul xampp. Conexiunea are **user: root si parola “” (nu are)**
- Pentru a rula fisierul .jar, fisierul comenzi.txt trebuie sa existe, iar jar-urile pentru itext si jdbc trebuie sa se regaseasca in acelasi fisier cu jar-ul principal.
- In folder se regasesc 2 SQL dump-uri, unul pentru crearea bazei de date fara popularea tabelelor, iar celalalt este cu popularea tabelelor.

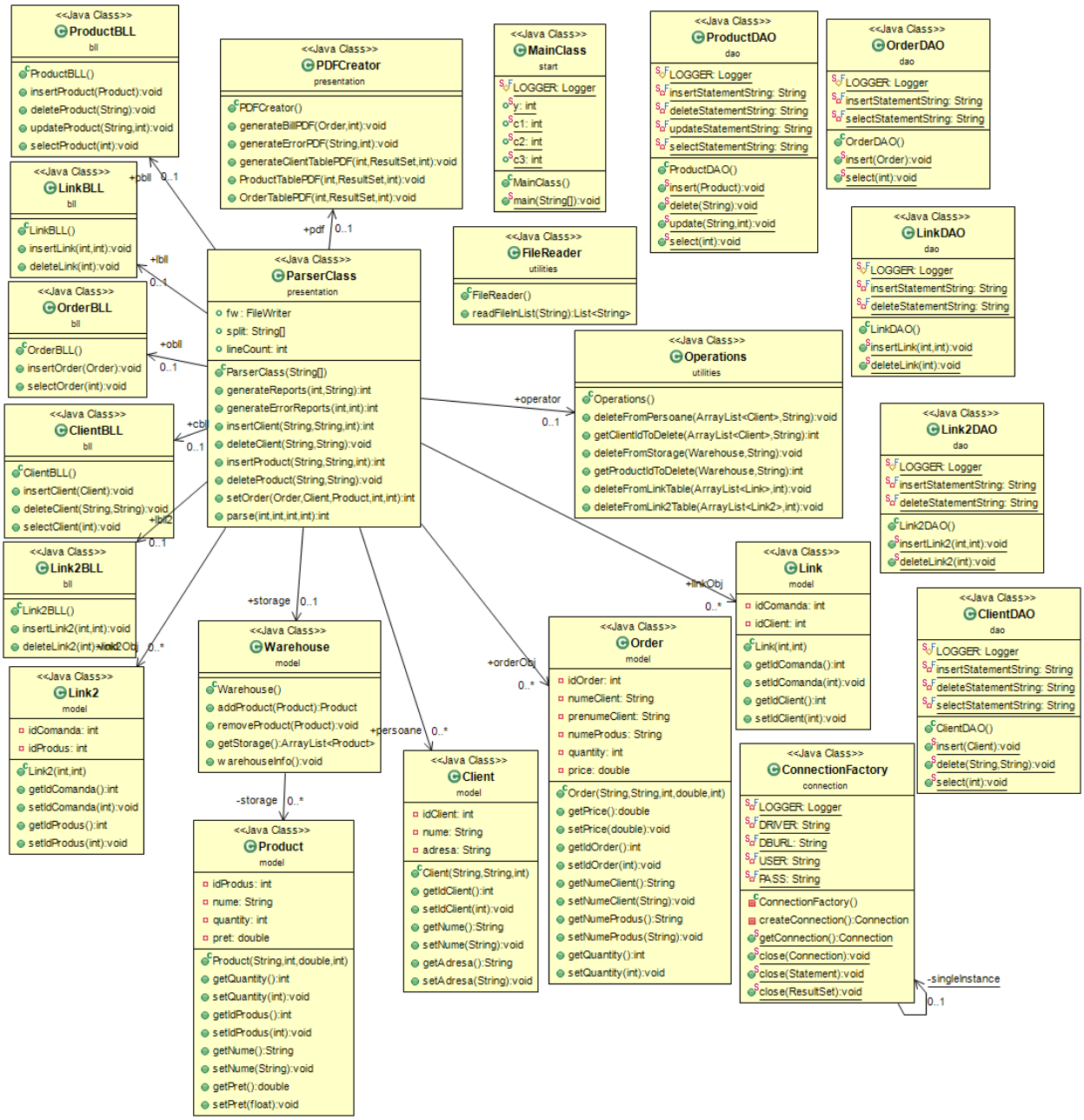
1. Obiectivul temei (Capitolul 1)

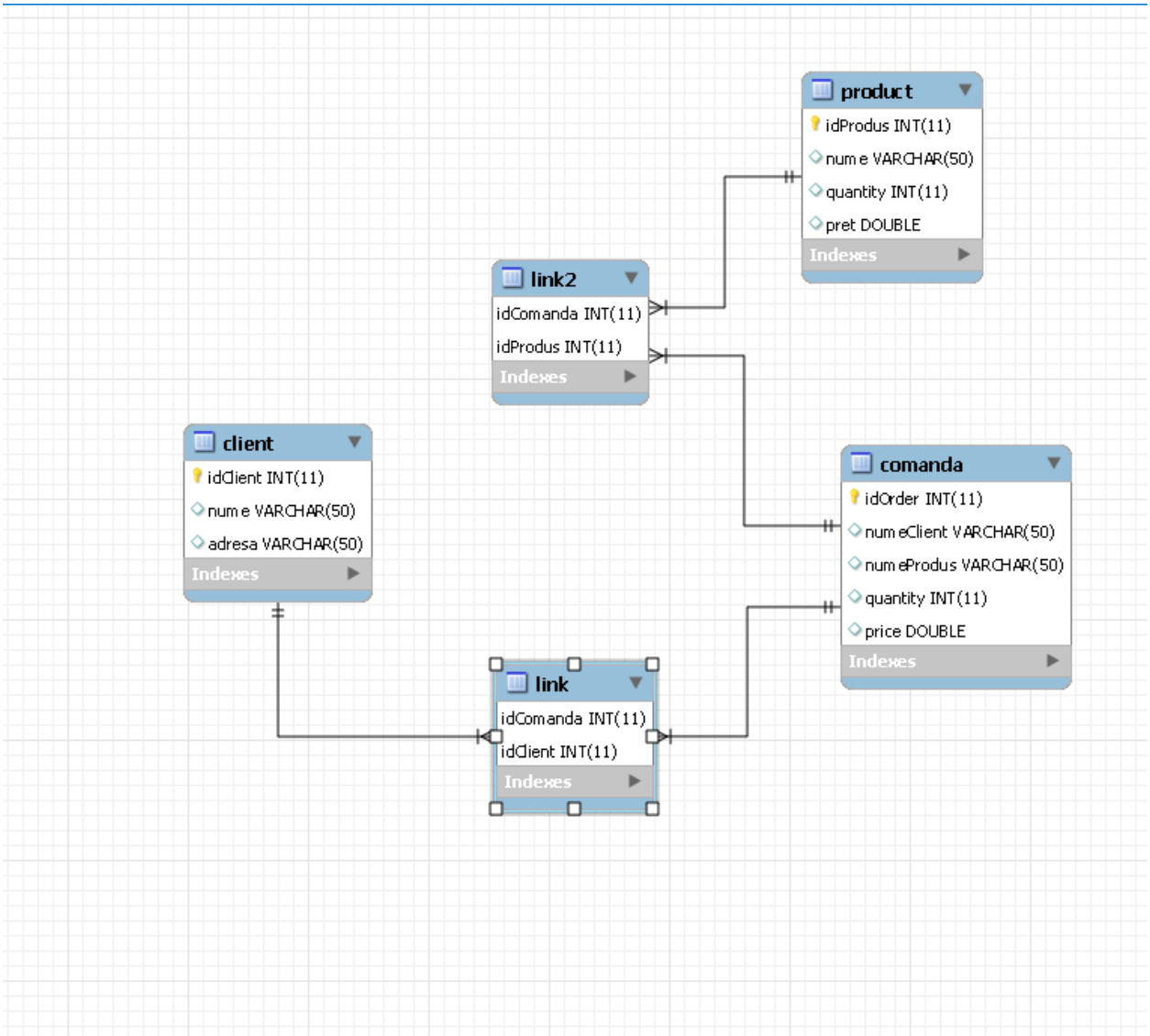
Obiectivul temei presupune implementarea unui sistem de gestiune al comenzilor unui depozit. Aplicatia simuleaza efectuarea mai multor operatii asupra unei baze de date (inserare client / produse, stergere client / produse, plasare de comenzi si generare de rapoarte cu privire la starea actuala a bazei de date). Aplicatia trebuie sa tina evidenta tuturor clientilor care au fost inserati / stersi si / sau au efectuat comenzi si sa raspunda prompt la operatiile la care este solicitata.

2. Analiza problemei (Capitolul 2)

Pasii de dezvoltare a problemei reprezinta lista de etape care trebuie urmata pentru a indeplini obiectivul temei. Pasii alesi pentru rezolvarea acestui proiect sunt, in ordine, urmatorii:

- **Crearea de scenarii in lumea reala.** Acest obiectiv presupune relationarea cu utilizatorul aplicatiei (in acest caz, managerul de depozit). Acesta trebuie sa gestioneze toti clientii care doresc sa comande ceva, sa tina evidenta produselor din depozitul sau si sa semnaleze eventualele imposibilitati de plasare a comenzilor. Pentru a face posibil acest lucru vom folosi un limbaj de modelare UML pentru a gestiona toate scenariile prezentate anterior.
- **Alegerea tipurilor / structurilor de date.** In continuare, in partea aceasta a proiectului, trebuie sa ne decidem asupra caror structure de date vor fi folosite pentru indeplinirea obiectivului principal al temei.
- **Crearea claselor / dezvoltarea algoritmilor.** Fara dar si poate, acest pas este cel mai complex din cadrul acestui proiect. Aici trebuie sa decidem impartirea proiectului in clase, apoi trebuie sa ne decidem asupra metodelor pe care le vom implementa, metode care trebuie sa asigure buna functionare a intregului proiect.


ProductBLL
bll
ProductBLL()
insertProduct(Product):void
deleteProduct(String):void
updateProduct(String,int):void
selectProduct(int):void
LinkBLL
bll
LinkBLL()
insertLink(int,int):void
deleteLink(int):void
OrderBLL
bll
OrderBLL()
insertOrder(Order):void
selectOrder(int):void
ClientBLL
bll
ClientBLL()
insertClient(Client):void
deleteClient(String,String):void
selectClient(int):void
Link2BLL
bll
Link2BLL()
insertLink2(int,int):void
deleteLink2(int):void
Link2
model
idComanda: int
idProdus: int
Link2(int,int)
getIdComanda():int
setIdComanda(int):void
getIdProdus():int
setIdProdus(int):void
PDFCreator
presentation
PDFCreator()
generateBillPDF(Order,int):void
generateErrorPDF(String,int):void
generateClientTablePDF(int,ResultSet,int):void
ProductTablePDF(int,ResultSet,int):void
OrderTablePDF(int,ResultSet,int):void
MainClass
start
LOGGER: Logger
y: int
c1: int
c2: int
c3: int
MainClass()
main(String[]):void
ProductDAO
dao
LOGGER: Logger
insertStatementString: String
deleteStatementString: String
updateStatementString: String
selectStatementString: String
ProductDAO()
insert(Product):void
delete(String):void
update(String,int):void
select(int):void
OrderDAO
dao
LOGGER: Logger
insertStatementString: String
selectStatementString: String
OrderDAO()
insert(Order):void
select(int):void
LinkDAO
dao
LOGGER: Logger
insertStatementString: String
deleteStatementString: String
LinkDAO()
insertLink(int,int):void
deleteLink(int):void
ParserClass
presentation
fw : FileWriter
split: String[]
lineCount: int
ParserClass(String[])
generateReports(int,String):int
generateErrorReports(int,int):int
insertClient(String,String,int):int
deleteClient(String,String):void
insertProduct(String,String,int):int
deleteProduct(String,String):void
setOrder(Order,Client,Product,int,int):int
parse(int,int,int,int):int
FileReader
utilities
FileReader()
readFileInList(String):List<String>
Operations
utilities
Operations()
deleteFromPersoane(ArrayList<Client>,String):void
getClientIdToDelete(ArrayList<Client>,String):int
deleteFromStorage(Warehouse,String):void
getProductIdToDelete(Warehouse,String):int
deleteFromLinkTable(ArrayList<Link>,int):void
deleteFromLink2Table(ArrayList<Link2>,int):void
Link2DAO
dao
LOGGER: Logger
insertStatementString: String
deleteStatementString: String
Link2DAO()
insertLink2(int,int):void
deleteLink2(int):void
Link
model
idComanda: int
idClient: int
Link(int,int)
getIdComanda():int
setIdComanda(int):void
getIdClient():int
setIdClient(int):void
ClientDAO
dao
LOGGER: Logger
insertStatementString: String
deleteStatementString: String
selectStatementString: String
ClientDAO()
insert(Client):void
delete(String,String):void
select(int):void
Warehouse
model
Warehouse()
addProduct(Product):Product
removeProduct(Product):void
getStorage():ArrayList<Product>
warehouseInfo():void
Order
model
idOrder: int
numeClient: String
prenumeClient: String
numeProdus: String
quantity: int
price: double
Order(String,String,int,double,int)
getPrice():double
setPrice(double):void
getIdOrder():int
setIdOrder(int):void
getNumeClient():String
setNumeClient(String):void
getNumeProdus():String
setNumeProdus(String):void
getQuantity():int
setQuantity(int):void
Client
model
idClient: int
nume: String
adresa: String
Client(String,String,int)
getIdClient():int
setIdClient(int):void
getNume():String
setNume(String):void
getAdresa():String
setAdresa(String):void
ConnectionFactory
connection
LOGGER: Logger
DRIVER: String
DBURL: String
USER: String
PASS: String
ConnectionFactory()
createConnection():Connection
getConnection():Connection
close(Connection):void
close(Statement):void
close(ResultSet):void
Product
model
idProdus: int
nume: String
quantity: int
pret: double
Product(String,int,double,int)
getQuantity():int
setQuantity(int):void
getIdProdus():int
setIdProdus(int):void
getNume():String
setNume(String):void
getPret():double
setPret(float):void
+obll
+lbll
+obll
+cbll
+lbl2
+pdf
+operator
+storage
+linkObj
+orderObj
+persoane
-storage
-singleInstance
0..1
0..*


product
idProdus INT(11)
nume VARCHAR(50)
quantity INT(11)
pret DOUBLE
Indexes
link2
idComanda INT(11)
idProdus INT(11)
Indexes
client
idClient INT(11)
nume VARCHAR(50)
adresa VARCHAR(50)
Indexes
comanda
idOrder INT(11)
numeClient VARCHAR(50)
numeProdus VARCHAR(50)
quantity INT(11)
price DOUBLE
Indexes
link
idComanda INT(11)
idClient INT(11)
Indexes

3. Proiectare(Decizii de proiectare, diagrame UML, structure de date, proiectare clase, algoritmi) (Capitolul 3)

3.1 Decizii de proiectare.

Utilizatorului ii este cerut sa introduca un set de comenzi intr-un fisier de intrare (de obicei numit "comenzi.txt"), comenzi care, simuleaza activitatea zilnica a unui depozit. Aceste comenzi (cu forma lor standard) sunt urmatoarele:

- Insert client: Nume, Adresa
- Delete client: Nume, Adresa
- Insert produs: Nume, Cantitate, Pret bucata
- Delete produs: Nume
- Order: Nume, Nume produs, cantitate
- Report client/order/product

Aceste comenzi trebuie sa fie "traduse" din string in comanda de catre noi, folosind un parser (pe care il vom implementa ulterior). Odata ce o comanda a fost tradusa cu succes, aceasta va fi apelata de metoda corespunzatoare (din nou, pe care o vom implementa ulterior).

3.2 Structuri de date

Pentru a asigura buna functionare a aplicatiei, suntem nevoiti sa folosim structure de date eficiente care sa ne permita controlul fluxului de date care se petrece intre tabelele bazei de date si codul nostru sursa. Din aceasta cauza, structurile de date pe care le vom folosi preponderent sunt **ArrayList,** deoarece ne ofera o buna flexibilitate asupra controlului datelor in interiorul listelor, **Document,** tip de date cu ajutorul careia generam rapoartele in format PDF, **PdfPCell si PdfPTable,** cu ajutorul carora cream si administram tabelele din interiorul documentelor de tip pdf. Alte doua structure de date importante pe care le-am folosit si functioneaza concurrent sunt **FileWriter si List.** Aceste doua tipuri de date ne ajuta la citirea si stocarea datelor dintr-un fisier intr-o lista de elemente de tip String.

3.3 Proiectarea claselor.

Clasa ConnectionFactory

Clasa ConnectionFactory se ocupa de realizarea conexiunii cu baza de date prin intermediul unui DriverManager din pachetul java.sql. Aceasta clasa foloseste modelul Singleton pentru a restrictiona instantierea unei clase in mai multe locuri, deoarece avem nevoie de aceasta doar o singura data. Aceasta cuprinde metode pentru realizarea si returnarea conexiunii pentru a putea fi data in folosinta in metodele claselor ce urmeaza in cursul acestui capitol.

Clasa Client (pachetul model)

In cadrul claselor din pachetul model, se "emuleaza" tabelele din cadrul bazei de date. Clasa client contine ca si attribute idClient, nume si adresa, iar ca si metode avem settere si gettere pentru atributele specificate.

Clasa Product (pachetul model)

In cadrul acestei clase cream modelul pentru un obiect de tip produs, obiect care are ca si attribute idProdus, nume, quantity si pret. Metodele din aceasta clasa sunt, deja bine cunoscutele, settere si gettere pentru atributele mentionate mai sus.

Clasa Order (pachetul model)

Aceasta clasa simuleaza obiectele specifice tabelului Comanda. Ca si attribute, in cadrul acestei clase regasim idOrder, numeClient, numeProdus, quantity (cantitatea totala a comenzii) si price (pretul total al comenzii).

Clasa Link (pachetul model)

Aceasta clasa reprezinta oglinda unui obiect de legatura Client-Comanda, precum in tabelul Link din cadrul bazei de date. Ca si attribute regasim idComanda si idClient (id-ul clientului care a plasat comanda cu id-ul idComanda).

Clasa Link2 (pachetul model)

Aceasta clasa este identica din punct de vedere al functionalitatii cu clasa Link, doar ca reprezinta legatura Produs-Comanda. Atributul idProdus reprezinta id-ul produsului specificat in comanda cu id-ul idComanda.

Clasa Warehouse (pachetul model)

Aceasta clasa contine o lista "storage" in care stocam produsele. Am ales sa ii fac o clasa noua pentru a fi mai sugestiva ideea de Warehouse. Metodele prezente in aceasta clasa sunt cele de adaugare a produsului (si eventualele actualizari asupra cantitatii, dar si de stergere a acestora din cadrul depozitului).

Clasa ClientDAO (pachetul dao)

In cadrul acestei clase cream o conexiune cu baza de date si declaram 3 string-uri, care, reprezinta statement-urile unor comenzi mysql (select, delete si insert), prelucrate astfel incat sa stergem atributele de care avem noi nevoie. Tot in cadrul acestei comenzi sunt declarate si metodele care apeleaza comenzile mysql (insert, delete si select) pentru a actualiza tabelele bazei de date.

Clasa ProductDAO (pachetul dao)

Din punct de vedere functional, aceasta clasa este identica cu ClientDAO. In plus fata de ClientDAO, aceasta clasa contine si un statement de tip UPDATE, folosit pentru actualizarea cantitatii in cazul in care o comanda a fost plasata cu success. (Actualizare care consta in reducerea cantitatii respectivului produs din depozit).

Clasa OrderDAO (pachetul dao)

Aceasta clasa incorporeaza metodele de preclucrare ale tabelei comanda din cadrul bazei de date. Aceste metode sunt Insert si Select (Insert pentru inserarea unei noi comenzi in tabela Comanda, iar Select pentru afisarea informatiilor despre comanda respective).

Clasele LinkDAO si Link2DAO (pachetul dao)

In cadrul acestor doua clase pereche, se realizeaza operatii asupra tabelelor Link si Link2 din cadrul bazei de date. Aceste doua tabele reprezinta legatura Client – Comanda si Produs – Comanda, realizand astfel legatura "Client – Comanda – Produs", legatura creata cu scopul de a pastra proprietatea de baza de date relationala.

Pachetul bll (impreuna cu clasele associate)

In cadrul claselor din acest pachet, se defines metodele care apeleaza metodele din clasele pachetului dao, adica metodele care prelucreaza tabelele din baza de date.

Pachetul presentation

In cadrul acestui pachet se defines doua clase, ParserClass si PDFCreator. Aceste clase ajuta la "prezentarea" rezultatului aplicatiie catre publicul exterior (in cazul nostrum, utilizatorul aplicatiei). In cadrul clasei ParserClass, transformam liniile fisierului de comenzi in comenzi specific, iar in cadrul clasei PDFCreator implementam metode care ne ajuta la generarea diferitelor rapoarte de tip .pdf in functie de comanda specificata.

Pachetul utilities

In cadrul acestui pachet avem doua clase, FileReader si Operations. Aceste clase ne ajuta la structurarea mai buna a codului. FileReader este folosita pentru citirea si stocarea continutului fisierului intr-o lista de Stringuri. In cadrul clasei Operations avem operatii care se aplica asupra listelor de legatura intre tabelele din baza de dat si codul sursa (persoane, storage, linkobj si link2obj, despre care voi vorbi mai in detaliu la descrierea metodelor).

Pachetul start

Pachetul start contina clasa MainClass, unde dam sa ruleze programul. Aceasta contine 5 variabile statice. Logger reprezinta conexiunea cu baza de date, y reprezinta contorul pentru denumirea fisierelor (explicatie la metode), iar c1,c2 si c3 reprezinta contoarele pentru id-urile clientilor, produselor si comenzilor (cu ajutorul acestora eliminam riscul de a primi erori si exceptii de tipul SQLException referitoare la Foreign Key update).

Dezvoltarea algoritmilor (Capitolul 4)

In cadrul acestui capitol voi prezenta ideile de dezvoltare a algoritmilor cei mai folositori din cadrul acestui proiect. Daca mai multi algoritmi sunt asemanatori sau aproape identici dpdv al functionalitatii voi descrie doar principiul unuia dintre ei.

Primele metode despre care voi discuta sunt cele din cadrul **clasei ProductDAO** (pachet dao). Toate clasele din pachetul dao contin metode cu functionalitati identice, doar ca aplicate asupra diferitor tabele din cadrul bazei de date.

Metoda **insert** preia ca si argument un Produs p, produs care urmeaza sa fie introdus in cadrul tabelei Product. Metoda incepe prin preluarea conexiunii cu baza de date si pregatirea Statementului. Dupa ce am pregatit statement-ul, setam argumentele exact in ordinea ceruta in statement. Dupa ce am terminat de facut si acest lucru, executam statement-ul. Dupa ce statement-ul s-a executat, inchidem toate conexiunile. (Procesul de inchidere al conexiunilor se repeata la fiecare metoda care actioneaza asupra tabelelor din baza de date).

In continuare, toate metodele din cadrul acestui pachet au funcitonalitati de insert, delete, update sau select, singurele diferente fiind tabelele asupra carora actioneaza si parametri necesari efectuarii operatiilor.

Urmatoarele metode sunt cele din cadrul clasei **Operations (pachetul utilities).** Aceste metode opereaza asupra unor liste auxiliare, folosite pentru asigurarea proprietatii de baza de date relationala. De ce am ales sa folosim liste auxiliare ? Raspunsul este pentru ca atunci cand dorim sa efectuam o comanda, iar comanda contine un foreign key la client, implicit ar fi o relatie Many to Many. Aceasta relatie trebuie rezolvata prin tabele auxiliare. In cadrul acestui proiect se regasesc 2 astfel de tabele (Link si Link2), tabele care realizeaza relatie Client – Comanda – Produs. Daca nu am realiza aceste tabele, baza de date nu ar mai fi relationala, iar legaturile nu ar fi rezolvate. Daca avem aceste tabele si dorim sa stergem un client de ex., trebuie s ail stergem si din tabela auxiliara. Daca nu o facem se va arunca o exceptie referitoare la Foreign Key. Toate metodele din cadrul acestei clase sunt metode de insert / stergere din arraylist. Stergerea este realziata cu ajutorul unui obiect de tipul Iterator pentru a nu primi exceptie de tipul ConcurrentModification, exceptii primite atunci cand incercam sa folosim metoda default remove() din cadrul unui arraylist in timp ce suntem intr-o structura repetitiva.

Metodele ce vor fi prezentate in continuare sunt cele din **ParserClass.** In cadrul constructorului acestei clase impartim continutului fisierului in substructuri cu ajutorul unui spliter, substructuri pe care le salvam intr-un array de string-uri. Pe acest array de substructuri vom face parsarea comenzilor. Tot in cadrul acestui constructor vom initializa obiecte si liste pentru a ne usura munca in cadrul metodelor.

Metoda generateReports verifica string-ul curent pentru a sti ce raport sa genereze. Cand se intra pe una dintre ramuri se genereaza raportul specific acelei ramuri cu ajutorul metodei de select(metoda care contine metoda de creare pdf din cadrul PDFCreator). Odata generat raportul, variabila generated se va face 1 si va fi returnata, semn ca un raport a fost generat si se poate continua iteratia pe sirul de mai sus.

Metoda generateErrorReports este asemanatoare cu cea prezentata mai sus, doar ca aceasta se apeleaza doar atunci cand se plaseaza o comanda. Se verifica codul de eroare dat de acea comanda in cazul in care nu a fost plasata cu success si in functie de acel cod ne dam seama ce tip de eroare sa generam in fisierul pdf.

Metoda insertClient verifica substring-ul actual ca sa verifice daca intr-adevar este “Insert client”. Daca da, salveaza ce este dupa intr-un subsir de string-uri, care la randul sau va fi spart in substring-uri care reprezinta in ordine, numele si adresa clientului. Dupa ce acestea au fost separate se va face un client nou cu aceste date, client ce va fi adaugat in tabelul din baza de date si in lista de persoane generate. Odata ce generarea a avut loc se va returna valoarea inserted (1 la acel moment), semn ca, clientul a fost inserat.

Metoda deleteClient este identica cu insertClient, singura diferenta fiind apelarea metodelor deleteFromPersoane si deleteFromLinkTable. Trebuie sa il stergem din tabelul de legatura pentru a mentine proprietatea de baza de date relationala, nu putem sterge un client fara sa stergem si id-ul sau din tabelul cu legatura client – comanda.

Metodele insertProduct si deleteProduct dpdv functional sunt la fel, doar ca atunci cand inseram un produs mai facem o verificare in storage ca nu cumva produsul sa existe deja. Daca exista, ii vom actualiza cantitatea.

Metoda parse este o metoda mai lunga in care se apeleaza toate metodele mentionate mai sus si in plus, se verifica daca nu cumva dorim sa facem si o comanda. Daca dorim sa facem comanda, verificam ca nu cumva cantitatea ceruta sa fie mai mare decat cea deja existenta in depozit. Daca totul e in regula setam comanda. Totodata se verifica si alte cazuri de eroare (daca persoana nu exista / daca produsul nu exista) si se actualizeaza codul de eroare necesar generarii rapoartelor de eroare.

In cadrul clasei **PDFCreator** se genereaza pdf-uri cu tabele pentru diferite rapoate(generare de erori, selecturi de client / order / pouse, generaide chitante etc.). In cadrul acestor metode se foloseste variabila statica y pentru a contoriza rapoatele. Exista posibilitatea ca unele sa se suprapuna, de aici vine si necesitatea unei variabila care ajunge sa fie concatenata la numele pdf-ului actual.

Capitolul 5. Concluzii si posibilitati de dezvoltaltare ulterioara. Bibliografie

Din punctul meu de vedere aceasta tema m-a ajutat sa inteleg utilizarea Layered Architecture, cum sa creez pdf-uri si cum sa organizez codul cat mai bine dpdv structural.

Ca si posibilitati de dezvoltare ulterioara, eu consider ca implementarea de procedure stocate direct in mysql este o alternative pentru a nu aglomera codul java.

Bibliografie

https://www.baeldung.com/javadoc
http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf
https://www.baeldung.com/java-pdf-creation
https://dzone.com/articles/layers-standard-enterprise
https://support.plesk.com/hc/en-us/articles/115000818754-How-to-create-a-database-dump-using-PHPMyAdmin-in-Plesk-