Documentaţie
Calitatea Sistemelor Software

Andreea Prodan
Dan Gabriel Cazacu
Ilinca-Magdalena Stumbea

Cuprins

Introducere

În cadrul acestui document este prezentată o aplicaţie ce poate fi utilizată pentru a realiza managementul bazelor de date relaţionale. Aplicaţia este scrisă în limbajul de programare Java şi nu conţine dependinţe suplimentare.

Implementare - Phase 1

Pachete

În cadrul aplicaţiei, pachetele au fost structurate în aşa manieră încât să reflecte modulele aplicaţiei.

Structura

1. **exception**
   Împachetează toate excepţiile definite în aplicaţie.
2. **graphicalInterface**
   Acest pachet conţine clasele folosite în modelarea interfeţei grafice a aplicaţiei. Sunt capturate atât elementele grafice, cât şi evenimentele ce au loc în urma interacţiunii utilizatorului cu acestea.
3. **model**
   Acesta este pachetul ce conţine nucleul aplicaţiei, reprezentând atât structura logică a bazei de date, cât şi operaţiile asupra entităţilor. În acest pachet se mai poate găsi şi funcţionalitatea import/export CSV.
4. **persistance**
   Aici este definit stratul de persistenţă al aplicaţiei folosit pentru stocarea datelor între sesiunile de lucru.
5. **service**
   Deşi definit generic, acest pachet conţine în momentul de faţă serviciul de parsare al comenzilor oferite de utilizator de la linia de comanda.
6. **util**
   Conţine clase ajutătoare, precum gruparea de constante şi crearea de date de test.

Modulele aplicaţiei

Nucleu

Acesta este modulul principal al aplicaţiei. Este un serviciu ce oferă functionalităţi similare unui sistem de management de baze de date relaţionale, precum: creare de baze de date, creare de tabele, inserare de valori în tabele etc. Accesul asupra datelor se face prin intermediul obiectelor Java.

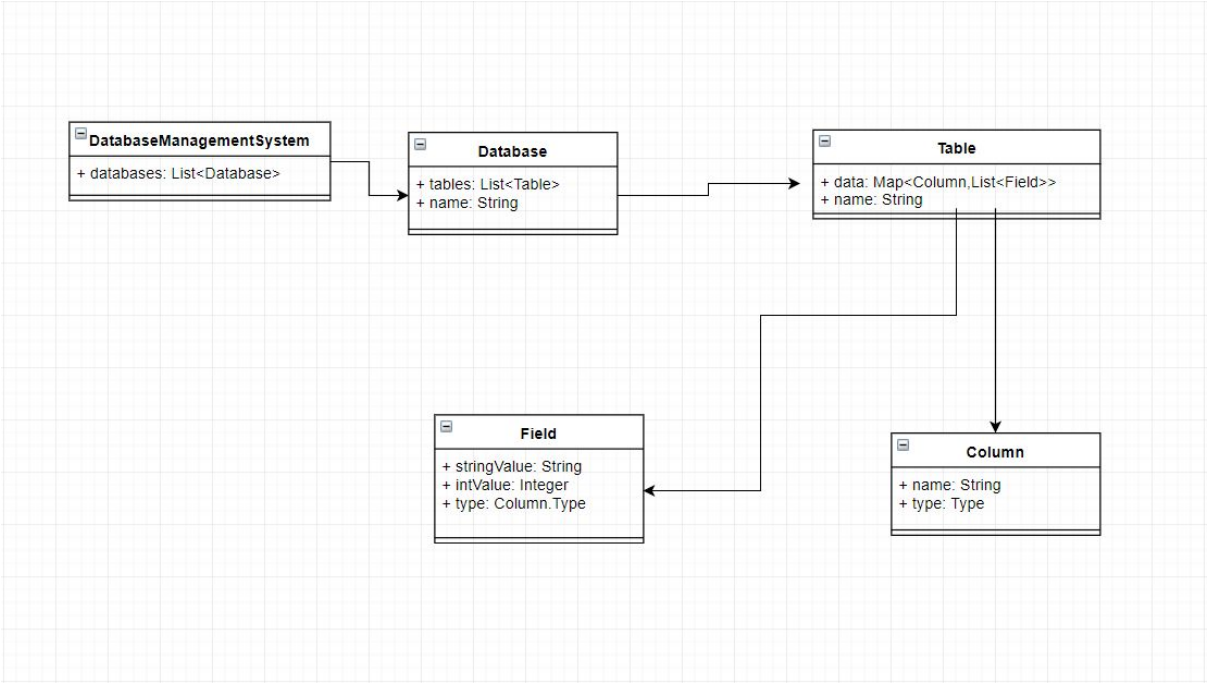

Fig. 1 Structura internă a modelului aplicaţiei

Clase pentru interacţiunea cu DBMS:

1. DatabaseManagementSystem (Singleton)

   Operaţii asupra bazelor de date:

   1. `boolean exists(String name)` - pentru a verifica existenţa unei baze de date;
   2. `Database createDatabase(String name)` - creează o bază de date şi returnează o instanţă a clasei Database dacă aceasta a fost creată cu succes. Va arunca excepţie dacă o bază de date cu acest nume există deja.
   3. `void deleteDatabase(String name)` - şterge baza de date cu numele dat ca input sau aruncă excepţie dacă baza de date nu există;
   4. `Database getDatabase(String name)` - returnează din memorie instanţa obiectului Database ce are numele specificat în parametru.

2. Database

   Operaţii asupra tabelelor:

   1. `void setName(String name)`- modifică numele bazei de date sau aruncă InvalidValue în cazul în care numele conţine caractere invalide (virgule sau ghilimele);
   2. `boolean exists(String tableName)`- returnează adevărat dacă există un tabel cu acest nume în instanţa curentă a bazei de date:
   3. `Table createTable(String name, List<Column> columnNames)`- încearcă crearea unui tabel. Aruncă excepţie dacă există deja un tabel cu numele primit ca input în această bază de date;
   4. `Table createTable(Table table)`- încearcă crearea unui tabel după o instanţă primită ca input. Aruncă excepţie dacă un tabel cu numele primit prin parametru există deja în această bază de date;
   5. `void deleteTable(String tableName)`- încearcă ştergerea unui tabel după nume. Aruncă excepţie dacă nu există niciun tabel cu numele primit în aceastp bază de date;
   6. `Table getTable(String name)`- încearcă returnarea unui tabel după nume. Aruncă excepţie dacă un tabel cu acest nume nu există în această bază de date.

3. Table

   Pentru operaţii asupra datelor din cadrul unui tabel:

   1. `List<String> getColumnNames()`- returnează o listă cu numele coloanelor din tabelul curent;
   2. `Column getColumn(String columnName)`- returnează coloana cu numele primit ca parametru dacă aceasta există, altfel aruncă o excepţie (`DoesNotExists`);
   3. `Map<Column, Field> getRow(int rowNumber)`- returnează rândul primit ca parametru, asemenea unui dicţionar între coloane şi valorile din cadrul acestora;
   4. `void insert(Map<String, Field> row)`- inserează un nou rând în tabel. Poate arunca excepţii dacă o coloană indicată nu exista în tabel sau tipul de date ce se doreşte a fi inserat este incompatibil cu tipul coloanei;
   5. `Map<Column,List<Field>> where(String columnName, FieldComparator.Sign sign, Field value)`- funcţie ce aplică clauza where din comenzile uzuale DBMS. Returnează toate rândurile ce satisfac condiţia primită ca parametru;
   6. `void updateWhere(String columnName, FieldComparator.Sign sign, Field value, Field newValue)`- similar cu funcţia where, doar că, în acest caz, atunci când codiţia este satisfăcută, valoarea este suprascrisă;
   7. `void deleteWhere(String columnName, FieldComparator.Sign sign, Field value)`- şterge rândurile ce satisfac condiţia primită ca parametru;
   8. `boolean deleteRow(int rowNumber)`- şterge rândul cu numărul primit ca parametru;

9. `void deleteRows(Map<Column, List<Field>> rowsAffected)` - șterge toate rândurile indicate în parametru. Pentru o utilizare eficientă, această funcție necesită, ca parametru, valorile de return de la funcția where;
10. `void deleteColumn(Column column)` - șterge o coloană după instanța obiectului ce se află în tabel. Instanța coloanei poate fi obținută prin funcția `Column getColumn(String columnName);`
11. `void insertColumn(Column column)` - inserează o nouă coloană. Poate arunca excepție `AlreadyExists` dacă există deja o coloană cu numele indicat în obiect;
12. `int getNumberOfRows()` - returnează numărul total de rânduri din tabel;
13. `Map<Column, List<Field>> select(Map<Column, List<Field>> data, List<Column> selectColumns)` - selectează și returnează doar acele coloane indicate de parametrul numărul 2. Pentru o utilizare eficientă, este sugerat faptul că primul parametru trebuie să fie valoarea de return a funcției `where`.

4. Column

   Clasa ce modelează obiectul de tip coloanaă din tabel.
   1. `void setName(String name);`
   2. `String getName();`
   3. `Type getType()` - returnează tipul de date al coloanei. Tipul este de formă Type.STRING sau Type.INT.
5. Field

   Structura pentru înregistrarea unei valori dintr-o coloană.
   1. `Field(String val)` sau `Field(int val)` în funcție de tip;
   2. `boolean isStringValueSet();`
   3. `boolean isIntValueSet;`
   4. `Integer getIntValue();`
   5. `String getStringValue();`
   6. `void setValue(String stringValue);`
   7. `void setValue(Integer intValue).`
6. FieldComparator

   Serviciu abstract necesar pentru compararea între două entități de tip Field, indiferent de tipul de date caractere/numeric.
   1. `boolean compareWithSign(Field o1, Field o2, Sign sign)` - returnează o valoare de adevăr, în funcție de evaluare. Poate aruncă excepție de `TypeMismatchException` dacă instanțele nu pot fi comparate;
   2. `public int compare(Field o1, Field o2)` - 0, dacă valorile sunt egale; <0, dacă primul argument este mai mic decât cel de-al doilea și >0, dacă argumentul 1 este mai mare decât argumentul 2.

Odată cu utilizarea nucleului, se pot întâlni următoarele excepţii principale:

1. DoesNotExist - pentru operaţii de ştergere;
2. AlreadyExists - pentru operaţii de inserare;
3. TypeMismatchException - când se încearcă inserarea unui număr într-o coloană de tip caracter sau când se încearcă compararea a două valori incomparabile.

Linia de comandă

Pentru implementarea acestui modul, s-a recurs la o modalitate cât mai simplistă şi, în acelaşi timp, restrictivă. Utilizatorul poate introduce de la linia de comandă doar comenzi prestabilite. Cu toate acestea, comenzile sunt validate după fiecare cuvânt, astfel încât utilizatorul să ştie punctul exact în care a greşit ceva în construcţia comenzii. Spre exemplu, pentru comenzi de ştergere a unui tabel: *delete table [table_name] from database [database_name]*, utilizatorul poate întâlni următoarele mesaje de validare:

- **Invalid command [comanda]** - în cazul în care nu introduce una dintre comenzile suportate;
- **Missing from/table/database keyword after delete** - în cazul în care tipul de ştergere nu este indicat corect;
- **Missing table name** - atunci când numele tabelei care se doreşte a fi ştearsă nu există;
- **Missing from keyword after table name** - atunci când numele tabelei nu este urmat de cuvântul cheie “from”;
- **Missing database keyword after from keyword** - atunci când cuvântul cheie “from” nu este urmat de cuvântul cheie “database”;
- **Missing database name after database keyword** - dacă nu este specificat numele bazei de date din care se doreşte a fi şters tabelul.

Pe lângă aceste mesaje, mai pot apărea excepţii aruncate de model pentru a arăta că un nume de entitate, specificat de utilizator, nu există.

Lista comenzilor suportate:

- `delete database [name]`
- `delete table [table_name] from database [database_name]`
- `delete from [table_name] from [database_name] where [column_name] [operator] [value]`
- `insert database [name] - insert database store`
- `insert table [table_name] into [database_name]`
- `insert column [column_name] [column_type] into [table_name] from [database_name]`

- `insert into [table_name] from [database_name] value [column_name1]=[value1],[column_name2]=[value2],[column_name3]=[value3]`
- `select [columnname1,columnname2,columnname3] from [table_name] from [database_name] where [column_name] [operator] [value]`
- `update database [database_name] name [new_database_name]`
- `update table [table_name] from [database_name] name [new_table_name]`
- `update data [table_name] from [database_name] set [new_value] where [column_name] [operator] [value]`
- `update column [column_name] from [table_name] from [database_name] name [new_column_name]`

Persistență

Modulul de persistență are rolul de a păstra starea aplicației între sesiunile de lucru. Acest lucru implică salvarea stării într-un fișier denumit “persistance.txt”, iar la repornirea aplicației, citirea acestui fișier și restaurarea stării sistemului.

Formatul fişierului este unul specific pentru aplicaţia noastră. Acesta este un format baza pe următoarele etichete:

1. `@@@START_DATABASE@@@` şi `@@@END_DATABASE@@@`, ce semnifică începerea şi, respectiv, terminarea definiţiei unei baze de date. Eticheta de start este urmată, pe rândul succesiv, de numele bazei de date. Între etichetele de *start* şi *end* pot apărea etichetele de mai jos;
2. `@@@START_TABLE@@@` şi `@@@END_TABLE@@@` - semnifică începerea definirii unui tabel şi este urmat de numele tabelului pe următoarea linie;
3. `@@@START_COLUMN@@@` şi `@@@END_COLUMN@@@` - reprezintă începerea definirii unei coloane în cadrul unui tabel. Trebuie să fie urmat de numele coloanei, tipul acesteia şi toate valorile de pe această coloană.

Un exemplu corect, din punct de vedere semantic al unui asemenea fişier de persistenţă, este următorul:

```
@@@START_DATABASE@@@
store
@@@START_TABLE@@@
persons
@@@START_COLUMN@@@
name
STRING
dan
andreea
```

ilinca
adelina
adelina
@@@END_COLUMN@@@
@@@START_COLUMN@@@
height
INT
187
169
169
150
150
@@@END_COLUMN@@@
@@@START_COLUMN@@@
age
INT
10
10
12
12
12
@@@END_COLUMN@@@
@@@START_COLUMN@@@
school
STRING
INFORMATICA
INFORMATICA
INFORMATICA
negruzzi
negruzzi
@@@END_COLUMN@@@
@@@START_COLUMN@@@
salariu
INT
0
0
0
0
0
@@@END_COLUMN@@@
@@@END_TABLE@@@
@@@END_DATABASE@@@
@@@START_DATABASE@@@
faculty
@@@START_TABLE@@@
employee
@@@START_COLUMN@@@
height
INT
187
169
169
@@@END_COLUMN@@@
@@@START_COLUMN@@@

```
name
STRING
dan
andreea
ilinca
@@@END_COLUMN@@@
@@@START_COLUMN@@@
age
INT
10
10
12
@@@END_COLUMN@@@
@@@START_COLUMN@@@
school
STRING
INFORMATICA
INFORMATICA
INFORMATICA
@@@END_COLUMN@@@
@@@END_TABLE@@@
@@@END_DATABASE@@@
```

Implementarea persistenţei constă în faptul că fiecare entitate (Database, Table, Column) are responsabilitatea de a expune o metodă *persist(Outputstream)*, ce are scopul de a serializa toate datele sale.

Interfaţa grafică

În pachetul `graphicalInterface` se pot găsi clasele răspunzătoare de crearea interfeţei grafice şi a evenimentelor ce pot fi acţionate prin aceasta.

1. **MainWindow**

Clasa ce extinde clasa `JFrame` din pachetul `javax.swing.*` este totodată elementul principal - interfaţa afişată la pornirea aplicaţiei. Această clasă are trei alte cadre: unul pentru obiectele de tip baze de date, unul pentru obiecte de tip tabele şi un al treilea pentru înregistrările dintr-un tabel şi evenimentele ce pot fi apelate pentru acestea.

În constructor, sunt create atât fereastra principală, cât şi celelalte trei; de asemenea, sunt şi populate cu instanţele existente în baza de date. Alte două metode sunt folosite pentru deschiderea şi aşezarea scheletului principal: `open()` şi `centerWindow()`.

În această clasă este setată proprietatea de persistenţă şi obiectul răspunzator de această acţiune, proprietate explicată în modulul de Persistenţă de mai sus.

Cele 3 cadre precizate mai sus au unele obiecte comune:

2. **GIConstants**: clasă unde pot fi găsite constante folosite în interfaţa grafică, precum titluri, mesaje etc.;

3. **ConfirmDialog**: clasă folosită pentru afişarea unui dialog modal de tip confirmare, în care titlul şi mesajul sunt customizabile, iar variantele de răspuns sunt: *Yes/No* sau închiderea ferestrei. Se oferă un răspuns prin metoda `public boolean confirm()`: ce returnează `adevărat` dacă utilizatorul a apăsat butonul de confirmare *Yes*, sau `fals`, în celelalte două cazuri posibile;

4. **InputTextPopUp**: clasă folosită pentru afişarea unui dialog modal în care utilizatorul poate introduce un text (de exemplu, un nume pentru o entitate). La fel ca în cazul clasei de mai sus, şi în această clasă, titlul ferestrei, mesajul acesteia şi tipul de mesaj (`ERROR_MESSAGE / QUESTION_MESSAGE`) sunt customizabile şi sunt setate în funcţie de nevoi. Mesajul de tip *QUESTION_MESSAGE* este folosit atunci când dialogul este afişat pentru prima dată pentru acţiunea curentă. În cazul în care dialogul este reafişat, acţiune ce are loc atunci când textul primit de la utilizator nu este valid (de exemplu: conţine caractere invalide, este este gol), este folosit mesajul de tip *ERROR_MESSAGE*. Diferenţa dintre acestea constă în iconiţa ce este afişată în fereastra de dialog şi este setată cu ajutorul unei variabile boolene în metoda responsabilă cu afişarea ferestrei `public Object openPopUp(String message, boolean isReopened)`.

Ambele clase de mai sus: `ConfirmDialog` si `InputTextPopUp` se folosesc de obiecte deja existente în pachetul `javax.swing`: ConfirmDialog se foloseşte de `JOptionPane.showConfirmDialog(..)`, iar InputTextPopUp de `JOptionPane.showInputDialog(..)`.

5. **PersistenceActionListener**: această clasă implementează interfaţa `ActionListener` din pachetul `java.awt.event` şi a fost creată pentru a putea seta proprietatea de persistenţă obiectelor ce au obiecte de tip listener. Acţiunea în sine este scrisă în metoda `beforePersist(ActionEvent e)`, după care este setată proprietatea. Astfel, după fiecare eveniment ce poate aduce schimbări în baza de date, informaţiile sunt persistate;

6. **DatabaseFrame**: clasa în care este creat scheletul pentru a afişa lista de baze de date existente şi punerea la dispoziţie a butoanelor pentru apelarea operaţiilor posibile pentru acestea.

Clasa conţine:

- o listă de baze de date ce este populată; această listă are proprietatea de derulare ce se activează atunci când dimensiunile setate pentru această fereastră este depaşită fie prin numărul de baze de date (derulare pe verticală), fie prin numele unei instanţe (derulare pe orizontală);
- butoane pentru acţiuni, precum şi obiectele de tip *listener* ale acestora;
- butonul de **creare**: prin acţionarea acestui buton este afişat un obiect de tip `InputTextPopUp` cu mesaj de tip *QUESTION_MESSAGE* astfel încât utilizatorul poate introduce numele bazei de date pe care doreşte să o creeze. Textul introdus este validat. În cazul în care o excepţie este aruncată: utilizatorul încearcă să introducă un text gol, o valoare deja existentă (un nume ce conţine caractere invalide), atunci dialogul va reafişa im nou mesaj de tip *ERROR_MESSAGE*;

- butonul de **actualizare** (nume) : prima dată este afișat un dialog de tipul celui de mai sus, iar dacă utilizatorul introduce un text valid, o fereastră de tip `ConfirmDialog` este afișată, iar utilizatorul este întrebat dacă este sigur de alegerea făcută printr-un mesaj precum “Sunteţi sigur că doriţi să schimbaţi numele bazei de date din *nume_curent* în *nume_nou*?“. La actualizarea unui nume se verifică și faptul că noul nume nu există deja în lista de baze de date existente;

- butonul de **ştergere:** după acţionarea acestui buton, este afișată o fereastră de dialog de tip `ConfirmDialog` pentru a cere de la utilizator confirmarea de ștergere a bazei de date selectate;

- butonul de **import de tabel:** prin acţionarea acestui buton este afișată o fereastră în care utilizatorul poate selecta fișierul de unde doreste sa faca importul de tabel in baza de date selectate. La fiecare apăsare a butonului, se creează un eveniment ce apelează o funcţie. Aceasta preia indexul bazei de date selectate din interfaţa grafică și numele bazei de date. După ce utilizatorul selectează fișierul pe care dorește să îl importe, se apelează metoda `importDataLineByLine`, cu adresa absolută a fișierului și numele bazei de date selectate;

- butoanele de actualizare, ștergere și importare de tabel sunt activate atunci când este selectată o instanță din lista de baze de date, sugerând faptul că operaţia poate fi făcută asupra selecţiei și dezactivate atunci când nu este făcută o selecţie. Activarea și, respectiv, dezactivarea sunt declanșate cu ajutorul metodei `valueChanged(ListSelectionEvent)` ce are loc atunci când utilizatorul face o (nouă) selecție;

- deoarece în clasa `TableFrame`, descrisă mai jos, a fost nevoie de valoarea asupra bazei de date selectate, a fost necesară adăugarea unei referinţe a acesteia.

7. **TableFrame**: clasa în care este creat scheletul pentru a afișa lista de tabele existente pentru baza de date selectată în fereastra anterioară și punerea la dispoziţie a butoanelor pentru apelarea operaţiilor posibile pentru acestea.

Această clasă se aseamănă foarte mult cu clasa anterioară, motiv pentru care mai jos se va face referire doar la diferențele existente în aceasta.

- butonul de **export de tabel** (spre deosebire de clasa de mai sus unde aveam butonul de import): prin acţionarea acestui buton, este realizat exportul într-un fișier de tip *CSV* a tabelului selectat. La apăsarea butonului, se creează un eveniment ce preia indexul tabelului selectat din interfață și numele bazei de date aferente acestuia și se apelează metoda `writeDataLineByLine`. Dacă operațiunea se realizează cu succes, utilizatorul va primi o notificare de tip pop-up.

- deoarece în clasa `TableContentFrame`, descrisă mai jos, a fost nevoie de valorile selecţiilor făcute (atât baza de date, cât și tabelul selectat), a fost necesară adăugarea unei referinţe a acesteia.

8. **TableContentFrame**: clasa în care este creat scheletul pentru a afișa informaţiile existente în tabelul selectat (în clasa descrisă anterior) și punerea la dispoziţie a butoanelor pentru apelarea operaţiilor posibile pentru acestea.

Clasa conţine un obiect de tip tabel: rânduri și coloane populate cu informația existent în sistem.

- butoane pentru acțiuni, precum și obiectele de tip listener ale acestora

- buton de **selectare**: prin acţionarea acestuia este afişat un obiect de tip `SelectPanel`. Acesta este format din lista de coloane din tabel şi opţiunea de adăuga o clauză “where”. Daca precizarea where nu este bifată, atunci utilizatorul poate bifa una sau mai multe coloane, astfel încât, după acţionarea butonului OK, va fi afişat un obiect de tip `ShowTableAfterSelect` cu tabelul populat doar cu coloana/coloanele selectată/selectate. În mod contrar, atunci când clauza where este bifată, utilizatorul poate selecta o singură coloană; suplimentar, trebuie să selecteze un comparator şi să introducă o valoare pentru comparare. Sunt făcute validări astfel încât valoarea introdusă pentru comparare să fie de tipul coloanei selectate şii să nu fie goală;

- buton de **inserare de coloană**: la acţionarea acestui buton, este afişată o fereastră în care utilizatorul poate introduce un nume de coloană şi selecta tipul acesteia (numeric/caractere). Se verifică faptul că numele coloanei este valid şi nu este deja existent în tabelul selectat;

- buton de **inserare de înregistrare în tabel**: prin acţionarea acestui buton, este afişat un obiect de tip `InsertRecordPanel`, în care se găseşte câte o înregistrare pentru fiecare coloană din tabel, unde utilizatorul are posibilitatea de a introduce noile înregistrări ce doreşte să le adauge în tabel. Valorile introduse sunt verificate ca să nu fie goale, să nu conţină caractere invalide şi să corespundă tipului de Coloană (nu se va putea introduce o valoare de tip caractere într-o coloană de tip numeric);

- buton de **actualizare a numelui unei coloane**: prin acţionarea acestui buton, este creat şi afişat un obiect de tip `UpdateColumnNamePanel`, în care utilizatorul are posibilitatea de a selecta o coloană şi de a introduce o valoare pentru noul nume pentru coloana selectată. Numele introdus de utilizator este validat pentru a nu putea fi null şi pentru a nu conţine caractere invalide;

- buton de **actualizare a valorii unor înregistrări**: prin acţionarea acestui buton, este creat şi afişat un obiect de tip `UpdateFieldPanel`, în care utilizatorul trebuie să introducă noua valoare, să selecteze o coloană şi un comparator şi valoarea pe care doreşte să o facă drept clauza where. Valoarea nou introdusă este validată precum este demonstrat mai sus (numele coloanei);

- buton de **ştergere**: prima coloană a tabelului conţine câte o casetă de bifat pentru fiecare înregistrare; butonul de ştergere poate fi acţionat atunci când cel puţin o selecţie este făcută, iar la acţionarea acestuia, dar şi la confirmarea de către utilizator a operaţiei, sunt şterse din tabel înregistrările selectate;

- pentru a putea crea schiţa tabelului, a fost nevoie de crearea o clasa `TableModel`, care să extindă clasa `AbstractTableModel`, clasă ce aparţine pachetului `javax.swing.table` . În această clasă, s-a creat schiţa tabelului după instanţa de `Table` primită şii s-a adăugat prima coloană cu căsuţe ce pot fi bifate.

Clasele: `UpdateColumnNamePanel`, `UpdateFieldPanel`, `InsertRecordPanel`, `SelectPanel`, `ShowTableAfterSelect` sunt clase de tip fereastră/dialog şi sunt folosite în comunicarea cu utilizatorul. Acestea au fost descrise pe scurt, pentru acţiunile ce se folosesc de ele.

CSV Service - import/export tabele

Clasa CsvService se ocupă de logica necesară gestionării operaţiilor de import şi export a tabelelor. Acest serviciu conţine două funcţii principale, denumite `writeDataLineByLine` şi *`importDataLineByline`*.

Atunci când se apelează funcţia *`writeDataLineByLine`*, se creează un fişier de tip CSV în directorul de lucru curent.

Funcţia *`writeDataLineByLine`* primeşte ca parametri două variabile de tip `String`, ce reprezintă numele bazei de date şi numele tabelului. În cadrul ei, se instanţiază obiectul `databaseManagementSystem` şi se extrage tabelul solicitat. Întâi se vor introduce în Excel numele bazei de date şi numele tabelului. Pentru fiecare linie din tabelul respectiv, sunt selectate numele coloanelor din baza de date şi se introduc în fişierul Excel. Mai departe, se parcurg liniile din baza de date şi se introduc în Excel.

	A	B	C	D
1	Database	Table		
2	store	employee		
3				
4	name	school	height	age
5	dan	INFORMA	187	10
6	andreea	INFORMA	169	10
7	ilinca	INFORMA	169	12
8	dan	INFORMA	187	10
9	andreea	INFORMA	169	10
10	ilinca	INFORMA	169	12
11	dan	INFORMA	187	10
12	andreea	INFORMA	169	10
13	ilinca	INFORMA	169	12
14	dan	INFORMA	187	10
15	andreea	INFORMA	169	10
16	ilinca	INFORMA	169	12

Fig. 2 Fişier Excel generat de metoda *`writeDataLineByLine`*

Funcţia `importDataLineByLine` primeşte ca parametri două variabile de tip `String`, ce reprezintă calea absolută către fişierul selectat şi numele bazei de date. În cadrul ei, se instanţiază obiectul `databaseManagementSystem` şi se începe parsarea fişierului Excel. Astfel, se extrag cele două valori (`databaseName` şi `tableName`) şi se verifică dacă numele tabelului există în baza de date. Dacă răspunsul este `true`, se vor actualiza toate liniile, iar dacă răspunsul este `false`, se va crea un nou tabel cu aceeaşi denumire şi aceleaşi coloane. În continuare, fişierul Excel va fi parcurs şi se va introduce linie cu linie în baza de date.

Testare - Phase 2

Nucleu

În modelul aplicaţiei, au fost testate toate entităţile ce contribuie la construirea sistemului. La acest moment, în ceea ce priveşte testarea, nucleul aplicaţiei are o acoperire de 84%.

Testarea a fost facută în manieră *sweetCase/failCase.* Mai exact, s-a testat atât cazul în care aplicaţia trebuie să se comporte bine în vederea efectuării schimbărilor, dar şi cazul în care sunt aşteptate erori/excepţii, cazuri excepţionale ce trebuiesc tratate de aplicaţie.

Acoperirea specifică pentru fiecare componentă din nucleu este următoarea:

1. Column - 97%
2. CsvService - 38%
3. Datatabase - 94%
4. DatabaseManagementSystem - 87%
5. Field - 69%
6. FieldComparator - 88%
7. Table - 93%

Linia de comandă

Pentru această parte, s-a recurs la multiple comenzi pentru fiecare operaţie. Aceste solicitări implică atât una sau mai multe comenzi valide, cât şi un număr considerabil de cereri invalide ce trebuiesc tratate corespunzător de aplicaţie.

Acoperirea specifică pentru fiecare componentă din nucleu este următoarea:

1. CommandLineParser - 72%
2. Delete - 72%
3. Insert - 98%
4. Select - 94%
5. Update - 98%

Un exemplu de listă de comenzi din cadrul unui test poate fi următorul (operaţia select):

```
public static final String[] GOOD_COMMAND = {String.format(
        "select age,height,name from %s from %s where age = 12",
        DataBuilder.TABLE,
        DataBuilder.DATABASE),

        String.format(
        "select * from %s from %s where age = 12",
        DataBuilder.TABLE,
        DataBuilder.DATABASE),

        String.format(
                "select * from %s from %s",
                DataBuilder.TABLE,
                DataBuilder.DATABASE)
};

private static final String[] wrongCommands = {
        "select",
        "select *",
        "select * from",
        "select * from inexistent_table_name",
        "select * from inexistent_table_name from",
        "select * from inexistent_table_name from inexistent_database_name ",
        "select * from inexistent_table_name from inexistent_database_name lala",
        "select * from inexistent_table_name from inexistent_database_name where ",
        "select * from inexistent_table_name from inexistent_database_name where age ",
        "select * from inexistent_table_name from inexistent_database_name where name * dan",
        "select status from " + DataBuilder.TABLE + " from " + DataBuilder.DATABASE + " where name = dan",
};
```

Fig. 3 Listă de comenzi din cadrul operației *select*

Persistența

Pentru acest modul s-a realizat un singur test. Acest test aduce o acoperire de 93%.

El are rolul de a trece aplicația prin stările de serializare și deserializare a datelor de test, cu verificarea integrității lor din momentul restaurării cu momentul dinaintea serializării.

Interfața grafică

În testarea interfeței grafice, s-a simulat interacțiunea cu utilizatorul prin folosirea `mock-urilor`. Nu s-a obținut code coverage de 100% deoarece în obiectele de tip fereastră/dialog au fost folosite metode statice ce aparțin obiectului `JOptionPane` din pachetul `javax.swing`, iar pentru acestea ar fi trebuit să fie folosite obiecte de tip `powermock`. Acest lucru nu a fost neapărat necesar însă, deoarece s-a putut face mock doar pentru răspunsul ce ar fi putut fi transmis prin acestea.

În teste, s-a urmărit în principiu ca, după apelul operațiilor prin butoane, informația alterată să fie corect înregistrată în baza de date și modul de interacțiune cu serviciile dorite. Validarea informației stocate s-a făcut prin salvarea valorilor ce s-a dorit a fi verificate înainte de apelul metodei `doClick()` a butonului și apoi compararea acestora cu noile valori obținute după interacțiunea cu serviciul. S-a mai urmărit și ca butoanele să aibă tipul de clasă listener dorit, pentru a asigura faptul că a fost atribuit listenerul corect și se va apela operația dorită.

În principal, înainte de fiecare test, a fost apelat un setup în care s-au creat informațiile necesare pentru testarea serviciilor.

Testele au fost făcute atât pentru cazurile în care execuția serviciului a fost făcută cu succes (de exemplu, introducerea unui nume pentru baza de date/tabel/coloană valide), cât

şi pentru cazuri de eşec (de exemplu, introducerea unei valori null/empty, introducerea unui nume invalid - care să conţină ghilimele sau virgule).

Aserţiuni - Phase 3

Prin aserţiunile adăugate, s-au urmărit în principal valorile parametrilor de intrare ce fac parte din precondiţii şi a valorilor de ieşire, ce fac parte din postcondiţii.

Rolul acestor aserţiuni este de a valida datele de intrare şi ieşire, astfel încât, la finalul oricărei tranzacţii din interiorul aplicaţiei, aceasta să rămână într-o stare consistentă.

S-au mai folosit aserţiuni pentru invarianţi pentru instrucţiunile de control de tip clasic folosit, iar pentru instrucţiunea de tip `foreach` s-a verificat ca starea fiecărui obiect ce este parcurs să fie integră. De exemplu, la parcurgerea unor coloane, s-a verificat ca numele coloanei să fie valid, iar tipul să fie setat corect (să fie una din valorile posibile: INT/STRING). Un alt exemplu este ca la parcurgerea înregistrărilor din coloane, înregistrările să fie valide: să aibă un tip setat corect, iar valoarea să fie în funcţie de tipul setat.

În interiorul serviciului CsvService, s-au folosit aserţiuni pentru a verifica dacă parametrii primiţi în apelarea funcţiilor nu au valoare nulă.

Ghid pentru utilizator

Utilizatorului îi va apărea următoarea imagine atunci când va accesa baza de date. După cum se poate observa, doar două butoane sunt activate momentan, cel de *Create Database* şi *Create Table*.

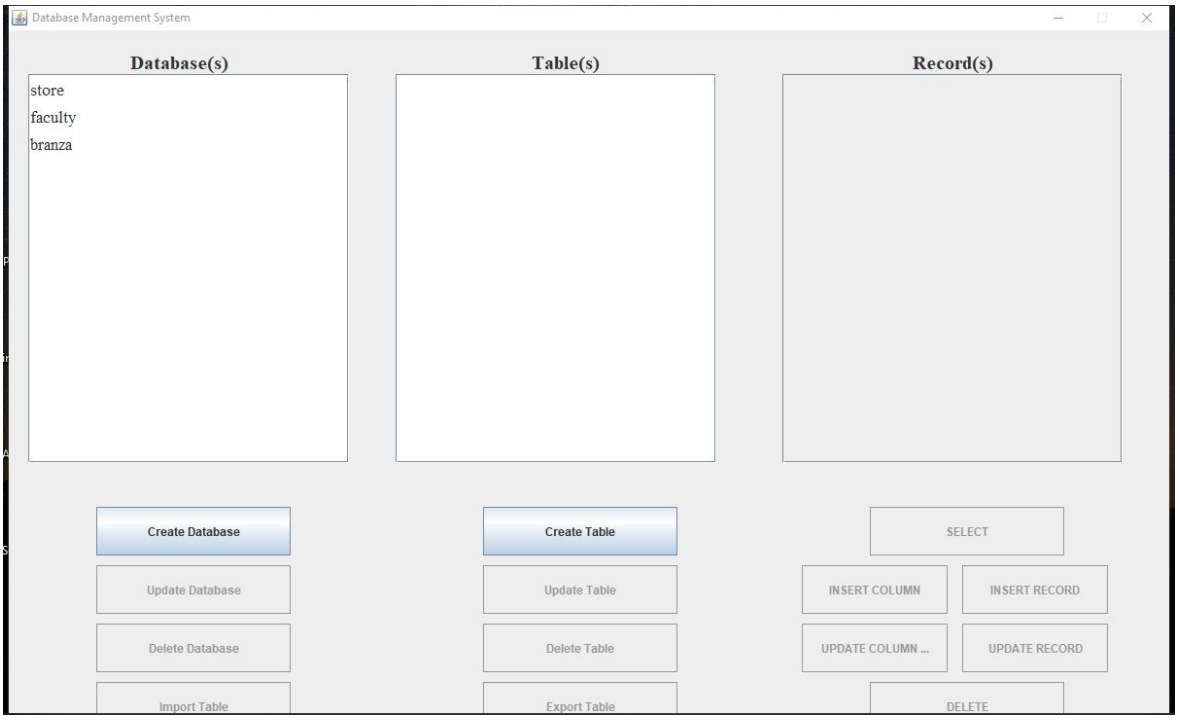

Fig. 1 Bază de date - imagine de ansamblu

În cazul în care utilizatorul doreşte să acceseze oricare dintre bazele de date afişate în tabela *Database(s)*, va trebui să dea click pe numele acesteia.

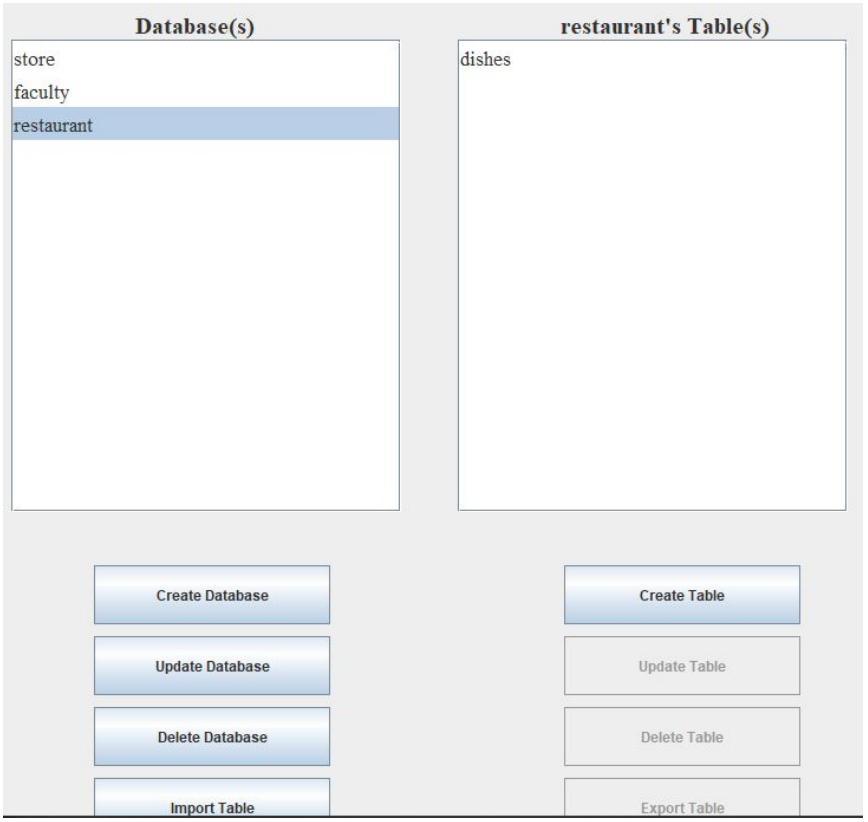

Fig. 2 Accesarea bazei de date *store*

Odată aleasă baza de date, se va permite accesul şi la celelalte butoane de comandă de sub tabela *Database(s)*:

- *Update Database* - permite actualizarea bazei de date;
- *Delete Database* - permite ştergerea bazei de date;
- *Import Table* - permite importul de tabele.

Fig. 3 Butoane de comandă pentru baza de date selectată

De asemenea, în tabela din dreapta, denumită *[insert name of the selected database]'s Table(s)*, vor apărea tabelele corespunzătoare bazei de date pe care utilizatorul a selectat-o.

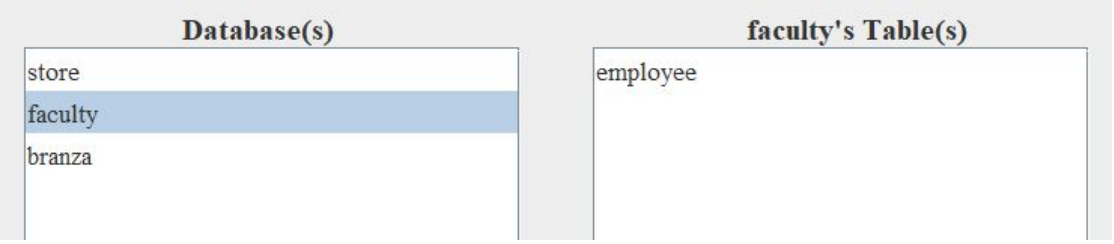

Fig. 4 Tabel aferent bazei de date *faculty*

Odată ales tabelul, se va permite accesul şi la celelalte butoane de comandă de sub tabela *Table(s)*:

- *Update Table* - permite actualizarea tabelului;
- *Delete Table* - permite ştergerea tabelului;
- *Export Table* - permite exportul de tabele.

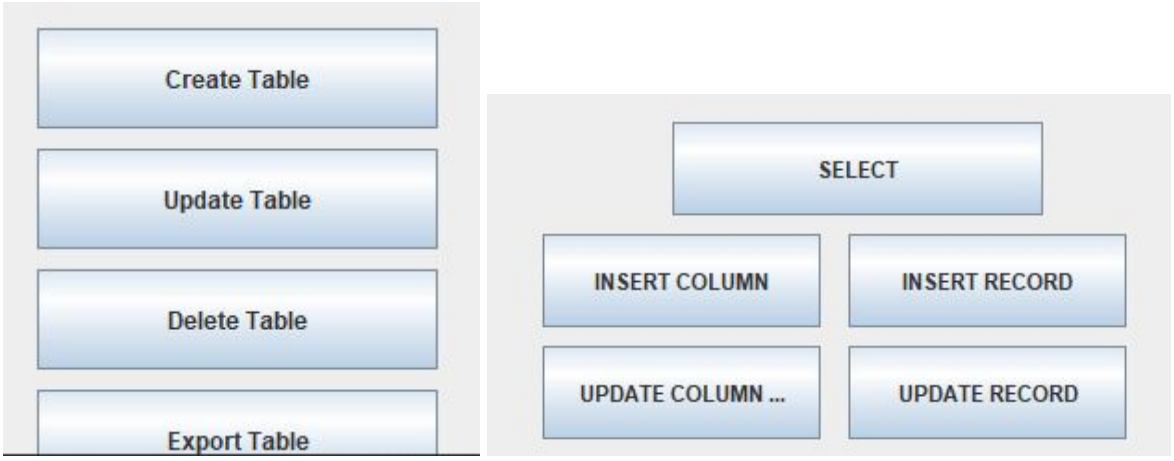

Fig. 5 Butoane de comandă pentru tabelul selectat, respectiv registrul selectat

În cea de-a treia tabelă, denumită *[insert name of the selected table's Record(s)*, uitilizatorul va putea vizualiza registrul corespunzător tabelului selectat de acesta. Fiecare registru poate conţine mai multe câmpuri, cum ar fi: *school*, *height*, *age*, *name* etc.

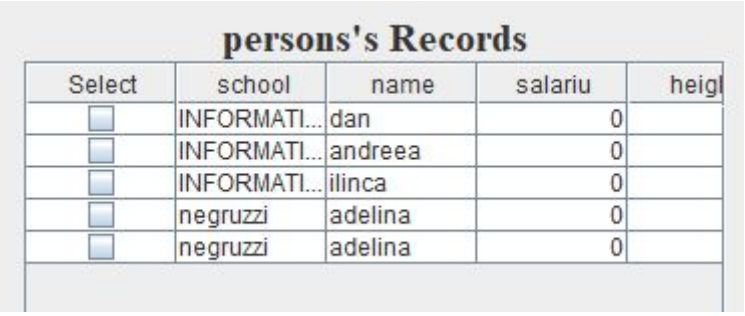

Fig. 6 Registru aferent tabelului *persons*

Totodată, se va permite accesul şi la butoanele de comandă de sub tabela *Record(s)*:

- *Select* - permite selectarea unui câmp şi filtrarea lui în funcţie de preferinţele utilizatorului;

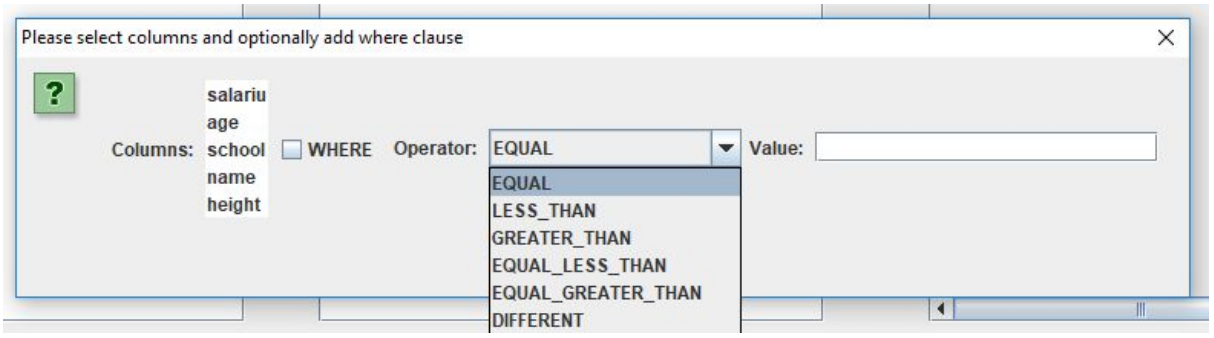

Fig. 7 Opţiunile corespunzătoare butonului *Select*

De exemplu, dacă utilizatorul vrea să afle câţi studenţi au înălţimea mai mică decât 1,70, va selecta coloana *height*, va bifa butonul *WHERE*, va selecta operatorul *LESS_THAN* şi va introduce valoarea 170 în secţiunea *Value*, urmând ca rezultatul să arate ca în imaginea de mai jos.

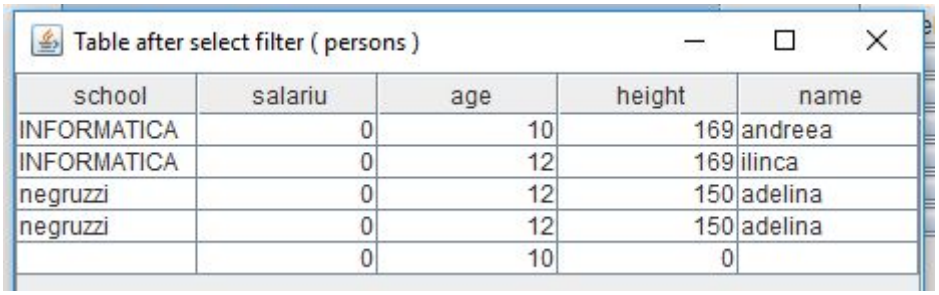

Fig. 8 Registrul *persons* după aplicarea filtrului *Select*

- *Insert Column* - permite introducerea unei coloane noi/unui câmp nou (ce poate fi de tip *int* sau *string*);

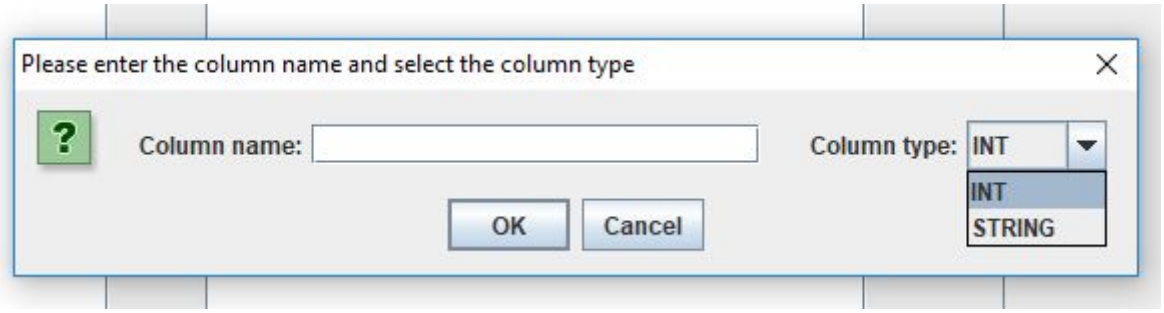

Fig. 9 Meniul pentru introducerea unei coloane noi

- *Update Column* - permite actualizarea unei coloane;
- *Insert Record* - permite introducerea unei înregistrări complete noi (o înregistrare este formată din mai multe câmpuri ce trebuie completate de către utilizator);

Fig. 10 Meniul pentru introducerea unei înregistrări noi

- *Update Record* - permite actualizarea unei înregistrări.

Butonul de comandă *Delete* de sub această tabelă se va activa doar în momentul în care utilizatorul va selecta câmpul/câmpurile pe care doreşte să îl/le şteargă (la fel şi în cazul bazelor de date şi al tabelelor).

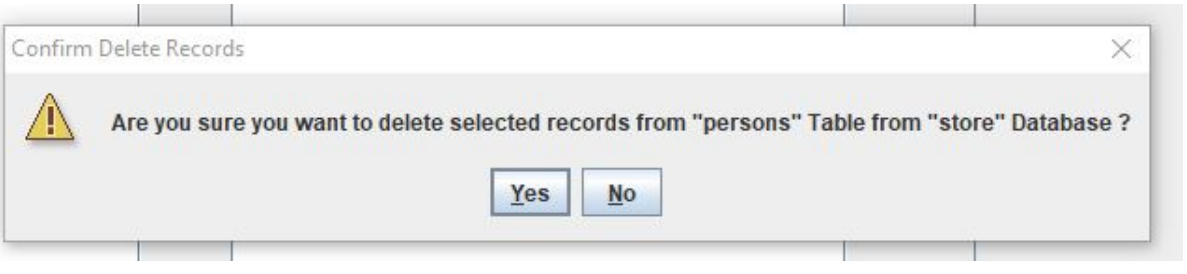

Fig. 7 Mesaj de înştiinţare la apăsarea butonului *Delete*

Utilizatorul va putea observa la fiecare buton de comandă un mesaj de înştiinţare ce îi sugerează ce trebuie selectat pentru a putea folosi acel buton, aşa cum este exemplificat şi în această imagine.

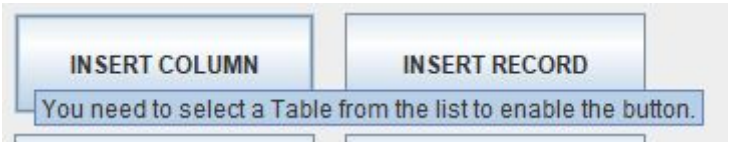

Fig. 11 Mesaj de înştiinţare la apăsarea butonului *Delete*

În cazul în care utilizatorul doreşte să aducă din propriul său calculator un tabel în aplicaţie, va trebui să dea click pe butonul *Import Table* şi să selecteze fişierul CSV respectiv. După cum se poate observa, este permis doar importul de fişiere CSV.

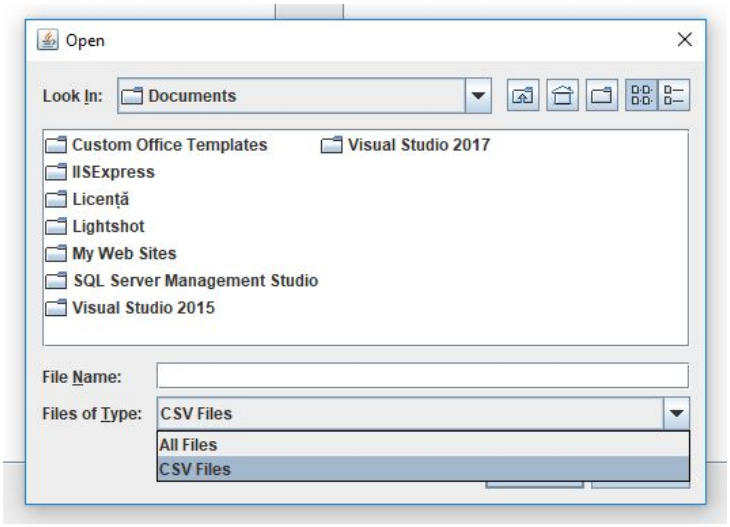

Fig. 12 Meniu pentru importul unui tabel

În cazul unui export de tabel, utilizatorul trebuie să dea click pe butonul de *Export Table*, după care va primi o înştiinţare ce îl informează că acţiunea a fost realizată cu succes.

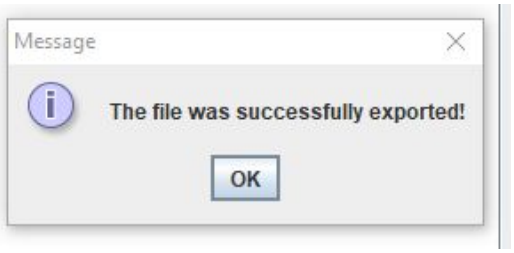

Fig. 13 Mesaj de înştiinţare - acţiune reuşită

După acest mesaj, utilizatorul poate consulta local fişierul Excel (tabelul) ce tocmai a fost exportat, aşa cum este demonstrat şi în imaginea de mai jos.

	A	B	C	D
1	Database	Table		
2	store	employee		
3				
4	name	school	height	age
5	dan	INFORMA	187	10
6	andreea	INFORMA	169	10
7	ilinca	INFORMA	169	12
8	dan	INFORMA	187	10
9	andreea	INFORMA	169	10
10	ilinca	INFORMA	169	12
11	dan	INFORMA	187	10
12	andreea	INFORMA	169	10
13	ilinca	INFORMA	169	12
14	dan	INFORMA	187	10
15	andreea	INFORMA	169	10
16	ilinca	INFORMA	169	12

Fig. 14 Fişier Excel generat după acţiunea butonului *Export Table*

Contribuții

Faza 1 - Dezvoltare

1. Nucleu - Dan-Gabriel Cazacu
2. Linia de comandă - Dan-Gabriel Cazacu
3. Persistenţă - Dan-Gabriel Cazacu
4. Interfaţa grafică - Andreea Prodan
5. Import/Export CSV - Ilinca-Magdalena Stumbea

Faza 2 - Testare

1. Nucleu - Dan-Gabriel Cazacu - 70 de teste
    a. Column - 97%
    b. Datatabase - 94%
    c. DatabaseManagementSystem - 87%
    d. Field - 69%
    e. FieldComparator - 88%
    f. Table - 93%
2. Linia de comandă - Dan-Gabriel Cazacu - 24 de teste
    a. CommandLineParser - 72%
    b. Delete - 72%
    c. Insert - 98%
    d. Select - 94%
    e. Update - 98%
3. Persistenţă - Dan-Gabriel Cazacu - 1 test
    a. DatabasePersistance - 93%
4. Import/Export CSV - Ilinca-Magdalena Stumbea - 4 teste
    a. CsvService - 38%

5. Interfaţa grafică - Andreea Prodan - 45 de teste
    a. MainWindow - 100%
    b. DatabaseFrame - 86 %
    c. TableFrame - 89%
    d. TableContentFrame - 80%
    e. TableModel - 82%
    f. UpdateColumnNamePanel - 79%
    g. UpdateFieldPanel - 84%
    h. ShowTableAfterSelect - 100%
    i. InsertRecordPanel - 100%
    j. PersistenceActionListener - 87%
    k. SelectPanel - 20%
    l. ConfirmDialog - 20%
    m. InputTextPopUp - 22%

Ultimele trei clase au un procent scăzut de acoperire, deoarece se folosesc de obiectul JOptionPane din clasa swing şi de metodele statice ale acestui obiect şi deoarece mai conţin metode de customizare ale ferestrei precum setarea unui titlu.

Faza 3 - Aserţiuni

La această etapă, Andreea Prodan a contribuit cu aserţiuni pentru toată aplicaţia (ajutată în cazurile in care nu era sigură de cum/ce verificat cu aserţiuni de către Dan-Gabriel Cazacu) , excepţie făcând serviciul de Import/Export CSV, ce a fost realizat de Ilinca-Magdalena Stumbea.

Faza 4 - Documentaţie

Pentru documentaţia generală, fiecare membru al echipei şi-a documentat propria muncă în acest document.

Manualul de utilizare a fost construit integral de Ilinca-Magdalena Stumbea.