LiteData.al

14.11.2020

AlbanianSkills 6

Software Development Competition

Project Statement

Overview

LiteData.al is the newest startup company in Albania founded by two Doctorate Graduates. They have found a new, faster way to store and operate on data. The company has just raised capital and is looking for a talented software engineer that can prove he can work on complex tasks while maintaining good code quality.

In order to test your skills, they want you to create a simulation of how their database will work. Your task will be to read some commands, interpret them and then show the output of those commands. These commands will operate on a "database" which you will store in runtime. This means that the data that will be provided from commands in the input, will be stored in objects in your code and not in external databases.

They are really interested in:

1. **Providing a functional, tested software.** A disastrous project which works is much better than a well worked project which doesn't work, results matter.
2. **Object Oriented Design.** There is no better way to reflect the real world inside a software than to use OOP.
3. **Code readability and re-usability.** Your code is going to be read by other team members, or even by you after a couple of months. You will also need to add new features to the app from time to time. Remember to KISS.
4. **Efficient solutions: fast algorithms, effective data structures usage.** We all want softwares that work fast and use a small amount of memory.

Project Specifications

LiteData.al wants you to deal with every feature separately, and then wants you to merge all those features together creating a fully woking program. The program they want you to create should have the following components:

- A single **database**. Your duty is to simulate how a single database should work. No need to create support for multiple databases.
    - *Example: school*

- The database will contain many **resources**. Resources represent objects/things that have a meaning in real life (entities). A resource has:
    - a name which is unique among all other resources
    - a set of predefined columns that represent the attributes that describe this resource
    - *Example: student with columns ID, name, surname, age*
- A **row** is a single instance of a resource. Every resource may contain many rows that represent different instances of that resource.
    - *Example: student => ID: 1, Name: John, Surname: John, Age: 22*

Problems

In order for a problem to be considered solved, it should pass all the test cases provided in the input file.

I. Create Resource

First things first, the company wants to be able to create resources on this "on-the-fly" database. A resource can have a "Primary Key column (PK)" which is the column that uniquely identifies one type of that resource. In our case, that PK would be an ID which is unique among all rows inside that resource.

Input Format:

CREATE RESOURCE {name} ({colName}_PK, {colName}, ...);

Constraints:

- Every resource name should be unique.
- Every resource should have at least one column.
- Every resource should have one and only one PK.
- PK will always be a long datatype.
- PK is identified by _PK after the column name(check the examples below).
- Resources cannot have different columns with the same name.
- The list of columns will be between brackets. The column names will be separated by a comma and in the end of the statement there will be a ';'.
- Resource name should contain only letters and _

Correct input examples:

CREATE RESOURCE person (id_PK,name,surname,age);

CREATE RESOURCE school (id_PK,name,numberOfStudents);

Wrong input examples:

CREATE RESOURCE person(id,name,surname); // missing PK

CREATE RESOURCE school(id_PK,name,name); // non-unique column names

Output format:

- In case of correct command, output:

  Created resource:{name} in the database
- In case of any error, output:

  Invalid operation

Input File 1	Output File 1
CREATE RESOURCE person(id_PK,name,surname,age);	Created resource:person in the database

Explanation 1:

The command is correct. Output is printed in the file.

Input File 2	Output File 2
CREATE RESOURCE person(id,name,surname,age);	Invalid operation

Explanation 2:

The command is incorrect because no PK is provided.

Input File 3	Output File 3
CREATE RESOURCE person(id_PK,name,name,age);	Invalid operation

Explanation 3:

The command is incorrect because two columns with the same names are provided.

II. Add To Resource

After creating a resource, the company would like to add data to these resources. Of course, they would like to write simple commands and the program should handle the process.

Every resource should be able to keep track of the current value its PK holds. When adding a new row to the resource, if a value for the PK column is not specified, then the resource uses the current value it holds for the PK, increments it by one, and assigns it to the new row that is being added. On the other hand, if a value for the PK column is specified in the "Add to Resource" operation, then the row is saved with the value specified and the current value being held by the resource is given the value of the newly added PK value. Consider the following example:

1. Initially current value of PK for resource person is PK = 1
2. ADD TO person (name, test) // this row gets ID = 1 and PK becomes 2
3. ADD TO person(name, test2) // this row gets ID = 2 and PK becomes 3
4. ADD TO person(ID, 77) // this row gets ID = 77 and PK becomes 78
5. ADD TO person(name, test3) // this row gets ID = 78 and PK becomes 79

Input Format:

ADD TO {resourceName} ({colName},{colValue}) (...);

Constraints:

- After the resource name, the list of pairs(colName, colValue) will be inside brackets separated by comma. The pairs will be separated by a space.
- The command ends with a semicolon.
- Column names cannot be repeated.
- If a column that is not PK is not included, it should be added as null.
- If PK is included, its value should be higher than existing PK.
- If PK is not included, its value should be added as one more than the last PK.
- The value of PK should always be a long datatype.

Output format:

- In case of correct command, output:

  Added row to resource:{name}

- In case of any error, output:

  Invalid operation

Input File 1	Output File 1
CREATE RESOURCE person (id_PK,name,surname); ADD TO person (id,2) (name,John) (surname,Smith);	Created resource:person in the database Added row to resource:person

Explanation 1:

The resource is created correctly and the person is added correctly.

Input File 2	Output File 2
ADD TO person (id,2) (name,John) (surname,Smith);	Invalid operation

Explanation 2:

Adding failed because the resource person doesn't exist.

Input File 3	Output File 3
CREATE RESOURCE person (id_PK,name,surname); ADD TO person (id,2) (name,John) (surname,Smith); ADD TO person (name, Jim) (surname,Carrey);	Created resource:person in the database Added row to resource:person Added row to resource:person

Explanation 3:

Create resource succeeded. Added row with id 2 to person. 3rd line doesn't have id provided which is PK, so we add the row with id = 3; name = Jim and surname=Carrey.

III. Remove From Resource

We just created the ability to add data, why not be able to remove them as well. Therefore the team of LiteData wants you to create an operation that removes data from any resource based on a condition provided.

Input Format:

REMOVE FROM {resourceName} at {colName}={colValue}

Constraints:

- You should remove from the resource each row that fulfills the condition.
- The resource should exist.
- The column name should exist.

Output format:

- In case of correct command, output:

  Remove operation finished at resource:{name}

- In case of any error, output:

  Invalid operation

Input File 1	Output File 1
CREATE RESOURCE person (id_PK,name,surname); ADD TO person (id,2) (name,John) (surname,Smith); REMOVE FROM person at id=2	Created resource:person in the database Added row to resource:person Remove operation finished at resource:person

Explanation 1:

The resource is created correctly. The person is added correctly. The remove command is correct because there is a person with id=2 and that person is removed.

Input File 2	Output File 2
CREATE RESOURCE person (id_PK,name,surname); REMOVE FROM person at age=24	Created resource:person in the database Invalid operation

Explanation 2:

Removing failed because age doesn't exist.

IV. Search

As more data are added to the database, finding a specific row becomes harder and harder. Therefore the team at LiteData would like to search for data easier using this command.

Input Format:

SEARCH {name} with {colName}={colValue}

Constraints:

- You should find all the rows in that resource that fulfill the condition.
- The resource should exist.
- The column name should exist.
- The resulting rows should be sorted based on the ID.
- The column arrangement when printing a row should be the same as provided in CREATE RESOURCE command.

Output format:

- In case of correct command, output the rows **in a single line**. **Keep in mind that the arrangement of columns should be the same as it was when the resource was created.**

  {colName}:{colValue} {colName}:{colValue}|

  In case of null, print null as a string.

  If no results in database, print No results

- In case of any error, output:

  Invalid operation

Input File 1	Output File 1
CREATE RESOURCE person (id_PK,name,surname); ADD TO person (id,2) (name,John) (surname,Smith); SEARCH person with name=John	Created resource:person in the database Added row to resource:person id:2 name:John surname:Smith|

Explanation 1:

The resource is created correctly. The person is added correctly. Search command is corrected because name is a column in the resource person.

The output format should end always with a '|' with no spaces.

Input File 2

```
CREATE RESOURCE person (id_PK,name,surname);
ADD TO person (id,2) (name,John) (surname,Smith);
ADD TO person (name,John) (surname,Carrey);
SEARCH person with name=John
SEARCH person with name=ZII
```

Output File 2

```
Created resource:person in the database
Added row to resource:person
Added row to resource:person
id:2 name:John surname:Smith|id:3 name:John surname:Carrey|
No results
```

Explanation 2:

All commands were correct.
The second add didn't include a PK, so autoincrement was used and id of John Carrey was 3.

V. Bulk operation

Sometimes you want to perform many operations as a single one. This means that sometimes some operations should either succeed altogether, or all of them should fail if one of them fails. In order to provide this functionality, LiteData.al wants you to create a bulk operation.
A bulk operation starts with a **START BULK** and ends with an **END BULK** operation. Between START BULK and END BULK there may be several different operations. Every time one of those operations fails, all operations inside the bulk need to be reverted. Therefore nothing should change in the database if something wrong happens.

Input Format:

START BULK
<OPERATIONS>
END BULK

Constraints:

- A START BULK should always be closed by an END BULK, therefore no two bulks can happen at the same time.
- You should print the corresponding message for every operation that is executed inside the bulk.
- In case of an error, the operations performed until that point need to be reverted / rollbacked.
- The database should be the same as before the bulk operation in case of an error.

Output format:

- You should print the following lines in case of:

  START BULK => print "Started bulk operation"

  END BULK with success => print "Ended bulk operation with no errors"

  END BULK with errors => print "Ended bulk operation with errors. Rollback applied"

  For every operation inside the bulk, print the correct message for each one until the error occurred.

Input File 1	Output File 1
START BULK CREATE RESOURCE person (id_PK,name,surname); ADD TO person (id,203) (name,Regan) (surname,Kaci); END BULK	Started bulk operation Created resource:person in the database Added row to resource:person Ended bulk operation with no errors

Explanation 1:

All commands are correct.

Input File 2

```
CREATE RESOURCE person (id_PK,name,surname);
START BULK
ADD TO person (id,203) (name,Regan) (surname,Kaci);
ADD TO person (id,204) (name,Patrik) (surname,Simixhiu);
CREATE RESOURCE school (id,rank);
ADD TO SCHOOOL (id,2) (rank,1);
ADD TO SCHOOL
END BULK
```

Output File 2

```
Created resource:person in the database
Started bulk operation
Added row to resource:person
Added row to resource:person
Created resource:school in the database
Invalid operation
Ended bulk operation with errors. Rollback applied
```

Explanation 2:

Created resource person success. Then “Started bulk operation” is printed.
Two correct additions completed. Created resource school.
Add to SCHOOL fails because there is no resource named SCHOOL and we print Invalid operation.

Therefore we roll back. So we delete the resource school and remove the two rows from person resource.

In the end, we don’t execute the other statements, just print “Ended bulk operation with errors. Rollback applied”

VI. State of database

Maintaining the state of a database is very important. Therefore the company is very interested in seeing how the database looks at every moment. For this reason, they want you to implement a command which prints all the resources in the database.

Input Format:

DB_STATE

Constraints:

- You are required to print all resources sorted alphabetically.
- **For every resource, you should print its data in a single row.**

Output format:

- You should print:

  Resource:{name} {size} rows

  {colName}:{colValue} {colName}:{colValue}|

Input File 1	Output File 1

`CREATE RESOURCE person (id_PK,name,surname);`
`ADD TO person (id,203) (name,Orges) (surname,Balla);`
`DB_STATE`
	`Created resource:person in the database`
	`Added row to resource:person`
	`Resource:person 1 rows`
	`id:203 name:Orges surname:Balla|`

Explanation 1:

All commands are correct.
The person resource has only 1 row, we print that row and add | in the end.

Input File 2

```
CREATE RESOURCE person (id_PK,name,surname);
ADD TO person (id,203) (name,Orges) (surname,Balla);
ADD TO person (id,204) (name,Aleks) (surname,Ruci);
CREATE RESOURCE school (id,rank);
ADD TO school (id,1) (rank,1);
ADD TO school (id,2);
DB_STATE
```

Output File 2

```
Created resource:person in the database
Added row to resource:person
Added row to resource:person
Created resource:school in the database
Added row to resource:school
Added row to resource:school
Resource:person 2 rows
id:203 name:Orges surname:Balla|id:204 name:Aleks surname:Ruci|
Resource:school 2 rows
id:1 rank:1|id:2 rank:null|
```

Explanation 2:

All operations are correct.
When DB_STATE is called, there are two resources in the database: person and school. Since we are sorting alphabetically person comes before school, so we print person first and the number of rows.

In the next line, we print all rows of resource person in a **single line** divided by |

Then we do the same for resource school.

HAVE FUN :)