JATABASE

DBMS Constructed using Java

Nabeel Mirza

Version 0.2

Index

About Author

Nabeel Mirza

Bachelor's in Computer Science (Class of 2023)

Insititute of Business Administration, Karachi

Introduction

Jatabase is a SQL inspired database solution. It is open source.

Initially, Jatabase wasn't supposed to be taken seriously. This project was a proof of concept for an interpreter for a different project. Jatabase started on September 21st of 2022 at 21:00. Within the next ten hours, the first few features (record insertion, primary keys, table creation and dynamic table-column allocation to name a few) were made. This was version 0.1

The concept was shown to my teacher. The teacher was satisfied by the progress of Jatabase. The project enhanced my knowledge of input processing and string manipulations. With better skills and know-how than before, I was able to implement more features to Jatabase.

My goal with this project is to completely wipe out the use of JDBC in Java Programs. Firstly, JDBC requires a 35MB driver to work properly. Then JDBC requires an external database solution to store and fetch data from. This external database solution will be a bigger size than the driver itself. This causes Java Programs to have Database capability at higher storage costs.

Whereas enterprise level applications may benefit from the security and versatility that JDBC and other components provide, the whole setup phase and API learning may be too much for smaller level applications that require simple record keeping and whose record count won't exceed 10 million records.

Jatabase brings together SQL-like syntax with intuitive commands and interface that allows for simple data entry and fetching. All the user has to do is import the code to his project and write easy to memorize and intuitive syntax to fulfill his or her DBMS demands.

Although I doubt that Jatabase will be a direct competitor to SQL but I hope my efforts are enough to at least replace SQL with Jatabase for Java Applications.

Setup

Jatabase version 0.2 is made up of only 4 classes. Put these classes in your project folder

Column.java
Table.java
ErrorWarning.java
Interpretor.java

To use Jatabase like a SQL Command Line Program, simple paste the code below in your main class' main function

```
Scanner Input = new Scanner(System.in);

while(true) {

 System.out.print("Input :");
 String S = Input.nextLine();
 if(S.toLowerCase().equals("bye")) {
  ErrorWarning.Throw(3001);
  break;
 }
 Interpretor.HitScan(Interpretor.MakeReadable(S));
 System.out.println();

}
```

If all the steps are followed correctly. The console of your IDE or the CMD should look as shown below.

```
Input :
```

Congratulations, now you can use Jatabase exactly how you would use SQL apart from the excessive downloading, installing and other caveats.

A simple life is its own reward.
George Santayana

Wow, this was less than a page long.

Syntax

Creating Tables

Jatabase tables are columns linked using a doubly linked list. Each node of this linked list contains an arraylist that corresponds to the assigned data type. Each node also contains a prefix, suffix, primary key boolean and its name.

To create a table, the syntax is as follows

```
CREATE  TABLE  <Table_Name>  <Column_Names>  <Datatypes>
```

<Table_Name> are simply the label you want to attach to your Table. All tables must have a unique table name. This field is case-sensitive.

You can say that **<Column_Names>** are an array of strings that serves to name the columns of your table. However, in a CLI, we cannot insert entire arrays but what we can do is use Seperators.

Finally, **<Datatypes>** are an array of strings. Datatypes should be respective of order of the datatypes you want to use for columns. Each datatype is separated by colons.

The default Seperator for Jatabase is : (colon)

An example to create a table :

```
CREATE  TABLE  Cars  Name:Brand:Speed:Fuel  String:String:Double:String
```

Displaying Table Details

Table details can be displayed in two ways. DET is simply DET(AILS).

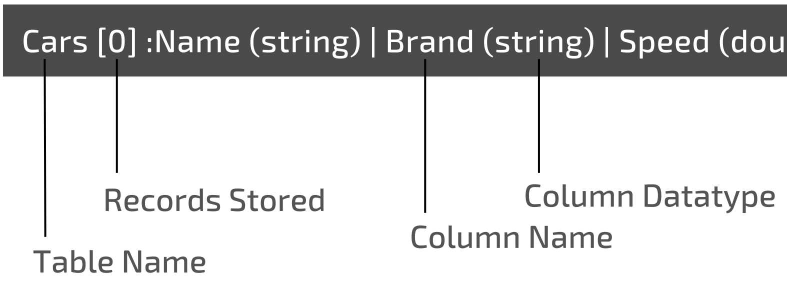
```
DET TABLES
```

```
DET TABLE <Table_Name>
```

The first query generates details of all the tables in the current database. The latter uses the table name to determine which table's details to print. The latter does not tell the number of records stored inside a table.

```
Cars [0] :Name (string) | Brand (string) | Speed (double) | Fuel (string) |
```

After querying DET TABLES



- ## Adding Records

Jatabase uses SQL-like record entry Syntax. ADDR is simply ADDR(ow).

```
ADDR  <Fields> TO  <Table_Name>
```

An example of adding a record to the table made previously is

```
ADDR Civic:Honda:177:Petrol TO Cars
```

ADDR adds entry to fields respective of order. Do not forget to check for case-sensitive related errors. Text in yellow is case-sensitive. Text in green is not case-sensitive.

- ## Fetching Records

To fetch records from a table, simply type

```
FETCH * FROM <Table_Name>
```

An example of printing records from the table we added records to is

```
FETCH * FROM Cars
```

In the syntax and example above, * means to select all fields. To fetch records in a different order or to fetch records by select columns, simply use

```
FETCH <Column_Names> FROM <Table_Name>
```

An example of printing records from the table we added records to with select columns is

```
FETCH Brand:Name:Fuel FROM Cars
```

You can also repeat field names and print the same columns but I do not know why you would do that!

```
FETCH Brand:Brand:Brand FROM Cars
```

```
Brand | Brand | Brand |
Honda, Honda, Honda,
```

After querying FETCH Brand:Brand:Brand from Cars

```
Brand | Name | Speed |
Honda, Civic, 177.0,
```

After querying FETCH Brand:Name:Speed from Cars

Adding Columns

There is nothing worser than making a table and then realizing that you missed out on adding a field. Jatabase has you covered though.

```
ADDC <Column_Name>:<Datatype> TO <Table_Name>
```

An example for the same table

```
ADDC Price:Integer TO Cars
```

This means that you want to add a new column of name **Price** of datatype **Integer** to your already made table **Cars**.

Jatabase automatically sets the value for the new columns to the default or intuitively default possible values.

The default values for the supported datatypes are

Integer : 0
Double : 0.0
Boolean : false
Character : '-'
String : "-"

Fetching Records now will lead to a different result.

```
Name | Brand | Speed | Fuel | Price |
Civic, Honda, 177.0, Petrol, 0,
```

*After querying FETCH * FROM Cars*

Editing, Modifying or Tampering Cells

It is safe to assume that Jatabase stores data in an excel-esque format.

Name	Brand	Speed	Fuel	Price
Civic	Honda	177.0	Petrol	0

However, what's not safe to assume is Honda selling their Civics at zero.

We know that excel stores its values in cells and that all rows are made up of N number of cells (where N = Total Columns). Expanding on that idea, we can say that a table is a grid of N x R (R = Total Records).

Using this logic, we can say that the cell with red value in it has the coordinates **5,1** .

To change the value of a cell, we write

```
TAMPERCELL  <x>:<y>:<New_Value> OF <Table_Name>
```

Now, to make Honda less likely to bankrupt.

```
TAMPERCELL  5:1:6600000 OF Cars
```

Editing, Modifying or Tampering Rows

To rewrite an entire row w.r.t. its index, use the following syntax

```
TAMPERROW <Row_Number> <New_Values> OF <Table_Name>
```

An example for the same table

```
TAMPERROW 1 NSX:Acura:212:Petrol:3450000 OF Cars
```

It is unknown why TAMPER was the choice instead of EDIT or MODIFY. Probably because changing the contents of a record is considered nefarious...

Deleting Rows

To delete an entire row, we write

```
ERASE <Row_Number> OF <Table_Name>
```

Deleting Tables

To delete an entire table, we write

```
ERASE TABLE <Table_Name>
```

Same old table, same old example

```
ERASE TABLE Cars
```

Deleting Data but keeping Table

To delete an entire table's data but keep its definitions, we write

```
ERASEDATA TABLE <Table_Name>
```

Same old table, same old example

```
ERASEDATA TABLE Cars
```

Exiting Jatabase

```
BYE
```

**More to be added in Version 0.3*

Roadmap

- ## Version 0.1

+ Basic Functionality
+ Dynamic Tables
+ Ordered Fetching
+ Custom Exceptions

- ## Version 0.2

+ Command Line Interface
+ Table and Row Erasing
+ Table Information
+ TAMPER Statements
+ More Custom Exceptions
+ Table Limit increased from 64 to 128

- ## Version 0.3 (Planned)

+ Custom Table Sizes
+ ADDCB and ADDCS, (CB = Column (in) Between, CS = Column (at) Start)
+ Column Primary Key, Suffix and Prefix
+ IF Statements
+ More Custom Exceptions

Journey

- September 17 2022 : Idea noted at FYP meeting
- September 21 2022 : Work starts
- September 22 2022 : Jatabase 0.1 Ready
- September 22 2022 : Project showed to Dr. Umair (Professor for Game Development)
- September 23 2022 : Jatabase 0.2 released
- September 23 2022 : Jatabase Manual released.

Mentors

If you feel that this project can go to grander heights, please feel free to email me at the address given below. Your suggestion/advice will be of highest value

- ## Mentors List

There are currently 0 mentors sworn into this project

Citations

1. Black Geometric Picture | Name : Geo no.2 | Author : 3psilon | Source : wall.alphacoders.com

End?