PeachyDB: a miniature relational database

- Author: Fang Han
- 2019.11
- Database Systems @ NYU Courant

TABLE OF CONTENTS

SPECIAL INSTRUCTIONS FOR GRADERS

- step 0: unzip the `.rpz` file
- step 1: put all **input data** files under `input/`
- step 2: put the file containin test queries under `input/`
- step 3: open `input/input_pipe`, change the 2nd line to match the name of the query file above
- step 4: at root dir, run `./run.sh`
- After the above steps, find outputs under `output/`

QUERIES SUPPORTED

1. showtables
2. showschema
3. quit
4. inputfromfile
5. outputtofile
6. select
7. project
8. join
9. concat
10. sort
11. count
12. sum
13. avg
14. countgroup
15. sumgroup
16. avggroup
17. movavg
18. movsum
19. hash
20. btree

SETUP

compile and run with maven

- download repo

```
$ git clone https://github.com/TakaiKinoko/PeachyDB.git
```

- compile

```
$ cd PeachyDB
$ mvn compile
```

- build jar

```
$ mvn package
```

- run interactively

```
$ java -cp target/peachyDB-1.0.jar Entry
```

- exit the database

  type `quit` when the database is running.

run with shell script

- `./run.sh` at root
- this will feed all query lines from `input/handout` to the database and direct **stdout** to `output/fh643_AllOperations`

DOCUMENTATION

table naming convention

- has to start with an alphabetic letter
- syntax using regular expression: `([a-zA-Z]+(.)*)`
- derivative tables:
  - **definition**: tables that are built 'on top of' another (more than one) existing table
  - to differentiate the derivative table columns from its parent(s), it's column names have the format of `<table_name>_<column_name>`
  - queries on the derivative tables should make sure that the columns are addressed according to the rule above

I/O

read from file

- syntax: `<table_name> := inputfromfile(<filepath>)`
- implementation: under `src/io/IO.java`
- note:
  i. a `<filepath>` must be assigned to a `<table_name>`
  ii. the database at default tries to read files from the `/input` folder. So `<filepath>` should be the relative path from `/input` to the file
  iii. reading in a new file will create a new table.
  iv. a **truncated view** of the table will be printed out to StdOut once data has been read in successfully, for example:

```
reading from file: input/sales2.txt into table: S...

+----------+---------+---------+------+------+------+-------------+
| saleid   | I       | C       | S    | T    | Q    | P           |
+----------+---------+---------+------+------+------+-------------+
| 3506     | 13517   | 16566   | 45   | 73   | 19   | expensive   |
| 78345    | 10528   | 4745    | 20   | 73   | 23   | supercheap  |
| 79991    | 6715    | 707     | 75   | 41   | 34   | expensive   |
| 90466    | 6697    | 8397    | 83   | 92   | 16   | outrageous  |
| 22332    | 9639    | 2435    | 29   | 17   | 31   | moderate    |
| 95047    | 11877   | 2020    | 44   | 79   | 29   | supercheap  |
| 48867    | 12387   | 15274   | 98   | 76   | 35   | supercheap  |
| 22220    | 10650   | 5746    | 57   | 73   | 24   | outrageous  |
| 53696    | 9958    | 11849   | 85   | 16   | 9    | supercheap  |
| 34328    | 11376   | 4042    | 50   | 66   | 44   | supercheap  |

  ...        ...       ...       ...    ...    ...    ...
```

```
| 62617    | 10689   | 15710   | 3    | 73   | 29   | supercheap   |
| 74088    | 6099    | 14086   | 37   | 95   | 44   | moderate     |
| 66449    | 10137   | 2465    | 41   | 73   | 31   | cheap        |
| 11662    | 9096    | 19072   | 6    | 16   | 21   | supercheap   |
| 33022    | 6259    | 5746    | 54   | 11   | 44   | supercheap   |
| 86141    | 10713   | 5746    | 71   | 73   | 4    | outrageous   |
| 64366    | 8775    | 18198   | 43   | 61   | 49   | supercheap   |
| 41918    | 10898   | 18816   | 61   | 92   | 18   | moderate     |
| 43539    | 8229    | 16589   | 14   | 92   | 47   | supercheap   |
| 2356     | 8909    | 14012   | 32   | 82   | 24   | supercheap   |
+----------+---------+---------+------+------+------+--------------+
Number of entries: 100000

Time cost: 0.1450 seconds
```

- example: `inputfromfile(sales1.txt)`, where `sales1.txt` is stored inside `/input`

write table to file

- syntax: `outputtofile(<table>, <filename>)`
- implementation: under `src/io/IO.java`
- note:
  1. the database at default tries to save files to the `/output` folder.
  2. **PrettyPrinter** (see `/src/util/PrettyPrinter.java`) is used to format the output table.
  3. sample pretty-printed result:

```
+-----------------------+-----------------------+
| groupby_pricerange    | avg_qty               |
+-----------------------+-----------------------+
| cheap                 | 20.546875             |
|-----------------------|-----------------------|
| expensive             | 24.954545454545453    |
|-----------------------|-----------------------|
| moderate              | 22.384615384615383    |
|-----------------------|-----------------------|
| outrageous            | 23.717047451669597    |
|-----------------------|-----------------------|
| supercheap            | 26.10126582278481     |
+-----------------------+-----------------------+
Number of entries: 5
```

algebraic

select

- syntax: `<target_table> := select(<from_table>, <condition1> [and/or <condition2>])`
- the `[and/or <condition2>]` part is optional, which means this select operation takes one or two conditions
- svntax of the condition: `(Column | Constant) [+|-|*|/ Constant] (< | <= | > | >= | != |=) (Column | Constant) [+|-|*|/ Constant])`
- within each condition, the `[+|-|*|/ Constant]` part is optional
- implemented in `src/algebra/Select.java`
- entries selected will be deep copy from the source table
- if a column within the conditions is indexed upon (by either Hash or BTree), the index will be used to perform selection

project

- syntax: `<target_table> := project(<from_table>, <col1>, ..., <coln>)`
- implemented in `src/algebra/Project.java`
- acturally fulfilled by the function `projectTable` in `src/db/Database.java`
- columns selected will be **shallow copy** (pointer) of the source table

join

- syntax: `<target_table> := join(<table1>, <table2>, <condition1> [and/or <condition2>])`
- the `[and/or <condition2>]` part is optional, which means this join operation takes one or two conditions
- svntax of the condition: `<table name1>.<column_name1> ([+|-|*|/] <constant1>) [>|<|!=|=|>=|<=] <table_name2>.<column_name2> [+|-|*|/] <constant2>`

- within each condition, the `([+|-|*|/] <constant>)` part is optional
- implemented in `src/algebra/Join.java`

concat

- syntax: `<target_table> := concat(<table1>, <table2>)`
- implemented in `src/algebra/Concat.java`
- acturally fulfilled by the function `concatTables` in `src/db/Database.java`

sort

- syntax: `<target_table> := sort(<from_table>, <col1>, ..., <coln>)`
- implemented in `src/util/Sort.java`

aggregate

count

- syntax: `<to_table> := count(<from_table>, <column_name>)`
- implemented in `src/aggregation/Aggregate.java`

sum

- syntax: `<to_table> := sum(<from_table>, <column_name>)`
- implemented in `src/aggregation/Aggregate.java`

avg

- syntax: `<to_table> := avg(<from_table>, <column_name>)`
- implemented in `src/aggregation/Aggregate.java`

countgroup

- count the number of entries of a column from a table grouped on an ordered list of columns serving as grouping conditions
- syntax: `<to_table> := countgroup(<from_table>, <column_name>, <groupby_col1>, ..., <groupby_coln>)`
- implemented in `src/aggregation/GroupAgg.java`
- based on internal method `groupby` implemented in `src/aggregation/GroupAgg.java`

sumgroup

- compute the sum of a column from a table grouped on an ordered list of columns serving as grouping conditions
- syntax: `<to_table> := sumgroup(<from_table>, <column_name>, <groupby_col1>, ..., <groupby_coln>)`
- implemented in `src/aggregation/GroupAgg.java`
- based on internal method `groupby` implemented in `src/aggregation/GroupAgg.java`

avggroup

- compute the average of a column from a table grouped on an ordered list of columns serving as grouping conditions
- syntax: `<to_table> := avggroup(<from_table>, <column_name>, <groupby_col1>, ..., <groupby_coln>)`
- implemented in `src/aggregation/GroupAgg.java`
- based on internal method `groupby` implemented in `src/aggregation/GroupAgg.java`

moving aggregates

moving average

- syntax: `<toTable> := movavg(<fromTable>, <col>, <window_len>)`
- implemented in `src/aggregation/Moving.java`
- fulfilled by private internal method `apply` within `src/aggregation/Moving.java`

moving sum

- syntax: `<toTable> := movsum(<fromTable>, <col>, <window_len>)`
- implemented in `src/aggregation/Moving.java`
- fulfilled by private internal method `apply` within `src/aggregation/Moving.java`

index

hash

- syntax: `Hash(<table>, <column>)`
- implemented in `src/index/Hash.java` through Java's native `HashMap` class

btree

- syntax: `Btree(<table>, <column>)`
- implemented in `src/index/Btree.java`
- Btree implementation: `src/btree`

utility

quit

- syntax: `quit` or `Quit`
- implemented in `src/io/QueryParser.java`

show tables

- syntax: `showtables()`
- implemented in `src/db/Database.java`
- sample output:

```
+-----------+----------+
| Table     | Size     |
+-----------+----------+
| R2        | 900      |
| R         | 1000     |
| S         | 100000   |
| T         | 3642     |
| T2prime   | 391      |
| T1        | 391      |
| T2        | 391      |
| R1        | 900      |
| T3        | 391      |
+-----------+----------+
```

show schemas

- syntax: `showschema()`
- implemented in `src/db/Database.java`
- sample output:

```
+---------+------------------------------------------------------------------+
| Table   | Schema                                                           |
+---------+------------------------------------------------------------------+
| R       | saleid | itemid | customerid | storeid | time | qty | pricerange |    |
| S       | saleid | I | C | S | T | Q | P |                                   |
+---------+------------------------------------------------------------------+
```

FEATURES

Pretty-Printer

- implemented in `src/util/PrettyPrinter.java`

STATISTICS

- line counts using: `$ find . -name '*.java' | xargs wc -l`

```
     232 ./src/main/java/aggregation/GroupAgg.java
     151 ./src/main/java/aggregation/Moving.java
      85 ./src/main/java/aggregation/Aggregate.java
      61 ./src/main/java/util/Sort.java
     218 ./src/main/java/util/PrettyPrinter.java
     102 ./src/main/java/util/GroupKey.java
     159 ./src/main/java/util/Cond.java
     142 ./src/main/java/util/Utils.java
      35 ./src/main/java/util/SortGroupKeyMap.java
     185 ./src/main/java/io/IO.java
```

```
   218 ./src/main/java/io/QueryParser.java
   186 ./src/main/java/parser/Parser.java
    18 ./src/main/java/btree/BTKeyValue.java
  1019 ./src/main/java/btree/BTree.java
    11 ./src/main/java/btree/BTIteratorIF.java
    26 ./src/main/java/btree/BTException.java
    61 ./src/main/java/btree/BTNode.java
    42 ./src/main/java/btree/SimpleFileWriter.java
    58 ./src/main/java/db/DynamicTable.java
   187 ./src/main/java/db/Table.java
   380 ./src/main/java/db/Database.java
    39 ./src/main/java/index/BTTestIteratorImpl.java
    84 ./src/main/java/index/Btree.java
    65 ./src/main/java/index/Hash.java
    65 ./src/main/java/index/BtreeKey.java
    71 ./src/main/java/Entry.java
   354 ./src/main/java/algebra/Join.java
   587 ./src/main/java/algebra/Select.java
    55 ./src/main/java/algebra/Project.java
    34 ./src/main/java/algebra/Concat.java
  4930 total
```