Implement an in-memory database that has the following functions. We'll be looking for your code to meet our functionality & performance requirements, be clear & easy to understand and be resilient to edge cases. Use libraries at will (but not database-like ones or actual databases). Use Google/Stack Overflow/online references at will as well.

The database should be a command line program that reads values from STDIN line by line and executes the functions as they happen. Please also include a README explaining how to run your program.

The name and value will be strings with no spaces in them..

Functions:

`SET [name] [value]`
Sets the name in the database to the given value

`GET [name]`
Prints the value for the given name. If the value is not in the database, prints `NULL`

`DELETE [name]`
Deletes the value from the database

`COUNT [value]`
Returns the number of names that have the given value assigned to them. If that value is not assigned anywhere, prints `0`

`END`
Exits the database

The database must also support transactions:

`BEGIN`
Begins a new transaction

`ROLLBACK`
Rolls back the most recent transaction. If there is no transaction to rollback, prints `TRANSACTION NOT FOUND`

`COMMIT`
Commits *all* of the open transactions

Performance Requirements:
The points in this section are goals for the performance of the solution.

- Aim for `GET`, `SET`, `DELETE`, and `COUNT` to all have a runtime of less than $O(log\ n)$, if not better (where n is the number of items in the database).
- The memory usage of the database shouldn't be doubled for every transaction.

Minimum Requirements:

The points in this section are the minimum requirements that a solution must meet in order to be scheduled for a live evaluation of the Tech Assessment.

- The first 3 test cases (that are outlined on the last page of this prompt) must pass
- Neither an actual database nor a database library is used
- The basic commands `GET`, `SET`, `DELETE`, and `COUNT` are implemented per the spec. They take the correct number of arguments and function properly.

Example #1

```
>> GET a
NULL
>> SET a foo
>> SET b foo
>> COUNT foo
2
>> COUNT bar
0
>> DELETE a
>> COUNT foo
1
>> SET b baz
>> COUNT foo
0
>> GET b
baz
>> GET B
NULL
>> END
```

Example #2

```
>> SET a foo
>> SET a foo
>> COUNT foo
1
>> GET a
foo
>> DELETE a
>> GET a
NULL
>> COUNT foo
0
>> END
```

Example #3

```
>> BEGIN
>> SET a foo
>> GET a
foo
>> BEGIN
>> SET a bar
>> GET a
bar
>> SET a baz
>> ROLLBACK
>> GET a
foo
>> ROLLBACK
>> GET a
NULL
>> END
```

Example #4

```
>> SET a foo
>> SET b baz
>> BEGIN
>> GET a
foo
>> SET a bar
>> COUNT bar
1
>> BEGIN
>> COUNT bar
1
>> DELETE a
>> GET a
NULL
>> COUNT bar
0
>> ROLLBACK
>> GET a
bar
>> COUNT bar
1
>> COMMIT
>> GET a
bar
>> GET b
baz
>> END
```