

# CSC3170 FINAL PROJECT

## -- OPTION 3

---

TEAM 21

李珈祺, 刘起, 连珈玮, 杨亮, 王茗萱,

Darren Boesono, Yohanes James

# PRESENTATION OUTLINE

---

- Introduction
- Project Design Logic
- Major Data Structures
- Test Samples & Implementation Demo
- Additional Features & GUI
- Summary & Future Improvement

# INTRODUCTION

---



# OVERVIEW

---

- We choose option 3 as our final project.
- In this project, we will write a miniature relational database management system (DBMS) that stores data *tables*, where a table consists of some number of labeled *columns* of information. Our system will include a database *query language* similar to SQL to extract information from these tables. Extra features and robustness support are provided in our database system.
- We will mainly use C++ to implement our code. Therefore, we do not adopt the original backbone.

# FUNCTIONS WE ACHIEVED

---

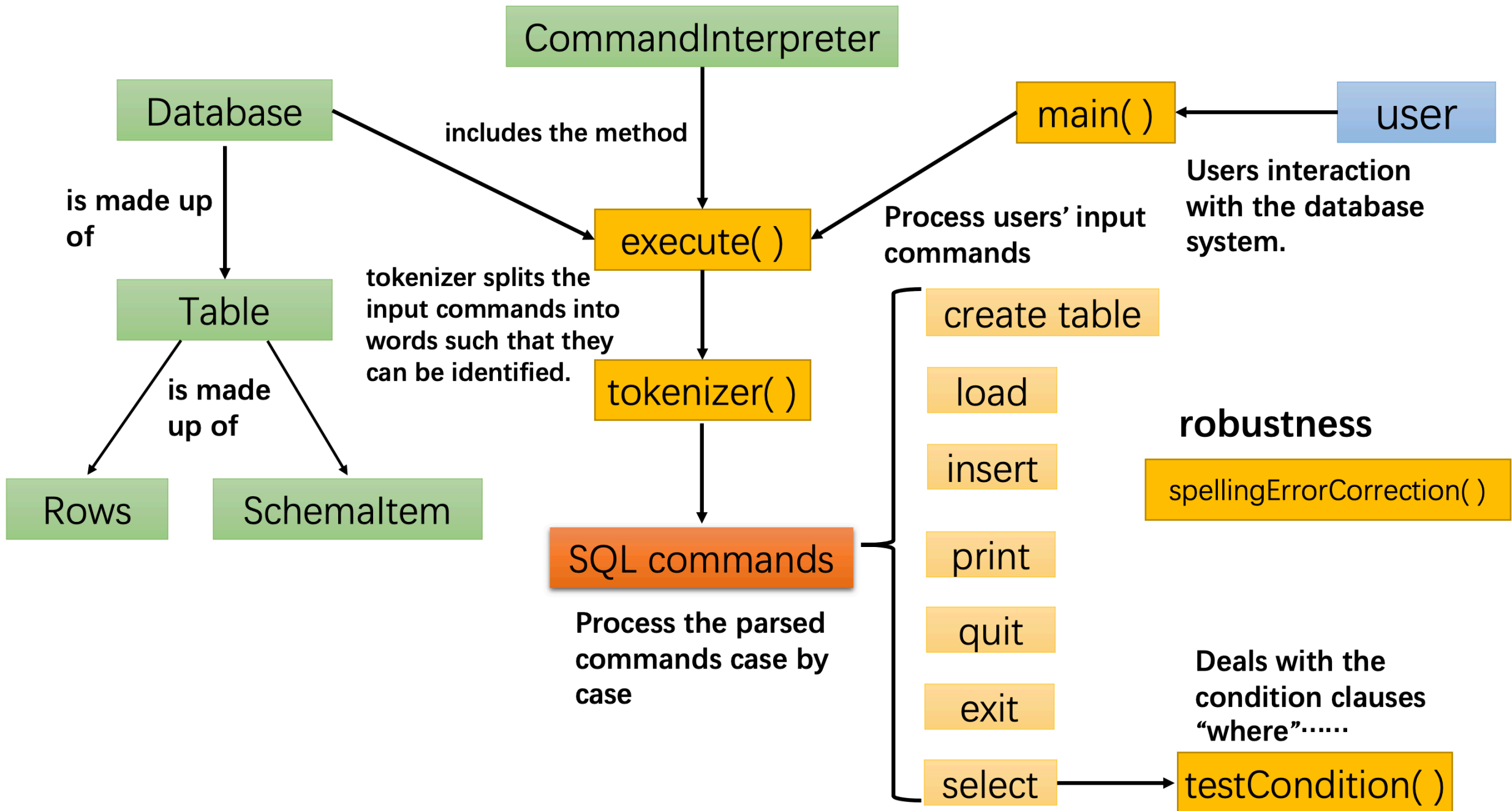
- Support a database *query language* similar to SQL
  - **create table (as...)** : create an empty table with the given name
  - **Load** : load data from the file *name.db* to create a table name *table*
  - **Store** : store data from the table *name* to the file *table.db*
  - **insert into** : add a new row to the given table
  - **Print** : print all rows of the table with the given name
  - **quit (exit)** : quit the database program
  - **help** : print help messages
  - **select <column(s)> from <table(s)> where <condition(s)>** : extract a new (unnamed) table consisting of the <column(s)> from the given <table(s)> with all rows that satisfy the <condition(s)>



# PROJECT DESIGN LOGIC

---





# MAJOR DATA STRUCTURES

---





# DATA STRUCTURES

---

- To implement the specific database and related methods, we divide it into a number of classes. The specific architecture we will adopt is as follows:
- Row class
- Schemaltem class
- Table class
- CommandInterpreter class
- Database class

# Row CLASS

---

- Serves as the underlying storage unit for information about tables in the database, recording row information. (A row corresponds to a vector variable)
- Methods:
  - `getValues`, `setValues`

students

SID	Lastname	Firstname	SemEnter	YearEnter	Major
101	Knowles	Jason	F	2003	EECS
102	Chan	Valerie	S	2003	Math
103	Xavier	Jonathan	S	2004	LSUnd
104	Armstrong	Thomas	F	2003	EECS
105	Brown	Shana	S	2004	EECS
106	Chan	Yangfan	F	2003	LSUnd

# Schemaltem CLASS

---

- Records tables' schemas. (Similar to row class)
- Methods:
  - getName, getType, getTypeFromString

students

SID	Lastname	Firstname	SemEnter	YearEnter	Major
101	Knowles	Jason	F	2003	EECS
102	Chan	Valerie	S	2003	Math
103	Xavier	Jonathan	S	2004	LSUnd
104	Armstrong	Thomas	F	2003	EECS
105	Brown	Shana	S	2004	EECS
106	Chan	Yangfan	F	2003	LSUnd

# Table CLASS

---

- A data structure that stores tables in a database. It contains three attributes, the rows (Row class) to record the row information, the schema (Schemaltem class) to record the schema, and the database (Database class) to record the database which the table belongs.
- Methods:
  - printOut, saveToFile, loadFromFile, getSchema, insertAt

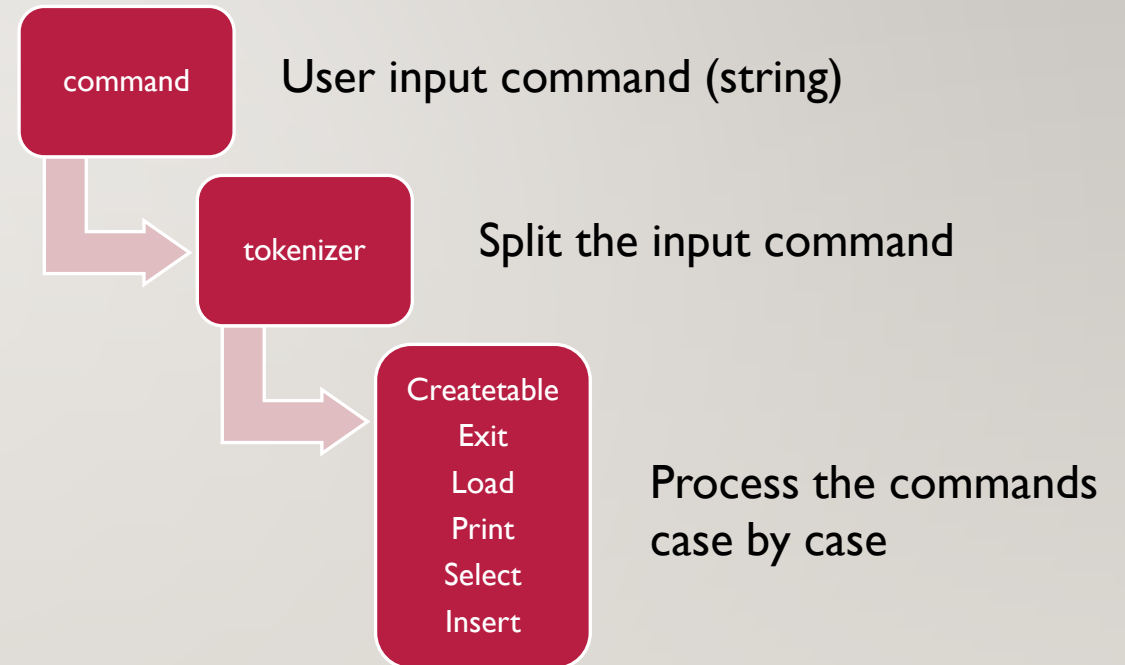
students					
SID	Lastname	Firstname	SemEnter	YearEnter	Major
101	Knowles	Jason	F	2003	EECS
102	Chan	Valerie	S	2003	Math
103	Xavier	Jonathan	S	2004	LSUnd
104	Armstrong	Thomas	F	2003	EECS
105	Brown	Shana	S	2004	EECS
106	Chan	Yangfan	F	2003	LSUnd



# CommandInterpreter CLASS

---

- Used to accept and execute commands. Contains the specific implementation method of the command. (exit, select, help...)
- It first decomposes the command using the token variable, and then implements the operations corresponding to the command.





# Database CLASS

---

- As a whole database, which contains instances of the Table and CommandInterpreter classes as attributes.

## Tables

- Students table
- Enrolled table
- Schedule table

## Methods

- removeTable, execute, switchTable, addTable, setTable, getDatabase

# IMPLEMENTATION DEMO: “LOAD” AND “PRINT”

---

Scenario: A database for CUHK(SZ) to record the data of students and courses

We have provided pre-stored sample tables: students, enrolled, and schedule, that can be directly loaded.

```
Welcome to Team 21's DB! Type SQL commands or 'help' or 'h' to get help, 'quit' or 'q' to exit
Note: All SQL commands should end with a semicolon (;)
```

```
> load students1;
```

```
Loaded students1.db
```

```
> load enrolled1;
```

```
Loaded enrolled1.db
```

```
> load schedule1;
```

```
Loaded schedule1.db
```

```
> print students1;
```

```
Contents of students1:
```

SID	Lastname	Firstname	SemEnter	YearEnter	Major
120030001	Knowles	Jason	F	2020	DSBDT
120030037	Chan	Valerie	S	2020	Math
119050638	Xavier	Jonathan	S	2019	CSC
120045628	Armstrong	Thomas	F	2020	EIE
120090532	Brown	Shana	S	2020	EIE
120032765	Chan	Yangfan	F	2020	CSC

# IMPLEMENTATION DEMO: QUERYING

```
> select * from schedule1 where Dept = 'SDS';
```

Search results:

CCN	Dept	CName	Sem	Year
21228	SDS	data-structures	F	2022
21231	SDS	algorithms	S	2021
21229	SDS	parallel-computing	F	2022
21232	SDS	operating-system	S	2021

```
>
```

```
> select * from schedule1 where Dept = 'SDS' and Year = 2022;
```

Search results:

CCN	Dept	CName	Sem	Year
21228	SDS	data-structures	F	2022
21229	SDS	parallel-computing	F	2022

```
>
```

```
> select Firstname, Lastname, Grade from students1, enrolled1 where CCN = 21228;
```

Search results:

Firstname	Lastname	Grade
Jonathan	Xavier	B
Jason	Knowles	A-
Shana	Brown	A

```
>
```

# IMPLEMENTATION DEMO: CREATE TABLE, INSERT

---

```
Welcome to Team 21's DB! Type SQL commands or 'help' or 'h' to get help, 'quit' or 'q' to exit
Note: All SQL commands should end with a semicolon (;)
> create table department as Dname(string), location(string), capacity(int);
Table department has been created.
> insert into department values MUS, Longgang, 1000;
Insert process completed.
> print department;
Contents of department:
  Dname  location  capacity
-----
    MUS   Longgang    1000
> store department;
Store process completed.
> q;
Bye!
```

We can verify this by starting another instance of the DB and load it into memory.

# ADDITIONAL FEATURES & GUI

---





# ADDITIONAL FEATURES - ROBUSTNESS SUPPORT

---

- Spelling error corrections
  - “Guess” the user’s command if the user gives a wrong one
  - Implementation detail:
    - compare the user input with each of the standard SQL commands (select, create, print...)
    - Function *CommandInterpreter::lcs(string a, string b)* obtains the length of longest common substring between 2 strings
    - the SQL command with *lcs()* value larger than threshold will be the possible input, and give user a hint

```
(base) jiaqi@hx-rs4810gs:~/3170/project-team-21/simple_db/build$ ./simple_db
Welcome to Team 21's DB! Type SQL commands or 'help' or 'h' to get help, 'quit' or 'q' to exit
Note: All SQL commands should end with a semicolon (;)
> loadd students;
Error: Invalid command. Please try again.
Do you want to type in command 'load'?
> paint students;
Error: Invalid command. Please try again.
Do you want to type in command 'print'?
> halp;
Error: Invalid command. Please try again.
Do you want to type in command 'help'?
```

# ADDITIONAL FEATURES - ROBUSTNESS SUPPORT

---

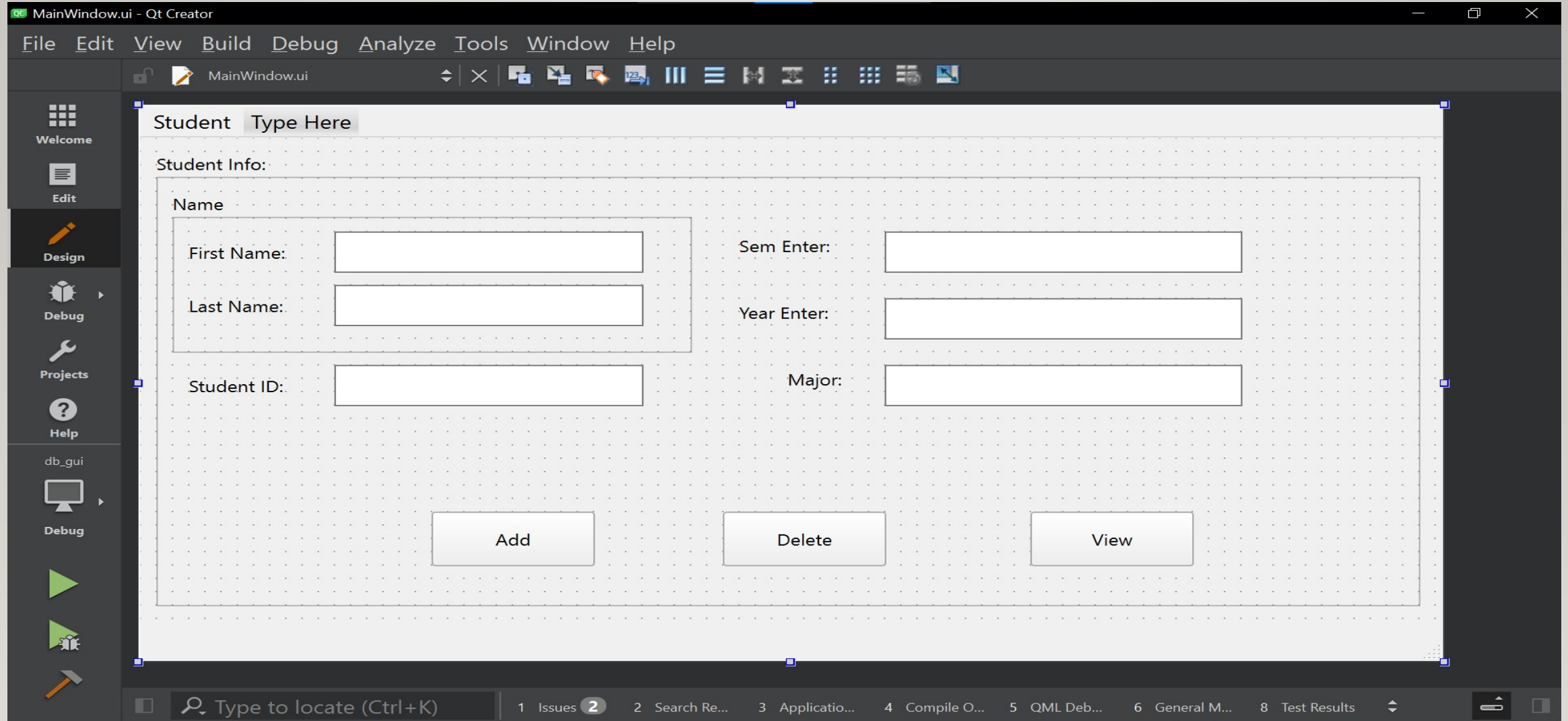
- Identify error cases and post error messages when
  - user operates(select, print, store...) a table that did not exist
  - user “*insert*” values with numbers that does not match the number of columns

# ADDITIONAL FEATURES - OTHER SUPPORTS

---

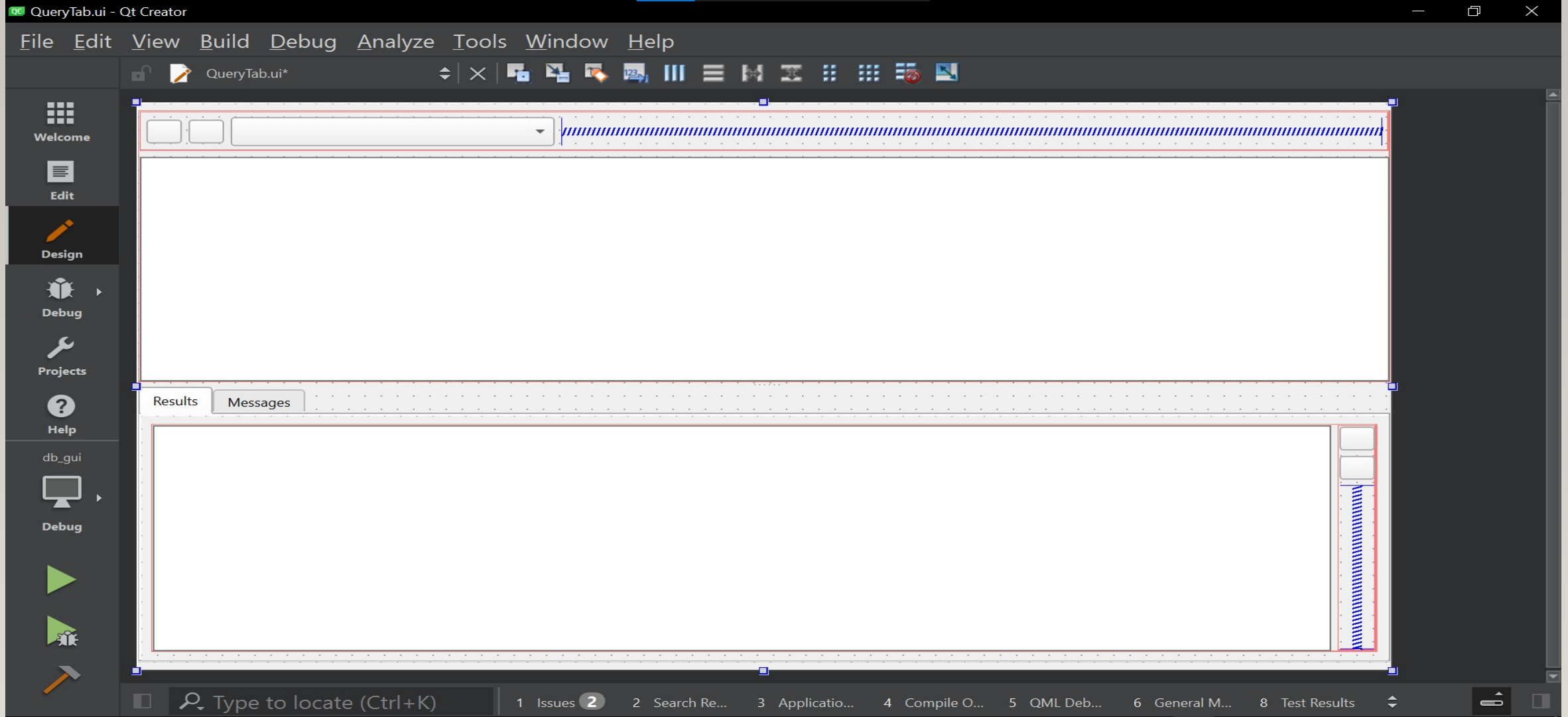
- Support “*comments*” inputs ( */\* ... \*/* )
- Support “*select*” command with conditional clause (*where...*)
- Support “*select*” multiple features from multiple tables
- Beautify the “*print*” outputs to make the tables tidy and aligned
- Standardize the outputs to keep consistent with the source UCB project
- ...

# GUI – LOAD STUDENTS





# GUI – QUERY TAB





# SUMMARY & FUTURE IMPROVEMENT

---



# SUMMARY & FUTURE IMPROVEMENT

---

- Have a deeper understanding of a database system by implementing one ourselves
  - Knowledge of natural inner join, database components are utilized
- Future improvements
  - Search efficiency improvements
  - GUI / user interaction improvement
  - Support larger-scale databases

# THANK YOU!

---

