Integrated CA OOC, Linear Algebra, Databases

Professors: Aldana Louzan
Francis Adepoju

Students: Robert Szlufik #2020358
Ingrid Menezes Castro #2020341

Summary

Challenges

The first big challenge doing this CA was attached to the cross-module aspect of it. Java, Linear Algebra and Databases by themselves have their own difficulties, but connecting them altogether involves a package of knowledge never before approached by any other CA.

To create the overall structure of the Java code using proper classes, interfaces and methods requires some thinking, coordination between the team and research. Most of the examples and materials found online provided a different overall structure, doing connection, queries and basically the whole project on the JFrame, but besides the fact that those examples were not best practice, we decided to approach this project in a way that it felt more organized and proper to the task intended.

On our initial plan, we thought of aiming our work towards the CLI requirement, as that was apparently simpler and more practical, but the GUI version was thought out and implemented as an option to make the program more user friendly and intuitive.

Coordinating as a team was definitely another big challenge, considering it requires a lot of planning and communication. This was the second OOC project working as a team, but this was by far the most complex project we've done. Many google meets happened to determine who would do what and trace a doable plan.

As far as mathematical challenges are concerned, it was vital to approach them in the right manner. We concluded that a matrix can be represented as a simple array of numbers, in our case they are of the type double. This was crucial to understand, because it allowed us to progress with the task. We concluded that since we can obtain an array of numbers from the user, to be processed, we can manipulate this array in whichever manner we decide. Since we were presented with specific and detailed algorithms to solve systems of equations, all we needed to do is simply apply them to our codebase.

MVC pattern. We have decided to implement an MVC pattern in our code. It is a very popular practice since it allows us to divide application logic from business logic. It also helps to improve flow of the program and increases performance. The main advantage of MVC pattern is to allow developers to work on different parts of the program without colliding and working on the same parts. It also changed the originally planned command line interface to a graphical one. The vast majority of code stayed the same.

Design Decision Rationale

Database

For the database we decided to have two tables: one for the storage of users and another for the results obtained from those users. It was not offered the option for solving equations without an account and all equations were automatically saved to the database and could be retrieved by the Admin.

As for attributes of each table, we decided to assign only those that were necessary to run the program without problems while offering the user and admin an appropriate amount of information that they can retrieve.

Although this database is relatively small, it is easy to add information that does not affect user experience. This approach allows us to predict that this application could scale up easily, and that wouldn't affect its performance.

It is worth mentioning that some attributes play a crucial role in the program. For example, user_id is in fact the primary key in users table and a foreign key in results table. Also, isAdmin attribute defines privileges and options that will be presented to the logged in user (or admin).

User table:

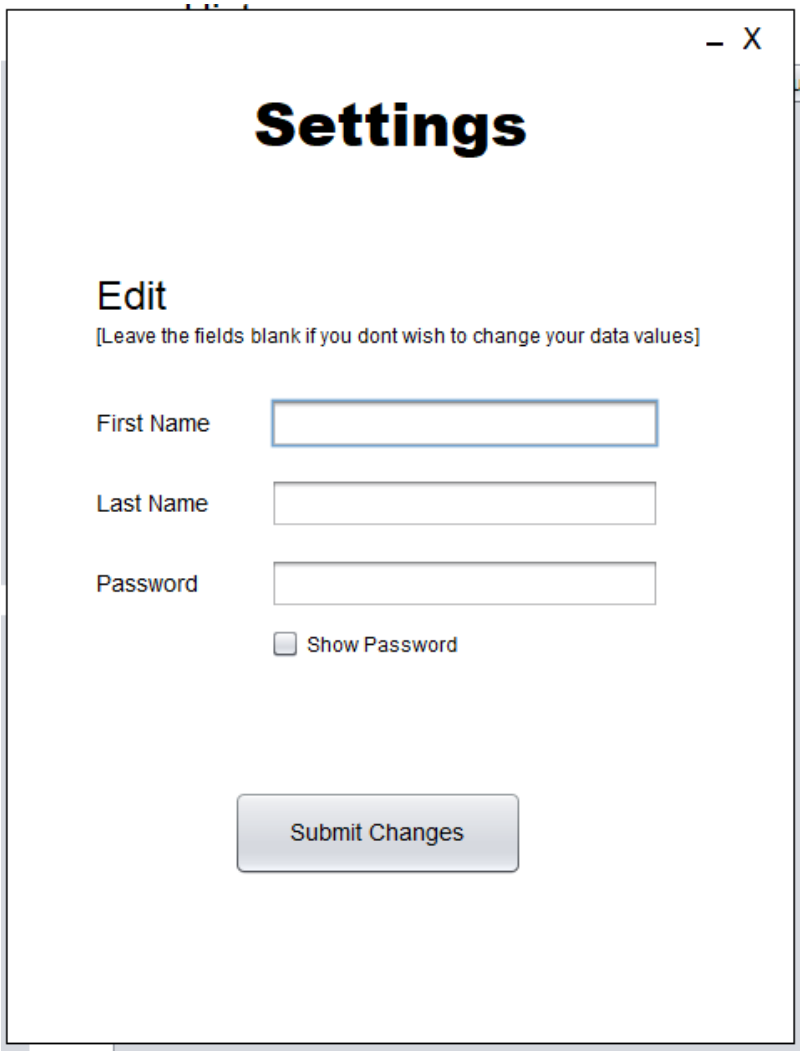
user_id	userName	password	firstName	lastName	isAdmin
1	CCT	Dublin	admin	admin	1
13	test3	test	test	test	0
14	test4	test	test	test	0
16	test6	test	test	test	0
17	test7	test	test	test	0
18	test8	test	test	test	0
19	test9	test	test	test	0
23	test10	test	test	test	0
25	imcastro	pass1234	Ingrid	Castro	0
29	usertest	test	test	test	0
NULL	NULL	NULL	NULL	NULL	NULL

We opted to keep it simple and ask the user for an Username, Password, First and Last name. The user_id was set as unique and auto incremented and Username was set as unique (and could not be changed in settings).

The login screen:

Error handling was made here through the controller to display correct messages for the user while logging in.

Settings screen:

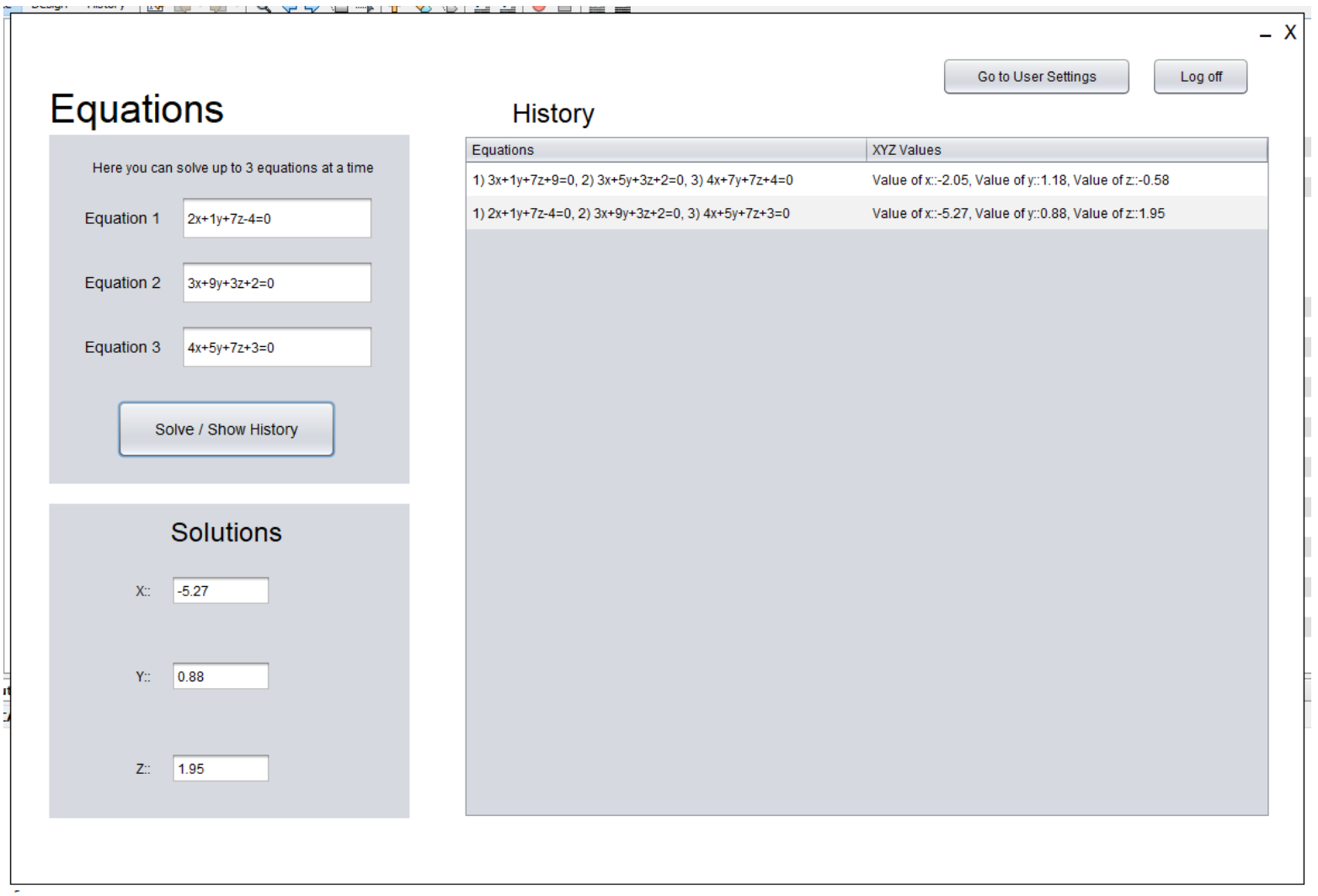


The field username is not present here because it's an unique attribute and we decided to not make it available for editing.

For the second table, named results, we came up with this set of attributes:

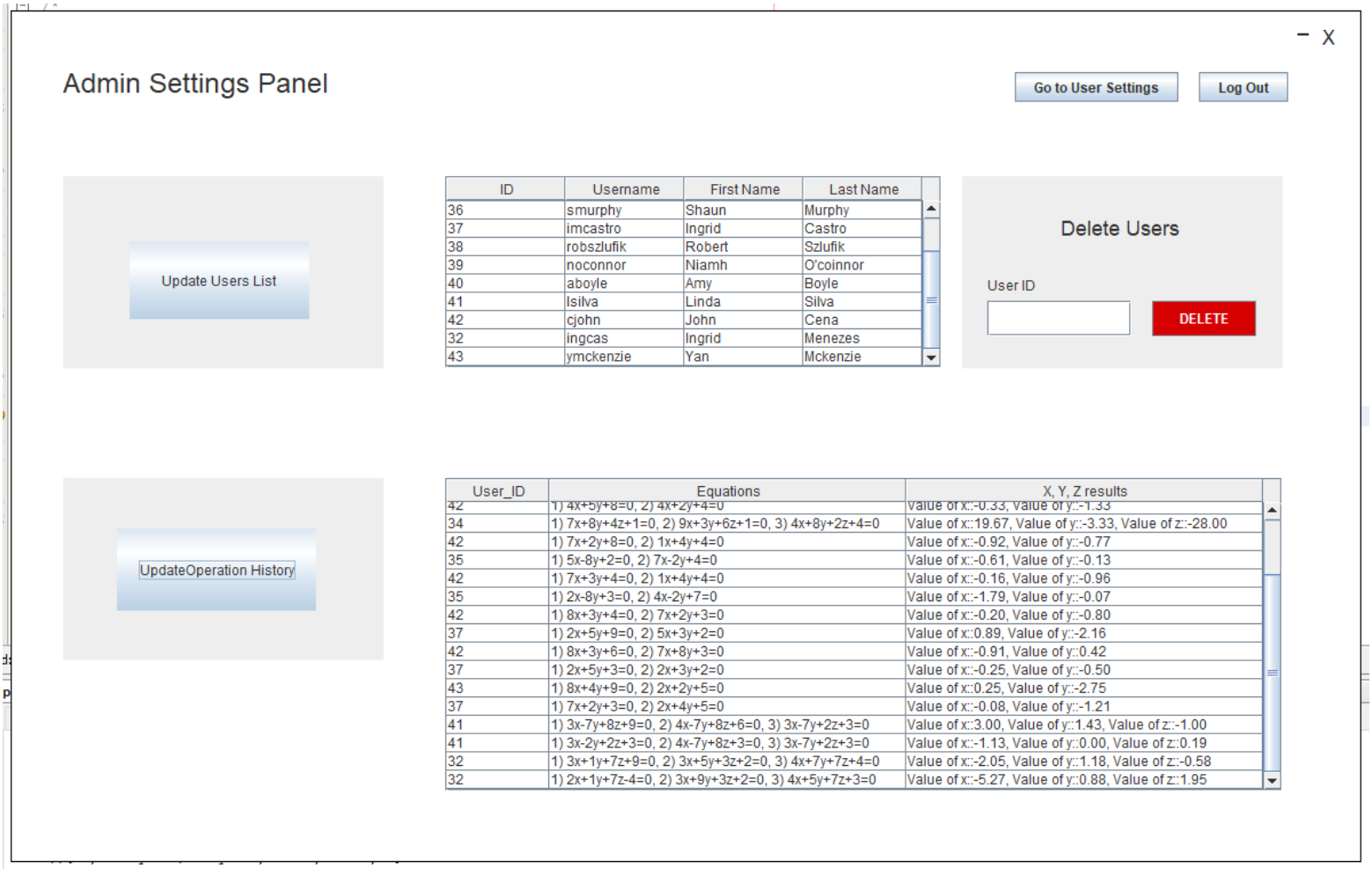
result_id	user_id	setOfEquations	valueOfX	valueOfY	valueOfZ
49	25	2x+5y+6=0;3x+4y+5=0;-	-0.14	-1.14	-
64	25	2x+5y+6=0;2x+7y+3=0;-	-6.75	1.50	-
97	32	3x+1y+7z+9=0;3x+5y+3...	-2.05	1.18	-0.58
98	32	2x+1y+7z-4=0;3x+9y+3...	-5.27	0.88	1.95
NULL	NULL	NULL	NULL	NULL	NULL

result_id is the primary key of this table (auto-incremented and unique), and to solve any partial dependencies user_id is brought up as a foreign key, just so we could identify for the user and admin which equations belong to which user.



In the GUI this data is displayed this way for the users.

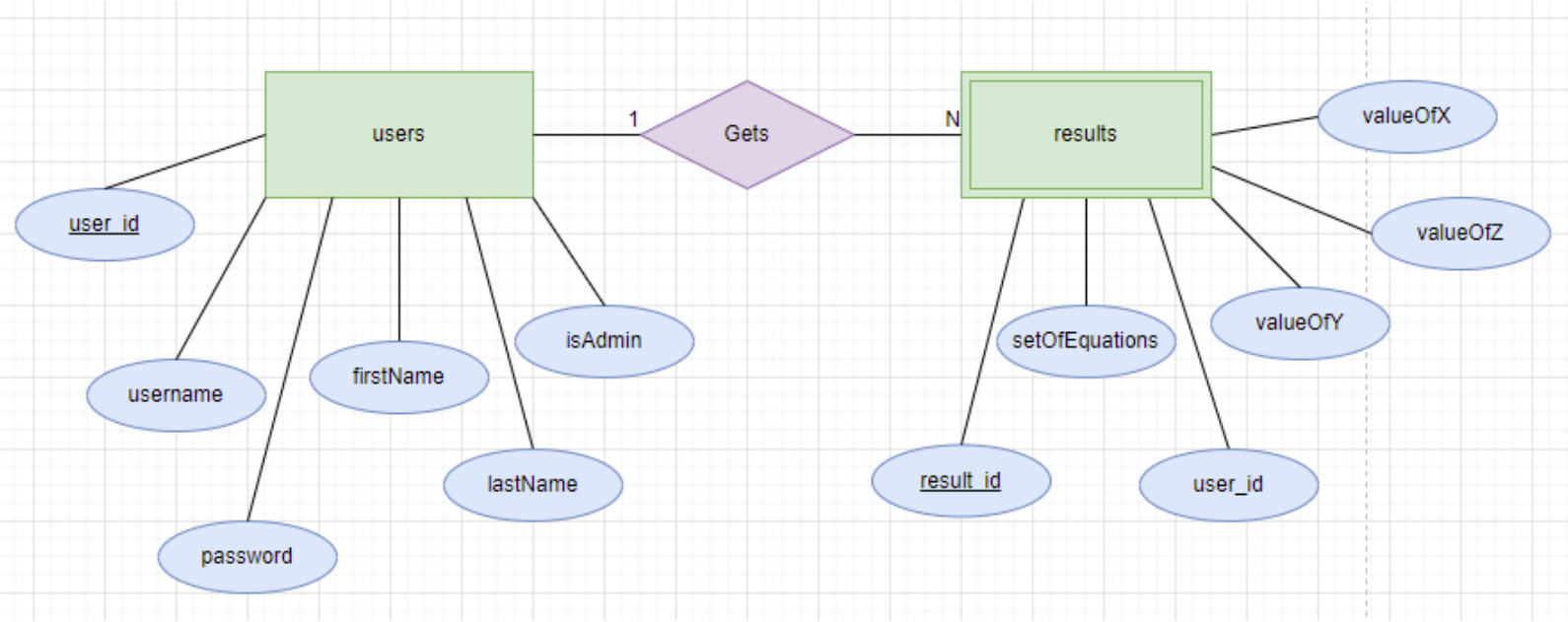
The Admin Screen:



The columns are adjustable in case you need a little bit more space to see the set of equations and it should look like this. To delete users you simply would update the user list and type their id on the space provided on the right and click delete. No confirmation message appears. Users are not able to exclude their accounts, as it was not asked as a feature on the descriptor. Only the admin can do that.

Database Conceptual Design (Chen Notation)

Here we have two entities, results being a weak one (it has a foreign key), and the relationship 1:N (one to many) since a user can do many operations and get many results.

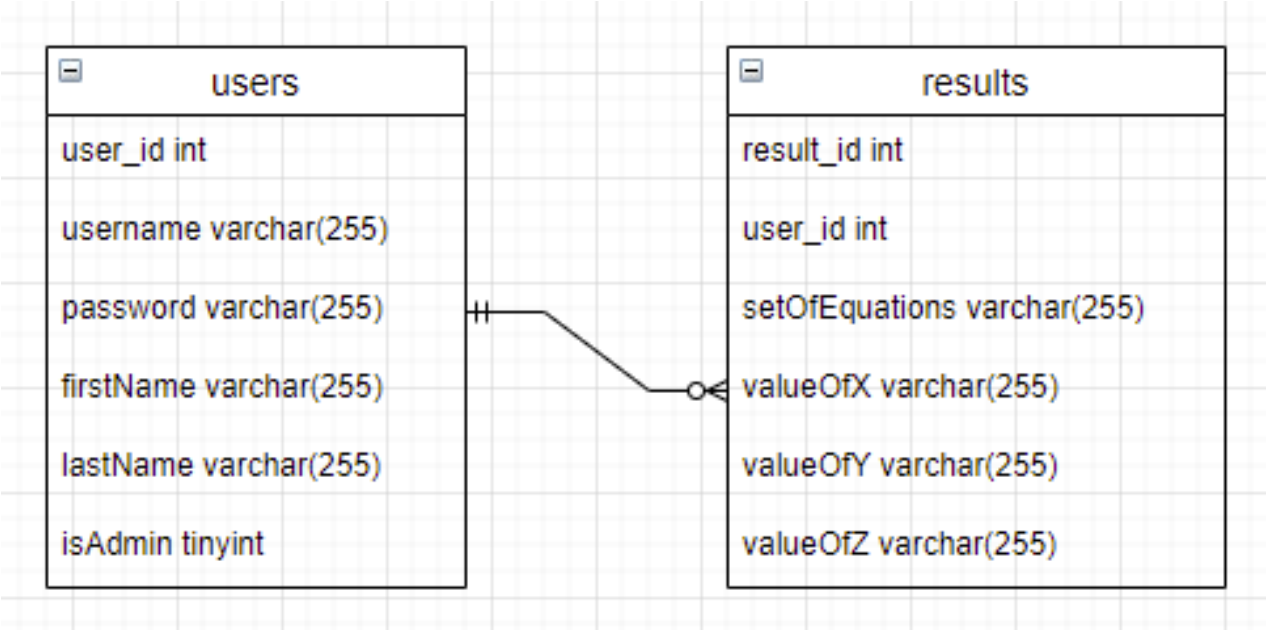


Logical Design

According to the material and classes provided a logical model consists of a little more elaborated display of the data entities and attributes, where the structure of data elements and relationships are provided, with the data types specified in precision and length, but no primary or secondary keys are defined.

We decided to adopt varchar(255) in most of the attributes as a general standard, having more specific data types (int and tinyint) for the IDs and the idAdmin attribute.

The relationship is 1 mandatory to many optionals.



Normalisation

The design was conceived observing the 3NF rule.

- **The 1st. Normal Form says that “no table should contain multiple columns displaying the same info” and “each row should have an unique primary key”.**

user_id	userName	password	firstName	lastName	isAdmin
1	CCT	Dublin	admin	admin	1
13	test3	test	test	test	0
14	test4	test	test	test	0
16	test6	test	test	test	0

result_id	user_id	setOfEquations	valueOfX	valueOfY	valueOfZ
49	25	2x+5y+6=0;3x+4y+5=0;-	-0.14	-1.14	-
50	25	2x+5y+6=0;3x+4y+5=0;-	-0.14	-1.14	-
51	25	2x+5y+6=0;3x+4y+5=0;-	-0.14	-1.14	-
52	25	2x+5y+6=0;3x+4y+5=0;-	-0.14	-1.14	-

As it can be seen all the columns keep one value at a time and have each an unique Primary Key. setOfEquations has all the equations on the same line, separated by “;” so we could display it as a String on our equation history panel (for admin and user).

- **The 2nd. Normal Form says that all partial functional dependency should be removed.**

On our database we chose to detach the results to the users by creating the table results and importing the user_id as foreign key, this way it was still trackable which user did which setOfEquations and obtained results X,Y, Z, but with no partial dependency.

- **The 3rd. Normal Form says there should be no transitive dependencies.**

There are no such values that could cause transitive dependencies, because no change in a non-key column has the power to change another column. If a change on the setOfEquations is made, for example, another record is created with different values of X, Y and Z. The picture above was based on a test with the same values, but the one below illustrates that better:

63	25	2x+5y+6=0;3x+4y+5=0;-	-0.14	-1.14	-
64	25	2x+5y+6=0;2x+7y+3=0;-	-6.75	1.50	-

Physical design: SQL create statements

Create:

```
USE ca2;
GO
DROP TABLE IF EXISTS results;
CREATE TABLE `results` (
  `result_id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `setOfEquations` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `valueOfX` varchar(255) COLLATE utf8_unicode_ci DEFAULT '-',
  `valueOfY` varchar(255) COLLATE utf8_unicode_ci DEFAULT '-',
  `valueOfZ` varchar(255) COLLATE utf8_unicode_ci DEFAULT '-',
  PRIMARY KEY (`result_id`),
  KEY `user_id_idx` (`user_id`),
  CONSTRAINT `user_id` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8
COLLATE=utf8_unicode_ci;

INSERT INTO results (user_id, setOfEquations, valueOfX, valueOfY, valueOfZ)
VALUES (‘user_id’, ‘setOfEquations’, ‘valueOfX’, ‘valueOfY’, ‘valueOfZ’);


USE ca2;
GO
DROP TABLE IF EXISTS users;
CREATE TABLE `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `userName` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `password` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `firstName` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `lastName` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `isAdmin` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `id_UNIQUE` (`user_id`),
  UNIQUE KEY `userName_UNIQUE` (`userName`)
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=utf8
COLLATE=utf8_unicode_ci;

INSERT INTO users (userName, password, firstName, lastName, isAdmin)
VALUES (‘userName’, ‘password’, ‘firstName’, ‘lastName’, 0);
```

Read:

login
SELECT password
FROM users
WHERE userName;

fetchId
SELECT user_id
FROM users
WHERE userName;

fetchAdmin
SELECT * FROM users
WHERE user_id = 1;

fetchUser
SELECT * FROM users
WHERE userName = ‘ ‘;

fetchAllUsers
SELECT * FROM users;";

fetchAllOperations
SELECT * FROM results;";

fetchUserByID
SELECT * FROM users
WHERE user_id = ";

Update:

password
UPDATE users
SET password;

firstName
UPDATE users
SET firstName;

lastName
UPDATE users
SET lastName;

Drop:

deleteUser
DELETE FROM users
WHERE user_id = '';

deleteSolutions
DELETE FROM users
WHERE user_id = '';

Repository link

https://github.com/lngDih/CA2OOC