Kushal Gevaria (kgg5247)
Akshay Pudage (ap7558)

CSCI 621: Database System Implementation
H2 DATABASE
Assignment 2

As a part of this assignment we have created a new feature to support a Email data type. With this addition, users can create a column for emails with the Email data type in a relation and the queries will be validated for a valid email on insertion.

Implementation Steps

1. **Creating custom Email data type class**
   We created a CustomEmailDataType class which implemented CustomDataTypesHandler class. All the abstract methods in this interface were then implemented. The constructor initializes an object for DataType class. We then assign a string name, integer type and SQL Type. Name variable holds the name we want for this new data type, in H2 each data type is assigned an unique integer id and this is stored inside the type variable. The SQL Type variable holds an unique integer to denote this is a Java object, this is used for object serialization and deserialization purposes. The methods which are implemented in this class are described as follows:
   a. **getDataTypeByName:**
      This method takes in a string data type name and returns the DataType object which was instantiated in the constructor.
   b. **getDataTypeById:**
      As the name implies, it returns the DataType object which was instantiated in the constructor by taking an integer argument. If the value for EMAIL in enum Value matches with this integer, the email DataType object is returned.
   c. **getDataTypeOrder:**
      In H2, each data type is assigned a order which is an integer in addition to unique integer value. This method return this order number given the data type integer id.
   d. **convert:**
      This method takes in a Value object, which is the base class for all data types and converts the object to a Email data type. This is mainly used for conversion from one data type to another.

e. **getDataTypeClassName:**
   Returns the custom data type class name, CustomEmaiDataType in this case, given its unique integer id.
f. **getTypeIdFromClass:**
   Returns the integer id for the email data type given its class identifier.
g. **getValue:**
   This method returns a generic Value object with the email data given the custom data type identifier and data.
h. **getObject:**
   Converts generic Value object to email data type class.
i. **supportsAdd:**
   Returns false as the email data type does not support arithmetic operation
j. **getAddProofType:**
   Returns type identifier for the email data type object.

2. **Creating Email entity**
   As H2 uses Java objects for each data types, we need to create an Email entity class which implements Serializable interface as it can be serialized to disk and deserialized back into an Email object. Following methods are a part of the Email entity:
   a. **verifyAndInitializeEmail:**
      This method is called by the constructor when the Email entity is initialized. This method takes in a string containing the possible email and applies regular expression to check if it is a valid email address, if yes we store the string inside this entity in an instance variable, if not, we throw an exception which displays the invalid email error code to the user.
   b. **equals:**
      This method is part of the Serializable interface which simply checks if an object is equal to the email object by checking if the two email addresses match.
   c. **toString:**
      Simply returns the email address as string.
   d. **hashCode:**
      We assign the Email object a randomly generated hash code. This can be used for checking if two Email objects are equal. Two email objects are equal if they have the same hashcode.

3. **Creating a unique Error code for data type**
   We need to create a unique static constant which acts as an error code for email data type. This error code is displayed to the user when an exception is thrown when the email address provided by the user is invalid. This error code can be further used by developers to troubleshoot problems. This is stored in ErrorCode class.

4. **Adding a message for error code**
   We add a message for error code for the unique error code value defined above in a properties file named _messages_en.prop. The error for this data type says '*Invalid email address format. Please enter the correct email address.*'

5. **Assigning unique type identifier**
   Each value or data type in H2 has a unique integer identifier associated with it. This needs to be defined in the Value class. We create a static constant for email and also add a type count. We also increment the value of the type count variable which is used for keeping a track of total number of data types. We also add an entry for email in getOrder method, which again returns a unique order number for the email data type.

6. **Creating specialized Value object for Email**
   We need to extend the generic Value object and create a specialized Value object for email which extends this generic Value object. This object is responsible for instantiating the Email entity object and storing the email address in it. This instantiation is done in the constructor. Following methods are a part of this class:
   a. **get:**
      This method creates specialized value object for email from the entity and returns the object. Basically, it creates a new object for the class it is in by using the Email entity and returns it.
   b. **getSQL:**
      It returns the email address for displaying it in the SQL expression.
   c. **getType:**
      Returns unique type identifier for email data type.
   d. **getPrecision:**
      Returns 0 as email object is not a float or double object.
   e. **getDisplaySize:**
      Returns display size in terms of number of characters.

f. **getString:**
Returns a string representation of the email address from the Email entity which is contained within the value object.

g. **getObject:**
Returns the Email entity object which is stored in this specialized value object.

h. **set:**
Sets the value as a parameter in a prepared statement. In this case, we set the object as a type of Java object.

i. **compareTypeSafe:**
Compares email from one value object with the email from another value object. Basically does a lexicographic comparison of the two email addresses using Java's built-in String compareTo method.

j. **hashCode:**
Assigns unique randomly generated hashcode for the object.

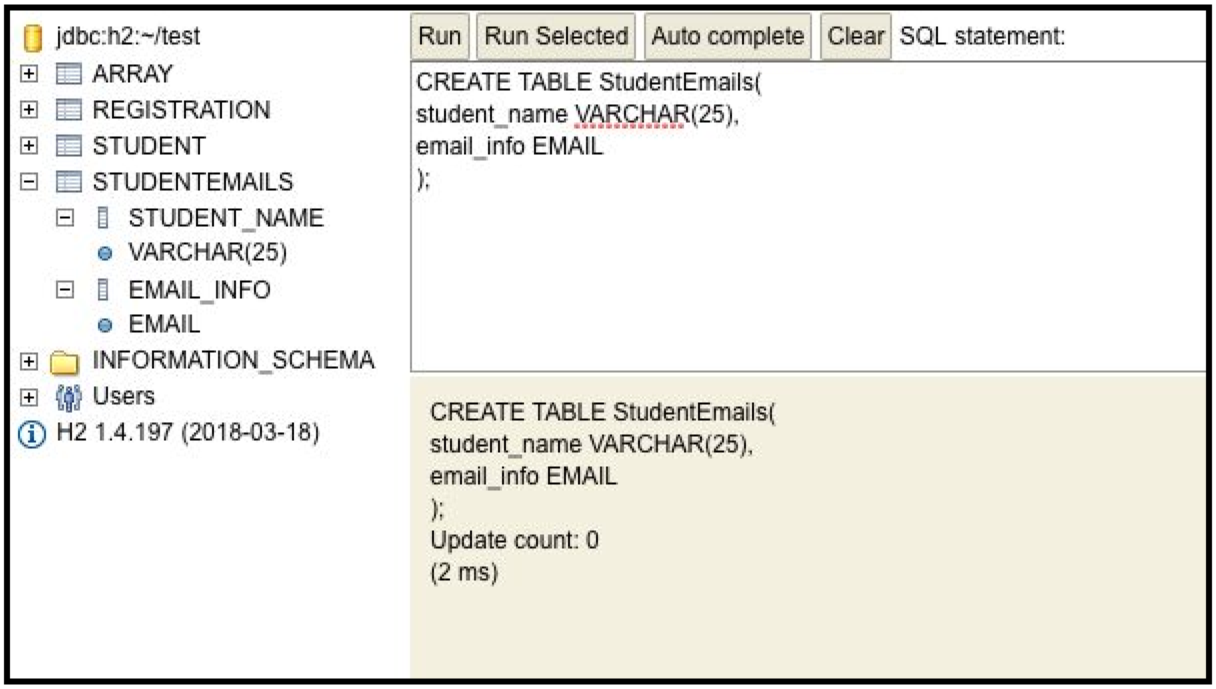
k. **equals:**
Checks if the email address within another Value object is the same as the email contained within this Value object.

l. **convertTo**
Converts the object to either bytes by serializing (for storing purposes), to string or to a Java object.

Kushal Gevaria (kgg5247)
Akshay Pudage (ap7558)

Demo Screenshots

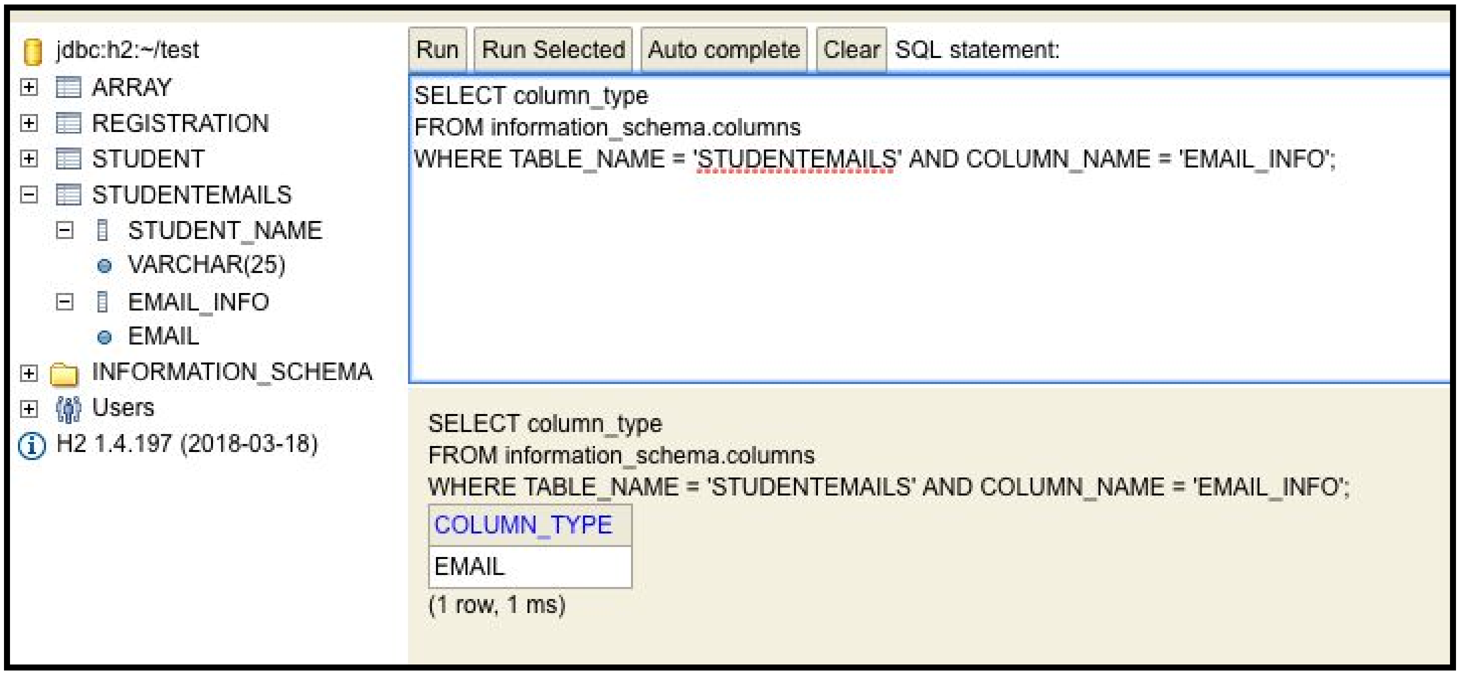
Creating a relation with custom Email data type



```
CREATE TABLE StudentEmails(
student_name VARCHAR(25),
email_info EMAIL
);
```

```
CREATE TABLE StudentEmails(
student_name VARCHAR(25),
email_info EMAIL
);
Update count: 0
(2 ms)
```

Viewing the column type from the system catalog



```
SELECT column_type
FROM information_schema.columns
WHERE TABLE_NAME = 'STUDENTEMAILS' AND COLUMN_NAME = 'EMAIL_INFO';
```

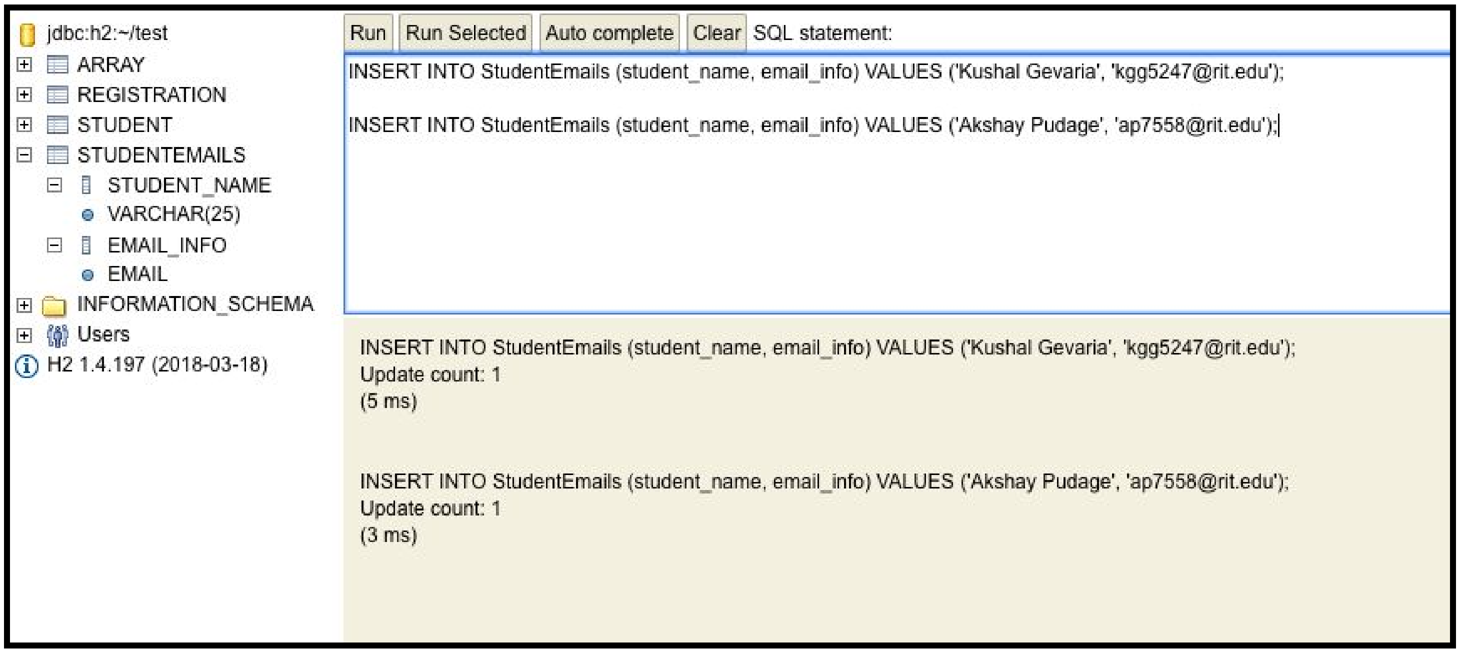
```
SELECT column_type
FROM information_schema.columns
WHERE TABLE_NAME = 'STUDENTEMAILS' AND COLUMN_NAME = 'EMAIL_INFO';
```

COLUMN_TYPE
EMAIL

(1 row, 1 ms)

Kushal Gevaria (kgg5247)
Akshay Pudage (ap7558)

Inserting valid values inside the newly created relation



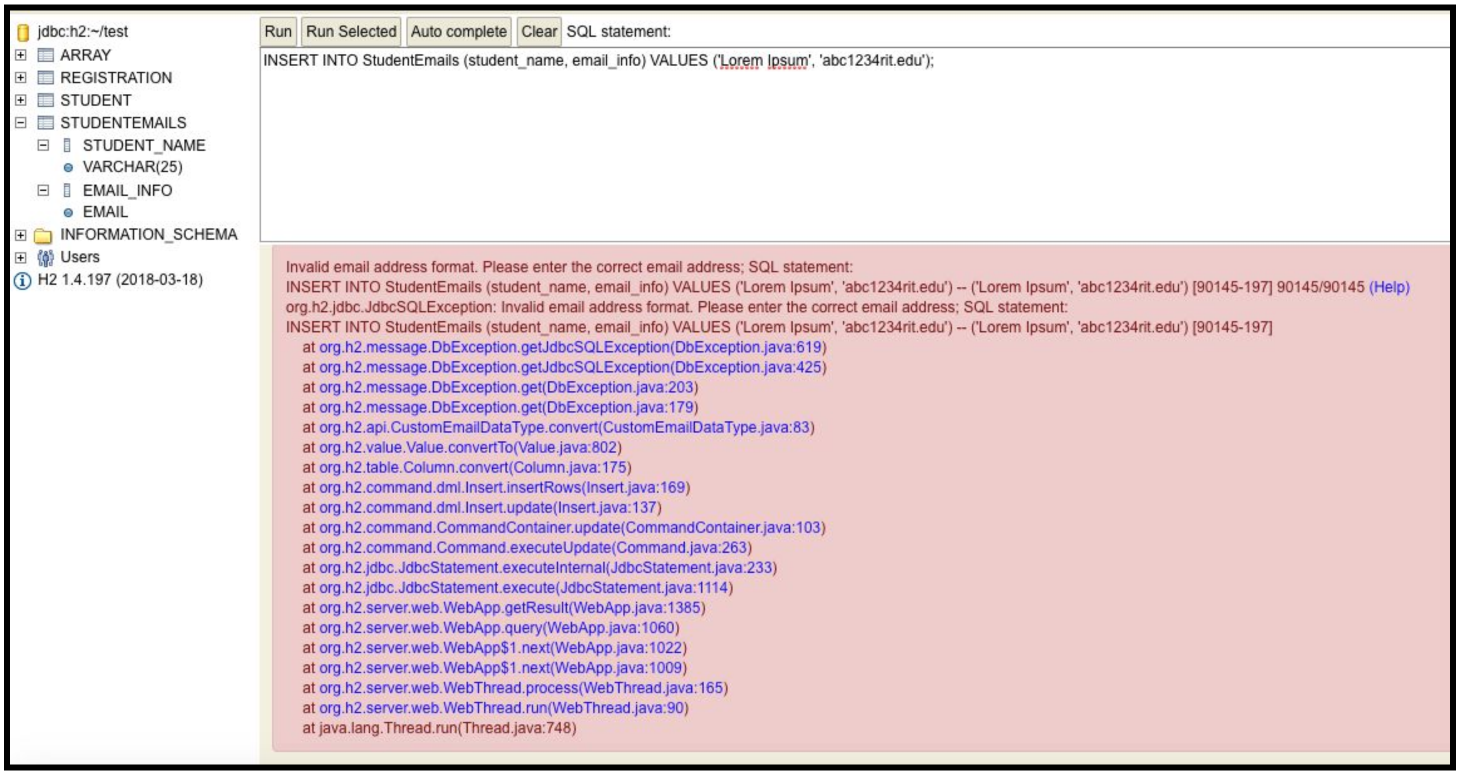
```
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Kushal Gevaria', 'kgg5247@rit.edu');

INSERT INTO StudentEmails (student_name, email_info) VALUES ('Akshay Pudage', 'ap7558@rit.edu');
```

```
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Kushal Gevaria', 'kgg5247@rit.edu');
Update count: 1
(5 ms)


INSERT INTO StudentEmails (student_name, email_info) VALUES ('Akshay Pudage', 'ap7558@rit.edu');
Update count: 1
(3 ms)
```

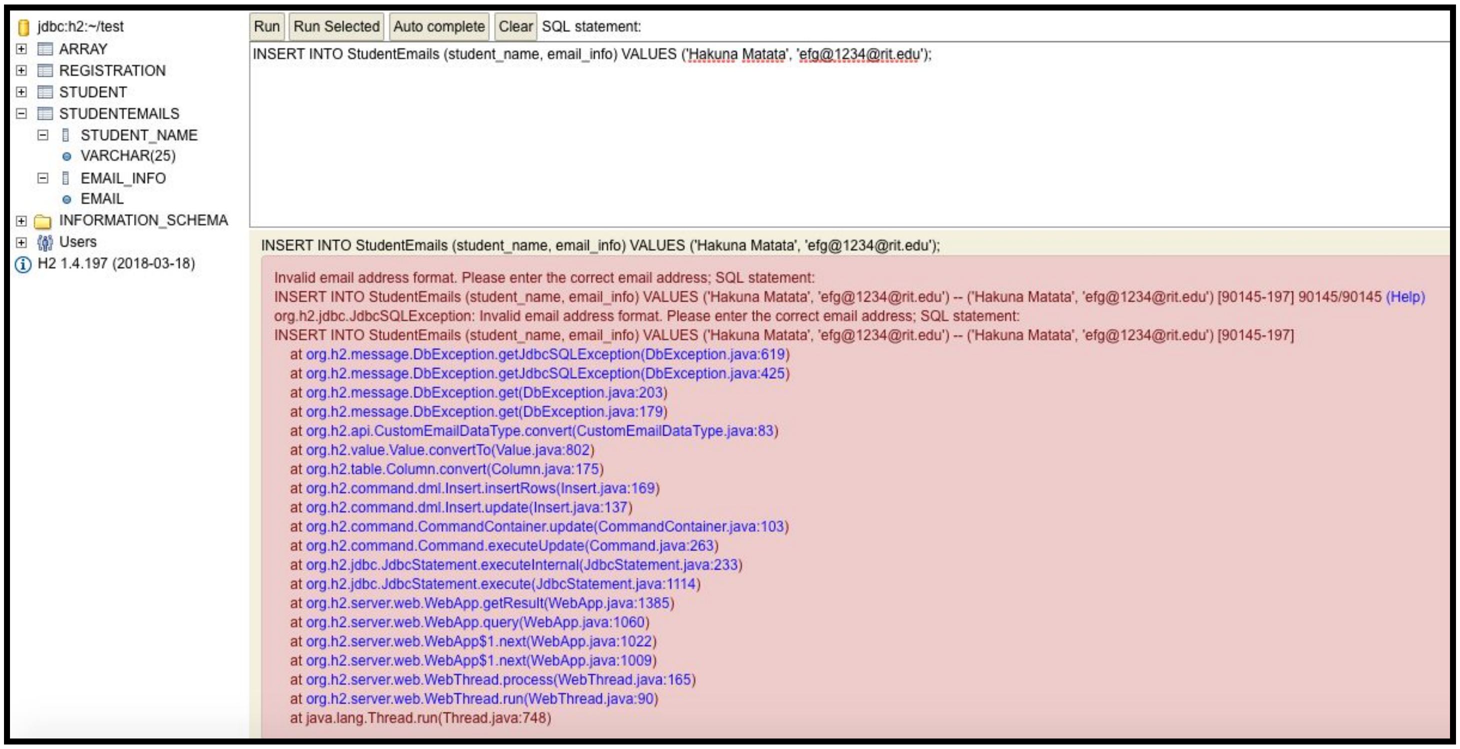
Inserting invalid values for the Email field



```
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Lorem Ipsum', 'abc1234rit.edu');
```

```
Invalid email address format. Please enter the correct email address; SQL statement:
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Lorem Ipsum', 'abc1234rit.edu') -- ('Lorem Ipsum', 'abc1234rit.edu') [90145-197] 90145/90145 (Help)
org.h2.jdbc.JdbcSQLException: Invalid email address format. Please enter the correct email address; SQL statement:
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Lorem Ipsum', 'abc1234rit.edu') -- ('Lorem Ipsum', 'abc1234rit.edu') [90145-197]
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:619)
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:425)
    at org.h2.message.DbException.get(DbException.java:203)
    at org.h2.message.DbException.get(DbException.java:179)
    at org.h2.api.CustomEmailDataType.convert(CustomEmailDataType.java:83)
    at org.h2.value.Value.convertTo(Value.java:802)
    at org.h2.table.Column.convert(Column.java:175)
    at org.h2.command.dml.Insert.insertRows(Insert.java:169)
    at org.h2.command.dml.Insert.update(Insert.java:137)
    at org.h2.command.CommandContainer.update(CommandContainer.java:103)
    at org.h2.command.Command.executeUpdate(Command.java:263)
    at org.h2.jdbc.JdbcStatement.executeInternal(JdbcStatement.java:233)
    at org.h2.jdbc.JdbcStatement.execute(JdbcStatement.java:1114)
    at org.h2.server.web.WebApp.getResult(WebApp.java:1385)
    at org.h2.server.web.WebApp.query(WebApp.java:1060)
    at org.h2.server.web.WebApp$1.next(WebApp.java:1022)
    at org.h2.server.web.WebApp$1.next(WebApp.java:1009)
    at org.h2.server.web.WebThread.process(WebThread.java:165)
    at org.h2.server.web.WebThread.run(WebThread.java:90)
    at java.lang.Thread.run(Thread.java:748)
```

Kushal Gevaria (kgg5247)
Akshay Pudage (ap7558)

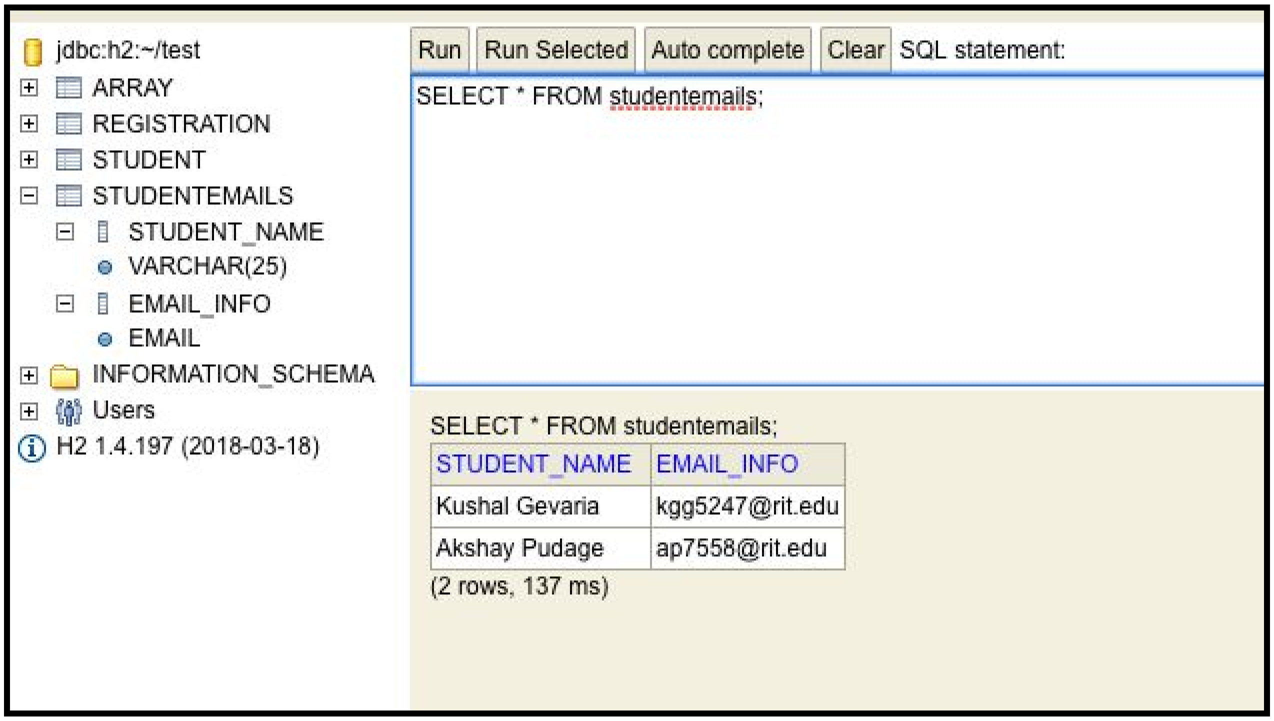


```
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Hakuna Matata', 'efg@1234@rit.edu');
```

```
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Hakuna Matata', 'efg@1234@rit.edu');

Invalid email address format. Please enter the correct email address; SQL statement:
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Hakuna Matata', 'efg@1234@rit.edu') -- ('Hakuna Matata', 'efg@1234@rit.edu') [90145-197] 90145/90145 (Help)
org.h2.jdbc.JdbcSQLException: Invalid email address format. Please enter the correct email address; SQL statement:
INSERT INTO StudentEmails (student_name, email_info) VALUES ('Hakuna Matata', 'efg@1234@rit.edu') -- ('Hakuna Matata', 'efg@1234@rit.edu') [90145-197]
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:619)
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:425)
    at org.h2.message.DbException.get(DbException.java:203)
    at org.h2.message.DbException.get(DbException.java:179)
    at org.h2.api.CustomEmailDataType.convert(CustomEmailDataType.java:83)
    at org.h2.value.Value.convertTo(Value.java:802)
    at org.h2.table.Column.convert(Column.java:175)
    at org.h2.command.dml.Insert.insertRows(Insert.java:169)
    at org.h2.command.dml.Insert.update(Insert.java:137)
    at org.h2.command.CommandContainer.update(CommandContainer.java:103)
    at org.h2.command.Command.executeUpdate(Command.java:263)
    at org.h2.jdbc.JdbcStatement.executeInternal(JdbcStatement.java:233)
    at org.h2.jdbc.JdbcStatement.execute(JdbcStatement.java:1114)
    at org.h2.server.web.WebApp.getResult(WebApp.java:1385)
    at org.h2.server.web.WebApp.query(WebApp.java:1060)
    at org.h2.server.web.WebApp$1.next(WebApp.java:1022)
    at org.h2.server.web.WebApp$1.next(WebApp.java:1009)
    at org.h2.server.web.WebThread.process(WebThread.java:165)
    at org.h2.server.web.WebThread.run(WebThread.java:90)
    at java.lang.Thread.run(Thread.java:748)
```

Viewing all values inside the relation



```
SELECT * FROM studentemails;
```

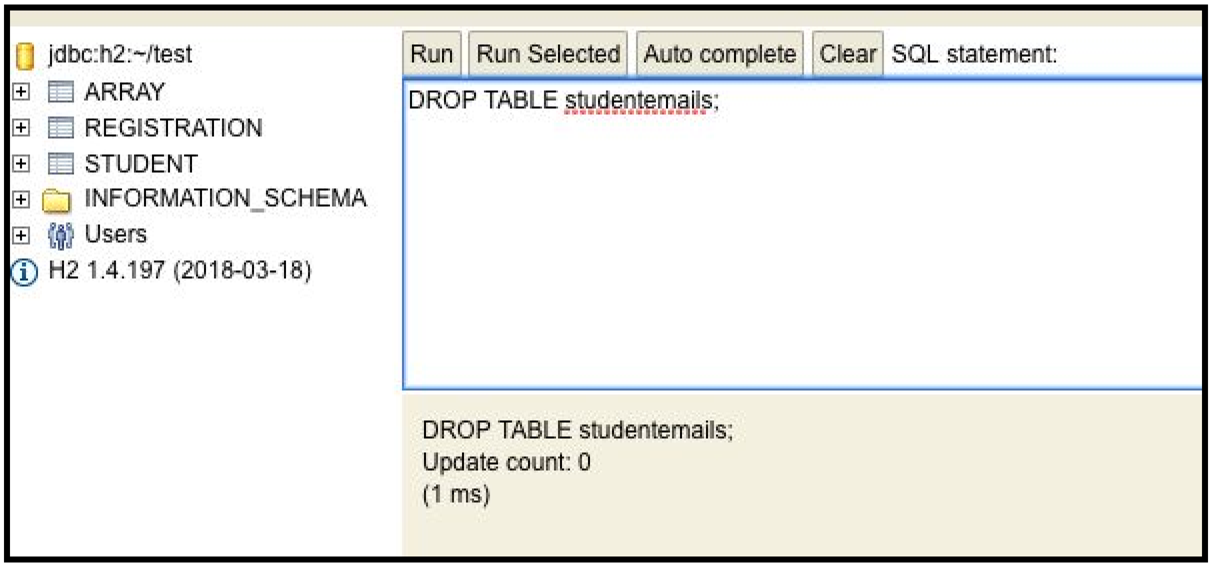
SELECT * FROM studentemails;

STUDENT_NAME	EMAIL_INFO
Kushal Gevaria	kgg5247@rit.edu
Akshay Pudage	ap7558@rit.edu

(2 rows, 137 ms)

Kushal Gevaria (kgg5247)
Akshay Pudage (ap7558)

Dropping relation



```
DROP TABLE studentemails;
```

```
DROP TABLE studentemails;
Update count: 0
(1 ms)
```