12-19-2018

Professor John Connor

Team Members: Samuel Youssef,

Connie Wu,

Bashir Ahamed,

Shofiqur Rahman.

Team Name: QuizBase

***Project 2 Report***

Tables in 3NF:

The following relations were implemented in MySQL and are in 3NF:

School(schoolId, name, location)

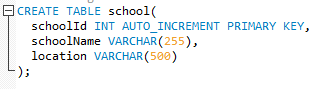


Figure 1: School schema.

Topic(topicId, topicName)

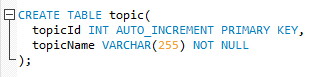


Figure 2: Topics schema.

User(userId, username, password)

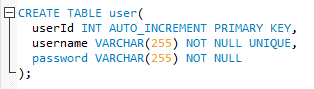


Figure 3: User schema.

Deck(deckId, name, classId, userId, topicId, creationDate, schoolId)

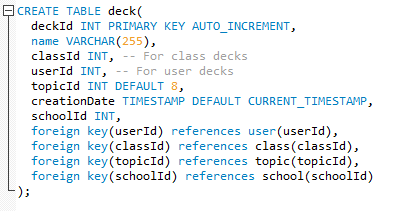


Figure 4: Decks schema.

Cards(cardId, deckId, cardName, description)

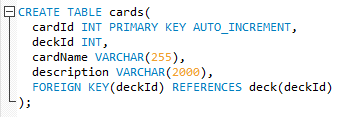


Figure 5: Cards schema.

Profile(userId, deckId)

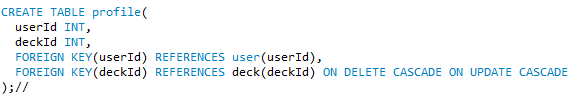


Figure 6: Profile schema.

Class(classId, ownerId <references userId>, name, topicId, description)

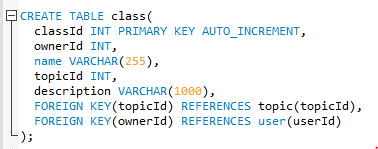


Figure 7: Class schema.

Members(userId, classId)

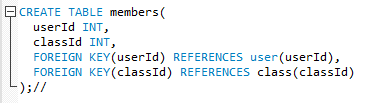


Figure 8: Members schema.

Request(userId, classId)

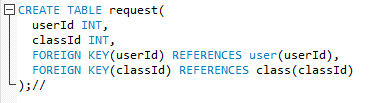


Figure 9: Request schema.

***The primary keys for the above relations are all underlined. Foreign key references are obvious.***

Additional functionalities added to the platform:

* The user can create online classes where the user can accept/deny requests from other user to join the newly created class. The following illustrations clarifies the process of creating classes and accepting/denying requests:

User ‘Connie’ logs into her account. The homepage is shown in figure 10:

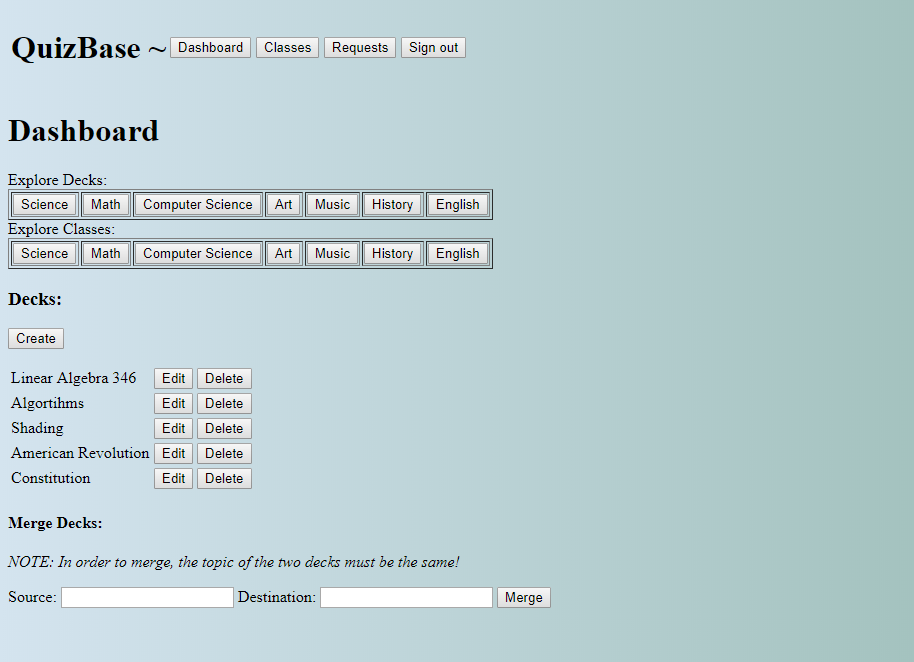


Figure 10: Homepage of user ‘Connie’.

Upon clicking on ‘classes’ button highlighted in figure 10, the user is going to be redirected to a page, shown in figure 11, where the user can:

1. Create a class.
2. Leave a class the user already joined in the past.
3. Go to a specific class where the user is able to create decks for classes the user owns or for classes the user is a member of.



Figure 11: Classes page of user ‘Connie’.

* The decks, which are created by users who joined a class, do not have a userId. Instead, those decks have a classId, and the information inside those decks remain available for other member of the class even if the user who created them decided to leave the class.
* Once the class is created, it becomes available to all other users. Users have to click on the specific topic they want to explore classes for. Once they click on a specific class, users are given the option to send a request of joining the class to the owner of the class. The following illustration clarifies the process:

User ‘Connie’ creates a new class called ‘csc 336 databases’ as shown in figure 12:

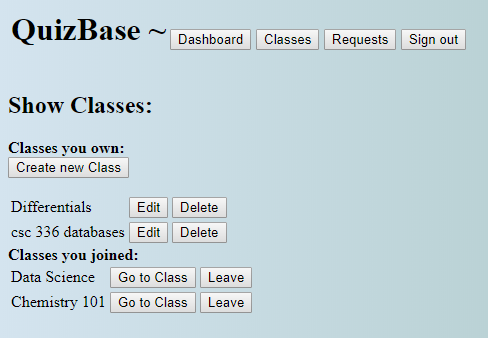


Figure 12: User ‘Connie’ created a new class ‘csc 336 databases’.

User ‘friend’ signs into her account, explores available classes, and decides to send a request to join the ‘csc 336 databases’ class created by user ‘Connie’ as shown in figure 12.

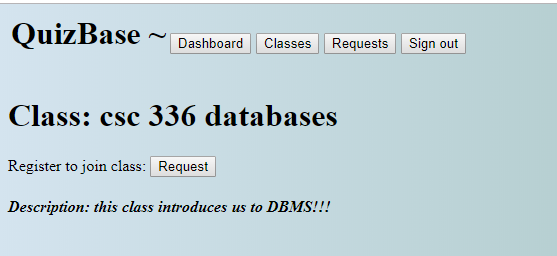


Figure 12: User ‘Friend’ sends a request to user ‘Connie’ to join ‘csc 336 databases’ class.

User ‘friend’ can also cancel her request of joining the class by clicking on ‘Remove Request’ button shown in figure 13:

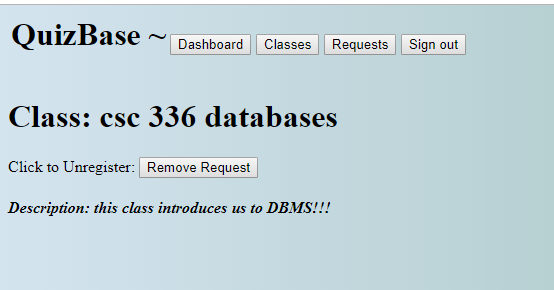


Figure 13: Option to roll back the request that has been sent to user ‘Connie’ is also available.

* The owner of the class—once signed into his/her account—can see the requests to join his/her class by clicking on the <requests> button at the top of the homepage. The owner of the class is given the option to accept/deny requests made by other users. The following illustration clarifies this process:

User ‘Connie’ has the option of accepting/denying user ‘friend’s’ request of joining ‘csc 336 databases’ class as shown in figure 14 below:

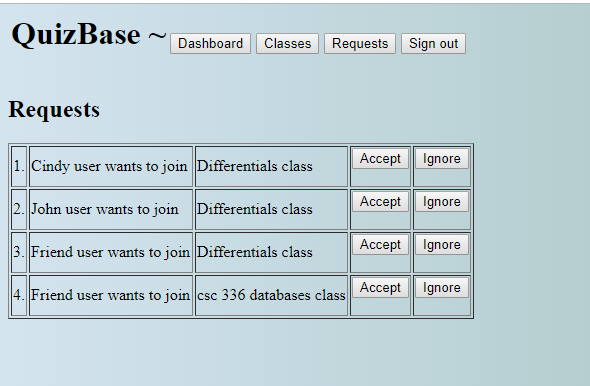


Figure 14: User ‘Connie’ has the option to accept/deny user ‘friend’s’ request of joining ‘csc 336 databases’ class.

* Once the request is accepted/denied, the record of the request gets deleted from the “Request” schema. If the request has been accepted, the information of the user who has been accepted to join a specific class is transferred over to the “Members” schema where the userId is associated with the classId which the user has joined.

The initial data population of users’ schema is shown below in figure 15:



Figure 15: Initial data which populated the attributes of users’ schema

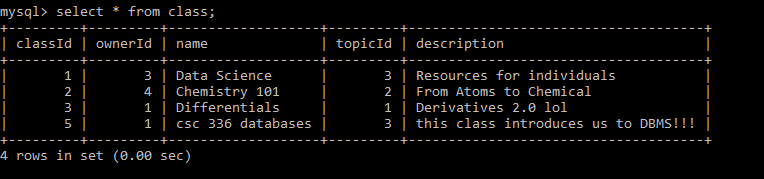


Figure 16: Class relation records after adding ‘csc 336 databases’ class by user ‘Connie’.

From figure 15 and 16, we can see that userId of user ‘Friend’ is 2, and the classId of ‘csc 336 databases’ class is 5.

Once user ‘Connie’ accepts friend’s request, the request get deleted from members schema, and user ‘friend’ is added to list of members where the classId is the same as ‘csc 336 databases’ classId as can be observed in figure 17, 18 below of memberslog and requestlog tables:

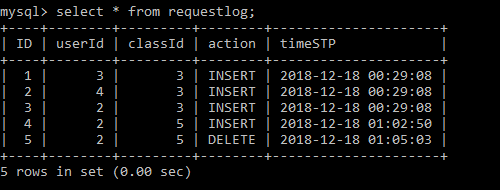


Figure 17: requestlog table where the request of user ‘friend’ has been deleted after user ‘Connie’ accepted the request.

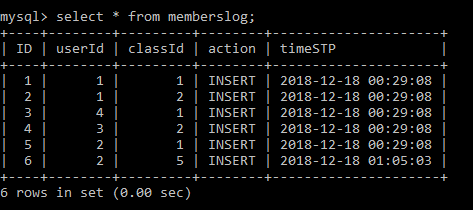


Figure 18: User ‘friend’ has been successfully added to members schema with a classId of 5 which is the classId of ‘csc 336 databases’.

Note: requestlog and memberlog tables are two tables which are used to log the actions done on the original tables of request and members respectively.

* Upon exploring decks on various topics by clicking on the topic name, the user can see decks that other users have created. The user cannot edit or delete decks of other users. However, the user who is currently signed in can click on the username of the user who have created the decks being explored. Once the username is clicked, the account owner who is signed in is able to see the classes created by the user who created the decks that are being explored by the currently signed-in user. The following illustration clarifies this process:

In the homepage, user ‘Connie’ clicks on ‘science’ button under ‘explore decks’. She then sees the following page in figure 19:

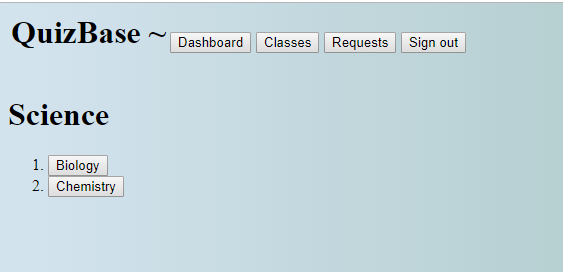


Figure 19: ‘Science’ button under ‘explore decks’ is clicked.

Upon clicking ‘Chemistry’ shown in figure 19 above, user ‘Connie’ is redirected to a page, shown in figure 20 below, showing decks related to ‘Chemistry’ topic that users across the platform have created over time.

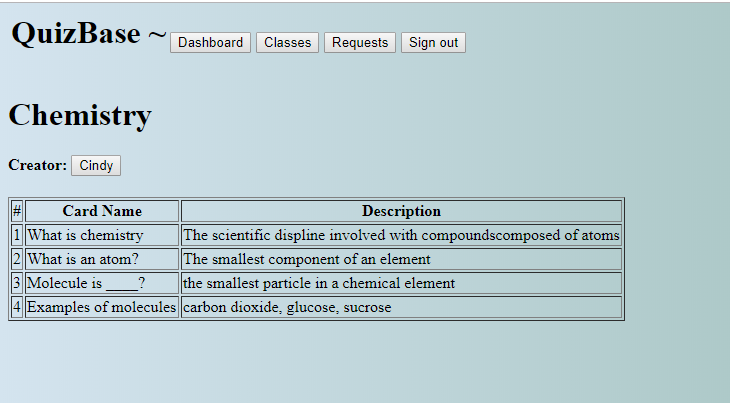


Figure 20: Page showing all decks (and cards) that users have created for ‘Chemistry’ topic.

Upon clicking on user ‘Cindy’ that is highlighted in figure 20, user ‘Connie’ <currently signed-in user> can view all the classes that whether ‘Cindy’ created or is a member of. This can be observed in Figure 21.

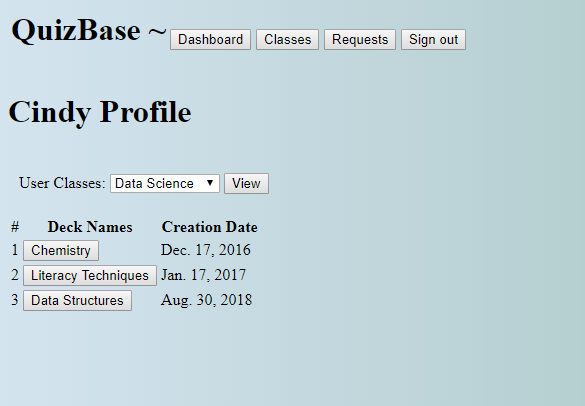


Figure 21: All classes that whether ‘Cindy’ created or is a member of.

* Triggers have been put in place to log the actions of users on the following schemas: User, Deck, Cards, Class, Members, Request. The triggers—upon ‘insert’, ‘update’, or ‘delete’ actions of users on those schemas—call specific function to log the aforementioned actions in the following tables accordingly: userlog, decklog, cardslog, classlog, memberslog, requestlog. The type of each logged action—<‘insert’, ‘update’, ‘delete’>— and the time stamp when the action took place are maintained under ‘action’ and ‘timeSTP’ attributes of each log table respectively. The following Figures illustrate the SQL code written to create decklog:

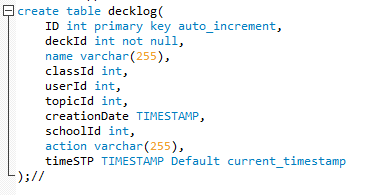


Figure 22: SQL code to create ‘decklog’ table.

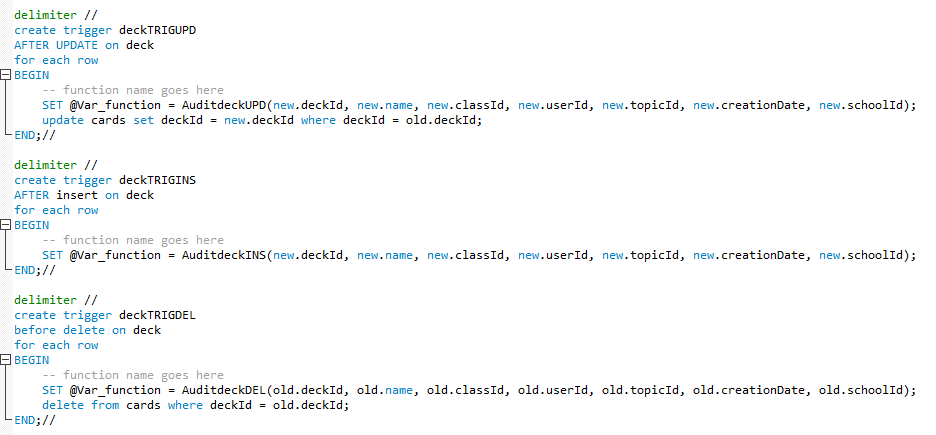


Figure 23: Three triggers used to log insert, delete, and update actions on deck schema.

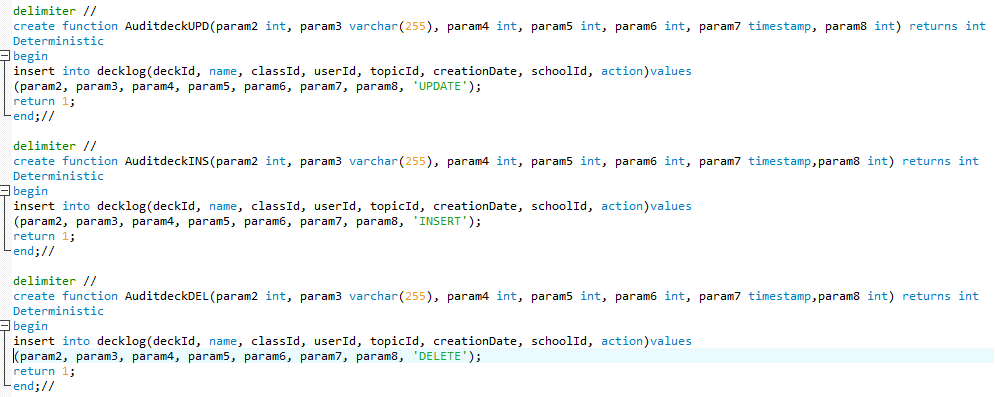


Figure 23: Functions called from three triggers shown in Figure 22 to insert into ‘decklog’ records of ‘insert’, ‘update’, or ‘delete’ actions taken by users on ‘deck’ schema.

Note: SQL code for other tables < userlog, cardslog, classlog, memberslog, requestlog> (including trigger code and functions called from various triggers) can be found in database.sql file inside the project folder submission package.

* Two procedures have been created to give the user the option to merge together two decks of the same topic that the user has created. The following illustration clarifies the SQL code for the procedure as well as the actual procedure in action:

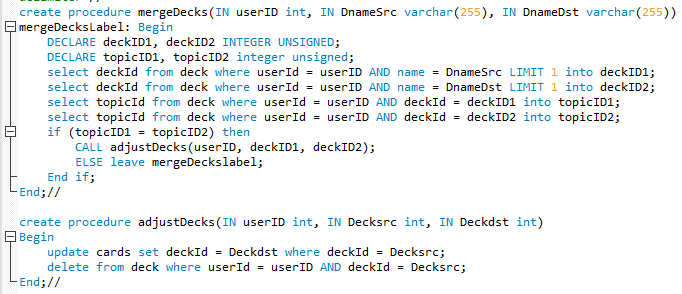


Figure 24: SQL code for two procedure that have been created to help the user merge two decks of the same topic.

User ‘Connie’ has the following decks:

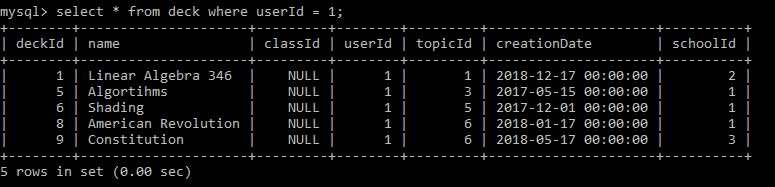


Figure 25: Decks created by user ‘Connie’.

From Figure 25, we can observe that:

1. ‘Linear Algebra 346’ and ‘Algorithms’ decks are of different topicId.
2. ‘American Revolution’ and ‘Constitution’ decks are of the same topicId.

Upon trying to merge the decks <‘Linear Algebra 346’, ‘Algorithms’>, nothing changes because the two decks are of different topics as can be observed from figure 26 below:

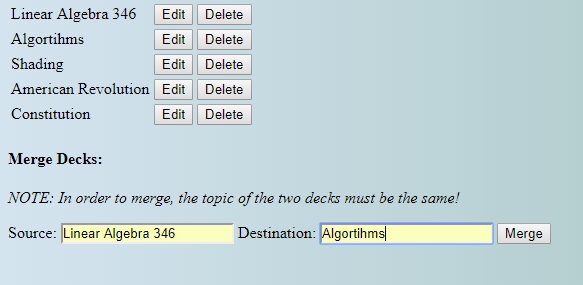


Figure 26: upon trying to merge <‘Linear Algebra 346’, ‘Algorithms’>, nothing get merged because the topics of the two decks are different.

upon trying to merge <‘American Revolution’, ‘Constitution’>, we notice that the source deck disappears (gets deleted), and the cards inside the source deck are transferred to the destination deck <Constitution>. This can be observed from Figure 27 below:

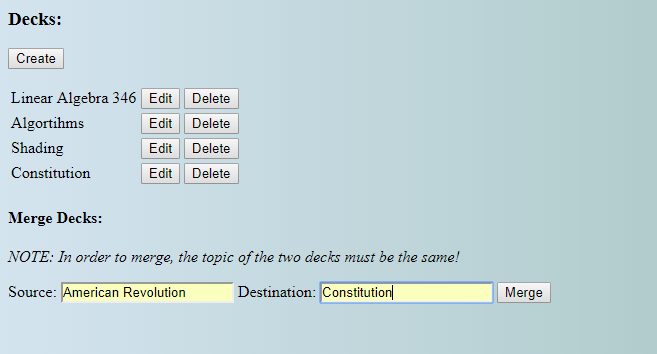


Figure 27: Upon successful merge, deck ‘American Revolution’ disappears, and deck ‘Constitution’ takes in the cards of deck ‘American Revolution’.

The following two Figures of ‘decklog’ and ‘cardslog’ tables show the successful transfer of cards from one deck to the other as well as the successful deletion of the source deck, in this case the ‘American Revolution’ deck.

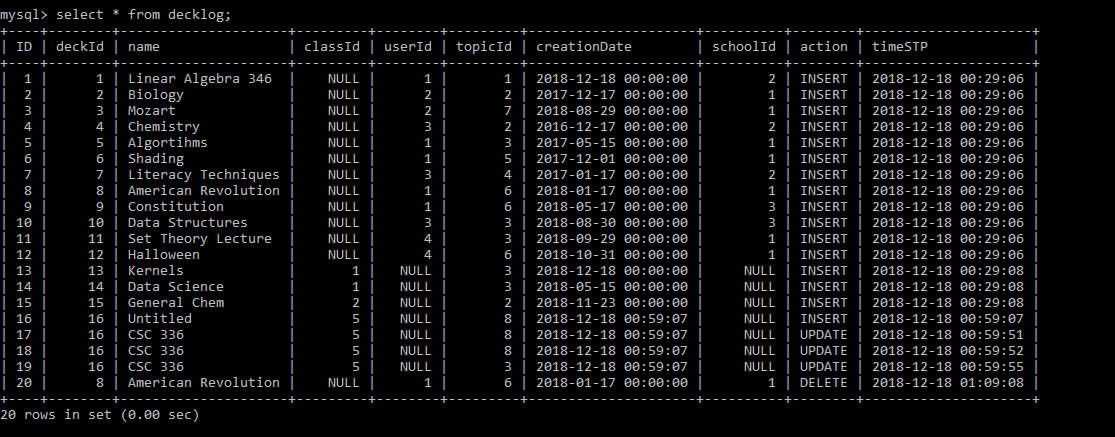


Figure 28: The source deck ‘American revolution’ is logged as ‘deleted’ from table deck.

Card ‘George Washington’ that once was part of deck ‘American Revolution’ <with deckId 8> is updated as highlighted at the end of the table shown in figure 29 with a new deckId of 9 which is the deckId of the destination deck ‘Constitution’.

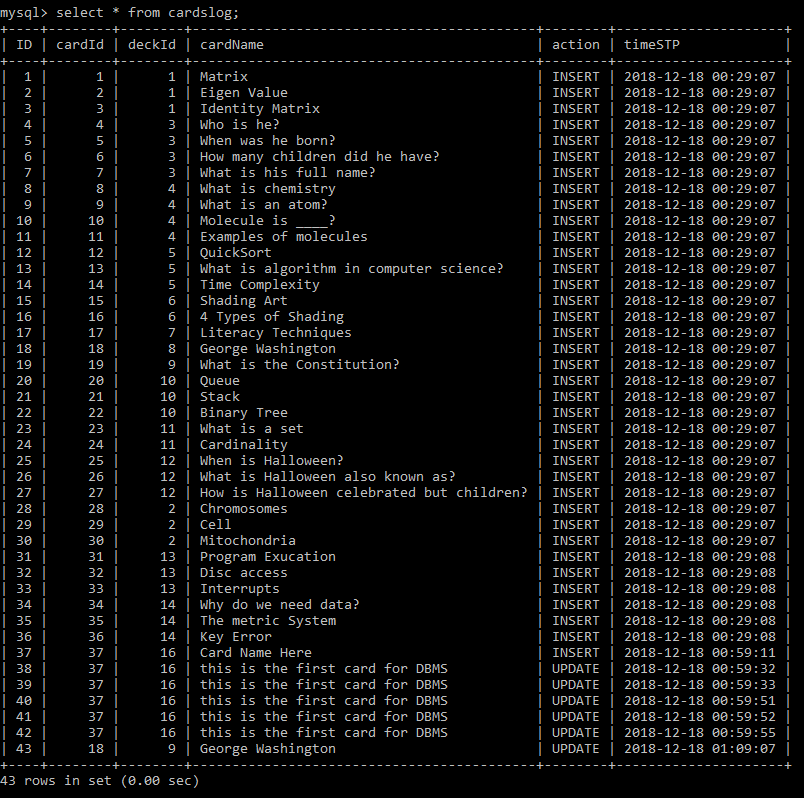


Figure 29: Successful transfer of cards from deck ‘American Revolution’ to deck ‘Constitution’ to complete the merging of the two decks into one deck, in this case ‘Constitution’ deck.

Note: For instructions on how to run the app successfully, please refer to the first part of project 1 report.