

1. Introduction: Overview

With only minutes before a test and weeks worth of information to cram, one does not have the time to page through an enormous textbook and pick out the important facts. It would be great if one could simply be handed the need-to-know knowledge and commit it immediately to short-term memory. Enter CardEd (Card Editor), the online flashcard repository. With CardEd, any procrastinator can quickly and easily locate the perfect flashcard deck and ensure that his or her GPA never dips below scholarship requirements.

CardED is designed to allow students from any college to quickly and easily locate a tested-and-approved flashcard deck for each class. The student will be offered several criteria to refine the search, such as college name, course name, course number, course type, and deck author. Once the student receives the search results, he or she will be able to determine which decks are most effective based upon the user ratings. User ratings for the decks will be derived from the ratings of each card in the deck. If a deck has a large number of cards with low user ratings, a registered user will have the option to edit the deck by adding or removing cards. If a deck receives a particularly low rating, an administrator will have the option to remove it from the server.

Because the cards are no longer physical sheets of paper, it is possible for the same card to appear in multiple decks. Furthermore, multiple courses from multiple schools will be able to share and exchange decks of cards. This will give new colleges more incentive to join the CardED service. For instance, imagine what would happen if a student learned that his or her college joined CardED only to discover that there were absolutely no cards available yet? With card and deck sharing, the student can know that even though his or her welcome page is initially devoid of decks, he or she can still receive immediate results from other college pages. Furthermore, once this student links to these decks, the decks will be available to everyone in his or her college.

The first version of the web interface will run on a Linux machine and will be built using a combination of PHP and MySQL. PHP was chosen for its ease of use and its ability to produce results quickly. It will be essential to use a language that can easily adapt while the database backend is being tested and extended. Because of the slow response time of php, the application will later be ported either to Javascript or to a stand-alone Java application. Ideally, this will allow students with PDA's to access the flashcards in real time rather than needing to wait for page updates every time they flip over a card. If the stand-alone application is created, it will also offer a method to import deck descriptions from the database in order to produce stand-alone decks. In this way, students will be able to use the flashcards whether or not they are connected to the Internet.

In short, CardED will not only eliminate the time required to create a brand new deck of flashcards, but it will also offer students decks that have been tested by several students. There is little doubt that CardED has the potential to revolutionize the art of studying.

2. Application Requirements Specifications

The diagram is a rough template for the overall pages of the site depicting the basic functionalities needed in each page. It is subject to change as the project evolves and we decide extra capabilities are needed or should be prioritized more, but currently this serves as a basis for any future designs.

Home

The Home page will mainly be a place for the site administrators to post news and a center for users to access the register, login/logout, and search pages. It will need to determine if the client is currently logged in or not.

User

The User page displays all of a logged in client's personal information as well as the cards and decks he has saved or created. This information will naturally be queried from the database. From this page, the user has the options to logout, edit his profile, create a new card, or compile a new deck which will all be implemented such that only logged in users can perform these tasks. Additionally the client may access the search page here as well, although it should be noted that non-logged in clients have access to this functionality as well.

Register/Edit

The register/edit profile page allows the user to create or modify the fields corresponding to his personal data which will in turn correspond to minor data manipulations in the database.

Query

The query page is the database intensive section of the site allowing the user to perform powerful queries aided by the assistance of a graphical web form. The client will be able to combine search terms using logical equivalents of AND, OR, and NOT. When returning a query, this page will output a table of results. Each card and deck returned from the search will in turn be anchored to the card page or the deck handler. Additionally logged in users will be able to select any number of returned results via checkboxes and either save them for further reference or add them to a list of proposed cards that a user wishes to compile into a deck.

Create Card

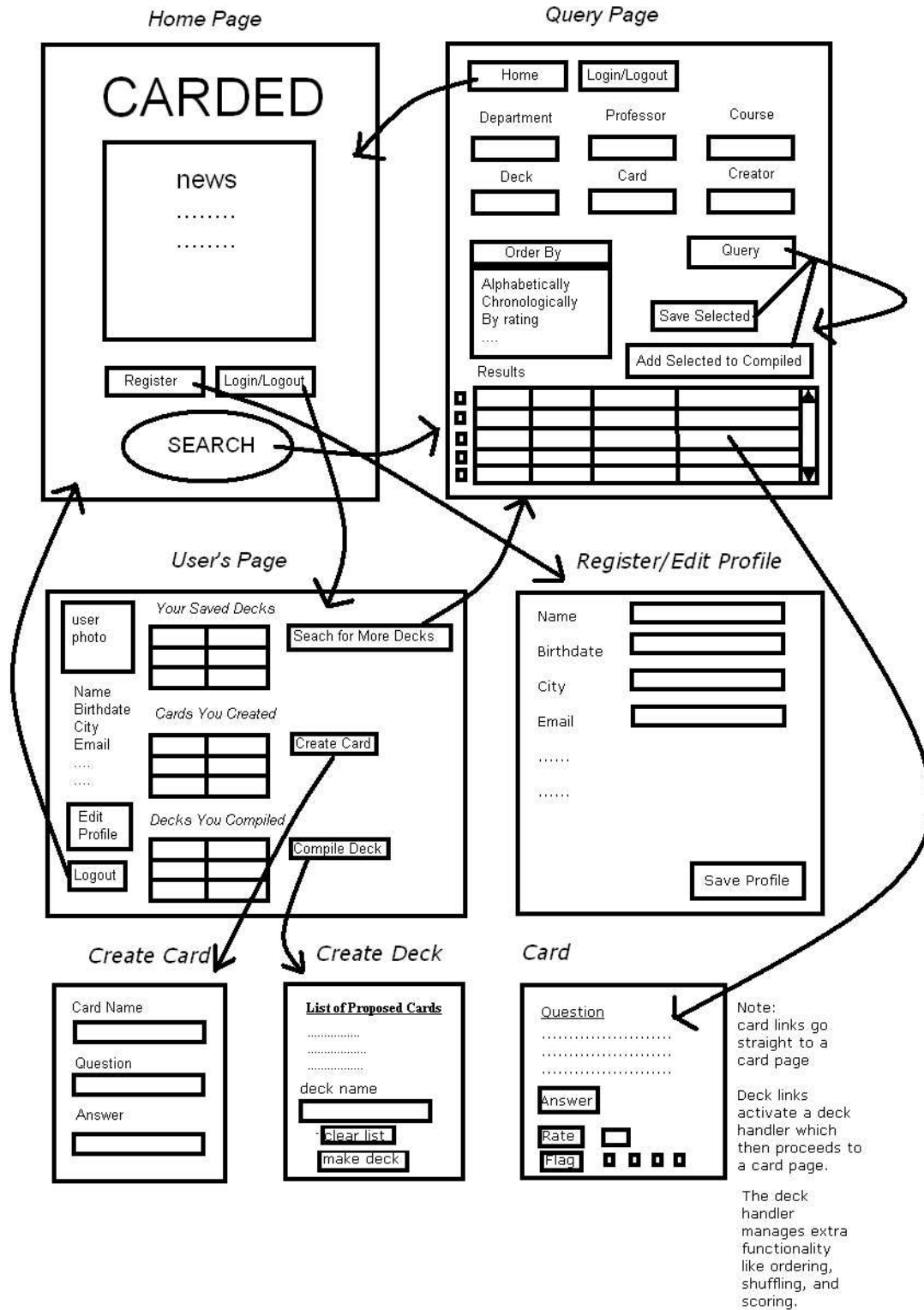
The create card page has fields for the name, question, answer and other applicable attributes of a card and is only accessible by a logged in user.

Create Deck

The create deck page outputs the user's current list of proposed cards. If he so decides, he may remove cards from this list, clear it altogether, add cards to the list via the search page, or finalize his proposal into a deck through the "make deck" button. Like the create card page, this page will grant functionality only to logged in users.

Card

Whenever a user clicks on a card link, the database will be queried for the specific card in question and return to the client a page with the question and answer (though hidden until the user requests to see it). Logged in users will also have the option to rate the card and set certain flag behaviors. When a deck link is clicked, the deck handler is activated which keeps track of the order of cards and the user's progress and score. Thus when the deck handler is linked, extra options appear on the card page such as the ability to move to the next card.



3. Database Requirements Specifications

The CardED database is driven by five primary entities: the School, the Course, the User profile, the flashcard Deck, and the individual Card. Furthermore, there are two fundamental relationships: every Deck consists of one or more Cards, and every School offers one or more Courses.

To the user accessing the CardED site, each Deck will appear to be its own unique entity, much like a traditional paper deck. Each Deck will be required to have a Name to help distinguish it from other Decks (although the Name does not need to be unique), and optionally it can offer a Description of its contents. Each Deck will appear to contain a set number of Cards geared directly towards its individual purpose. Perhaps most comforting of all, Users will not need to worry about other people modifying their favorite deck. Much like regular flashcards, one cannot simply change the contents of an individual card.

At this point, however, the similarities with traditional decks come to an end. Under the hood, these digital flashcard Decks are simply a list of desired Cards packaged with a Name, a Description, and a small amount of Versioning information. Digital flashcard Decks have the ability to share useful Cards and inherit Cards from previously defined Decks. For example, consider a Deck designed to be a general introduction into computer science. Perhaps a student or Professor would like to modify this deck so that is more focused towards the topic of interprocess communication. He or she would simply select the general computer science deck as a base and derive the new deck by including new Cards or excluding old Cards. Still, another student may want to derive another deck from the generic computer science deck. By storing the Versioning information in a table relating Decks to Decks, it becomes possible to create such multiply-derived Decks.

To protect the information in Decks from becoming corrupted, the CardED database will allow individual Cards to be created but not modified. For this reason, the Names and Descriptions of the Cards do not need to be unique (it would be senseless to require a User to modify the name of a Card just for fixing a typo). Users will have permission to derive new Decks and “remove” (omit) unwanted Cards, but only administrators will have permission to delete a Card from the database. If an administrator deletes a Card, then it will be removed from every Deck that references it using foreign keys.

At a higher level, each School will contain a list of Courses that it is currently offering, and these Courses will Register zero or more Decks of cards that relate to the Course's topic. This presents the User with a convenient way to locate Decks for a course that he or she is enrolled in. For instance, the student could search all of the Decks associated with the Name of a School, or perhaps the student would like to narrow the search and find Decks that have been approved for Courses in a certain Department (department being an attribute of a Course, not an individual entity). Further still, the User will have the option to Register with a School and select Courses that he or she is attending. This will allow CardED to display all Decks relevant to a User's Course load upon login.

Database Queries:

Validate User:

Logging in will require a simple query returning user information where the username and password hash match those provided by the user.

Displaying User Decks:

This query will be a union of deck information for a user's saved decks, course decks, and created decks. Each returned row will include deck name, average card rating, number of cards contained, and, for course decks, related course.

Search Query:

This will return a list of all decks and/or cards fitting a set of criteria. Criteria will include: deck name, card name, course name, teacher name, card rating, average deck rating, school, deck owner.

Class Decks:

This query will return, given a set of course keys, all of the decks associated with those courses. Included in the returned rows will be course and deck information.

School Search:

This is a simple query returning a list of schools matching one or more of: name, city, or state. School names will be returned.

Deck Cards Display:

This will return a list of all cards in a certain deck, along with any flags associated with those cards. Card name, description, and average rating will be returned along with boolean values indicating the presence or absence of certain card flags.

Card Display:

Given a certain card key or keys, this will return all of the information about a card, including card name, description, average rating, owner, question, answer, and booleans for certain card flags.

User Information:

Given a single user key, this will return information on a user including real name, username, email, average created card rating, number of cards created, and number of owned decks.

Transactions:

Create/Update User Information:

Given a single user key and valid user information, this will create or update a user's record. Valid user information includes: email, password hash, school, and, for creation, real name and username.

Card Creation:

Given information on a card, this will create a new card and its associated Created relationship. Card information includes: card name, description, question, answer, and creating user.

Deck Creation:

Given information on a deck, this will create a new deck and its associated Owner and Parent relationships. Deck information includes: parent deck key, deck name, deck description, and owner.

Course Creation:

This will create a new course and its associated Offers relationship from course department, teacher, number, name, and school.

School Creation:

This will create a new school from school name, city, and state.

Add Course Deck:

This will create a Uses relationship given course and deck keys.

Add Deck Card:

This will create a Contains relationship given deck and card keys.

Rate Card:

This will increment a card's Times Rated and add a new rating to its Total Rating.

Save Deck:

This will add a new Saved Deck relationship given user, deck, and, optionally, course keys.

Add User Course:

This will add a new Takes relationship given user and course keys.

Flag Card:

This will add a new Flagged Card relationship given a user, card, and a flag type.

4. ER Data Model Design

Primary Key
Candidate Key
(Full, Partial)

Entities

User

Username – String

Real Name – Composite

 First – String

 Last – String

Email – String

Password Hash – String

School

Name - String

City – String

State – String[2]

Course (Weak)

Department – String

Number – Integer (>0)

Teacher – String

Name – String

Deck

Name – String

Description – String

Card

Name – String

Description – String

Question – String

Answer – String

Rating – Composite

 Times Rated – Integer

 Total Rating – Integer

Relationships

Offers (School, Course) 1 □ N (Defines Course)

Uses (Course, Deck) M □ N

Contains (Deck, Card) M □ N

Goes To (User, School) 1 □ N

Takes (User, Course) M □ N

Saved Deck (User, Deck, Course)

Owns Deck (User, Deck) 1 □ N

Created Card (User, Card) 1 □ N

Parent (Deck, Deck) 1 □ N

Flagged Card (User, Card) M □ N

Flag Type – Enumeration (Learned, Skipped, etc.)

5. Transforming the ER Model to the Relational Model

```
CREATE DOMAIN dSTATE CHAR(2)
  CHECK (VALUE IN ('OH', 'IN', 'FL', ...));

CREATE DOMAIN dCARD_FLAG VARCHAR(8)
  CHECK (VALUE IN ('KNOWN',      -- User knows the answer to this card
                  'PROBLEM'     -- This is a problem card for user
                  -- We need to decide on our other flags
                  ));

*****
                          Entities
*****

CREATE TABLE User
(
  UserID      INTEGER AUTO_INCREMENT,
  Username    VARCHAR(16) NOT NULL,
  Password    CHAR(20) NOT NULL,
  Email       VARCHAR(32),
  FirstName   VARCHAR(16),
  LastName    VARCHAR(16),
  SchoolID   INTEGER,
  PRIMARY KEY(UserID),
  UNIQUE(Username)
);

CREATE TABLE Card
(
  CardID      INTEGER AUTO_INCREMENT,
  Owner       INTEGER NOT NULL,
  Name        VARCHAR(16) NOT NULL,
  Desc        VARCHAR(64),
  Question    VARCHAR(255) NOT NULL,
  Answer      VARCHAR(255) NOT NULL,
  NumberStars INTEGER DEFAULT 0,
  NumberRatings INTEGER DEFAULT 0,
  PRIMARY KEY(id)
  FOREIGN KEY(Owner) REFERENCES User(UserID)
);

CREATE TABLE Deck
(
  DeckID      INTEGER AUTO_INCREMENT,
  Owner       INTEGER NOT NULL,
  Name        VARCHAR(32) NOT NULL,
  Desc        VARCHAR(255),
  PRIMARY KEY(id),
  FOREIGN KEY(Owner) REFERENCES User(UserID)
);

CREATE TABLE School
(
```

```

    SchoolID      INTEGER AUTO_INCREMENT,
    Name          VARCHAR(64) NOT NULL,
    City          VARCHAR(32),
    State         dSTATE,
    PRIMARY KEY(SchoolID)
);

```

```

CREATE TABLE Course
(
    CourseID      INTEGER AUTO_INCREMENT,
    SchoolID     INTEGER,
    Name          VARCHAR(32) NOT NULL,
    Teacher       VARCHAR(32),
    Dep           VARCHAR(5) NOT NULL,
    CourseNo     INTEGER NOT NULL,
    PRIMARY KEY(CourseID)
    FOREIGN KEY(SchoolID) REFERENCES School
);

```

```

*****
                                Relationships
*****

```

```

CREATE TABLE DeckContainsCard
(
    DeckID       INTEGER NOT NULL,
    CardID       INTEGER NOT NULL,
    PRIMARY KEY(DeckID, CardID),
    FOREIGN KEY(DeckID) REFERENCES Deck,
    FOREIGN KEY(CardID) REFERENCES Card,
);

```

-- Used to list Decks associated w/ Courses at the home screen

```

CREATE TABLE UserTakesCourse
(
    UserID       INTEGER NOT NULL,
    CourseID     INTEGER NOT NULL,
    PRIMARY KEY(Username),
    FOREIGN KEY(Username) REFERENCES User,
    FOREIGN KEY(CourseID) REFERENCES Course,
);

```

```

CREATE TABLE CourseUsesDeck
(
    CourseID     INTEGER NOT NULL,
    DeckID       INTEGER NOT NULL,
    PRIMARY KEY(CourseID, DeckID),
    FOREIGN KEY(CourseID) REFERENCES Course,
    FOREIGN KEY(DeckID) REFERENCES Deck,
);

```

-- Allows users to save their favorite decks

```
CREATE TABLE UserSavesDeck
(
  UserID      INTEGER NOT NULL,
  DeckID      INTEGER NOT NULL,
  PRIMARY KEY(UserID, DeckID),
  FOREIGN KEY(UserID) REFERENCES User,
  FOREIGN KEY(DeckID) REFERENCES Deck
);
```

-- Required to prevent users from voting for cards multiple times

```
CREATE TABLE UserRatedCard
(
  UserID      INTEGER NOT NULL,
  CardID      INTEGER NOT NULL,
  PRIMARY KEY(UserID, CardID),
  FOREIGN KEY(UserID) REFERENCES User,
  FOREIGN KEY(CardID) REFERENCES Card
);
```

-- Users can flag cards to change the behavior of the deck

```
CREATE TABLE UserFlagsCard
(
  UserID      INTEGER NOT NULL,
  CardID      INTEGER NOT NULL,
  Flag        dCARD_FLAG NOT NULL,
  PRIMARY KEY(UserID, CardID, Flag),
  FOREIGN KEY(UserID) REFERENCES User,
  FOREIGN KEY(CardID) REFERENCES Card,
);
```

8. SQL Queries

```
-----  
--                               Validating a User                               --  
-- (If a valid Username / Password pair is given, the username                ) --  
-- will be returned. Otherwise, the select will be empty                       ) --  
--                                                                              --  
-- $USERNAME: The username of the person logged into the site                 --  
-- $PASSWORD: The Unencrypted user password                                   --  
-----
```

```
SELECT U.UserID  
FROM   User U  
WHERE  U.Username = $USERNAME  
        AND U.Password = SHA1 ($PASSWORD)
```

```
username ( username=$username AND password=Password($password)(User))
```

```
(t | 3x (User(x)) X.Username=$username x.password=SHA1($password) t.1=x.username)
```

```
-----  
--                               Select Decks created by User                               --  
--                                                                              --  
-- $USERNAME: The username of the person logged into the site                 --  
-----
```

```
SELECT D.DeckID  
FROM   User U, Deck D  
WHERE  U.Username LIKE $USERNAME  
        AND U.UserID = D.Owner
```

```
DeckID ( username=$username (UserOwnsDeck))
```

```
(t | 3x (UserOwnsDeck(x)) X.Username=$username t.1=x.DeckID)
```

```
-----  
--                               Finding Decks Associated with Classes                               --  
--                                                                              --  
-- $Classes: A vector containing Course department and number pairs         --  
-----
```

```
SELECT D.DeckID, C.Dep, C.CourseNo  
FROM   Course C, CourseUsesDeck D  
WHERE  <C.Dep, C.CourseNo> IN $CLASSES  
        AND C.CourseID = D.CourseID
```

```
DeckID, CourseID (CourseUsesDeck) |X| Dep, CourseNo( <dep, CourseNo> in $class(Course))
```

```
(t | 3x 3y (Deck(x)) Course(x)) X.CourseID=y.CourseID $classes (y.Dep, Y.CourseNo)  
t.1=x.DeckID t.2=y.Dep t.3=y.CourseNo)
```

```
-----  
--                               Get General User Information                               --  
--                                                                              --  
-- $Username: The name of any user                                           --  
-----
```

```
SELECT U.FirstName, U.LastName, U.Username, U.Email
```

FROM User U
WHERE U.Username = \$Username

FirstName, LastName, Username, email (username=\$username (User))

(t | 3x (User(x)) X.Username=\$username t.1=x.FirstName t.2=x.LastName
t.3=x.Username t.4=x.email)

```
-----  
--                               Get General Card Information                               --  
-- (Note: Average card rating is not computed here because there ) --  
-- (      is a risk of a divide-by-zero error.  Let php handle it) --  
--                                                                                   --  
-- $CARDS: A vector containing CardID's                                             --  
-----
```

```
SELECT *  
FROM   Card C  
WHERE  C.CardID IN $CARDS
```

```
-----  
--                               Find all Cards belonging to a Deck                               --  
--                                                                                   --  
-- $Deck: a DeckID                                                                 --  
-----
```

```
SELECT C.CardID  
FROM   DeckContainsCard C  
WHERE  C.DeckID = $DECK
```

```
-----  
--                               Find the number of cards created by User                               --  
--                                                                                   --  
-- $Username: The name of any user                                             --  
-----
```

```
SELECT COUNT(*)  
FROM   User U, Card C  
WHERE  U.Username = $Username  
        AND  U.UserID = C.Owner
```

```
-----  
--                               Find all cards created by a User                               --  
--                                                                                   --  
-- $Username: The name of any user                                             --  
-----
```

```
SELECT C.CardID  
FROM   User U, Card C  
WHERE  U.Username = $Username  
        AND  U.UserID = C.Owner
```

```
-----  
--                               Find number of Decks owned (Created) by User                               --  
--                                                                                   --  
-- $Username: The name of any user                                             --  
-----
```

```
SELECT COUNT(*)  
FROM   User U, Deck D  
WHERE  U.Username = $Username  
        AND  U.UserID = D.owner
```

```
-----  
-- Find the number of Decks saved by User --  
-- --  
-- $Username: The name of any user --  
-----
```

```
SELECT COUNT(*)  
FROM UserSavesDeck D  
WHERE D.Username = $Username
```

```
DeckID ( username=$username (UserSavesDeck))
```

```
(t | 3x (UserSavesDeck(x)) X.Username=$username t.1=x.DeckID)
```

```
-----  
-- Find all decks saved by User --  
-- --  
-- $Username: The name of any user --  
-----
```

```
SELECT D.DeckID  
FROM UserSavesDeck D  
WHERE D.Username = $Username
```

```
-----  
-- Retrieve Information needed to rate Decks --  
-- (Note: Once again, php will be used to calculate the average) --  
-- --  
-- $DECKS: A vector of DeckID's --  
-----
```

```
SELECT D.DeckID, SUM(C.NumberStars), SUM(C.NumberRatings)  
FROM DeckContainsCard D, Card C  
WHERE D.DeckID IN $DECKS  
AND D.CardID = C.CardID  
GROUP BY D.DeckID
```

```
-----  
-- Find all Flags given to Card by User --  
-- --  
-- $Username: The name of any user --  
-- $Card: The CardID --  
-----
```

```
SELECT F.Flag  
FROM User U, UserFlagsCard F  
WHERE U.Username = $USERNAME  
AND U.UserID = F.UserID  
AND F.CardID = $CARD
```

9. Integrity Constraints

```
*****
  STORED PROCEDURES
*****

CREATE PROCEDURE RateCard (user_id INT,card_id INT,rating INT)
BEGIN
  DECLARE rated INT;

  IF rating >= 1 AND rating <= 5 THEN

    SELECT COUNT(*) INTO rated FROM UserFlagsCard
      WHERE UserID=user_id AND CardID=card_id AND Flag='RATED' LIMIT 1;

    IF rated = 0 THEN
      UPDATE Card
        SET NumberStars=NumberStars+rating ,
          NumberRatings=NumberRatings+1
        WHERE CardID=card_id LIMIT 1;

      INSERT INTO UserFlagsCard (UserID,CardID,Flag)
        VALUES (user_id,card_id,'RATED');
    END IF

  END IF

END

CREATE PROCEDURE SetPassword (user_id INT,plainpass CHAR(32))
BEGIN
  DECLARE user_name CHAR(16);

  SELECT Username INTO user_name FROM User
    WHERE UserID=user_id LIMIT 1;

  IF user_name!=plainpass AND LENGTH(plainpass)>=6 THEN
    UPDATE User SET Password=SHA1(plainpass)
      WHERE UserID=user_id LIMIT 1;
  END IF

END

*****
  FOREIGN KEY CONSTRAINTS
*****

CREATE TABLE User
  ...
  FOREIGN KEY(SchoolID) REFERENCES School ON DELETE SET NULL,
```

```
CREATE TABLE Card
...
FOREIGN KEY(Owner) REFERENCES User (UserID) ON DELETE SET NULL,
```

```
CREATE TABLE Course
...
FOREIGN KEY(SchoolID) REFERENCES School ON DELETE CASCADE,
```

```
CREATE TABLE DeckContainsCard
...
FOREIGN KEY(DeckID) REFERENCES Deck ON DELETE CASCADE,
FOREIGN KEY(CardID) REFERENCES Card ON DELETE CASCADE,
```

```
CREATE TABLE UserTakesCourse
...
FOREIGN KEY(Username) REFERENCES User ON DELETE CASCADE,
FOREIGN KEY(CourseID) REFERENCES Course ON DELETE CASCADE,
```

```
CREATE TABLE CourseUsesDeck
...
FOREIGN KEY(CourseID) REFERENCES Course ON DELETE CASCADE,
FOREIGN KEY(DeckID) REFERENCES Deck ON DELETE CASCADE,
```

```
CREATE TABLE UserSavesDeck
...
FOREIGN KEY(Username) REFERENCES User ON DELETE CASCADE,
FOREIGN KEY(DeckID) REFERENCES Deck ON DELETE CASCADE,
```

```
CREATE TABLE UserRatedCard
..
FOREIGN KEY(Username) REFERENCES User ON DELETE CASCADE,
FOREIGN KEY(DeckID) REFERENCES Deck ON DELETE CASCADE,
```

```
CREATE TABLE UserFlagsCard
..
FOREIGN KEY(UserID) REFERENCES User ON DELETE CASCADE,
FOREIGN KEY(DeckID) REFERENCES Deck ON DELETE CASCADE,
```