Assignment 4: Trigger

> Contributers:
>
> Designer: ZHU Yueming, WANG Shuo, REN Yiwei
>
> Idea provider: WANG Ziqin
>
> Online judge: XIANG Jiahong, LUO yunhao, ZHANG Taoyue

Introduction

An entrepreneur with absolutely no knowledge of databases bought a user account management system from one of our SAs. This system is dangerous if used for commercial purposes, because the user passwords are all stored in plaintext in the database. Recently a hacker hacked into the system and took out a large number of passwords, which brings users' passwords into danger. The entrepreneur learns that you are well versed in database technology and would like to ask you to help him improve the security of password storage.

You are strongly recommended to read https://crackstation.net/hashing-security.htm to learn how passwords should be properly stored and the reasons behind.

Preparation

This assignment requires the use of the prescribed encryption method.

1. Pgcrypto

The pgcrypto module provides cryptographic functions for PostgreSQL. To enable this module for a database, you need to execute this statement:

```
create extension if not exists pgcrypto;
```

Verify that it is enabled:

```
select * from pg_extension where extname = 'pgcrypto';
```

Note that the module version we use is 1.3.

2. Functions

The functions `crypt()` and `gen_salt()` are specifically designed for hashing passwords. `crypt()` does the hashing and `gen_salt()` prepares algorithm parameters for it.

Learn more from

2.1 Gen_salt ( )

```
gen_salt(type text) returns text
```

Return

A new random salt string for use in `crypt()`. This salt string also tells `crypt()` which algorithm to use.

Usage

This assignment requires you use *blowfish* hashing algorithm. When you invoke the function `gen_salt()`, the parameter must be 'bf', i.e.,

```
gen_salt('bf');
```

2.2 Crypt ( )

Suppose `123456` is the plaintext of password, and 'bf' is the hashing algorithm, it returns a salted hash of the password with a length of 60 characters.

```
select crypt('123456', gen_salt('bf'));
```

Return

A hash of password.

Usage

1. When storing a new password, you need to use `gen_salt()` to generate a new *`salt`* value.
2. To check a password, pass the stored hash value as *`salt`*, and test whether the result matches the stored value.

Example

For example, you want to create an account named `zhuym` with a password of `Db123456`. When you store this password, you have to use

```
insert into account (id,username, password)
values (30001219, 'zhuym', crypt('Db123456', gen_salt('bf')));
```

When you want to check if an input password (the string literal `'Db123456'` here) is equal to the password hash (the `password` column here) stored in the table (the `account` table here), you have to use

```
select (password = crypt('Db123456', password)) as password_match
from account
where username = 'zhuym';
```

3. Create two tables

3.1 account

```
create table if not exists account
(
    id       int primary key,
    username varchar(20)  not null,
    password varchar(60)  not null,
    role varchar(20)
);
```

Column Name	Type	Constraint	Description
id	int	Primary key	Student id or faculty id
username	varchar(20)	Not null	Name
password	varchar(60)	Not null	Salted password hash
role	varchar(20)		School student, exchange student or teacher

3.2 account_log

```
create table if not exists account_log
(
    user_id     int,
    update_date timestamp,
    password    varchar(60) not null ,
    primary key (user_id, update_date),
    foreign key (user_id) references account (id)
);
```

Column Name	Type	Constraint	Description
user_id	int	Foreign key, primary key	references account (id)
update_date	timestamp	Primary key	update time
password	varchar(60)	Not null	previous password hash

The task you need to accomplish

Step 1. Register Account

1.1 ID checking

The **length of student ID should be 8**, other than that, we can mark sure that all ID in test cases are valid.

1.1 Password checking

When a user wants to create a new account, you have to check if the password he submitted meets the requirements.

1. The password contains both uppercase letters, lowercase letters, and digits.
2. The length of the password is larger than 8 digits.
3. The password cannot contain user's username.
4. Passwords cannot contain characters other than upper and lower case letters, digits, underscores(`_`), asterisks (`*`), and dollars (`$`).

1.2 Analyze the role of account

If the password meets these requirements, you have to insert a new row into the account table. Please note that you need to analyze the first digit of the ID to determine the role of the account (**Case sensitive !**).

We can make sure that the first digit of ID in test cases only can be 1,2,3 or 5.

First digit of ID	Role
1	School students
2	Exchange students
3 or 5	Teacher
Other	null

Insert test cases:

```
insert into account(id, username, password) values (30020824,'Molly','abAB12e');
-- less than 8 digits
insert into account(id, username, password) values
(50020825,'Harmony','abcdefgh'); -- only lowercase
insert into account(id, username, password) values
(30020826,'Bright','Abcaac12'); -- success
insert into account(id, username, password) values
(50020827,'Firm','abc123asd'); -- no uppercase
insert into account(id, username, password) values
(11913558,'Roswell','123578AbdeFa'); -- success
insert into account(id, username, password) values
(22011249,'Robust','asdaA12999999'); -- success
insert into account(id, username, password) values
(11937541,'Ross','Ba1sad19a'); -- success
insert into account(id, username, password) values
(11991451,'Sirena','ACzc9O1a'); -- success
insert into account(id, username, password) values
(11841390,'Frederick','asdasdzc12esLsadz'); -- success
insert into account(id, username, password) values
(11751923,'Wesley','aszcAsa12sd'); -- success
insert into account(id, username, password) values
(50019827,'Lloyd','zxcA21_9asd'); -- success
insert into account(id, username, password) values
(50019981,'Bob','123Bob2ads'); -- include user's own name
insert into account(id, username, password) values
(30891203,'Smith','_asd*91Ab'); -- success
insert into account(id, username, password) values
(11913724,'Zoo','Abc12&}asad'); -- include invalid character "}"
insert into account(id, username, password) values
(1194567,'HUANG','hasd19Hasd'); -- too short
```

Step 2. Change Password

When a user who already has an account wants to change his/her password, you have to check

1. if the new password submitted meets the requirements mentioned above.
2. the new password cannot be the same as the last three successfully set passwords (In account_log table). If the number of successful updates for this user is less than 3, then the new password cannot be the same as all previous successful settings.
3. A user can update his password to the same one he has now. For example, if the password is "abcdEF12" now, it is allowed to change the password to "abcdEF12". In this case, "abcdEF12" will be stored in the table `account_log`. If you still change the password to "abcdEF12" next time, it will trigger an exception.

If the password meets these requirements, you have to update the account table and insert a new row into table `account_log`.

Update test cases:

```
-- Initial Password (id=30020826): Abcaac12
update account set password = '' where id = 30020826; -- fail, the new password
is too short
update account set password = 'Absazxc1213' where id = 30020826; -- success, the
new password is different from the old "Absazxc1213"
update account set password = 'Abcaac12' where id = 30020826;  -- fail, the new
password is the same as one of the previous three passwords
update account set password = 'Absazxc1213' where id = 30020826; -- success, the
new password is the same as last password, but log table has added a new row
update account set password = 'Absazxc1213' where id = 30020826; -- fail, the
new password is the same as the previous two passwords
update account set password = 'Abasd12_a' where id = 30020826; -- success, the
new is different from the previous three
update account set password = 'asd1sadA8z' where id = 30020826; -- success
update account set password = 'Absazxc1213' where id = 30020826; -- fail
update account set password = 'Abcaac12' where id = 30020826; -- success, the
new is different from the previous three
update account set password = 'onjkn98_Q'where id = 30020826; -- success
update account set password = 'Absazxc1213' where id = 30020826; -- success

-- Finally, password is equals to "Absazxc1213"
```

Submission

1. There is a limit to the **number and name** of functions you can submit.

Name	Description
`password_trigger`	Before trigger, check passwords before inserting and updating
`updatePwd_trigger`	After trigger
`on_update`	Function used by `updatePwd_trigger`
`password_check`	Function used by `password_trigger`

2. The above four functions are submitted in four separate files with the file extension ".sql".
3. For testing purposes, we give the framework of the functions, which you need to complete.

```
create or replace function on_update()
    returns trigger
as
$$
begin
    -- write your sql here
end
$$ language plpgsql;
```

```
create or replace function password_check()
    returns trigger
as
$$
    declare
                -- declare variables here
begin
                -- write your code here
end
$$ language plpgsql;
```