Interactive - 37 - encrypted/hashed passwords storage

It is a good idea to store passwords in a way that if the entire database is lost you do not loos anybody's password. You need to hash them or encrypt them in the table.

There are hash functions that are designed for this. Of of these is
the blowfish algorithm. It is denoted to in the encryption parameters as 'bf'.

First, we need to enable pgcrypto:

```
CREATE EXTENSION if not exists pgcrypto;
```

Then, we can create a table for storing user credentials:

```
DROP TABLE IF EXISTS example_users ;

CREATE TABLE example_users (
    id SERIAL PRIMARY KEY,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL
);


CREATE OR REPLACE function example_users_insert()
RETURNS trigger AS $$
DECLARE
    l_salt text;
    l_pw text;
BEGIN
    select gen_salt('bf')
        into l_salt;
    l_pw = NEW.password;
    NEW.password = crypt(l_pw, l_salt);
    RETURN NEW;
END
$$ LANGUAGE 'plpgsql';


CREATE TRIGGER example_users_insert_trig
BEFORE insert or update ON example_users
FOR EACH ROW
EXECUTE PROCEDURE example_users_insert();
```

When we insert into the table we can then save the hash of the password instead of the password itself.

```
INSERT INTO example_users (email, password) VALUES
    ( 'pschlump@uwyo.edu', 'my-very-bad-password')
;
```

When we want to validate that a password that has been passed in is correct we compare to the hashed value with:

```
SELECT id
    FROM example_users as t1
    WHERE t1.email = 'pschlump@uwyo.edu'
      AND t1.password = crypt('my-very-bad-password', t1.password)
;
```

This would be a good candidate to encapsulate into a stored procedure.

```
CREATE or REPLACE FUNCTION login_correct ( un varchar, pw varchar )
RETURNS varchar
AS $$
DECLARE
    data text;
BEGIN

    BEGIN
        SELECT 'VALID-USER'
            INTO data
            FROM example_users as t1
            WHERE t1.email = un
              AND t1.password = crypt(pw, t1.password)
            ;
        IF not found THEN
            data = 'Incorrect username or password.';
        END IF;
    EXCEPTION
        WHEN no_data_found THEN
            data = 'Incorrect username or password.';
        WHEN too_many_rows THEN
            data = 'Incorrect username or password.';
        WHEN others THEN
            data = 'Incorrect username or password.';
    END;

    RETURN data;
END;
$$ LANGUAGE plpgsql;

select login_correct ( 'pschlump@uwyo.edu', 'my-very-bad-password' );

select login_correct ( 'pschlump@uwyo.edu', 'my-VERY-bad-password' );
```

Tags: “password”,“hash password”,“encrypted”

Validate: SQL-Select,“select ‘PASS’ as x”