

CS 386/586 ASSIGNMENT 5 FALL 2011

Part I: Checking Constraints

Sometimes it is necessary or desirable for an application to check constraints on a database, for example, when the DBMS cannot enforce a particular type of constraint. Another example is a “soft constraint”, where we do not want to absolutely forbid a violation, but we want to know if one occurs.

One approach to checking a constraint C is to derive a query Q_C that will detect violations of C . Then, if Q_C returns an empty result, we know the current database satisfies the constraint.

Questions 1-5 below involve creating queries to check different kinds of constraints on the schema below, which corresponds to this (partial) ER diagram:



`Emp(empID: integer, eName: string, age: integer, salary: real)`

`Works(empID: integer, deptID: integer, fraction: real)`

`Dept(deptID: integer, budget: real, managerID: integer, dName: string)`

Question 1 (10 points): Give an SQL query that checks if `eName` is a key for `Emp` by finding every `eName` value that appears in two or more rows.

```
SELECT  eName
FROM    Emp
GROUP BY eName
HAVING  COUNT (*) > 1
```

Question 2 (10 points): Give an SQL query that checks the foreign key from Works to Emp by finding any rows in Works that do not connect to some row in Emp.

```
SELECT *
FROM Works
WHERE empID NOT IN ( SELECT empID
                    FROM Emp )
```

Question 3 (15 points): Give an SQL query that checks the constraint that every department has at least one employee, by finding any Dept rows that are not related to at least one Emp row.

```
SELECT *
FROM Dept
WHERE deptID NOT IN ( SELECT deptID
                    FROM Emp NATURAL JOIN Works )
```

Question 4 (15 points): Give an SQL query that checks the constraint that an employee can work in at most five departments.

```
SELECT empID
FROM Emp NATURAL JOIN Works NATURAL JOIN Dept
GROUP BY empID
HAVING COUNT (deptID) > 5
```

Question 5 (15 points): Give an SQL query to check the constraint that no employee works more than full time (that is, the fractions for that employee add up to no more than 1.0).

```
SELECT empID
FROM Emp NATURAL JOIN Works NATURAL JOIN Dept
GROUP BY empID
HAVING SUM (fraction) > 1
```

Part II: Injection Attacks

Questions 6 and 7 concern a PHP program that accesses a table with the following schema:

```
Account(custID: integer, AcctNum: integer, AType: string, Balance: real)
```

Consider a PHP page that makes use of two variables:

`$UID` – a string set to a customer ID by a login routine

`$ANum` – string that the user provided in a web form

Suppose we use these two strings in creating a query:

```
<?php
$query = "SELECT *
        FROM Account
        WHERE custID = $UID AND AcctNum = $ANum"
?>
```

This query is then submitted to the database and the results printed.

Question 6 (10 points): Explain how this scenario allows a customer of the bank to get the account balance for the account of another customer.

Suppose the user with customer ID12345 wants to get information on account 22777. Then he or she could supply a value for `$ANum` of “00000 OR AcctNum = 22777”, which would result in the query:

```
SELECT *
FROM Account
WHERE custID = 12345 AND AcctNum = 00000 OR AcctNum = 22777
```

Since the AND operator has a higher precedence than OR operator in SQL, the information on account 22777 in *Account* will be retrieved in this case.

Question 7 (10 points): Suppose we try to ensure that the `$ANum` string contains a number, by using the query below. Is it still possible for a customer to access another customer's balance?

```
<?php
$query = "SELECT *
        FROM Account
        WHERE custID = $UID AND AcctNum = ($ANum * 1)"
?>
```

Yes, it is possible.

Suppose the user with customer ID12345 wants to get information on account 22777. Then he or she could supply a value for `$ANum` of "00000) OR (AcctNum = 22777", which would result in the query:

```
SELECT *
FROM Account
WHERE custID = 12345 AND AcctNum = (00000) OR (AcctNum = 22777 * 1)
```

And the information on account 22777 in *Account* will be retrieved in this case.

Question 8 (10 points): Describe two ways to prevent such "SQL injection attacks" in PHP. You are free to consult any sources you want, but give the answer in your own words, and cite any document you consult.

1. Never trust user input: validate all textbox entries using validation controls, regular expressions, code, and so on.
2. Never use dynamic SQL: use parameterized SQL or stored procedures

Reference:

<http://msdn.microsoft.com/en-us/magazine/cc163917.aspx>