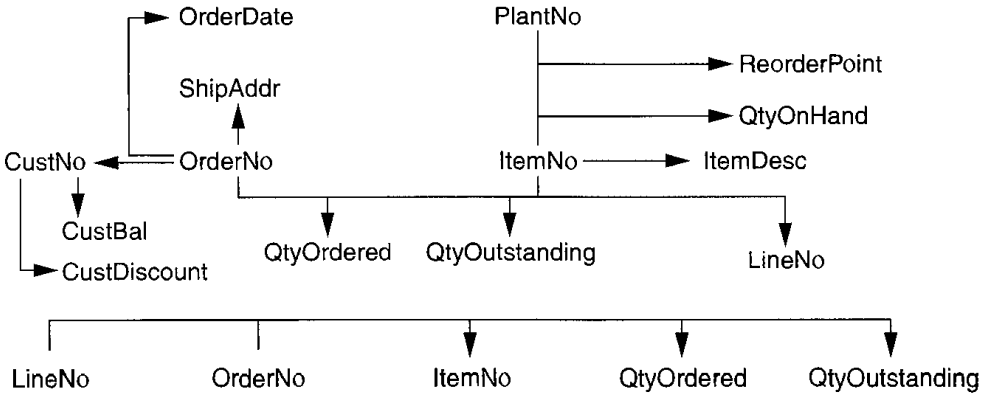
COMP 3335 – Database Security (Fall 2024) Assignment

This is an individual assignment. You may use the course material and Internet resources to answer the questions. However, you should not post the questions online and ask for help. GenAI tools should NOT be used to help you with this assignment. Copying or paraphrasing answers from GenAI tools into your assignment is a form of plagiarism. Discussion among your peers is encouraged; however, you must produce answers by yourself and in your own words. Never share code and textual answers with your peers. Any suspicion of plagiarism will be thoroughly investigated. This assignment is due before Saturday, **Oct 26, 2024**, 23:59.

Late submissions will be subjected to a 15% penalty per day, starting at 00:01.
Total: 100 marks. Course weight: 10%. *Version updated Oct 2 evening.*

Question 1: Functional Dependency and Normalization (30 marks)

The FD diagram below depicts relationships among attributes in an order entry database. It shows FDs with determinants *CustNo, OrderNo, ItemNo,* the combination of *OrderNo* and *ItemNo,* the combination of *ItemNo* and *PlantNo,* and the combination of *OrderNo* and *LineNo.* In the bottom FDs, the combination of *LineNo* and *OrderNo* determines *ItemNo* and the combination of *OrderNo* and *ItemNo* determines *LineNo.*



(a) Convert the dependency diagram into a list of dependencies organized by LHSs. (6 marks)
(b) Using the FD diagram and the FD list (the solution of (a)) as guidelines, make a table with sample data. There are two candidate keys for the underlying table: the combination of *OrderNo, ItemNo,* and *PlantNo* and the combination of *OrderNo, LineNo*, and *PlantNo*. Using sample data, identify insertion, update, and deletion anomalies in the table. (6 marks)
(c) Derive 2NF tables starting with the FD list from the answer of (a) and (b). (4 marks)
(d) Derive 3NF tables starting with the FD list from the answer of (a) and the 2NF tables from (c). (4 marks)
(e) Find out the possible BCNF tables with the information in (a) to (d). (4 marks)
(f) Modify your table design if the shipping address (*ShipAddr*) attribute determines customer number (*CustNo*). Do you think that this additional FD is reasonable? Briefly explain. (6 marks)

Question 2: Securing Application Passwords [28 marks]

While it is an industry standard to hash account passwords using a specialized cryptographic one-way hash function, certain scenario may require access to the plaintext password, e.g., research studies on password selection.

a) Explain how symmetric encryption can be used to store passwords in a non-plaintext form so an attacker who is able to *dump* the database content cannot access the plaintext passwords, while permitting administrators to access the plaintext. Also explain where the relevant keys should be stored. (8 marks)

b) Is this method of storing password immune to dictionary attacks? Why? (8 marks)

c) Given the following database schema for storing plaintext account passwords, and the associated PHP code to authenticate a user:
   1. Provide a modified database schema with only the necessary changes to implement your solution given in a). (6 marks)
      Provide this schema in a separate file named `q2-schema.sql`.
   2. Provide the modified PHP code to perform authentication under your new method, with only the necessary changes. (6 marks)
      Provide this code in a separate file named `q2-auth.php`.

The existing SQL table (see attached `q2-schema-unsafe.sql`):

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL, -- Storing plaintext passwords (unsafe)
    email VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

The PHP code snippet to perform authentication (see attached `q2-auth-unsafe.php`):

```
<?php
// Database connection
$conn = new mysqli('localhost', 'username', 'password', 'ecommerce_db');

// Retrieve user input (from login form)
$username = $_POST['username'];
$password = $_POST['password'];

// Use prepared statements to prevent SQL injection (see relevant lecture)
$query = "SELECT * FROM users WHERE username = ? AND password = ?";
$stmt = $conn->prepare($query);
$stmt->bind_param("ss", $username, $password);
```

```php
$stmt->execute();
$result = $stmt->get_result();

// Check if a user was found
if ($result->num_rows > 0) {
    // Fetch user data
    $user = $result->fetch_assoc();

    // Authentication successful
    echo "Welcome, " . htmlentities($user['username']) . "!";
} else {
    // Authentication failed
    echo "Invalid username or password.";
}

// Close connection
$stmt->close();
$conn->close();
```

Question 3: Securing Data-in-motion in MySQL [42 marks]

This question aims to familiarize students with securing data transmission between a MySQL client and server using encrypted connections. By the end of this question, students should be able to create, implement, and verify encrypted connections in MySQL.

Installing OpenSSL (version 1.1.x or 3.x):

- **Windows**: Download the compiled x64 binaries from Blackboard (taken from Git).
- **macOS**: Try first the command `openssl version` in a terminal. If the command fails, install OpenSSL using Homebrew with the command: `brew install openssl`.
- **Ubuntu**: Similarly, try the command `openssl version` in a terminal. If it fails, install OpenSSL using the command: `sudo apt-get install openssl`.

Refer to the online documentation at https://www.openssl.org/docs/manpages.html according to the version you are using for the questions below.

Installing MySQL Community Edition: Follow the instructions on this page to install MySQL Community Edition 8.0.16+. Download the binaries from this link. Note that you do not need an account to download, simply click on "No thanks, just start my download." Connect to your MySQL Server using the root account. Then, execute the command `SHOW VARIABLES LIKE '%ssl%'` and look at the line `has_ssl`. If not YES, then your MySQL installation does not support SSL/TLS as it should. Please make sure to upgrade your installation to version 8.0.16 or above.

(a) Connect to your MySQL Server using `mysql -u root -p -h localhost --ssl-mode DISABLED`. Then, execute the command `STATUS` (or `\s` for short) and give a screenshot showing the lines "SSL" and "Server version." (5 marks)

(b) In MySQL's documentation, read about the ssl-mode option, and change the mode from `DISABLED` to a mode that forces the use of TLS but does not verify the server certificate. Which mode do you choose? Check the connection status again and report a new screenshot showing the value for "SSL." (5 marks)

(c) Using the command `SHOW VARIABLES LIKE '%ssl%'`, find the TLS server certificate filename that the server is configured to use. Locate this file on disk, then use OpenSSL to parse the certificate and show you its content. Give the value of the Issuer of the certificate. Hint: The certificate follows the X.509 standard, so look into `openssl x509`. (5 marks)

(d) If the client does not verify the server's certificate, an adversary could actively interpose on the connection and pretend to be the server. The client would still report that the connection is encrypted. By enforcing certificate validation, we can prevent this attack. Utilize OpenSSL to create your own CA certificate (and corresponding private key), along with **two** server certificates (and private keys) issued by your CA: one with the Common Name = IAmAHacker.com, the other with Common Name = localhost. Note down the exact commands you used. (12 marks)

   a. Hint 1: Use `openssl ecparam -out <outfile> -name prime256v1 -genkey` command to create an elliptic curve (EC) private key file.
   b. Hint 2: Use `openssl req -new -key <keyfile> -out <CSR>` command to create a certificate signing request (CSR) to be signed by the CA.
   c. Hint 3: Use `openssl req -x509 -new -key <keyfile> -out <certificate> -days 365` command to create a self-signed certificate using the provided private key. Note that the CA certificate is self-signed, i.e., it is signed by its own private key.
   d. Hint 4: Use `openssl x509 -req -in <CSR> -CA <CA> -CAkey <keyfile> -CAcreateserial -out <CRT> -days 365` to sign a CSR by the CA's private key and create a certificate.
   e. Hint 5: Use the parameter `-config openssl.cnf` to point to the config file attached to this assignment with `openssl req` commands.

(e) Configure explicitly the MySQL server and client for encrypted connections using the certificates created in the previous step (select the first server certificate created) by writing configurations to my.cnf (or my.ini on Windows) under `[mysqld]` for the server side and under `[mysql]` for the client side. The server should mandate TLS encryption. Show the lines you added to the config file. (10 marks)

(f) For each server certificate created in (d), connect to your MySQL Server using `--ssl-mode VERIFY_CA` and `VERIFY_IDENTITY` and report on the success or failure. Make sure potential failures are expected and not the result of an issue in your configuration. Explain

the importance of the `--ssl-mode` setting on the client side and how it affects the verification process. Make sure to use `-h localhost` just as in Q1. (5 marks)