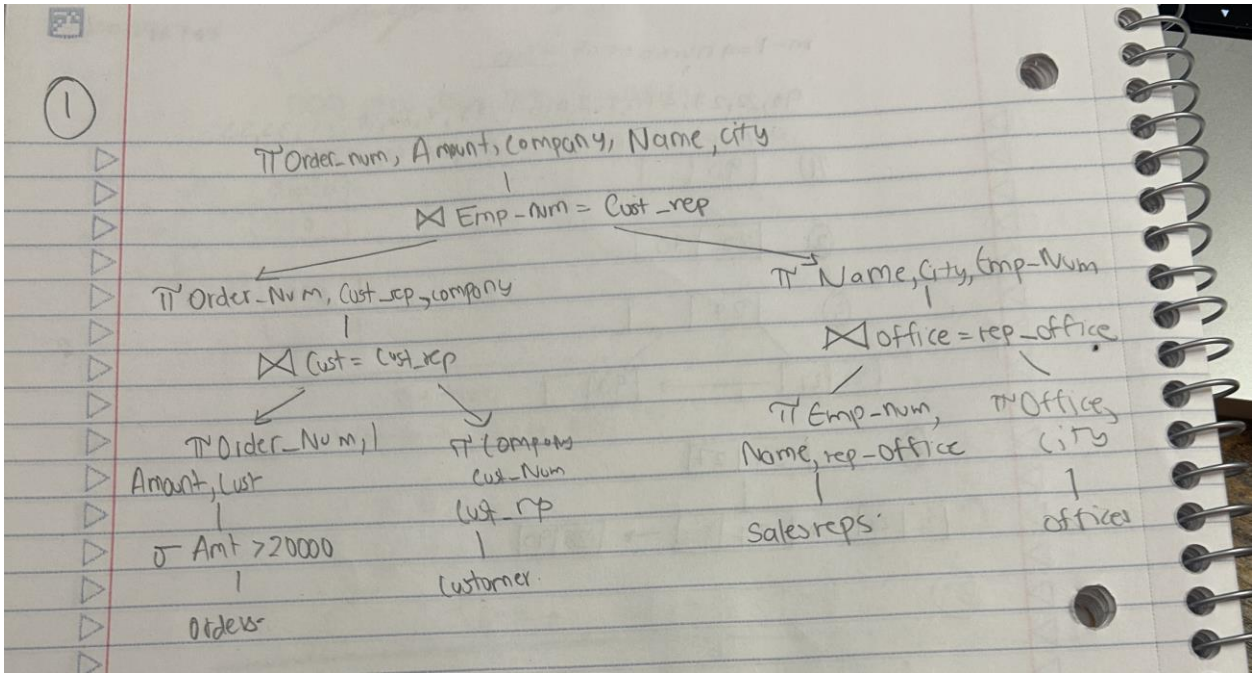
Lab 6

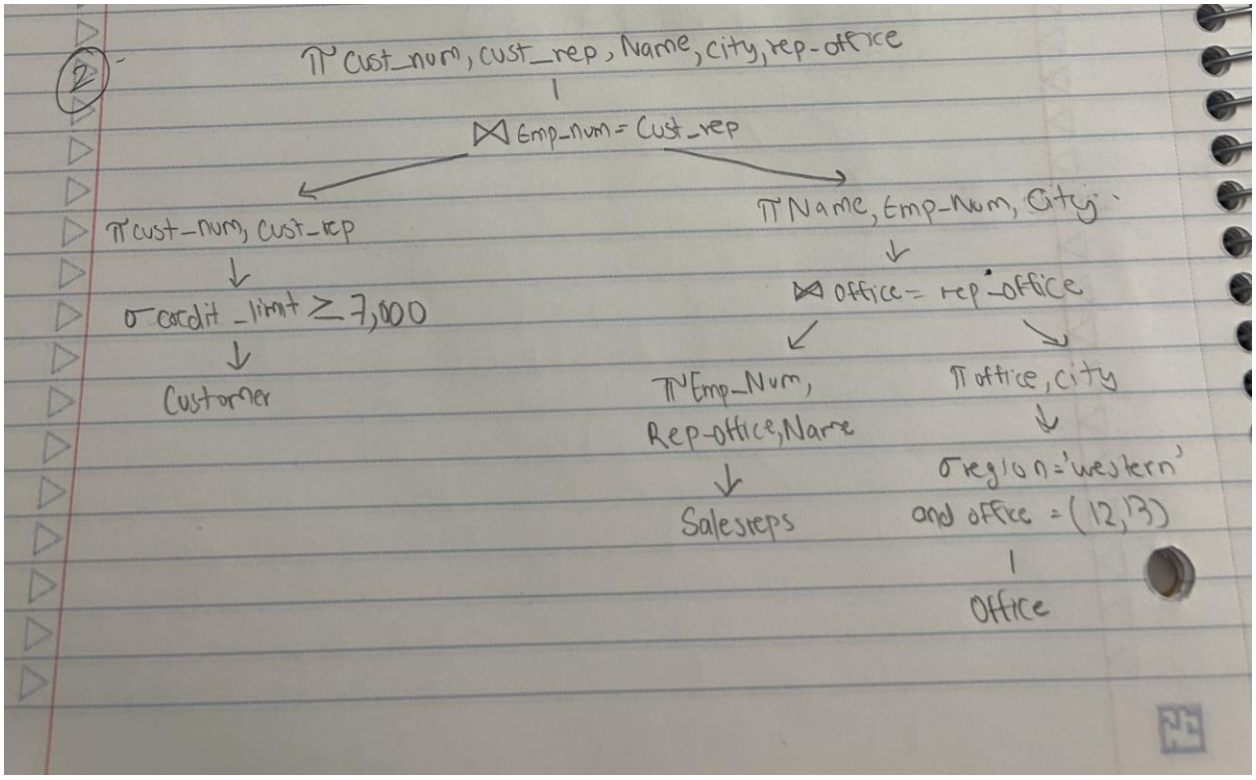
1) Draw the optimization tree for the following query.

```
SELECT Order_Num, Amount, Company, Name, City
FROM Orders, Customers, Salesreps, Offices
WHERE Cust = Cust_Num
AND Cust_Rep = Empl_Num
AND Rep_Office = Office
AND Amount > 20000
```



2) Draw the optimization tree for the following query.

```
SELECT cust_num, cust_rep, name, rep_office, city
FROM Customers, Salesreps, Offices
WHERE empl_num = cust_rep, and rep_office = office
AND credit_limit >= 7000
And region = “western”
And office in (12, 13)
```



3) Some of frequent attacks on the database include: Unauthorized Privilege Escalation, Privilege Abuse, Denial of Service, and Weak authentication. Explain Privilege Abuse and weak Authentication is a few lines. **(Slide 2)**

   Privilege abuse: This is done by authorized (privileged) users rather than unauthorized users. For example, database administrator intentionally changes some student's grade.

   Weak Authentication: If the user authentication scheme (such as userid and password) is weak, an attacker can impersonate or copy the identity of a legitimate user by obtaining the login credentials of the legitimate user.

4) One form of Security Injection Attack is called "SQL manipulation". Suppose we have the following query

   SELECT *
   FROM users
   WHERE username = 'user_input' AND password = 'password_input';

   How can an attacker insert a code in the above query to retrieve the database data without authorization? **(Slide 4)**

   SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '' OR '1'='1';

   The attacker is able to log in without needing a valid username or password, because the query is always true due to the OR '1'='1' condition

5) Rewrite the following query with a php code to prevent the attack **(Slide 6)**

   SELECT *
   FROM users
   WHERE username = 'user_input' AND password = 'password_input';

   $stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username AND password = :password"); $stmt->execute(['username' => $user_input, 'password' => $password_input]);

6) What are the four types of “Code Injection”? Just name them (no need to explain them). **(Slide 7)**
   - SQL Injection
   - Command Injection
   - Script Injection
   - Buffer Overflow

7) Provide examples of “Function Call Injection” where an attacker
   a. can find the user id of a database user. Just provide the query. **(Slide 14)**
   SELECT * FROM employees WHERE first_name = '' OR 1=1; SELECT user() --';
   b. can find the version of database. Just provide the query. **(Slide 15)**
   SELECT * FROM employees WHERE first_name = '' OR 1=1; SELECT version() --';
   c. can find the currently connected database. **(Slide 16)**
   SELECT * FROM employees WHERE first_name = '' OR 1=1; SELECT database() --';
   d. can drop a table in the database. **(Slide 17)**
   SELECT * FROM employees WHERE first_name = '' OR 1=1; EXEC xp_cmdshell('DROP TABLE table') --;

8) How to prevent the Function call Injection? Give a PHP code with pdo→Prepare function that blocks attacker from function call injection. **(Slide 19)**

```
$stmt = $pdo->prepare("SELECT * FROM employees WHERE first_name = :first_name");
$stmt->execute(['first_name' => $user_input]);
```

9) What are the typical risks associated with SQL Injection. Just name them, no need to provide any explanation. **(Slides 20, and 21)**
   - Database Fingerprinting
   - Denial of Service
   - Bypassing Authentication
   - Identifying Injectable Parameters
   - Executing Remote Commands
   - Performing Privilege Escalation

10) What are the three forms of protections against SQL Injection. Give example queries of each method. **(Slide 22, 23, and 26)**

    Bind Variables o The use of bind variables protects against injection attacks and also improves performance.

    example: PrepareStatement (Select * FROM EMPLOYEE WHERE EMPLOUYEE_ID=? AND PASSWORD=?"); Stmt.setString1.employeee_id; Stmt.setString2.password;

    Filtering input: this technique can be used to remove escape characters from input strings by using the SQL Replace function

Example: cursor.execute("SELECT * FROM users WHERE username = ?", (input_username_escaped,))

Function Security: • Database functions, both standard and custom, should be restricted, as they can be exploited, misused, or abused in SQL function Injection attacks

Example: cursor.execute("SELECT * FROM users WHERE username = ' " + user_input + " ' ")