Concept Of Relevant Security Standards On Relational Databases

Holong Marisi Simalango, A.Md., S.T., M.Kom,
Gilang Bagus Ramadhan, A.Md.Kom.
Ahmadi Irmansyah Lubis, S.Kom., M.Kom.

Learning Outcomes:

Students are able to **apply** practically from the concept of relevant security standards on relational databases

Learning Materials:

- Relational Database
- MySQL
- Threats in Relational Databases
- Basic Access Management

Relational Database

The core concepts of the **relational database model**, which is the most widely used way to organize data.

Essentially, this model stores data in **tables**.

Key Components:

- A **table** is also called a **relation**.
- Each **row** (or **record**) in a table represents a single, specific object or entity. For example, in a table for **Employees**, each row would be one individual employee.
- Each **column** (or **attribute**) represents a specific piece of information or characteristic about that entity. For example, the **Employees** table might have columns for **Name**, **Employee ID**, and **Job Title**.

MySQL®

MySQL is a relational database.

- Transactional
- InnoDB storage engine
- Online Transaction Processing (OLTP) : low latency, high throughput
- MySQL offers advanced security features to protect data integrity and privacy.

Threats in Relational Databases

Threats in Relational Databases

Relational databases face threats like

- SQL injection
- unauthorized data access, and
- data breaches.

```
irror_mod = modifier_ob.
mirror object to mirror
irror_mod.mirror_object
peration == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
 operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
 operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True

selection at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.activ
("Selected" + str(modifier
irror_ob.select = 0
 bpy.context.selected_obj
ata.objects[one.name].se

int("please select exactly

 OPERATOR CLASSES ----

types.Operator):
 X mirror to the selected
ject.mirror_mirror_x"
rror X"

ontext):
ext.active_object is not
```

SQL Injections (SQLi)

SQL injection is a cyber-attack technique where an attacker can execute malicious SQL statements in a web application's database. It primarily targets applications that have poor security practices in place and directly use user input in their database queries.

Three main classes of SQL Injection (SQLi) attacks.

1. In-Band SQLi. The attacker uses the same communication channel to both inject the code and retrieve data from the database : Error-based SQL Injection and Union-based SQL Injection
2. Blind SQLi. The attacker must infer the data indirectly based on the application's behavior : Boolean-based Blind SQL Injection and Time-based Blind SQL Injection.
3. Out of band SQLi. The attacker uses a different communication channel to exfiltrate the stolen data.

SQL Injections (How it works)

Vulnerable Input Fields:

- Web applications often take user input to query the database. If this input is not properly sanitized or validated, it can be exploited.
- Attackers insert malicious SQL code into these fields, hoping the system will run it.

Malicious Payloads:

- The attacker crafts payloads that the database will interpret as valid SQL commands.
- For instance, admin' OR '1'='1' in a login form can be interpreted as always true, potentially bypassing authentication.

Data Manipulation:

- Beyond just retrieving data, attackers can use SQL injection to modify, delete, or even create new data in the database.

Advanced Exploits:

- In more sophisticated attacks, SQL injection can be used to execute commands on the host operating system, potentially escalating to a full server takeover.

SQL Injections (Preventions)

Prepared Statements/Parameterized Queries:

- By using prepared statements, user input is always treated as data and not executable code. This means that even if an attacker tries to input malicious code, the database will not execute it as a command.
- For example, in PHP, one might use PDO (PHP Data Objects) to prepare statements and bind user input to them.

Web Application Firewalls (WAFs):

- These are security systems specifically designed to monitor, filter, and block data packets as they travel to and from a web application.
- A good WAF can detect typical SQL injection patterns and stop them before they reach the application.

Regular Code Reviews:

- Regularly reviewing and updating the application code helps in identifying potential vulnerabilities.
- Developers should be trained to recognize and avoid coding practices that lead to vulnerabilities.

SQL Injections (Preventions) (Cont.)

Limit Database Permissions:

- The principle of least privilege should be applied. This means giving only the necessary permissions needed for a task.
- For instance, a web application's database user should not have permissions to DROP tables or create new users. If an attacker gains access, their actions will be limited by these permissions.

Input Validation and Sanitization:

- Always validate user input. For instance, if you're expecting a phone number, ensure the input consists only of numbers.
- Sanitize input by removing or neutralizing characters that have special meanings in SQL.

Error Handling:

- Avoid showing detailed database error messages to users. These can provide attackers with clues about the database structure and configuration.
- Instead, use generic error messages and log the detailed errors for internal review.

Detecting SQL Injection Vulnerabilities

SIMPLE CHARACTERS

Inserting special character

‘, “, #, ;, /, and) Into input field.

A vulnerable application will often respond with a database error, an unusual change in the page's content, or a different response time, indicating that the input was processed as part of a SQL query

Unauthorized Data Access

How It Works:

Attackers might guess or crack weak passwords to gain access.

Exploiting vulnerabilities in the software or in the operating system on which it runs (Ex., SQL Injections).

Misconfigured settings might allow public access or give more permissions than intended

Preventions:

- Implement strong authentication mechanisms, including strong passwords and two-factor authentication.
- Regularly review user permissions and ensure the principle of least privilege: users should only have access to the data they need.
- Keep database software and related applications updated to patch known vulnerabilities.
- Employ network-level security measures like firewalls to restrict unauthorized access.

Data Breaches

A data breach occurs when confidential data is accessed, stolen, or released without authorization. This can be a result of both external attacks and internal mishandlings

How It Works:

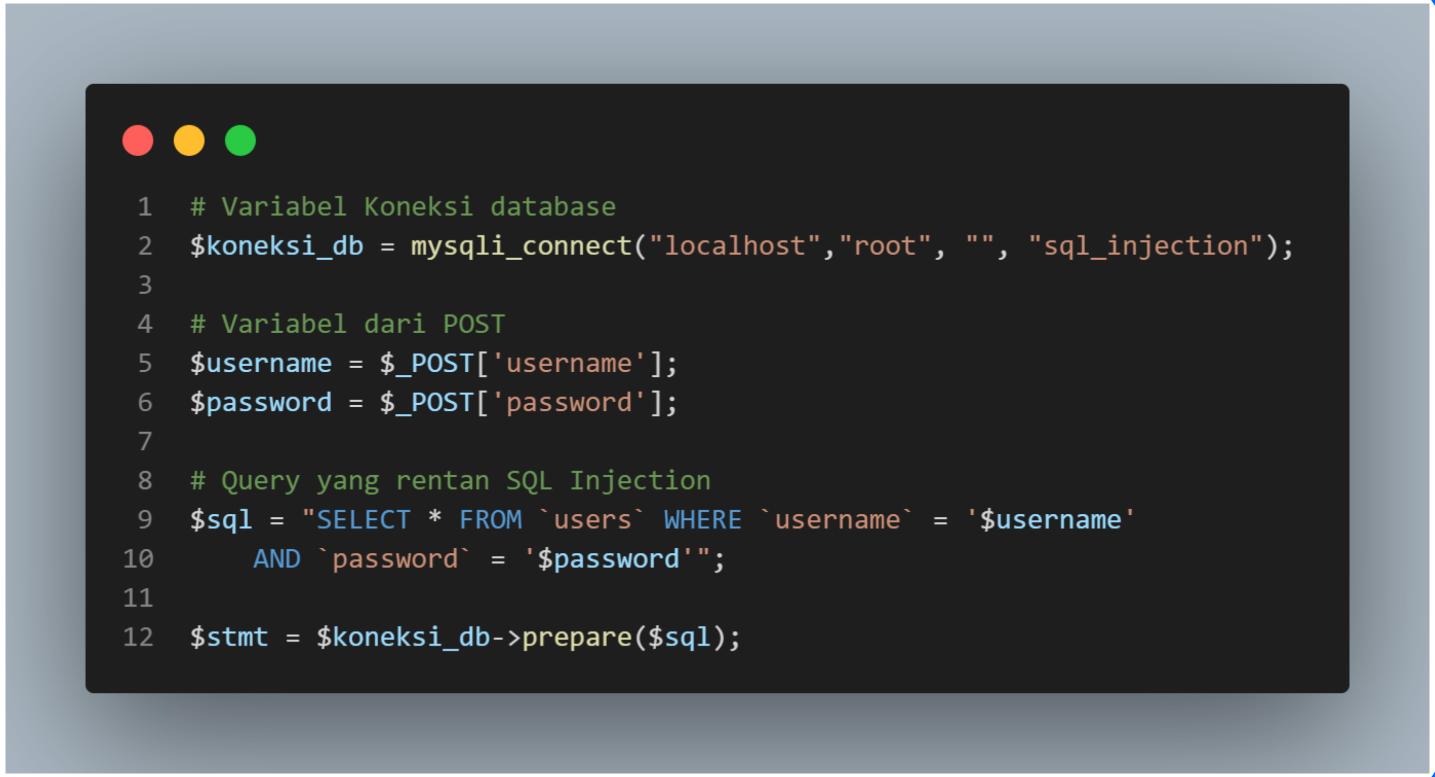
- External attackers might exploit vulnerabilities (SQL Injections, etc) , use phishing attacks to gain credentials, or employ malware to gain access.
- Insiders, such as disgruntled employees, might intentionally leak data.
- Accidental breaches can occur due to misconfigurations, unintended data sharing, or loss of physical devices containing data

Prevention:

- Encrypt sensitive data both at rest and in transit.
- Regularly backup data and ensure backups are secure.
- Train employees on security best practices and the importance of data confidentiality.
- Monitor database access and set up alerts for suspicious activities.
- Conduct regular security audits to identify and rectify potential vulnerabilities.

SESI PRAKTIKUM

Contoh kode SQL Injection Login



```php
# Variabel Koneksi database
$koneksi_db = mysqli_connect("localhost","root", "", "sql_injection");

# Variabel dari POST
$username = $_POST['username'];
$password = $_POST['password'];

# Query yang rentan SQL Injection
$sql = "SELECT * FROM `users` WHERE `username` = '$username'
    AND `password` = '$password'";

$stmt = $koneksi_db->prepare($sql);
```

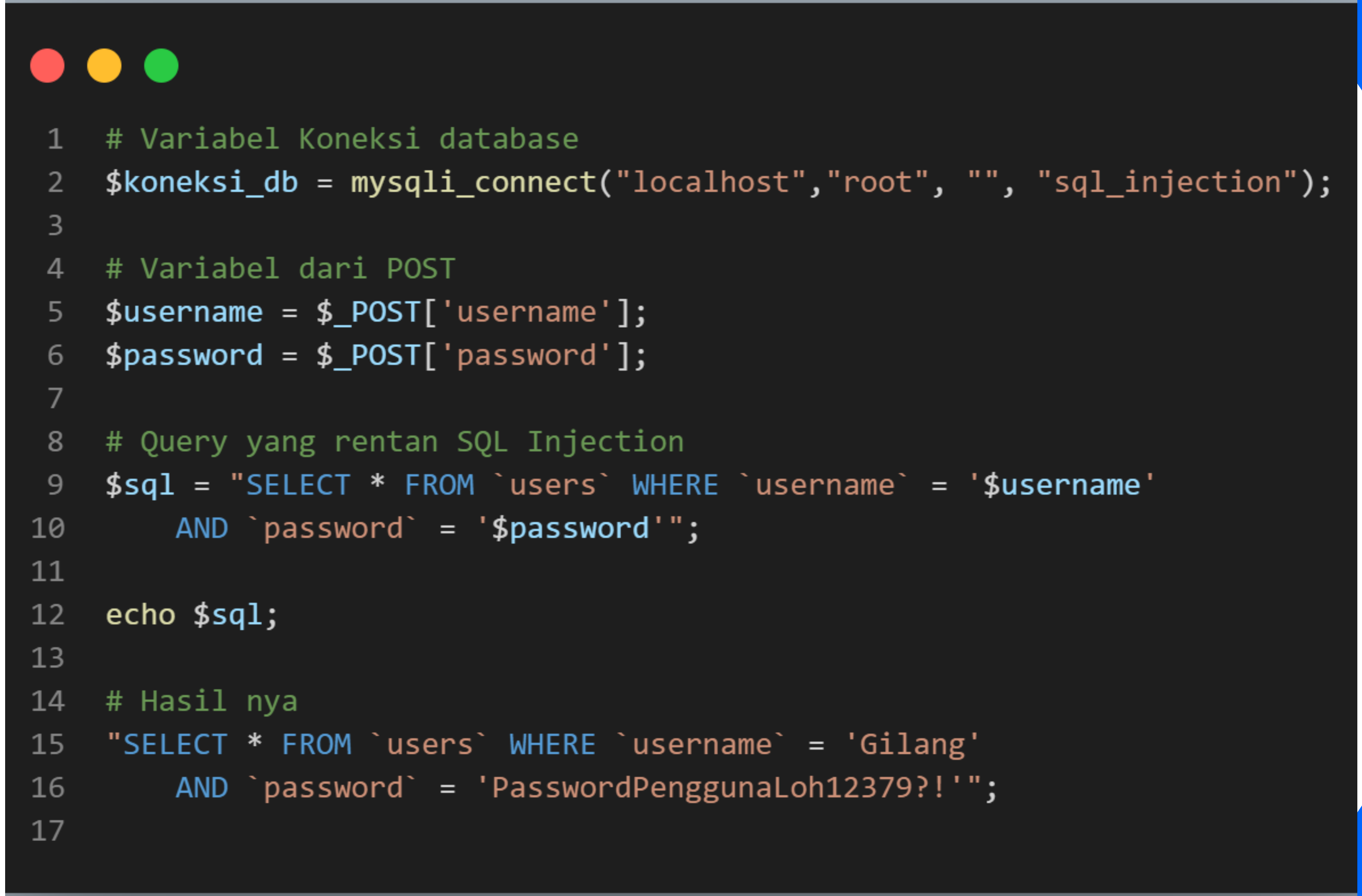
Simulasi Kode

Input Username = **Gilang**

Input Password = **PasswordPenggunaLoh12379?!**

Hasil nya

```
SELECT * FROM `users` WHERE
`username` = 'Gilang' AND
`password` =
'PasswordPenggunaLoh12379?!
'
```



```
# Variabel Koneksi database
$koneksi_db = mysqli_connect("localhost","root", "", "sql_injection");

# Variabel dari POST
$username = $_POST['username'];
$password = $_POST['password'];

# Query yang rentan SQL Injection
$sql = "SELECT * FROM `users` WHERE `username` = '$username'
    AND `password` = '$password'";

echo $sql;

# Hasil nya
"SELECT * FROM `users` WHERE `username` = 'Gilang'
    AND `password` = 'PasswordPenggunaLoh12379?!'";

```

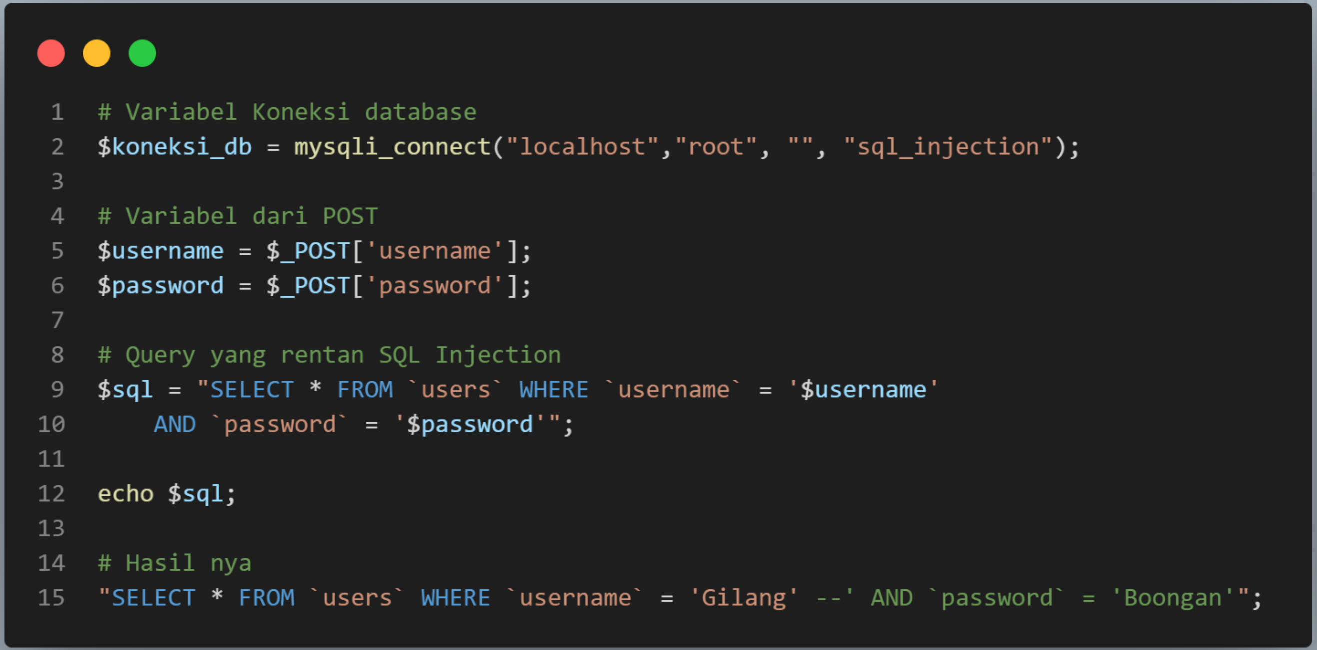
Simulasi Payload SQL Injection

Input Username = **Gilang' --**

Input Password = **Boongan**

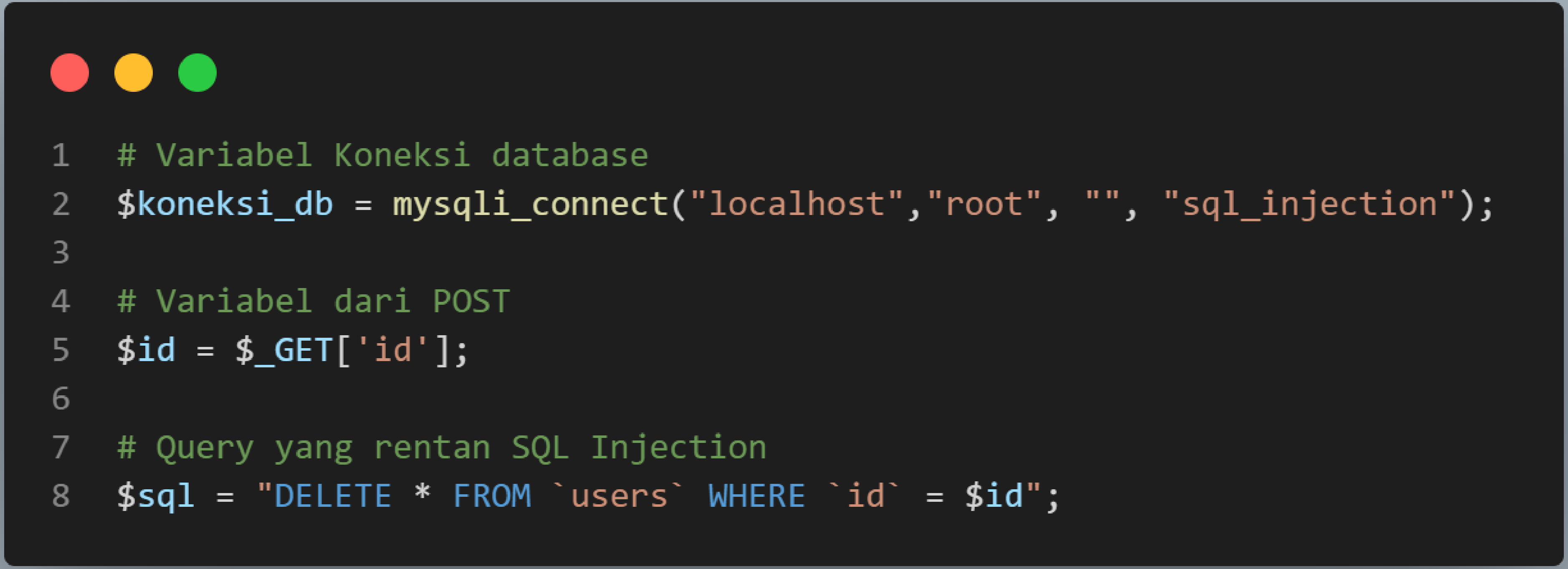
Hasil nya

```
SELECT * FROM `users`
WHERE `username` =
'Gilang ' --' AND
`password` = 'Boongan'
```



```
# Variabel Koneksi database
$koneksi_db = mysqli_connect("localhost","root", "", "sql_injection");

# Variabel dari POST
$username = $_POST['username'];
$password = $_POST['password'];

# Query yang rentan SQL Injection
$sql = "SELECT * FROM `users` WHERE `username` = '$username'
    AND `password` = '$password'";

echo $sql;

# Hasil nya
"SELECT * FROM `users` WHERE `username` = 'Gilang' --' AND `password` = 'Boongan'";
```

Efeknya adalah sintaks – pada SQL merupakan komen, yang artinya baris warna hijau akan terkomen dan tidak akan dianggap sebagai baris kode


```
# Variabel Koneksi database
$koneksi_db = mysqli_connect("localhost","root", "", "sql_injection");

# Variabel dari POST
$id = $_GET['id'];

# Query yang rentan SQL Injection
$sql = "DELETE * FROM `users` WHERE `id` = $id";
```

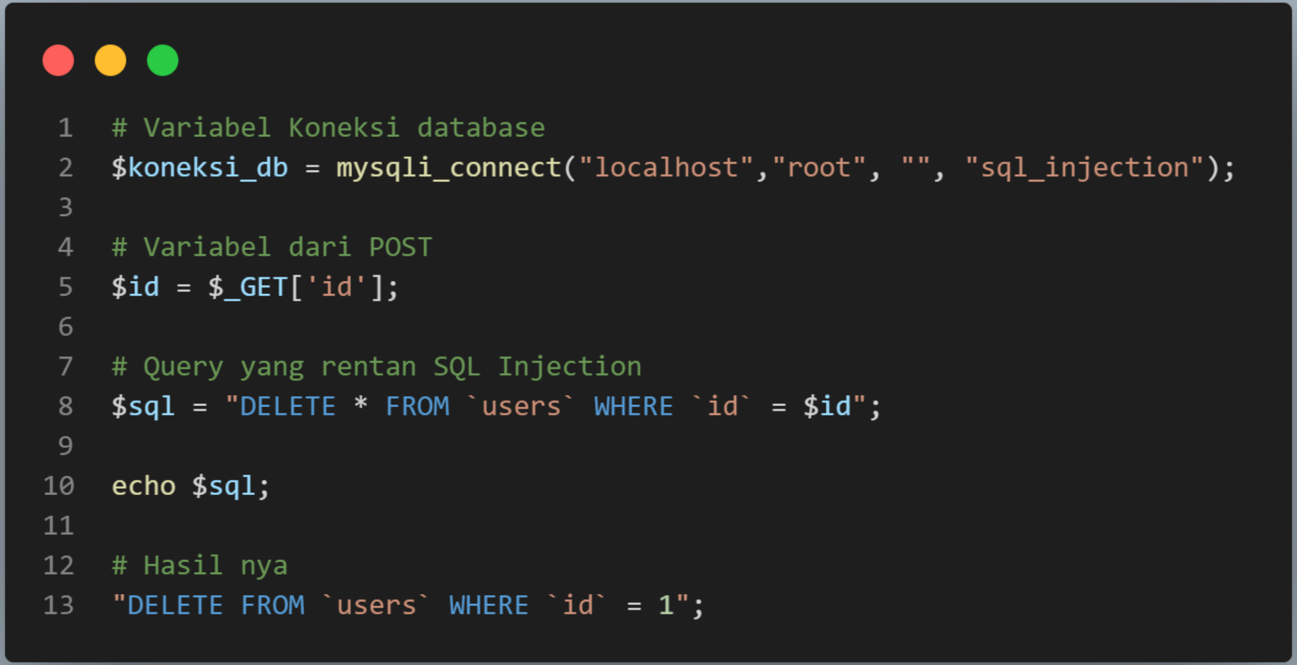
Simulasi Kode

Input Id = **1**

Input Password = **PasswordPenggunaLoh12379?!**

Hasil nya

```
DELETE FROM `users` WHERE `id` = 1
```



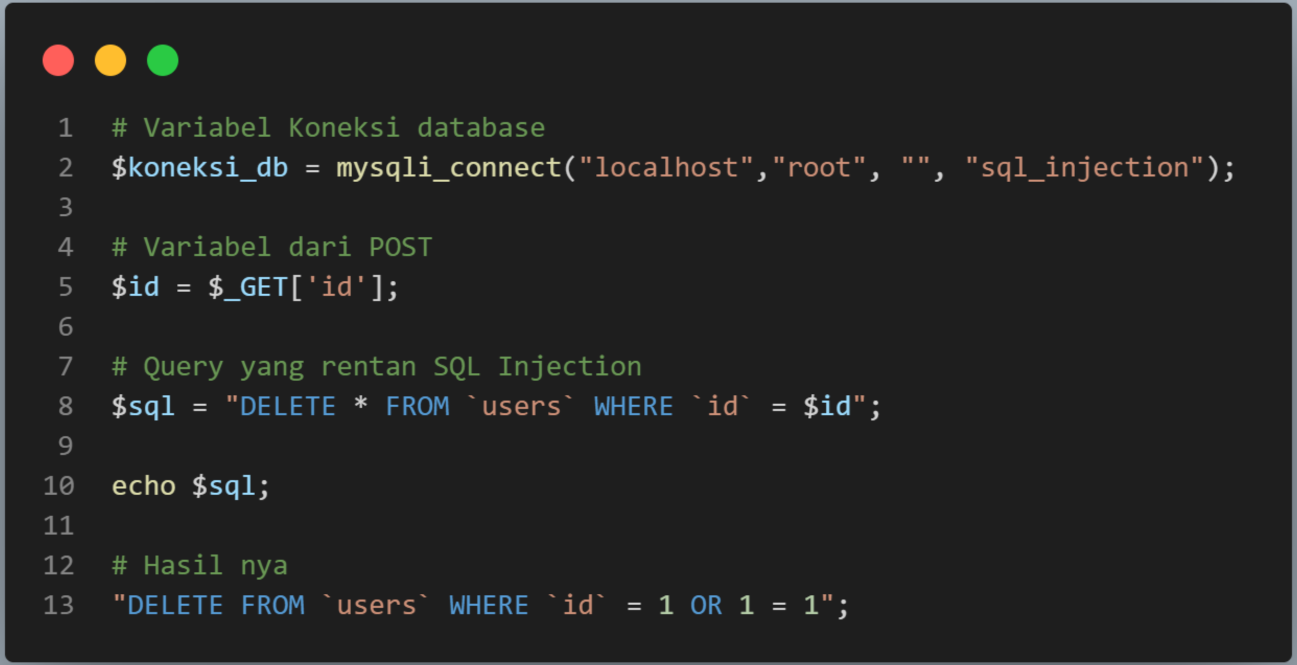
```php
# Variabel Koneksi database
$koneksi_db = mysqli_connect("localhost","root", "", "sql_injection");

# Variabel dari POST
$id = $_GET['id'];

# Query yang rentan SQL Injection
$sql = "DELETE * FROM `users` WHERE `id` = $id";

echo $sql;

# Hasil nya
"DELETE FROM `users` WHERE `id` = 1";
```

Simulasi Kode

Input Id = **1 OR 1 = 1**

Hasil nya

```
DELETE FROM `users`
WHERE `id` = 1 OR 1 =
1
```

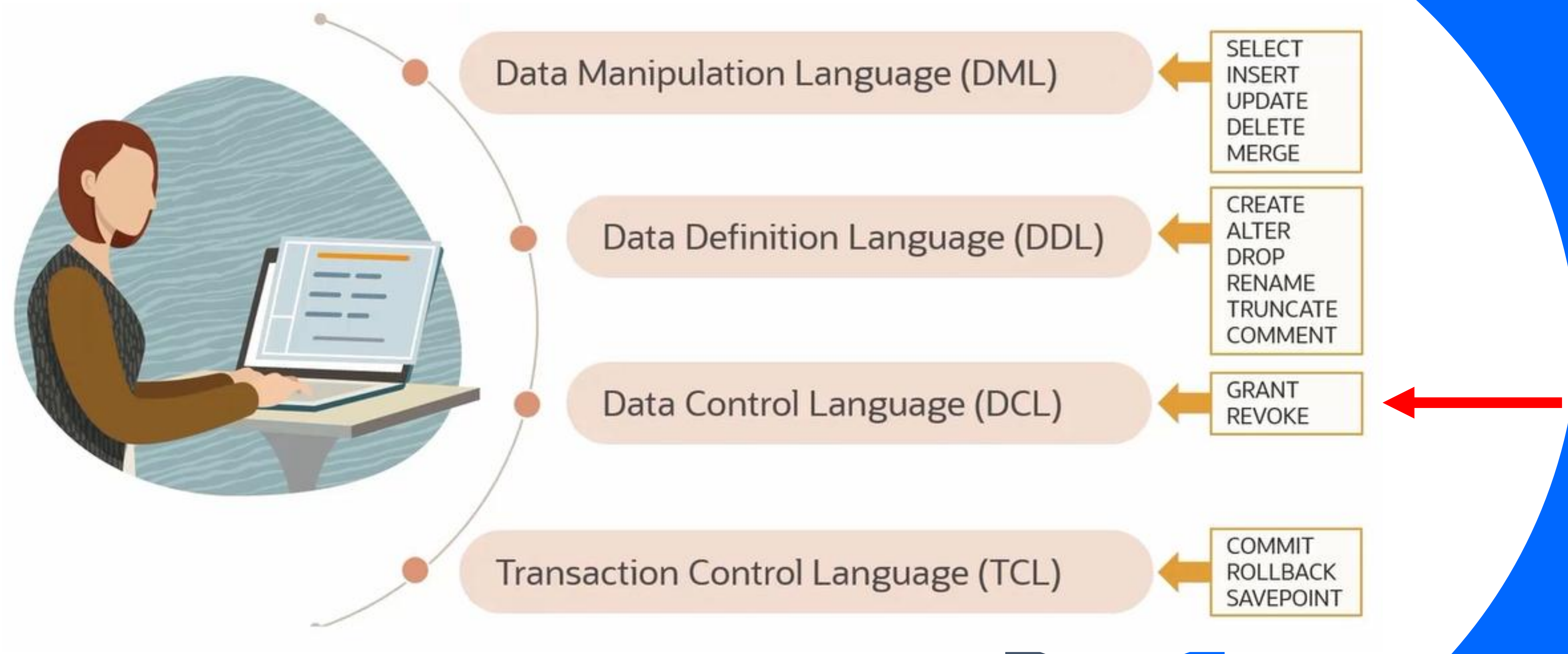


```
# Variabel Koneksi database
$koneksi_db = mysqli_connect("localhost","root", "", "sql_injection");

# Variabel dari POST
$id = $_GET['id'];

# Query yang rentan SQL Injection
$sql = "DELETE * FROM `users` WHERE `id` = $id";

echo $sql;

# Hasil nya
"DELETE FROM `users` WHERE `id` = 1 OR 1 = 1";
```

Efeknya adalah karena WHERE nya mengggunakan OR, walaupun User dengan id = 1 tidak ada, namun karena hasil 1=1 bernilai true, lalu penghubung OR, maka hasil nya akan sama seperti DELETE FROM user / Truncate data user

Basic Access Management

Tugas


Data Manipulation Language (DML)
SELECT
INSERT
UPDATE
DELETE
MERGE
Data Definition Language (DDL)
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT
Data Control Language (DCL)
GRANT
REVOKE
Transaction Control Language (TCL)
COMMIT
ROLLBACK
SAVEPOINT

Thank you