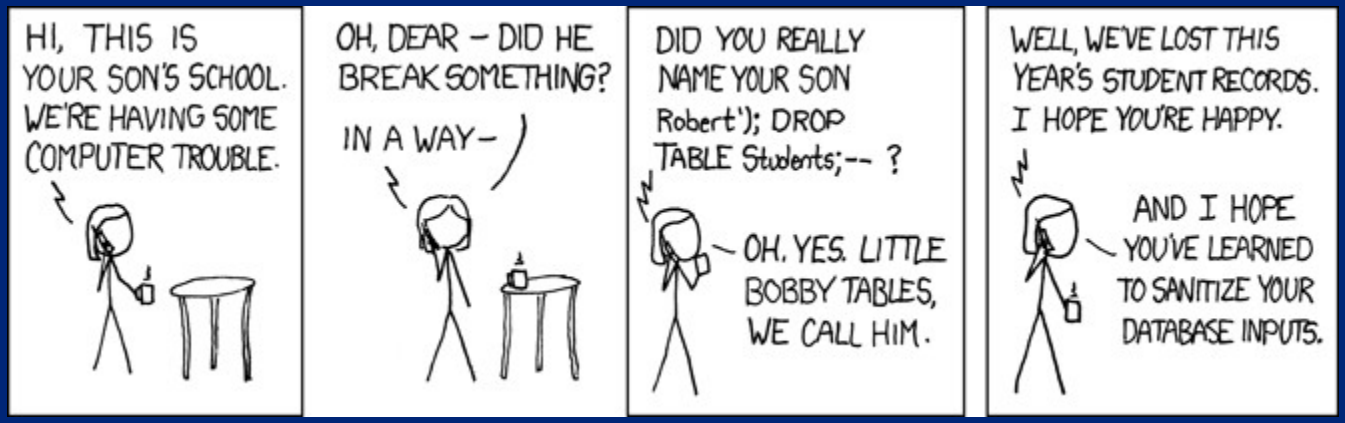
Università di Genova

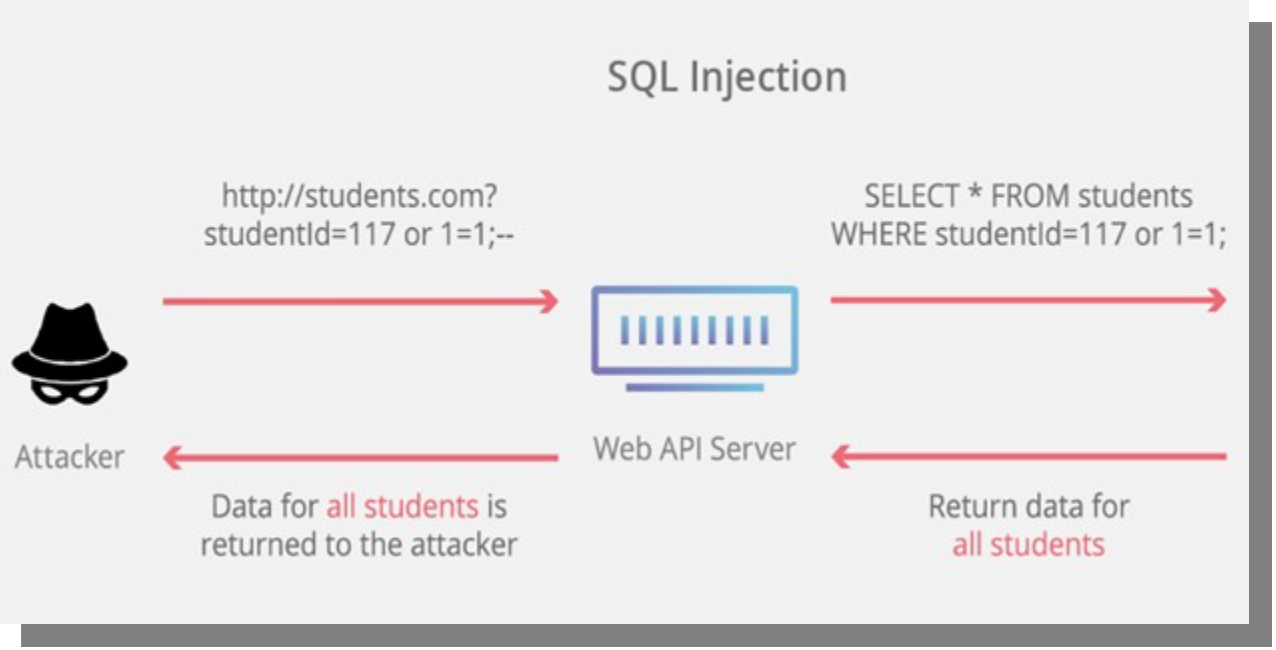
PHP (6)



Marina Ribaudo, marina.ribaudo@unige.it

SQL Injection

2



- An **injection of code** happens when an attacker sends **invalid** and **untrusted data** to the application as part of a command or query. The attacker has the malicious intention to trick the application into **executing unintended behavior to collect data or create damage**.
- https://owasp.org/www-project-top-ten/

SQL Injection

- Le query come quelle viste fino ad ora, per es.

```
$query = “SELECT * FROM user WHERE name='$name';
```

che combinano

- codice SQL
- input fornito dall’utente

sono **potenzialmente pericolose** se i dati forniti in input non sono opportunamente validati

Prepared Statement

*“In database management systems, a **prepared statement** or **parameterized statement** is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a **template** into which certain constant values are substituted during each execution.”*

Vedi: http://en.wikipedia.org/wiki/Prepared_statement

Prepared Statement

- Rispetto alle query SQL standard, i **prepared statement** (o query parametriche)
  - vengono **eseguiti più volte**, per valori diversi dei parametri: prima il DBMS prepara il **piano di esecuzione** della query, e poi si possono avere esecuzioni multiple, per **valori diversi dei parametri**
  - sono **"robusti" rispetto a SQL injection**, perché i valori dei parametri sono trasmessi in un secondo momento e non sono usati nella costruzione della query

Nota: *"Using a prepared statement is not always the most efficient way of executing a statement. **A prepared statement executed only once** causes more client-server round-trips than a non-prepared statement..."*

Prepared Statement

- MySQL supporta i **prepared statement**
- Sono necessari 2 passi
  - **Prepare:** si scrive un **template** con dei **parametri** (**placeholder**) che viene inviato al DBMS. Il DBMS esegue il controllo sintattico e inizializza le risorse per gli usi successivi
  - **Execute:** il **client invia i valori dei parametri al server** che esegue la query sfruttando le risorse precedentemente allocate

MySQLi e Prepared Statement

- **Prepare**
  - mysqli_prepare, mysqli::prepare

```
$stmt =
mysqli_prepare($con, "SELECT * FROM user WHERE email=?");
```

```
$stmt =
$con->prepare("SELECT * FROM user WHERE email=?");
```

Vedi: http://php.net/manual/it/mysqli.prepare.php

MySQLi e Prepared Statement

“Bound variables are sent to the server separately from the query and thus cannot interfere with it. The server uses these values directly at the point of execution, after the statement template is parsed. ***Bound parameters do not need to be escaped as they are never substituted into the query string directly****. A* ***hint*** *must be provided to the server* ***for the type*** *of bound variable, to create an appropriate conversion...”*

https://dev.mysql.com/doc/apis-php/en/apis-php-mysqli.quickstart.prepared-statements.html

MySQLi e Prepared Statement

- **Bind parameter**
  - mysqli_stmt_bind_param, mysqli_stmt::bind_param

mysqli_stmt_bind_param($stmt, 's', $email);
$email = “user@mail.it”;

**$stmt->bind_param**('s',$email);
$email = “user@mail.it”;

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double
s	corresponding variable has type string
b	corresponding variable is a blob and will be sent in packets

Vedi: http://www.php.net/manual/it/mysqli-stmt.bind-param.php

MySQLi e Prepared Statement

- **Execute**
  - mysqli_stmt_execute, mysqli_stmt::execute

```
mysqli_stmt_execute($stmt);
```

```
$stmt->execute();
```

MySQLi e Prepared Statement

- **Ottenere il risultato** nel caso di SELECT

```
$res=mysqli_stmt_get_result($stmt);
```

```
$res = $stmt->get_result();
$row = $res->fetch_assoc();
All records: $res->fetch_all();
```

Per le query di SELECT viene restituita una risorsa con i record (0,1,...) selezionati in caso di successo, FALSE in caso di fallimento

TRUE (successo) o FALSE (fallimento) per le altre query

https://www.php.net/manual/en/mysqli-stmt.get-result.php

MySQLi e Prepared Statement

- **Ottenere il risultato** nel caso di INSERT, UPDATE, DELETE

```
mysqli_execute($stmt);
if ($mysqli_affected_rows($stmt) === 0)
        echo “no rows inserted/updated/canceled”;
….
```

```
$stmt->execute();
if ($stmt->affected_rows === 0)
        echo “no rows inserted/updated/canceled”;
….
```

MySQLi e Prepared Statement

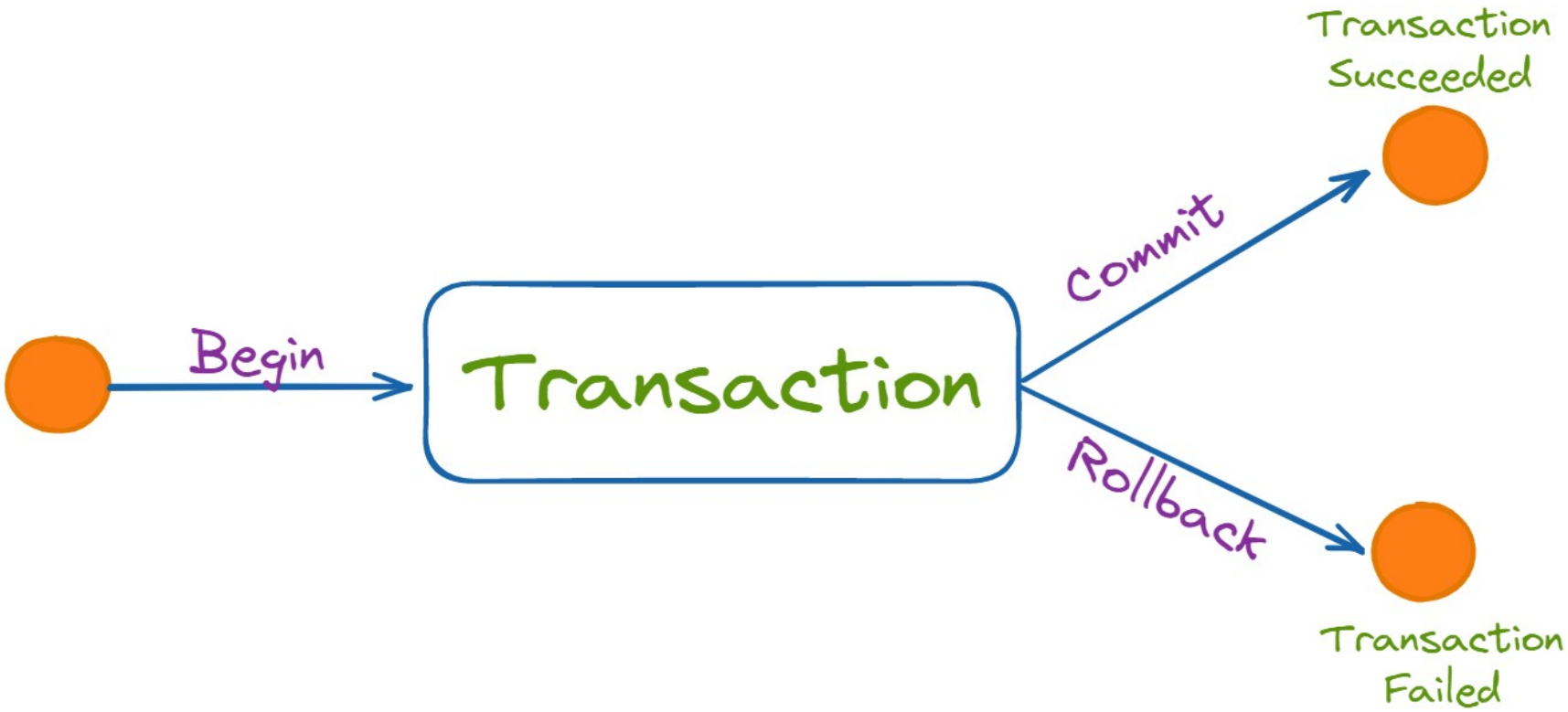
- Alla fine si deve/può chiudere lo statement

```
mysqli_stmt_close($stmt);
```

```
$stmt->close();
```

- E poi si deve/può chiudere la connessione, $con->close()
- Esiste anche $stmt->free()

Transazioni (cenni)



Transazioni (cenni)

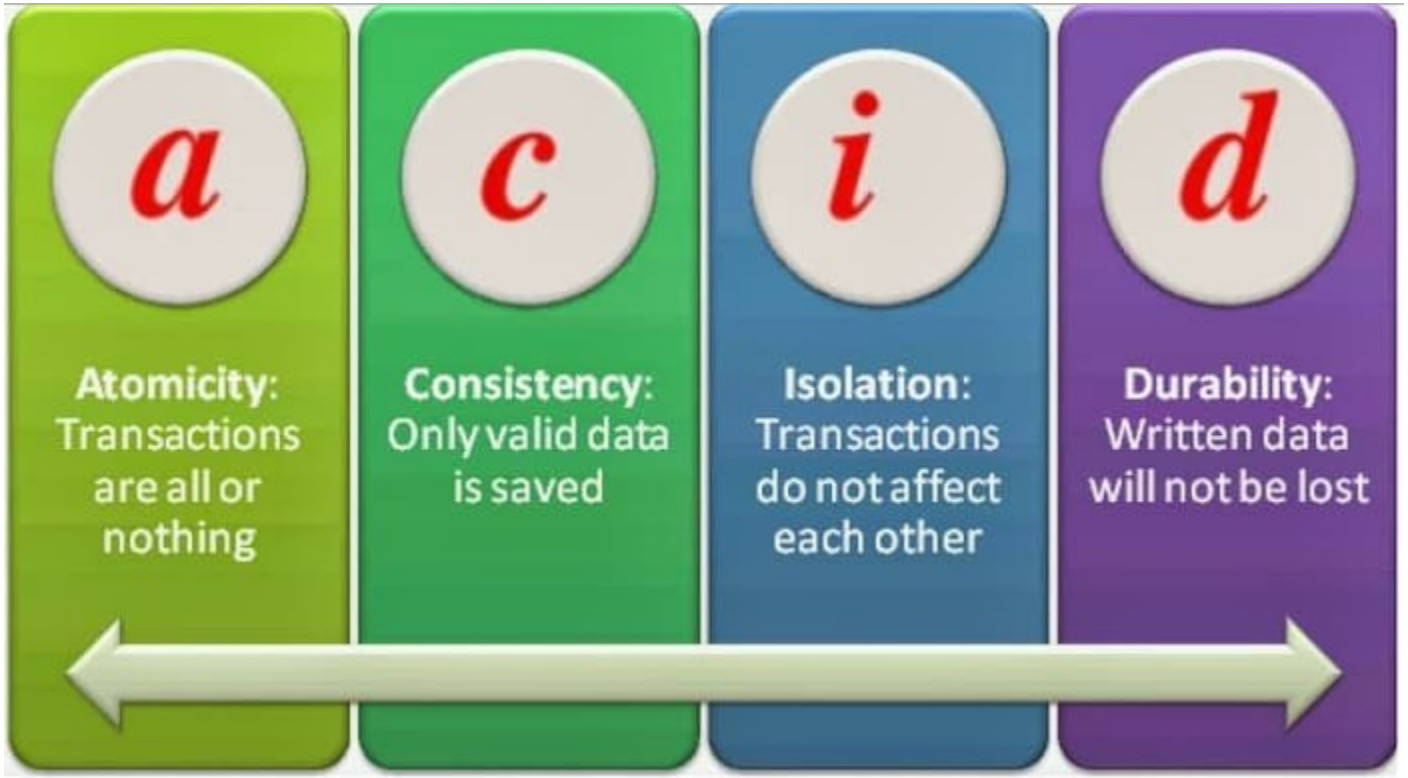
“A single transaction consists of **one or more independent units of work**, each reading and/or writing information to a database or other data store. When this happens it is often important to ensure that all such processing **leaves the database or data store in a consistent state**.”

Steps

- **Begin** the transaction
- **Execute** a set of data manipulations and/or queries
- If no errors occur then **commit** the transaction and end it
- If errors occur then **rollback** the transaction and end it

Vedi: http://en.wikipedia.org/wiki/Database_transaction

Proprietà ACID



https://en.wikipedia.org/wiki/ACID

MySQLi e Transazioni

- L'estensione MySQLi introduce nuove funzioni per sfruttare le caratteristiche transazionali di MySQL
- Queste funzioni sono equivalenti ai comandi SQL
  - **START TRANSACTION (**oppure **BEGIN)**
  - **COMMIT**
  - **ROLLBACK**
- **Nota:** per implementare le transazioni le tabelle del database devono essere di tipo **InnoDB**, BerkeleyDB, Falcon

MySQLi e Transazioni

- Guardate la documentazione online …

```
/* set autocommit to off */
$con->autocommit(FALSE);

/* execute some queries (can be also prepared statements) */
$con->query("INSERT ...");
$con->query("UPDATE ...");

/* trigger commit transactions */
$con->autocommit(TRUE);
```

Vedi: http://www.php.net/manual/en/mysqli.commit.php

MySQLi e Transazioni

- Se le cose non vanno a buon fine si può tornare indietro usando il rollback (undo)

```
/* rollback */
$con->rollback();
```

Vedi: http://php.net/manual/it/mysqli.rollback.php

Esercizio 24 Parte 2 su AulaWeb

- **Facoltativo**
  - Supponiamo di avere una tabella che memorizza le informazioni dei conti corrente di una banca
  - Se prendiamo dei soldi da un conto e li depositiamo su un altro conto dobbiamo assicurarci che entrambe le operazioni vadano a buon fine, altrimenti si perde del denaro
  - Ci vogliono due query di UPDATE che devono andare entrambe a buon fine, altrimenti bisogna riportare tutto nella situazione iniziale

Database Abstraction Layer

Good programmers write solid code, while great programmers reuse the code of good programmers

Database Abstraction Layer

- Si tratta di **librerie** che **offrono API** che permettono di **astrarre dal DBMS sottostante**

  “Libraries unify access to databases by providing a single low-level programming interface to the application developer. Their advantages are most often speed and flexibility because they are not tied to a specific query language (subset) and only have to implement a thin layer to reach their goal.”

Vedi: http://en.wikipedia.org/wiki/Database_abstraction_layer

PHP Data Objects

- Per la **connessione al database** si crea un'istanza della **classe PDO**
- Il costruttore accetta come primo parametro una **stringa nella quale si specifica il tipo di DBMS cui si vuole accedere**

```
<?php
   $conn =
   new PDO("mysql:dbname=$dbname;host=$host", $username, $password);
?>
```

```
<?php
   $conn =
   new PDO("pgsql:dbname=$dbname;host=$host", $username,  $password);
?>
```

Vedi: http://www.php.net/manual/en/class.pdo.php

PHP Data Objects

- Per chiudere la **connessione al database**

```
<?php
   $conn = null
?>
```

- Gli **errori** sono gestiti tramite **blocchi try/catch**

```
<?php
try {
        $conn = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
}
catch(PDOException $e) {
        error_log($e→getMessage());
        exit();
}
?>
```

PHP Data Objects

- Per tutte le altre operazioni si usano **gli stessi metodi indipendentemente dal DBMS**


```
<?php
$id = …
$name = ..
$stmt = $conn->prepare("SELECT * FROM user WHERE id=? AND name=?")
$stmt->bindParam(1, $id, PDO::PARAM_INT);
$stmt->bindParam(2, $name, PDO::PARAM_STR, 40);
$stmt->execute();
?>
```

Sintassi alternativa

```
$stmt = $conn->prepare("SELECT * FROM user WHERE id = :id AND name = :name");
```

PHP Data Objects

- Per tutte le altre operazioni si usano **gli stessi metodi indipendentemente dal DBMS**


```
<?php
  $stmt->setFetchMode(PDO::FETCH_NUM);
  while ($row = $stmt->fetch())  {
        print $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
 }
?>
```

- PDO::FETCH_ASSOC
  PDO::FETCH_BOTH
  PDO::FETCH_OBJ
  ecc.
- https://websitebeaver.com/php-pdo-prepared-statements-to-prevent-sql-injection

Domande ricorrenti

- Meglio usare mysqli_ nella versione procedurale oppure object oriented?
- Meglio usare le query SQL "standard" viste nella prima lezione su MySQL oppure i prepared statement?
- Nel caso dei form (registrazione, login, ecc) meglio fare tanti file separati oppure un solo file?
- **Attenzione!**
  - in rete ci sono parecchi esempi ma non sempre sono fatti bene :(
  - non fidatevi al 100% delle risposte di ChatGPT e simili