- injection - input, ki ga daš noter, nekaj pokvari, ker je programer mešal svojo logiko in vnešene podatke
- `' OR 1=1; --`
- ^to nas bo loginalo v prvega userja v bazi - kaj če bi nujno želeli admin:
  - `admin'; --`
  - `admin' #` - # dela samo v mariadb kot komentar
- rešitev:
  - string sanitization (prva najlažja rešitev, ampak ponavadi želimo prepared statement ali ORM)
  - prepared statement
  - ORM - sam pravilno pripravi SQL stavke, ki jih opišemo s funkcijami v kodi; včasih se nam zgodi, da bomo vseeno rabili na roke SQL stavke napisati in rabimo spet paziti na SQL injection

```php
$POST['username'] = $conn->real_escape_string($POST['username']);
$POST['password'] = $conn->real_escape_string($POST['password']);
```

Prepared statement:

- posebej pošljemo podatke in naš query, potem se na bazi izvede ločeno

```php
// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

- injection ni samo SQL, lahko je tudi XSS, injection v shell (OS insertion)
- `ping -c 4 1.1.1.1 && ls` - bo malo trajalo, da se ping konča, zato je bolje narediti, da bo ping failal
- lahko samo `; ls`
- `;echo 'a' > tmp.php`
- rešitev:
  - php escpaeshellarg
  - ločimo vnešene podatke (argumente) od našega ukaza - php `proc_open()`

- probaj čim več uporabljati knjižnice ali svojo kodo, ki nekaj požene, namesto da direktno nek ukaz v shellu poganjaš, ker je zelo verjetno, da se da z nekimi argumenti doseči nekaj, kar si ne bi želeli