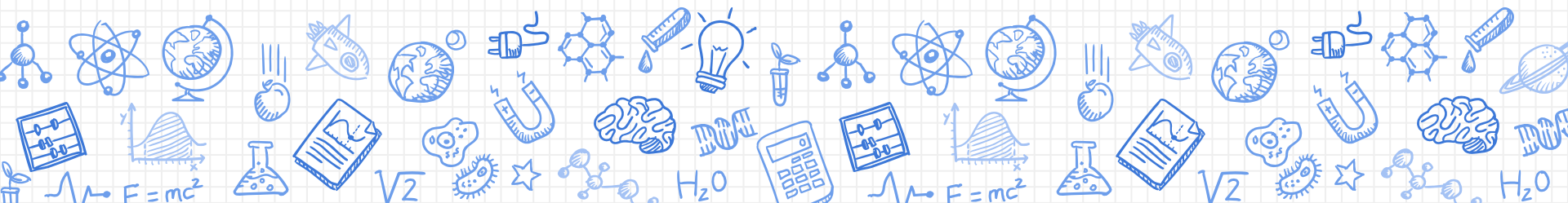
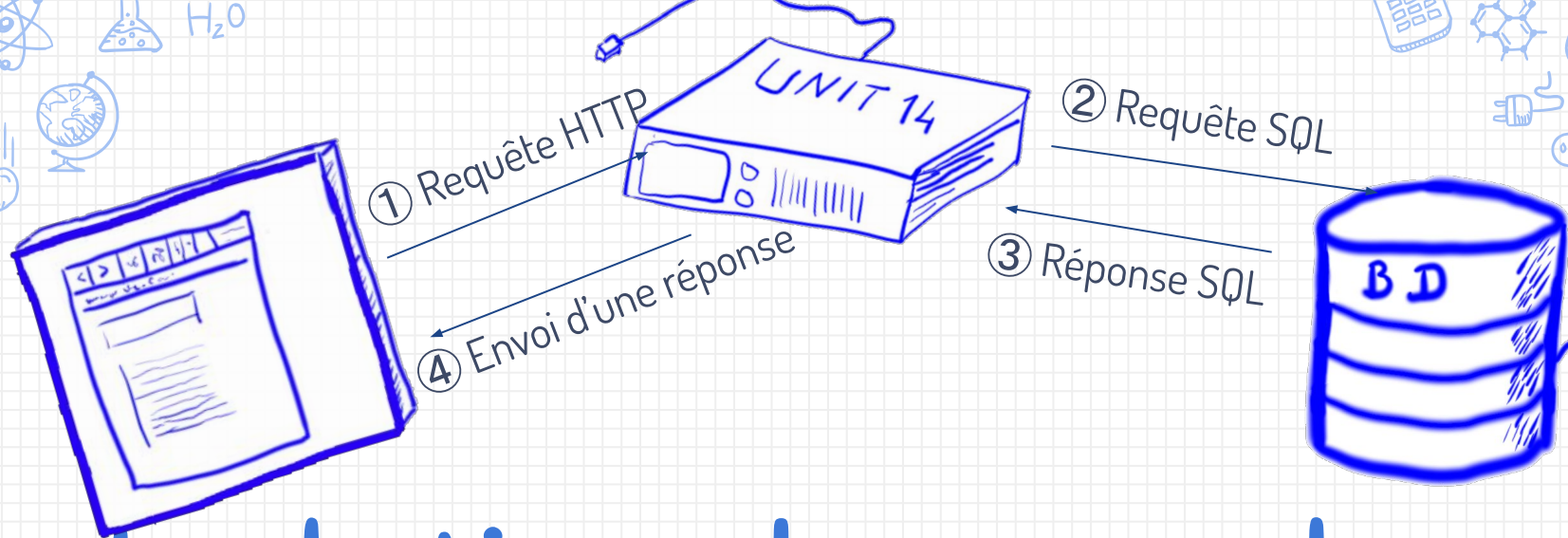


Comment se connecter à une base de données ?

HTML, CSS et PHP : pas assez de langages, ajoutons SQL





Isolation des couches

Et reconnexion à chaque requête 🤖

(enfin des fois on peut économiser, heureusement!)



Utilisation de PDO

- ✗ Utilisation d'une librairie multi-BD : PDO
- ✗ Indépendance (relative) de la BD
- ✗  Dialecte SQL peut changer lorsqu'on écrit les requêtes
- ✗ Gestion uniforme de la connexion (DSN)
- ✗ Code d'accès aux données uniforme
- ✗ **Requêtes préparées**
- ✗ Erreurs uniformisées (try ... catch ...)
- ✗ Introspection (getColumnType, getColumnName)



Comment faire une requête ?

Connexion

```
$bd = new PDO(
    'pgsql:host=localhost;\n
    dbname=bdCours', // key, values
    // dépend du type de BD
    // souvent serveur + dbname
    'jcdubacq', // utilisateur
    'M0tD3pass3'); // mot de passe
```

Requête

```
$requete = $bd->prepare('SELECT
Nom, Prenom FROM Personnes');
$requete->execute();
```

Récupération

```
$resultat =
    $bd->fetch(PDO::FETCH_ASSOC);
echo '<p>Nom: '
    .htmlentities($res['Nom'])
    . ' prénom: '
    .htmlentities($res['Prenom']);
```

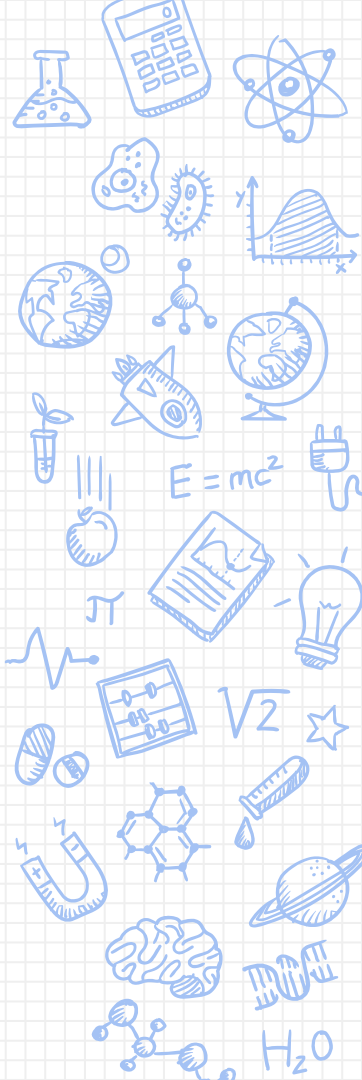
Les modes de récupération

- ✗ fetch récupère une ligne. Le résultat est faux si fin des résultats (utile pour un while)
- ✗ fetchAll récupère un tableau de lignes toutes formées pareil
- ✗ Argument sert à indexer la ligne
PDO::FETCH_ASSOC : nom de colonne en index,
PDO::FETCH_NUM : numéro de colonne,
PDO::FETCH_BOTH : les deux (⚠ tableau hybride)
PDO::FETCH_OBJ crée un objet (\$res->Nom)



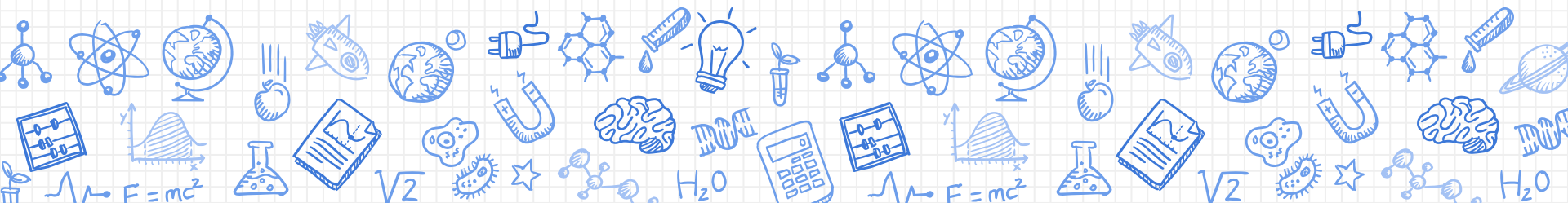
Les requêtes préparées

```
$type = 'feu';
$req = $bd->prepare('SELECT nom, type_primaire, forme,
FROM pokemon WHERE attaque > :att AND type= :type');
$req->bindValue(':att', 150);
$req->bindValue(':type', $type);
$req->execute();
# Exécute SELECT nom, type_primaire, forme,
# FROM pokemon WHERE attaque > 150 AND type= 'feu'
# notez bien les apostrophes pour la chaîne
echo "<ul>";
while ($res=$req->fetch(PDO::FETCH_ASSOC)) {
    echo "<li>".htmlentities($res['nom'])."</li>";
}
echo "</ul>";
```



Éléments de sécurité

Un esprit sain dans un code sain





Considérer toute donnée étrangère comme hostile

Formulaires

Toujours vérifier ce qui vient des formulaires, même si le navigateur (via HTML ou Javascript) le contrôle.

Les voleurs ne se limitent pas à passer par la porte d'entrée

Affichage (faille XSS)

Toute chaîne qui n'a pas été filtrée (par ex. par preg_match) doit être affichée telle quelle : utilisation de htmlentities (ou htmlspecialchars).

Emballer les données sales en HTML, c'est htmlentities

Stockage (faille SQLi)

Toutes les données non filtrées doivent être entrées en les protégeant contre l'interprétation SQL : utilisation des requêtes préparées.

Emballer les données sales en SQL, c'est prepare/bindValue

L'avez-vous vraiment
appelé **Robert**); DROP
TABLE eleves; -- ?

Oh mince — Est-ce qu'il a
cassé quelque chose ?

D'une certaine façon...

Oui, c'est ça. On le surnomme
le petit Bob Tables.

Eh bien nous avons
perdu toutes nos
données élèves cette
année. J'espère que vous
êtes contente.

Et moi j'espère
que vous avez
appris à nettoyer
vos entrées de BD.

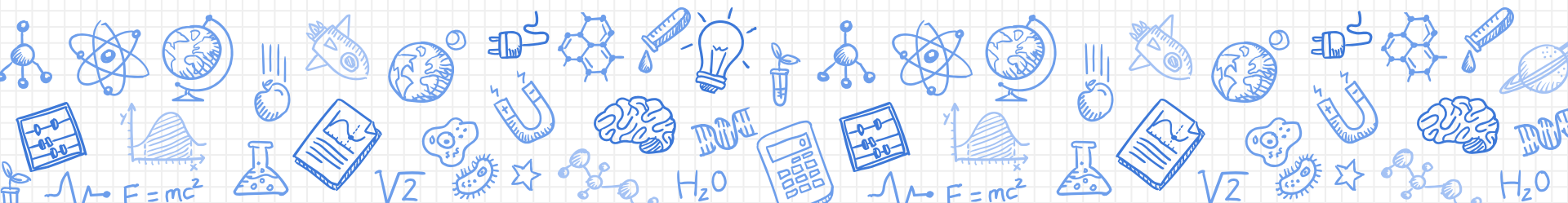


```
INSERT INTO eleves (nom,prenom) VALUES('Smith','Robert'); DROP  
TABLE eleves; -- '');
```

<https://xkcd.com/327/>

Regrouper les données

Abstraction, implémentation



Design pattern singleton : one model to rule them all

```
private $bd;
```

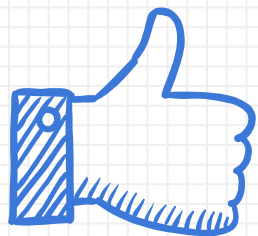
Class Model

```
private static $instance  
= null;
```

```
private function __construct() {  
    initialisation $bd;  
}
```

```
public function getModel() {  
    if self::$instance==null {  
        self::$instance = new self();  
    }  
    return self::$instance;  
}
```





C'est tout!

Des questions?

- ✗ Dans le Discord
- ✗ Par mail