

Livrable 4

# Rapport de BD

---

VACQUIER Hugo, BIRET-TOSCANO Esteban, FERNANDEZ Mickael  
21 mars 2023



## Sommaire :

I - Nouveaux scripts PHP :	2
II - Nouveaux tests :	5
III - Architecture de l'application future :	10

## I - Nouveaux scripts PHP :

Après avoir normalisé notre base de données, nous nous devons de reconcevoir notre procédure stockée ainsi que notre vue qui s'adapteront à cette dernière. De ce fait, les scripts PHP devenaient obsolètes car en effet, ces derniers s'exécutaient sur notre base de données primaire : nous nous devons de les réajuster.

Pour chaque nouveau script, un nouveau fichier PHP était dédié à celui-ci (nous avons décidé de garder l'ancienne base de données, associée à ses anciens fichiers PHP et anciens tests) :

AjouterMatchV2.php :

Avec une nouvelle table "MATCHV2", celle-ci possède moins d'attributs. En effet, pour éviter les redondances, les données ont été réparties dans d'autres tables afin d'avoir une normalisation la plus complète possible (d'où le fait qu'auparavant, 11 attributs étaient nécessaires dans la procédure ; maintenant, nous n'en avons que 5) :

```
1  <?php
2
3  function AjouterMatchV2() {
4      require_once("connect.inc.php");
5      error_reporting(0);
6
7      $req = 'begin AjouterMatchV2(:p_jrn, :p_stade, :p_na, :p_ne1, :p_ne2); end; ';
8      $appelProcStock = oci_parse($connect, $req);
9
10     $jrn = 1;
11     $stade = 'stade arletty' ;
12     $na = 1;
13     $ne1 = 'FRA';
14     $ne2 = 'BLA';
15
16     oci_bind_by_name($appelProcStock, ':p_jrn', $jrn);
17     oci_bind_by_name($appelProcStock, ':p_stade', $stade);
18     oci_bind_by_name($appelProcStock, ':p_na', $na);
19     oci_bind_by_name($appelProcStock, ':p_ne1', $ne1);
20     oci_bind_by_name($appelProcStock, ':p_ne2', $ne2);
21
22     $result = oci_execute($appelProcStock);
23
24     if (!$result) {
25         $e = oci_error($appelProcStock);
26         return (htmlentities($e['message'].' pour la fonction : '.$e['sqltext']));
27     } else {
28         return "Match ajouté";
29     }
30
31     oci_commit($connect);
32     oci_free_statement($appelProcStock);
33     oci_close($connect);
34 }
35
36 $res = AjouterMatchV2();
37 echo $res;
38
39 ?>
```

AjoutVoiComposerV2.php :

Ce fichier interagit directement auprès des tables mais nécessite que certaines valeurs soient, au préalable, déjà présentes au sein de la base de données. En effet, pour pouvoir insérer une composition, il est nécessaire d'avoir un match ainsi qu'un joueur. Néanmoins, après avoir normalisé notre base de données, les insertions ont changé. Désormais, il est nécessaire de réaliser des insertions avec les tables "SELECTIONNEURV2", "EQUIPEV2", "JOUEURV2", "ARBITREV2", "STADEV2" et "MATCHV2" comme ci-dessous :

```
INSERT INTO SELECTIONNEURV2 (noms, pres, dtns, nats)
VALUES ('Esteban', 'Pablo', '30/05/1975', 'Colombie');
```

```
INSERT INTO SELECTIONNEURV2 (noms, pres, dtns, nats)
VALUES ('Paolo', 'Diego', '31/08/1961', 'Bresil');
```

```
INSERT INTO EQUIPEV2 (ne, nome, noms, pres)
VALUES ('COL', 'Colombie', 'Esteban', 'Pablo');
```

```
INSERT INTO EQUIPEV2 (ne, nome, noms, pres)
VALUES ('BRE', 'Bresil', 'Paolo', 'Diego');
```

```
INSERT INTO JOUEURV2 (nj, nomj, prej, pst, ne)
VALUES (300, 'Toto', 'Pierro', 'Pilier', 'COL');
```

```
INSERT INTO JOUEURV2 (nj, nomj, prej, pst, ne)
VALUES (301, 'Titi', 'Eris', 'Pilier', 'COL');
```

```
INSERT INTO JOUEURV2 (nj, nomj, prej, pst, ne)
VALUES (302, 'Victory', 'Marco', 'Pilier', 'BRE');
```

```
INSERT INTO JOUEURV2 (nj, nomj, prej, pst, ne)
VALUES (303, 'Teste', 'Laurent', 'Pilier', 'BRE');
```

```
INSERT INTO ARBITREV2 (na, noma, prea, nata)
VALUES (1, 'Peyper', 'Jaco', 'Afrique du Sud');
```

```
INSERT INTO STADEV2 (stade, ville)
VALUES ('Aviva Stadium', 'Dublin');
```

```
INSERT INTO MATCHV2 (nm, jrn, stade, na, ne1, ne2)
VALUES (20, 1, 'Aviva Stadium', 1, 'COL', 'BRE');
```

Vient enfin, la possibilité de réaliser l'insertion automatique des données, comprise dans le formulaire. De même, en comparaison à l'ancien fichier, une fonction destinée à compter les valeurs ajoutées dans le tableau est réalisée. En voici le nouveau fichier :

```

25 <?php
26
27 require_once("connect.inc.php");
28
29 echo "<p>Veuillez entrer les informations de la nouvelle composition </p><br/>";
30 echo "<form method='post' enctype='multipart/form-data'>";
31 echo "<p>";
32     echo "nm : <input type='text' value='20' name='nm' /> <br><br>";
33     echo "nj : <input type='text' value='300' name='nj' /> <br><br>";
34     echo "maillot : <input type='text' value='2' name='maillot' /> <br><br>";
35     echo "<input type='submit' name='Envoyer' value='Valider' />";
36 echo "</p>";
37 echo "</form>";
38
39 if (isset($_POST['Envoyer']) && isset($_POST['nj']) && isset($_POST['nm']) && isset($_POST['maillot'])) {
40
41     $req = 'INSERT INTO COMPOSERV2 VALUES (:pNm, :pNj, :pMaillot)';
42     $insertComp = oci_parse($connect, $req);
43
44     oci_bind_by_name($insertComp, ":pNm", $_POST['nm']);
45     oci_bind_by_name($insertComp, ":pNj", $_POST['nj']);
46     oci_bind_by_name($insertComp, ":pMaillot", $_POST['maillot']);
47
48     $result = oci_execute($insertComp);
49     if (!$result) {
50         $e = oci_error($insertComp);
51         print htmlentities($e['message']).' pour cette requete : '.$e['sqltext'];
52     }
53 }
54
55 $req1 = "SELECT * FROM COMPOSERV2 WHERE nj >= 300";
56 $lesComp = oci_parse($connect, $req1);
57 $result = oci_execute($lesComp);
58 if (!$result) {
59     $e = oci_error($lesComp);
60     print htmlentities($e['message']).' pour cette requete : '.$e['sqltext'];
61 }
62
63 echo "<H1> Les Compositions pour les joueurs de numéro sup. ou égal à 300</H1>";
64 echo "<table border='2'>";
65 echo "<th>Num Match</th><th>Num Joueur</th><th>Num Maillot</th>";
66 while (($leComp = oci_fetch_assoc($lesComp)) != false) {
67     echo "<tr>";
68     echo "<td>".$leComp['NM']. "</td>";
69     echo "<td>".$leComp['NJ']. "</td>";
70     echo "<td>".$leComp['MAILLOT']. "</td></tr>";
71 }
72 echo "</table>";

```

```

74 function getNbValeursAjoutees() {
75
76     global $connect;
77     $nbValeursAjoutees = 0;
78     $req2 = "SELECT * FROM COMPOSERV2 WHERE nj >= 300";
79
80     $lesComp = oci_parse($connect, $req2);
81     $result = oci_execute($lesComp);
82     if (!$result) {
83         $e = oci_error($lesComp);
84         return null;
85     }
86     while (($leComp = oci_fetch_assoc($lesComp)) != false) {
87         $nbValeursAjoutees += 3;
88     }
89     return $nbValeursAjoutees;
90 }
91
92 oci_commit($connect);
93
94 $res = getNbValeursAjoutees();
95 echo ("Nombre de valeurs ajoutées : ").$res;
96
97 oci_free_statement($lesComp);
98 oci_close($connect);
99
100
101 >>

```

VoirCartonsParArbitreV2.php :

Par rapport à l'ancien script, rien ne change à l'exception que ce fichier prendra en considération la nouvelle vue et une autre fonction, cette fois-ci calculant les cartons rouges (pour les tests) qui sera réalisée :

```
1 <?php
2 require_once("connect.inc.php");
3 error_reporting(0);
4 $req2 = "SELECT * FROM VCARTONS_PAR_ARBITREV2";
5
6 $lesCartons = oci_parse($connect, $req2);
7 $result = oci_execute($lesCartons);
8 if (!$result) {
9     $e = oci_error($lesCartons);
10    return (htmlentities($e['message'], ' pour cette requete : '. $e['sqltext']));
11 }
12
13 echo "<H1> Les Cartons distribués par arbitre</H1>";
14 $tableCarton = "<table border='2'>";
15 $tableCarton .= "<tr><th>Prenom Arbitre</th><th>Nom Arbitre</th><th>Type Carton</th><th>Num match</th><th>Minute</th><th>Prenom Joueur</th><th>Nom Joueur</th></tr>";
16
17 while (($leCarton = oci_fetch_assoc($lesCartons)) != false) {
18     $tableCarton .= "<tr>";
19     $tableCarton .= "<td>". $leCarton['PRENOMARBITRE']. "</TD><td>". $leCarton['NOMARBITRE']. "</TD><td>". $leCarton['CARTON']. "</TD><td>". $leCarton['NUMMATCH']. "</TD>".
20     "<td>". $leCarton['TEMPS']. "</TD><td>". $leCarton['PRENOMJOUEUR']. "</TD><td>". $leCarton['NOMJOUEUR']. "</TD></tr>";
21 }
22 $tableCarton .= "</TABLE>";
23 echo($tableCarton);
24
25 function getNbCartonsRouges() {
26
27     global $connect;
28     $nbCartonsRouges = 0;
29     $req3 = "SELECT * FROM VCARTONS_PAR_ARBITREV2";
30     $allCartons = oci_parse($connect, $req3);
31     $result = oci_execute($allCartons);
32     if (!$result) {
33         $e = oci_error($allCartons);
34         return null;
35     }
36     while (($scarton = oci_fetch_assoc($allCartons)) != false) {
37         if ($scarton['CARTON'] == "Carton Rouge") {
38             $nbCartonsRouges++;
39         }
40     }
41     return $nbCartonsRouges;
42 }
43
44 $res = getNbCartonsRouges();
45 echo ("Nombre de cartons rouges : ".$res);
46
47 oci_free_statement($lesCartons);
48 ?>
```

## II - Nouveaux tests :

Avant même d'en réaliser les nouveaux tests, nous nous devons de comprendre pourquoi le test primaire (celui déjà donné par nos enseignants) ne fonctionnait pas. Et malgré de multiples tentatives pour essayer de faire fonctionner ce dernier, le test restait toujours invalide (le résultat attendu ne coïncidait pas à celui retourné par la/les fonction(s)).

```
public function testAjouterMatch() {
    // GIVEN - WHEN - THEN
    $aMatch = AjouterMatch($connect);
    $this->assertSame("Match ajouté", $res);
}
```

- Premièrement, nous nous sommes aperçus que le nombre de paramètres de la procédure SQL différait du nombre de paramètres lors de l'appel de cette procédure en PHP. En effet, les attributs scr1 et scr2 provenaient directement de la procédure SQL et étaient complètement ignorés pour les variables \$scr1 et \$scr2 (ne servant à rien). Pour cela, nous avons privilégié l'utilisation des paramètres nommés. Néanmoins, l'erreur du test persistait.
- Deuxièmement, nous avons compris que lors de l'exécution de la fonction, il était logique que le résultat différait lors d'une deuxième exécution de cette dernière car les données étant insérées, elles ne peuvent l'être une deuxième fois (les données restent similaires à l'exécution de la fonction). Toutefois, le test récupérait constamment la chaîne de caractère associée à une erreur de la procédure stockée alors même que le match était inséré dans notre base de données.

```
// $res = AjouterMatch();
// echo $res;
```

```
1) StackTest::testAjouterMatch
Failed asserting that two strings are identical.
--- Expected
+++ Actual
@@ @@
-'Match ajouté'
+' pour la fonction : '
```

- Troisièmement, en comparant à l'exécution des tests réalisés dans le module *Qualité de Développement*, nous nous sommes aperçus qu'avant même d'utiliser une méthode, il est nécessaire de la déclarer dans une classe pour pouvoir y avoir accès. Ainsi, la nécessité de programmer un constructeur fut évidente :

```

class AjouterMatch {

    private $connect;
    private int $jrn;
    private int $na;
    private int $scr1;
    private int $scr2;
    private String $stade;
    private String $ville;
    private String $prea;
    private String $noma;
    private String $nata;
    private String $ne1;
    private String $ne2;

    public function __construct($connect) {

        $this->jrn = 1;
        $this->stade = 'stade arletty' ;
        $this->ville = 'Lyon';
        $this->na = 1;
        $this->prea = 'Nic';
        $this->noma = 'Berry';
        $this->nata = 'Ecosse';
        $this->ne1 = 'FRA';
        $this->ne2 = 'BLA';
        $this->scr1 = 12;
        $this->scr2 = 7;
        $this->connect = $connect;

    }
}

```

```

public function AMatch() {

    $req = 'begin AjouterMatch(:p_jrn, :p_stade, :p_ville, :p_na, :p_prea, :p_noma, :p_nata, :p_ne1, :p_ne2, :p_scr1, :p_scr2); end; ';
    $appelProcStock = oci_parse($connect, $req);

    oci_bind_by_name($appelProcStock, ':p_jrn', $this->jrn);
    oci_bind_by_name($appelProcStock, ':p_stade', $this->stade);
    oci_bind_by_name($appelProcStock, ':p_ville', $this->ville);
    oci_bind_by_name($appelProcStock, ':p_na', $this->na);
    oci_bind_by_name($appelProcStock, ':p_prea', $this->prea);
    oci_bind_by_name($appelProcStock, ':p_noma', $this->noma);
    oci_bind_by_name($appelProcStock, ':p_nata', $this->nata);
    oci_bind_by_name($appelProcStock, ':p_ne1', $this->ne1);
    oci_bind_by_name($appelProcStock, ':p_ne2', $this->ne2);
    oci_bind_by_name($appelProcStock, ':p_scr1', $this->scr1);
    oci_bind_by_name($appelProcStock, ':p_scr2', $this->scr2);

    $result = oci_execute($appelProcStock);

    if (!$result) {
        $e = oci_error($appelProcStock);
        return (htmlentities($e['message'].' pour la fonction : '.$e['sqltext']));
    } else {
        return "Match ajouté";
    }
    oci_commit($connect);
    oci_free_statement($appelProcStock);
    oci_close($connect);
}

// $res = AMatch();
// echo $res;

```



Ainsi, l'exécution du test différerait car nous devions faire appel à la classe, avant de pouvoir accéder à la méthode. Dans ce dernier, nous demandons à l'utilisateur d'intégrer le fichier de connexion à la base de données, qui sera passé en paramètre du constructeur pour qu'une cohérence entre notre test et la base de données se réalise mais en vain :

```
/**
 * @test
 */
//attaque la premiere procédure stockée
public function testAjouterMatch() {
    // GIVEN - WHEN - THEN
    $connect = require_once("connect.inc.php");
    $aMatch = new AjouterMatch($connect);

    $res = $aMatch->AMatch();
    $this->assertSame("Match ajouté", $res);
}
```

Car oui, nous nous sommes également interrogés sur la question du nom de la fonction étant similaire au nom du fichier (puis de la classe). Même après avoir modifié le nom de cette dernière, le résultat du test restait tel quel.

- Dernièrement, nous avons pensé à utiliser le gestionnaire de paquets “composer” pour pouvoir réaliser les tests. Au début, nous nous contentions du dossier “vendor” transmis par notre professeur mais après réflexion, nous nous sommes rendus compte qu’aucun package, ni aucun paramétrage autour de ce dernier n’était réalisé. C’est alors que nous avons commencé à paramétrer l’environnement par le biais des commandes suivantes :

```
composer unit
composer update
```

Ainsi, pour chaque fichier PHP ainsi que notre fichier de tests, nous avons inclus le package “SAEBD”, ainsi paramétré lors de l’exécution de la première commande composer :

```
namespace SAEBD;
```

```
use SAEBD\AjouterMatch;
```

Malgré cela, impossible d'exécuter les tests. C'est alors que nous avons finalement décidé de reprendre la même structure déjà imposée par nos enseignants pour l'exécution de l'ensemble des tests PHPUnit :

```
/**
 * @test
 */
//attaque la premiere procédure stockée
public function testAjouterMatch() {
    // GIVEN - WHEN - THEN
    $aMatch = AjouterMatch($connect);
    $this->assertSame("Match ajouté", $res);
}

/**
 * @test
 */
//attaque la seconde procédure stockée
public function ajouterMatchV2() {
    // GIVEN - WHEN - THEN
    $res = AjouterMatchV2();
    $this->assertSame('Match ajouté', $res);
}

/**
 * @test
 */
//attaque la première vue
public function testNbCartonsJaunes() {
    // GIVEN - WHEN - THEN
    $res = getNbCartonsJaunes();
    $this->assertEquals(4, $res);
}

/**
 * @test
 */
//attaque la seconde vue
public function testNbCartonsRougesV2() {
    // GIVEN - WHEN - THEN
    $res = getNbCartonsRouges();
    $this->assertEquals(2, $res);
}
```

```

/**
 * @test
 */
//attaque directement la table
public function VoirLignesComposer() {
    // GIVEN - WHEN - THEN
    $res = getNbLignes();
    $this->assertEquals(1, $res);
}


/**
 * @test
 */
//attaque directement la/les nouvelle(s) table(s)
public function VoirValeursAjouteesV2() {
    // GIVEN - WHEN - THEN
    $res = getNbValeursAjoutees();
    $this->assertEquals(3, $res);
}

```

### III - Architecture de l'application future :

Notre application possède désormais des fonctions PHP plus optimisées et adaptées :

- Une nouvelle fonction "AjouterMatchV2()" qui, comme son nom l'indique, permettra d'ajouter un match de rugby dans la nouvelle BD. Cette fonction prend uniquement en charge les détails de base tels que le numéro de journée, le stade et les noms d'équipe abrégés avec leurs nationalités et stocke dans des variables locales, les autres informations.
- Le fichier ajoutVoirComposerV2 qui possédait auparavant un compteur qui s'incrémentait, retourne l'ensemble des lignes sélectionnées grâce à une boucle while. Le nouveau fichier utilise simplement "getNbLignes()" pour récupérer le nombre de lignes et l'afficher dans une phrase sur la page web. Cette fonction est plus simple et permet d'afficher le nombre de lignes sur la page web sans effectuer d'autres traitements.
- La fonction "getNbCartonsRouges()", qui calcule le nombre de cartons rouges distribués en parcourant tous les enregistrements de la vue "VCARTONS\_PAR\_ARBITRE" et en comptant ceux avec un type de carton égal à "Carton Rouge". On possède aussi la même fonction "getNbCartonJaunes" qui agit de la même manière mais pour les cartons jaunes.



Une classe StackTest :

- Cette classe définit une série de tests unitaires pour vérifier le bon fonctionnement de différentes fonctionnalités liées à l'ajout de matchs et à la visualisation de statistiques de cartons et de lignes pour un arbitre. Elle hérite de la classe TestCase de PHPUnit et contient plusieurs méthodes de test annotées avec @test. Chaque méthode de test utilise les assertions de PHPUnit pour vérifier le résultat retourné par les différentes fonctions testées. Les méthodes de test testent les procédures stockées et les vues, ainsi que les tables liées aux nouvelles fonctionnalités.

Une procédure stockée :

- La procédure insère les données dans la table "MatchV2" à l'aide de l'instruction "INSERT INTO" et utilise la clause "VALUES" pour insérer les données. Contrairement à la procédure initiale elle n'insère que les valeurs (p\_jrn, p\_stade, p\_na, p\_ne1 et p\_ne2) car les autres arguments sont stockés dans la fonction ajouterMatchV2 dans des variables locales.

Une méthode d'encapsulation des données :

- On utilise une méthode d'encapsulation de données entre la base de données et sa procédure stockées "ajouterMatchV2" et la fonction PHP "ajouterMatchV2()". La fonction va appeler la procédure stockée en lui passant les paramètres nécessaires, puis la procédure stockée va insérer dans la base de données les arguments qu'elle a en paramètre et renvoyer le résultat à la fonction PHP. Cette encapsulation va permettre d'éviter les erreurs d'insertion en SQL et mieux contrôler l'accès à la base de données avec une meilleure protection des données.