

??????????

???????????

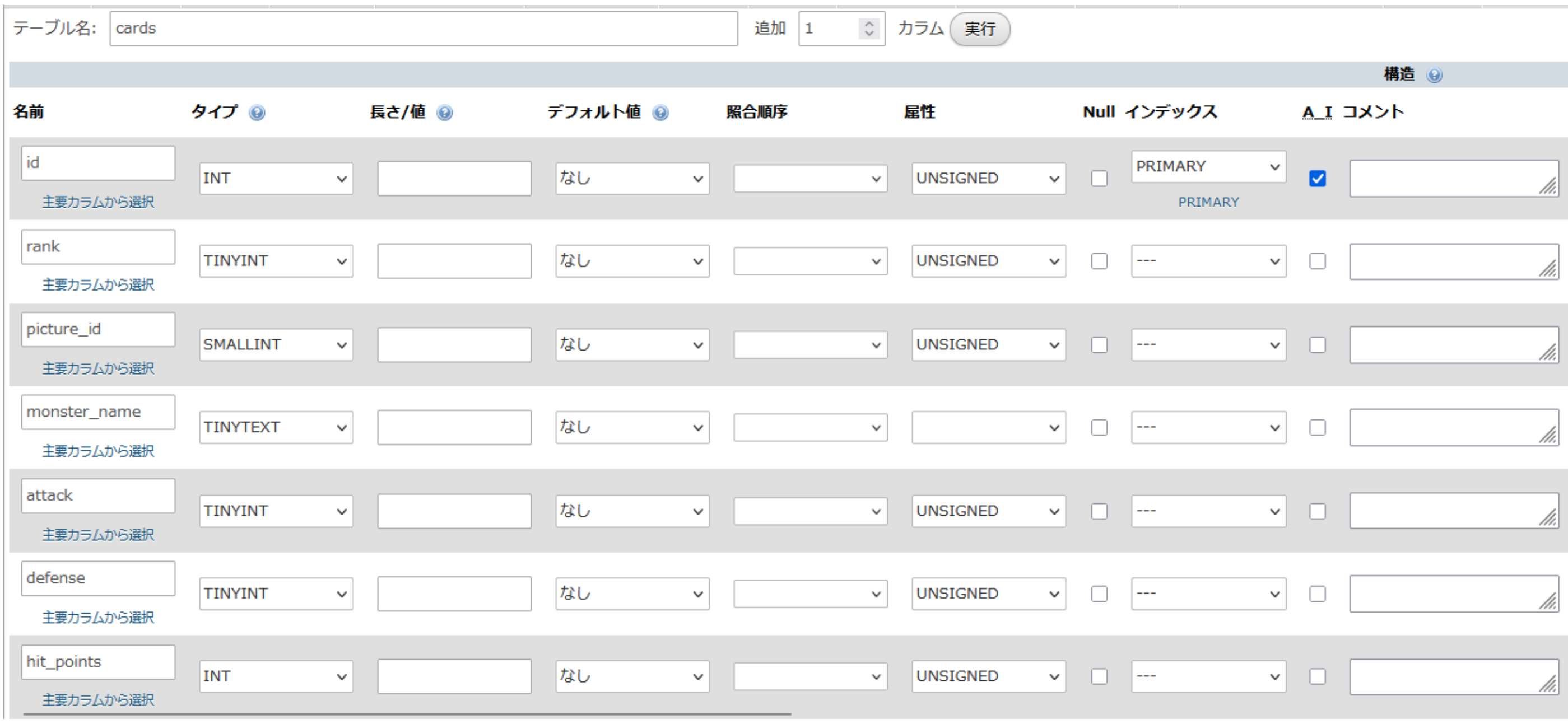
Unityで「　CardTest」シーンを開く。以下のような画面が表示される。

このシーンを使って、カード表示機能をテストできる。使用する値は次の通りである。

- ランク: 0～ 4の値で、それぞれ C、B、A、S、SSに対応する
- イラスト番号: 中央のイラストを選択するための 0～ 9の値
- モンスター名: 表示するモンスターの名前
- 攻撃力、防衛力、体力: 0～ 99の攻撃力、防衛力、体力の値

これらはすべて `CardDisplay` クラスによって処理される。特に、以下に示す `SetData` メソッドで扱われる。

```
// データを設定し、カードを更新
public void SetData(JObject data)
{
    // スプライトを読み込む
```



```
        cardRank.sprite = CardResources.GetRankSprite((int)data["rank"]);
        monsterPicture.sprite = CardResources.GetMonsterSprite((int)data["picture_id"]);

        // テキストを更新
        monsterName.text = (string)data["monster_name"];
        atkValue.text = (string)data["atk_value"];
        defValue.text = (string)data["def_value"];
        hpValue.text = (string)data["hp_value"];
    }
```

??????????

???????

この知識を使って、ゲーム内の全カードを保存するための新しいテーブルをデータベースに作成しよう。

phpMyAdmin をもう一度開き、以下のように必要なカラムを持つ「 cards 」という名前のテーブルを作成しよう。

そして、データベースにいくつかカードを追加しよう。

←T→	id	rank	picture_id	monster_name	attack	defense	hit_points
編集 コピー 削除	1	0	0	弱虫スケルトン	3	1	10
編集 コピー 削除	2	0	1	黒オニ	4	1	8
編集 コピー 削除	3	0	0	ホネボネ	3	2	15
編集 コピー 削除	4	0	1	悪魔赤ちゃん	5	2	10
編集 コピー 削除	5	1	4	ファイヤー猫	8	2	12
編集 コピー 削除	6	1	3	スカルゴースト	6	5	10
編集 コピー 削除	7	1	8	アルミ戦士	4	8	10
編集 コピー 削除	8	2	5	カエルる	10	5	20
編集 コピー 削除	9	2	6	メデゥーサ	8	8	15
編集 コピー 削除	10	3	7	木マン	5	15	22
編集 コピー 削除	11	3	8	闇ナイト	12	15	12
編集 コピー 削除	12	4	9	ギャラクシー	20	12	35

□□□　SQL□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
INSERT INTO `cards` (`id`, `rank`, `picture_id`, `monster_name`, `attack`, `defense`,
`hit_points`) VALUES
(1, 0, 0, '□□□□□□□□', 3, 1, 10),
(2, 0, 1, '□□□', 4, 1, 8),
(3, 0, 0, '□□□□', 3, 2, 15),
(4, 0, 1, '□□□□□□□', 5, 2, 10),
(5, 1, 4, '□□□□□□□', 8, 2, 12),
(6, 1, 3, '□□□□□□□□', 6, 5, 10),
(7, 1, 8, '□□□□□□', 4, 8, 10),
(8, 2, 5, '□□□□', 10, 5, 20),
(9, 2, 6, '□□□□□□', 8, 8, 15),
(10, 3, 7, '□□□', 5, 15, 22),
(11, 3, 8, '□□□□', 12, 15, 12),
(12, 4, 9, '□□□□□□□', 20, 12, 35);
```

??????????????

□□□□□□□□□□□□□□□□□□□□□□□□□□□□　　users□□□□□□□　　cards
□□□□□□□□□□□□□□□□□□□□□□□□□□□□　　2□□□□□□□□□□□　　user_cards
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

見ての通り、このテーブルにはレコード ID、ユーザー ID、カード ID、そのカードの所持枚数という 4 つのカラムしかない。テストユーザーとカードはすでに用意してあるので、それぞれのプレイヤーにカードを持たせよう。手動で追加してもいいし、以下の SQL文を使って手早く初期カードを用意してもよい。

	id	user_id	card_id	count
編集 コピー 削除	1	1	1	2
編集 コピー 削除	2	1	2	1
編集 コピー 削除	3	1	5	1
編集 コピー 削除	4	1	8	2
編集 コピー 削除	5	1	6	1
編集 コピー 削除	6	2	3	5
編集 コピー 削除	7	2	6	3
編集 コピー 削除	8	2	12	1

```
INSERT INTO `user_cards` (`id`, `user_id`, `card_id`, `count`) VALUES
(1, 1, 1, 2), (2, 1, 2, 1), (3, 1, 5, 1),(4, 1, 8, 2),
(5, 1, 6, 1), (6, 2, 3, 5), (7, 2, 6, 3),(8, 2, 12, 1);
```

API?????

テーブルとテストデータが揃った。次は、プレイヤーが所有するカードを返す PHPスクリプトを作る。 Unity 側のコードは詳細なカード情報を必要とするため、ランク、イラスト番号、モンスター名、各種数値などすべてを送る必要がある。ただし、データを重複させたくないので、カードの枚数も一緒に送ることにする。

最終的な JSONは次のような形になる

```
{
  "card_list" : [
    {
      "card" : {
```

```
        "id" : 1,
        "rank" : 0,
        "picture_id" : 0,
        "monster_name" : "弱虫スケルトン",
        "atk_value" : 3,
        "def_value" : 1,
        "hp_value" : 10
      },
      "count" : 2
    },
    ...
  ]
}
```

PHP ?????

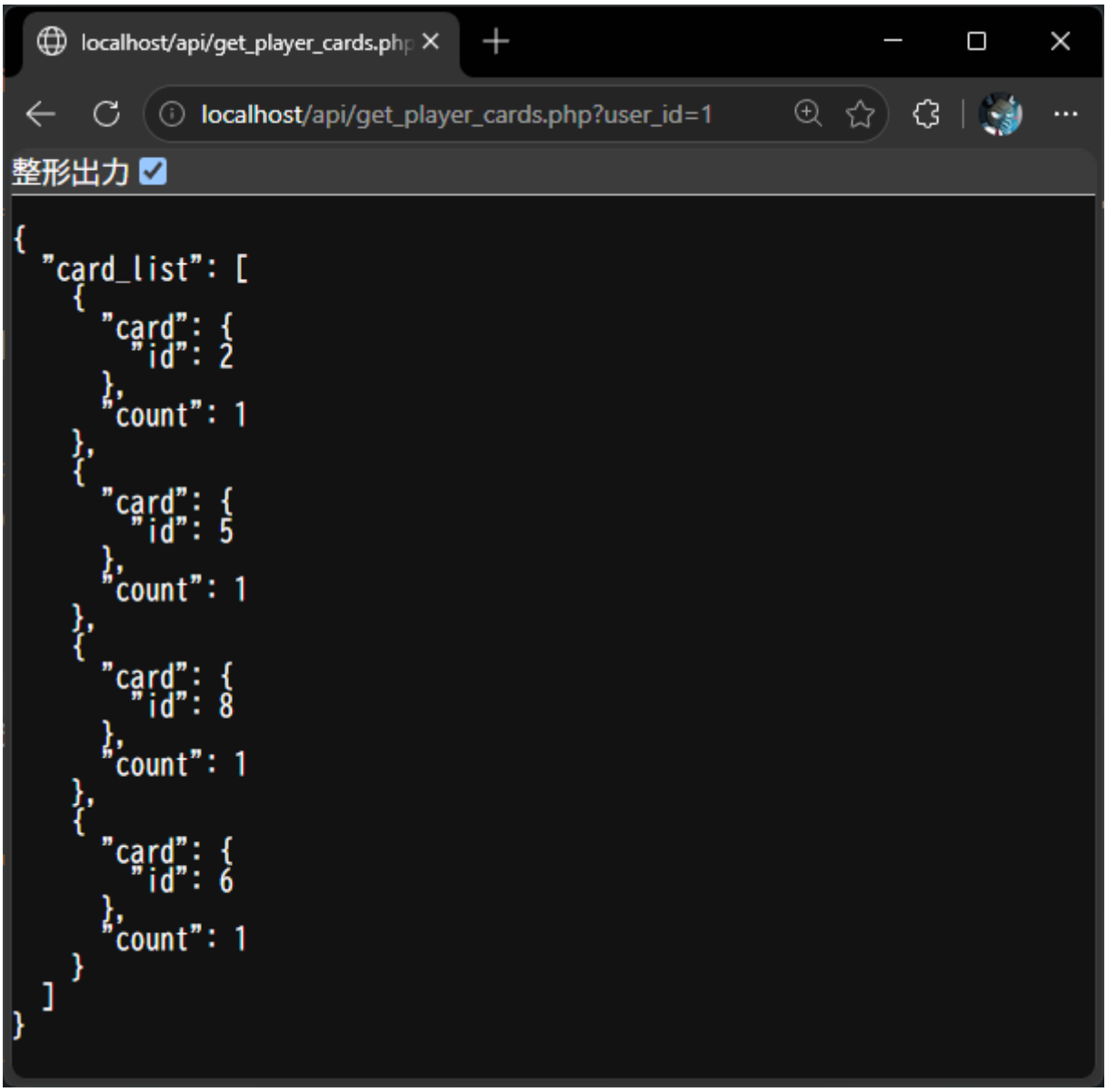
まず、データベースにクエリを送る。パラメータとしてユーザー ID
を受け取り、そのユーザーが持っているすべてのカードを検索する。カードを持っていない場合や、ユーザーが存在しない場合は、空の配列を返す。

テスト用として、ユーザー IDは `GET` パラメータで渡し、最終版では `POST` に変更する。
`get_player_cards.php` ファイルを作成し、以下のように実装しよう。

```
<?php
// JSONで返すべき
header("Content-Type: application/json; charset=utf-8");

//接続
$dbh = new PDO('mysql:host=localhost; dbname=card_game', "root", "", [PDO::ATTR_PERSISTENT => true]);

//ユーザーIDを取得
$user_id = $_GET['user_id'];

//そのユーザーを持っているカードを検索
$sql = "SELECT * FROM user_cards WHERE user_id = " . $user_id;
$result = $dbh->query($sql);

//返すデータを準備する
$json["card_list"] = [];
```

```
□
□while ($record = $result->fetch())
□{
□□//□ード1枚に関するデータ
□□$card_data['card']['id'] = $record['card_id'];
□□$card_data['count']      = $record['count'];
□□
□□//□果に追加
□□array_push($json["card_list"], $card_data);
□}

□print(json_encode($json));
?>
```

ここでは、ユーザー ID を受け取り、そのユーザーが持っているすべてのカードを検索して返している。しかし、以下のテストからわかるように、



これはカードの詳細情報ではなく、カードの ID を返しているだけである。Unity で必要なデータを取得するには、2 つのテーブルを結合して同時に検索する必要がある。これは JOIN 操作と呼ばれる。まずは phpMyAdmin でテストしてみよう。以下の SQL クエリから始める。

```
SELECT * FROM user_cards WHERE user_id = 1;
```



□□□ PHP□□□□□□□□□□□□□ cards□□□□□□□□□ card_id□ cards□□□□□□□ id □□□□□□□□□□□□□□□□□□□□□□□

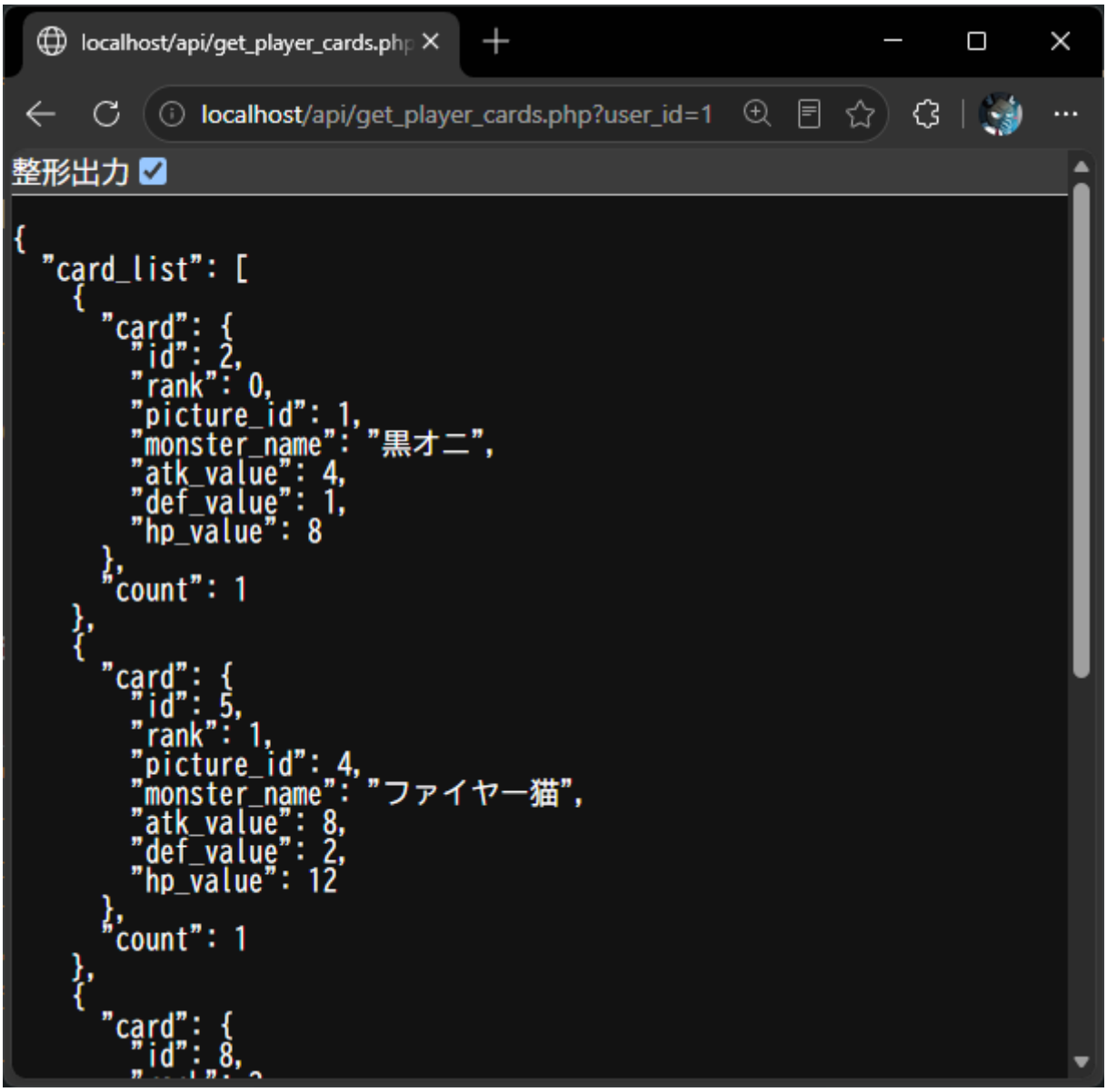
```
SELECT * FROM user_cards, cards WHERE user_id = 1 AND user_cards.card_id = cards.id;
```



□□□□□□□□□□ user_id = 1 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ SQL □□□□□□□□□ PHP□□□□□□□□□□□□□□□□□□□□□□□□

```php
<?php
// JSONレスポンス
header("Content-Type: application/json; charset=utf-8");

//接続
$dbh = new PDO('mysql:host=localhost; dbname=card_game', "root", "", [PDO::ATTR_PERSISTENT => true]);

//ユーザーIDを取得
$user_id = $_GET['user_id'];

//ユーザーが所持しているカードを取得するSQL
$sql = sprintf("SELECT * FROM user_cards, cards WHERE user_id = '%s' AND user_cards.card_id = cards.id", $user_id);
$result = $dbh->query($sql);

//カードリストを初期化
$json["card_list"] = [];

while ($record = $result->fetch())
{
    //1枚のカード情報をUnityのJObjectに合わせる
    $card_data['card']['id']           = $record['id'];
    $card_data['card']['rank']         = $record['rank'];
    $card_data['card']['picture_id']   = $record['picture_id'];
    $card_data['card']['monster_name'] = $record['monster_name'];
    $card_data['card']['atk_value']    = $record['attack'];
    $card_data['card']['def_value']    = $record['defense'];
    $card_data['card']['hp_value']     = $record['hit_points'];

    $card_data['count'] = $record['count'];

    //1枚追加
    array_push($json["card_list"], $card_data);
}

print(json_encode($json));
?>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□



これは、求めていた JSON形式である。あとはスクリプト内の `$_GET` を `$_POST` に変更すれば、次のステップに進む準備は完了である。

Unity????API????

`APIRequest` クラスを更新し、 `get_player_cards.php` にアクセスして、使える形に変換する新しいメソッドを追加しよう。

```
// プレーヤーが所有しているカードを求める
// user_id: ユーザーID
public static async Awaitable<JArray> GetPlayerCardsAsync(string userID)
{
    // POSTでデータを準備
    var postData = new WWWForm();
```

```
    postData.AddField("user_id", userID);

    // APIのURLを設定（適宜変更）
    const string url = ServerAPI + "/get_player_cards.php";
    var webRequest = UnityWebRequest.Post(url, postData);

    // リクエスト送信
    await webRequest.SendWebRequest();

    // レスポンス取得
    var webResponse = webRequest.downloadHandler.text;

    // JSONからPlayerCardリストを作成
    JObject response = JObject.Parse(webResponse);

    // 仮
    return (JArray)response["card_list"];
}
```

??????????????

□□□□　　ID□ `LoginController.UserID`
□□□
`CardListController` □□□□□□□

```
// カードリストを管理
public class CardListController : MonoBehaviour
{
    // 表示する
    public void Show()
    {
        screenSlide.Show();
        _ = LoadCardsAsync();
    }

    // 選択したカードを削除
    private void DeleteSelectedCards()
    {
        // 仮実装
    }
```

```
    // カードを読み込み、画面に追加し、表示する
    private async Awaitable LoadCardsAsync()
    {
        // すべてのカードーを削除する
        foreach (Transform child in cardsList.transform)
            Destroy(child.gameObject);

        // オンライン機能を実装する前、いくつかのカード手動に追加
        await Awaitable.NextFrameAsync();
        var cards = FakeCards();

        // 1枚ずつを追加
        foreach (var item in cards)
        {
            var count = (int)item["count"];
            for (var i = 0; i < count; i++)
            {
                var card = (JObject)item["card"];
                var copy = Instantiate(cardPrefab, cardsList.transform);
                copy.SetData(card);
                copy.SetSelectable(true);
            }
        }
    }

    // オフラインテスト用のカード
    private JArray FakeCards()
    {
        // （省略）
    }
}
```

これは非常にシンプルなスクリプトである。画面に表示されているカードをすべて消してから、テスト用のカードを読み込んでいる。「削除」ボタンは現時点では何もせず、「戻る」ボタンはトップメニューに戻る。

APIを使うには、オフラインのテストカードを置き換えて、サーバーにカードをリクエストするだけでよい。`APIRequest` メソッドはすでに用意してあるので、ここで使ってみよう。

```
// カードを読み込み、画面に追加し、表示する
private async Awaitable LoadCardsAsync()
{
    // すべてのカードーを削除する
    foreach (Transform child in cardsList.transform)
        Destroy(child.gameObject);

    // サーバーにカードの一覧を要求
    var cards = await APIRequest.GetPlayerCardsAsync(LoginController.UserID);

    // 1枚ずつを追加
    foreach (var item in cards)
    {
        var count = (int)item["count"];
        for (var i = 0; i < count; i++)
        {
            var card = (JObject)item["card"];
            var copy = Instantiate(cardPrefab, cardsList.transform);
            copy.SetData(card);
            copy.SetSelectable(true);
        }
    }
}
```

もう一度ログインして、プレイヤーのカードを読み込んでみよう。



□□□□□□□□□□□□□□□□□□□□□□□□□

✔ 行 0 - 4 の表示 (合計 5, クエリの実行時間： 0.0005 秒。)

```
SELECT * FROM user_cards, cards WHERE user_id = 1 AND user_cards.card_id = cards.id;
```

プロファイリング [インライン編集] [編集] [EXPLAIN で確認] [PHP コードの作成] [更新]

すべて表示 | 行数: 25 行フィルタ: このテーブルを検索

拡張オプション

id	user_id	card_id	count	id	rank	picture_id	monster_name	attack	defense	hit_points
1	1	1	2	1	0	0	弱虫スケルトン	3	1	10
2	1	2	1	2	0	1	黒オニ	4	1	8
3	1	5	1	5	1	4	ファイヤー猫	8	2	12
4	1	8	2	8	2	5	カエルる	10	5	20
5	1	6	1	6	1	3	スカルゴースト	6	5	10

Revision #12
Created 2026-08-04 07:13:09 UTC by Menendez Francisco
Updated 2026-08-25 05:18:26 UTC by Menendez Francisco