セクション 9-ハンズオンスクリプト

■SELECT の活用①

まずは

use ecsite;

と入力して、データベースを指定します。

次に

show tables;

と入力して、インポートされたテーブルを確認します。

次に

SHOW COLUMNS FROM users;

と入力すると、users テーブル内の列（カラム）を確認することができます。

これによって、どういったデータのタイプが使用されていて、Null 値が可能かどうかなどのテーブル構造の設定状況も確認することができます。

users テーブルのそれぞれのカラムの設定状況を確認できました。

次に

SHOW keys FROM users;

と入力してみます。これによって、プライマリーキーとして設定している値を確認することができます。Users テーブルでは ID がプライマリーキーとなっているようです。

このように show コマンドを利用することで、様々な確認をすることが可能となります。

これ以外にも show status ステータスを表示させたり、show logs でログを表示させるといったコマンドなど多数のコマンドがありますので、ご自身で活用してみてください。

それではここから SELECT コマンドを利用して、様々なデータを読み込んでみます。

まずは

SELECT * from users;

と入力して、ユーザーテーブルのユーザーデータを取得します。この*マークは全てのデータを取得する際に利用します。

これを実行すると

このように多数のユーザーの名前やメールアドレスなどの基本情報がリストとして表示されるのがわかります。

次は特定のカラムのみを取得してみます。

ここでは名前だけを抽出します。

SELECT name from users;

と入力して、実行してみると、name 列だけを取得することができます。

次に複数列を取得するコマンドを入力していきます。

id と name を出力してみます・

SELECT id, name from users;

と入力して、実行すると id と name の２つの列を取得することができます。

さらにもう 1 列増やして、id, name, birthday を取得してみます。

SELECT id, name,　birthday from users;

と入力して実行します。

このように 3 列が取得できます。

このように SELECT は最も利用する SQL コマンドと言えるものであり、データ抽出やデータ表示をする際に絶対に利用するコマンドです。抽出範囲を条件づけてさらに絞り込んだりするために、この SELECT コマンドに他のコマンドを付与して利用していくことになります。次のレクチャー移行では、SELECT+@の方法を学習していきます。

■WHERE の活用

早速先ほどのクエリコマンドを入力してみましょう。

SELECT * FROM users WHERE birthday = ‘2002-03-09’;

と入力して実行すると、誕生日が 2000 年のユーザーだけが抽出されます。

次に

SELECT name, birthday FROM users WHERE birthday = ‘2002-03-09’;

と入力してみましょう。すると、条件を付けたうえで、列を指定してデータを抽出することができます。

次に複数条件にしてみましょう。

SELECT name, birthday FROM users WHERE birthday = ‘2002-03-09’ AND birthday = ‘2003-06-07’;

と入力してみると、２つの誕生日に該当するユーザーのラストネームと誕生日列が抽出されることがわかります。

このように WEHRE 句を利用することで、SELECT 文によるデータ抽出に条件づけることができます。

さらに様々な条件をつけることができますが、演算子のレクチャーにおいて、様々な演算式の形式を学習する際に説明させていただきます。このレクチャーはここで終了となります。

■AS の活用

SELECT id AS user_id FROM users;

と入力して、実行します。

すると、id が user_id として名称変更された状態で出力されました。

つぎに複数行で実施してみます。

SELECT id AS user_id, name AS fullname, birthday AS DOB FROM users;

と入力して、実行してみましょう。これで複数行の列名を変更してデータを抽出することができることが分かったと思います。

このように AS 句を SELECT 文に加えることで、表示される列名を好きなように変更することができるようになります。

■DISTINCT の活用

まずは order_details テーブルにどういったデータが入っているのかを確認してみましょう。

SELECT * FROM order_details;

と入力して、実行します。

このようにユーザーid となる id に紐づいて、注文番号となる order_id と製品番号となる product_id が記録されており、amount が注文された総量になっています。[Created_at]はデータが作成された日付で、updated_at はデータが更新された日付を示しています。これはデータ更新履歴を保持する際に標準的に利用される形式です。

それでは order_id が複数重複しているため、DISTINCT を利用して重複を排除して表示してみましょう。

SELECT DISTINCT order_id FROM order_details;

と入力して、実行します。すると、重複が排除された状態で出力されました。

次に product_id の重複削除をしつつ、出力します。

SELECT DISTINCT id , product_id FROM order_details;

と入力して、実行します。すると、製品番号については、重複削除はされておらず、1~9 までの番号が使用されていることがわかります。

次に、二つの列を指定します。

SELECT DISTINCT order_id, product_id FROM order_details;

と入力して、実行します。

列を複数選択した場合は、その二つのカラムの組み合わせで重複削除を行います。

このように DISTINCT を利用することで、列ごとにデータに重複がないようデータ抽出を実行することができるようになります。

■ORDER BY の活用

まずは、users テーブルの全体をみます。

SELECT name FROM users;

デフォルトでは id の昇順で表示していいます。

SELECT name FROM users ORDER BY name ;

と入力して、実行します。すると name で昇順ソートされました。

次に複数列を抽出してみましょう。

SELECT id, name, birthday FROM users ORDER BY name,;

と入力して、実行します。これにより name によって昇順で表示されることがわかります。

また、このように出てきた列はデフォルトで、昇順で表示されます。これを DESC というオプション

を加えることで降順にすることができます。

name で降順にしてみましょう。DESC は最後に入力します。

SELECT name FROM users ORDER BY name DESC;

と入力して、実行します。このように後順に表示されます。

複数列でも実行してみましょう。

SELECT id, name, birthday FROM users ORDER BY name DESC;

と入力して、実行します。

次に birthday で降順にしてみましょう。

SELECT id, name, birthday FROM users ORDER BY birthday DESC;

と入力して、実行します。

このように birthday に基づいて降順で表示されます。

ORDER BY の指定列の番号を指定することで書くことも可能です。。たとえば id で

ORDER BY を指定する場合は 1 でも指定が可能です。

SELECT id, name, birthday FROM users ORDER BY 1:

と入力して、実行します。すると name に基づいて昇順で抽出されます。

次に 2 列目を指定見ましょう。

SELECT id, name, birthday FROM users ORDER BY 2:

と入力して、実行します。すると name に基づいて昇順で抽出されます。

次に３列目を指定見ましょう。

SELECT id, name, birthday FROM users ORDER BY 3:

と入力して、実行します。すると birthday に基づいて昇順で抽出されます。

また複数で順序を設定することもできます。id と birthday を両方とも降順に設定してみましょう。

SELECT id, name, birthday FROM users ORDER BY id, birthday;

このデータだとわかりづらいですが、複数行で順番を指定することができます。

このように ORDER BY を利用することで、データの表示順序を特定の列を指定して設定することができるようなります。

■LIMIT の活用

まずはリミットをかけずに実行してみます。

```
SELECT
    id, name
FROM
    users
ORDER BY name DESC;
```

name で降順表示されています。

それでは limit をかけて実行してみます。コマンドは下記になります。

```
SELECT
    id, name
FROM
    users
ORDER BY name DESC
LIMIT 10;
```

実行します。

すると 10 行だけ id と name が表示されることがわかります。

数値を 5 に変えてみましょう

```
SELECT
    id, name
```

```
FROM
    users
ORDER BY name DESC
LIMIT 5;
```

5 行だけ表示されます。数字を変更することで表示数を自由に変更できるわけです。

それでは 3 行目から 5 行取得するといった表示方法はどのように実施すれば良いでしょうか？

その場合は、このように入力をしていきます。

```
SELECT
    id, name
FROM
    users
ORDER BY name DESC
LIMIT 2,5;
```

と入力して、実行すると 2 行抜かして 3 行目から 5 行分のデータを抽出してくれます。3 行目から取得したい場合に２と１つ数字を減らして入力することがポイントです。

それでは、8 行目から 10 行取得してみましょう。

```
SELECT
    id, name
FROM
```

```
    users
ORDER BY name DESC
LIMIT 7,5;
```

と入力して、実行すると 8 行目から 10 行分のデータを取得してくれます。

次に OFFSET 句を追加することで、データ表示の範囲を変更するやり方を説明します。

このように LIMIT と OFFSET を利用することでデータ抽出するデータ数を変更することができます。

```
SELECT
    id, name
FROM
    users
ORDER BY name DESC
LIMIT 10 OFFSET 7;
```

と入力して、実行すると先ほどと同じ結果となることがわかります。

OFFSET 句は LIMIT の後ろに設定して、表示される行を何行目からにするのかを指定することができるコマンドとなっています。さきほどの OFFSET 7 と入力することで、7 行分を飛ばして表示するということになります。

■LIKE の活用

早速先ほどのコマンドを入力しています。

```
SELECT
    name
FROM
    users
WHERE
    name LIKE '%da%';
```

と入力して、実行します。

このように名前のどこかに da が含まれているユーザーが一覧として表示されることがわかりま

す。

次に後ろだけ%を利用したコマンドを入力します。

```
SELECT
    name
FROM
    users
WHERE
    name LIKE 'da%';
```

と入力して、実行します。

すると da から始まる名前の方が 2 名だけ表示されることがわかります。

次に前だけ%を利用したコマンドを入力します。

```
SELECT
    name
FROM
    users
WHERE
    name LIKE '%da';
```

と入力して、実行すると

こうすると名前の一番後ろが da で終わる名前のみを抽出します。

次に特定の桁数の数値を抽出する LIKE の使い方を実践します。

```
SELECT
    id, name
FROM
    users
WHERE
    id LIKE '_';
```

と入力して、実行すると一桁の id を有しているユーザーのみが抽出されます。

次に 2 桁の id を持っているユーザーを抽出します。

```
SELECT
    id, name
FROM
```

```
    users
WHERE
    id LIKE '__';
```

と入力して、実行します。すると二けたの ID を持っているユーザーが表示されたことがわかりま

す。

次に 3 桁の ID を持っているユーザーを抽出します。

```
SELECT
    id, name
FROM
    users
WHERE
    id LIKE '___';
```

と入力して、実行します。すると 3 けたの ID を持っているユーザーが表示されたことがわかりま

す。

このように LIKE を利用することで、部分的な情報からデータを検索して、絞り込むことができ

るようになり、LIKE は検索を柔軟に実施したい際に非常に便利なコマンドとなっています。

■GROUP BY の活用

さっそく先ほどのコマンドを入力してみます。

SELECT * FROM order_details GROUP BY amount;

と入力して、実行します。

すると、amount は１と２と３までの数値しか利用されていないため、3 つのグループに分かれることがわかります。

ちなみに他の行に入っている値は全てグループ化された amount の単位が昇順で最初に登場した行の値をとってきています。例えば、amount が 30 で id1 のようなものは取ってきていないです。

実際に GROUP BY を削除して実行していきましょう。

SELECT * FROM order_details;

入力して、全てを表示すると、それぞれの amount が初登場した ID が１と２と５の行が取得されていたことが確認できます。

それでは、次に条件をさらにつけて、order=id が 2 桁の注文内容からデータを取得してみましょう。

```
SELECT
    *
FROM
    order_details
WHERE
    order_id LIKE '_'
GROUP BY amount;
```

と入力して、実行します。そうすると order_id が 2 桁のデータの中で amount によってグループ化してくれます。

このように条件を付けたうえで、グループ化するといった指定方法ができるわけです。

これによって、特定のデータをグループとしてまとめてデータを抽出する際に、GROUP BY を利用して操作することになります。

■HAVING の活用

まずは、HAVING を使用する前に、GROUP BY を使用して出力してみます。

```
SELECT
    *
FROM
    order_details
GROUP BY amount;
```

こちらの通り、amount により三つのグループに分かれました。

HAVING はこのグループ化された後に、条件を絞り込み抽出する役割があります。

それでは下記のコマンドを入力してください。

```
SELECT
    *
FROM
    order_details
GROUP BY amount
HAVING id = 5;
```

と入力して、実行します。

すると id=5 が指定されたグループ化後のデータしか抽出されないため、1 行だけデータが抽

出されたことがわかると思います。

また、条件を変更するとどうなるかも見てみましょう。

```
SELECT
    *
FROM
    order_details
GROUP BY amount
HAVING id > 1;
```

最後の id > 1 と変更して実行してみます。

すると 1 よりも大きな 2 以上の ID が表示されるため、２と５が表示されます。

ここまでで、HAVING も含めて、様々なデータ抽出に利用するクエリコマンドを学習してきました。最後に各コマンドの記載順序について整理してみましょう。

コマンドの記載順序は

1. SELECT
2. DISTINCT
3. AS
4. FROM
5. WEREH
6. LIKE
7. GROUP BY
8. HAVING

9. LIKE（再度 HAVING に対して LIKE が利用できます）
10. ORDER BY
11. LIMIT
12. OFFSET

という順序で記載していくことになります。

この順序を実際のコマンド例としては、以下のように記載していくことになります。

```
SELECT
    id AS order_details_id
FROM
    order_details
WHERE
    order_id LIKE '_'
GROUP BY amount
HAVING id > 10
ORDER BY id
LIMIT 1 OFFSET 1;
```

このコマンド自体は冗長ですので効果的ではありますが、実行できるかを確認してみましょ

う。

```
SELECT
    id AS order_details_id
FROM
    order_details
WHERE
    order_id LIKE '_'
GROUP BY amount
```

```
HAVING id > 10
ORDER BY id
LIMIT 1 OFFSET 1;
```

と入力して、実行します。

すると 1 行だけ絞り込まれて表示されたので、実行されたことがわかります。

ここまでで、様々なコマンドを利用したデータ抽出の学習は終了となります。最後にケーススタディを実施して、学習した内容を定着化していきましょう。

■SELECT 文を駆使したケーススタディ演習 1 解説

それでは演習 1 の回答を実施します。

まず答えとなるデータ抽出をする前に、どのように抽出するべきかを検討することが必要です。

そのために一旦指定されている ID とユーザー名と誕生日を全て出力して確認してみましょう。

SELECT id, name, birthday FROM users;

この中から 6 月生まれの方だけを抽出することがお題となっています。

そのためにはどのコマンドを利用するべきかを考えてみます。

まずは WHERE 句を利用することはわかりますが、WHERE で 6 月だけを指定するのはそのままでは無理だとわかるでしょう。

そのため、LIKE を加えて、“%06%”といったように誕生日に 06 が含まれているユーザーを検索できるようにすることが思いつけば回答に近づいています。

ただし、これだけでは、2006 年や 6 日といった年と月が合致したパターンも抽出してしまう可能性があるため、不十分です。

そこで 6 月の 06 の間が「-ハイフン」で囲まれていることに注目してください。

したがって、"%-06-%"と指定してあげれば、6 月だけを抽出することができることになりま

す。

それでは入力してみましょう。

```
SELECT
    id, name, birthday
FROM
    users
WHERE
    birthday LIKE '%-06-%';
```

と入力して、実行します。

これで 6 月誕生日のユーザーのみを取得することができました。

■SELECT 文を駆使したケーススタディ演習２解説

それでは演習 2 の回答を実施します。

まずはどのように抽出すればよいのかを考えるため、全てのユーザーリストを抽出してみます。

SELECT * FROM users;

ここから一番大きな ID のユーザーのみを抽出するのですから、100 番を条件づければ一発で抽出することができます。

SELECT * FROM users WHERE id =100;

簡単でしたね。しかし、これで良いでしょうか？

本来は最大の ID を持っているユーザーはユーザー登録がどんどん実施されていくことで、更新されていくことになります。つまり、時間の経過とともに ID の 100 番が最も ID が大きいユーザーとは限らなくなってくるわけです。

その場合、他に最大の数を抽出する方法は何があるでしょうか？

データを ID の降順に並び替えれば、一番トップに出てくるデータが最大の ID を有したユーザーということになります。そして、LIMIT 1 と設定して、先頭行の 1 行だけを抽出すれば、ID が追加されたとしても必ず一番大きな ID を抽出してくれるクエリ文になります。

それでは入力してみましょう。

```
SELECT
    *
FROM
    users
ORDER BY id DESC
LIMIT 1;
```

と入力して、実行します。

これで例え 101 番や 1000 番が追加されたとしても、一番 ID が大きなユーザーを常に抽出されることができます。

■SELECT 文を駆使したケーススタディ演習 3 解説

それでは演習 3 の回答を実施します。

まずはどのように抽出すればよいのかを考えるため、全ての order_details テーブル内のデータを抽出してみます。

```
SELECT * FROM order_details;
```

今回はここから、order ID と Product ID に基づいてデータを絞り込むことが必要となります。

そのためには、order ID を LIKE を利用して 2 桁に絞り込む条件検索を実施します。

```
SELECT
    *
FROM
    order_details
WHERE
    order_id LIKE '_';
```

と入力して、実行します。これで order ID が二桁のものを絞ることができました。

これで order ID が 2 桁のデータのみが抽出されます。さらに Product_id が 6 番の製品に対する発注のみにしぼりこみます。

```
SELECT
    *
FROM
    order_details
WHERE
    order_id LIKE '_' AND product_id = 6;
```

と入力して、実行します。

これで order ID と Product ID に基づいてデータを絞り込むことができました。

■SELECT 文を駆使したケーススタディ演習 4 解説

それでは演習 4 の回答を実施します。

まずはどのように抽出すればよいのかを考えるため、全てのユーザーリストを抽出してみます。

SELECT * FROM users;

ここから 20 代のユーザーのみに絞り込みますが、ここで年齢がないため、利用するのは誕生日ということになります。そのため条件文を加えて、20 歳となる誕生日を絞り込むことができれば、20 代のユーザーのみを抽出することができます。

現在が 2020 年 1 月 1 日と指定されているため、1990 年代の誕生日ユーザーは全て 20 代のユーザーとなります。

どのように 1990 年代に絞り込めばよいかというと、LIKE を利用することになります。

```
SELECT
    id, name, birthday
FROM
    users
WHERE
    birthday LIKE '1990%';
```

と入力して、実行してみます。すると 1990 年生まれの方だけが抽出されてしまいました。

これでは失敗です。

90 年代生まれのユーザーを抽出するには、"1990%"ではなく、"199%"と指定する必要があ

るからです。

```
SELECT
    id, name, birthday
FROM
    users
WHERE
    birthday LIKE '199%';
```

入力しましょう。

これで 1990 年代の誕生日のユーザーのみが抽出されました。

さらに表示される列名 name を変更します。

```
SELECT
    id, name AS 20s_users, birthday
FROM
    users
WHERE
    birthday LIKE '199%';
```

これで 20 代のユーザーリストが抽出できました。

■SELECT 文を駆使したケーススタディ演習５

それでは演習５の回答を実施します。

ここではどこのテーブルを利用するかをまずは考える必要があります。

まずは users テーブルの中身を確認して、一度でも購入したことがある人を探してみましょう。

SELECT * FROM users;

このようにユーザー一覧が出ますが、購入履歴は確認できません。

次に orders テーブルを見てみます。確かに購入履歴は表示されていますが、本当にこのまま orders テーブルを読み込めばいいのでしょうか？

結論としては答えになりません。

その理由としては、user_id が重複してしまっているためです。そのため、orders テーブルから取得する際に、重複を削除した上で出力しなければなりません。

したがって、正解はこちらになります。

SELECT DISTINCT user_id FROM orders;

これで一度でも購入履歴があるユーザーのリストが抽出されました。

SELECT 文を駆使したケーススタディ演習６解説

それでは演習６の回答を実施します。

まずは WHERE を利用して、グループ化をしていきます。

コマンドはこちらになります。

```
SELECT
	*
FROM
	order_details
WHERE
	amount > 1
GROUP BY product_id;
```

と入力して、実行します。

そうすると WEHRE によって最初に 2 以上の発注があった注文内容に限定した上で、製品

は 9 つありますので、グループ化されてまとめられて、9 行のデータが抽出されます。

グループ化を削除して実行してみます。

```
SELECT * FROM order_details WHERE amount > 1;
```

で実行します。

そうすると amount が 2 以上のデータが大量に抽出されていることがわかります。このデータに対して、product_id によるグループ化を行っているわけです。

次にグループ化を先に実施した上で、HAVING を利用して総量を 2 つ以上の注文に限定します。

```
SELECT
    *
FROM
    order_details
GROUP BY product_id
HAVING amount > 1;
```

と入力して、実行します。すると先ほどとちがって、6 行しかデータが抽出されませんでした。これは先にグループ化を実施していることが理由です

HAVING を抜いて確認してみましょう。

```
SELECT
    *
FROM
    order_details
GROUP BY product_id;
```

とし入力して、実行すると、製品は 9 つありますので、グループ化されてまとめられて、9 行のデータが抽出され、そのあとに 9 つのデータに対して、HAVING を利用して総量を 2 つ以上の注文に限定することになり、最終的に 6 つのデータに絞られているわけです。

このように似ているような条件式ですが、結果が変わることに注意してください。