セクション 11−ハンズオンスクリプト

■基本的な演算式の活用

論理式の際にやりましたが、四則演算などの実行は SELECT を実行することで関数を実行することができます。

SELECT 1 + 2;

で足し算を実行できます。

SELECT 1 − 2;

で引き算を実行できます。

SELECT 1 * 2;

で掛け算を実行できます。

SELECT 3 / 2;

で割り算を実行できます。

関連する関数を実行してみます。

SELECT 3 DIV 2;

を実行しても、小数点以下を省略して整数値のみの割り算の結果を算出します。

SELECT 3 % 2;

を実行すると、割り切れなかった余りが算出されます。

SELECT 3 MOD 2;

を実行しても、割り切れなかった余りが算出されます。

これが基本的な四則演算式として、よく活用される演算子と関数です。

テーブル内のデータに対しても検証してみましょう。今回はあまり意味はありませんが、

order_details テーブルの id と order_id を合計した数を sum_id として出力してみます。

まずは中身を見てみます。

SELECT * FROM order_details WHERE id = 9;

すると、order_details の id9 の order_id は 4 となっています。

9 と 4 で 13 となるはずです。コマンドは下記の通りです。

```
SELECT
    id + order_id AS sum_id
FROM
    order_details
WHERE
    product_id = 9;
```

と入力して、実行します。

これで sum_id という新しい合計した ID を利用した算出値が抽出されました。

今度は MOD を使って、あまりを出してみましょう。

9÷4 であまりは 1 になるはずです。

```
SELECT
	id MOD order_id AS MOD_id
FROM
	order_details
WHERE
	product_id = 9;
```

すると 1 が出力されたと思います。

このようにテーブルからデータ抽出する際にその中を計算してから、算出するといったようにクエリ句内で計算を実行する必要がある際に、四則演算式を利用することができるわけです。

次に数値に関する計算に利用する関数を実行していきます。

絶対値を算出する ABS 関数を確認します。

```
SELECT ABS(-10);
```

と実行すると１０が

```
SELECT ABS(10);
```

と実行しても１０が表示されます。

これは絶対値と呼ばれるものです。絶対値はプラスでもマイナスでも同じ整数値として表示します。

次に、四捨五入を実施する ROUND 関数を確認します。

SELECT ROUND(2. 4);

で四捨五入をすることができるため、2 が算出されます。

指定した数値よりも高い整数を取得する CEILING 関数を確認します。

SELECT CEILING(2.4);

では 3 が、この数値よりも高い数値で一番近いため表示されます。

指定した数値以下の最小の整数を取得する FLOOR 関数を確認します。

SELECT FLOOR(2.6);

と入力して実行すると、2 が表示されます。

これは CEILING とは逆で、指定した値よりも低い整数を算出します。

小数点の切り捨てを指定する TRUNCATE 関数を確認します。

SELECT TRUNCATE(1.0384,4);

では、小数点第 4 位まで出すということで、1.0384 と表示する範囲を小数点以下を指定して表示することができます。四捨五入はされません。

これを 1.03 まで表示したい場合は下記のコマンドになります。

SELECT TRUNCATE(1.0384,2);

このような関数を利用して、SELECT 文から抽出するデータの表示範囲を変えることができるわけです。

■集計関数①

まずは最初に COUNT 関数を利用していきましょう。

```
SELECT
	COUNT(amount)
FROM
	order_details
WHERE
	product_id = 9;
```

と入力して、実行すると、product_id = 9 番となっている製品が order_details

テーブルの中にいくつあるのかを数えてくれます。列数を数えていますので、列指定は

Amount になっていますが、どの列でも同じ結果となります。すると 66 レコードあることがわかり

ます。

```
SELECT
	COUNT(id)
FROM
	order_details
WHERE
	product_id = 9;
```

id にしてみても、同じ結果となることがわかります。

```
SELECT
	COUNT(*)
FROM
	order_details
```

```
WHERE
    product_id = 9;
```

アスタリスクで全部指定してみても、かわりません。あくまでも行を数えるためです。

DISTINCT を入れて、少し複雑にしてみます。

```
SELECT
    COUNT(DISTINCT amount)
FROM
    order_details
WHERE
    product_id = 9;
```

で実行してみると、amount は１と２と３しかデータが入っていないため、DISTINCT で重複

された AMOUNT を数えますので、３という結果になりました。

以前実行した GROUP BY でこの amount によるグループ化をしてみます。

```
SELECT
        amount
FROM
    order_details
WHERE
    product_id = 9
GROUP BY amount;
```

で実行します。

というように、１， ２， ３と 3 行でてきました。GROUP BY は COUNT と同じような集計をして、グループごとにまとめてくれるという意味では、集計関数と同じ働きをしている関数になります。

先ほどの GROUP BY でさらに COUNT してみます。

```
SELECT
        count(amount)
FROM
    order_details
WHERE
    product_id = 9
GROUP BY amount;
```

で実行します。

このようにグループ化した上で、グループ内の数をカウントしてくれました。

COUNT は出てくるデータ数を数えるのに非常に便利な関数であり、利用頻度も高いため、ぜひ使えるようになっておきましょう。

■集計関数②

それでは次に SUM を実行してみましょう。

先ほどと同じ order_details テーブルの製品 9 番の amount を今度は、合計していきます。

```
SELECT
    SUM(amount)
FROM
    order_details
WHERE
    product_id = 9;
```

と入力して、実行します。

すると今度は、注文された製品の amount を合計して、全体の注文量を抽出することができました。136 と表示されました。product_id が 9 の製品番号の発注送料が 136 を示しています。COUNT と違って列ごとの合計を算出するため、列を変えれば結果も変わります。

あまり意味はありませんが、id を合計してみると

```
SELECT
    SUM(id)
FROM
    order_details
WHERE
    product_id = 9;
```

ID が積みあがって合計されるため、かなりの合計値になります。

次に同じように平均値を算出してみましょう。

```
SELECT
        AVG(amount)
FROM
        order_details
WHERE
        product_id = 9;
```

と入力して、実行します。注文ごとの総量を平均するため、小数点以下までかなり細かく表

示されているのがわかります。これが一回の注文当たりの、平均発注数となります。

先ほどの SUM と同時に使うこともできます。せっかくなので列名も変更してみましょう。

コマンドはこちらになります。

```
SELECT
    AVG(amount) AS avg_amount
    SUM(amount) AS sum_amount
FROM
        order_details
WHERE
        product_id = 9;
```

と入力して、実行します。するとそれぞれの平均と合計が同時に算出できます。こういった感

じで集計リストなどをいっぺんに抽出することも可能となるわけです。

最後にグループ化と組み合わせてみましょう。

```
SELECT
AVG(amount) AS avg_amount,
SUM(amount) AS sum_amount
FROM
    order_details
WHERE
    product_id = 9
GROUP BY amount;
```

とすると amount の１と２と３ごとに集計をかけてくれます。これは平均は意味がなくなってしまっていますが、合計値は amount ごとにだしてくれていて便利です。

合計と平均についてはここで終了となります。この関数もかなり使うことが多いため、是非使いこなせるようになりましょう。

■集計関数③

次に最大値と最小値を出していきましょう。

コマンドは下記です。

```
SELECT
    MAX(price)
FROM
    products;
```

と入力して、実行すると、products テーブルにある製品の中で、最も高い価格を抽出しま

す。

その際に抽出する項目は price になります。

次に最小値を算出してみます。

```
SELECT
    MIN(price)
FROM
    products;
```

とすれば、最小値を算出することができます。

少し複雑なデータの出し方を考えていきます。これだけだと最大と最小の価格をだしているだけなのですが、最大の価格である製品情報をすべてデータ抽出する場合はどうすればよいでしょうか？

MIN の指定値に*をおいても、これは実現できません。

長いのですが、以下のように設定します。

```
SELECT
    *
FROM
    products
WHERE
    price =
            (
        SELECT
            MAX(price)
        FROM
            products
            );
```

と入力して、実行します。WHERE 句の中に SELECT 文を記載します。

すると価格が最大の製品情報をすべて抽出してくれます。条件文自体にさきほどの最大値の算出式をもってきているわけです。

最小値でもやってみましょう。

```
SELECT
```

```
    *
FROM
    products
WHERE
    price =
            (
        SELECT
            MIN(price)
        FROM
            products
            );
```

これで最小価格の製品情報を抽出することができました。

グループ化とも組み合わせてみましょう。

あまり意味はありませんが、Amount でグループ化した場合の最大の製品番号を抽出してみます。

```
SELECT
    MAX(product_id)
FROM
    order_details
GROUP BY amount;
```

と実行してみますとグループごとに抽出されます。

このように集計関数は条件付けなどを実施する様々なコマンドを組み合わせ使うことになります。以上で集計関数のレクチャーは終了となります。

■文字列に関する関数②

まずは CONCAT 関数から実施します。これは文字列と文字列を連結する関数です。

データベース内の文字列を連結させて抽出させたい場合に重宝します。

SELECT 文のカラムを指定する個所で利用します。

select concat('A','B','C');

と入力して、実行すれば、ABC と連携した文字が返ってきます。

また

select concat('A',' ','C');

といったように、スペースを空けて文字をつなげることもできます。

つぎにテーブルに対して実行してみましょう。

```
SELECT
    CONCAT(name, birthday)
FROM
    users;
```

と入力して実行すると、

名前と誕生日がつながった連結したデータを抽出することができます。

このテーブルは名前が 1 列ですが、ファーストネームとラストネームというように性別が分かれて

いる場合にフルネームを取得する際などは便利な関数です。

次は LOWER 関数を実行してみましょう。これは文字列を小文字に置換する関数です。

SELECT LOWER('AAA');

と入力して、実行すれば小文字で帰ってくるのがわかります。

それでは users テーブルの名前をすべて小文字で返してみましょう。

```
SELECT
    LOWER(name)
FROM
    users;
```

と入力して、実行すると全てが小文字の名前が抽出されます。

UPPER 関数を確認します。これは文字列を大文字に変更します。

SELECT UPPER('aaa');

と入力して、実行すれば大文字で帰ってくるのがわかります。

それでは users テーブルの名前をすべて大文字で返してみましょう。

```
SELECT
    UPPER(name)
```

```
FROM
    users;
```

と入力して、実行すると全てが大文字の名前が抽出されます。

次に REPLACE 関数を実行していきましょう。これは REPLACE('abc', 'a', 'A')と利用すると、文字列 abc の中の a を A に変更してくれます。

```
SELECT REPLACE('abc', 'a', 'A');
```

と入力して、実行してみます。

このようにリプレースされるのがわかります。

それでは users テーブルの名前のアルファベットの小文字 a だけを大文字にリプレースしてみましょう。

```
SELECT
    REPLACE(name,'a','A')
FROM
    users;
```

このように名前の途中にある小文字であるはずの a が A に代わっていると思います。

■文字列に関する関数③

次に INSERT 関数を利用してみます。

これは INSERT(‘abcd’,1,2,’xy’)と利用していくと、文字列 abcd の 1 文字目から 2 文字目

を

`XY`に置き換えることができます。

SELECT INSERT('abcd',1,2,'xy');

を実行してみます。と実際に ab が xy に変更されているのがわかります。

それでは users テーブルの名前の先頭 5 文字をすべて A にしてみます。

```
SELECT
    INSERT(name,1,5,'AAAAA')
FROM
    users;
```

と入力して、実行します。

このように全ての名前の最初の 5 文字が AAAAA に変更されてデータが抽出されます。

次に LPAD 関数を利用してみます。

これは LPAD(`ab`,6,`A`)と利用していくと、文字列 ab が 6 文字になるように左側を A で埋

めてくれます。

SELECT LPAD('ab',6,'A');

と入力して、実行してください。このように左サイドが AAAA でうまって、6 文字になりました。

同様の結果になります。

それでは users テーブルの名前の A で埋めてみましょう。

```
SELECT
    LPAD(name, 30 ,'A')
FROM
    users;
```

と入力して、実行します。

すると、A で 30 文字になるまで左サイドを埋めてくれます。

逆サイドを埋める RPAD にしてみましょう。

同じく

SELECT RPAD('ab',6,'A');

と入力して、実行してください。このように右サイドが AAAA でうまって、6 文字になりました。

それでは users テーブルの名前の A で埋めてみましょう。

```
SELECT
    RPAD(name, 30 ,'A')
```

```
FROM
    users;
```

と入力して、実行します。

すると、A で 30 文字になるまで右サイドを埋めてくれます。

次は REPEAT 関数を使ってみます。これは指定した数だけ文字列を繰り返してくれます。

```
SELECT REPEAT('OK',5);
```

と入力して、実行します。すると OK を 5 回繰り返した文字列が返ってきます。

先ほどと同じように name 列に適用してみます。

```
SELECT
    REPEAT(name, 2)
FROM
    users;
```

と入力して、実行すると名前を 2 回繰り返してくれます。

■文字列に関する関数④

次に LENGTH 関数を利用してみます。

SELECT LENGTH('abcd');と入力して、

実行するとバイト数を数えてくれます。バイト数を数えますので、日本語の場合は数がことなります。

SELECT LENGTH('日本語');

と入力して、実行すると、想定より多い数値がかえってきました。これは日本語という言葉は3 バイトずつ使用されているため、9 バイトを数えてしまうためです。

この場合は、文字数単位で数える関数を利用します。

それが CHAR_LENGTH 関数です。

SELECT CHAR_LENGTH('日本語');

と入力して、実行すると 3 が返ってきました。日本語の文字数を数える場合はこちらを使わないといけません。

ちなみにアルファベットに使ってみると

SELECT CHAR_LENGTH('abcd');

というように変わらず 4 が抽出されるため、どちらにも適用可能です。

このように様々な関数がありますが、データベースからデータ抽出する際にこういった形式で文字列を出したいなってときに、どの関数を利用すれば実現できるのかを考えて利用していくことになります。あまり使わない関数も出てきてしまいますが、頭に存在だけでもいれておいて、思い出せるくらいで良いでしょう。

■日付と時刻に関する関数①

まずは基本となる CURRENT_DATE と CURRENT_TIME と CURRENT_TIMESTAMP の３つの関数を同時に学習しましょう。これは現在の日付や時刻を表示させることができる関数です。

コマンドは下記になります。

SELECT CURRENT_DATE (), CURRENT_TIME (), CURRENT_TIMESTAMP ();

と３ついっぺんに実行してみましょう。

CURRENT_DATE ()が年月日のみ、CURRENT_TIME (),は時刻のみ、

CURRENT_TIMESTAMP()は両方の現在時点のデータを表示またはデータ登録させることができます。

これは TIMESTAMP などの実行で利用することが多いので、ここでついでに TIMESTAMP 関数も一緒に学習していきましょう。

あとで削除しますが、一時的に simpleid テーブルというものを作成してみます

その際に TIMESTAMP 型を利用して、デフォルト値に先ほどの関数を入れてみます。

CREATE TABLE simpleid

```
(
	id INT(5),
	created_at TIMESTAMP DEFAULT NOW()
);
```

と入力して、実行します。

これだけでは CURRENT_TIME は実行されないため、データを入れてみましょう。

```
INSERT INTO simpleid (id)
VALUES(1);
```

と入力して、実行して ID 1 のデータを入れていきます。

データを確認しましょう

```
SELECT * FROM simpleid;
```

こうすることで TIMESTAMP 型を指定して、デフォルト値に現在の年月日時刻を記入することができます。データが作成された時刻や変更された時刻をデータベースに記憶する際にはこの方法を利用します。

別の方式で日時を抽出する方式をやってみましょう。

次の 3 つの関数

TIMESTAMPDIFF
DATEDIFF
IMEDIFF

は日付や時刻同士の差分を抽出することができる関数です。

SELECT DATEDIFF('2020-11-31','2020-01-20');

を実行すれば、日付の差分を抽出できます。

SELECT TIMEDIFF('2020:02:02 23:00:00','2020:01:30 00:00:00');

と実行すれば、時刻の差分を抽出することができます。

SELECT TIMESTAMPDIFF(MONTH,'2020-01-01','2020-02-01');

と実行すれば日時の差分を抽出することができます。

■日付と時刻に関する関数②

次に時刻を取得する関数を複数実行していきます。

まずは DAYNAME 関数を利用してみましょう。

これは曜日を返答する関数です。

SELECT DAYNAME('2020-10-12');

と入力して、実行します。曜日が出力されました。

次は MONTH 関数を利用してみましょう。

これは月だけを出力する関数です。

SELECT MONTH('2020-10-12');

と入力して、実行します。月が出力されました。

同様に DAY と YEAR でも日と年を出力してくれます。リストにはいれていませんでしたが、つ

いでにやってみましょう。

SELECT YEAR('2020-10-12');

を実行すれば年が出力されます。

SELECT DAY('2020-10-12');

を実行すれば日が出力されます。

次から指定した日数を数える関数を利用してみます。

SELECT DAYOFMONTH('2020-10-12');

を入力して実行すると、その月における経過日数を出力します。

SELECT DAYOFYEAR('2020-10-12');

を入力して実行すると、その年における経過日数を出力します。

SELECT DAYOFWEEK('2020-10-12');

を入力して実行すると、その週における経過日数を出力します。

それでは、次は日付の出力形式を変更する関数を見ていきます。

DATE_FORMAT 関数は日付の出力形式フォーマットを変更することができます。

今回は４つのフォーマットに変えていきます。

SELECT DATE_FORMAT('2020-10-12', '%a');

曜日が頭文字 3 文字で表示されます。

SELECT DATE_FORMAT('2020-10-12', '%b');

月が頭文字 3 文字で表示されます。

SELECT DATE_FORMAT('2020-10-12', '%c');

月が数字で表示されます。

SELECT DATE_FORMAT('2020-10-12', '%D');

日付が英語表記で表示されます。

このほかにも様々なフォーマット形式がありますが、フォーマット形式については MySQL の公式ページなどに掲載されていますので、日付や時刻の見せ方を変えたいとおもったら、そういったページを参考にフォーマットを変更してみてください。

最後に日付の利用するタイムゾーンを変更する CONVERT_TZ 関数を利用してみます。

NOW 関数などを利用すると現在は皆様のローカルなタイムゾーンを利用して時刻が刻印されるようになっているのですが、システムを作成する際にアメリカのタイムゾーンを利用したいといった設定をすることがあります。そういった際に利用するのが CONVERT_TZ 関数です。

指定した東京の時刻をニューヨークの時刻に変更してみます。

SELECT CONVERT_TZ('2020-10-1 06:34:24','Asia/Tokyo','America/New_York');

で実行しますと、タイムゾーンをニューヨークに変更した上で日付が出力されます。

このように様々な日付関連の関数はあり、他にも様々な関数がありますが、利用することが少ない者も多いため、主要なものだけを学習しました。それではこのレクチャーを終了します。

■その他の関数①

まずは現在利用しているデータベースに関する情報を取得する関数を利用します。

SELECT DATABASE();

を入力して、実行すると現在利用しているデータベースが表示されます。

SELECT CURRENT_USER();

を入力して、実行すると現在利用しているユーザー名を表示します。

SELECT VERSION();

を入力して、実行すると現在利用している MySQL バージョンを表示します。

次に文字セットを確認する関数を学習します。

SELECT CHARSET('Abbbrc');

と入力して、実行すると、現在利用されている文字セットを確認することができます。

SELECT COLLATION ('Abbbrc');

と入力して、実行すると、現在利用されているより文字セットに加えて、照合順序を確認す

ることができます。

次にデフォルト値を確認する関数を学習します。

```
SELECT
    DEFAULT(id)
FROM
```

users;

と入力して、実行するとデフォルト値に何が設定しているかがわかります。

Id は 0 がデフォルト値として設定されてます。

最後にデータ型を変更する関数を利用していきます。

CAST 関数と CONVERT 関数で、どちらも同じ役割をしており、指定した値を別のデータ型に変換します。

CAST 関数は（変更前のデータ型 AS　変更後のデータが）と入力して実行します。

SELECT CAST(NOW() AS signed);

と入力すると、NOW 関数によって表示される日付時刻が signed 型で表示されるため数値データ化します。

CONVERT 関数は（変更前のデータ型、変更後のデータ型）と入力して実行します。

SELECT CONVERT(NOW(), signed);

と入力すると、先ほどと同じ結果がでてきました。NOW 関数によって表示される日付時刻が signed 型で表示されるため数値データ化されています。

これを利用して、現在のデータベースの型を変更してみましょう。

SELECT

```
    CONVERT( id , DECIMAL (5 , 3 ))
FROM
    users;
```

と入力して、実行すると、INT 型で登録されている整数が、　DECIMAL 型で、小数点 3 桁まで表示される設定で変更されます。

このようにデータ型を変更してデータを抽出することができます。

ここまでで様々な関数の学習が終了しますが、これ以外にも様々な関数が存在します。だいたいは普段使わないことが多いため、ここで学習した主要な関数のみで十分ですが、MySQL 公式ページなどに全ての関数が掲載されていますので、適時どういった関数があるのかを確認しながら使っていくと良いかと思います。

それでは、最後に必要のなくなったテーブルを削除します。

```
DROP TABLE simpleid;
```

で simpleid テーブルを削除します。

これで、その他の関数に関するレクチャーを終了します。

■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。

まずはどのように週出するのかを order_details を確認して、検討してみましょう。

```
SELECT
    *
FROM
    order_details;
```

で確認してみましょう。ここで amount の合計値を算出することが目的ですので、先ほど学習した関数から SUM を利用することがわかると思います。

リストは product_id と合計発注量を表示します。また、合計発注量は sum_amount を列名に表示することになりますので、AS を利用します。

```
SELECT
    product_id, SUM(amount)  AS sum_amount
FROM
    order_details;
```

入力して、実行してみましょう。

こうすると全ての amount が合計されてしまいました。これを product_id ごとに仕分ける必要があります。そのためには GROUP BY を利用して、product_id ごとにグループ化することになります。

```
SELECT
    product_id, SUM(amount) AS sum_amount
FROM
    order_details
GROUP BY product_id;
```

これで product_id ごとにグループ化することができましたので、ほぼ要件どおりの表になってきました。しかしながら、まだ、リストは売れ筋順に並べる対応ができていません。

これを実現するには amount に対して ORDER BY を追加する必要があります。また売れ筋順なので降順ということになるため DESC を加えることも必要です。

```
SELECT
    product_id, SUM(amount) AS sum_amount
FROM
    order_details
GROUP BY product_id
ORDER BY amount DESC;
```

これで出来たと思いきや、よく見ると降順になっていな個所もあるようです。これは

SUM(amount)ではなく、amount に対して並び順を整理しているためです。

SUM(amount)に基づいて降順に整理するためには、ORDER BY で SUM(amount)を指定することが必要です。

```
SELECT
    product_id, SUM(amount) AS sum_amount
FROM
    order_details
GROUP BY product_id
ORDER BY SUM(amount) DESC;
```

これで目的のリストができました。少々難しかったですが、１つ１つじっくりと仕上げればできるようになりますので、このステップを覚えていきましょう。

■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。

先ほどの SQL クエリに対して、抽出項目を追加します。

```
SELECT
    product_id,
    SUM(amount) AS sum_amount,
    AVG(amount) AS avg_amount,
    MAX(amount) AS max_amount,
    MIN(amount) AS min_amount
FROM
    order_details
GROUP BY product_id
ORDER BY SUM(amount) DESC;
```

と入力して、実行すれば、このように各集計が列挙されたリストが表示されます。

■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。

これを実行してみます。

さきほどの文章を表示させるには、CONCAT 関数を利用して抽出したデータと、追加する文字を結合することになります。

```
SELECT
    CONCAT('製品 ID','product_id', 'の商品の総注文量は', SUM(amount),'です。') AS sum_amount
FROM
    order_details
GROUP BY product_id
ORDER BY SUM(amount) DESC;
```

と入力して、実行します。

すると実行はできましたが、上手く表示されません。このようになっているときはデータ型や文字コードがあっていない可能性があります。抽出されたデータの文字コードを確認してみます。CHARSET

または COLLATION を利用しますが、CONCAT されたデータに対して、どのように設定するのかが難しいところです。これは CONCAT 関数の前に設定することで実施することができます。

```
SELECT
    CHARSET(CONCAT('製品 ID','product_id', 'の商品の総注文量は', SUM(amount),'
です。') ) AS sum_amount
FROM
    order_details
GROUP BY product_id
ORDER BY SUM(amount) DESC;
```

と入力して、実行すると、全てバイナリ値として出力されていることがわかります。これでは文字を表示することができません。それではデータ型を変更します。CAST 関数または CONVERT 関数を利用して変換します。これも先ほどと同じように CONCAT 関数の前に指定します。

```
SELECT
    CAST(CONCAT('製品 ID',product_id, 'の商品の総注文量は', SUM(amount),'です。
') AS CHAR) AS sum_amount
FROM
    order_details
GROUP BY product_id
ORDER BY SUM(amount) DESC;
```

と入力して、実行すると、これで文章形式で結合した内容が 1 列で表示されました。

この問題はかなり難易度が高かったと思いますが、それぞれの関数の使い方の関連性を深く学ぶことができたのではないでしょうか。

■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。

これは基礎的な内容の復習に近いケーススタディです。

テーブルを作成して、NOW 関数や CURENT_TIMESTAMP 関数などを TIMESTAMP として

設定していきます。

```
CREATE TABLE case4
(
      id INT(5) PRIMARY KEY AUTO_INCREMENT,
          name VARCHAR(255),
      created_at TIMESTAMP DEFAULT NOW()
);
```

ID を auto_increment,にしたため、インプットデータは name だけで大丈夫です。

```
INSERT INTO case4(name)
VALUES('Andy'),('Mary'),('Sam');
```

で実行すれば、全てのデータが投入されます。

では中身を確認します。

```
SELECT * FROM case4;
```

問題なく実行されていることがわかりました。

■ケーススタディ演習５解説

それでは演習５の回答を実施します。

１つ目の要件として、・Andy という名前を Mr.Andy に変更するということをクエリにしていきま

す。

```
SELECT
	REPLACE(name, name, 'Mr.Andy')
FROM
	case4
WHERE name = 'Andy';
```

という条件式になります。これで Andy が'Mr.Andy に変更されてデータ抽出されます。

その上で、・次のステップとして全てを大文字にして、データを抽出していきます。

```
SELECT
	UPPER(REPLACE(name, name, 'Mr.Andy'))
FROM
	case4
WHERE name = 'Andy';
```

となります。これで完了です。

最後にいらない今後はテーブルなので、作成したテーブルを削除します。

DROP TABLE case4;

を実行します。これで削除できました。