セクション 12-ハンズオンスクリプト

■INNER JOIN①

まずは今回利用するデータベースを準備します。

以前作成した HR データベースをもう一度作成していきます。今回は講義の進行としてはロ

ーマ字にした日本文がない HR データベースを利用していきます。また、今回の講義用に投

入するデータは少し増やします・

作成はめんどうですので、データベースの作成自体は添付してあり、TEXT ファイルからコピー

して貼り付けて実施していきましょう。

まずはデータベースを作成します。

```
CREATE DATABASE hr_data2;
```

次にデータベースを指定してからテーブルを作成します。

```
USE hr_data2;
```

Employees テーブルを作成します。

```
CREATE TABLE employees
(
   employee_id INT(6) PRIMARY KEY,
      last_name VARCHAR (15) NOT NULL,
      first_name VARCHAR(15) NOT NULL,
```

```
    sex VARCHAR(1) NOT NULL,
    joining_date DATE NOT NULL,
    age INT(3) NOT NULL,
    department VARCHAR(30) NOT NULL DEFAULT "unassigned",
    rater INT(6)
);
```

データを投入します。

```
INSERT INTO employees
 (employee_id, last_name, first_name, sex, joining_date, age, department, Rater)
 VALUES
 (1001,'Sato','takashi','m',20180401,23,'Sales',1006),
 (1002,'Endo','Maki','w',20160401,24,'HR', 1006),
 (1003,'Kudo','Takaaki','m',20100401,30,'Devlopment', 1005),
 (1004,'Shin','Ando','m',20100401,32,'Devlopment', 1005),
 (1005,'Andy','Bogard','m',20100401,35,'Devlopment',''),
 (1006,'Kow','Murasame','m',20100401,38,'Devlopment','');
```

これは黄色の警告がでますが、データは投入されますので無視してください。

evaluations テーブルを作成します。

```
CREATE TABLE evaluations
 (
        id INT(6) PRIMARY KEY AUTO_INCREMENT,
        employee_id INT(6) NOT NULL,
        year YEAR NOT NULL,
        rating VARCHAR(1) NOT NULL,
        explanation VARCHAR(1000) NOT NULL,
 FOREIGN KEY (employee_id) REFERENCES employees (employee_id)
 );
```

データを投入します。

```
INSERT INTO evaluations
 (employee_id, year, rating, explanation)
VALUES
 (1001,2019, 'A','Very good'),
 (1002,2019, 'B','Good'),
 (1003,2019, 'E','Need Training'),
 (1005,2019, 'A','Very good');
```

Payrolls テーブルを作成します。

```
CREATE TABLE payrolls
 (
    id INT(6) PRIMARY KEY AUTO_INCREMENT,
employee_id INT(6) NOT NULL,
      position VARCHAR(20) NOT NULL,
      decided_date DATE NOT NULL,
      payroll_rate INT(20) NOT NULL,
      FOREIGN KEY (employee_id) REFERENCES employees (employee_id)
 );
```

データを投入します。

```
INSERT INTO payrolls
 (employee_id, position, decided_date, payroll_rate)
 VALUES
 (1001,'Sales staff',20190301,200000),
 (1002,'Leader',20190301,250000),
 (1003,'Manager',20190301,300000),
 (1006,'Manager',20190301,400000);
```

それでは作成したデータベースのテーブル内容を確認しましょう。

SHOW TABLES;

でテーブルを確認して、

SELECT 文で各データが投入されていることを確認します。

SELECT * FROM employees;

SELECT * FROM evaluations;

SELECT * FROM payrolls;

■INNER JOIN②

それでは JOIN を実行します。

まずは左サイドにもってきたいテーブルを先に入力します。今回は evaluations テーブルです。SELECT 文でデータを抽出するのと同じコマンドとなります。

SELECT * FROM evaluations

次に INNER JOIN と入力した後に、先ほどのテーブルに結合したいテーブル名を入力します。

INNER JOIN employees

そして、ON のあとに結合する外部キーを条件として指定します。今回は employee_id を外部キーとして利用して結合します。employee_id がどちらの employee_id であるかを、ドットをつけて指定することが必要です。

ON evaluations.employee_id = employees.employee_id;

完成コマンドはこちらです。

```
SELECT * FROM evaluations
INNER JOIN employees
ON evaluations.employee_id = employees.employee_id;
```

それでは実行します。

このように実行した結果、evaluations テーブルの左に employees テーブルが結合していることがわかります。

そして、結果としては 4 行のデータが抽出されています。INNER JOIN の場合は結合に用いる外部キーが両方のテーブルに存在している行のみを結合するため、この 4 行が結合されているわけです。

先ほどの ON の条件付けの際に、ドットをなくして実行してみます。

```
SELECT * FROM employees
INNER JOIN evaluations
ON evaluations.employee_id = employee_id;
```

employee_id がどの employee_id かわからないため、曖昧ですというエラーが発生してしまいました。

テーブルで一意の ID であれば、指定しなくても大丈夫ですが、どちらにも同じカラム名称を利用している場合は指定しないとカラムを特定ができないため、エラーとなります。

次に指定するテーブル名を入れ替えてみます。employees テーブルを先に指定して、

evaluations テーブルを後からもってきます。

```
SELECT * FROM employees
INNER JOIN evaluations
ON evaluations.employee_id = employees.employee_id;
```

すると予想通りですが、左サイドに employees がくるように結合されました。JOIN の結合順

序は指定した順序となるわけです。

次に INNER JOIN を単なる JOIN にして実行してみます。

```
SELECT * FROM employees
JOIN evaluations
ON evaluations.employee_id = employees.employee_id;
```

これは全く同じ結果となります。INNER JOIN と JOIN は同じ結果となるコマンドなわけです。

JOIN の基本が INNER JOIN であり、あえて INNER を指定しなくても、JOIN でも良いわけ

です。しかしながら、現在利用しているのが INNER JOIN ですよと明記していた方が後でコマ

ンドを見直した際にわかりやすいこともあり、INNER JOIN とあえて記載することをお勧めしま

す。

次に条件を付けて実行してみます。男性だけを表示してみましょう。

```
SELECT * FROM employees
```

```
INNER JOIN evaluations
ON evaluations.employee_id = employees.employee_id
WHERE SEX = 'm';
```

と入力して実行すると、男性だけに絞ることができました。

順番を変更してみます。

```
SELECT * FROM employees
INNER JOIN evaluations
ON evaluations.employee_id = employees.employee_id
WHERE SEX = 'm'
ORDER BY employees.employee_id DESC;
```

これで employee_id の降順にすることができました。

このように JOIN したテーブルも WHERE 句や GROUP BY 句や HAVING 句などでデータを絞り込むことができるわけです。

INNER JOIN は最も利用することが多い JOIN の 1 つであるため、使いこなせるようにしましょう。

■LEFT JOIN

先ほどのコマンドを実行します。

```
SELECT * FROM employees
LEFT JOIN evaluations
ON evaluations.employee_id = employees.employee_id;
```

と入力して実行すると、employees テーブルが左テーブルとなるのですが、その左テーブルの内容をすべて表示していることがわかります。これが外部結合です。内部結合の場合は両方のテーブルで関連した外部キーがマッチしているデータのみを表示していましたが、外部結合はマッチしていないデータも表示する方式です。つまり外部キーがマッチしている"外側"のデータまでも表示するから外部結合というわけです。

その中で LEFT JOIN は左テーブルの外部のデータも表示する方式となります。右テーブルでマッチしていないレコードは NULL 値が表示されています。

LEFT JOIN を LEFT OUTER JOIN としてみましょう。

```
SELECT * FROM employees
LEFT OUTER JOIN evaluations
ON evaluations.employee_id = employees.employee_id;
```

すると同じ実行結果がでました。

LEFT JOIN と LEFT OUTER JOIN は同じコマンドであるため、LEFT OUTER JOIN と明示して入力することもできるわけです。

■RIGHT JOIN

先ほどのコマンドを実行する前に、まずは employees テーブルと evaluations テーブルを見て

おきましょう。

確認が終わったら次のコマンドを確認して実行します。

```
SELECT * FROM evaluations
RIGHT JOIN employees
ON evaluations.employee_id = employees.employee_id;
```

ここでは右テーブルを employees テーブルにすることで、内容がすべて表示されると同時に、

evaluations のマッチしていないレコードが Null 値になっていることを確認します。

実行すると想定通りの表示となっています。

RIGHT JOIN を RIGHT OUTER JOIN としてみましょう。

```
SELECT * FROM evaluations
RIGHT OUTER JOIN employees
ON evaluations.employee_id = employees.employee_id;
```

LEFT JOIN と同じように、RIGHT JOIN と RIGHT OUTER JOIN も同じコマンドということが

確認できました。

これで外部結合としては終了なのですが、右テーブルも左テーブルもどちらも全てのデータを

取得するという外部結合はできないのでしょうか？

SQL としては FULL JOIN または FULL OUTER JOIN というコマンドを実行することで、これ

を実現することができます。

```
SELECT * FROM evaluations
FULL JOIN employees
ON evaluations.employee_id = employees.employee_id;
```

と入力して実行すると、エラーとなってしまいました。

実は MYSQL5.7 ではこの FULL JOIN が実行できるようにはなっていません。

そもそも全てを外部結合するという結合方式はあまり意味がある結合とはならないため、実

行することがないということで、省かれてしまっているわけです。

■CROSS JOIN

先ほどのコマンドを実行してみます。

```
SELECT * FROM employees
CROSS JOIN evaluations
```

実行するとこのように多数の行が取得されました。

24 行取得されているのは、右テーブルが 6 行、左テーブルが 4 行あるため、全ての交差の組合せを取得するため、6×4 となって 24 行が取得されています。

今度は ON 句を入れて実行してみます。

```
SELECT * FROM employees
CROSS JOIN evaluations
ON evaluations.employee_id = employees.employee_id;
```

すると 4 行だけになってしまいました。これは今回実施した INNER JOIN と同じ結果となってしまいました。CROSS JOIN で外部キーのマッチを指定してしまうと、その外部キーを指定した状態で全ての交差条件を表示するようになりますが、外部キーをマッチした結果 1 行に限定される場合は INNER JOIN と同じ結果となります。

念のために、INNER JOIN を試してみましょう。

```
SELECT * FROM employees
INNER JOIN evaluations
ON evaluations.employee_id = employees.employee_id;
```

この通り、同じ結果になりました。

次に元の CROSS JOIN に対して、WHERE 句で条件を指定してみましょう。

```
SELECT * FROM employees
CROSS JOIN evaluations
WHERE evaluations.employee_id = employees.employee_id;
```

とすると、INNER JOIN の結果となりました。つまり、CROSS JOIN とは条件が指定されてい

ないで実行される結合処理ということです。そして、ON 句で実施していた条件句は、実は

WHERE 句でも実現できることがわかります。

CROSS JOIN と同じように

次に条件なしで、JOIN で実行してみます。

```
SELECT * FROM employees
JOIN evaluations;
```

とすると 24 行取得され、CROSS JOIN と同じ結果となってしまいました。

条件が指定されないで JOIN を実行すると CROSS JOIN と同じ実行をすることになります。

つまり、JOIN と CROSS JOIN はほとんど変わらない動きをしていることになります。

次に INNER JOIN としてみましょう。

```
SELECT * FROM employees
INNER JOIN evaluations;
```

これも同じ実行結果となります。

このように CROSS JOIN と INNER JOIN と JOIN はほとんど同じものであることがわかりましたが、CROSS JOIN と明記して実行するのは、コマンドを後から見て CROSS JOIN とわかりやすくするために必要ですので、常に CROSS JOIN と入力するようにしてください。

■SELF JOIN

今回の SELF JOIN では employees テーブルの従業員に対して、評価者（rater）となる上司が設定されており、その上司の名前などの詳細情報を結合したデータ抽出を行います。

早速、先ほどのコマンドを実行します。

```
SELECT
    *
FROM
    employees emp01
        JOIN
    employees emp02 ON emp02.employee_id = emp01.rater;
```

今回は結合するテーブルが同じなので、emp01 と emp02 のように明示的に名称を付けています。

実行すると、2 つの employees が結合されて、それぞれの従業員に対して、評価者の従業員が紐づいて表示されるリストがでてきました。

このように 1 つのテーブル内を結合するということもできるわけです。

テーブルの順序を変えてみましょう。

```
SELECT
    *
```

```
FROM
    employees emp02
        JOIN
    employees emp01 ON emp02.employee_id = emp01.rater;
```

すると評価者が先に来てしまうため、望んでいた表ではなくなってしまいました。これでは意味がありません。

このように SELF JOIN は結合順序が大事ですので、間違わないように設定してください。

■ 3つ以上のテーブルへの JOIN

下記のコマンドを実行します。

```
SELECT
    *
FROM
    employees
        INNER JOIN
    evaluations ON evaluations.employee_id = employees.employee_id
        INNER JOIN
    payrolls ON payrolls.employee_id = employees.employee_id;
```

と INNER JOIN を 2 回利用して結合します。

すると、3 つのテーブルで外部キーが共通している 3 つのレコードのみが抽出されました。

次に表示するカラムを限定してみましょう。

カラムを指定する際にテーブル名をいれて、どのカラムであるかがわかるようにすることが必要で

す。

```
SELECT
    employees.employee_id, employees.last_name, employees.first_name,
evaluations.rating, payrolls. payroll_rate
FROM
    employees
        INNER JOIN
    evaluations ON evaluations.employee_id = employees.employee_id
        INNER JOIN
    payrolls ON payrolls.employee_id = employees.employee_id;
```

と入力して実行すると、必要な情報のみを抽出した結合テーブルができました。

外部結合にすることもできます。INNER JOIN を RIGHT JOIN にしてみます。

```
SELECT
    employees.employee_id, employees.last_name, employees.first_name,
evaluations.rating, payrolls. payroll_rate
FROM
    employees
        RIGHT JOIN
    evaluations ON evaluations.employee_id = employees.employee_id
        RIGHT JOIN
    payrolls ON payrolls.employee_id = employees.employee_id;
```

NULL 値がある行が増えており、外部結合が実施されていることがわかります。片方は外部

結合、もう片方は内部結合とすることもできます。

まずは２つめを INNER JOIN としていきます。

```
SELECT
    employees.employee_id, employees.last_name, employees.first_name,
evaluations.rating, payrolls. payroll_rate
FROM
    employees
        RIGHT JOIN
    evaluations ON evaluations.employee_id = employees.employee_id
        INNER JOIN
    payrolls ON payrolls.employee_id = employees.employee_id;
```

すると最後に payrolls との外部キーがあっているテーブルにかぎってしまうと、NULL 値の行がなくなってしまいました。

最初を INNER JOIN にして次を RIGHT JOIN にしてみます。

```
SELECT
    employees.employee_id, employees.last_name, employees.first_name,
evaluations.rating, payrolls. payroll_rate
FROM
    employees
        INNER JOIN
    evaluations ON evaluations.employee_id = employees.employee_id
        RIGHT JOIN
    payrolls ON payrolls.employee_id = employees.employee_id;
```

最後は payrolls テーブルとアンマッチしている行も表示しているため、NULL 値レコードが抽出されています。

このように様々な結合パターンに利用することで、3 つ以上のテーブルをうまく結合するように実行することになります。

ここまでで JOIN で実行できることの、ほとんどを実施してみました。様々な結合方法がありますので、どういったリストを表示したいかに応じて使い分けられるようにしていきましょう。

■UNION

UNION は SELECT 文を使って 2 つの ID を指定して全カラムを抜き出します。

```
SELECT
	*
FROM
	employees
WHERE
	employee_id = 1001
UNION
SELECT
	*
FROM
	employees
WHERE
	employee_id = 1002;
```

と入力して、実行すると、2 つの SELECT 文を合わせた結果が抽出されました。

これだと単純ですので、条件を変更してみます。

たとえば、片方や id を条件に片方は性別を条件に指定してみます。

```
SELECT
	*
FROM
	employees
WHERE
	employee_id = 1001
UNION
SELECT
	*
```

```
FROM
    employees
WHERE
    sex = 'm';
```

と入力して、実行すると 1001 と男性のデータがすべて抽出されます。

この結果は男性と 1001 のユーザーが重複しているため、実際のところ UNION の効果が得ら

れていません。UNION は重複チェックを自動で実施してくれているわけです。

そこで今度は UNION ALL を利用してみます。

先ほどのコマンドに ALL を加えて、実行します。

```
SELECT
    *
FROM
    employees
WHERE
    employee_id = 1001
UNION ALL
SELECT
    *
FROM
    employees
WHERE
    sex = 'm';
```

すると ID1001 が二行抽出されました。このように UNION ALL は重複も含めてすべての行を

抽出することができます。

次に 3 つ以上の SELECT 文を UNION によって結合してみます。

先ほどのクエリに、3 つめに 30 歳以上の SELECT 文を加えてみます。

```
SELECT
	*
FROM
	employees
WHERE
	employee_id = 1001
UNION ALL
SELECT
	*
FROM
	employees
WHERE
	sex = 'm'
UNION ALL
SELECT
	*
FROM
	employees
WHERE
	age >= 30;
```

UNION ALL を使っているので、重複も含めて出力しています。

このように 3 つの SELECT 文を結合して結果を表示させてくれました。

■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。

今回のケーススタディ演習では先ほどまで利用していた hr_data2 を利用するわけではないので、以前のセクションで利用していた ecsite データベースを指定してあげましょう。

USE ecsite;

次に今回の要件に合致したデータをどこから取得するのかを考える必要があります。

今回取得する対象データは

・ユーザーの個人情報

・注文した商品の発注数

となります。users テーブルを確認しましょう。

SELECT * FROM users;

を実施してみましょう。このテーブルを確認したところ個人情報にあたる id、name, email, gender, birthday の 5 つをとってくることにします。

次に注文した商品の発注数です。order_details テーブルを確認してみます。

SELECT * FROM order_details;

を実行して、ここにある amount と product_id がセットになれば発注数と発注した商品がわかるようになるので、これを users テーブルのユーザーごとに結合して取得することで目的のリストが完成することがわかります。

それでは次に結合するための共通データを選択します。

すると order_details テーブルには users テーブルと共通したデータ項目がないことがわかります。これでは結合することができません。条件が指定できなければ、実施できるのは CROSS JOIN のみとなってしまいます。

そこで orders テーブルを確認してみます。

SELECT * FROM orders;

を実行してみると、ここに id となっているのが order_details における order_id となっており、user_id との組み合わせが設定されていることがわかります。したがって、

今回はこの 3 つのテーブルを結合してあげる必要があります。

まずは users テーブルと orders テーブルを結合していきます。それぞれ user_id を共通データとして指定して結合します。JOIN のタイプは「これまでに注文がないユーザーについてもリストとして抽出して把握できるようにする」ことが条件であったため、OUTER JOIN で users テー

ブル側のデータをすべて表示させることが必要となります。users テーブルは左テーブルですの

で、LEFT JOIN を利用します。

```
SELECT
    *
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id;
```

と入力して、実行します。これで結合ができました。

次に orders テーブルに対して、order_details テーブルを結合させます。orders テーブルと

order_details テーブルは過不足なくセットで作成されるデータであるため、INNER JOIN を利

用して結合していきますが、LEFT JOIN でも結果は変わりません。指定する共通データは

orders.id です。

```
SELECT
    *
FROM
    users
        LEFT JOIN
    orders
  ON users.id = orders.user_id
        INNER JOIN
    order_details
  ON order_details.order_id = orders.id;
```

と入力して、実行します。これで結合ができました。

あとは表示するカラム名を

Users テーブルから id、name, email, gender, birthday の 5 つをとってくることにします。

order_details テーブルから product_id と amount を取得するように絞り込んでリストを見やすくしましょう。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id;
```

となります。

これで目的のリストが取得できました。

■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。

ここで追加するべきデータが存在するテーブルを確認します。

商品名と金額は products テーブルにあります。

SELECT * FROM products;

を実行して、確認します。

ここで id となっているのは order_details テーブルにおける product_id ですので、product_id

を共通データとして指定してあげれば、JOIN が利用できることがわかりました。

演習 1 のクエリに追加していきましょう。order_details テーブルにおける product_id は確実に

products テーブルに存在しているはずですので、どの JOIN でも同じ結果になりますが、

INNER JOIN で結合するので良いでしょう。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, products.price
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
```

```
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
     INNER JOIN
    products
    ON order_details.product_id = products.id;
```

と後ろに加えて、実行します。

これで取得するデータに対するテーブル結合として問題ありませんが、価格が製品価格になってしまっていることで、注文数とあっていません。そのため、注文数に合わせた合計金額にする必要があります。

合計金額を出すためには、amount * price が必要なことがわかりますが、これをデータ抽出させるためには、どうしたら良いのか迷うところです。

じつはデータ抽出対象となるカラム名の個所に、直に計算式をもってくれば可能です。リストをわかりやすくするため、AS を利用して total_fee と名称を設定することにします。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, amount * price AS
total_fee
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
```

```
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id;
```

これで目的のリストができました。さらに消費税がわかるようにもしてみましょう。消費税は

10%と仮定して設定していきます。また、結果は user_id で昇順で表示されるようにします。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, amount * price AS
total_fee, amount * price * 1.1 AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
    ORDER BY users.id;
```

で実行すると、注文履歴リストが完成しました。

■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。

これは GROUP BY を利用してユーザーID でまとめればよいということはわかると思います。

それでは GROUP BY を付け加えてみましょう。

```
SELECT
	users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, amount * price AS
total_fee, amount * price * 1.1 AS tax_included
FROM
	users
		LEFT JOIN
	orders
	ON users.id = orders.user_id
		INNER JOIN
	order_details
	ON order_details.order_id = orders.id
			INNER JOIN
	products
	ON order_details.product_id = products.id
	    GROUP BY users.id
	ORDER BY users.id;
```

すると合計金額がうまく表示されていないようです。金額を合計するには SUM を利用していく必要があるためです。合計金額と税込み金額の個所に直接 SUM をいれていくことで計算してくれます。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, SUM(amount * price)
AS total_fee, SUM(amount * price * 1.1) AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
      GROUP BY users.id
    ORDER BY users.id;
```

これでユーザーごとの全ての注文に対する合計金額が算出されました。Product_id と発注数と商品名の列がユーザーごとの 1 行目を表示しており、価格と不整合が発生しているため、リストから除外します。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday, SUM(amount *
price) AS total_fee, SUM(amount * price * 1.1) AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
```

```
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
        GROUP BY users.id
    ORDER BY users.id;
```

となります。

最後にこれまでの合計注文した数がわかるように COUNT を利用して order の数を数えます。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday, COUNT(order_id)
AS order_count, SUM(amount * price) AS total_fee, SUM(amount * price * 1.1) AS
tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
        GROUP BY users.id
    ORDER BY users.id;
```

これで目的のリストが完成しました。

■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。

まずはレビュー内容をどこから取得するのかを考える必要があります。product_reviews というテーブルがありますので、その中身を確認しましょう。

```
SELECT * FROM product_reviews;
```

で確認すると、ここの body という列にレビュー内容が登録されていることがわかります。これを製品情報に加えればよさそうです。レビューが投稿されていない製品にはレビューがないことを把握したいため、左テーブルを products テーブルとして LEFT JOIN を利用したいと思います。結合する際に利用する共通データは product_id となります。

```
SELECT
    *
FROM
    products
        LEFT JOIN
    product_reviews
    ON products.id = product_reviews.product_id;
```

と入力して、実行すると結合ができました。さらにレビューを投稿したユーザー名と ID を取得するためには users テーブルを結合する必要があります。レビューしたユーザーと Users テーブ

ル内のユーザーは過不足なくマッチするため、INNER JOIN を利用します。結合に使用する

共通データは users_id です

```
SELECT
    *
FROM
    products
        LEFT JOIN
    product_reviews
    ON products.id = product_reviews.product_id
        INNER JOIN
    users
    ON product_reviews.user_id = users.id;
```

また、取得するカラムも必要なカラムに限定しましょう。次のように実施します。

```
SELECT
    products.id, products.name, product_reviews.body, product_reviews.user_id,
users.name
FROM
    products
        LEFT JOIN
    product_reviews
    ON products.id = product_reviews.product_id
        INNER JOIN
    users
    ON product_reviews.user_id = users.id;
```

これで目的のリストが完成しました。

■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。

まずは演習 2 で作成したリストをコピーして実行してみましょう。

```
SELECT
	users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, amount * price AS
total_fee, amount * price * 1.1 AS tax_included
FROM
	users
		LEFT JOIN
	orders
	ON users.id = orders.user_id
		INNER JOIN
	order_details
	ON order_details.order_id = orders.id
			INNER JOIN
	products
	ON order_details.product_id = products.id
	ORDER BY users.id;
```

このリストはユーザーの個別の注文詳細が把握できるリストです。

次に演習 3 で実施した、リストをコピーして実行してください。

```
SELECT
	users.id, users.name, users.email, users.gender, users.birthday, COUNT(order_id)
AS order_count, SUM(amount * price) AS total_fee, SUM(amount * price * 1.1) AS
tax_included
```

```
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
      GROUP BY users.id
    ORDER BY users.id;
```

これはユーザー毎の注文回数と合計金額を抽出するリストになっています。

この 2 つをつなげることで目的のリストができそうなので、UNION ALL を利用して結合していくことになります。

UNION ALL を利用するためには、取得するカラム数をあわせることが必要ですので、演習 2 を結合される最初の SELECT 文として設定して、演習 3 は SELECT 文をつなげるので、演習 3 側のカラムを合わせることが必要です。

次のように' ' で空白で埋めて、カラム数を一致させるようにしましょう。

```
SELECT
    users.id, '', '', '', '','', '', '合計', SUM(amount * price) AS total_fee, SUM(amount *
price * 1.1) AS tax_included
```

```
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
      GROUP BY users.id
    ORDER BY users.id;
```

これで実行してみます。このように合計金額だけを抽出することができました。

では 2 つの SELECT 文を結合していきましょう。

UNION の前に ORDER BY を入れていると上手くいかないので、ORDER BY を最後にもって

いきます。重複はありませんが、UNION ALL を利用しましょう。

```
SELECT
    users.id, users.name, users.email, users.gender, users.birthday, COUNT(order_id)
AS order_count, SUM(amount * price) AS total_fee, SUM(amount * price * 1.1) AS
tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
```

```
                INNER JOIN
    products
    ON order_details.product_id = products.id
        GROUP BY users.id
UNION ALL
SELECT
    users.id, '', '', '', '','', '', '合計', SUM(amount * price) AS total_fee, SUM(amount *
price * 1.1) AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
        GROUP BY users.id
    ORDER BY users.id;
```

しかしながら、これで実行すると 1250 エラーが発生して上手くいきません。

このエラーに対応するためには、各 SELECT 文を （） でくくることで、ORDER BY の適用範

囲を明確にしてあげることが必要となります。

```
(SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, amount * price AS
total_fee, amount * price * 1.1 AS tax_included
FROM
    users
```

```
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
            INNER JOIN
    products
    ON order_details.product_id = products.id)
UNION ALL
(SELECT
    users.id, "", "", "", "",  "", "", "合計", SUM(amount * price) AS total_fee,
SUM(amount * price * 1.1) AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
            INNER JOIN
    products
    ON order_details.product_id = products.id
      GROUP BY users.id)
    ORDER BY id;
```

で実行すると UNION が上手く実行されました。

しかしながら、よく見ると、合計金額の行が必ずしもユーザーID ごとに最後に表示されるよう

になっていないため、見た目がわるくなっています。そこで、最後の ORDER BY に total_fee を

加えます。

```
(SELECT
    users.id, users.name, users.email, users.gender, users.birthday,
order_details.product_id, order_details.amount, products.name, amount * price AS
total_fee, amount * price * 1.1 AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id)
UNION ALL
(SELECT

    users.id, "", "", "", "",   "", "", "合計", SUM(amount * price) AS total_fee,

SUM(amount * price * 1.1) AS tax_included
FROM
    users
        LEFT JOIN
    orders
    ON users.id = orders.user_id
        INNER JOIN
    order_details
    ON order_details.order_id = orders.id
                INNER JOIN
    products
    ON order_details.product_id = products.id
      GROUP BY users.id)
    ORDER BY id, total_fee;
```

これで目的のリストが完成しました。

以上でケーススタディ演習は終了となります。