セクション 6-ハンズオンスクリプト

■プライマリーキーの設定②

まずはデータベースを作成します。注文情報を登録するため、orders データベースという名称にします。

CREATE DATABASE orders;

と入力して、実行します。

次に users テーブルにプライマリーキーを設定して、作成します。プライマリーキーの設定自体は最初のテーブル作成でも実施しましたが、今回は設定意図を踏まえつつ実施してみてください。

USE orders;と入力して、作成したデータベースを指定します。

その上で、先ほどのコマンドを入力します。

```
CREATE TABLE users
(
    id INT(5),
    name VARCHAR(255),
    email VARCHAR(255),
    age INT(3),
    PRIMARY KEY (id)
);
```

と入力して、実行します。これによって、id をプライマリーキーとして設定したテーブルが作成されました。

この設定内容を確認してみます。

左タブの schema にいっていただいて、order テーブルをクリックすると、テーブルが表示されます。このテーブルを開くと users テーブルがでてきますので、そのテーブルをクリックしてください。

そうすると、カラムの設定状況が下に表示されます。id に PK と表示されていますが、この PK がプライマリーキーを表しています。このようにプライマリーキーとして設定されていることが確認できました。さらにユーザーテーブルの横にインフォメーションマークと設定マークが表示されるので、インフォメーションマークをクリックすると、詳細な情報を確認することができます。

ここで一番右の DDL というタブを開くと、プライマリーキーとしての設定した状況が開示されています。ここで USER_ID がプライマリーキーとして設定されているのに加えて、NOT NULL が設定されています。先ほどのコマンド内では NOT NULL は指定しませんでしたが、プライマリーキーとして設定することで自動的に NOT NULL が設定されることになります。

また、INDEX タブにいっていただきますと、ここでもプライマリーキーとして設定を確認することができます。また、プライマリーキーが INDEX としても設定されていることがわかります。プライマリーキーは自動的に INDEX としての設定もされるわけです。

次にテーブル名にあるインフォメーションマークの右にあるレンチマークをクリックしてみましょう。

すると、ここの画面で PK、NN といった列が記載されていて、チェックできるようになっています。ここのチェックを実施して、下にある APPLY ボタンを押すことでも、プライマリーキーなどの設定を後から加えたり、削除したりといった操作が可能となっています。

PK がプラマリーキーのことで、NN が NOT NULL、UQ がユニークキー、B が BINARY のことで、2 進数であらわされるデータのことです。バイナリ型として設定する際に利用するものになります。UN は Unsigned のことで、負の数を表さない符号のない長い整数として利用する際に設定するものです。マイナスの数値を設定できなくすることができます。ZF は Zero to Fill の略称で、デフォルトでは数値に 0 を設定しておきたい場合に利用します。AI は Autoincrement の略で、自動で数値を増分で設定することができます。id をプライマリーキーにした際に、増分で自動で採番されるようにすることができます。G は Generated　生成の略称で、実カラムの値の計算結果を収納するための特殊なカラムとして設定することができます。

また、これらはすべて、SQL 句のカラム名の後ろにオプションとして入力することで設定するこ

とが可能になっています。

それでは次に INSERT を利用してこちらのテーブルにデータを挿入していきます。

INSERT INTO users (id, name, email, age) VALUES(1,

'Smith','smith@hotmail.com', 58);

と入力して、実行します。

実行結果を見ていきます。

SELECT * FROM users;

を実行して、中身を確認します。

先ほどのデータが無事登録されていることがわかります。

それでは、次にプライマリーキーを設定しないで登録したらどうなるでしょうか？

プライマリーキーは必須項目ですので、データ登録でエラーとなってしまうのでしょうか？

```
INSERT INTO users (name, email, age)
VALUES('Akira','akira@hotmail.com', 24);
```

と入力して、実行します。

データは登録されますが、Action Output には黄色いマークが表示されていて、プライマリーキーが入力されていなかったことが示されています。

実際に値を確認してみましょう。

```
SELECT * FROM users;
```

を実行して、中身を確認します。

すると、id に 0 が入力されていることがわかります。勝手に 0 を代入してくれたようです。

それでは次に同じように、ID を空白にして別のデータを登録してみると、どうするのかを確認してみましょう。同じ番号を重複して登録できないため、0 は利用できないはずです。

```
INSERT INTO users (name, email, age) VALUES('Lily','
lily@hotmail.com', 30);
```

と入力して、実行します。

そうすると、0 が重複していますというエラーメッセージとともに、エラーが発生してしまいました。プライマリーキーを空白で登録していくと、0 が代入されるようになっていますが、常に 0 を代入しようとするため、2 回目はエラーになってしまいます。これが基本的なプライマリーキーのデータ登録上の制約となります。

一旦、先ほど作成した users テーブルを削除します。

```
DROP TABLE users;
```

と入力して、実行します。

再び users テーブルを作成していきます。先ほどと設定方法を変更してみます。

```
CREATE TABLE users
 (
     id INT AUTO_INCREMENT PRIMARY KEY,
     name VARCHAR(255),
     email VARCHAR(255),
     age INT(3)
);
```

と入力して設定します。プライマリーキーの設定位置がかわっていることと、

auto_increment が設定されていることがわかります。

再び同じデータをいれていきます。3 つとも実行してみましょう。

```
INSERT INTO users (id, name, email, age) VALUES(1,
'Smith','smith@hotmail.com', 58);
```

```
INSERT INTO users (name, email, age) VALUES('Lily','
lily@hotmail.com', 30);

INSERT INTO users (name, email, age)
VALUES('Akira','akira@hotmail.com', 24);
```

エラーや黄色いマークが発生することなく、実行できました。

実際に値を確認してみましょう。

```
SELECT * FROM users;
```

を実行して、中身を確認します。

ID が自動で採番されて入力されていることがわかりました。auto_increment を設定することで、ID が自動的に増加しながら設定されることになるわけです。

このようにプライマリーキーを設定する際には、様々な制約が付与されるため、

auto_increment を設定して自動で採番する場合は、より詳細な採番ルールが必要な

場合などに合わせて、ID の付与の仕方まで考慮して利用していくことになります。

■外部キーの設定②

それでは Workbench を開いて外部キーの設定をしていきましょう。

実際に先ほどのテーブルを作成してみましょう。

先ほどのコマンドを入力していきます。

```
CREATE TABLE orders
 (
     order_id INT(10),
     user_id INT,
     amount INT(20),
     price INT(20),
   PRIMARY KEY (order_id),
   FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

と入力して、実行します。

テーブルが完成しましたら、実際に外部キーの設定が上手くいっているのかを先ほどのプライマリーキーを設定と同じように確認していきましょう。

左タブのナビゲーターからスキーマタブにいっていただいてデータベースを指定できるようにした上で、先ほど作成した orders テーブルを開いいてクリックして指定してください。

インフォメーションマークをクリックして、テーブルの設定状況を確認します。

DDL タブを確認すると、プライマリーキーと外部キーの SQL クエリ上の設定状況が確認できます。

FOREIGN KEYs タブを開いてみると、user_id が外部キーとして設定されていることがわかります。その外部キーがどこのテーブルのどのカラムを参照しているのかまで確認することができます。

ここでいうと、oders テーブルの user_id と user テーブルの user_id が紐付いていることがわかります。

INDEX タブを開いてみると、プライマリーキーが INDEX として設定されているため、

order_id が INDEX となっていることがわかります。そして、もう１つ外部キーとして設定さ

れた user_id も INDEX として登録されています。

次に INSERT を利用してデータを挿入していきます。

```
INSERT INTO orders (order_id, user_id, amount, price)
VALUES(1, 1,2, 1100);
```

と入力して、実行します。

実行結果を見ていきましょう。

```
SELECT * FROM orders;
```

を実行して、中身を確認します。

先ほどのデータが無事登録されていることがわかります。

これで外部キーが設定されていることで、users テーブルの id が 1 のユーザーの注文内容として関連付けられた注文データとして登録されることになります。

それではもう 1 つのテーブルを作成して、外部キーの設定をしていきたいと思います。

さきほどの ER 図にはありませんでしたが、追加で製品情報を登録する、Products テーブルを作成します。

```
CREATE TABLE products
(
    id INT(10),
    category VARCHAR(255),
    price INT(20),
    PRIMARY KEY (id)
);
```

で作成していきます。

そして、ここで作成された ID を orders テーブル内の product_id に対する外部キーとして

設定していきたいと思います。

ここで問題となるのは既に orders テーブルが作成されてしまっているということです。

また、orders テーブルには products_id のカラムが存在しません。

そこで後から変更して設定する方法を実施していきたいと思います。

まずが外部キーとして利用したい product_id を orders テーブルに作成します。

```
ALTER TABLE orders

ADD COLUMN product_id INT AFTER user_id;
```

と入力して、実行します。

```
SELECT * FROM orders;
```

を実行して、結果を確認してみましょう。

このように product_id が追加されていることがわかります。

既にデータが入っている行については NULL 値が入っていることがわかります。

それではこの product_id に対して、外部キーを設定します。

ALTER TABLE orders

ADD FOREIGN KEY (product_id) REFERENCES products(id) ON DELETE

CASCADE;

と入力して、実行します。

これによって、外部キーとして設定ができましたので、再度確認していきたいと思います。

左タグからナビゲーター画面にいってデータベースを指定してあげます。その前にリフレッシュします。

ordersテーブルにいっていただいて、インフォメーションマークをクリックします。

foreign key タグを開いていただくと外部キーの設定情報を確認することができます。このようにproduct_id が外部キーとして追加されて、2つの外部キーが設定されていることがわかります。

ALTERコマンドを利用することで、あとからテーブル内容を変更することができるわけです。

ALTER の後に ADD コマンドを実行してデータを追加する操作ができます。

■外部キーの設定③

それでは、この ALTER コマンドを利用して、この外部キーの設定を削除してみましょう。

```
ALTER TABLE orders

DROP FOREIGN KEY product_id;
```

と入力して、実行してみます。

しかしながら、これはエラーとなってしまいます。

Action Output には「キーが存在しているか確認してください」

「キーが存在しているか確認してください」

とのエラーメッセージが表示されています。実は外部キーとして指定されたキーは外部キー独自のキー名称が付与されてしまい、その名称を指定する必要があるのです。

再び、orders テーブルのインフォメーションにいってください。

foreign key タグを開いて外部キー名称を確認すると、

orders_ibfk_1

orders_ibfk_2

という名称で外部キーが設定されています。したがって、外部キー名称はこちらを利用することが必要となります。Product_id に紐づいている外部キーは orders_ibfk_2 の方ですので、こちらを利用して、削除コマンドを実行します。

ALTER TABLE orders

DROP FOREIGN KEY orders_ibfk_2;

と入力して、実行します。

これで実行が上手くいきました。

再度、foreign key タグを開いていただくと外部キーの設定情報を確認してみましょう。

orders_ibfk_2 がなくなっているため、上手く削除されていることがわかります。

プライマリーキーであれば、作成後に Workbench 内のテーブル設定において、プライマリーキーとして後から設定することが可能でした。同じく外部キーも設定することができます。

orders テーブルにいっていただいて、レンチマークをクリックして設定画面を開きます。

中央の画面の下に 6 つのタグがならんでいることがわかります。Columns、 index、 foreign key 、triggers 、 partitioning 、opitions の 6 つです。このタグからそれぞれ細かい設定をテーブルに行うことができます。foreign key を開いていただくと、外部キーを設定することができる画面が開きます。

この画面で、FOREIGN キーネームで、先ほど削除した orders_ibfk_2 を入力して、

reference table において、pruducts テーブルを参照先として指定します。

右の画面にうつっていただいて、

キーとして指定するカラムとして product_id にチェックをいれて、Reference カラムとして

Product テーブルの ID を指定します。

次に右の Foreing Key Opition において、ON DELETE と ON PUDATE の両方に

CASCADE 設定を実施することができます。

先ほどの SQL クエリと同じ設定とするためには、

このように設定してください、

入力が完了したら下にある APPLY をクリックしてください。

すると SQL スクリプトの確認画面がでますので、こちらで実行内容を確認して、

さらに APPLY をクリックしてください。

ふたたび、foreign key タグを開いていただくと外部キーの設定情報を確認することができます。

orders_　　ibfk_2 が再び追加されていることがわかります。

このようにして、外部キーを追加したり削除したりといった操作があとからでも実施することができるわけです。

最後に、ON DELETE CASCADE の効果を実際に確認してみたいと思います。 ON DELETE CASCADE を指定することで、親テーブルが削除や更新された際に、子テーブルで外部キーとして利用されているカラムでも削除や更新がされる設定です。

まずはデータを設定していきます。

製品情報を登録していきましょう。

INSERT INTO products (id, category, price)

VALUES(1, "type1", 1000), (2, "type2", 2000), (3, "type3", 3000);

と入力して、実行して、3つの製品を登録していきます。

SELECT * FROM products;

で登録状況を確認してみます。

次に注文情報を登録していきましょう。

INSERT INTO orders (order_id, user_id, product_id, amount, price)

VALUES(1, 1, 1, 2, 2000), (2, 1, 2, 2, 4000), (3, 3, 1, 3, 9000);

と入力して、実行して、3つの注文を登録していきます。

SELECT * FROM orders;

で登録状況を確認してみます。

外部キーとして参照されている親テーブル内のキーとなっている ID を変更してみましょう。

UPDATE users SET id = 5 WHERE id = 1;

と入力して、実行します。

これによって、users テーブル内のユーザーID が 1 番のユーザーの ID を 5 番に変更します。

SELECT * FROM users;

で ID が 5 に変更されていることが確認できます。

次に、この user_id を外部キーに指定して参照している Orders テーブルないで変更がどのように影響しているかを確認します。

SELECT * FROM orders;

でリストを取得すると、user_id 5 のオーダーに変更されています。

次に親テーブル側のデータを削除した場合にどうなるかを確認してみます。

先ほど変更した id 5 のデータを削除します。

DELETE FROM users WHERE id = 5;

と入力して、実行することで ID 5 を指定して users テーブルから該当するデータを削除することができます。

SELECT * FROM users;

で確認してみると、削除されていることがわかります。

次に注文内容を確認すると

SELECT * FROM orders;

このように user_id の 5 を参照していた注文データも削除されていることがわかります。

このように ON DELETE CASCADE を指定することで、親テーブルが削除や更新された際に、子テーブルで外部キーとして利用されているカラムでも削除や更新が反映されることにな

ります。これは外部参照をしているテーブル同士の整合性を担保するために利用される機能となります。

他の設定としては、

ON UPDATE RESTRICT

ON DELETE RESTRICT

を利用することで、参照されている親データのデータを削除や変更されることができなくなる設定も可能です。途中で変えられないようにすることで整合性を担保することになります。

ON UPDATE SET NULL

ON DELETE SET NULL

とどうなるのかも確認していきましょう。

テーブルの設定を、Workbench のテーブルを指定して、設定タグに行っていただき、外部キーの設定を変更します。

それぞれ ON UPDATE SET NULL と ON DELETE SET NULL に設定を変更して

APPLY をおして、設定変更を進めてください。

それでは SETNULL の場合の動作を確認していきます。

現在 user_id が 3 番の注文データが残っていますので、

この 3 番のデータを変更してみます。

3 番を 4 番にかえてみましょう。

UPDATE users SET id = 4 WHERE id =3 ;

と入力して、実行することで変更されます・

SELECT * FROM users;

で ID が 4 に変更されていることが確認できます。

次に、この user_id を外部キーに指定して参照している Orders テーブルで変更がどのように影響しているかを確認します。

SELECT * FROM orders;

でリストを取得すると、3 番の注文であった注文内容にある user_id が null になっていることがわかります。SET NULL とすることで、親テーブル側のデータが変更や削除されると、子テーブル側では null 値に変更されることがわかりました。

外部キーの親テーブルを削除してみましょう。

DROP TABLE products;

と入力して、実行します。

この実行はエラーとなってしまいます。

を実行すると、外部キーの制約によって削除できないというエラーメッセージが表示されています。

このように参照元となっているテーブルが削除ができないことになります。

次に参照先となっている orders テーブルを削除してみます。

DROP TABLE orders;

と入力して、実行します。

これは実行がうまくいって、削除されます。参照先のテーブルについては上手く削除ができます。

つぎに products テーブルを再び削除してみます。

DROP TABLE products;

と入力して、実行します。

すると今回は削除がうまくいきます。このように外部キー制約がある場合は、

このように外部キーの利用には様々な制約があり、それをうまくコントロールして設定することが必要となります。ON DELETE CASCADE など複雑なデータ設計が必要ですが、上手く活用して関連性と整合性を担保するようにしてください。

最後に、データをいろいろと変更してしまったため、残った users テーブルも削除してテーブルをきれいにします。

DROP TABLE users;

で実行します。

以上で外部キーの設定のレクチャーを終了します。

■ユニークキーの設定

改めて users テーブルを設定していきます。

その中で email をユニークキーに設定します。

```
CREATE TABLE users
 (
     id INT(3) AUTO_INCREMENT,
     name VARCHAR(255),
     email VARCHAR(255),
     age INT(3),
     PRIMARY KEY (id),
     UNIQUE KEY (email)
);
```

と入力します。ユニークキーはプライマリーキーと同じように設定して、実行します。

上手く登録できました。

それでは、データを入力していきます。

```
INSERT INTO users (id, name, email, age)
VALUES(1, 'Smith','smith@hotmail.com', 58),
(2,'Akira','akira@hotmail.com', 24), (3,'Lily',' lily@hotmail.com', 30);
```

と入力して、実行して 3 人分のデータを登録します。

それでは登録結果を確認しましょう。

```
SELECT * FROM users;
```

で登録できたことが確認できました。

ではユニークキーの登録状況をインフォメーションから確認していきます。

左タブで ORDER スキーマ内の users テーブルを指定して、インフォメーションマークをクリックしてください。そこで DDL タブを見てみると、ユニークキーが設定されていることがわかります。

次に INDEX タブをクリックしてみてみると、INDEX として設定されていることがわかります。ユニークキーとなるということは検索キーとして利用できるようになるため、プライマリーキーと同じで INDEX に自動で登録されることになります。

つぎにユニークキーの動作を確認するため、重複した EMAIL アドレスを登録してみます。

```
INSERT INTO users (id, name, email, age)
VALUES(4, 'Smith','smith@hotmail.com', 34);
```

と同じスミスさんのデータを同じメールアドレスで登録してみます。

実行してみると、エラーとなり、

Acution Output にはメールアドレスが重複していますというメッセージが表示されていることがわかります。このようにユニークキーが設定されていると、重複したデータを登録することができなくなってしまいます。

それでは、このユニークキーを削除すると、先ほどのデータが登録されるのかを確認してみましょう。

ユニークキーを削除する際は INDEX として削除してあげます。

ALTER TABLE users DROP INDEX email;

と入力して実行します。

もう一度インフォメーションから登録情報を確認していきます。

インフォメーションマークをクリックしてください。そこで DDL タブを見てみると、ユニークキーが設定されていないことがわかります。

次に INDEX タブをクリックしてみてみると、INDEX として email が設定されていないことがわかります。このようにユニークキーを削除する際は INDEX として削除することが必要です。

先ほどのデータをもう一度登録してみましょう。

```
INSERT INTO users (id, name, email, age)
VALUES(4, 'Smith','smith@hotmail.com', 34);
```

今回はエラーにならずに実行されました。データを見てみます。

```
SELECT * FROM users;
```

で登録できたことが確認できました。Email が重複して登録されているため、ユニークキー制約がなくなっていることがわかります。このようにユニークキーは重複を防ぎたい場合に設定することができるものとなっています。

■ロールバックの活用

Workbench は初期設定ではデフォルトで、AutoCommit が設定されています。つまり

SQL クエリによって実行されるトランザクションは自動的にコミットされて、データベースのディス

ク側に反映される設定となっているわけです。

そこでこの AutoCommit を解除することで、ロールバックを手動で実行できるようにします。

SET autocommit=0;

と入力して、実行します。

これで AutoCommit がオフになります。

これでロールバックを手動で実行できるようになったため、操作を確認してみます。

Users テーブルの ID 1 を 5 に変更してみます。

UPDATE users SET id = 5 WHERE id =1 ;

と入力して、実行します。

それでは実行結果を確認します。

SELECT * FROM users;

すると、反映がされてしまっているようです。

次にロールバックを実施してみましょう。

ROLLBACK;

と入力して、実施します。

それでは実行結果を確認します。

SELECT * FROM users;

すると先ほどの変更が元に戻っています。したがって、ROLEBACK が成功したことがわかります。つまりコミット前ではありますが、テーブルデータを抽出すると変更は見かけ上は反映されているのですが、確定はしていないため、ロールバックを実施することができることになります。

もう一度先ほどの値を入力します。

UPDATE users SET id = 5 WHERE id =1 ;

そして次はコミットをしてみます。

COMMIT;

と入力して、実行します。

それでは実行結果を確認します。

SELECT * FROM users;

これでコミットされて先ほどの変更が確定することになります。ロールバックができないかを確認

してみましょう。

ROLLBACK;

と入力して、実施します。

それでは実行結果を確認します。

SELECT * FROM users;

このように変更が反映されたままとなっているため、ロールバックがされませんでした。一度コミットされたデータはロールバックができないからです。したがって、コミット句によって変更が確定されていることがわかります。

次に AutoCommit を解除しないでロールバックを利用する方法を実践してみます。

SET autocommit=1;

と入力して、実行します。

これで AutoCommit がオンになります。

START TRANSACTION;

コミットとロールバックを実施したい SQL クエリの前に実行することによって、そのクエリに対してのみ AutoCommit をオフにして処理を進めることができます。それでは実際にやってみましょう。

先ほど ID1 を ID5 に変更しているため、今度は元に戻してみましょう。

START TRANSACTION;

と入力して、実行してまずはトランザクションを設定します。

次に 5 を 1 に変更する処理を入力して、実行します。

UPDATE users SET id = 1 WHERE id =5 ;

それでは実行結果を確認します。

SELECT * FROM users;

すると変更が反映されているリストが表示されます。

ROLLBACK;

と入力して、実施します。

それでは実行結果を確認します。

SELECT * FROM users;

このようにロールバックが適用されて、処理が元に戻ったことがわかります。データベースの削除や変更は間違えると大きな問題が発生する可能性があります。AutoCommit をオフにするか、TRANSACTION を設定することで、ロールバックが適用すれば安全に処理を進めることができるようになります。

■INDEX の追加

先ほどのコマンドを実際に入力して実行していきます。

```
CREATE INDEX index_name
ON users(name);
```

と入力します。このコマンドは users テーブルの name カラムを INDEX として指定することになります。index_name という INDEX として設定します。

それでは実行します。実際に設定されたかをインフォメーションから確認してみましょう。

左タブで ORDER スキーマ内の users テーブルを指定して、インフォメーションマークをクリックしてください。そこで DDL タブを見てみると、KEY として index_name が設定されていることがわかります。

次に INDEX タブをクリックしてみてみると、ここでも index_name が INDEX として設定されていることがわかります。そして、カラムの name に index_name が設定されていることがわかります。

これで INDEX の設定が完了しました。

■NOT NULL の設定

再び以前作成した products テーブルを NOT NULL を設定して作っていきましょう。

```
CREATE TABLE products
(
    id INT(10),
    category VARCHAR(255) NOT NULL,
    price INT(20) NOT NULL,
    PRIMARY KEY (id)
);
```

と入力します。

プライマリーキーには NOT NULL はいれなくても、デフォルトで設定されているため、NOT NULL をオプションで設定してもしなくても大丈夫です。他のカラムも同様に NULL 値が入ることを防ぎたいと考えている場合は、オプションとして NOT NULL を設定することが必要です。外部キーやユニークキーに対しても設定が必要となりますので注意してください。

それでは実行してみましょう。

設定が上手くいったかを確認します。

左タブで ORDER スキーマ内の products テーブルを指定して、インフォメーションマークをクリックしてください。そこで DDL タブを見てみると、全てのカラムに NOT NULL が設定されていることがわかります。

次に columns タブをクリックしてみてみると、全てのカラムに NULLABLE が NO と設定されていることがわかります。

次に NOT NULL の動作を確認するため、データを NULL で設定してみたいと思います。

Category が NULL となるデータを挿入してみます。

INSERT INTO products (id, category, price)

VALUES(1, "", 1000);

と入力して、実行してみます。

実はこれは実行できてしまいます。""で入力されると登録されるデータは NULL ではなく、空文字データが入力されてしまうからです。

では次はどうでしょうか？

INSERT INTO products (id, category, price)

VALUES(2, , 1000);

これは SQL の文法違反となってしまい、実行もされません。最初に 3 つのカラムを指定したため、3 つのカラムに対応した VALUES は SQL クエリとして必須となるためです。

では、category だけ除外して設定すれば NULL になるのではないでしょうか？

INSERT INTO products (id, price)

VALUES(2, 1000);

これは実行されて、空文字が入っている状態と同じになります。これも NULL ではありません。また、デフォルト値が設定されていればデフォルト値が入ることになります。

では NULL と直接入力していきましょう。

INSERT INTO products (id, category, price)

VALUES(3, NULL, 1000);

これは NOT NULL 制約でエラーとなって登録されません。NULL を入力するには、NULL と直接記載することが必要なわけです。

小文字ではどうでしょうか？

INSERT INTO products (id, category, price)

VALUES(3, null, 1000);

これも NOT NULL 制約でエラーとなって登録されません。大文字も小文字も同じように NULL として扱ってくれるわけです。

では“”で閉じてみてはどうでしょうか？

```
INSERT INTO products (id, category, price)
VALUES(3, "null", 1000);
```

これは登録されてしまいました。この場合は null という文字データが登録されたことになります。このように意外と NULL になりづらかったりして、空文字データが登録されてしまうところが厄介なところではあります。したがって、次のレクチャーで実施するデフォルトの設定も重要となります。

さて、既に作成したテーブルに対して、後から NOT NULL 制約を付与するやり方をみていきます。

現在、Users テーブルはプライマリーキー以外には NOT NULL 制約をつけていないため、そのカラムに NOT NULL 制約を追加したいと思います。

ALTER コマンドで、MODIFY を利用します。

ALTER TABLE users

MODIFY email VARCHAR(255) NOT NULL;

と入力して、実行してみます。

左タブで ORDER スキーマ内の users テーブルを指定して、インフォメーションマークをクリックしてく、NOT NULL 制約の有無を確認してみてください。

Email に NOT NULL 制約が付与されたことが確認できます。

また、ALTER コマンドで **CHANGE COLUMN** を利用しても、変更することができます。

列名も含めたカラムの構造（列名、データ型、制約）をすべて変更する際は、**CHANGE COLU**MN を利用して、列名はそのままにデータ型や制約を変更したい場合は MODIFY を利用するといった使い訳をしていきます。したがって、CHANGE COLUMN を利用する際は次のようにまとめて変更する際に使っていきます。

今度は age に対して設定をしていきます。

ALTER TABLE users

CHANGE COLUMN age age2 INT(3) NOT NULL;

これを実行すると、age が age2 に変更して、データ型と NOT NULL 制約を再設定してくれます。現在のカラム名と次のカラム名を併記することが必要です。

実行してみます。

左タブで ORDER スキーマ内の users テーブルを指定して、インフォメーションマークをクリックしてく、NOT NULL 制約の有無を確認してみてください。

Age が age2 に変更されて、NOT NULL 制約も付与されていることがわかります。

このようにして NOT NULL 制約を付与したり、カラムデータを修正したりといった操作が可能となるわけです。

■デフォルト値の設定

一旦既存のテーブルを削除してから、改めてデフォルト値を設定していきたいと思います。

DROP TABLE products;

で先ほど作成したテーブルを削除します。

再度テーブルを作成します。

```
CREATE TABLE products
(
    id INT(10),
    category VARCHAR(255) NOT NULL DEFAULT 'Not Categorized',
    price INT(20) NOT NULL DEFAULT 0,
    PRIMARY KEY (id)
);
```

と入力してください。このコマンドでは category のデフォルト値として、Not Categorized'という文字データが自動で入力されるようにします。Price のデフォルト値としては、0 をデフォルト値を設定します。それでは実行してみてください。

設定内容を確認します。

左タブで ORDER スキーマ内の products テーブルを指定して、インフォメーションマークをクリックしてください。そこで DDL タブを見てみると、全てのカラムにデフォルト値が設定されていることがわかります。

次に columns タブをクリックしてみてみると、Default Value という個所でデフォルト値の設定状況をカラムごとに確認することができます。category のデフォルト値として、Not Categorized'、Price のデフォルト値としては、0 が設定されていることが確認できました。

それでは実際にデータを挿入して、動作を確認してみます。

INSERT INTO products (id)

VALUES(1);

と入力して、id 以外の設定をしないまま実行してみます。

登録したデータを確認してみましょう。

SELECT * FROM products;

と入力して、実行しますと、先ほど登録していなかったカテゴリーとプライスのカラムにデフォルト値が設定されていることがわかります。

このようにデータの登録が一部されない場合には、データを空白にしないでデフォルト値を設定することができます。