1806b 版

システム開発プロジェクト総合実践・補助資料

2020 年 5 月 18 日～5 月 22 日

（Java プログラミング v1.1）

Microsoft CERTIFIED Solutions Developer
Web Applications
Windows Store Apps Using C#

Microsoft CERTIFIED Professional Developer
Windows Azure Developer
Windows Phone Developer
Windows® Developer 4
Enterprise Application Developer 3.5

2020 年 5 月 21 日

ニュートラル株式会社
林 広宣
http://www.neut.co.jp/
hayashi.hironori@neut.co.jp

（このページが、表紙の裏になるように印刷をお願いします。）

ご意見・ご質問は下記まで

ニュートラル株式会社　林広宣
hayashi.hironori@neut.co.jp

目　　次

（空白ページ）

第1章. はじめに

1.1. 課題の説明

1.1.1.概要

フライトチケットの予約をシミュレーションするプログラムを作成する。

1.1.2.開発の背景

我々は日本のある航空会社のシステム部門に配属された新入社員である。
部門の大きな使命は航空全般の安全かつ効率的な運用および管理である。

そこで、我々新入社員には、仕事の内容を早く理解してもらうため、簡単なフライトチケット予約システムの作成という課題が与えられた。

このシステムは本番データではなく、サンプルデータを使い、機能も単純なものでよいということである。
ただし、プログラミング方法と航空券の予約の仕組みの理解が深まるようなアプリケーションを作成することが課題の目的である。

1.1.3.目標

できる範囲までの機能を実装すればよい。

例えば、予約機能まで作成できなくとも、「日付と空港コードを入力すると、その日に発着するフライトの一覧を表示する」というような部分的な機能が、正しく動作すればよい。

1.1.4.制約

1. 空港

国内線の予約のみとし、次の 7 つの空港の便に限定する。

1. . 札幌・新千歳
2. . 福岡
3. . 東京・羽田
4. . 大阪・伊丹
5. . 大阪・関西
6. . 名古屋・中部
7. . 成田

2. 期間

予約できる便の期間は 2020/5/15 から 2020/6/30 までとする。

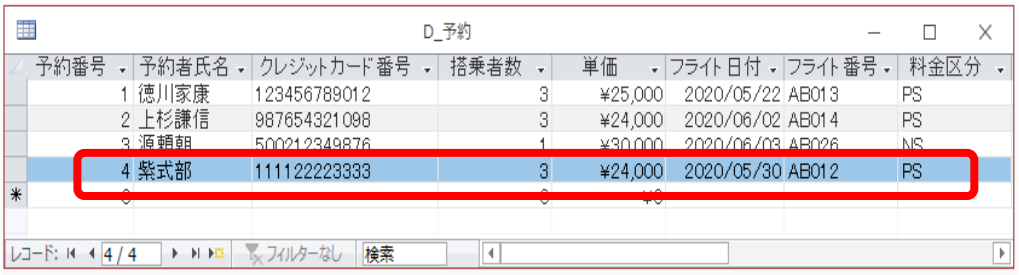
1.1.5.一例

あくまで例であるが、下記のような機能を持ったアプリケーション。

```
予約したい日付の「月」を入力してください。(1--12, END=999) >5
予約したい日付の「日」を入力してください。(1--31, END=999) >30

出発する空港を選んでください。
  1..札幌・新千歳
  2..福岡
  3..東京・羽田
  4..大阪・伊丹
  5..大阪・関西
  6..名古屋・中部
  7..成田

番号は？(END=999)>2

==========================================
出発地は、FUK-福岡  です。

目的地の空港を選んでください。
  1..札幌・新千歳
  2..福岡
  3..東京・羽田
  4..大阪・伊丹
  5..大阪・関西
  6..名古屋・中部
  7..成田

番号は？(END=999)>6

==========================================
目的地は、NGO-名古屋・中部  です。

人数は何人ですか？
1..1 人  2..2 人  3..3 人  4..4 人  5..5 人
番号は？ (1--5, END=999) >3
```

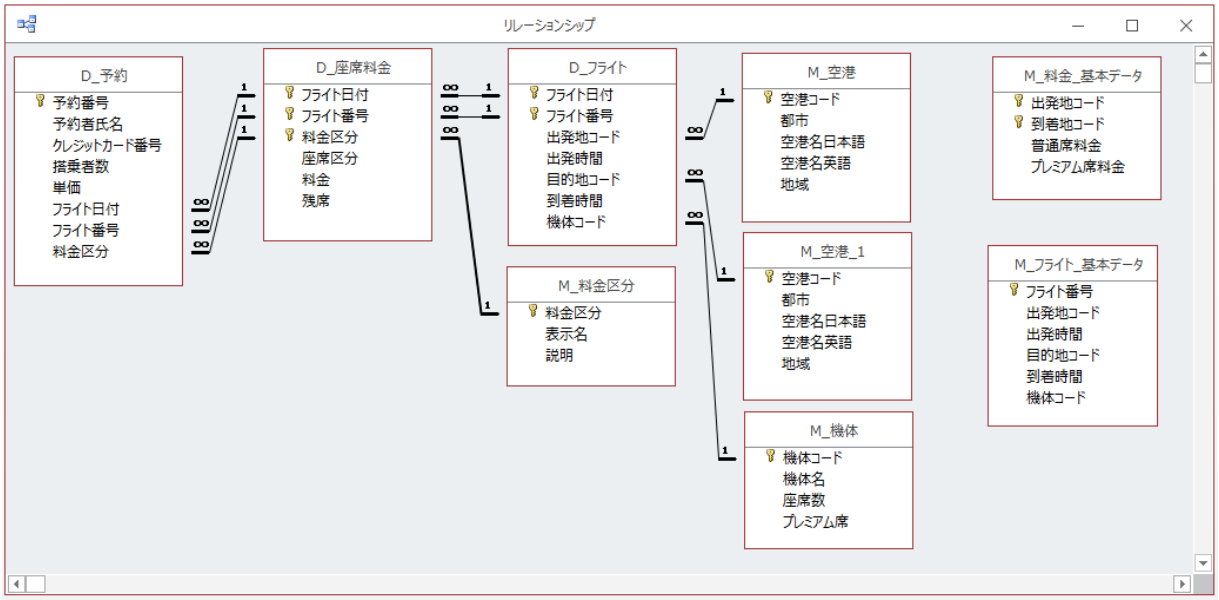
```
==========================================

どの便を利用しますか？
  1..AB012 09:10-10:40
  2..AB014 21:10-22:40

番号は？（END=999）>1
==========================================
選択したフライトは、AB012(2020-05-30)　福岡　09:10　名古屋・中部　10:40 です。

どのクラスを利用しますか？
  1..スーパーバリュー　￥11200/人　(3 名 ￥33600)
  2..普通席　￥16000/人　(3 名 ￥48000)
  3..プレミアム　￥24000/人　(3 名 ￥72000)

番号は？（1--5, END=999）>3
空席を確認中です...

予約可能です。
クレジットカードの番号を入力してください。(END=999)>111122223333

予約者の氏名を入力してください。(END=999)>紫式部

予約できました。

下記の予約番号で搭乗券と引き換えてください。
予約番号: 4
予約者氏名: 紫式部
予約情報
    AB012 便　2020 年 5 月 30 日　福岡 09:10 発 - 名古屋・中部 10:40 着
    プレミアム 3 名 ￥72000

終了します。
続行するには何かキーを押してください . . .
```

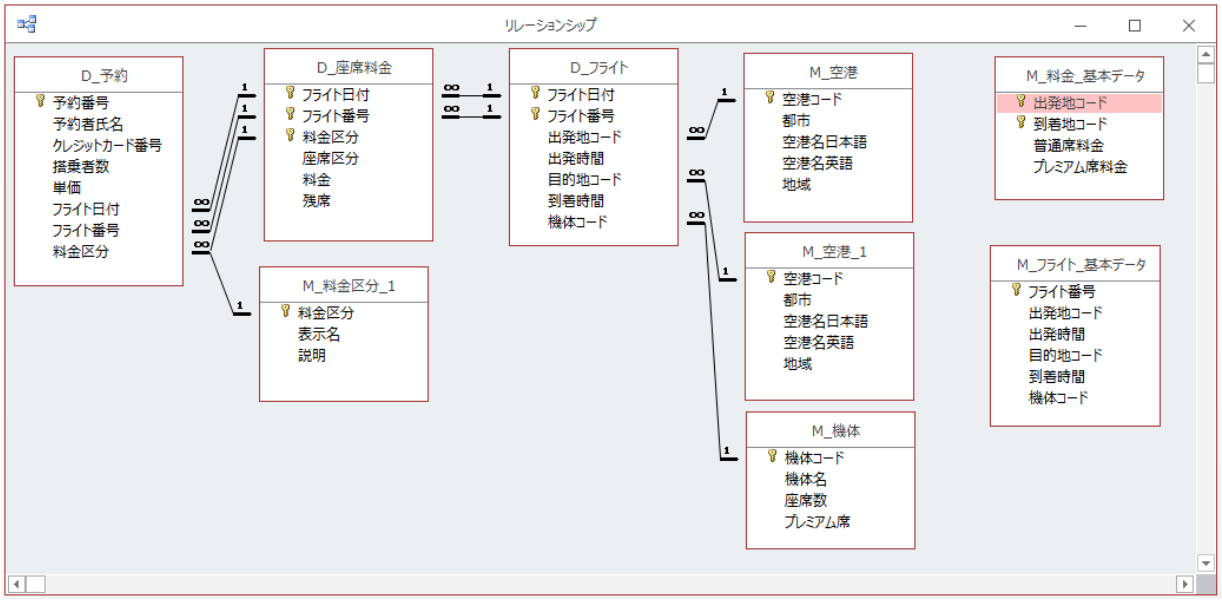
この結果、データベース（flight.accdb）の［D_予約］というテーブルにデータが１件追加される。（下図参照）



第2章. データベース構造

2.1. データベース全体の構造





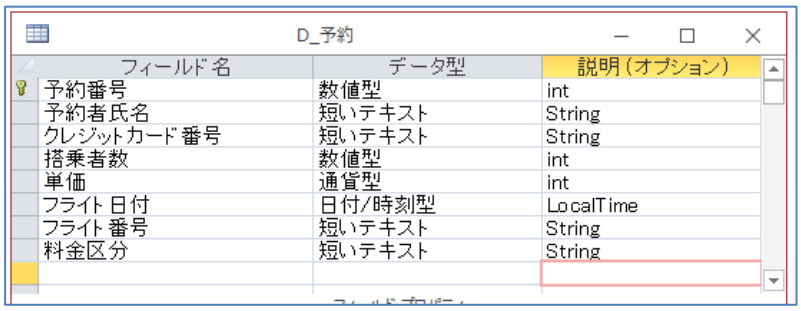
［M_料金_基本データ］と［M_フライト_基本データ］は、実習では使用しない。
これらは、［D_座席料金］や［D_フライト］を生成するためのものである。

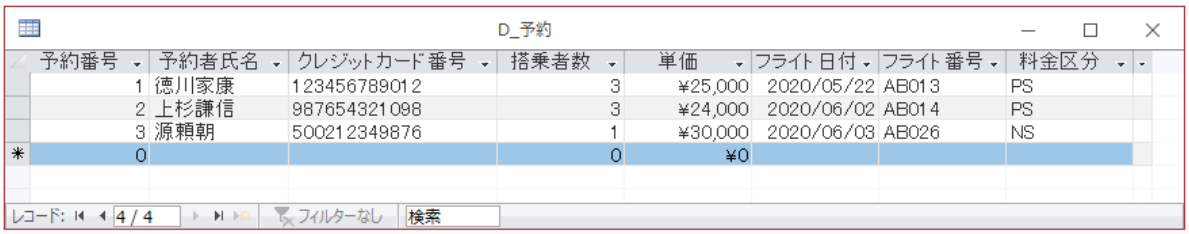
2. 2. 各テーブルの構造

2.2.1.D_予約

今回作成するアプリケーションが最終的に生成するデータがこのデータとなる。

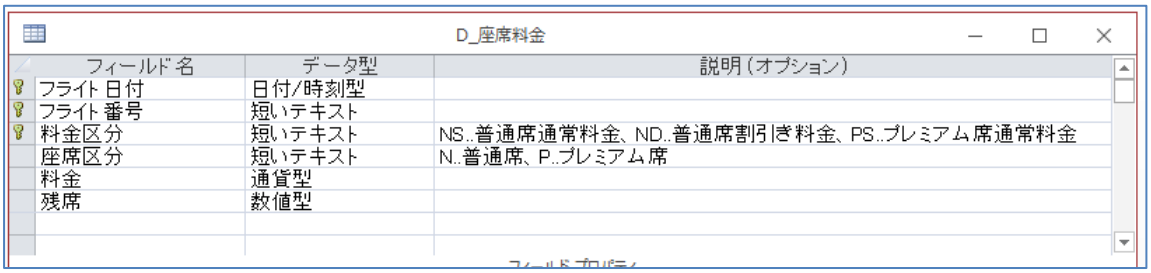
「誰がいつのどのフライトを予約し、人数何人でいくらだったか」というような情報を格納する。





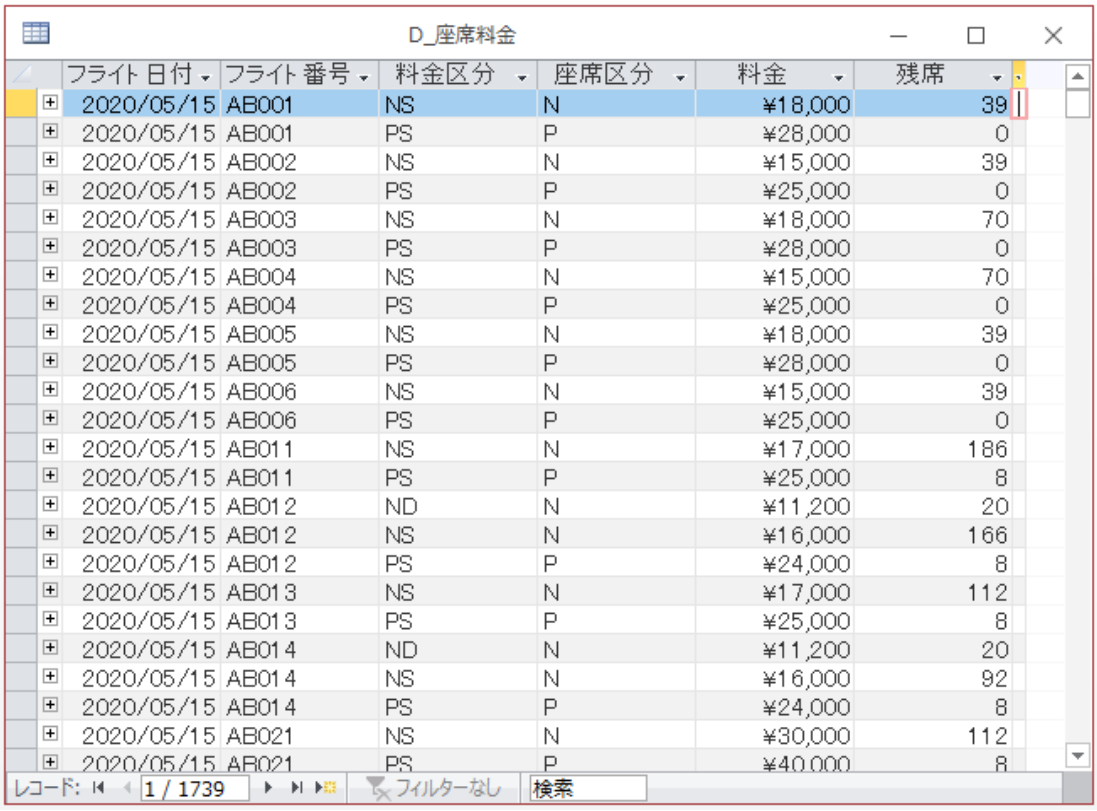
2.2.2.D_座席料金

予約したフライトの料金と残席数を格納しているテーブルである。



D_座席料金

フィールド名	データ型	説明（オプション）
フライト日付	日付/時刻型	
フライト番号	短いテキスト	
料金区分	短いテキスト	NS..普通席通常料金、ND..普通席割引き料金、PS..プレミアム席通常料金
座席区分	短いテキスト	N..普通席、P..プレミアム席
料金	通貨型	
残席	数値型	



D_座席料金

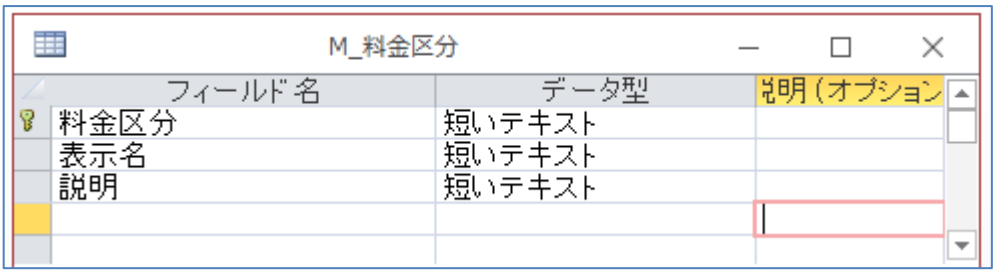
フライト日付	フライト番号	料金区分	座席区分	料金	残席
2020/05/15	AB001	NS	N	¥18,000	39
2020/05/15	AB001	PS	P	¥28,000	0
2020/05/15	AB002	NS	N	¥15,000	39
2020/05/15	AB002	PS	P	¥25,000	0
2020/05/15	AB003	NS	N	¥18,000	70
2020/05/15	AB003	PS	P	¥28,000	0
2020/05/15	AB004	NS	N	¥15,000	70
2020/05/15	AB004	PS	P	¥25,000	0
2020/05/15	AB005	NS	N	¥18,000	39
2020/05/15	AB005	PS	P	¥28,000	0
2020/05/15	AB006	NS	N	¥15,000	39
2020/05/15	AB006	PS	P	¥25,000	0
2020/05/15	AB011	NS	N	¥17,000	186
2020/05/15	AB011	PS	P	¥25,000	8
2020/05/15	AB012	ND	N	¥11,200	20
2020/05/15	AB012	NS	N	¥16,000	166
2020/05/15	AB012	PS	P	¥24,000	8
2020/05/15	AB013	NS	N	¥17,000	112
2020/05/15	AB013	PS	P	¥25,000	8
2020/05/15	AB014	ND	N	¥11,200	20
2020/05/15	AB014	NS	N	¥16,000	92
2020/05/15	AB014	PS	P	¥24,000	8
2020/05/15	AB021	NS	N	¥30,000	112
2020/05/15	AB021	PS	P	¥40,000	8

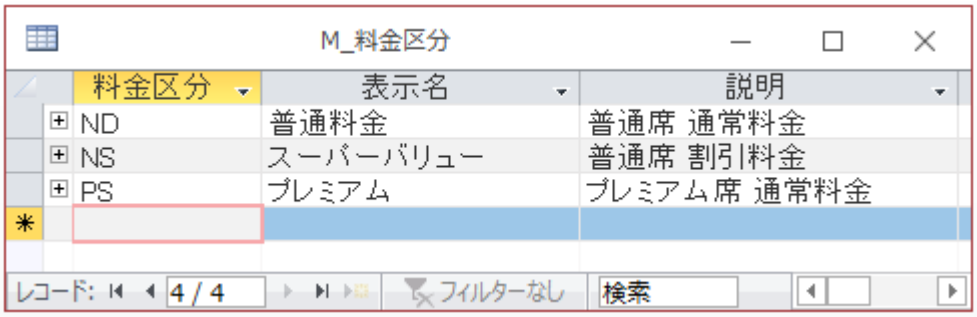
レコード: 1 / 1739　フィルターなし　検索

2.2.3.M_料金区分

座席によって料金が異なる。その種類を定義したテーブルである。

金額はフライトと日付によって異なるので、前述の D_座席料金テーブルに定義されている。





2.2.4.D_フライト

フライトとは、フライト番号と日付が主キーとなっているテーブルである。

出発地と目的地、出発時間と到着時間、機体の種類が格納されている。

D_フライト

	フィールド名	データ型	説明（オプション）
🔑	フライト日付	日付/時刻型	
🔑	フライト番号	短いテキスト	
	出発地コード	短いテキスト	[M_空港]の[空港コード]
	出発時間	日付/時刻型	
	目的地コード	短いテキスト	[M_空港]の[空港コード]
	到着時間	日付/時刻型	
	機体コード	短いテキスト	[M_機体]の[機体コード]

D_フライト

フライト日付	フライト番号	出発地コー	出発時間	目的地コー	到着時間	機体コード
2020/05/15	3001	NGO	6:00:00	NRT	7:00:00	Q82
2020/05/15	AB002	NRT	9:00:00	NGO	10:00:00	Q82
2020/05/15	AB003	NGO	12:00:00	NRT	13:00:00	CR7
2020/05/15	AB004	NRT	15:00:00	NGO	16:00:00	CR7
2020/05/15	AB005	NGO	18:00:00	NRT	19:00:00	Q82
2020/05/15	AB006	NRT	21:00:00	NGO	22:00:00	Q82
2020/05/15	AB011	NGO	6:10:00	FUK	7:40:00	321
2020/05/15	AB012	FUK	9:10:00	NGO	10:40:00	321
2020/05/15	AB013	NGO	18:10:00	FUK	19:40:00	73P
2020/05/15	AB014	FUK	21:10:00	NGO	22:40:00	73P
2020/05/15	AB021	NRT	6:00:00	FUK	8:00:00	73P
2020/05/15	AB022	FUK	9:00:00	NRT	11:00:00	73P
2020/05/15	AB023	NRT	12:00:00	FUK	14:00:00	CR7
2020/05/15	AB024	FUK	15:00:00	NRT	17:00:00	CR7
2020/05/15	AB025	NRT	18:00:00	FUK	20:00:00	789
2020/05/15	AB026	FUK	21:00:00	NRT	23:00:00	789
2020/05/16	AB001	NGO	6:00:00	NRT	7:00:00	Q82
2020/05/16	AB002	NRT	9:00:00	NGO	10:00:00	Q82
2020/05/16	AB003	NGO	12:00:00	NRT	13:00:00	CR7
2020/05/16	AB004	NRT	15:00:00	NGO	16:00:00	CR7
2020/05/16	AB005	NGO	18:00:00	NRT	19:00:00	Q82
2020/05/16	AB006	NRT	21:00:00	NGO	22:00:00	Q82
2020/05/16	AB011	NGO	6:10:00	FUK	7:40:00	321
2020/05/16	AB012	FUK	9:10:00	NGO	10:40:00	321

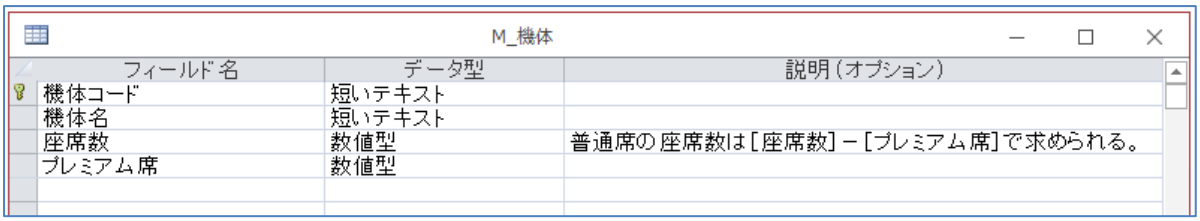
レコード: 1 / 752　フィルターなし　検索

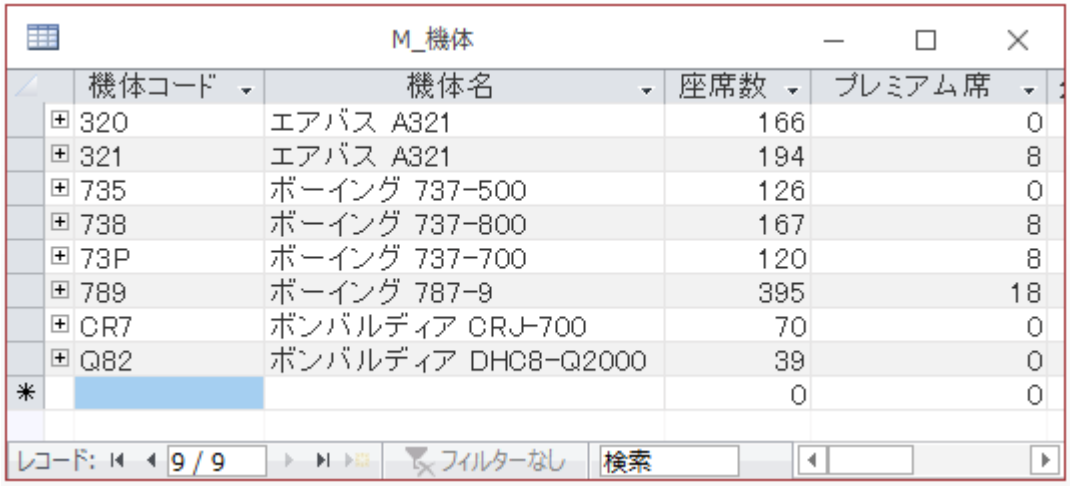
2.2.5.M_空港

空港を表すテーブル。

2.2.6.M_機体

機体の種類（エアバス XXX、ボーイング XXX など）とその期待の座席数を格納したテーブル。



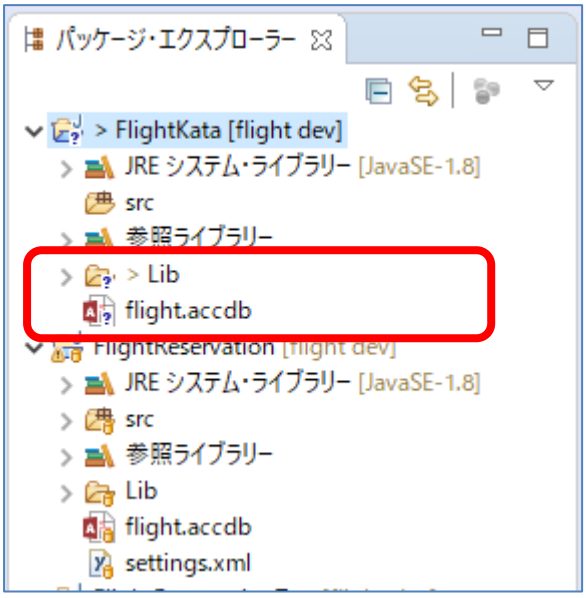


第3章. データベースプログラミング

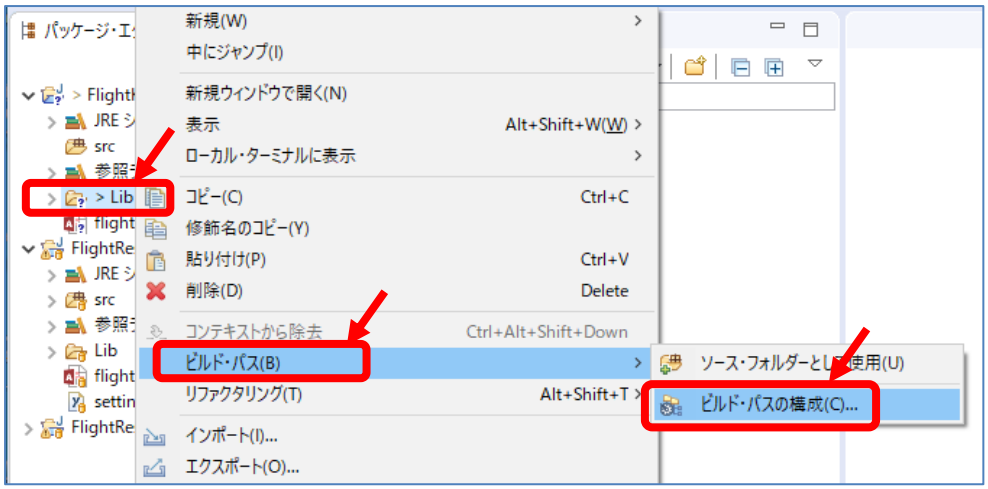
3.1. 事前準備

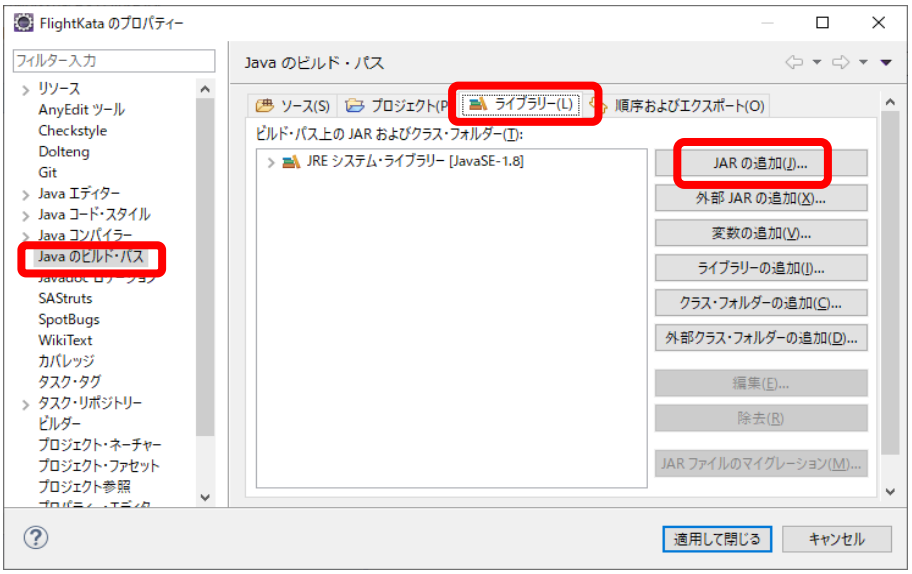
データベース（flight.accdb）を読み書きできるようにするための準備。

Eclipse でプロジェクトを作成し、プロジェクト直下に(pre_req 内の)Lib フォルダーをコピーする。
同様に、flight.accdb ファイルもコピーする。



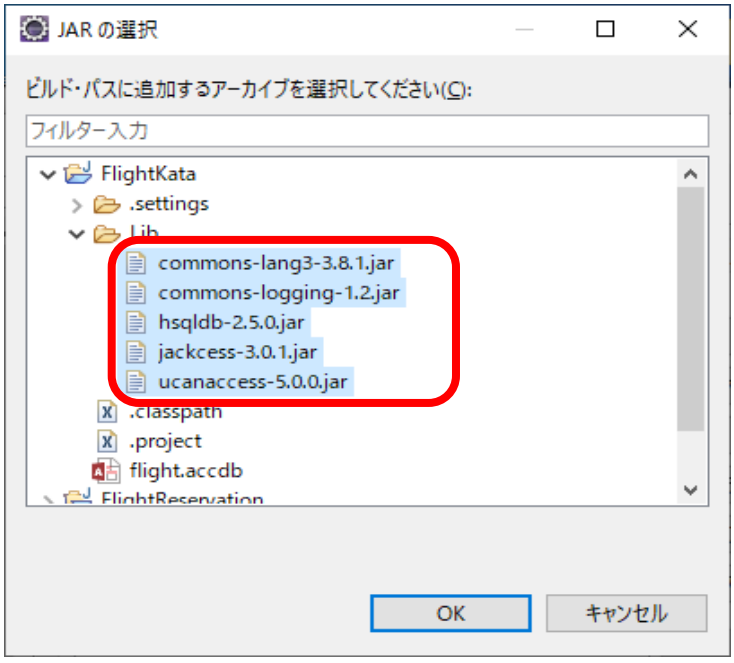
プロジェクトの Lib フォルダーを右クリックし、［ビルド・パス］－［ビルド・パスの構成］とたどっていく。



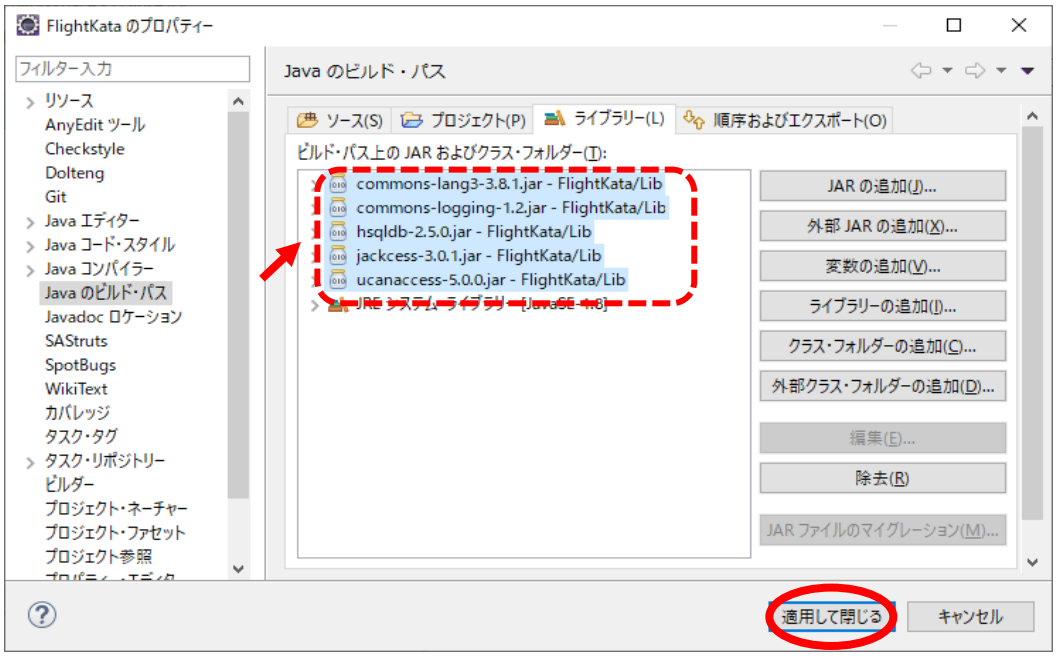


［Shift］キーを押しながら、Lib フォルダー内の JAR ファイルをすべて選択し、［OK］ボタンをクリックする。

全て選ぶのではなく、1 つずつ選んで、この処理を繰り返し、すべて（5 つ）の JAR ファイルが追加されるようにしてもよい。

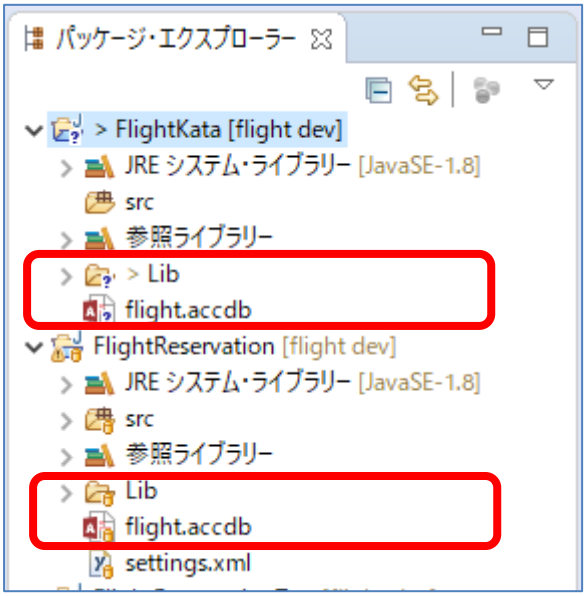


下図のように JAR ファイルが追加されていることを確認し、[適用して閉じる] ボタンをクリックする。



以上の作業は、プロジェクトを作成するたびごとに実施すること。

下図は２つのプロジェクト（FlightKata と FlightReservation）に対して、同じ作業が実施されたことを示している。



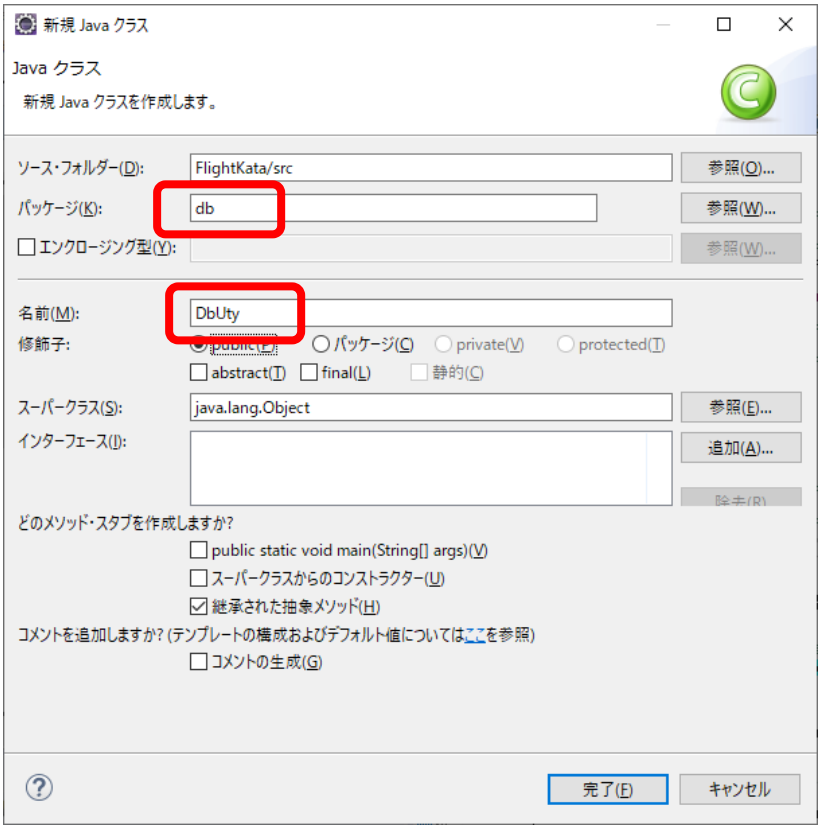
3.2. 簡単なプログラム

3.2.1.注意事項

1. パッケージを必ず指定しよう

クラスを作成するときは、必ずパッケージ名も指定して作成するようにしよう。
パッケージは、クラスの機能の役割分担だけでなく、チーム内のプログラマーの役割分担にも利用できる。

下記は、db.DBUty というクラスを Eclipse で作成する場合の例。



3.2.2.とにかくテーブルの内容を一覧表示する。

1. クラス main.Sample01

```
package main;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Sample01 {
    public static void main(String[] args) {

        try {
            String prefiex = "jdbc:ucanaccess://";
            String fname = "flight.accdb";
            // このプログラムが起動された場所（フォルダー）を取得する。
            String dir = System.getProperty("user.dir");
            //
            String url = prefiex + dir + "/" + fname;
            //
            // 上記までの結果、url は、例えば次のような文字列となる。
            //
            //      jdbc:ucanaccess://C:/Java/db/flight.accdb
            //
            //
            String sql_select = "SELECT * FROM M_機体";
            //
            Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql_select);
            while (rs.next()) {
                System.out.println(
                    String.format(
                        "%s %s %s",
                        rs.getString(1),
                        rs.getString(2),
                        rs.getString(3)
                    )
                );
            }
        }
        catch (SQLException e) {
            System.out.println("err: " + e.getMessage());
        }

    }

}
```

2. 実行結果

```
320 エアバス A321 166
321 エアバス A321 194
735 ボーイング 737-500 126
738 ボーイング 737-800 167
73P ボーイング 737-700 120
789 ボーイング 787-9 395
CR7 ボンバルディア CRJ-700 70
Q82 ボンバルディア DHC8-Q2000 39
```

3.2.3.何度も利用するコードは１カ所にまとめよう

1. クラス db.Dbuty

```
package db;

public class DbUty {

    //
    //  起動ディレクトリの flight.accdb を指し示す JDBC 用の URL の取得
    //
    public static String getDbUrl() {
        //
        //  下記の４行（コメントを入れると６行）は、Sample01 からコピーしてよい。
        //
        String prefiex = "jdbc:ucanaccess://";
        String fname = "flight.accdb";
        // このプログラムが起動された場所（フォルダー）を取得する。
        String dir = System.getProperty("user.dir");
        //
        String url = prefiex + dir + "/" + fname;
        //
        return url;
    }

}
```

2. クラス main.Sample02

```
package main;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import db.DbUty;

public class Sample02 {
    public static void main(String[] args) {

        try {
            String url = DbUty.getDbUrl();
            //
            String sql_select = "SELECT * FROM M_空港";
            //
            Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql_select);
            while (rs.next()) {
                System.out.println(
                    String.format(
                        "%s %s %s",
                        rs.getString(1),
                        rs.getString(2),
                        rs.getString(3)
                    )
                );
            }
        }
        catch (SQLException e) {
            System.out.println("err: " + e.getMessage());
        }

    }

}
```

3. 実行結果

```
CTS 札幌・新千歳 新千歳空港
FUK 福岡 福岡空港
HND 東京・羽田 東京国際空港（羽田空港）
ITM 大阪・伊丹 大阪国際空港・伊丹空港
KIX 大阪・関西 関西国際空港
NGO 名古屋・中部 中部国際空港
NRT 成田 成田国際空港
```

3.2.4.大量のデータから一部だけ抽出しよう

1. クラス main.Sample03

```
package main;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;

import db.DbUty;

public class Sample03 {
    public static void main(String[] args) {

        try {
            String url = DbUty.getDbUrl();
            //
            String sql_select =
                    "SELECT * FROM D_フライト " +
                    " WHERE フライト日付=?" +
                    " AND 出発地コード=?" +
                    "";
            //
            //  パラメータを補完するデータ（フライト日付と出発地コード）
            //       （ここのデータを変えて、結果を確認してみよう）
            //
            LocalDate flightDate = LocalDate.of(2020, 5, 15);
            String fromAirport = "NGO";
            //
            Connection conn = DriverManager.getConnection(url);
            PreparedStatement ppst = conn.prepareStatement(sql_select);
            ppst.setDate(1, Date.valueOf(flightDate));
            ppst.setString(2, fromAirport);
            ResultSet rs = ppst.executeQuery();

            while (rs.next()) {
                System.out.println(
                    String.format(
                        "%s %s %s %s %s %s %s",
                        rs.getString(1),
                        rs.getString(2),
                        rs.getString(3),
                        rs.getString(4),
                        rs.getString(5),
                        rs.getString(6),
                        rs.getString(7)
                    )
                );
            }
        }
        catch (SQLException e) {
            System.out.println("err: " + e.getMessage());
        }
    }
}
```

2. 実行結果

フライト日付：2020/5/15、出発地：NGO の場合

```
2020-05-15 00:00:00.000000 AB001 NGO 1899-12-30 06:00:00.000000 NRT 1899-12-30 07:00:00.000000 Q82
2020-05-15 00:00:00.000000 AB003 NGO 1899-12-30 12:00:00.000000 NRT 1899-12-30 13:00:00.000000 CR7
2020-05-15 00:00:00.000000 AB005 NGO 1899-12-30 18:00:00.000000 NRT 1899-12-30 19:00:00.000000 Q82
2020-05-15 00:00:00.000000 AB011 NGO 1899-12-30 06:10:00.000000 FUK 1899-12-30 07:40:00.000000 321
2020-05-15 00:00:00.000000 AB013 NGO 1899-12-30 18:10:00.000000 FUK 1899-12-30 19:40:00.000000 73P
```

1899-12-30 という日付には意味がない。この列で意味があるのは、時間の部分（時分秒）だけである。

フライト日付：2020/5/30、出発地：NRT の場合

```
2020-05-30 00:00:00.000000 AB002 NRT 1899-12-30 09:00:00.000000 NGO 1899-12-30 10:00:00.000000 Q82
2020-05-30 00:00:00.000000 AB004 NRT 1899-12-30 15:00:00.000000 NGO 1899-12-30 16:00:00.000000 CR7
2020-05-30 00:00:00.000000 AB006 NRT 1899-12-30 21:00:00.000000 NGO 1899-12-30 22:00:00.000000 Q82
2020-05-30 00:00:00.000000 AB021 NRT 1899-12-30 06:00:00.000000 FUK 1899-12-30 08:00:00.000000 73P
2020-05-30 00:00:00.000000 AB023 NRT 1899-12-30 12:00:00.000000 FUK 1899-12-30 14:00:00.000000 CR7
2020-05-30 00:00:00.000000 AB025 NRT 1899-12-30 18:00:00.000000 FUK 1899-12-30 20:00:00.000000 789
```

3.3. オブジェクトに読み込む

3.3.1.概要

データをデータベースから読み込んで画面に表示する、あるいは画面で入力されたデータをデータベースに格納するというような処理の流れの場合、データを格納するオブジェクトには３種類考えられる。

1. ビジネス・エンティティ（Business Entity）
   DB や画面に依存せず、ビジネスモデルを素直に表現したオブジェクト。
   社内の他のプロジェクトでも使用される。

   データベースから読み書きする場合は、DTO を経由して行われる。
   画面に表示／画面から入力される場合は、ビュー・モデルを経由して行われる。
   「顧客オブジェクト」「商品オブジェクト」「注文明細オブジェクト」など。

2. DTO（Data Transfer Object データ・トランスファー・オブジェクト）
   データベースの構造に即したオブジェクト。
   プログラムしやすいようにテーブルのカラムと、ほぼ１対１対応するように設計することが多い。

   なお、DB から DTO へ読み込んだり、DTO から DB へ書き込んだりする処理を行うオブジェクトを、DAO（Data Access Object）と呼ぶ。

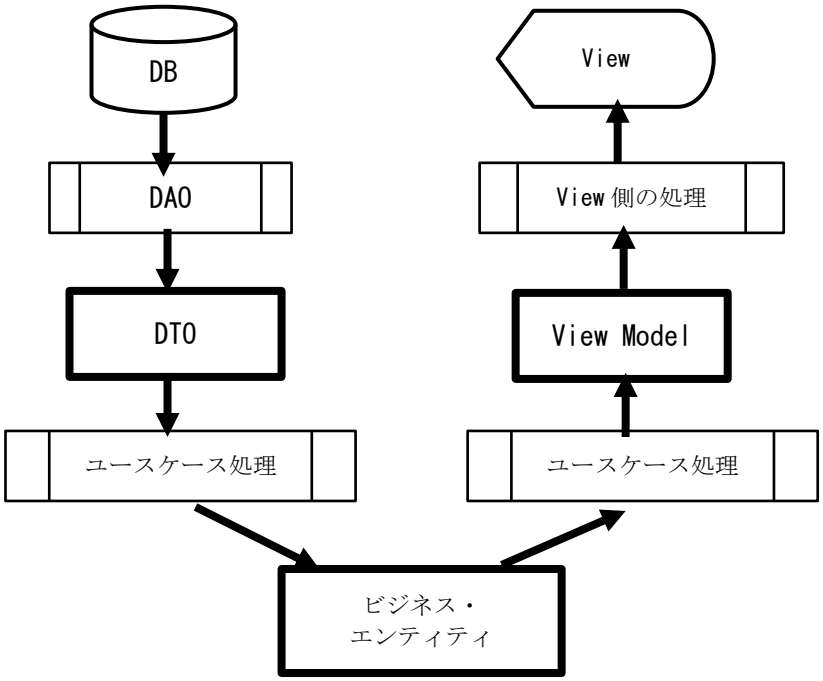
   例えば、ビジネス・エンティティとして「注文オブジェクト」があったとする。
   「注文オブジェクト」はさらに「顧客オブジェクト」と「商品オブジェクト」を含んでいるとする。
   （○○さんが▲▲商品を注文した）
   このような場合は、オブジェクト指向では注文オブジェクトの内部に直接「顧客オブジェクト」や「商品オブジェクト」を含めることができるが、リレーショナルデータベースではそれができないので、その代わりに「顧客 ID」「商品 ID」というインデックス（コード番号）を保持する。
   （いわゆる外部キー）

   このように、オブジェクト指向とリレーショナルデータベースでは、データの保持の方法が異なることがあるので、データベースから読み込んだデータは、直接ビジネス・エンティティに保持するのではなく、いったん DTO に保持するということがよく行われる。

3. ビュー・モデル（View Model）
   画面（ビュー）に表示しやすいように、また、画面の入力を受け取りやすいように設計されたオブジェクト。

   ビジネス・エンティティには色や文字列フォーマットなどの情報は存在しないが、ビュー・モデルには存在する。
   例えば、ビジネス・エンティティに「金額」という情報があったとすると、ビュー・モデルには「金額」だけでなく、「マイナスの場合は赤色、プラスの場合は黒色」とか「３桁ごとにカンマ区切りをして通貨記号を付ける」というような情報も保持する。

矢印の向きは、データの流れを示す。（依存関係や処理の流れを示すものではない）
なお、処理の流れは、この矢印とは正反対の向きになることが多い。



3.3.2.３種類は多すぎると思う場合

データの内容としては１種類のデータを格納するとに、形式としては３種類も定義するのは煩雑すぎる場合がある。

例えば、DTO もビジネス・エンティティも ViewModel もほとんど同じ形式で構わないような場合である。

そのような場合にまで、わざわざ３種類のデータを定義する必要はない。
最も重要なデータ形式は、ビジネス・エンティティなので、まずそれを定義し、DTO と ViewModel は省略してかまわない。

その場合は、DAO は直接ビジネス・エンティティに読み込んだり書き込んだりする。また、画面は直接ビジネス・エンティティの情報を表示したり格納したりする。

DTO だけ省略する、ViewModel だけ省略するということもしばしば行われる。