▼データベースシステム構築技法

▼RDB 設計・構築

サーバ構築実習

RDB 構築実習

DB 設計実習

RDB とは
RDB 設計構築の概要
物理データベースの構築
データベース設計の重要性
RDB の操作
データベース設計手法

ICT エンジニア

▲氏名＿＿＿＿＿＿＿＿＿＿

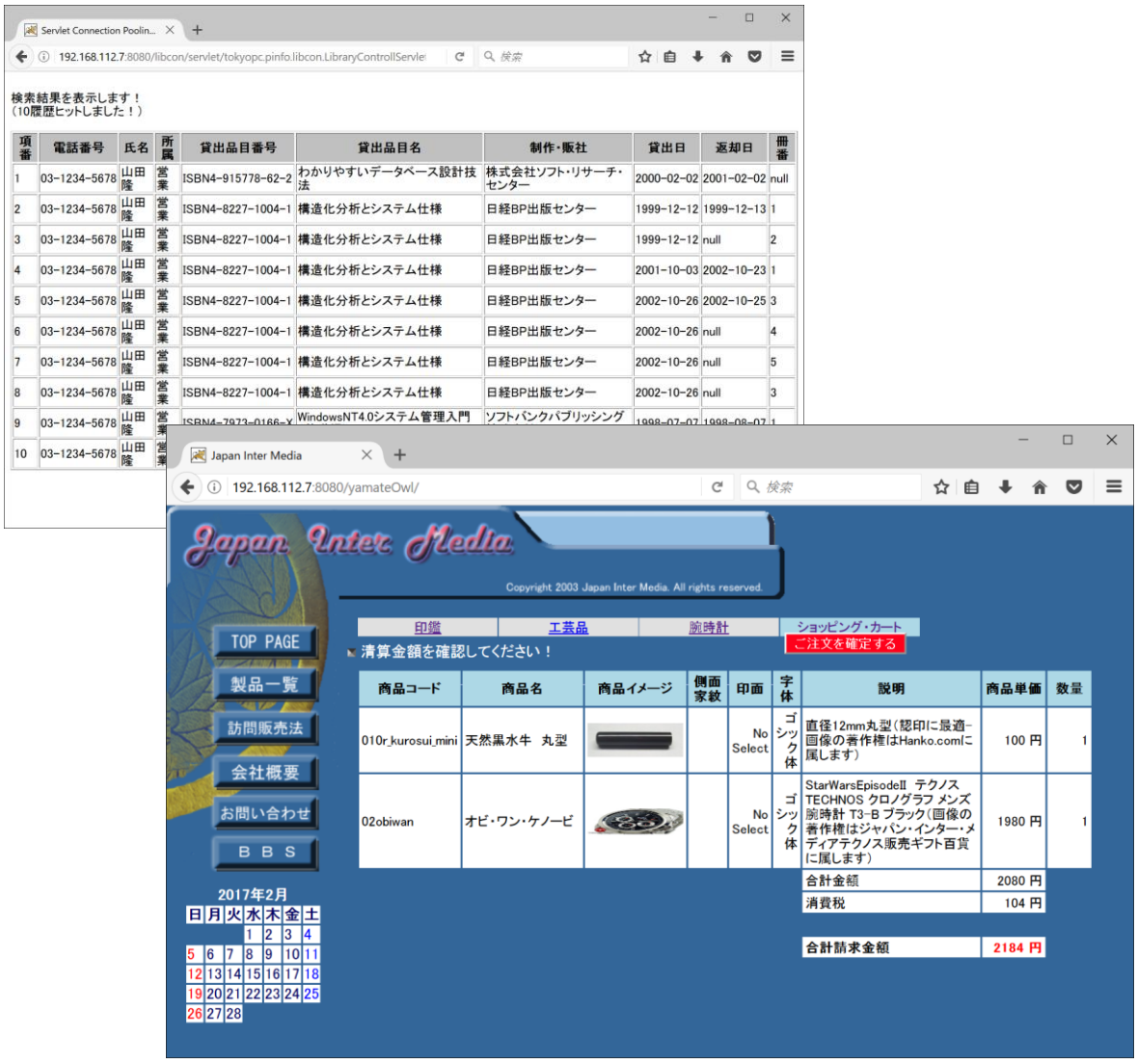
▲2025 年版

はじめに

データベースシステムとは

データベースシステムとは、一言でいうと「データを効率よく蓄積・管理・活用するための仕組み」です。もう少し具体的に言うと、データを保存・検索・更新・削除する機能を持つ**データベース管理システム（DBMS）と、それを利用するアプリケーションや利用者**を含めた全体を指します。

今回、例示的にデータベースシステムを実現するために、データベースの表示・更新・削除を行うアプリケーションを作成することにします。そのためには、大前提となるデータベースシステムの理解とデータの設計方法の習得が必須です。そこでまずは、オープンソースの DBMS である①PostgreSQL の環境を構築し、②正規化や ERD モデルなどの設計手法を学び、最終的に③Swing を使った Java のデスクトップアプリケーションを構築することにいたします。



1. RDB（リレーショナルデータベース）とは

1.1 DOA（データ中心アプローチ）技法

ビジネスモデリングのための DFD 技法

・1970 年代末に T.Demarco 氏によって提唱されたデータフローダイアグラムによるモデリング技法（構造化分析技法）。

・データフローダイアグラムとは対象世界でどのような情報がどのように流れていくかを情報のネットワークとして捕らえることを目的とし、同時に情報をどのように加工操作して新しい流れとして変質させていくかを捕らえるための技法。

・まず概要をつかみ少しずつ詳細化していくという段階的詳細化と呼ばれるパラダイムを用い「構造的なシステム開発技法」の中心として捕らえられている技法。

データ構造構築のための ERD モデリング技法

・1976 年に Peter Chen 氏によって提唱された意味データモデリング手法。

・意味データモデリングとは数学的な集合論ではなく分析対象範囲の実体（Entity）の意味上の関連（Relationship）に視点を置いて表現する技法。

・ 端的に定義すると「必要最小限のデータを使って安定したデータ構造を構築する」こととなり、一般的に E.F.Codd 博士のセット理論（データ正規化手法）を使う。

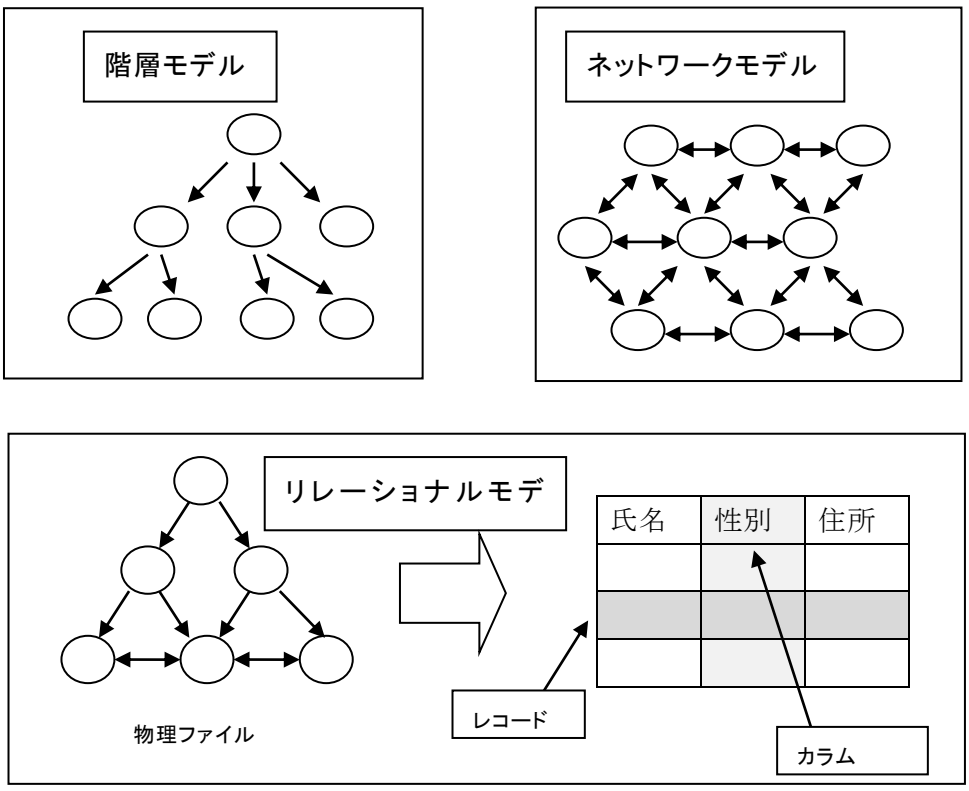
・データベース構造はリレーショナルモデルを使う。

1.2 データベースの種類

データベースとは「複数の独立した利用者に対して、要求に応じてデータを受け入れ、格納し、供給するためのデータ構造」と定義されます。

データベースには次のような種類があります。

データベースのモデル



1.3 リレーショナルモデル

リレーショナルモデルは、データをテーブルすなわち表にして論理的に表します。この概念はとらえやすく現在のデータベースの主流となっているものです。これは表形式のデータを操作できる表計算ソフトウェアが普及していることを見てもよくわかると思います。

リレーショナル型データベース（以下 RDB という）はレコード（行、タプル）とカラム（列、属性）で構成される単純な表（テーブル）です。しかし、複数のテーブル間の関連を定義することができ、それらをさまざまな視点から表示することができます。さらに RDB では複数のテーブルから必要なレコードやカラムのみを検索して取り出すことも可能です。

RDB ではこのようなデータベースに対する定義や操作をユーザが SQL 文を発行することにより行います。

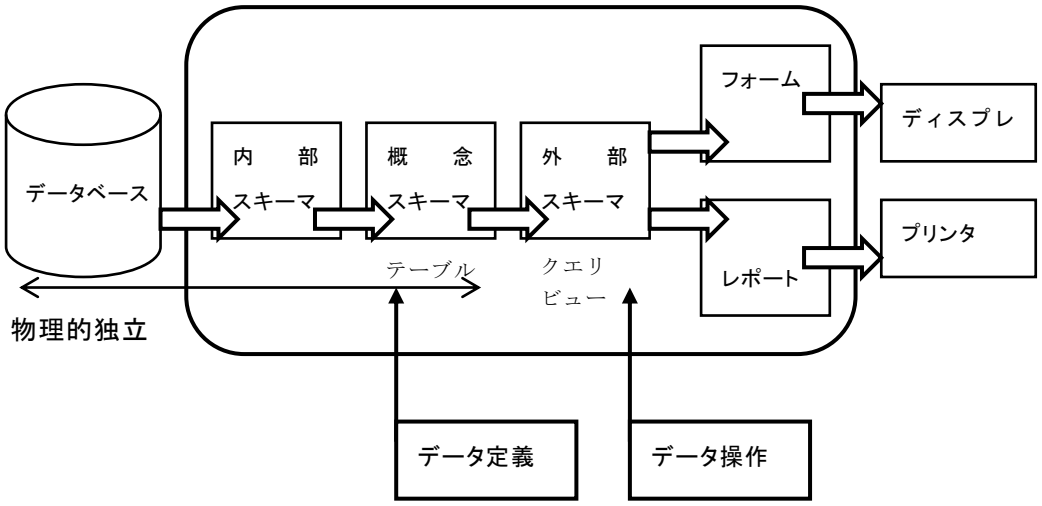
1.4 RDB の処理構造

RDB ではユーザがデータベースを処理するためには SQL 文を発行しなければなりません。SQL 文とはデータ処理の方法を RDB に対して指示する命令文のことで、「問い合わせ言語」とも呼ばれます。SQL には大きく分けて次の二つの機能があります。

データ定義	関係表の形を作ること、実テーブルを作ること
データ操作	関係表の演算をすること
データ制御	関係表の整合性の制御をすること

データ定義は SQL-ＤＤＬ（Data Definition Language）文を用いて行い
データ操作は SQL-ＤＭＬ（Data Manipulation Language）文を用いて行われます。
データ制御は SQL-ＤＣＬ（Data Control Language）文を用いて行われます。

1.5 データ中心思想



データベースにはデータの独立という考え方が根底にあります。データの管理には、実体をどのように表すかという論理的問題とファイルやディスクにどのように格納するかという物理的問題があります。データベースではこれをそれぞれ独立させ論理的な管理をする場合には、物理的構造を意識せずに行うことができるようになっています。

上図で表されているスキーマとはデータの定義構造のことで、内部スキーマ、概念スキーマ、外部スキーマという三つの観点からデータベースの構造を捕らえることを三層スキーマ構造と呼んでいます。

1.6 RDB の三層スキーマ構造

内部スキーマ：データをディスクに格納するための物理的な管理の定義
システムに依存する（ファイルアクセス方式、ファイル編成方式）

概念スキーマ：実体をデータで表すときどんな属性で表すのかという関係表の定義
概念設計（テーブル自体の構造の設計）　→　実表

外部スキーマ：演算要求に応じて別の表を作成提供する定義
論理設計（View,Query の定義）　→　仮想表

RDB を使う時、通常ユーザは内部スキーマを意識することはありません。論理的に表の形でテーブルを設計、作成しデータをその中に格納すると考えればよいのです。

ここで作成された関係表は実表と呼ばれ、実データが格納されています。実表を元に必要な部分を演算により導出されたものが仮想表です。仮想表は別の実表が作成されたわけではなく「あたかも別の表のように見せている」だけです。

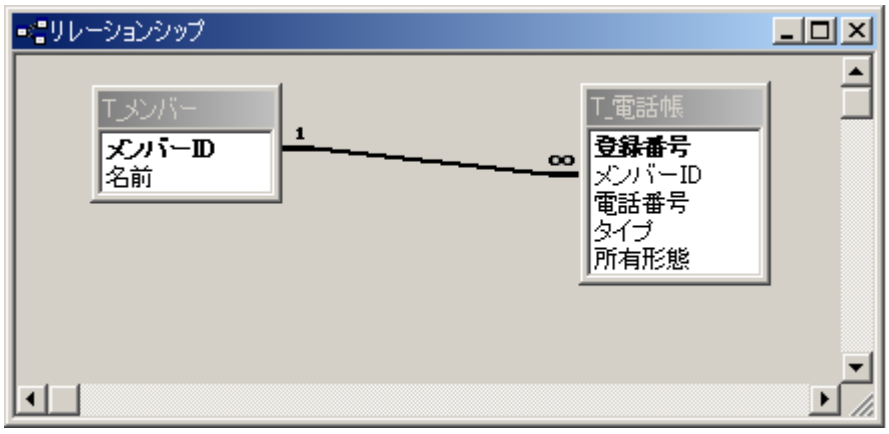
論理的独立とは実表を変更することなく、仮想表の変更で必要とする情報を取得することだと言い換えることができます。また**物理的独立とは**物理的なデータベース環境を変更することがあっても実表の定義を変える必要はないということをあらわしています。

1.7 RDB の設計（概念設計と正規化）

テーブルの構造設計を行うには **Entity Relation Diagram（ERD）**と呼ばれるモデリング手法が用いられます。また、ERD を作成するにあたって正規化手法からアプローチを行うことができます。正規化とは、与えられたデータの関係を分析し、相互の関連性を維持した冗長性のないデータの集合に分解し、整理することです。これにより、基本的なデータ項目グループの識別、同一属性を複数箇所で保持するといった冗長性の排除、データの整合性の維持をはかることができます。今回は構築主体となりますので設計手法については詳しくは述べません。

2. RDB の設計構築の概要

2.1 概念設計　Entity Relation Diagram（ERD）　-- データベースの設計図 --



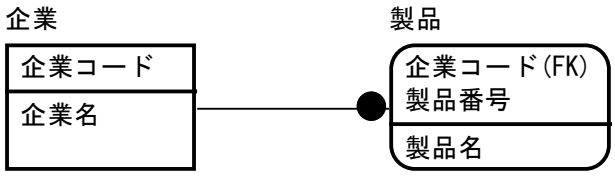
ＥＲＤは、データ中心アプローチのもっとも大切な道具です。これを利用すれば、対象となるデータをエンティティ（実体・管理対象）を四角い箱、リレーション（関係・関連）を線として、視覚的、直感的に表現できます。

ＥＲＤによるモデリングというのは、日常システム設計に関わっている限りでは聞き慣れない言葉かもしれませんが、1970 年代半ばにピーター・チェン（Peter Chen）が提唱した「意味データモデル」と呼ばれる理論です。これはネットワークモデルのように集合間の関係を数学的なルールを適用して表現するモデルとは異なり、実世界の意味上の関係を中心に表現するものです。ＥＲＤを理解するにはエンティティ及びリレーションシップ（関連）という二つの概念を理解する必要があります。

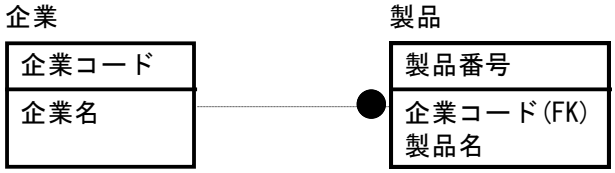
ERD は実体（Entity）と関連（Relationship）の 2 つの概念で世の中の存在を表現する

例：依存と非依存関係の例

・依存関係　　子エンティティの主キー（データを一意に定めることのできる属性）にキー属性「企業コード(FK)」を入れる



・非依存関係　親エンティティのキー属性は子エンティティの主キーに入れない

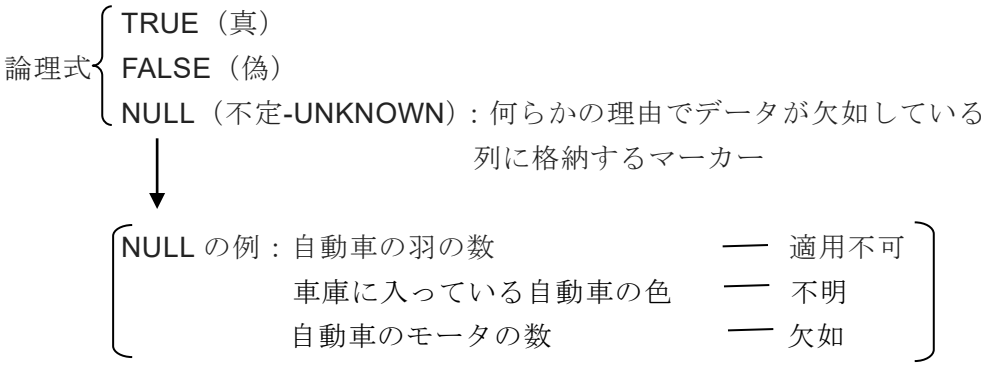


2.2 構築　Structured Query Language（SQL）　-- データベース操作言語 --

SQL は 1970 年代に IBM によって開発されたデータベース操作言語です。1979 年に Oracle によって最初に商用実装され、現在の標準 SQL の最新の規格は ISO の SQL：2011 となっていますが、対応状況は開発会社ごとにまちまちです。

SQL の特徴

集合単位の操作：データの任意に大きな集合に対して一度に処理を行う
３値論理　　：3 Valued Logic (TRUE, FALSE,NULL)
※コンピュータの世界でもっとも広く利用されているのは 2 値論理(TRUE, FALSE)



SQL の機能的分類

SQL の機能は、大きく分けて２つの機能があります。1 つは、スキーマ（データベースの枠組み）の定義機能（SQL-DDL）で、もう 1 つはデータの操作機能（SQL-DML）です。

SQL は ISO や JIS で規格化されたリレーショナルデータベース言語ですが、COBOL 言語や PL/ I 言語といったプログラミング言語とは処理対象や起動のされ方を異にします。

① 処理対象

SQL の処理対象は、あくまでもデータベースの操作が中心であり、細かな計算や編集、プログラムの流れの制御は得意としません。

② 起動方法

・直接起動：利用者が直接端末から SQL を指令として起動します。
・モジュール呼び出し： 親言語から、SQL の規定するモジュール言語で書かれた SQL（ストアドプロシージャ）文からなるモジュールを CALL 文等によって呼び出します。
・埋め込み SQL: 親言語で書かれたソースプログラムの中に直接 SQL 文を埋め込んだ、埋め込み SQL 構文を使って結果を処理します。
・動的 SQL：親言語で書かれたソースプログラムから実行時に SQL 文を実行する方法。

SQL はスキーマ定義機能によって、表の定義、列の定義、データの型の定義を行います。以下のような構文が定義されています。

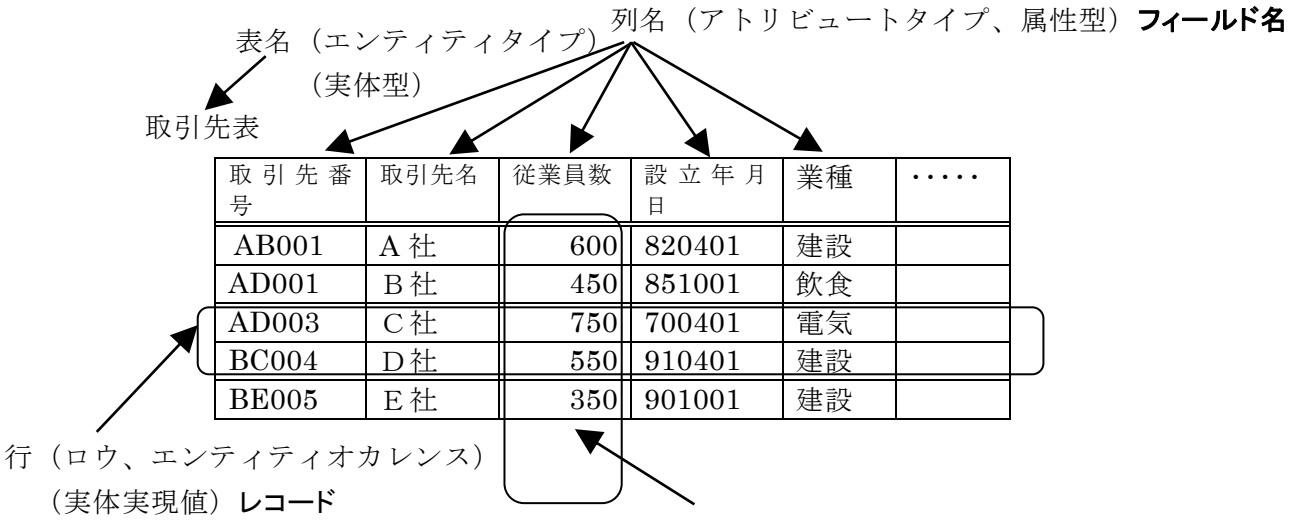
データ定義言語（SQL-DDL）
- CREATE TABLE
- CREATE VIEW

データ操作言語（SQL-DML）
- SELECT
- INSERT
- UPDATE
- DELETE

データ制御言語（SQL-DCL）
- GRANT
- REVOKE
- START TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK TRANSACTION

2.2.1 Table Structure （テーブル構造）

テーブル（表）の構成は以下のようになります。



取引先番号	取引先名	従業員数	設立年月日	業種	・・・・・
AB001	A 社	600	820401	建設	
AD001	B 社	450	851001	飲食	
AD003	C 社	750	700401	電気	
BC004	D 社	550	910401	建設	
BE005	E 社	350	901001	建設	

フィールドとレコード

テーブルの各列がフィールドで、各フィールドの先頭にある項目がフィールド名です。また、各フィールドによって構成される１行がレコードです。

フィールドプロパティ

フィールドにはそこに格納するデータ型（数値、文字、画像など）や長さ（文字長、少数以下が必要か)、どんな形式で入力させるかなど細々とした設定が必要です。これらをフィールドプロパティと呼びます。

2.2.2 データ定義言語（SQL-DDL）

(1)表定義の具体例

```
CREATE TABLE    取引先表
    (取引先番号   CHAR(8),
     取引先名     VARCHAR(20)
     従業員数     INT
     設立年月日   DATE
     ・・・・・・・・・・・・・・・・・・
```

SQL の主なデータの型は以下の通りです。

データ型	意味
CHAR(n)	指定された長さを持つ文字列のデータ型。 固定長文字列。 n は長さ（1～255）
VCHAR(n)	長さが可変長である文字列のデータ型 可変長文字列。 n に最大長を指定
INTEGER	整数のデータ型
DATE	日付（yyyy-mm-dd、mm/dd/yy……）

(2)ビュー表の具体例

SQL スキーマで定義できる表には、実表とビュー表の 2 種類あります。実表とは、実在する表のことで、ディスク上に物理的に記録されています。ビュー表とは、対応するデータがディスク上には記録されていません。実表を別の角度から見た仮想の表という位置付けになります。

以下の例は、従業員数が 500 名以上で、業種が“建設”といった取引先の行の集合からなる仮想表の例です。

ビュー表の例

```
CREATE VIEW     建設 500 人以上表（取引先番号，取引先名）
  AS SELECT     ＊
    FROM        取引先表
    WHERE       従業員数＞＝500
    AND         業種＝“建設”
```

(3)整合性制約と外部キー

整合性制約とは、表の中のデータの正しさを保証する仕組みのことで、一貫性制約とも呼ばれています。ERD において、テーブルの関係を示す線であらわされています。

整合性制約には以下のものがあります。

整合性制約の例

表制約	一意性制約	表中の行の一意性を制約
	PRIMARY KEY	表中の行が 1 つの実体を表現することの宣言
	UNIQUE	制約の指定された列あるいは列の集合に対し、データ値の重複を禁止
	NOT NULL	非ヌル値制約。制約の指定された列に対して、値が常に確定していることの宣言
	参照制約	表間の参照関係指定
	REFERENCES	列の値が別の表の値を参照することを宣言 FOREIGN KEY と共に用いられる

ある表の列が別の表の基本キーの値と同じモノを持っている場合、その列を外部キー（FOREIGN KEY）といいます。

外部キーと基本キーの関係はいわば主と従の関係ともいえます。こうした主従関係においては次の規則を守る必要があります。

- 外部キーの値は主表の基本キーに存在しなければならない。
- 外部キーの削除は無条件に行われる。
- 外部キーの挿入と更新では制限付き操作が行われる。
  つまり挿入（更新）後の外部キーが主表の基本キーに存在するときだけ挿入（更新）が行われる。

- 基本キーの削除と更新に関しては次のいずれかの操作が行われる。

① NO ACTION または RESTRICT 操作（制限付操作）
外部キーに同じ値がないときのみ削除（更新）される。同じ値があるときには、主表の基本キーの削除（更新）は行われない。

② CASCADE 操作（波及的操作）
主表の基本キーも関連する外部キーも波及的に削除（更新）される。

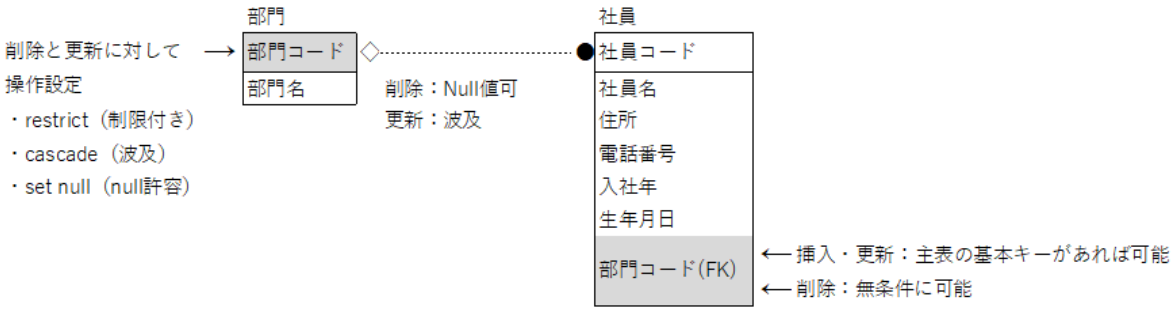
③ SET NULL 操作（空白値化操作）
主表の基本キーは削除（更新）され、関連する外部キーは空白値に設定される。当然、外部キーは NotNull であってはならない。

※Null とは値が定まらない（不定）の意味

この、外部キーに関する操作の定義は、外部キーを持つ表の CREATE TABLE 文の中で DELETE、UPDATE それぞれに①～③のいずれか一つを指定できる。

ここに示した規則にしたがってデータが常に正しい状態になっているとき、参照の保全性または参照の整合性が確立しているといいます。この意味で上述した規則のことを参照制約といいます。

参照制約を定義するには CREATE TABLE 文で外部キーを示す FOREIGN KEY に続けて指定します。つまり参照制約の規則は外部キーが存在する従属表の側に定義します。これは最初に主表の基本キーを定義してから、外部キーを定義することに起因しています。



つまり

CREATE TABLE 文でテーブルを作成する場合、当然のことながら参照される側のテーブルが先に作られていないといけません。INSERT 文でデータを入力するときも参照される側に先にデータがないといけません。

部門と社員の複合テーブルの例

社員コード	社員名	住所	電話番号	入社年	生年月日	部門名
M001	山田一郎	神奈川県横浜市	045-641-1234	2007-04-01	1961-04-30	営業一課
M002	田中二郎	神奈川県藤沢市	0466-82-3774	2010-04-01	1971-05-30	営業二課
F001	佐藤華子	神奈川県横浜市	045-123-5432	2015-04-01	1981-06-30	総務
↑						↑
Primary key						Foreign Key

2.2.3 データ操作言語（SQL-DML）

データ操作の SQL 例

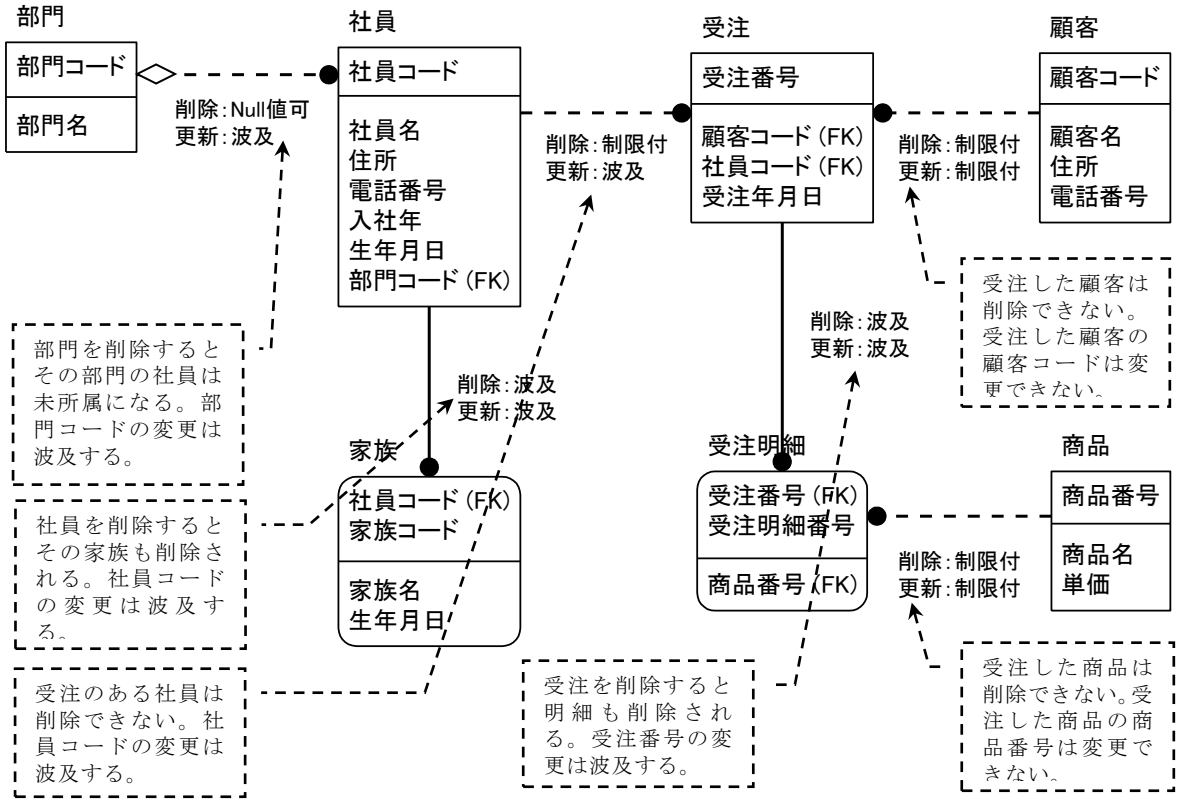
項目	SQL 文	内容
データ操作	SELECT	表から条件に合ったデータを検索する
	INSERT	表に行を挿入する
	UPDATE	表のデータを更新する
	DELETE	表の行を削除する

2.2.4 データ制御言語（SQL-DCL）

データ制御の SQL 例

項目	SQL 文	内容
データの制御	GRANT	作成したユーザに対して権限を設定する
	REVOKE	ユーザに対して設定した権限を削除する
	START TRANSACTION	データベースの更新処理を開始する
	COMMIT	データベースの更新処理を確定する
	ROLLBACK	データベースの更新処理を取り消す

2.3 実習で使用するデータベースの設計図



この ERD をもとに物理データベースを作成します。

2.4 実習環境の RDB のセットアップ（SQL によるデータベースの作成）

今回の実習で使用するデータベースサーバには、PostgreSQL を用いることにします。これは WindowsPC のローカルな環境でも Linux のネットワーク環境でも実行可能であり、かつ Free で動作実績の高いシステムを考慮した結果です。データベースはいわば箱ですので、それを利用するクライアントシステムが必要になります。Java 等でクライアントソフトを作成することもかのですが、今回は既存の SQL 操作環境である cse（Common SQL Environment）を用いることにします。最初は Windows 環境に DataBase をインストールし SQL に慣れた後に Linux サーバに DataBase をセットアップし運用する方法を学ぶことにします。インストールにかかわる資料は別途提示しますので、そちらを参照してください。以下なような手順で実施します。

(1)PostgreSQL のインストール

①PostgreSQL のインストール
②PostgreSQL 動作環境設定
③PostgreSQL 起動確認

(2) SQL 実行環境 CSE のインストール

①CSE インストール
②CSE から PostgreSQL 接続

(3)SQL 実習環境構築

①CSE から SQL 投入
（DB_CREATE.sql）

3. 物理データベースの構築　　-PostgreSQL-

3.1 SQL-DDL（テーブル作成）

●2 社員テーブルの定義

```
CREATE TABLE 社員
 (社員コード CHAR(4) NOT NULL,
  社員名 VARCHAR(10) NOT NULL,
  住所 VARCHAR(20),
  電話番号 CHAR(20),
  入社年 DATE,
  生年月日 DATE,
  部門コード CHAR(4),
 PRIMARY KEY (社員コード),
 FOREIGN KEY (部門コード)
  REFERENCES 部門 ON DELETE SET NULL
                  ON UPDATE CASCADE)
;
```

●6 受注テーブルの定義

```
CREATE TABLE 受注
 (受注番号 INT NOT NULL,
  顧客コード CHAR(4) NOT NULL,
  社員コード CHAR(4) NOT NULL,
  受注年月日 DATE NOT NULL,
 PRIMARY KEY (受注番号),
 FOREIGN KEY (社員コード)
  REFERENCES 社員 ON DELETE RESTRICT
                  ON UPDATE CASCADE,
 FOREIGN KEY (顧客コード)
  REFERENCES 顧客 ON DELETE RESTRICT
                  ON UPDATE RESTRICT)
;
```

SQL 文はメモ帳などを使って記入しコピー&ペーストで CSE の Window に張り付けるようにしてください。
作成した SQL 文は保存しておいてください。

●3 家族テーブルの定義

```
CREATE TABLE 家族
 (社員コード CHAR(4) NOT NULL,
  家族コード CHAR(4) NOT NULL,
  家族名 VARCHAR(10) NOT NULL,
  生年月日 DATE,
 PRIMARY KEY (社員コード,家族コード),
 FOREIGN KEY (社員コード)
  REFERENCES 社員 ON DELETE CASCADE
                  ON UPDATE CASCADE)
;
```

●7 受注明細テーブルの定義

```
CREATE TABLE 受注明細
 (受注番号 INT NOT NULL,
  受注明細番号 INT NOT NULL,
  商品番号 INT NOT NULL,
  数量 INT NOT NULL,
 PRIMARY KEY (受注番号,受注明細番号),
 FOREIGN KEY (受注番号)
  REFERENCES 受注 ON DELETE CASCADE
                  ON UPDATE CASCADE,
 FOREIGN KEY (商品番号)
  REFERENCES 商品 ON DELETE RESTRICT
                  ON UPDATE RESTRICT)
;
```

●5 商品テーブルの定義

```
CREATE TABLE 商品
 (商品番号 INT NOT NULL,
  商品名 VARCHAR(10) NOT NULL,
  単価 INT,
 PRIMARY KEY (商品番号))
;
```

●4 顧客テーブルの定義

```
CREATE TABLE 顧客
 (顧客コード CHAR(4) NOT NULL,
  顧客名 VARCHAR(10) NOT NULL,
  住所 VARCHAR(20),
  電話番号 CHAR(20),
 PRIMARY KEY (顧客コード))
;
```

●1 部門テーブルの定義

```
CREATE TABLE 部門
 (部門コード CHAR(4) NOT NULL,
  部門名 VARCHAR(10) NOT NULL,
 PRIMARY KEY (部門コード))
;
```

3.2 SQL-DML（データ入力）

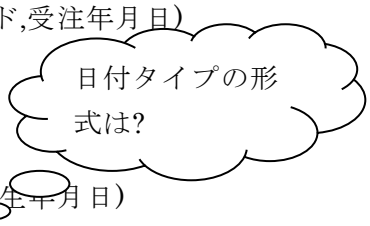
ここからは入力用のファイルを提供します。テーブルが問題なく作成されていればデータが投入されるはずですが、投入順に注意してください。

●2 社員データ入力

```
INSERT INTO 社員(社員コード,社員名,住所,電話番号,入社年,生年月日,部門コード)
 VALUES('M001','山田一郎','神奈川県横浜市'
        ,'045-641-1234','2007/04/01','1961/04/30','E01')
;
INSERT INTO 社員(社員コード,社員名,住所,電話番号,入社年,生年月日,部門コード)
 VALUES('M002','田中二郎','神奈川県藤沢市'
        ,'0466-82-3774','2010/04/01','1971/05/30','E02')
;
INSERT INTO 社員(社員コード,社員名,住所,電話番号,入社年,生年月日,部門コード)
 VALUES('F001','佐藤華子','神奈川県横浜市'
        ,'045-123-5432','2015/04/01','1981/06/30','S01')
;
```

●6 受注データ入力

```
INSERT INTO 受注(受注番号,顧客コード,社員コード,受注年月日)
 VALUES(100,'C005','M001','1999/07/28')
;
INSERT INTO 受注(受注番号,顧客コード,社員コード,受注年月日)
 VALUES(101,'C005','M002','1999/07/29')
;
INSERT INTO 受注(受注番号,顧客コード,社員コード,受注年月日)
 VALUES(102,'D010','M002','1999/07/30')
;
INSERT INTO 受注(受注番号,顧客コード,社員コード,受注年月日)
 VALUES(103,'G001','M001','1999/08/01')
;
```



●3 家族データ入力

```
INSERT INTO 家族(社員コード,家族コード,家族名,生年月日)
 VALUES('M002','実母','田中洋子','1941/08/01')
;
INSERT INTO 家族(社員コード,家族コード,家族名,生年月日)
 VALUES('F001','実父','山田太一','1921/09/01')
;
```

●7 受注明細データ入力

```
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(100,01,1001,20)
```

```
;
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(100,02,1003,15)
;
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(100,03,2001,10)
;
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(101,01,2001,5)
;
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(102,01,3000,30)
;
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(102,02,2001,6)
;
INSERT INTO 受注明細(受注番号,受注明細番号,商品番号,数量)
 VALUES(103,01,1001,25)
;
```

●5 商品データ入力

```
INSERT INTO 商品(商品番号,商品名,単価)
 VALUES(1001,'プリンタ１型',300)
;
INSERT INTO 商品(商品番号,商品名,単価)
 VALUES(1003,'プリンタ３型',250)
;
INSERT INTO 商品(商品番号,商品名,単価)
 VALUES(2001,'ディスク１型',910)
;
INSERT INTO 商品(商品番号,商品名,単価)
 VALUES(3000,'システム０型',300)
;
```

●4 顧客データ入力

```
INSERT INTO 顧客(顧客コード,顧客名,住所,電話番号)
 VALUES('C005','東京商事','東京都千代田区神田','03-1234-5678')
;
INSERT INTO 顧客(顧客コード,顧客名,住所,電話番号)
 VALUES('D010','大阪商会','大阪府大阪市東区北浜','06-432-0123')
;
INSERT INTO 顧客(顧客コード,顧客名,住所,電話番号)
 VALUES('G001','福岡商会','福岡県太宰府市都府楼','092-901-1278')
;
```

●1 部門データ入力

```
INSERT INTO 部門(部門コード,部門名) VALUES('E01','営業一課')
;
INSERT INTO 部門(部門コード,部門名) VALUES('E02','営業二課')
;
INSERT INTO 部門(部門コード,部門名) VALUES('S01','総務')
;
```

4. データベース設計の重要性

4.1 SQL-DML（データ確認）

●顧客データ確認

```
select * from 顧客
;
```

●商品データ確認

```
select * from 商品
;
```

●受注データ確認

```
select * from 受注
;
```

●受注明細データ確認

```
select * from 受注明細
;
```

●部門データ確認

```
select * from 部門
;
```

●社員データ確認

```
select * from 社員
;
```

●家族データ確認

```
select * from 家族
;
```

無題 - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
//●親を作成
CREATE TABLE 企業
(企業コード VARCHAR(5) NOT NULL,
企業名 VARCHAR(10) ,
 PRIMARY KEY(企業コード));
//●子を作成
CREATE TABLE 製品
(企業コード VARCHAR(5) NOT NULL,
製品番号 VARCHAR(5) NOT NULL,
製品名 VARCHAR(10) ,
 PRIMARY KEY(企業コード,製品番号),
 FOREIGN KEY(企業コード)
 REFERENCES 企業 ON DELETE CASCADE
 ON UPDATE CASCADE);

//●親を入力
INSERT INTO 企業 VALUES('1','Apple');
INSERT INTO 企業 VALUES('2','DELL');
INSERT INTO 企業 VALUES('3','FUJITSU');
//●子を入力
INSERT INTO 製品 VALUES('1','12345','MacPro');
INSERT INTO 製品 VALUES('2','67890','DeskPower');
INSERT INTO 製品 VALUES('3','12345','FMV');
//●親子を確認
SELECT * FROM 企業;
SELECT * FROM 製品;

//●更新波及
UPDATE 企業 SET 企業コード='4' WHERE 企業コード='1';
SELECT * FROM 企業;
SELECT * FROM 製品;

//●削除波及
DELETE FROM 企業 WHERE 企業コード='4';
SELECT * FROM 企業;
SELECT * FROM 製品;

//●製品一覧作成（仮想表）
SELECT 企業.企業名, 製品.製品番号, 製品.製品名
FROM 企業,製品 WHERE 企業.企業コード = 製品.企業コード;

//●製品一覧ビュー
CREATE VIEW v製品一覧 AS
SELECT 企業.企業名, 製品.製品番号, 製品.製品名
FROM 企業,製品 WHERE 企業.企業コード = 製品.企業コード;

//●ユーザーtestuser2 に View の参照権を付与
GRANT select ON v製品一覧 TO testuser2;
```

4.2 SQL-DML（参照制約確認）

; --参照整合性制約

; --更新波及の確認

;部門コードの変更は波及する

```
select * from 部門
;
select * from 社員
;
update 部門 set 部門コード='S00' where 部門コード='S01'
;
select * from 部門
;
select * from 社員
;
```

; --Delete Set Null（削除 Null 値可）の確認　部門を削除するとその部門の社員は未所属になる

```
select * from 部門
;
select * from 社員
;
delete from 部門 where 部門コード='S00'
;
select * from 部門
;
select * from 社員
;
```

; --更新波及の確認　社員コードの変更は波及する

```
select * from 社員
;
select * from 家族
;
select * from 受注
;
update 社員 set 社員コード='M003' where 社員コード='M002'
;
select * from 社員
;
select * from 家族
;
select * from 受注
;
```

; --Delete RESTRICT（削除制限付き）の確認　M003 は削除制限付の受注があるため削除できない

```
select * from 社員
;
select * from 家族
;
select * from 受注
;
; --;
delete from 社員 where 社員コード='M003'
;
select * from 社員
;
select * from 家族
;
select * from 受注
;
```

; --Delete CASCADE（削除波及）の確認　社員 F001 は受注がないため連鎖削除可能

```
select * from 社員
;
select * from 家族
;
delete from 社員 where 社員コード='F001'
;
select * from 社員
;
select * from 家族
;
```

; --更新波及の確認　受注番号の変更は波及する

```
select * from 受注
;
select * from 受注明細
;
update 受注 set 受注番号=999 where 受注番号=100
;
select * from 受注
;
select * from 受注明細
;
```

; --Delete CASCADE（削除波及）の確認　受注を消せば明細も消える

```
select * from 受注
;
select * from 受注明細
;
delete from 受注 where 受注番号=999
;
select * from 受注
;
select * from 受注明細
;
```

4.3 SQL-DDL/DML（再構成）

作成した 7 つのテーブルを一旦削除し、再度生成してください。

***テーブル削除時にもデータの有無と削除順に注意する必要があります。作成時の逆順です！**

削除用 SQL

```
DROP TABLE 受注明細;
DROP TABLE 受注;
DROP TABLE 商品;
DROP TABLE 顧客;
DROP TABLE 家族;
DROP TABLE 社員;
DROP TABLE 部門;
```

再生成用 SQL　再度生成するときは 3.1 と 3.2 で作った SQL 文を CSE にコピーして
テーブルの作成とデータの入力を行ってください。

5. RDB の操作

5.1 データ検索

5.1.1 ある表にある全てのデータを見る

【構文】 Select ＊ from 表名

《例》表「受注明細」の全てのデータを表示する。

Select ＊ from 受注明細

【Exp 1】表「商品」 の全てのデータを表示してください。

5.1.2 ある表の特定の列を見る

【構文】 Select 列名１, 列名２,・・ from 表名

《例》表「社員」の列名が 社員コード、社員名、であるデータを表示する。

Select 社員コード, 社員名 from 社員

【Exp 2】表「社員」の列名が 社員名, 住所, 電話番号のデータを表示して下さい。

5.1.3 条件指定による検索

【構文】 Select ＊ from 表名 Where 条件 （And または Or 条件）
【構文】 Select 列名１, 列名２,・・ from 表名 Where 条件
（And または Or 条件）

5.1.3.1 ある値と等しいものを検索する

《例》表「商品」の列値 単価 = 300 の 商品番号、商品名 を表示する。

Select 商品番号, 商品名 from 商品 Where 単価 = 300

検索条件は、列のデータ型によって記述が異なります。

＜文字型＞

variant char, char などの文字型を比較するときは,値をシングルコーテーション「‘」でくくります。大文字、小文字も正しく入力する必要があります.

＜数値型＞

数値型のタイプを比較するときは、値をそのまま記述します。

＜日付型＞

Date など日付型はシングルコーテーション「‘」でくくります。ただし,日付の表現は幾通りかありますので、複数の書き方があります。

Date 型の場合、’99/07/01’ でも’99-07-01’ 、’99/7/1’ でもかまいません。

[注意] データベースによっては ‘yyyy-mm-dd’ でないとダメなものもあります

【Exp ３】表「受注」 の列名　社員コード の値が M002 のデータを表示して下さい。
(大文字小文字区別されます)

【Exp ４】表「受注」 の列名　受注年月日 の値が 99/08/01 のﾃﾞｰﾀを表示して下さい。
(書式違いでもヒットする場合あり)

5.1.3.2 大小を比較する

比較記号は次ぎのようになります。　　＊ ある値と等しくないものの記述

＞、 ＞＝、 ＜、 ＜＝　　！＝、 ＜＞

※日付の比較の場合 DATE 型は年月日とともに時分秒を保持している場合があり注意が必要です。

《例》 表「受注明細」の列名　数量 の値が　10　より小さいデータを表示する。

Select 受注番号,受注明細番号,商品番号,数量 from 受注明細 Where 数量 ＜ 10

【Exp ５】表「受注」の列名 受注年月日 の値が 99/8/1 より前のデータを表示して下さい。

5.1.3.3 値の範囲で比較する

【構文 １】 値を含む　　Where 列名 Between 値１ And 値２ 【構文 ２】 値を含まない Where 列名 Not Between 値１ And 値２

《例》表「商品」の列名　単価　の値が２００以上３００以下のデータを表示する。

Select 商品番号,商品名,単価 from 商品 Where 単価 Between 200 AND 300

《例》表「商品」の列名　単価　の値が２００以上３００以下でないデータを表示する。

Select 商品番号,商品名,単価 from 商品 Where 単価 Not Between 200 AND 300

【Exp ６】表「受注」の列名 受注年月日 の値が 99/7/29 から 99/7/30 のデータを表示して下さい。

5.1.3.4 あいまいな文字列を検索する

【構文】 Where 列名 LIKE ‘検索パターン’

検索パターンは「% 」と「_ 」の２種類の記号を使って表します。
「% 」は 0 文字以上の任意の文字列を意味し、「_ 」は１文字を意味します。
［‘%MU’］であれば、MU で終わる全ての文字列を表し、［‘MU%’］は MU で始まる全ての文字列になります。また［‘%MU%’］とすれば MU を含む（文字列のどこかに MU がある）文字列となります。
［‘_ _MU’］は、頭に任意の２文字で終わりが MU で終わる文字列を意味し、［‘MU_ _’］

は、MU で始まり後ろに２文字続く文字列を意味します。
《例》表「顧客」の列名　顧客名　の値の最後に商会がつくデータを表示する。

Select * from 顧客 where 顧客名 like '%商会'

《例》表「顧客」の列名　住所　の値に千代田区が含まれるデータを表示する。

Select * from 顧客 where 住所 like '%千代田区%'

【Exp ７】表「顧客」の列名 電話番号 の値が 092 で始まるデータを表示して下さい。

5.1.3.5 値の集合により検索する

【構文】 Where 列名 IN （値 1，値 2，…… ）

列の内容が括弧中のどれかの値と等しければ検索対象になります。
《例》表「商品」の列名　商品名　の値が　システム０型　と　ディスク１型　のデータを表示する。

Select * from 商品 where 商品名 in ('システム０型', 'ディスク１型')

《例》表「商品」の列名　商品名　の値が　システム０型　と　ディスク１型　以外のデータを表示する。

Select * from 商品 where 商品名 not in ('システム０型', 'ディスク１型')

【Exp ８】表「顧客」の列名 顧客コード　の値が　C005　のデータを表示して下さい。

5.1.3.6 データを並べ替えて表示する

【構文】 Order　By 列名 1，列名 2，……

複数の列名を指定すると,最初の列名が第一キーとなり昇順で並び替えます。
降順で並び替えたいときは、列名の後に DESC とつけます。

《例》表「商品」 を第一キー　単価　で昇順に並び替えてデータを表示する。

Select * from 商品 order by 単価

《例》表「商品」 を第一キー　単価　で昇順に、第二キー　商品番号　で降順に並び替えてデータを表示する。

Select * from 商品 order by 単価, 商品番号 DESC

Where 句を記述するときは、Order By 句の前に記述します。

《例》**Select * from 商品 Where 単価 < 900 order by 単価**

5.1.4 検索内容を加工する

5.1.4.1 演算子を使って計算する

《例》表「商品」の列名　単価　の値を　×1.05　にして表示する。

Select 商品番号,商品名,単価*1.05 from 商品

5.1.4.2 関数を使用して計算する

ROUND（数値、桁数）：四捨五入　　TRUNC（数値、桁数）：切り捨て

《例》表「商品」の列名　単価　の値を　×1.05　にして円以下四捨五入して表示する。

Select 商品番号,商品名,単価*1.05, ROUND(単価*1.05, 0) from 商品

【Exp 9】表「商品」の列名　単価　の値を　×1.05　にして円以下切り捨てて表示して下さい。

関数は、数値関数、文字列関数、日付関数、集計関数等様々なものがあります。

5.1.4.3 列名の表示を変更する

SQL には、検索時に列のタイトルを変更して表示し、利用者に解りやすいものに指定する機能があります。

【構文】 Select 列名 as 表示したい列名，列名 as 表示したい列名，… From …

《例》表「商品」の列名 単価 の値を ×1.05 にして円以下四捨五入した列を 税込価格 として表示する。

Select 商品番号,商品名,単価, ROUND(単価*1.05, 0) as 税込価格 from 商品

この場合、[税込価格] 列名の間にスペースを入れたい場合［税込　価格］をダブルコーテーションでくくります。

【Exp 10】表「商品」の列名 単価 の値を ×1.05 にして円以下切り捨てた列を 税込価格　端数切捨　として表示して下さい。

5.1.4.4 条件判断を行い表示する

SQL の中で場合わけを行うことができます。

【構文】① Select 列名 as 表示したい列名，列名 as 表示したい列名，
CASE 列名
WHEN 値式 THEN 値式
ELSE 値式
END
FROM 表名
② Select 列名 as 表示したい列名，列名 as 表示したい列名，
CASE
WHEN 探索条件 THEN 値式
ELSE 値式
END
FROM 表名

《例》表「商品」の列名 単価 の値が 300 円の商品は一律 10%の値上げを行い新単価とする。

Select 商品番号,商品名,単価, CASE 単価 WHEN 300 THEN 単価*1.1 ELSE 単価 END as 新単価 from 商品

【Exp 11】表「商品」の列名 商品番号 の値が 1000 番台のプリンタは一律 10%の値下げを行い値下げ単価として表示して下さい。

5.1.5 グループ処理（グループ集計）

集計関数（例えば SUM 関数等）は、ひとつの結果しか返さないが、部門ごとの集計等複数の結果を返すようにするための SQL 文があります。

【構文】 Group By 列名

《例》単純な集計

表「受注明細」の列名 受注番号 の値が 102 番の数量の合計と件数を表示する。

Select SUM(数量) as 受注番号 102 の数量合計,COUNT(*) as 明細件数
From 受注明細 Where 受注番号=102

《例》グループ毎の集計

表「受注明細」の列名 受注番号 毎に数量の合計と明細件数を表示する。

Select 受注番号,SUM(数量) as 数量合計,COUNT(*) as 明細件数
From 受注明細 GROUP By 受注番号

【Exp 12】 上記例を数量合計で昇順に並べて表示して下さい。

グループ化した項目に対する検索条件指定

【構文】 HAVING 条件

《例》上記例で数量合計が 5 より大きいものを表示する。

Select 受注番号,SUM(数量) as 数量合計,COUNT(*) as 明細件数
From 受注明細 GROUP By 受注番号
Having SUM(数量) > 5

【Exp 13】表「受注明細」の列名　受注番号　毎に数量の合計と明細件数を表示する際に
数量合計が 5 より大きいものを受注番号順に表示して下さい。

SELECT ステートメントの主な句の記述順を以下にまとめます。

```
SELECT select_list
[INTO new_table_]
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC] ]
```

5.2 表の結合

5.2.1 二つの表の結合（等結合：内部結合：INNER JOIN）

通常、データベースでは,複数の表を組み合わせて初めて見やすい表示形式になります。そこで、2 つのテーブルに共通のフィールドで関連付け、表を結合します。その結果、2 つの表が 1 つの表のイメージになるので、今までのように検索条件を指定できるようになります。

等結合は,最も一般的な結合で、Where 句に次のように書きます。

【構文】 Where 表名．列名 ＝ 表名．列名

《例》表「社員」と表「家族」 をお互いの表にある列名　社員コード　で結合して社員名、住所、家族コード、家族名を表示する。

```
Select 社員.社員名,社員.住所,家族.家族コード,家族.家族名
        from 社員,家族
        where 社員.社員コード = 家族.社員コード

Select 社員.社員名,社員.住所,家族.家族コード,家族.家族名
        from 社員 Inner Join 家族 on 社員.社員コード = 家族.社員コード
```

（列名（フィールド名）が各々の表で異なれば、表名は省略できます。）

5.2.2 二つの表の結合（外部結合：LEFT JOIN,RIGHT JOIN,FULL JOIN）

等結合では互いの結合列で合致するものだけを取り出していました。これは言い換えると一方の表には存在するが、他方には存在しないものは結果の表には現れないということになります。そこでどちらか一方の表にデータがあれば取り出す方法を考えてみます。

相手先の表に対応するレコードがあるなしにかかわらず、全てのレコードを抽出する結合を外部結合といいます。例えば「得意先マスター」と「売上」表において売上があるなしにかかわらず全得意先を抽出するといった場合に使います。

外部結合には、左右に結合する表のうち左にだけ存在するものを結びつける左外部結合（反対を右外部結合)、左と右の外部結合を両方含む全外部結合があります。

《例》表「社員」と表「家族」において、家族がいない社員についても社員コードで結合して、社員名、住所、家族コード、家族名を表示する。

【構文】 From 全データを取り出す表名 LEFT JOIN 相手方の表名
ON 結合条件

RIGHT
FULL

※これでは家族のいない山田一郎さんは表示されません

```
Select 社員.社員名,社員.住所,家族.家族コード,家族.家族名
        from 社員,家族
```

where 社員.社員コード = 家族.社員コード

※よって外部結合を使います。

Select 社員.社員名,社員.住所,家族.家族コード,家族.家族名
from 社員 LEFT Join 家族 on
社員.社員コード = 家族.社員コード

5.2.3 三つ以上の表の結合

where 句に and 演算子を用いて 3 つ以上の表を決合することも可能です。

《例》表「受注」と表「受注明細」と表「商品」において、受注番号と商品番号でそれぞれの表を結合して受注番号、受注明細番号順に、受注番号、受注明細番号、商品名、単価、数量、金額を表示する。

```
Select 受注.受注番号,受注明細.受注明細番号,商品.商品名,商品.単価,受注明細.数量,
    商品.単価*受注明細.数量 as 金額
    from 受注,受注明細,商品
    where 受注.受注番号 = 受注明細.受注番号 and
        受注明細.商品番号 = 商品.商品番号
        Order by 受注.受注番号,受注明細.受注明細番号
```

【Exp 14】表「受注」と表「受注明細」と表「商品」と表「顧客」において、受注番号と商品番号と顧客コードでそれぞれの表を結合して受注番号､受注明細番号順に、受注番号、顧客名、受注明細番号、商品名、単価、数量、金額を表示して下さい。

【Exp 15】上記の条件にさらに受注した社員の情報も担当者として顧客名の前に付け加えて表示して下さい。

5.3 データ登録・更新・削除

前節では、すでにあるデータの検索について基本的なことを説明しました。この節では、データの登録・更新・削除について説明します。

5.3.1 レコードの追加

新しいレコード（行）を追加するには、INSERT 文を使用します。

【構文】 INSERT INTO 表名 （列名 1，列名 2，……）
VALUES （値 1，値 2，……）

括弧の中に表の列名を記述します。次の VALUES 句の後の括弧の中に、先に指定した列の順番で値を記述します。指定していない列には、デフォルト値が指定してあればそのデフォルト値が、そうでなければ NULL が代入されます。値の記述で、文字列と日付はシングルコーテーション［‘］でくくります。NULL を禁止している列に値を指定しないと（NULL を代入する）INSERT 文は実行できません。

すべての列に値を代入する場合は，列名を省略できます。この場合、表の列の順序で値を記入しなければいけません。

【構文】 INSERT INTO 表名 VALUES （値 1，値 2，……）

《例》表「商品」に次のようなレコードを追加する。

INSERT INTO 商品(商品番号,商品名,単価)
VALUES(3001,’システム 1’,500)

次ぎの SQL 文を実行して、結果を確認してみましょう。

Select * From 商品

5.3.2 レコードの更新

すでにあるレコードの更新をするには、UPDATE 文を使用します。

【構文】 UPDATE 表名 SET 更新列名 1 ＝更新値，更新列名 2=更新値，……
Where 条件式

Where 句で条件が設定されていないと、SET で指定した列名を全て更新してしまいます。したがって、どのレコードを更新するかという指定が重要です。1 レコードだけ更新することも、複数レコードまとめて更新することも可能です。

《例》表「商品」のレコードを次のように変更する。

UPDATE 商品 **SET** 商品番号= 4001,商品名='ディスク X'
Where 商品番号=2001

この更新はうまくいきません。表「受注明細」の外部キー設定における参照制約で「更新：制限付」の定義がなされているためです。そこで「更新：波及」になっている表「社員」の社員コードを変更してみます。

UPDATE 社員 **SET** 社員コード= 'M004'
Where 社員コード='M002'

次ぎの SQL 文を実行して、結果を確認してみましょう。

Select * From 社員

更新が波及したテーブルの状態も確認してみます。

Select * From 家族
Select * From 受注

5.3.3 レコードの削除

レコードを削除するには、DELETE 文を使用します。

【構文】 DELETE FROM 表名 Where 条件式

Where 句で削除するレコードを指定する条件を指定します。条件にマッチする複数のレコードが削除されます。Where 句の指定がないと、全てのレコードが削除されます。注意が必要です。

《例》表「部門」の営業二課のデータを削除する。

DELETE FROM 部門 **Where** 部門コード='E02'

次ぎの SQL 文を実行して、結果を確認してみましょう。

Select * From 部門

波及する表「社員」の内容も合わせて確認しましょう（列名　部門コード　の該当列値が Null になっているはずです)。

Select * From 社員

5.4 トランザクション（データ制御言語 SQL-DDL）

今までの操作では、SQL が実行された瞬間に、データベースのテーブルに反映されていました。このままでは、システムの障害があったり、間違えた処理であったりした場合どこまで正しく実行されていて、どこからやり直せばいいかわかりません。データの一貫性と整合性を維持するための処理が、トランザクション処理です。

トランザクションとは、アプリケーションが行う一連の処理（Insert,Update,Delete 等）から構成される作業の論理単位です。わかりやすく言えば、いくつかの処理を一つのグループにまとめ、そのグループ単位で、処理を完結させるということです。

簡単な例を考えてみます。

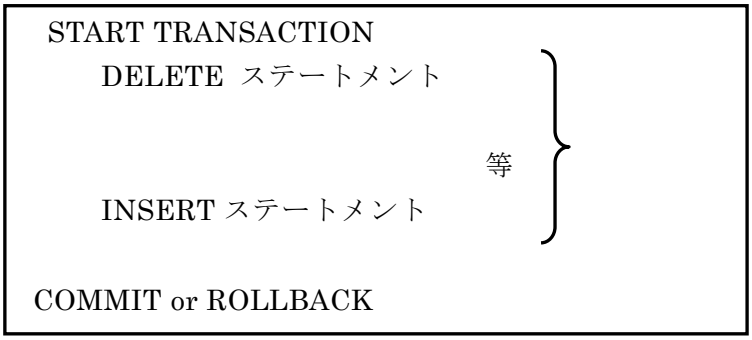
A さんが B 銀行から 5 万円を引き出し、C 銀行に振り込む処理は、

① B 銀行の A さんの口座から 5 万円引き出す。

② C 銀行の口座に振り込む。

これでこの処理が完結します。もし①と②の間で処理がストップしてしまった場合、B 銀行の口座から 5 万円は差し引かれています。しかし C 銀行には 5 万円が振り込まれていません。この場合、①と②の間で処理の区切りがついてはいけません。２つの処理が正しく行われたときは振り込み処理が正しく行われたと認識し、①と②の間で処理が異常に終了してしまったときは、振込み処理が失敗したとして、①の処理を一旦キャンセルし、再び①~実行するようにしなければいけません。こういった一連の処理をグループ化して考えることがトランザクションです。ここで一旦行った処理をキャンセルする命令が［ROLLBACK］、処理が正しく行われたとしてテーブルに書き込むの命令が［COMMIT］です。

また、①と②で一つのトランザクション（処理のグループ）であると明記しなければいけません。これが［START TRANSACTION］、［COMMIT］です。



こうすることにより、ROLLBACK を実行すると、COMMIT が発行されていない直前の［START TRANSACTION］まで処理がキャンセルされます。

いくつかの例で、COMMIT と ROLLBACK を見てみましょう。

【Exp 16】

INSERT INTO 部門(部門コード,部門名) **VALUES**('X01','テスト１');

Select * From 部門;

これは START TRANSACTION が記入されていないので、暗黙に COMMIT が発行されています。
そこで、明示的にトランザクションを開始してみます。次のように入力してください。

【Exp 17】

```
START TRANSACTION;
INSERT INTO 部門(部門コード,部門名) VALUES('X02','テスト２');
COMMIT;
Select * From 部門;
```

テスト２のデータが追加されています。
では、てもう１件レコードを追加しましょう。つぎの SQL 文を実行してください。

【Exp 18】

```
START TRANSACTION;
INSERT INTO 部門(部門コード,部門名) VALUES('X03','テスト３');
```

CSE のメッセージは以下のようになり正常に処理は終了しています。

```
1行が処理されました。
SQLの一括実行が完了しました。
```

しかし、 Select * From 部門; で部門テーブルを見ようとしても以下のようなエラーとなります。

```
SQLを実行中です...
SQL実行中に以下のエラーが発生しました。
ERROR:  現在のトランザクションがアボートしました。
トランザクションブロックが終わるまでコマンドは無視されます
```

COMMIT が記述されていないので、トランザクションがまだ終わっていないと理解し、INSERT 処理が確定されていません。すなわちメモリの中だけに記録されている状態なので混乱を防ぐためにデータの参照をできなくしています。
ここで、間違った処理だったということで、ROLLBACK を実行しましょう。

【Exp 19】

```
ROLLBACK;
Select * From 部門;
```

データはどこまで戻ったでしょうか？テスト３のレコードが消えています。
もう一度

```
INSERT INTO 部門(部門コード,部門名) VALUES('X03','テスト３');
COMMIT;
```

を行ってください。最後の COMMIT まで、更新は進みます。１つのトランザクションは、［START TRANSACTION］から［COMMIT,ROLLBACK ］までです。

これらの処理は、実行される前にトランザクションログファイルに書き込まれます。処理の途中、何らかの理由で Data Base Server が異常停止した場合などに、次回の起動時に Data Base Server はこのトランザクションログを利用して、コミット済みはロールフォワード、コミットされていないものはロールバック作業を行って、データの整合性を保証する作業を行うこともできます。

5.5 ロック（データ制御言語 SQL-DDL）

複数のユーザが一つのデータを操作しようとしたときに起こりうるデータの不整合を回避する機能がロック機能です。

ロック機能について先ほどのように簡単な例で説明します。

A さんが B 銀行の預金から１０万円引き出す例で考えてみましょう。

① A さんは５０万円の預金がある

② A さんは１０万円引き出している

③ ５０万円から１０万円引いた４０万円を新たに記帳する。

このようなプロセスでデータを書き込んでいきます。

②のタイミングで、C さんが A さんの口座に５万円振り込んだとします。５０万円の預金がありましたから、③の時点で４５万円残高がなくてはいけません。しかし、１０万円引き出しのプロセスでは、４０万円と書き込みます。このままでは、５万円の振込みが失われてしまいます。５万円振込みプロセスは、１０万円振り込みプロセスが終了した後実行されなければなりません。このように、あるプロセスの実行途中に入った他のプロセスは、そのプロセスが終了するまで待って、終了した後実行されなければなりません。これがロック機能です。

Data Base Server のロックは、データベース内で、さまざまなレベルの粒度で適用されます。たとえば、行、ページ、キー、キー範囲、インデックス、テーブル、またはデータベース単位でロックをかけることができます。性能上最も問題になるのはテーブルに対するロックです。テーブルの中の行にアクセスする際、テーブル全体をロックする方法をテーブルロックと呼びます。それに対してアクセスする対象の行のみをロックする方式を行ロックと呼びます。当然ながら行ロックのほうがロックが衝突する確率が低く性能が向上します。

ロックされる状態を確認してみましょう。

環境準備

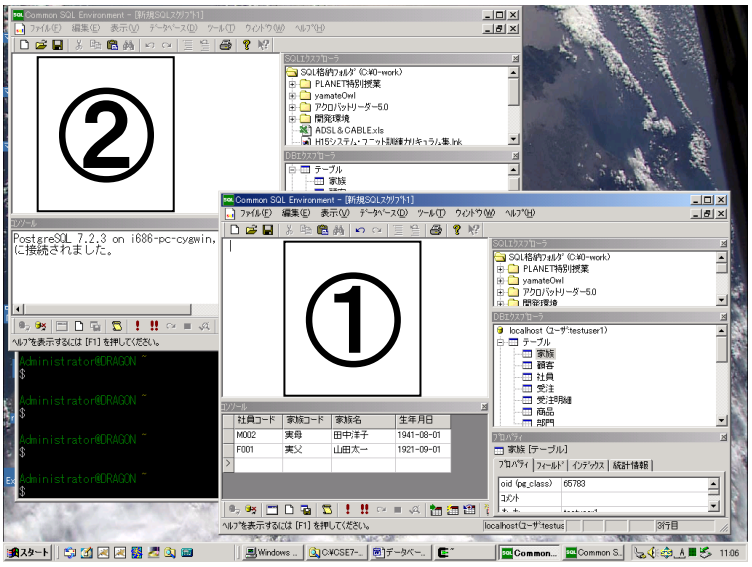
(1) postgres ユーザで接続している状態で以下の SQL を発行して user1 と user2 を作成します。

```
create user "user1" with password 'user1';
create user "user2" with password 'user2';
```

(2) postgres ユーザで接続している状態で以下の SQL を発行して user1 と user2 に表「部門」にアクセスする権限を与えます。

```
grant all on 部門 to "user1";
grant all on 部門 to "user2";
```

(3) 同時に一つの表のデータを変更するという環境にするために、CSE を user1 と user2 で２つ起動してください。これで、２台のクライアントから、データ変更トランザクションを実行する環境になります。右図のように、２つの画面が見えるようにしてください。



どちらか一方の画面（ここでは①とする）で表「部門」 のテスト１のデータを次のようにに変更すします。

【Exp 20】①で実行

```
Start Transaction;
UPDATE 部門
        SET 部門名='テスト X'
        Where 部門コード='X01';
```

commit は記述しない。ということは、まだトランザクションの途中ということになります。この状態で、もう一方の画面（②とする）で以下を実行してみます。

【Exp 21】②で実行

```
Select * From 部門;
UPDATE 部門
        SET 部門名='テスト Y'
        Where 部門コード='X01';
Select * From 部門;
```

SQLを実行中です...

処理が止まってしまいます。

①側では COMMIT が実行されていないため、トランザクションが終了していません。そのためデータに対して、ロックがかかっているので、②側から UPDATE のデータ操作ができないのです（待ち状態になる）。

【Exp 22】

ここで、①に COMMIT を実行してみてください。

実行した瞬間、トランザクションが終わったので、ロックが解除され、②の UPDATE 文以下が実行されます。最終的には部門コード「X01」の部門名は「テスト Y」となります。このように、Data Base Server ではロック機能をシステムが管理し、データの不整合を回避しています。

6. データベース設計手法

6.1.1. 正規化とは

ER モデリング（ERD の作成）を行うに際して正規化手法からアプローチを行うことができます。正規化とは、与えられたデータの関係を分析し、相互の関連性を維持した冗長性のないデータの集合に分解し、整理することです。これにより、基本的なデータ項目グループの識別、同一属性を複数箇所で保持するといった冗長性の排除、データの整合性の維持をはかることができます。

6.1.2. 正規化の手順

(1) 帳票収集

注文書　　担当部署：外商１部　　**注文番号**：1234

担当者　：山口太郎　　注文日　：2001/05/30

お客様名　田中一郎

ご住所　〒231-0854　横浜市中区山手 1-5-36

商品番号	商品名	単価	数量	金額
G0013	ハイドロカルチャー	¥300	10	¥3,000
P1021	グリーンポッド	¥580	10	¥5,800
T0011	ベンジャミン	¥5,000	5	¥25,000

(2)非正規形

注文書＝注文番号＋顧客名＋住所＋注文年月日＋担当部署＋担当者
＋｛商品番号＋商品名＋単価＋数量＋金額｝

｛｝は繰り返しを意味する。　＊バッカス記法に準じている

＿＿＿＿は主キーを表す。

● 非正規形

注文書＝注文番号＋顧客名＋住所＋注文年月日＋担当部署＋担当者
　　＋｛商品番号＋商品名＋単価＋数量＋金額｝

※注文書を一意に特定する項目は何？下線を引く

⑶第一正規化

繰り返し、配列、構造体をなくしたフラットなフィールド構成にする。

第一正規化することで、非正規形の注文書の繰り返し部分を切り離し、次のような２つのテーブルに分割する。

注文書＝注文番号＋顧客名＋住所＋注文年月日＋担当部署＋担当者
注文明細＝注文番号＋商品番号＋商品名＋単価＋数量＋金額

● 第１正規形

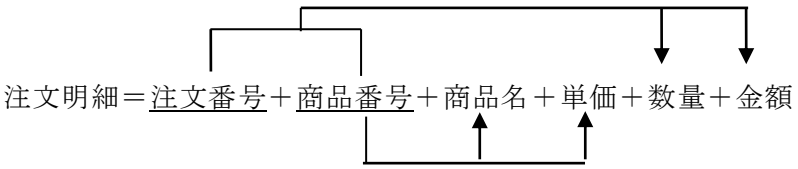
注文書＝注文番号＋顧客名＋住所＋注文年月日＋担当部署＋担当者
注文明細＝注文番号＋商品番号＋商品名＋単価＋数量＋金額

※繰り返しを分離したら親（元）のキーを付加すること。

※かつ子を一意に特定するキーを付加する（項目を増やす場合もある）。

⑷第二正規化

主キーが複数のフィールドから構成される場合、キー以外のフィールドが主キーを構成するフィールドのいずれかのみに従属する関係があれば、その関係を分離し、テーブルを分割する。



また、金額のような導出項目（金額＝単価×数量）があれば、削除する。

注文書明細＝注文番号＋商品番号＋数量
商品＝商品番号＋商品名＋単価

● 第２正規形

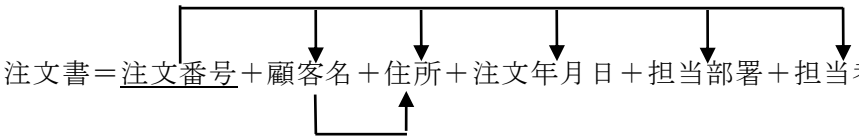
注文書＝注文番号＋顧客名＋住所＋注文年月日＋担当部署＋担当者
注文明細＝注文番号＋商品番号~~＋商品名＋単価~~＋数量~~＋金額~~
商品=商品番号＋商品名＋単価

※キーに関連する項目を切り出して親から消す（親のキーは残すこと）。

※導出項目を削除する（計算で算出される項目）。

⑸第三正規化

主キー以外のフィールド間で従属関係がある場合、その関係を分離する。



注文書＝注文番号＋顧客番号＋注文年月日＋担当部署＋担当者
顧客＝顧客番号＋顧客名＋住所

● 第３正規形

注文書＝注文番号＋~~顧客名＋住所~~＋顧客番号＋注文年月日~~＋担当部署＋担当者~~＋
社員番号
注文明細＝注文番号＋商品番号＋数量
商品=商品番号＋商品名＋単価
顧客＝顧客番号＋顧客名＋住所
担当＝社員番号＋担当者＋担当部署

※主キー以外の関連項目を切り出して、子の主キーで親とリンクする。

⑹再考

最終的には次のようなテーブル設計になります。

注文書＝注文番号＋顧客番号＋注文年月日＋担当部署＋担当者

注文書明細＝注文番号＋商品番号＋数量

顧客＝顧客番号＋顧客名＋住所

商品＝商品番号＋商品名＋単価

正規化は第一正規形から第五正規形までの５ステップあります。リレーショナルデータベースの生みの親とも言える E.F.Ｃｏｄｄ氏が提唱しました。実用的には５つのステップをすべてチェックしていかなくても、第三正規形までチェックしておけば、ほとんどの場合、各テーブルは第五正規形になっているはずです。実用例において第三正規形で、第四正規形、第５正規形でない例を探すのが難しいくらいです。ただし、この段階ではあくまでも正規化という作業を行っただけで、物理的なデータベースを設計したわけではありません。実務上は、パフォーマンスを考えてテーブルをあえてくっつけたり（同時に使用する頻度の高いフィールドをまとめる）、導出項目をフィールドにしたり（単価の変更を考慮して金額フィールドを設ける）する必要があります。

6.2. ERD（Entity Relationship Diagram）

○ＥＲ図（エンティティーリレーションシップ ダイアグラム ＥＲＤ）

ＥＲＤは、データ中心アプローチのもっとも大切な道具です。これを利用すれば、対象となるデータをエンティティ（実体・管理対象）を四角い箱、リレーション（関係・関連）を線として、視覚的、直感的に表現できます。

ＥＲＤによるモデリングというのは、日常システム設計に関わっている限りでは聞き慣れない言葉かもしれませんが、「意味データモデル」と呼ばれる理論です。これはネットワークモデルのように集合間の関係を数学的なルールを適用して表現するモデルとは異なり、実世界の意味上の関係を中心に表現するものです。

ＥＲＤを理解するにはエンティティ及びリレーションシップ（関連）という二つの概念を理解する必要があります。

6.2.1 ＥＲＤの構成要素

○エンティティ　（実体）

エンティティとは、管理対象となるもののことです。エンティティ抽出の要件のめやすは、以下のように考えます。

・業務の管理対象となるもの。物理的なものだけではなく、抽象的なものイベントも含まれるが永続的に存在するもの。

・名詞で表現できるもの。

・固有の属性を持つもの。

・エンティティ・オカレンスが、複数存在するもの。

・エンティティ・オカレンスを一意で識別できるキーをもつもの。

[例]
人・・・従業員、顧客、契約者　　事象・・・契約、作業　　金銭・・・口座、借入
モノ・・部品、商品、支店　　概念・・・計画、時間、評価

エンティティとは、物理的なものであれ、抽象的なものであれシステムの管理対象の基礎になる項目の塊です。「実体」と訳されることもあります。例えば、従業員、顧客、商品、支店などが考えられます。エンティティの持つ本質的な意味には、「エンティティ・タイプ（実態物）」と「エンティティ・オカレンス（実態実現値）」の 2 種類があります。

エンティティ・タイプとはエンティティの種類のことであり、部門、従業員、プロジェクトなどが上げられます。これらはそれぞれ、複数の部門、複数の従業員などの構成要素から成り立っているので集合と考えることができます。

エンティティ･オカレンスとはエンティティに繰り返し現れてくるデータのことであり、部門、従業員などの構成要素がこれに該当します。

ひどく単純な言い方をすれば

従業員は、エンティティ・タイプで、山田一郎は、エンティティ・オカレンスです。もっと解りやすく言えば、「エンティティ」はテーブル、「エンティティ・タイプ」はテーブル名、「エンティティ・オカレンス」はテーブルのロウ（Row：行）です。

エンティティにはプロパティ（性質）があり、このプロパティを示すものがアトリビュート（属性）です。したがって各エンティティはエンティティの性質を表す複数の属性で構成されていると考えることができます。データ項目の別名と理解しても言いでしょう。

［例］

従業員というエンティティには、氏名、生年月日、所属部門、入社年月日、などの属性があると考えられます。

エンティティの属性のうち実際のデータを一意に定めることのできる属性をキー（識別子）といいます。エンティティには必ず主キーが存在しなければなりません。

キーの種類

１．主キー、プライマリ・キー Primary key
　　前述のキーがそれに当たります。Ｎｕｌｌは認められません。

２．複合キー Compound key （Composite key）
　　単独のアトリビュートでキーを表現できない場合、二つ以上のアトリビュートでキーを構成することができます。このような場合、構成する複数のアトリビュートのことを複合キーと言います。

３．外部キー Foreign key
　　エンティティが他のエンティティを利用・引用したい場合、利用したい他のエンティティのキーを自らのアトリビュートとして取り込みます。このようなアトリビュートを外部キーと言います。参照のための接点の役目をするので参照キーとも言います。

独立エンティティ　[例]　部門、社員、商品、顧客、など
それ自体が他のエンティティとは独立した存在と考えられるもの。

従属エンティティ　[例]　受注明細、従業員に対する家族、など
他のエンティティを説明・特徴付けるもの。
他のエンティティがあって初めて存在しえるもの。

関連エンティティ　[例]　受注明細、共著、貸し出し履歴、など
2 つのエンティティ間に多：多の関係があるときに、それらの関係を表すもの。

サブタイプ　（エンティティ・サブタイプ）

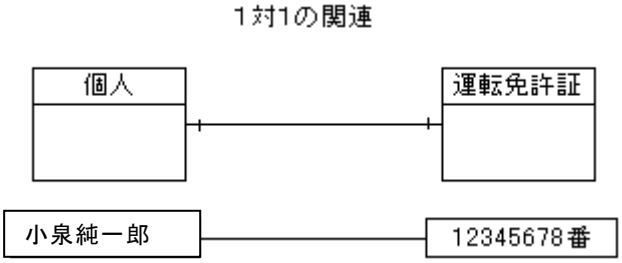
エンティティをさらに詳細に分類したものです。例えば、顧客と言うエンティティに対し、個人顧客と法人顧客の２つの分類がある場合個人顧客、法人顧客をサブタイプと言います。各サブタイプは、それぞれ、独自のアトリビュートを持つことが出来ます。また、上位の共通部分はスーパータイプといいます。

○リレーションシップ　（関連）

エンティティ間の結びつきです。「関連」「関係」と訳されます。エンティティ間の関連には、１対１、１対多、多対多の３種類が存在します。この１や多のことを「カーディナリティ」または、「基数制約」と呼びます。

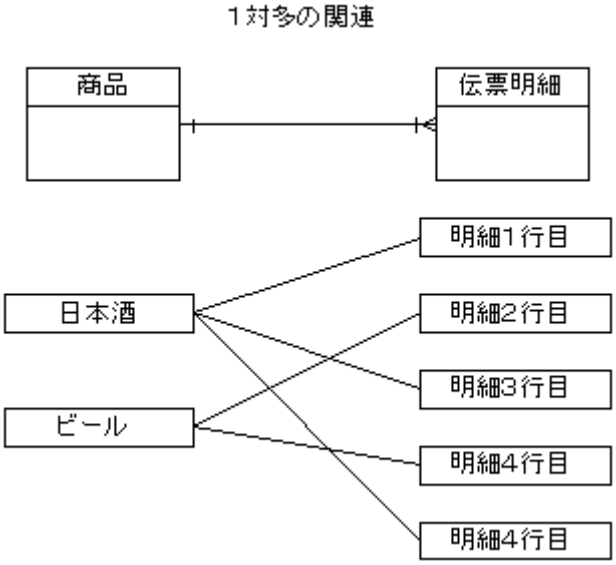
１対１の関連

エンティティ・オカレンス間の関係が１対１の対応関係を言います。例としては海外旅行者とパスポートの関係があります。海外旅行者はパスポートを 1 冊しかもてませんし、１冊のパスポートの所有者は海外旅行者一人だけです。また小泉純一郎=12345678（運転免許証番号）はどうでしょうか。これも、それ以外の関係は存在し得ないので１対１の関連になります。ただし、１対１はもともと同一のエンティティである場合が多いので、よく見極める必要があります。



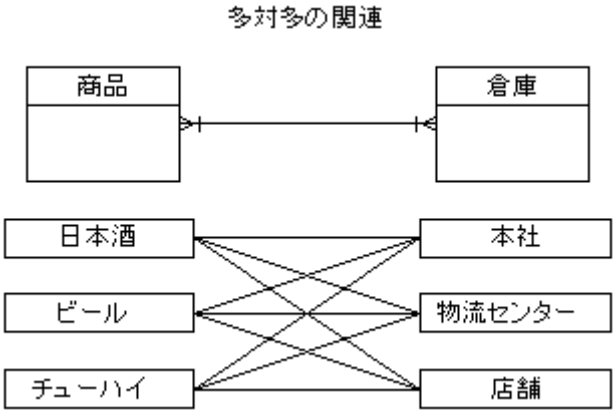
１対多の関連

エンティティ・オカレンス間の関係が１対複数の対応関係を言います。「伝票」と「明細行」、「明細行」と「商品」などが、典型的な例になります。これこそが、データベースの主役になる関連です。

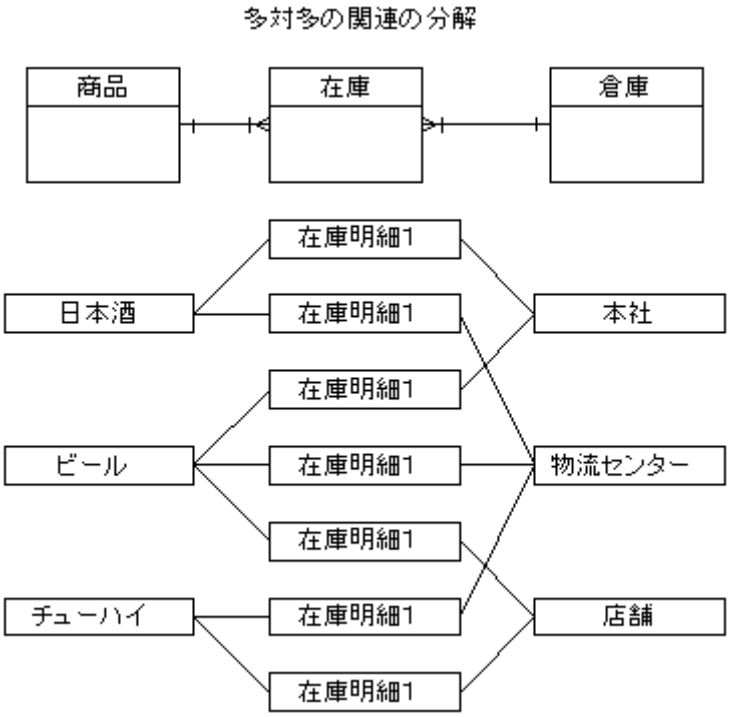


多対多の関連

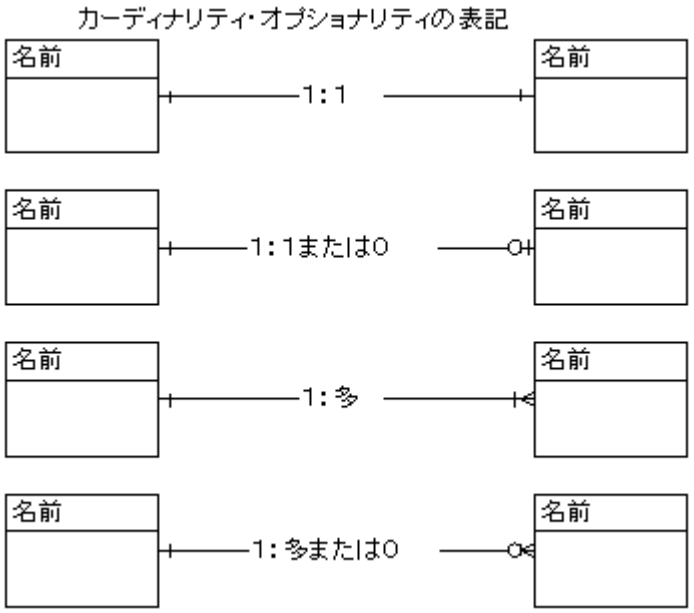
エンティティ・オカレンス間の関係が複数対複数の対応関係を言います。実際の世界ではこのような関連はたくさんありますが、データベースでは管理できないので、データモデルの世界では、このような関連は認められません。



多対多の連関がある場合は、エンティティの間に関連付けのための抽象的なエンティティを生成し３つのエンティティ間の２つの１対多の関連にしてデータベースで管理できる形にモデル化します。このように中間に生成されるエンティティを「関連エンティティ」と呼びます。



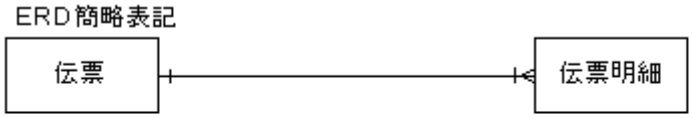
リレーションの種類を鳥脚図法で表現すると以下のようになります。

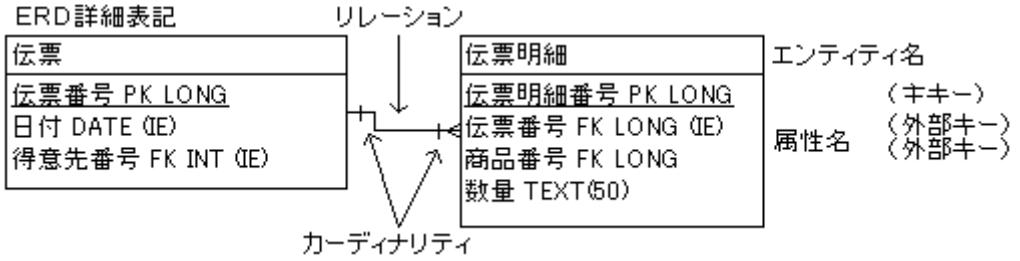


さらに、エンティティ間の結びつきに関して「存在依存制約」という考え方があります。これは、一方のエンティティ・オカレンスの存在が、もう一方のエンティティ・オカレンスの存在に依存するかどうかの関係のことです。

[例] 部門と従業員のエンティティについて考えてみます。
「部門を削除した場合にはそこに所属している従業員も削除され、従業員を追加する場合には必ず所属する部門情報が存在していなければならない」としたとき、この二つのエンティティには存在依存の関連がある、といいます。

6.2.2 ＥＲＤの具体例（鳥脚図法）



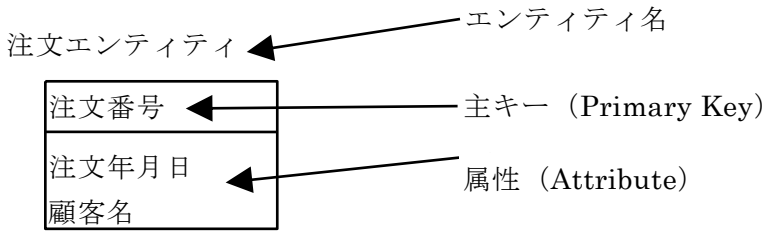


6.2.3 IDEF1X（Integration DEFinition for information modeling 1 eXtended）

IDEF1X のエンティティとエンティティの種類

エンティティの種類は、ＥＲＤの表記法によりいろいろあります。ここではＦＩＰＳ（米連邦情報処理規格）として認定されているＩＤＥＦ１Ｘをベースに説明します。

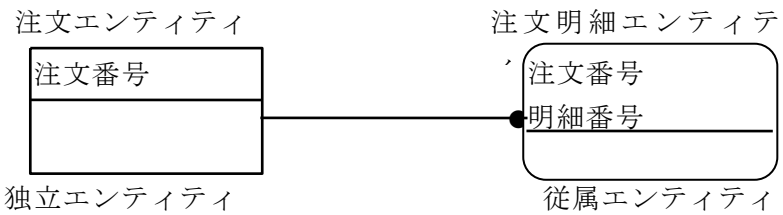
①エンティティの表記法



②独立エンティティと従属エンティティ

独立エンティティとは、データモデル内において他のエンティティに依存しないエンティティのことです。

従属エンティティとは、その存在と識別をデータモデル内において他のエンティティに従属するエンティティのことです。

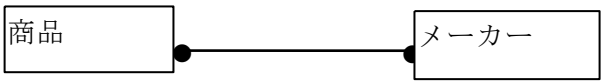


③関連付けエンティティ

関連付けエンティティとは、２つまたはそれ以上のエンティティ間の複数のリレーションシップから作成されるエンティティです。

例えば、あるスーパーで販売している商品とメーカーの関係は、多対多で表せます。つまり商品Ａは複数のメーカーから仕入れ、また、メーカーは複数の商品を納品しているという関係です。

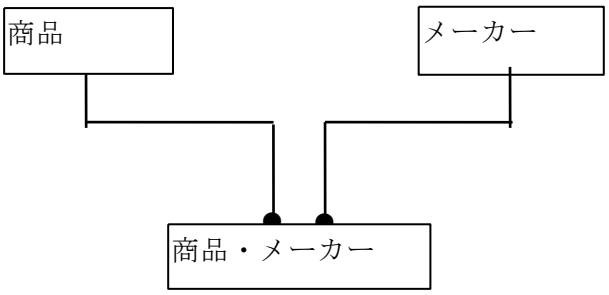
多対多の関係



この場合、メーカー別商品の在庫はどのエンティティに格納すればよいでしょうか。商品エンティティでは、メーカー毎の情報は保有できません。また、メーカーエンティティでは、商品毎の情報は保有できません。

このためには、主キーが商品コード、メーカーコードである商品・メーカーエンティティを作り、そこにメーカー別商品の在庫の情報を格納します。これを関連エンティティといいます。

関連エンティティ



④スーパータイプとサブタイプ

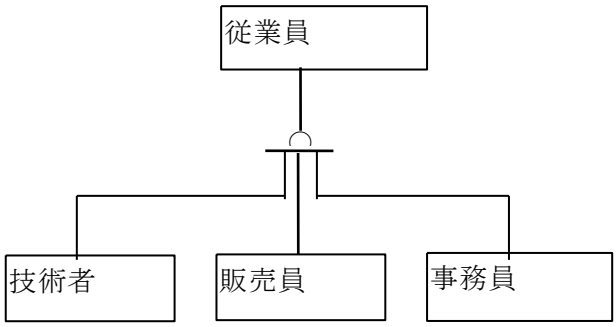
スーパータイプとサブタイプは、共通の特性を持った一連のエンティティをグループ化する概念です。

例えば、人事システムにおいて従業員と技術者、販売員、事務員のエンティティを考える場合、社員番号以外に性別、生年月日、入社年月日、最終出身校、所属組織等の共通の属性と技術資格、販売用社用車保有区分等の個別の属性の扱いが問題とります。

従業員エンティティは、技術者エンティティ、販売員エンティティ、事務員エンティティの上位のエンティティと位置付けられ、共通の属性を代表的に持ちます。また、技術者

エンティティ、販売員エンティティ、事務員エンティティは、従業員エンティティの下位のエンティティと位置付けられ、それぞれのエンティティは個別の属性を持ちます。

この上位のエンティティをスーパータイプといい、下位のエンティティをサブタイプといいます。スーパータイプとサブタイプでできる階層を汎化階層といいます。



【重要】

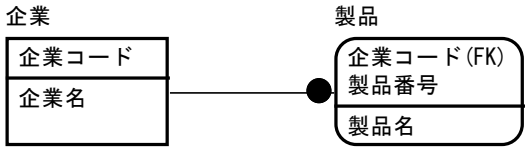
ERD にを用いたモデリングの基本は以下の 2 つのパターンです。

- 基本的な 1 ：N の関係を組み合わせた形
- M：N を分割し関連エンティティを配置した形

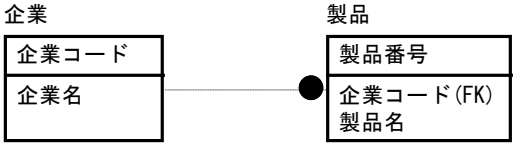
基本的な 1 ：N の関係には依存・非依存の関係がありますので、まとめると以下のような 3 つのパターンになります。このパターンをビジネスモデルを読み取りながら適宜に配置していけば、複雑なデータ項目をすっきりと表現することができるはずです。

依存と非依存関係の例

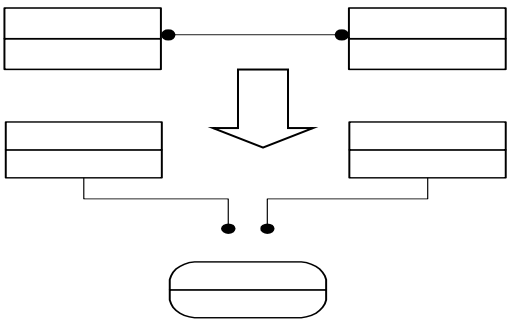
・依存関係　　子エンティティの主キーにキー属性「企業コード(FK)」を入れる



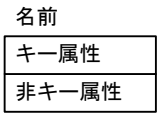
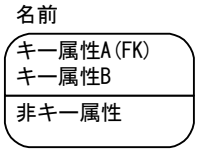
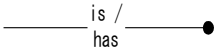
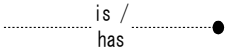
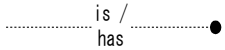
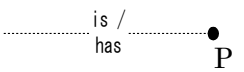
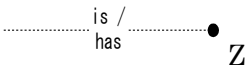
・非依存関係　親エンティティのキー属性は子エンティティの主キーに入れない

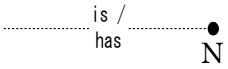
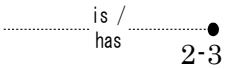
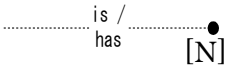
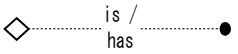
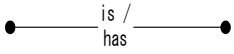


・多:多 カーディナリティの表現例



【参考】IDEF1X の規約

名　　称	意　　味	規　　則
エンティティ [entity] （親エンティティ）	記録したい補助情報についてのオブジェクト	
子エンティティ [child entity]	他のエンティティからその名前を引き出すエンティティ	
依存ﾘﾚｰｼｮﾝ [Identifying relationship]	エンティティ間に同一視関係(identifier-dependent)があるとき表すオブジェクト（子エンティティの存在は、親エンティティの存在なくしてはない）	
非依存ﾘﾚｰｼｮﾝ [Non-Identifying relationship]	エンティティ間に同一視関係(identifier-independent) がないとき表すオブジェクト（子エンティティの存在は、親エンティティが存在しなくてもよい）	
カーディナリティ	依存／非依存リレーションの特性 ①1: 0～N カーディナリティ制約 各親エンティティはゼロまたは それ以上の子エンティティと関係 を持つ	
	②1: 1～N カーディナリティ制約 各親エンティティは１またはそれ 以上の子エンティティと関係を持つ	
	③1: 0～1 カーディナリティ制約 各親エンティティはゼロまたは １つの子エンティティと係を持つ	

名　　称	意　　味	規　　則
カーディナリティ（続き）	④1: N カーディナリティ制約 N 個のの子エンティティと関係を持つ	 （N は定数）
	⑤1: N カーディナリティ制約 2 個から 3 個の子エンティティと関係を持つ	
	⑥1: [N] カーディナリティ制約 1 から注[N]の子エンティティと関係を持つ	
	⑦オプショナルカーディナリティ制約 子エンティティは関係する親エンティティを持たなくてよい	
	⑧多:多 カーディナリティ制約 個々の親エンティティはゼロまたはそれ以上の子エンティティと関係を持ち、そしてその逆の関係も持つ	
不完全カテゴリ [Incomplete category]	親エンティティがサブタイプとして子エンティティを持つ、すらわち他のサブタイプが存在する可能性有	
完全カテゴリ [Complete category]	親エンティティがサブタイプとして子エンティティを持ち、この子エンティティは完全なカテゴリセットが確定している	

(1) IDEF1 規約　　http://www.idef.com/downloads/Downloads.htm