環境情報技術者
Chapter11
データベース

担当：新田晃啓

DBMS（Data Base Management System）

データベース管理システム

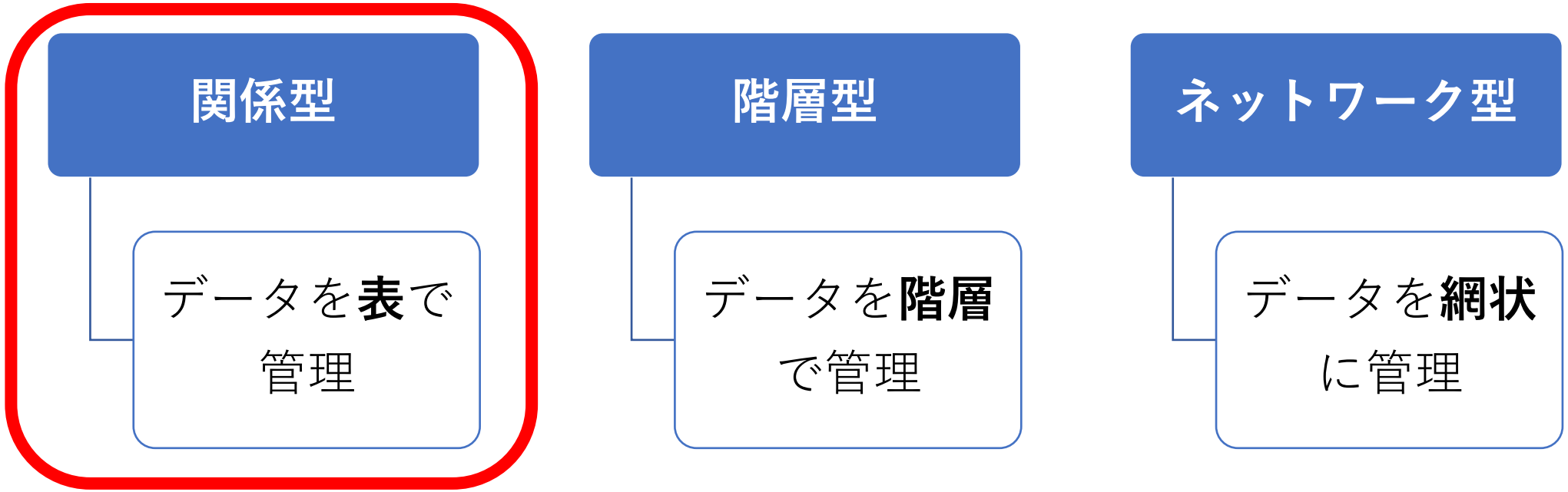
データベースの定義や操作、制御などの機能を持つミドルウェア

データベース内のデータの検索や追加、更新削除などを行うほか、以下の機能がある。

機能	概要
保全機能	参照制約や排他制御など、データの完全性を保つ機能
障害回復機能	ロールフォワードやロールバックなど、データベースの障害を回復する機能
機密保護機能	ユーザ認証やアクセス制御など、データの改ざんや漏洩を未然に防ぐ機能

データモデル

データベースを設計する際に、実世界におけるデータの集合をデータベース上で利用可能にするもの



関係データベース

行と**列**から構成される**2次元の表**で表現されるデータベース

関係データベース（RDB: Relational Database）

表の形でデータを管理するデータベース

定義域
属性が取り得る値の集合
整数型, 文字列型など

表　テーブル

受給者番号	氏名	日付	開始時間	終了時間
1111111111	田中〇〇	6/25	10:00	16:00
2222222222	高橋△△	6/27	10:00	13:00
3333333333	佐藤□□	6/27	13:00	16:00
1111111111	田中〇〇	6/27	10:00	15:00
4444444444	高橋××	6/28	10:00	13:00

行　レコード

列　フィールド　属性

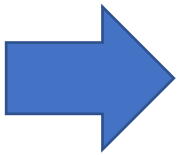
関係演算

選択(Projection)

行を取り出す演算。特定の条件に合致する行だけ抽出することができる。

社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009003	佐藤三郎	2
2009004	鈴木四郎	3
2009005	山田五郎	2

部署IDが2の行だけ取り出す



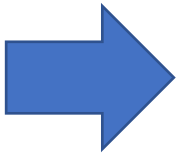
社員番号	氏名	部署ID
2009002	高橋二郎	2
2009003	佐藤三郎	2
2009005	山田五郎	2

射影(Selection)

列を取り出す演算。特定の条件に合致する列だけ抽出することができる。

社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009003	佐藤三郎	2
2009004	鈴木四郎	3
2009005	山田五郎	2

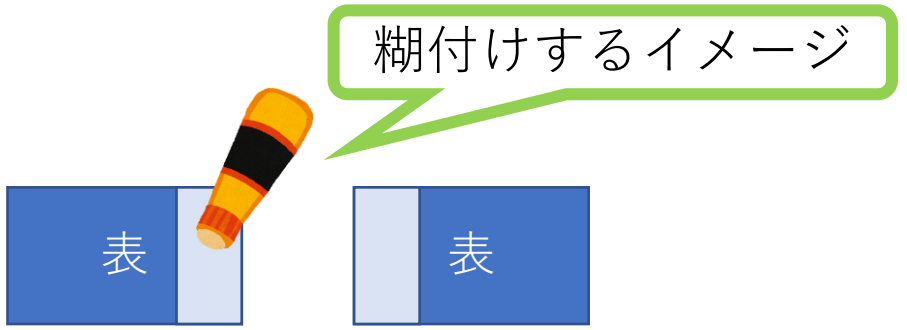
社員番号と氏名の列だけ取り出す



社員番号	氏名
2009001	田中一郎
2009002	高橋二郎
2009003	佐藤三郎
2009004	鈴木四郎
2009005	山田五郎

結合(JOIN)

表と表をくっつける演算。表の中にある**共通の列を介して**2つの表をつなぎあわせる。



社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009003	佐藤三郎	2
2009004	鈴木四郎	3

部署ID	部署名
1	営業部
2	開発部
3	経理部

部署IDを介してくっつける

社員番号	氏名	部署ID	部署名
2009001	田中一郎	1	営業部
2009002	高橋二郎	2	開発部
2009003	佐藤三郎	2	開発部
2009004	鈴木四郎	3	経理部

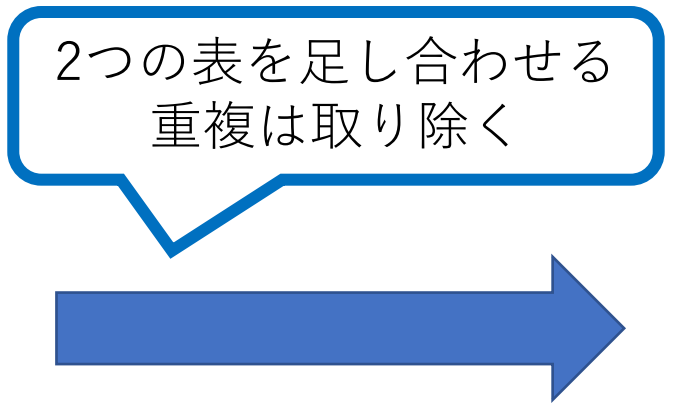
集合演算

和

共通の列をもつ**2つの表の行を結合して**つなぎ合わせる。
同じ行は1つにまとめる。

社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009004	鈴木四郎	3

社員番号	氏名	部署ID
2009001	田中一郎	1
2009003	佐藤三郎	2
2009005	山田五郎	2



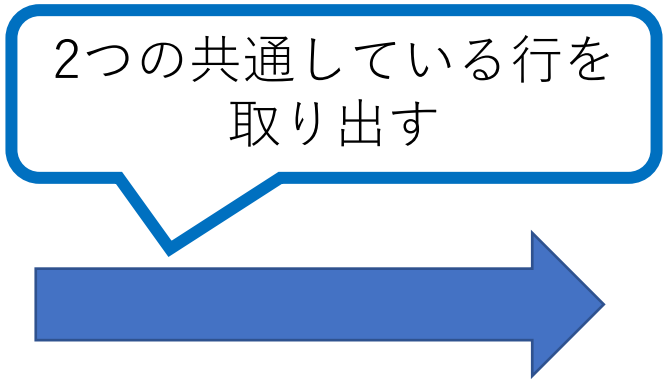
社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009003	佐藤三郎	2
2009004	鈴木四郎	3
2009005	山田五郎	2

積

２つの表に共通している行を取り出す。

社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009004	鈴木四郎	3

社員番号	氏名	部署ID
2009001	田中一郎	1
2009003	佐藤三郎	2
2009005	山田五郎	2



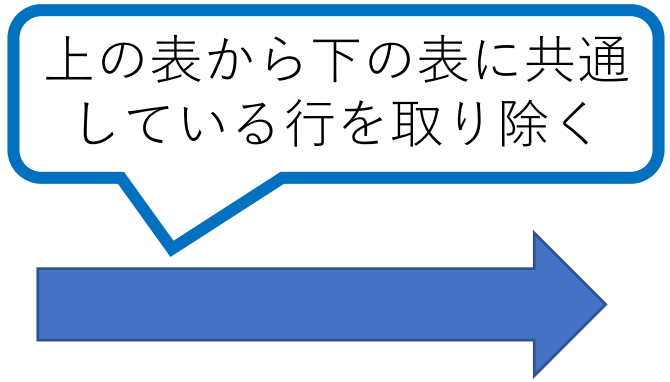
社員番号	氏名	部署ID
2009001	田中一郎	1

差

一方の表から**他方の表**を**取り除く**。

社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009004	鈴木四郎	3

社員番号	氏名	部署ID
2009001	田中一郎	1
2009003	佐藤三郎	2
2009005	山田五郎	2



社員番号	氏名	部署ID
2009002	高橋二郎	2
2009004	鈴木四郎	3

ビュー表

- 実際に存在する実表から必要な部分を取り出して、**一時的に作成した表**。
- **外部スキーマ**に相当する。
- 磁気ディスクには存在しない**仮想表（導出表）**。

社員番号	氏名	部署ID	部署名
2009001	田中一郎	1	営業部
2009002	高橋二郎	2	開発部
2009003	佐藤三郎	2	開発部
2009004	鈴木四郎	3	経理部
2009005	山田五郎	2	開発部

社員番号	氏名
2009001	田中一郎
2009002	高橋二郎
2009003	佐藤三郎
2009004	鈴木四郎
2009005	山田五郎

関係演算を用いるとデータベースから様々なビューを作成できる。

スキーマ

データの形式や性質、ほかのデータとの関連などのデータ定義の集合

外向き

外部スキーマ

- 利用者の必要とするデータの見方を表現
- ビュー表　SQLによる問い合わせ

概念スキーマ

- データの論理的関係を表現
- 表の定義、データベース本体

内部スキーマ

- データの物理的関係を表現
- 物理的な記憶（ハードウェアなど）の定義

内向き

3層スキーマ構造

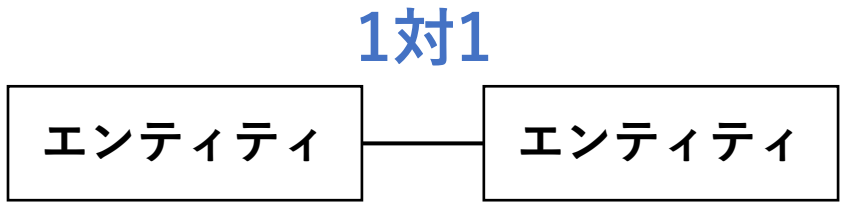
定義を分けることで
データの独立性を
高めている。

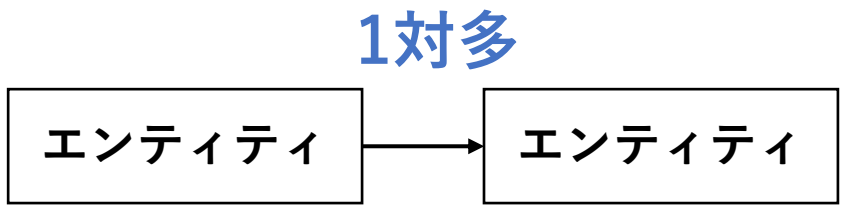
例えば
データの物理的な格納機能
構造を変更してもアプリ
ケーションには影響がない

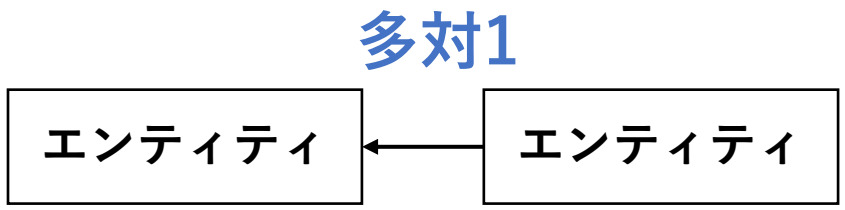
データベース設計

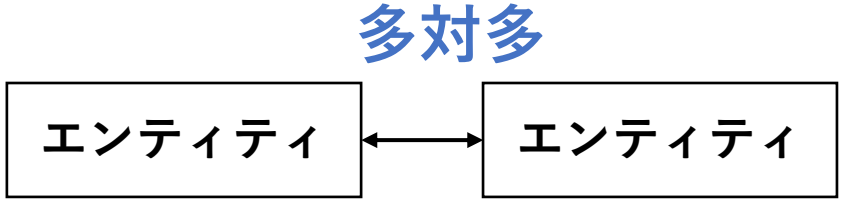
E-R図(Entity-Relationship Diagram)

- 対象業務を構成する**実体**(人・モノ・場所・事業など）と**実体間の関連**を視覚的に表した図。
- 実体を長方形の箱で表し、各実体の関係を矢印で表す。
- 実体を**エンティティ**(Entity)、実体の関連を**リレーションシップ**(Relationship)と呼ぶ。

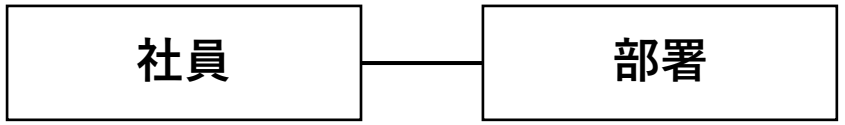






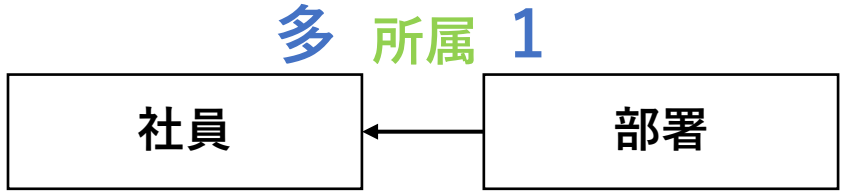


例）社員が部署に所属する

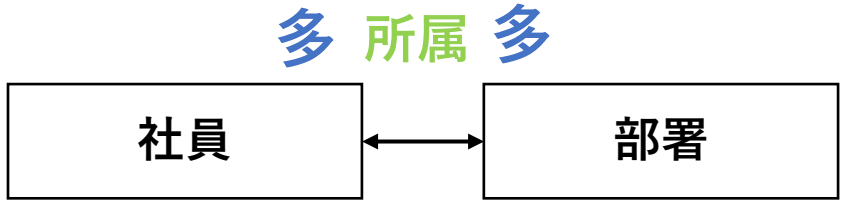


社員から部署の方向　部署から社員の方向を考える

例）社員50人は部署は5つ。社員一人は1部署に必ず所属するとき

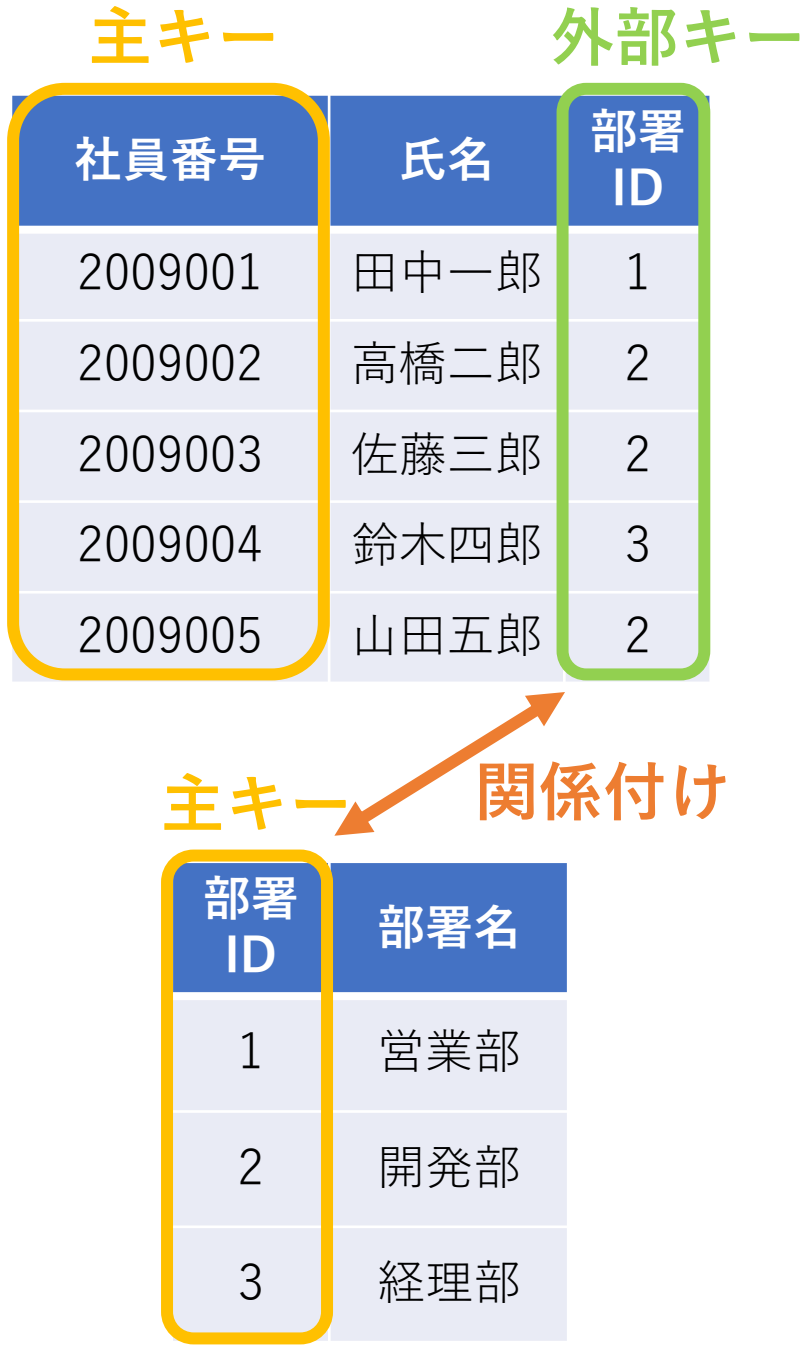


例）社員50人は部署は5つ。社員一人は複数の部署に所属することがあるとき



表の設計

- 主キー
  - 表の中で各行を識別するために使う列。
    例）社員番号、部署ID、商品コード
  - 表の中で**重複しない。（一意制約）**
  - 内容が**空でない。（非NULL制約）**
  - 複数列を組み合わせて主キーとしたものを**複合キー**という。
- 外部キー
  - **他の表の主キーを参照**する列、
  - 表と表を関係づけることができる。
  - 親テーブルにないデータを子テーブルが持つことないようにする。**(参照制約)**



正規化

データに矛盾や重複が生じないように、表を最適化(分割）する

非正規形

受注No	日付	顧客名	商品	数量	商品	数量	商品	数量

繰り返しの部分があるため、**各レコードの長さがバラバラ**な状態。関係データベースでは管理することができない。

第1正規形

受注No	日付	顧客名	商品	数量

非正規形から**繰り返しの部分を取り除いた**もの。
素直な2次元の表。

関数従属と部分関数従属

- 主キーが決まれば、列の値が一意に定まる関係

主キー

社員番号	名前	部署
20220001	田中一郎	営業部

- 複合キーの一部の項目だけで、列の値が一意に定まる関係

主キー(複合キー)　　主キー(複合キー)

受注NO	日時	顧客コード	顧客名	商品コード	商品名	単価	数量
20220001	2022/7/1	C001	AA出版	B107	ファイル	50	10

第2正規形

非正規形
正規化
第1正規形
正規化
第2正規形
正規化
第3正規形

主キー(複合キー)　　主キー(複合キー)

受注NO	日時	顧客コード	顧客名	商品コード	商品名	単価	数量
20220001	2022/7/1	C001	AA出版	B107	ファイル	50	10

部分関数従属　　部分関数従属

受注NO	日時	顧客コード	顧客名
20220001	2022/7/1	C001	AA出版

商品コード	商品名	単価
B107	ファイル	50

第1正規形の表から、**部分関数従属**している列を切り出したもの
主キーを構成している項目によって、決定される項目を別の表に分割する。

第3正規形

非正規形
正規化
第1正規形
正規化
第2正規形
正規化
第3正規形

主キー

受注NO	日時	顧客コード	顧客名
20220001	2022/7/1	C001	AA出版

関数従属

受注NO	日時	顧客コード
20220001	2022/7/1	C001

顧客コード	顧客名
C001	AA出版

第2正規形の表から、**主キー以外の列に関数従属**している列を切り出したもの。
計算で求まるような項目は削除する。

正規化の手順まとめ

非正規形

正規化

第1正規形

正規化

第2正規形

正規化

第3正規形

第１正規化

- 表から**繰り返し部分を取り除く**。
- 素直な二次元の表の形にする。

第２正規化①

- **主キー(複合キー)**および**関数従属している列**を探す。
- 一意になる要素を探す。

第２正規化②

- **主キーを構成**している項目と**部分関数従属**している項目を**別の表に分割**する。

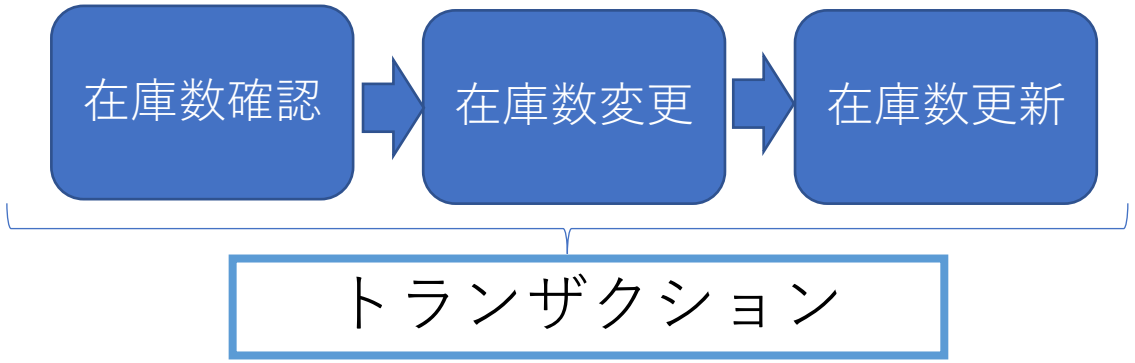
第３正規化

- **主キー以外の列に関数従属**している項目を別の表に分割。
- 計算で求まる列は削除する。

トランザクション管理と排他制御

- **トランザクション**

  一連の処理を**ひとまとめ**にしたもの



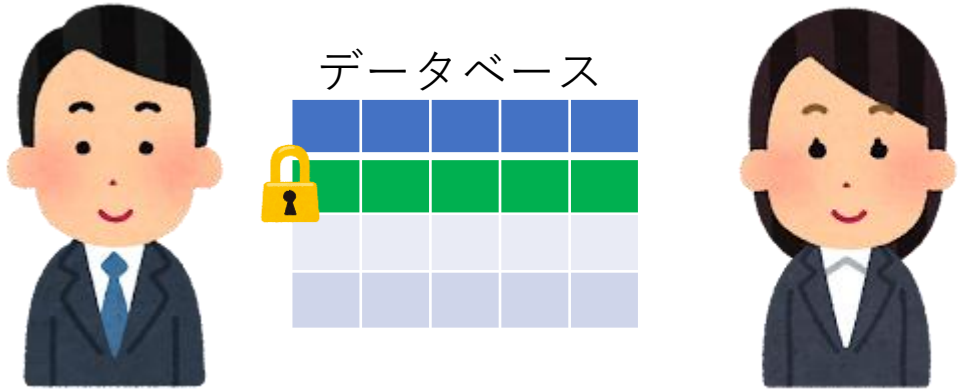
- **排他制御**

  データベース更新時に**データの不整合が発生しないように**、データの更新中は**アクセスを**ロックして別のトランザクションから更新できないように制御する。

  ロックする方法には**共有ロック**と**専有ロック**がある。

共有ロック

各ユーザは読むことはできるが書くことはできない

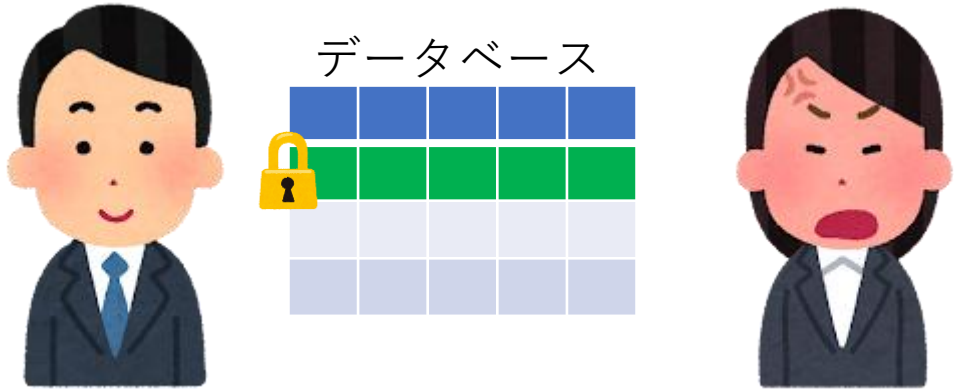


ロックして参照中
自分も読むだけ

読むことはできる

専有ロック

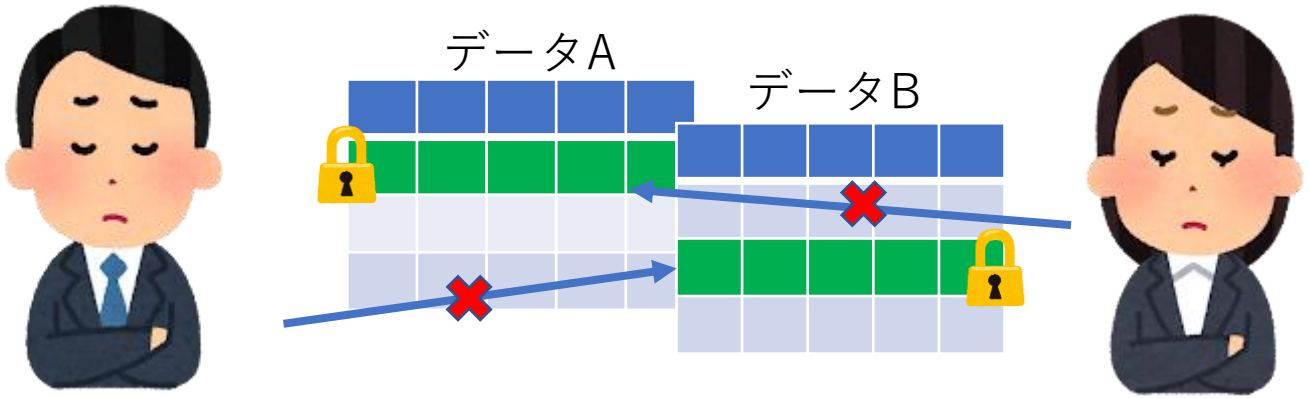
他のユーザは読むことも書くこともできない



ロックして編集中
自分は読み書き可能

読むこともできない

デッドロック



ロック解除待ち

ロック解除待ち

複数のトランザクションがお互いに
使いたい**データをロック**

お互いが**ロック解除待ち**となり
処理が進行できない。

ACID特性

Atomicity(原子性)

- トランザクションの処理結果は「すべて実行される」か「まったく実行されない」のいずれかで終了すること。

Consistency(一貫性)

- データベースの内容が矛盾のない状態であること。

Isolation(隔離性)

- トランザクション実行中の処理過程がほかの処理などに影響を与えないこと。

Durability(耐久性)

- トランザクションの更新結果は障害が生じても失われないこと。復旧手段が保証されていること。

ストアドプロシージャ(stored procedure)

DBMSの機能の一つで、データベースを操作する一連の**処理手順(SQL文)を一つのプログラムにまとめ、データベース管理システム側にあらかじめ保存**できるようにしたもの。

- SQL文でデータのやり取りをする必要がないので、ネットワークの負荷が軽減できる。
- 構造解析などが済んだ状態で保存されているため、高速に実行できる。

ストアドプログラム
・ストアドプロシージャ（戻り値なし）
・ストアドファンクション（戻り値あり）

データベースの障害管理

- データベースは障害の発生に備えて定期的に**バックアップ**を取る必要がある。
- バックアップ後の更新は**ジャーナル**と呼ばれるログファイルに、更新前の状態（**更新前ジャーナル**）と更新後の状態（**更新後ジャーナル**）を逐一記録して、データベースの更新履歴を管理している。
- これらのファイルを使って**ロールバック**や**ロールフォワード**などの障害回復処理を行い、復旧する

コミットとロールバック

コミット

トランザクション処理が問題なく完了できたとき、最後にその**更新を確定**し、データベースへと更新内容を反映させること。

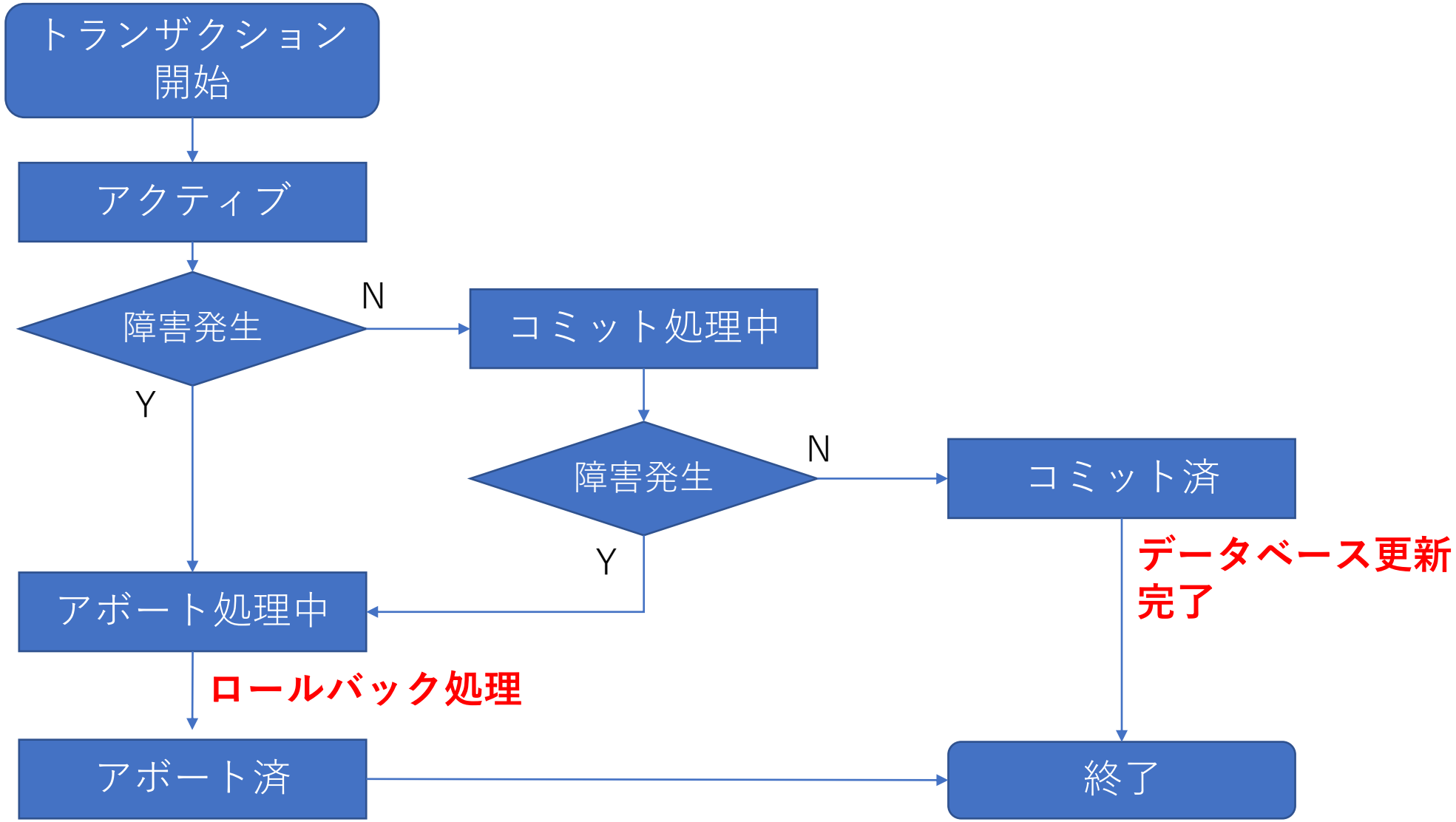
ロールバック

トランザクション処理中に障害が発生した場合、更新や処理を中断し(**アボート**)、データベースを**トランザクション開始直前の状態まで戻す**こと。
データベース更新前の状態は更新前ジャーナルから取得する。

Atomicity(原子性)

- トランザクションの処理結果は「すべて実行される」(コミット)か「まったく実行されない」(ロールバック)のいずれかで終了すること。

トランザクションの状態遷移



データベースを復旧させるロールフォワード

ロールフォワード

ディスク障害などで突然データベースが故障した場合、定期的に保存してある**バックアップファイル**からデータを復元する必要があるが、バックアップ後に加えられた変更は反映されていない。
データベースに行った更新情報を**更新後ジャーナル**から取得し、データベースを**障害発生直前状態にまで復旧**させる。

Durability(**耐久性**)

- トランザクションの更新結果は障害が生じても失われないこと。復旧手段が保証されていること。

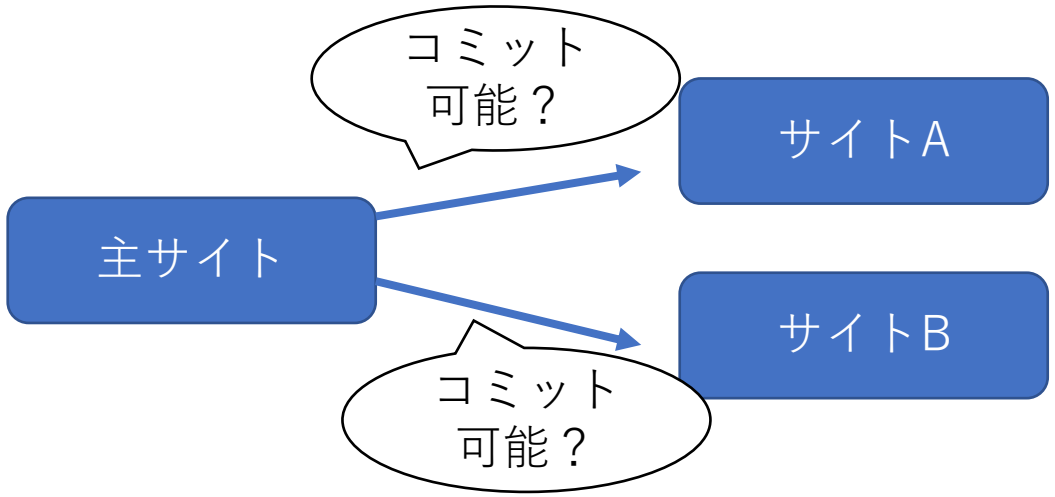
ロールフォワード

分散データベースと2相コミット

- 物理的に分かれている複数のデータベースを見かけ上一つのデータベースとして扱えるようにしたシステムを**分散データベースシステム**と呼ぶ。
- トランザクション処理は全体の同期をとってコミットやロールバックを行うようにして、**データの整合性**を取る。

2相コミット

① すべてのサイトにコミットの可否を問い合わせる。
② 全サイトで可能という返答があったときに全サイトに**コミット**を指示。
③ 一部でも不可なら全サイトに**ロールバック**を指示。



SQL(Structured Query Language)

DBMSへ指示を伝えるために用いる言語

DDL (Data Definition Language)

- テーブルの作成やスキーマの定義

DML (Data Manipulation Language)

試験の中心

- テーブルの中身のデータに対して操作（抽出、挿入、更新、削除）

DCL (Data Control Language)

- ユーザー権限や処理の操作

DDL

- 表の作成 (CREATE TABLE)
- ビュー表の作成(CREATE VIEW)
- **制約(NOT NULL, PRIMARY KEYなど)**

DML

- **SELECT句**
- **FROM句**
- **WHERE句**
- **表の結合**
- 別名(ASキーワード)
- **DISTINCT**キーワード
- 比較演算子 (=, <, >)
- 論理演算子(AND, OR, NOT)
- **BETWEEN演算子**
- **IN演算子**
- IS NULL演算子
- LIKE演算子
- **ORDER BY句**
- **集合関数（SUM,MAX,AVG,COUNT**など)
- **GROUP BY句**
- **HAVING句**
- **副問い合わせ**
- **EXISTS演算子**
- **NOT EXISTS演算子**
- 集合演算子 (UNION, INTERSECTなど)
- 内部結合 (INNER JOIN)
- 外部結合 (OUTER JOIN)
- 条件式（CASE式)
- レコードの追加（INSERT)
- レコードの更新（UPDATE)
- レコードの削除（DELETE)

表の定義(CREATE TABLE文)

概念スキーマに相当するもの。
データ構造を定義したもので磁気ディスクに存在する実表。

CREATE TABLE 表名 (
列名　データ型　オプション,
・・・・・・・・・
)

テーブルを作成する。
データ型
・NUMERIC/INTEGER　整数型
・CHAR　　固定長文字列型
・VARCHAR　可変長文字列型
・DATE　　日付型
オプションでは**制約***を設定可能

制約

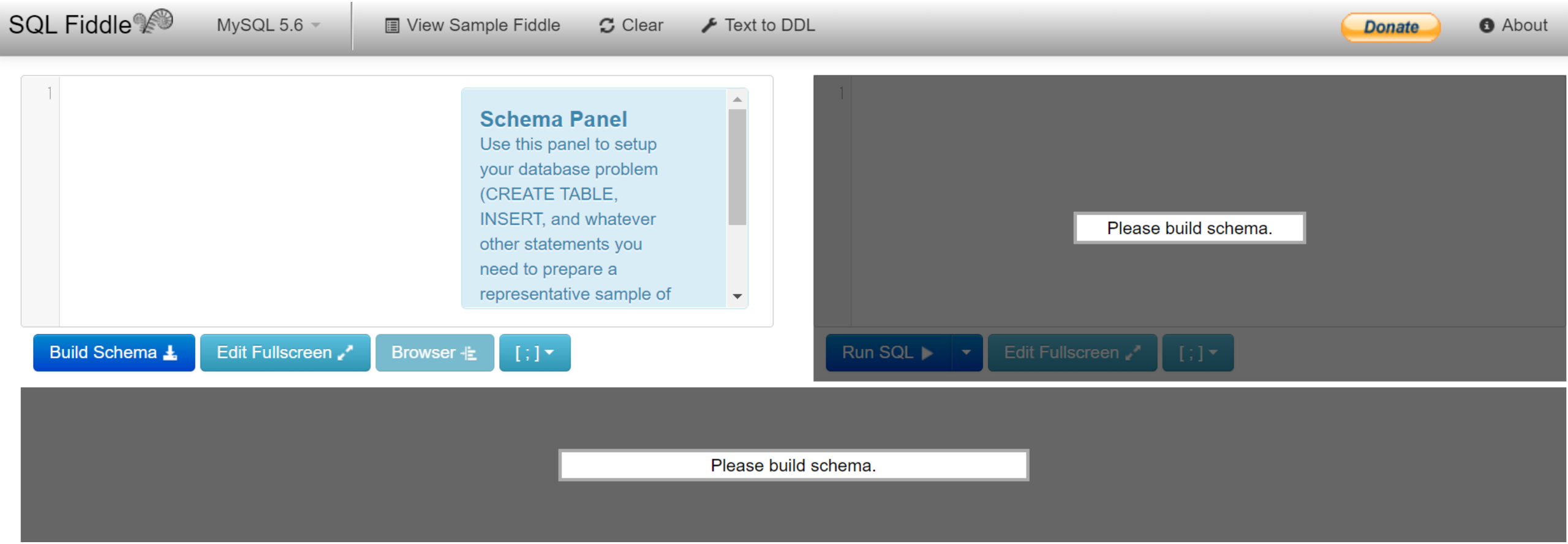
テーブルの列に入れられる値の**ルール**を決め、ルールに違反したデータは格納できないようにする。

制約名	説明
NOT NULL制約	NULL（値なし）は格納できない。
UNIQUE制約(一意制約)	重複した値は格納できない。
CHECK制約	指定した条件に合致しない値は格納できない。
DEFAULT制約	値を入力しなかった場合、初期値として指定した値が格納される。
PRIMARY KEY制約 (主キー制約)	主キー制約が設定された列はテーブルの主キーとなり、NULLおよび重複した値は格納できない。
FOREIGN KEY制約 (外部キー制約)	テーブルの親子関係を指定し、子テーブルにデータを追加するとき、外部キー制約が設定された列は親テーブルに格納されている値しか格納できない。（参照整合性を保証する。）

SQL文を書いてみよう

SQL Fiddle　　http://sqlfiddle.com/

ちょくちょく落ちているらしいので、エラー表示が出ていたら時間をおいてアクセス
ダメそうなら　DB Fiddle　https://www.db-fiddle.com　MySQL v5.6を選択してください



画面の見方


SQL Fiddle
MySQL 5.6
View Sample Fiddle
Clear
Text to DDL
Donate
About
CREATE TABLE test(id INTEGER, name VARCHAR(10), age INTEGER);
INSERT INTO test VALUES(1, "AAA", 20);
INSERT INTO test VALUES(2, "BBB", 24);
INSERT INTO test VALUES(3, "CCC", 31);
INSERT INTO test VALUES(4, "DDD", 28);
INSERT INTO test VALUES(5, "EEE", 35);
テーブルのSQL文
このボタンでテーブル作成
Build Schema
Edit Fullscreen
Browser
[;]
SELECT *
FROM test
WHERE age<30;
問い合わせ
このボタンでSQLを実行
Run SQL
Edit Fullscreen
[;]
id
name
age
1
AAA
20
2
BBB
24
4
DDD
28
結果
Record Count: 3; Execution Time: 4ms
View Execution Plan
link

```
/*SQLの練習用にテーブルを用意したので、このページをコピーして下さい*/
CREATE TABLE 名簿(
ID INTEGER(4) NOT NULL PRIMARY KEY,
名前 VARCHAR(20),
ローマ字表記 VARCHAR(20),
性別 CHAR(1),
年齢 INTEGER(2),
メールアドレス VARCHAR(40));
INSERT INTO 名簿 VALUES(1, '佐藤', 'SATO', '女', 26, 'sato@yahoo.co.jp');
INSERT INTO 名簿 VALUES(2, '鈴木', 'SUZUKI', '男', 24, 'suzuki@gmail.com');
INSERT INTO 名簿 VALUES(3, '高橋', 'TAKAHASHI', '女', 31, 'tkhs@gmail.com');
INSERT INTO 名簿 VALUES(4, '田中', 'TANAKA', '男', 39, 'tanaka@gmail.com');
INSERT INTO 名簿 VALUES(5, '渡辺', 'WATANABE', '男', 36, 'nabe@gmail.com');
INSERT INTO 名簿 VALUES(6, '小林', 'KOBAYASHI', '女', 27, 'koba@gmail.com');

CREATE TABLE 出席記録(
ID INTEGER(4),
体温 DECIMAL(3,1),
タイピングスコア INTEGER(4),
日付 DATE,
CONSTRAINT 出席記録_fk FOREIGN KEY(ID) REFERENCES 名簿(ID));
INSERT INTO 出席記録 VALUES(1, 36.2, NULL, '2022-07-04');
INSERT INTO 出席記録 VALUES(3, 35.7, 3900, '2022-07-04');
INSERT INTO 出席記録 VALUES(4, 36.6, 2800, '2022-07-04');
INSERT INTO 出席記録 VALUES(5, 35.5, 6900, '2022-07-04');
INSERT INTO 出席記録 VALUES(6, 35.8, 4900, '2022-07-04');
INSERT INTO 出席記録 VALUES(3, 35.5, 3700, '2022-07-05');
INSERT INTO 出席記録 VALUES(6, 36.1, 5300, '2022-07-05');
INSERT INTO 出席記録 VALUES(1, 36.3, NULL, '2022-07-06');
INSERT INTO 出席記録 VALUES(5, 35.2, 7300, '2022-07-06');
INSERT INTO 出席記録 VALUES(6, 36.1, 5100, '2022-07-06');

CREATE TABLE スコア評価表(
評価 CHAR(1) NOT NULL PRIMARY KEY ,
スコア下限値 INTEGER(4),
スコア上限値 INTEGER(4));
INSERT INTO スコア評価表 VALUES('S', 7000, 9999);
INSERT INTO スコア評価表 VALUES('A', 5000, 6999);
INSERT INTO スコア評価表 VALUES('B', 3000, 4999);
INSERT INTO スコア評価表 VALUES('C', 0, 2999);

CREATE TABLE 本リスト(
本ID INTEGER(4) NOT NULL PRIMARY KEY,
書名 VARCHAR(30));
INSERT INTO 本リスト VALUES(1, 'よくわかるMOS EXCEL');
INSERT INTO 本リスト VALUES(2, 'よくわかるMOS WORD');
INSERT INTO 本リスト VALUES(3, 'ITパスポート入門');
INSERT INTO 本リスト VALUES(4, '基本情報技術者');
INSERT INTO 本リスト VALUES(5, 'C言語');
INSERT INTO 本リスト VALUES(6, 'JAVA入門');

CREATE TABLE 本貸出記録(
貸出記録ID INTEGER(4) NOT NULL PRIMARY KEY,
名前ID INTEGER(4),
本ID INTEGER(4),
貸出日 DATE,
返却日 DATE,
CONSTRAINT 貸出人_fk FOREIGN KEY (名前ID) REFERENCES 名簿(ID),
CONSTRAINT 本_fk FOREIGN KEY (本ID) REFERENCES 本リスト(本ID));
INSERT INTO 本貸出記録 VALUES(22, 2, 6, '2022-07-01', NULL);
INSERT INTO 本貸出記録 VALUES(23, 3, 4, '2022-07-04', '2022-07-05');
INSERT INTO 本貸出記録 VALUES(24, 6, 5, '2022-07-05', '2022-07-06');
INSERT INTO 本貸出記録 VALUES(25, 1, 2, '2022-07-06', NULL);
/*ここまで表定義*/
```

例文で使用している表の定義　　実線の下線は主キー、点線は外部キー

名簿

ID	名前	ローマ字表記	性別	年齢	メールアドレス

出席記録

ID	体温	タイピングスコア	日付

スコア評価表

評価	スコア下限値	スコア上限値

本リスト

本ID	書名

本貸出記録

貸出記録ID	名前ID	本ID	貸出日	返却日

データ操作言語

関係データベースの表から必要なデータを抽出する**SELECT**文。
データを抽出することを**問い合わせ（クエリ）**と呼ぶ。

SELECT　列名　または計算式
FROM　表名
(WHERE　条件)
(GROUP BY　グループ化する列名)
(HAVIING　絞り込み条件)
(ORDER BY　整列する列名)

- 表をすべて抽出
  SELECT　＊
  FROM　表名

SQL文の最後に「;」（セミコロン）を挿入してください。

```
SELECT *
FROM 出席記録;
```

- 列を抽出**(射影)**
  SELECT　列名
  FROM　表名

列を複数挙げるときは「,」（カンマ）で区切る

```
SELECT ID, 名前, 性別
FROM 名簿;
```

- 行を抽出**(選択)**
  SELECT　＊
  FROM　表名
  WHERE　条件

```
SELECT *
FROM 名簿
WHERE ID = 1;
```

- 表と表の結合（**結合**）

```
SELECT  *
FROM    表1, 表2
WHERE  表1.結合させる列 = 表2.結合させる列
```

どちらの表の列を参照しているのか明示的にするため**表名.列名**と書く

重複していなければ表名は書かなくてもOK

```
SELECT 名前, 日付
FROM 名簿, 出席記録
WHERE 名簿.ID = 出席記録.ID;
```

- 別名の書き方

元の名 **AS** ○○

または

元の名_○○　半角スペースで空ける

```
SELECT ID AS 番号
FROM 名簿;
```

```
SELECT タイピングスコア SCORE
FROM 出席記録;
```

- 重複行の排除

DISTINCTキーワード

重複した値を省いてくれるので、種類を列挙するときなどに便利

```
SELECT DISTINCT 日付
FROM 出席記録;
```

- 比較演算子での条件指定

WHERE　〇〇 >= 〇〇など

比較演算子

=	等しい
>	左辺が大きい
>=	左辺が大きいor等しい
<	左辺が小さい
<=	左辺が小さいor等しい
<>	等しくない

```
SELECT *
FROM 出席記録
WHERE ID = 1;
```

```
SELECT *
FROM 名簿
WHERE 年齢 >= 30;
```

```
SELECT *
FROM 出席記録
WHERE 日付 <> 20220705;
```

- 論理演算子での条件の組合わせ

WHERE　条件1 **AND** 条件2

論理演算子

AND　論理積（～かつ～）
OR　論理和（～または～）
NOT　否定（～ではない）

```
SELECT *
FROM 出席記録
WHERE 日付 = 20220704 AND
体温 < 36.0;
```

```
SELECT *
FROM 名簿
WHERE 年齢 >=35 OR 年齢
<=25;
```

```
SELECT *
FROM 名簿
WHERE NOT(性別 = 女 OR 年齢
>= 30);
```

- BETWEEN演算子

WHERE 列名 **BETWEEN** 下限値 **AND** 上限値

下限値と上限値の間に該当するレコードを取得

列名 >=下限値 AND 列名 <=上限値と同じ

例）SELECT *

FROM 出席記録

WHERE 日付 BETWEEN 20220704 AND 20220705;

- IN演算子

WHERE 列名 **IN** (A, B, …)

()の中に列挙されたのものに該当するレコードを取得

列名 = A OR 列名 = B OR… と同じ

```
SELECT *
FROM 出席記録
WHERE ID IN (1, 3, 5);

SELECT *
FROM 本リスト
WHERE 本ID NOT IN (1, 2, 3);
```

NOTで否定

- IS NULL演算子

WHERE 列名 **IS NULL**

列の値がNULL（値がない状態）のレコードを取得

```
SELECT *
FROM 本貸出記録
WHERE 返却日 IS NULL;

SELECT *
FROM 出席記録
WHERE タイピングスコア IS NOT NULL ;
```

NOTで否定

- LIKE演算子

WHERE 列名 **LIKE** '文字列'
　　指定した文字パターンに一致する文字を検索

ワイルドカード文字

%　0桁以上の文字列

_　_1つが1桁の文字を表す

例）SELECT *
　　FROM 名簿
　　WHERE ローマ字表記 LIKE '_A%';

名前の2文字目がA

- 行の並び替え　ORDER BY句

ORDER BY 列名 [**ASC** / **DESC**]
(ASCは昇順 (省略可) / DESCは降順)

```
SELECT *
FROM 本貸出記録
ORDER BY 貸出日 DESC;
```

```
SELECT *
FROM 名簿
ORDER BY 性別, 年齢;
```

複数並び替え条件がある場合は[,](カンマ)で区切る

- 関数を使って集計する

集合関数

MAX(列名）列の最大値
MIN(列名）列の最小値
AVG(列名）列の平均値
SUM(列名）列の合計値
COUNT(*)　行の個数
COUNT(列名）行の個数
（値のあるもの）

```
SELECT MIN(年齢)
FROM 名簿;
```

```
SELECT AVG(体温) AS 平均体温
FROM 出席記録
WHERE ID = 3;
```

```
SELECT COUNT(*) AS 出席人数
FROM 出席記録
WHERE 日付 = 20220704;
```

- データをグループ化する　　GROUP BY句

  SELECT　列名,　**集合関数**(列名)
  FROM　表名
  GROUP BY　列名

SELECT ID, MAX(タイピング）AS 最高スコア
FROM 出席記録
GROUP BY ID;

SELECT ID, COUNT(*) AS 出席回数
FROM 出席記録
GROUP BY ID;

- グループに条件を付けて絞り込む　　HAVING句

```
SELECT　列名,　集合関数(列名)
FROM　表名
GROUP BY　列名
HAVING　絞り込み条件
```

例）SELECT ID, AVG(体温）AS 平均体温
FROM 出席記録
GROUP BY ID
HAVING AVG(体温) >= 36.0;

例）SELECT 日付, COUNT(*) AS 出席人数
FROM 出席記録
GROUP BY 日付
HAVING COUNT(*)<=4;

・**副問い合わせ**　　SELECT文の問い合わせ結果を検索条件にする

副問い合わせの結果が確実に1つのみの場合 (単一行問い合わせ)

```
SELECT　列名
FROM　表名
WHERE　列名 比較演算子 (SELECT 列名 FROM 表名 WHERE…)
```

本問い合わせ　　**副問い合わせ**

①先に副問い合わせを行う。
②副問い合わせで得られた条件で主問い合わせを行う。

```
SELECT ID, タイピングスコア
FROM 出席記録
WHERE タイピングスコア > (SELECT AVG(タイピングスコア) FROM 出席記録);
```

集合関数を使って平均値のみが戻ってくるので、
比較演算子を使って条件を指定できる。
主キーに対しての問い合わせも確実に戻る結果が
１つになるためこの方法でもOK。

- **副問い合わせ**　SELECT文の問い合わせ結果を検索条件にする

副問い合わせの結果が1つ以上の場合 (複数行問い合わせ)

```
SELECT　列名
FROM　表名
WHERE　列名 IN (SELECT 列名 FROM 表名 WHERE…)
```

主問い合わせ　　**副問い合わせ**

①先に副問い合わせを行う。
②副問い合わせで得られた条件で主問い合わせを行う。

```
SELECT *
FROM 本リスト
WHERE 本ID IN (SELECT 本ID FROM 本貸出記録 WHERE 返却日 IS NULL);
```

IN演算子を使うことで、副問い合わせの結果と等しいものを検索条件にできる。
ANY演算子やALL演算子を使うと等しい以外の比較もできる。

EXISTS演算子を使った**相関副問い合わせ**

```
SELECT　列名
FROM　表名
WHERE　EXISTS (SELECT 列名 FROM 表名 WHERE…)
```

主問い合わせ　　**副問い合わせ**

主問い合わせから副問い合わせに1行ずつ渡し、存在の有無を判断して
副問い合わせの**結果が存在する（TRUE）**ときのみ返す。
2つのテーブルに存在するレコードだけを取得するときによく使われる。

①1行ずつ主問い合わせを行う。
②主問い合わせで得られた結果が副問い合わせの条件で存在するか調べる。
③副問い合わせから主問い合わせに返す。　→①に戻る

```
SELECT *
FROM 名簿
WHERE EXISTS(SELECT * FROM 本貸出記録 WHERE 名簿.ID = 本貸出記録.名前ID);
```

NOT EXISTS演算子を使った**相関副問い合わせ**

```
SELECT　列名
FROM　表名
WHERE　NOT EXISTS (SELECT 列名 FROM 表名 WHERE…)
```

主問い合わせ　　**副問い合わせ**

主問い合わせから副問い合わせに1行ずつ渡し、存在の有無を判断して
副問い合わせの**結果が存在しない（FALSE）**ときのみ返す。

①1行ずつ主問い合わせを行う。
②主問い合わせで得られた結果が副問い合わせの条件で存在するか調べる。
③副問い合わせから主問い合わせに返す。　→①に戻る

```
SELECT *
FROM 名簿
WHERE NOT EXISTS(SELECT * FROM 出席記録 WHERE 名簿.ID = 出席記録.ID);
```

- 集合演算子

１つの問い合わせによって戻される行を1つの集合として扱い、
複数の集合(表)間で**論理和**や**論理積**を求めることができる。

UNION ALL(論理和 重複あり ソートなし)
UNION(論理和 重複なし ソートあり)
INTERSECT(論理積 ソートあり)
EXCEPT(否定(差分))

SELECT 列名	SELECT 列名	SELECT 列名	SELECT 列名
FROM 表名1	FROM 表名1	FROM 表名1	FROM 表名1
UNION ALL	**UNION**	**INTERSECT**	**EXCEPT**
SELECT 列名	SELECT 列名	SELECT 列名	SELECT 列名
FROM 表名2	FROM 表名2	FROM 表名2	FROM 表名2

SELECT 列名
FROM 表名1
集合演算子
SELECT 列名
FROM 表名2

問合わせによって
戻ってきた行の集合

表1

ID		
1		
2		
3		

問合わせによって
戻ってきた行の集合

表2

ID		
1		
3		
5		
7		

UNION ALL
論理和(重複)

ID		
1		
2		
3		
1		
3		
5		
7		

UNION
論理和

ID		
1		
2		
3		
5		
7		

INTERSECT
論理積

ID		
1		
3		

EXCEPT
否定(NOT 表2)

ID		
2		

```
SELECT *
FROM  名簿
WHERE EXISTS(SELECT * FROM 出席記録 WHERE
出席記録.ID =名簿.ID AND 日付 = 20220706)
UNION
SELECT *
FROM 名簿
WHERE EXISTS(SELECT * FROM 本貸出記録
WHERE 本貸出記録.名前ID = 名簿.ID);


SELECT *
FROM  名簿
WHERE EXISTS(SELECT * FROM 出席記録 WHERE
出席記録.ID =名簿.ID AND 日付 = 20220706)
INTERSECT
SELECT *
FROM 名簿
WHERE EXISTS(SELECT * FROM 本貸出記録
WHERE 本貸出記録.名前ID = 名簿.ID);
```

・結合(JOIN)

複数の表の異なる行を1行にして表示する。
外部キーの値と主キーの値が一致している状態にするための結合条件を記述する。

外部キー

社員番号	氏名	部署ID
2009001	田中一郎	1
2009002	高橋二郎	2
2009003	佐藤三郎	2
2009004	鈴木四郎	3

主キー

部署ID	部署名
1	営業部
2	開発部
3	経理部

社員番号	氏名	部署ID	部署名
2009001	田中一郎	1	営業部
2009002	高橋二郎	2	開発部
2009003	佐藤三郎	2	開発部
2009004	鈴木四郎	3	経理部

結合の種類

- **内部結合**　　結合条件に一致する行のみを表示

- **外部結合**　　結合条件に一致しない行も含めて表示
  - 左側外部結合　　左側の表はすべての行を表示
  - 右側外部結合　　右側の表はすべての行を表示
  - 完全外部結合　　両方の表はすべての行を表示

※外部結合の際、結合条件を満たさない行の列の値はすべて**NULL**になる。

- クロス結合　　2つの表のクロス積（デカルト積）を作成
  行の総組み合わせを表示（結合条件なし）

内部結合

SELECT 列名
FROM 表1名 **INNER JOIN** 表2名
ON 表1.外部キー ＝ 表2.主キー
結合条件

クロス結合

SELECT 列名
FROM 表1名 CROSS OUTER JOIN 表2名

外部結合

SELECT 列名
FROM 表1名 { LEFT / RIGHT / FULL } **OUTER JOIN** 表2名
ON 表1.外部キー ＝ 表2.主キー
結合条件

左側外部結合の場合は	LEFT
右側外部結合の場合は	RIGHT
完全外部結合の場合	FULL

```
SELECT *
FROM 出席記録 INNER JOIN 名簿
ON 出席記録.ID = 名簿.ID
ORDER BY 日付;
```

結合する条件は**非等価**も可能。
結合条件は**ON**句のあとに記載する。

SELECT ID, 日付, タイピングスコア, 評価
FROM 出席記録 **INNER JOIN** スコア評価表
ON タイピングスコア BETWEEN スコア下限値 AND スコア上限値;

結合条件にBETWEEN演算子を使っている

条件式　CASE式

```
CASE 列名  WHEN 条件値1 THEN 値1
           WHEN 条件値2 THEN 値2
           WHEN 条件値n THEN 値n
           ELSE デフォルト値
END
```

```
CASE WHEN 列名 演算子 条件値1 THEN 値1
     WHEN 列名 演算子 条件値2 THEN 値2
     WHEN 列名 演算子 条件値n THEN 値n
     ELSE デフォルト値
END
```

- IF-THEN-ELSEの機能で条件付き問い合わせをすることができる。
- 1つの式全体で1つの値を戻す。
- 条件に一致するものが見つかると、後続の条件は評価しない。
- デフォルト値(ELSE句）を省略した場合、NULLを戻す。

```
SELECT ID,
CASE 性別 WHEN '男' THEN CONCAT(名前, '君')
          WHEN '女' THEN CONCAT(名前, 'さん') END AS 敬称付き名前
FROM 名簿;
```

CONCATは文字列を連結させる関数

- レコードの追加　INSERT文

INSERT INTO 表名 (列1名, 列2名, ...)
VALUES（値1, 値2, ...)

表の全項目に値を入れる場合は、列名は省略可能
INSERT INTO 表名 **VALUES** (値1, 値2, ...)

INSERT INTO 出席記録(ID, 体温,日付)
VALUES(1, 36.5, 20220707);

- レコードの変更　UPDATE文

  UPDATE 表名
  SET　列名 = 更新後値, 列名 = 更新後値, ...
  WHERE 条件

  条件に合致するレコードすべてが更新される。
  WHERE句がない場合、すべてのレコードが更新される。

  UPDATE 名簿
  SET メールアドレス= 'abcd@gmail.com'
  WHERE ID = 1;

- レコードの削除　DELETE文

DELETE FROM 表名
WHERE 条件

WHERE句の条件に合致するレコードすべてが削除される
WHERE句を記載しない場合、すべてのレコードが削除される

DELETE FROM 本リスト
WHERE 本ID = 3;

ビュー表の作成

CREATE VIEW ビュー名（列名,・・）AS SELECT文

ビューとして表示する内容を指定するSQL文を記述する

CREATE VIEW 連絡表(ID, 名前, メールアドレス)
AS SELECT ID, 名前, メールアドレス FROM 名簿;

NoSQL(Not Only SQL)

- **SQLを使わないで**操作するデータベース。
- データベースの構造を柔軟に変更でき、データの増加に対応しやすい。
- データの正規化や表の結合はできない。
- 集計や検索は不得意。

分類	特徴
キーバリュー型	保存したいデータとそのデータを一意に識別できるキーを組として管理
カラム志向型	キーに対するカラム（項目）を自由に追加できる
ドキュメント志向型	ドキュメント1件が1つのデータとなる。データ構造は自由。

データベースの応用

企業ではデータベースに大量のデータを蓄積し、様々な形で有効活用している。

データウェアハウス	企業の様々な活動を通して得られた大量のデータを整理・統合して蓄積したデータベース。意思決定支援などに葛生する。Wearhouse：倉庫
データマート	利用者が情報を利用するために、データウェアハウスから抽出した目的別のデータベース。あらかじめ承継処理をしておくと、探索時間を短縮できる Mart：小売店
データマイニング	大量のデータを統計的・数学的手法で分析し、新たな法則や因果関係を見つけ出すこと。 Mining：発掘
BI(Business Intelligence)ツール	データウェアハウスやデータマートに蓄積された情報をデータマイニングなどで分析し、経営判断上の有用な情報を取り出すツール。専門知識がない利用者も操作可能。

ビッグデータ

- サーバのログや売上データ、購入履歴、GPSやRFIDのデータ、SNSのコメント、電子メール、動画など、**多種多様で高頻度に更新される大量のデータ。**
- これらを組み合わせて分析することで、新たな価値を生み出すことを期待する。

オープンデータ

- 機械判読に適した形式で自由に二次利用できるというルールの下で、国・自治体・企業などが公開するデータ。

SQLインジェクション

ユーザの入力値をデータベースに問い合わせて処理を行うWebサイトに対し、その入力内容に悪意のある問い合わせや操作を行う**SQL文を埋め込み**、データベースのデータを不正に取得したり、改ざんしたりする手法

例えば、ログイン認証で↓ようなSQL文が利用されていた場合

SELECT id,pass FROM login_user WHERE id='入力されたID'
AND pass='入力されたパスワード';

IDを打ち込む欄に **'1' OR '1'='1';--**といれると…

読み取られない

SELECT id,pass FROM login_user WHERE id='1' OR '1'='1';--AND pass='';

常に条件式が成立するため、パスワードがなくても不正にログイン可能

過去問

問　関係XとYを自然結合した後，関係Zを得る関係代数演算はどれか

X

学生番号	氏名	学部コード
1	山田太郎	A
2	情報一郎	B
3	鈴木花子	A
4	技術五郎	B
5	小林次郎	A
6	試験桃子	A

Y

学部コード	学部名
A	工学部
B	情報学部
C	文学部

Z

学部名	学生番号	氏名
情報学部	2	情報一郎
情報学部	4	技術五郎

ア 射影と和　　イ 選択　　ウ 選択と射影　　エ 選択と和

問　ある企業では，顧客マスタファイル，商品マスタファイル，担当者マスタファイル及び当月受注ファイルを基にして，月次で受注実績を把握している。各ファイルの項目が表のとおりであるとき，これら四つのファイルを使用して当月分と直前の3か月分の出力が可能な受注実績はどれか。

ファイル	項　　目	備　考
顧客マスタ	顧客コード，名称，担当者コード，前月受注額，2か月前受注額，3か月前受注額	各顧客の担当者は1人
商品マスタ	商品コード，名称，前月受注額，2か月前受注額，3か月前受注額	―――
担当者マスタ	担当者コード，氏名	―――
当月受注	顧客コード，商品コード，受注額	当月の合計受注額

ア 顧客別の商品別受注実績

イ 商品別の顧客別受注実績

ウ 商品別の担当者別受注実績

エ 担当者別の顧客別受注実績

問　中間テスト表からクラスごと，教科ごとの平均点を求め，クラス名，教科名の昇順に表示するSQL文中のaに入れるべき字句はどれか。

中間テスト(クラス名,教科名,学生番号,名前,点数)

〔SQL文〕SELECT クラス名，教科名，AVG(点数) AS 平均点
FROM 中間テスト
a

ア GROUP BY クラス名, 教科名 ORDER BY クラス名, AVG(点数)
イ GROUP BY クラス名, 教科名 ORDER BY クラス名, 教科名
ウ GROUP BY クラス名, 教科名, 学生番号 ORDER BY クラス名, 教科名, 平均
エ GROUP BY クラス名, 平均点 ORDR BY クラス名, 教科名

問　注文表と製品表に対して，次のSQL文を実行したときに得られる結果はどれか。

```
SELECT 製品名,数量 FROM 注文,製品
        WHERE 注文.製品コード = 製品.製品コード
```

注文

日付	製品コード	数量
4月10日	P2	120
4月15日	P1	100
4月22日	P4	50
4月30日	P8	80
5月 6日	P1	100
5月 8日	P3	70

製品

製品コード	製品名
P1	PC
P2	テレビ
P3	掃除機
P4	冷蔵庫
P5	エアコン
P6	電話機
P7	時計

ア

製品名	数量
テレビ	120
PC	100
冷蔵庫	50
掃除機	70

イ

製品名	数量
テレビ	120
PC	200
冷蔵庫	50
掃除機	70

ウ

製品名	数量
テレビ	120
PC	100
冷蔵庫	50
PC	100
掃除機	70

エ

製品名	数量
テレビ	120
PC	100
冷蔵庫	50
Null	80
PC	100
掃除機	70

問　商品表，在庫表に対する次のSQL文と，同じ結果が得られるSQL文はどれか。ここで，下線部は主キーを表す。

```
SELECT 商品番号 FROM 商品
    WHERE 商品番号 NOT IN (SELECT 商品番号 FROM 在庫)
```

商品

商品番号	商品名	単価

在庫

在庫番号	商品番号	在庫数

ア SELECT 商品番号 FROM 在庫 WHERE EXISTS (SELECT 商品番号 FROM 商品

イ SELECT 商品番号 FROM 在庫 WHERE NOT EXISTS (SELECT 商品番号 FROM 商品)

ウ SELECT 商品番号 FROM 商品 WHERE EXISTS (SELECT 商品番号 FROM 在庫 WHERE 商品.商品番号 = 在庫.商品番号)

エ SELECT 商品番号 FROM 商品 WHERE NOT EXISTS (SELECT 商品番号 FROM 在庫 WHERE 商品.商品番号 = 在庫.商品番号)

問　関係データベース"注文"表の"顧客番号"は"顧客"表の主キー"顧客番号"に対応する外部キーである。このとき、参照の整合性を損なうデータ操作はどれか。ここで、ア～エの記述におけるデータの並びは、それぞれの表の列の並びと同順とする。

注文

伝票番号	顧客番号
0001	C005
0002	K001
0003	C005
0004	D010

顧客

顧客番号	顧客名
C005	福島
D010	千葉
K001	長野
L035	宮崎

ア “顧客”表の行 L035 宮崎 を削除する。
イ “注文”表の行 0005 D010 を追加する。
ウ “注文”表の行 0006 F020 を追加する。
エ “注文”表の行 0002 K001 を削除する。