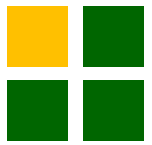


データベース入門

MySQL編

Ver. 8.0



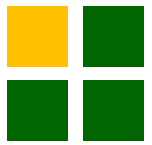
アジェンダ

- データベース
- DBMSとは
- データベースの構成
- データベースの種類
- NoSQLの種類
- NoSQLとRDBMS
- KVSとRDBの違い
- テーブル(表)とは
- データ型
- RDBのイメージ
- DBのインストール手順
- MySQLの操作
- SQL
- SQLの分類
- DML
- DDL
- データベースの作成
- ユーザの作成
- 作成ユーザでのログイン
- ログインのショートカット作成
- 作成データベースへのログイン
- テーブルの作成
- ユーザ情報の登録
- データ操作言語
- 比較演算子
- 論理演算子



アジェンダ

- 制約とは
- 各種制約一覧
- 制約の設定
- 正規化
- 非正規形
- 第一正規形
- 関数従属
- 第二正規形
- 第三正規形
- サロゲートキー
- 1対1／1対多／多対多の
関係
- ER図
- マスタと
トランザクション
- テーブル結合
- サブクエリ(副問合せ)
- エイリアス(別名)
- ソート
- グループینگ
- 集約関数
- 集約関数とグループینگ
- あいまい検索
- ワイルドカード
- スキーマ
- 3層スキーマ
- インデックス管理
- ストアドプロシージャ
- 分散データベース



アジェンダ

- 2相コミット
- レプリケーション
- トランザクション制御
- デッドロック
- 障害管理とバックアップ
- ジャーナルファイル
- 参考資料・DDL文
- 参考資料・DML応用



データベース

データベースとは、複数のユーザが情報を共有して使用するための場所の事です。

1950年頃、米軍が各地に点在していた膨大な量の情報を一ヶ所の基地に集め、そこにアクセスすれば全ての情報が得ることが出来るように効率化を図る目的でこのシステムを作りました。

「情報(データ)の基地(ベース)」ということで、このシステムの事を「情報基地(データベース)」と呼ぶようになったと言われています。



DBMSとは

データベースは、よく図書館に例えられます。

読みたい・借りたい本があった場合、どのようにその本を探すのでしょうか。きっと、図書館の司書に聞くでしょう。

何という本で、著者が誰と伝えれば、どこの本棚にあって、ジャンルはこれで、貸出可能になっているかも教えてくれるでしょう。

その様に、司書の仕事は本の情報や位置を把握し、図書館を快適に活用できるようにすることです。

本棚が漠然と置いてあり、膨大な本が並んでいるだけでは、全く活用されません。常に最新で正確な情報を整理しておく必要があります。

データベースも同じく、情報のメンテナンスを行いデータの管理が必要になります。

その情報(本棚)を管理する人(司書)が

DBMS(DatabaseManagementSystem)

というわけです。



データベースの構成

- **スタンドアロン構成**

データベースとアプリケーションを同一マシン内で動作させる構成です。ネットワークを経由することなく、直接データベースを操作することができます。個人または少数のユーザで使⽤します。

- **クライアント/サーバ構成**

データベースをサーバマシン上で動作させ、クライアントアプリケーションからDBサーバ上のデータベースへアクセスをする構成です。ネットワークはLANやWANで構成されるのが一般的です。

- **ウェブ構成**

現在主流の構成で、クライアントはブラウザを利用してインターネットにアクセスし、Webサーバに接続します。

WebサーバはAPサーバ上で動作しているアプリケーションを使用して、DBサーバ上のデータベースに接続します。インターネットを利用することで、世界中どこからでも接続可能です。



データベースの種類

- 各種RDBMS

データベースの中でも、テーブル(表)によってデータを関連付けして管理するものをリレーショナルデータベースと呼びます。
また、管理するソフトウェアをRDBMSと呼びます。

RDBMS名称	有償/無償	ベンダ
Oracle Database	有償	Oracle
SQL Server	有償	Microsoft
DB2	有償	IBM
MySQL	無償	Oracle
PostgreSQL	無償	PostgreSQL Global Development Group



NoSQLの種類

- NoSQLとは

RDBMS以外のデータベースを指すおおまかな分類語で、大量データの扱いや大量アクセスの処理が得意で、Key-Value型のDBを使用することが多いです。

近年では取り扱うデータが肥大化(ビッグデータなど)しており、データの関係性が多様化しています。

そのため、RDBでは困難(または不可能)なデータ格納となることも増えてきています。

またスピーディなデータアクセスという視点でもNoSQLは非常に有用です。

NoSQL名称	有償/無償	ベンダ
BigTable	有償	Google
MongoDB	無償	MongoDB Inc.
Cassandra	無償	Apache Software Foundation
HBase	無償	Apache Software Foundation



NoSQLとRDBMS

- NoSQLとRDBを選択する

ビッグデータに代表される、膨大なデータ群からポイントを絞ったデータを高速に取得したいときなどに使われることが多いです。

他にも、アクセスから描画までの時間を極力押さえたい、SNSタイムラインやIoT、アクセス解析など、【システム-NoSQL-RDB】というような構成にすることが増えてきています。

このように、システムに求められる要件によって、より柔軟な選択をすることがより大切になってきています。



KVSとRDBの違い

- 分散化のコストDown

これが一番の特徴だと思います。KVS(Key-Value-Store)はデータ(キー)同士に関連性が無いため、分散化(複数台のサーバを使用 などのコストが低くなっています。

これにより負荷分散や可用性の向上などが制限なく可能になります。

欠点としては、KVSに入れられるデータの型に決まりがなくバイナリデータだったら何でも入れられてしまいます。そのため統一的なクライアント側(SQLなど)が用意されていません。(とはいえDBMS毎にテーブル構造すら違うものを統一するのは困難なので、そこまでデメリットではないような気も・・・)

これらを踏まえ状況によって使いこなしましょう。

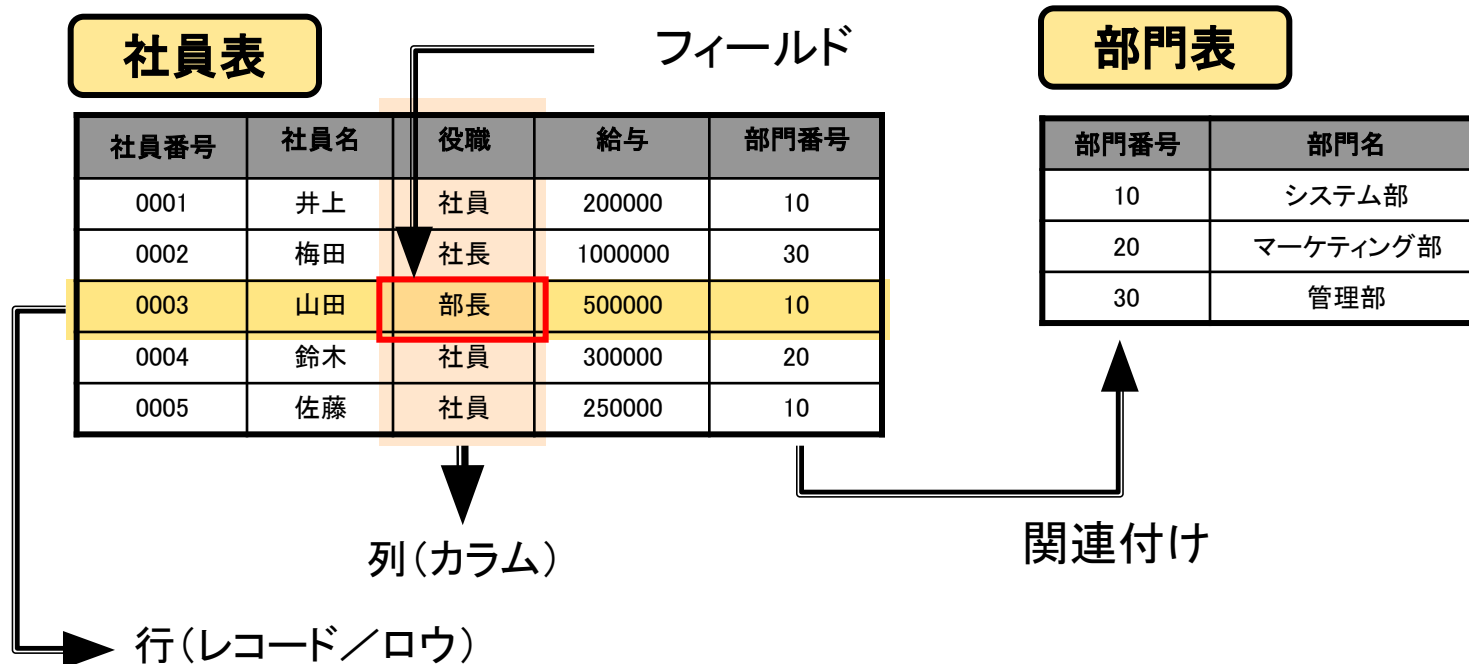


テーブル(表)とは

表(テーブル)とは、データを格納するオブジェクトで、リレーショナルデータベースの基本となるものです。

Excelのように縦軸と横軸の2次元形式で、縦軸を列(カラム)、横軸を行(ロウまたはレコード)といいます。

縦軸はデータの属性をあらわします。図の例では「社員番号」や「社員名」になります。横軸はデータです。





データ型

データ型とは、列に対して格納するデータを制限する方法です。

データ型によって整数であるか、文字列であるかといったデータの種類を限定することができます。

また、格納できる文字数や、小数点以下第何位まで格納するかといったサイズや精度指定を行うことも可能です。

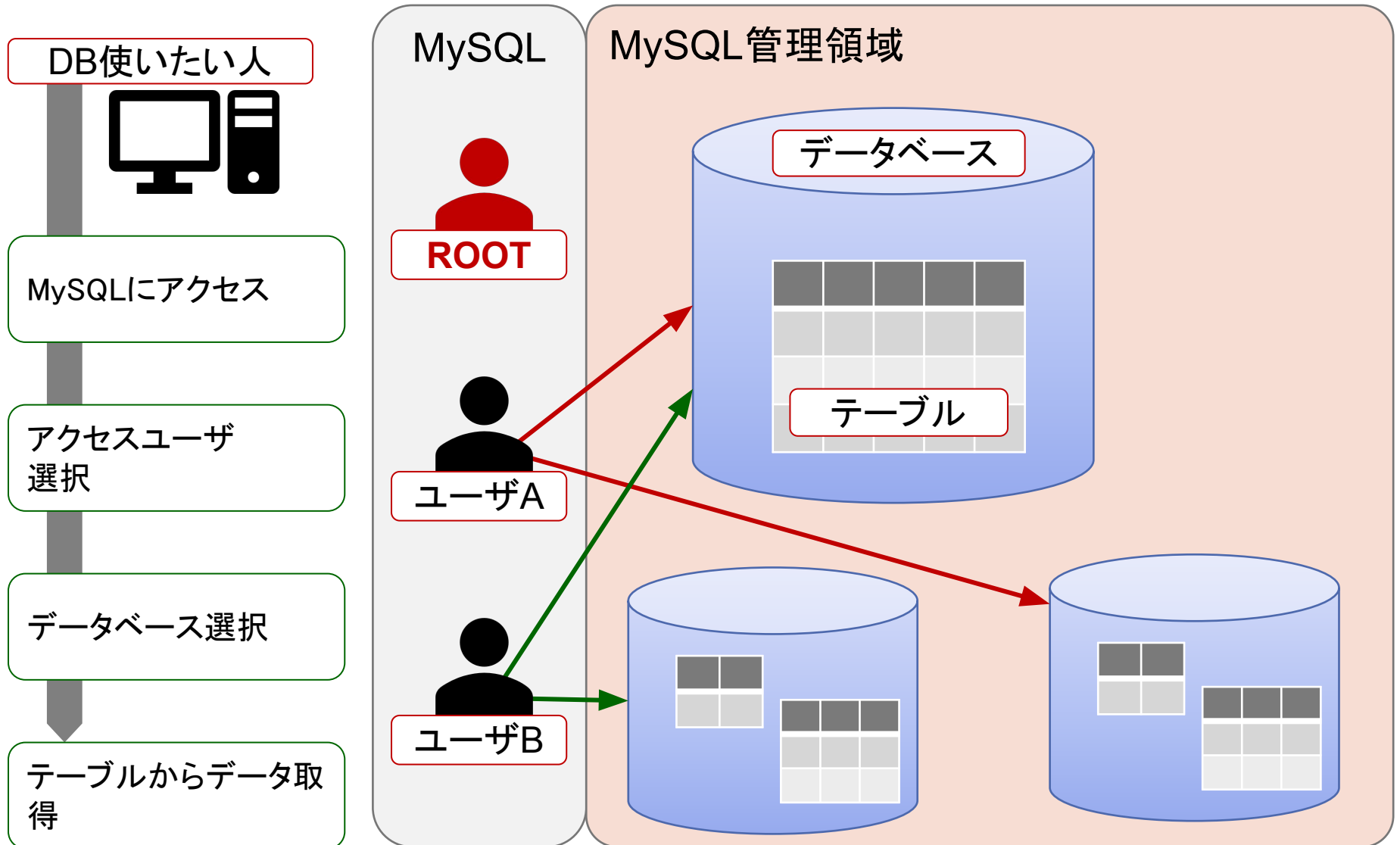
【主要なデータ型一覧】

属性	データ型	解説
文字	varchar	最大サイズ指定可変長文字列
	char	固定長文字列
	text	制限なし可変長文字列
数値	int	4バイト整数型
	numeric	可変精度数値型
日付	date	日付型

※文字型や数値型、その他バイナリ型など他にも多くのデータ型が存在しています。



RDBのイメージ





DBのインストール手順

今回の講座では MySQL (マイエスキューエル) という RDBMS を使用します。

Oracle (オラクル) の公式サイトからインストーラをダウンロードします。
手順は以下の通りになります。講師の指示を聞き、MySQLを導入してみましょう。

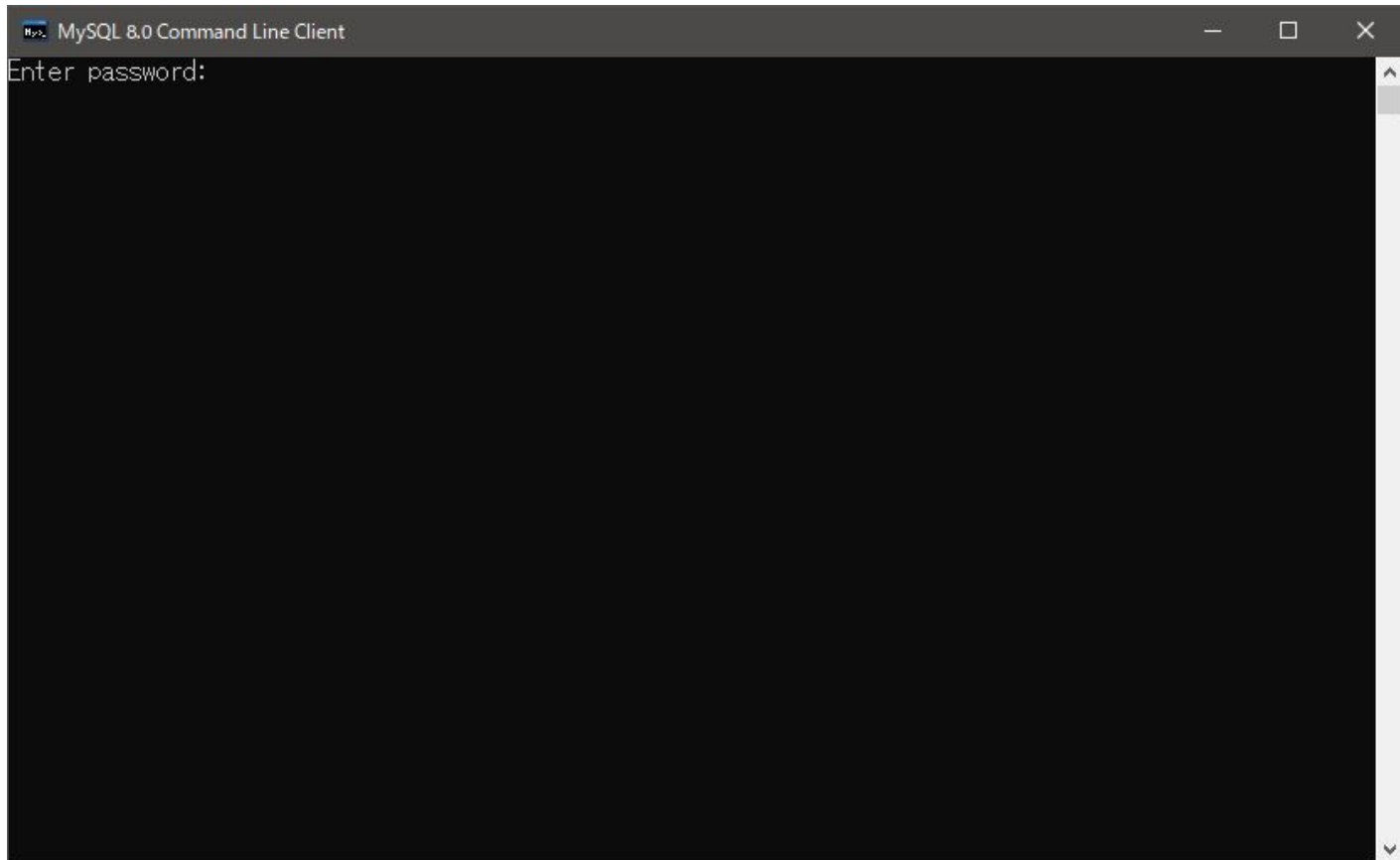
1. MySQLの公式ホームページにアクセス
<http://www.mysql.com/>
2. ダウンロードページからMySQL Community Editionをダウンロード
3. インストーラを起動してMySQLをインストール
4. MySQLへの接続確認



MySQLの操作

これまでに、実際に皆さんのマシンにはDBがインストールされたわけですが、実際にそれを触るにはどうすればいいのでしょうか？

MySQLでは**MySQL 8.0 Command Line Client**を起動しDBを操作します。





MySQLの操作

スタートメニューから、【MySQL Command Line Client】を起動します。

The image shows a Windows Start Menu on the left and a MySQL 8.0 Command Line Client window on the right. The Start Menu is open, displaying various applications. The 'MySQL 8.0 Command Line Client' is highlighted with a red box. A red arrow points from this box to a blue instruction box. The MySQL window is open, showing a black terminal with yellow text 'Enter password *****'. A red arrow points from a blue instruction box to this text.

1. スタートメニューをクリック
2. MySQL 8.0 Command Line Client をクリックし、起動
3. root のパスワードを入力してエンターを押す



いまの状態

DB使いたい人



MySQLにアクセス

アクセスユーザ
選択

データベース選択

テーブルからデータ取
得

MySQL



ROOT

MySQL管理領域



SQL

SQL(エスキューエル)とは、RDB(リレーショナルデータベース)を操作するための言語です。

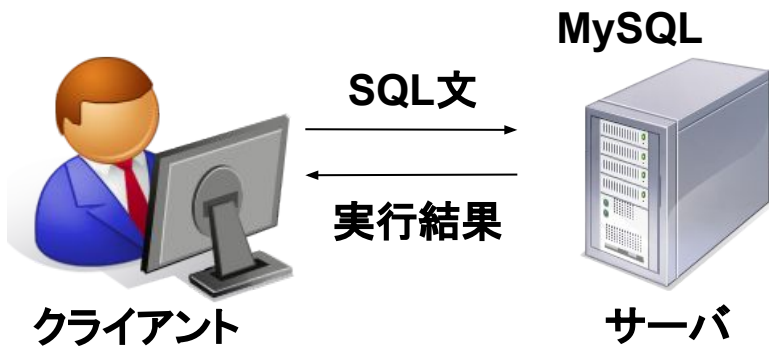
RDBを扱うのに、SQLが分からないと何もすることができません。

SQLは、ANSI(米国規格協会)やISO(国際標準化機構)で言語仕様の標準化が行われているため、

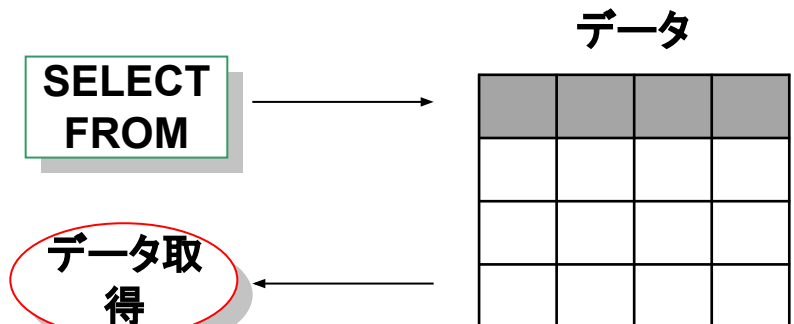
MySQL以外のRDBMS(RDBマネジメントシステム)でもほとんど同じように利用することができます。

※多少の違いはあります

SQLを使って、データを取得



SELECT文を使ってデータを取得

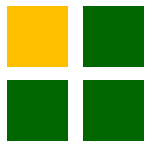




SQLの分類

SQLはデータを扱う上での分類がわかれています。

分類		SQL文
DML : Data Manipulation Language	データ操作	SELECT / INSERT / UPDATE / DELETE
DDL : Data Definition Language	データ定義	CREATE DATABASE / DROP DATABASE / CREATE USER / DROP USER / CREATE TABLE / DROP TABLE など..
DCL : Data Control Language	データ制御	START TRANSACTION / COMMIT / ROLLBACK / GRANT / REVOKE



DML

DMLは、データそのものを操作するときに用います。

業務においても非常に高い頻度で利用するSQLとなりますので、それぞれのSQL文の役割を覚えておきましょう。

SQL文	説明	例
SELECT	テーブルに格納されているデータを 取得 します	SELECT * FROM a_table
INSERT	テーブルにデータを 登録 します	INSERT INTO a_table VALUES (100, 'a')
UPDATE	テーブルに格納されているデータを 修正 します	UPDATE a_table SET id = 200
DELETE	テーブルに格納されているデータを 削除 します	DELETE FROM a_table



DDL

DDLはデータを格納するための場所を操作するときに用います。

データそのものを操作するわけではありませんが、データベースやユーザ、テーブルと操作する対象が多岐に渡るため、操作対象を明確にしておきましょう。

SQL文	説明	例
CREATE DATABASE	データベースを 作成 します	CREATE DATABASE a_db
DROP DATABASE	データベースを 削除 します	DROP DATABASE a_db
CREATE USER	ユーザを 作成 します	CREATE USER itc_user
DROP USER	ユーザを 削除 します	DROP USER itc_user
CREATE TABLE	テーブルを 作成 します	CREATE TABLE a_table (id INT ,name VARCHAR(10))
DROP TABLE	テーブルを 削除 します	DROP TABLE a_table



データベースの作成

次に、今後、DBアクセスに使用するユーザを新規作成します。
rootログインの状態で、rezodbというデータベースを作成します。

```
Enter password: *****
```

```
Welcome to the MySQL monitor.  Commands end with ; or \q.
```

```
Your MySQL connection id is 35
```

```
Server version: 8.0.13 MySQL Community
```

Enter password: インストール時に指定したパスワード

```
CREATE DATABASE [ データベース名 ];  
rezodb          今回作成するデータベース名
```

～中略～

```
Type 'help;' or '\h' for help. Type '\c' to clear the current  
input statement.
```

```
mysql> CREATE DATABASE rezodb;
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql>
```

まずは作業用のデータベースを作成します。
上記のように Query OK と表示されていれば、Databaseが作成される
ことになります。



いまの状態

DB使いたい人



MySQLにアクセス

アクセスユーザ
選択

データベース選択

テーブルからデータ取
得

MySQL



ROOT

MySQL管理領域

rezodb



ユーザの作成

次に、作業を行うユーザを作成します。

先ほど作成した rezodb を扱えるように設定するため、対象となるデータベースの操作権限も付与する必要があります。

```
mysql> CREATE USER 'rezouser'@'localhost' IDENTIFIED BY 'Rezo_0000';
```

```
Query OK, 0 rows affected (0.00 sec)
```

CREATE USER [ユーザ名@接続ホスト名] IDENTIFIED BY [パスワード] ;

```
mysql>
```

```
mysql>
```

```
mysql>
```

```
mysql>
```

```
mysql> GRANT ALL ON rezodb.* TO 'rezouser'@'localhost';
```

```
Query OK, 0 rows affected (0.05 sec)
```

GRANT ALL ON [データベース名] TO [ユーザ名@接続ホスト名] ;

```
mysql>
```

```
mysql>
```

```
mysql> \q
```

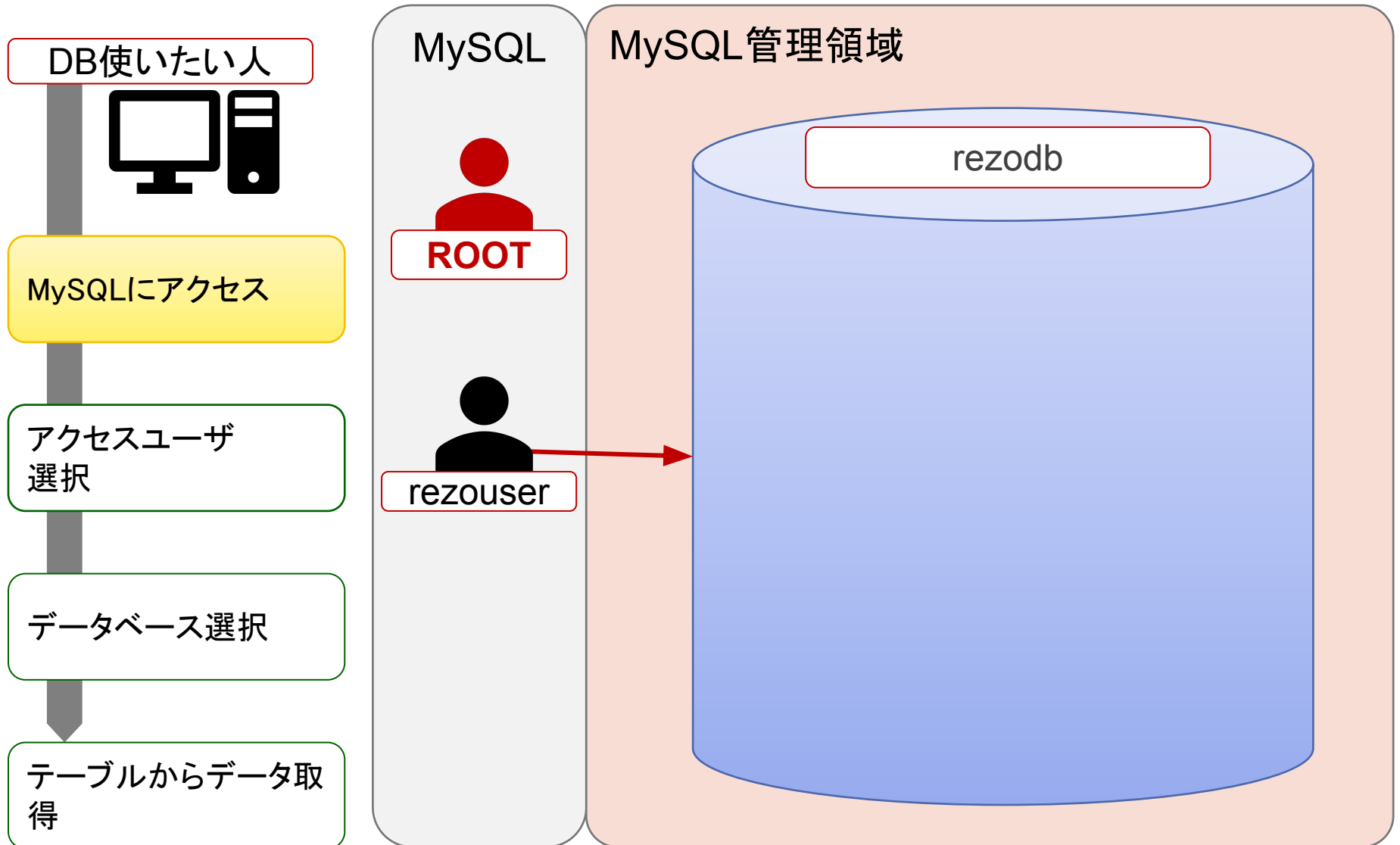
```
Bye
```

rezodb.* rezodbのすべてに対して付与する
'rezouser' @ 'localhost' 付与する対象ユーザ

\q はMySQLからログアウトする際に使用するコマンドです。
quit コマンドやexit コマンドでも同じことが可能です。



いまの状態





作成ユーザでのログイン

先ほど作成したユーザでMySQLにログインを行うには、少し手順が必要です。

MySQL Command Line Client でログインする場合、ユーザrootとなってしまうため、作成した rezouser ログインになりません。

方法は、以下の2つがあります。

1. **rezouser ログイン専用のショートカットを作成**
2. **環境変数でpathを設定し、コマンドプロンプトからログイン**

どちらの方法でも問題ありませんが、ここでは①の方法でログインを行う方法を設定します。

環境変数のpath設定の方法は、調査してみてください。

pathの設定を行えば、コマンドプロンプトから下記コマンドでログインできるようになります。

```
C:\Users\User>mysql -u [ユーザ名] -p [データベース名]
```



ログインのショートカット作成

先ほどMySQLにログインする際に起動した MySQL Command Line を

右クリック → コピー → (任意の場所に)貼り付け

この図は、MySQL Command Line Clientのショートカットを作成する手順を示しています。

1. Windows Startメニューで「MySQL 8.0 Command Line Client」を右クリックし、「スタートにピン留めする」を選択します。

2. File Explorerで「MySQL Server 8.0」を開き、「MySQL 8.0 Command Line Client - Unicode」と「MySQL 8.0 Command Line Client」のショートカットを確認します。これらをコピーします。

3. デスクトップに貼り付けると、MySQL 8.0 Command Line Clientのショートカットアイコンが作成されます。

4. ショートカットアイコンを右クリックし、「プロパティ」を選択します。

5. ショートカットの「プロパティ」ダイアログボックスで、任意の場所に貼り付けます。



ログインのショートカット作成

プロパティを開くと、右図のような画面が出てきます。
作業フォルダやリンク先などはインストールした場所により変動します。

リンク先の語尾の

修正前: `"-uroot" "-p"`

修正後: `"-urezouser" "-p" "rezodb"`

と変更しておきます。

これで先ほど作成したユーザでログインすることができます。





作成データベースへのログイン

ショートカットの作成を行えば、作成したユーザでのログインができるようになりますので、作成したユーザでログインしてみましょう。

```
Enter password: ****
```

Enter password: ユーザ作成時に指定したパスワード

```
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 5
```

～中略～

```
Type 'help;' or '\h' for help. Type '\c' to clear the current  
input statement.
```

```
mysql>
```

select database(); 現在ログインしているデータベースを確認

```
mysql> SELECT database();
```

```
+-----+  
| database() |  
+-----+  
| rezodb      |  
+-----+
```

```
1 row in set (0.00 sec)
```



テーブルの作成

ユーザ情報を格納するためのテーブル `user_table` を作成します。

テーブル作成SQLである CREATE 文 (DDL) を実行し、rezodb にテーブルを作成します。

```
mysql>
mysql> CREATE TABLE user_table (
  ->     user_id    INT,
  ->     user_name  VARCHAR(50),
  ->     tel_no     VARCHAR(50)
  -> );
```

Query OK, 0 rows affected (0.45 sec)

```
mysql>
```

テーブル作成クエリ

CREATE TABLE ([カラム名] [データ型], ...);

user_id	カラム名(ユーザID)
user_name	カラム名(ユーザ名)
tel_no	カラム名(電話番号)

INT	データ型(数値)
VARCHAR(50)	データ型(可変長文字列(50文字))

エラーメッセージが表示されず、上記のように CREATE TABLE と表示されていれば、テーブル作成クエリが正常に実行され、rezodb 内に `user_table` が作成されます。



いまの状態

DB使いたい人



MySQLにアクセス

アクセスユーザ
選択

データベース選択

このへん

テーブルからデータ取得

MySQL



ROOT



rezouser

MySQL管理領域

rezodb

user_id	user_name	tel
---------	-----------	-----

(まだ何も入っていない)

user_table



ユーザ情報の登録

ユーザ情報を格納するためのテーブルが作成できたので、そのテーブルに格納するデータを作成します。データを登録するINSERT文(DML)を実行し、rezodbの user_table にレコードを登録します。

レコード登録クエリ

```
INSERT INTO [ テーブル名 ] VALUES ( [ 値 ], ... );
```

```
mysql>
mysql> INSERT INTO user_table VALUES (1, '田中', '031112222');
Query OK, 1 row affected (0.07 sec)

mysql> INSERT INTO user_table VALUES (2, '鈴木', '057112222');
Query OK, 1 row affected (0.05 sec)
```

正しく登録されているか確認するために、SELECT文(DML)を実行します。

```
mysql> SELECT * FROM user_table;
```

上記のSQLを実行すると、指定したテーブルに登録されているレコードを表示することができます。

さきほど、2件のレコードを登録したため、2件のデータが表示されていることを確認してみてください。



いまの状態

DB使いたい人



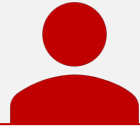
MySQLにアクセス

アクセスユーザ
選択

データベース選択

テーブルからデータ取
得

MySQL



ROOT



rezouser

MySQL管理領域

rezodb

user_id	user_name	Tel
1	田中	03 1111 2222
2	鈴木	057 111 2222

user_table



データ操作言語 (DML・基礎)

SELECT文

SELECT * FROM user_table;
選択する 全て から テーブル名



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
.	.	.

SELECT user_name FROM user_table;
選択する 名前を から テーブル名



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
.	.	.

SELECT user_name FROM user_table
選択する 名前を から テーブル名



WHERE user_id = 1;
ただし user_idが『1』のみ

複数のカラム名を指定する場合
カンマで区切ることが可能



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
.	.	.



データ操作言語(DML・基礎)

INSERT文

```
INSERT INTO user_table  
VALUES (10, '伊藤', '01011112222');
```

テーブル名の後にカラムを指定しない場合
全てのカラムに対応するデータを格納する



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
10	伊藤	0101112222

```
INSERT INTO user_table  
(user_id, user_name) VALUES (20, '高橋');
```

テーブル名の後にカラムを指定した場合
VALUES節には対応するデータのみ



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
20	高橋	



データ操作言語 (DML・基礎)

UPDATE文

```
UPDATE user_table  
  SET tel_no = '111122223'  
  WHERE user_id = 10;
```

WHERE節で条件を指定しない場合は
変更するカラムを全件更新するので注意



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
10	伊藤	111122223

DELETE文

```
DELETE FROM user_table  
  WHERE user_id = 10;
```

WHERE節で条件を指定しない場合は
全件を削除するので注意が必要



user_id	user_name	tel_no
1	田中	0311112222
2	鈴木	0571112222
10	伊藤	111122223



比較演算子

SELECTやUPDATE、DELETEではWHERE句を指定することができます。

これは条件を指定することで、対象となるレコードを絞り込むために使用しますが、その条件指定には比較演算子が使用されます。

演算子	説明	例
A = x	Aとxは等しい	user_no = 10001
A != x A <> x	Aとxは等しくない	user_no <> 10010
A > x	Aはxより大きい	user_no > 10002
A < x	Aはxより小さい	user_no < 10010
A >= x	Aはx以上	user_no >= 10001
A <= x	Aはx以下	user_no <= 10004
A in (x, y)	Aがxかyである	user_no in (10001, 10004)
A is null	Aがnullである	user_no is null



論理演算子

論理演算子は、複数の条件を付与するときに使用する演算子です。

例えば、連絡先が【0311112222】である【田中】という2つの条件を組み合わせで絞り込みを行いたい時に使用されます。

A(上記で言う連絡先=0311112222の部分)とB(上記で言う名前=田中)の2つの条件がありますが、【A且つB】なのか【AまたはB】なのかは非常に重要な部分にあたるのでしっかり覚えておきましょう。

演算子	説明	例
条件A AND 条件B	A且つB	user_name = '田中' AND tel_no = '0311112222'
条件A OR 条件B	AまたはB	user_name = '田中' OR tel_no = '0571112222'



制約とは

制約とは、テーブルに格納するデータを制限する方法です。

データ型によって整数であるか、文字列であるかといったデータの種類を限定することはできますが、それでは正数のデータのみを受け付けるとか、同じデータが重複してはいけないとかといった制限を行うことができません。

制約はこのような制限を列やテーブルに対して定義することができます。

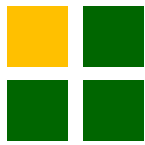
制約に違反するデータを格納しようとするすると制約違反のエラーが発生します。



各種制約一覧

● 制約一覧

制約名	解説
NOT NULL	NOT NULL制約は、単一の列に対して定義されるルールで、この列の NULL値(値がない)の入力を禁止するという制約です。即ち、テーブルのすべての行で必ずその列に NULL以外の値が格納されるように制限する制約です。
一意性 (UNIQUE)	一意性制約は、列、または複数の列の組み合わせに対して定義されるルールで、それらの列で値が一意(ユニーク)であるデータに限りデータの挿入・更新を許可する制約です。ユニークキーとも呼ばれます。
主キー (PRIMARY KEY)	主キー制約は、列、または複数の列の組み合わせに対して定義されるルールで、表の中の行を一意(ユニーク)に識別するための制約です。一意性制約と違って、主キーは1つのテーブルに1つしか設定出来ません。また主キー制約は、一意性制約に NOT NULL制約が加わったものとして見ることができます(つまり定義された列、または複数の列は重複が許されず NULLも許されない)。
外部キー (FOREIGN KEY)	外部キー制約は、テーブルとテーブルの間をテーブル内の列(または列の組み合わせ)で、参照という関連付けを定義するルールといいます。外部キー制約はテーブル同士の関連付けを行い、子テーブルは親テーブルを参照し、子テーブルの参照列を外部キーと呼びます。親テーブルと子テーブルの間でデータの整合性を保ちます。子テーブルに入力される値は親テーブルに存在する値か NULLでなければなりません。参照整合性制約やREFERENCE KEYなどとも呼ばれます。
チェック (CHECK)	チェック制約は、入力条件を定義した制約で、列に条件を満たす値のみ入力 that 許可されます。



制約の設定

- 使用頻度が高いNOT NULLと主キー

制約の中でも特に使用頻度が高いものがNOT NULL制約と主キー制約です。それぞれ、テーブル作成時に設定します。

右のSQLが制約を設定している例になります。

PRIMARY KEYが主キー設定している部分です。

右下は複数のカラムが主キーになる場合に使われる形になります。

もちろん、NOT NULLや主キー以外の制約もありますし、一意性や外部キーも使用頻度は決して低くありません。

他の制約に関しては公式リファレンスを参考にしてみましょう。

カラム定義に設定する方法

```
CREATE TABLE user_table (  
    user_id    INT          PRIMARY KEY  
    ,user_name VARCHAR(50) NOT NULL  
    ,tel_no    VARCHAR(50) NOT NULL  
);
```

テーブル定義に設定する方法

```
CREATE TABLE user_table (  
    user_id    INT  
    ,user_name VARCHAR(50) NOT NULL  
    ,tel_no    VARCHAR(50) NOT NULL  
    ,PRIMARY KEY(user_id)  
);
```

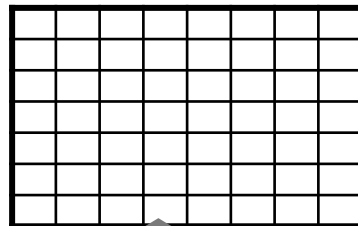


正規化

● 正規化

正規化とは、データベース内のデータを構築する処理のことです。

正規化には、データを保護しながらデータベースをより柔軟にするよう設計された規則に従い、冗長性の除去及び、矛盾する従属関係を除去することによってテーブルを作成する作業、及びテーブル間のリレーションシップを確立する作業が含まれます。



データ

1. 第一正規形

重複するデータを排除する操作のこと

2. 第二正規形

部分関数従属する項目を分離すること

3. 第三正規形

推移関数従属する項目を分離すること

4. その他の正規形

ボイスコッド正規形や第四～六正規形などが存在していますが、第三正規形までで十分なことが多いです。

業務においてもそこまで気にする必要はありません。



非正規形

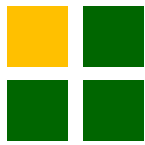
● 伝票や管理票のような必要情報のまとめ

小テスト	時間	教室	講師	受講生	会社名	メールアドレス	回数	受験日	点数
制御構文	30	A	山田	山田太郎	リゾーム	yamada@hoge.jp	1	4/20	7
				鈴木一郎	Kaisya	suzuki@hoge.jp	2	4/20	8
				佐藤ハナコ	Rhizome	sato@hoge.jp	1	4/20	5
				鈴木一郎	Kaisya	suzuki@hoge.jp	1	4/21	2
メソッド	15	A	山田	鈴木一郎	Kaisya	suzuki@hoge.jp	1	4/25	5
配列	15	B	鈴木	高橋三郎	ABC	taka@hoge.jp	1	4/25	3
			鈴木	田中次郎	ABC	tana@hoge.jp	1	4/25	4

上記のようなデータを非正規形と言います。

今回の例はテスト管理になりますが、伝票や評価表と言ったもので表現されるものですね。

実際の業務においても、伝票や管理票といったデータを扱いますが、テーブルでデータを扱いにくい形になっているため、これを分解しています。



第一正規形

- 繰返し項目をなくしてフラットに

<u>小テスト</u>	時間	教室	講師	<u>受講生</u>	会社名	メールアドレス	<u>回数</u>	受験日	点数
制御構文	30	A	山田	山田太郎	リゾーム	yamada@hoge.jp	1	4/20	7
制御構文	30	A	山田	山田太郎	リゾーム	yamada@hoge.jp	2	4/20	8
制御構文	30	A	山田	鈴木一郎	Kaisya	suzuki@hoge.jp	1	4/20	5
制御構文	30	A	山田	佐藤ハナコ	Rhizome	sato@hoge.jp	1	4/21	2
メソッド	15	A	山田	鈴木一郎	Kaisya	suzuki@hoge.jp	1	4/25	5
配列	15	B	鈴木	高橋三郎	ABC	taka@hoge.jp	1	4/25	3
配列	15	B	鈴木	田中次郎	ABC	tana@hoge.jp	1	4/25	4

第一正規形は、各データを1レコードとしてテーブルに格納していく形になります。

繰返しになっている情報を排除し、一番細かい単位にレコード化してフラットにします。

また、レコードとして一意に識別できるよう主キーを設定します。
主キーとなっているものは、上記例の赤下線の項目になります。



関数従属

- 項目ごとの関係性を知るところがスタート

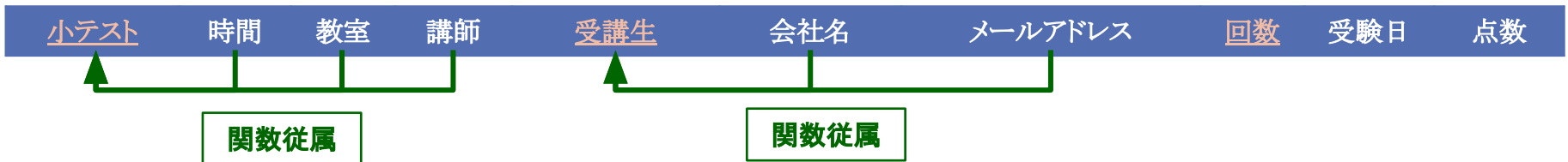
現状のテーブルに存在している項目の関係性を考えてみましょう。

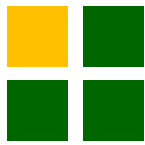
各項目がどの項目と関係があるのか、依存しているのかを見てみましょう。

①{小テスト=制御構文}であった場合、

②{時間=30・教室=A・担当=山田}ということが分かります。

このように、【①がわかれば②が決定する(①→②)】という関係を
【関数従属】と言います。





第二正規形

● 部分関数従属の項目を分離する

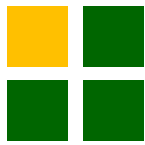
小テスト	時間	教室	講師
制御構文	30	A	山田
メソッド	15	A	山田
配列	15	B	鈴木

受講生	会社名	メールアドレス
山田太郎	リゾーム	yamada@hoge.jp
鈴木一郎	Kaisya	suzuki@hoge.jp
佐藤ハナコ	Rhizome	sato@hoge.jp
高橋三郎	ABC	taka@hoge.jp
田中次郎	ABC	tana@hoge.jp

小テスト	受講生	回数	受験日	点数
制御構文	山田太郎	1	4/20	7
制御構文	山田太郎	2	4/20	8
制御構文	鈴木一郎	1	4/20	5
制御構文	佐藤ハナコ	1	4/21	2
メソッド	鈴木一郎	1	4/25	5
配列	高橋三郎	1	4/25	3
配列	田中次郎	1	4/25	4

第二正規形は、第一正規形から部分関数従属の項目を分離することになります。

部分関数従属とは、主キーの一部に関数従属していることで、今回は小テストに関数従属しているものと受講生に関数従属しているものを分離して別テーブルに切り離しています。



第三正規形

● 推移関数従属の項目を分離する

小テスト	時間	教室
制御構文	30	A
メソッド	15	A
配列	15	B

教室	講師
A	山田
B	鈴木

受講生	会社名	メールアドレス
山田太郎	リゾーム	yamada@hoge.jp
鈴木一郎	Kaisya	suzuki@hoge.jp
佐藤ハナコ	Rhizome	sato@hoge.jp
高橋三郎	ABC	taka@hoge.jp
田中次郎	ABC	tana@hoge.jp

小テスト	受講生	回数	受験日	点数
制御構文	山田太郎	1	4/20	7
制御構文	山田太郎	2	4/20	8
制御構文	鈴木一郎	1	4/20	5
制御構文	佐藤ハナコ	1	4/21	2
メソッド	鈴木一郎	1	4/25	5
配列	高橋三郎	1	4/25	3
配列	田中次郎	1	4/25	4

第三正規形は、第二正規形から推移関数従属を分離することになります。

推移関数従属は、Aが決まればBが決まり、さらにCが決まるという関係従属のことで、A→Bは第二正規形で整理した部分になります。

B→Cの部分は主キーでない項目に関数従属している部分にあたり、それを取り除くことが第三正規形になるわけです。



サロゲートキー

● 主キーを代理キー(サロゲートキー)とする

テスト№	小テスト	時間	教室
1	制御構文	30	A
2	メソッド	15	A
3	配列	15	B

教室	講師
A	山田
B	鈴木

テスト№	受講生№	回数	受験日	点数
1	1	1	4/20	7
1	1	2	4/20	8
1	2	1	4/20	5
1	3	1	4/21	2
2	2	1	4/25	5
3	4	1	4/25	3
3	5	1	4/25	4

受講生№	受講生	会社名	メールアドレス
1	山田太郎	リゾーム	yamada@hoge.jp
2	鈴木一郎	Kaisya	suzuki@hoge.jp
3	佐藤ハナコ	Rhizome	sato@hoge.jp
4	高橋三郎	ABC	taka@hoge.jp
5	田中次郎	ABC	tana@hoge.jp

複数のテーブルに分離した際に考えていくべきことが、主キーについてです。

主キーは、【不変であること】が望ましいとされています。

名前は、変わる可能性がある・同姓同名の場合一意でなくなる可能性があります。

そのため、識別用に代理の主キー(サロゲートキー)をつけることが業務ではよくあります。

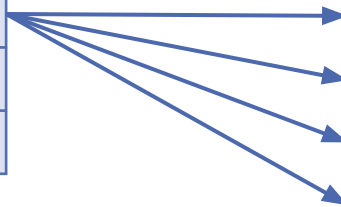


1対1／1対多／多対多の関係

RDBでは、通常複数のテーブルを用いてデータを整理します。

そのときのテーブル間のリレーション(関係)を見ていきましょう。

テスト№	小テスト	時間	教室
1	制御構文	30	A
2	メソッド	15	A
3	配列	15	B



テスト№	受講生№	回数	受験日	点数
1	1	1	4/20	7
1	1	2	4/20	8
1	2	1	4/20	5
1	3	1	4/21	2
2	2	1	4/25	5
3	4	1	4/25	3
3	5	1	4/25	4

上記のような関係を、【1対多】の関係と言います。

左右のテーブルは{テスト№}カラムをキーとして繋がっています。

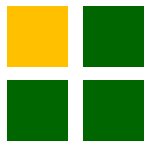
左テーブルが1件に対して、右側のテーブルでは複数件のレコードが存在しています。



1対1／1対多／多対多の関係

1対多の関係の他にも、テーブル間のリレーションには下記のようなタイプがあります。

リレーション	説明
1対多	テーブル間のレコードが1対多で対応している。 通常のRDBで最も見る状態で、特に理由がなければテーブル設計時に目指すべきリレーション。
1対1	テーブル間のレコードが1対1で対応しており、同様の主キーが設定されていることが多い。 この場合、分割しなくても良いテーブルを分割している状態。 業務上仕方がない場合や、パフォーマンスの関係でこのリレーションになることはあるが、特に理由がなければ避けるべき状態。
多対多	テーブル間のレコードが多対多で対応している。 RDBMSによっては、コレクション型や配列型で表現されたりもするが、通常は中間テーブルを定義し、1対多のリレーションを複数用意することで、このリレーションにならないよう注意する。

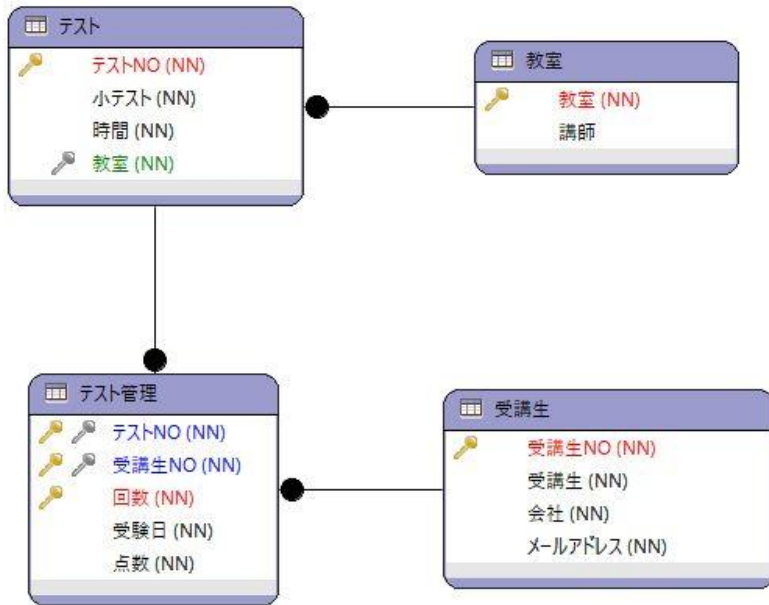


ER図

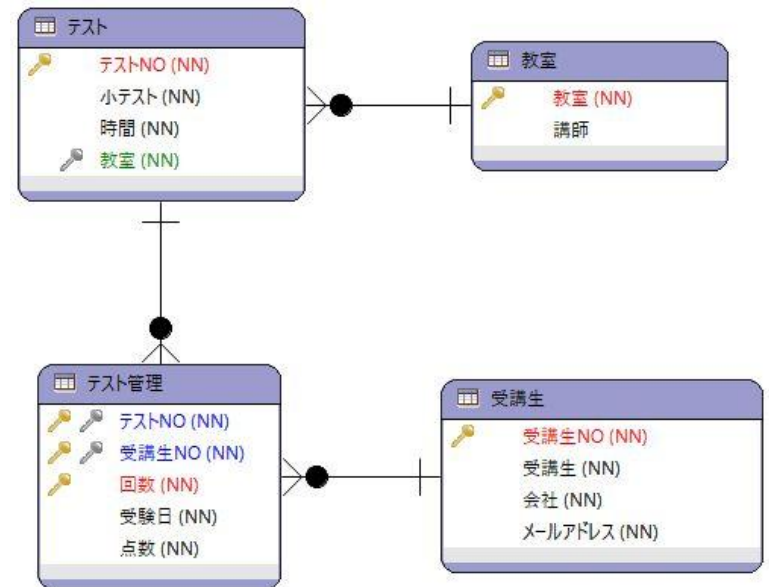
ER図(Entity-Relationship Diagram、ERD)とはデータ項目の集まりであるエンティティ(テーブル)と、エンティティ同士の繋がりであるリレーションシップを表したものです。

表記方法はいくつかありますが、1対多などのリレーションがわかりやすく、業務においてもよく扱われるテーブルの設計図になります。









IDEF1X(アイデフワンエックス)表記



IE(アイイー)表記



テーブル間のリレーションは線で表現されますが、表記によって表現が異なるため注意が必要です。

リレーション	IDEF1X表記	IE表記
1対多		
1対多(0含む)		
1対1		
1対0~1		

他にも表記方法がありますが、現在上記に挙げた2種類の方法が多く、業務でも多く見かけます。



マスタとトランザクション

テーブル間の関係や正規化によって複数のテーブルを使ってデータを整理しますが、マスタデータとトランザクションデータという概念を知っておきましょう。

● マスタデータ

- 業務を行う上で基礎情報となるデータのこと。
また、それらを集約したファイルやデータベースのテーブルなどをマスタと呼び、固定的なデータと認識されていることも多いが、あくまで基礎情報という点に注意。

▪ 例) 企業マスタ、顧客マスタ、テストマスタ など

● トランザクションデータ

- 企業の情報システムなどが扱うデータの種類の一つで、業務に伴って発生した出来事の詳細を記録したデータのこと。
流動的なデータとして認識されていることが多く、履歴や記録データをトランザクションとして扱う。

▪ 例) 試験結果データ、伝票データ、購入履歴データ など



テーブル結合

- 以下テーブルを作成

テスト№	小テスト	時間	教室
1	制御構文	30	A
2	メソッド	15	A
3	配列	15	B

教室	講師
A	山田
B	鈴木

受講生№	受講生	会社名	メールアドレス
1	山田太郎	リゾーム	yamada@hoge.jp
2	鈴木一郎	Kaisya	suzuki@hoge.jp
3	佐藤ハナコ	Rhizome	sato@hoge.jp
4	高橋三郎	ABC	taka@hoge.jp
5	田中次郎	ABC	tana@hoge.jp

テスト№	受講生№	回数	受験日	点数
1	1	1	4/20	7
1	1	2	4/20	8
1	2	1	4/20	5
1	3	1	4/21	2
2	2	1	4/25	5
3	4	1	4/25	3
3	5	1	4/25	4



テーブル結合

正規化を行うと、データの不整合が起きにくくなりますが、ほしい情報を得るためには、多くのテーブルを見なければならず、複雑さも増していきます。

データを検索するにしても、何回もSQLを実行しなければいけません。

例えば、【鈴木一郎】が【どの単元のテスト】で【何点取ったか】を知りたい場合は、以下の手順を踏まなければなりません。

1. 受講生名＝鈴木一郎で受講生Noを取得する
2. 取得した受講生Noでテスト名とテストNoを取得する
3. 取得した受講生Noで受験日と点数を取得する

上記のように、複数の手順が必要になります。

DBへのアクセスが増えれば、その分パフォーマンスも落ちてしまうため、アクセスを最小限に抑えるため、テーブル結合という手法を用います。



テーブル結合

テーブル結合は、複数のテーブルを繋げてあたかも1つのテーブルであるかのように振る舞うことができます。

リレーションシップ(関連付け)をもとに、項目同士を連結させ関連データを複数テーブルから取得するときに扱われます。

テーブル結合の書式

```
SELECT
    テーブル名. 列名 [, テーブル名. 列名……. ]
FROM
    テーブル名 INNER JOIN テーブル名
    ON テーブル名. 列名 = テーブル名. 列名
```

結合時は、列名に対してどのテーブルに存在しているかの明示的な指定が必要になります。

これは、同じ列名が結合した複数のテーブルに存在していたときに、どちらのテーブルから取得すれば良いのか、SQLでは判断できないためです。



テーブル結合

では、実際にどのようなSQLになるかを見てみましょう。

鈴木一郎が受験したテストの一覧を表示する

```
SELECT
  受講生. 受講生
, 受講生. 会社
, テスト. 小テスト
, テスト管理. 回数
, テスト管理. 受験日
, テスト管理. 点数
FROM
  テスト INNER JOIN テスト管理
    ON テスト. テストNO = テスト管理. テストNO
  INNER JOIN 受講生
    ON テスト管理. 受講生NO = 受講生. 受講生NO
WHERE
  受講生. 受講生 = '鈴木一郎';
```

抽出したい項目(列名)を記述
テーブル名を付与することを忘れずに！

INNER JOINの後に結合したいテーブル名、ON
の後ろに結合に使う列を記述します

INNER JOINは複数書いてもOK

複数のテーブルを繋げて検索することで、複数回実行しなければならなかったSQLが1回の実行で情報を取得することができるようになります。

結合に使用する項目をER図や関連を見て決めることが大切です。



テーブル結合

今回の例では、INNER JOINという結合を行っています。

内部結合と呼ばれる結合方法で、他に外部結合と呼ばれる結合方法も存在しています。

外部結合の書式

```
SELECT
    テーブル名. 列名 [, テーブル名. 列名…….]
FROM
    テーブル名 LEFT [OUTER] JOIN テーブル名
    ON テーブル名. 列名 = テーブル名. 列名
-- OUTERは省略可能です
```

左外部結合と呼ばれる方法で、左側のテーブルをベースとして検索し、結合した右側のデータは、左側と一致するもののみ表示されるという特性を持っています。

INNER JOINとLEFT OUTER JOINは、業務での使用頻度も非常に高いです。

RIGHT OUTER JOINというものも存在していますが、あまり見かける機会はありません。



サブクエリ(副問合せ)

サブクエリという手法も学んでおきましょう。

SQLの中にSELECT文を入れるという手法で、SELECT文を実行した結果をSQLに利用するという形になります。

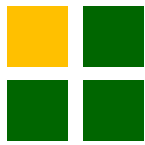
結合、サブクエリとどちらも使用頻度の高い手法となっています。

WHERE句に使った例

```
SELECT
    受講生. 受講生
    , 受講生. 会社
    , テスト. 小テスト
    , テスト管理. 回数
    , テスト管理. 受験日
    , テスト管理. 点数
FROM
    テスト INNER JOIN テスト管理
        ON テスト. テストNO = テスト管理. テストNO
    INNER JOIN 受講生
        ON テスト管理. 受講生NO = 受講生. 受講生NO
WHERE
    テスト管理. 受講生NO IN (
        SELECT 受講生NO
        FROM 受講生
        WHERE 受講生 = '鈴木一郎'
    );
```

FROM句に使った例

```
SELECT
    TEMP. 受講生
    , TEMP. 会社
    , テスト. 小テスト
    , テスト管理. 回数
    , テスト管理. 受験日
    , テスト管理. 点数
FROM
    テスト INNER JOIN テスト管理
        ON テスト. テストNO = テスト管理. テストNO
    INNER JOIN (
        SELECT 受講生NO, 受講生, 会社
        FROM 受講生
        WHERE 受講生 = '鈴木一郎'
    ) TEMP
        ON テスト管理. 受講生NO = TEMP. 受講生NO;
```



エイリアス(別名)

サブクエリにも使われていましたが、SQLではエイリアスという手法を使って、そのSQL内でのみ扱える別名をつけることができます。

FROM句に使った例の抜粋

```
(  
  SELECT 受講生NO, 受講生, 会社  
  FROM 受講生  
  WHERE 受講生 = '鈴木一郎'  
) TEMP
```

TEMP		
受講生NO	受講生	会社
2	鈴木一郎	Kaisya

取得結果を【TEMP】というテーブルとして扱う

上記のような形になり、TEMPテーブルという名前を付けてSQLを実行しています。

エイリアスはサブクエリに限った話ではなく、実在するテーブルそのものに対して別名を定義することも可能です。

テスト = A
テスト管理 = B
受講生 = C

とそれぞれのテーブル名に
別名をつけて扱っているSQL

エイリアスを使った例

```
SELECT  
  C. 受講生  
  , A. 小テスト  
  , B. 回数  
FROM  
  テスト A INNER JOIN テスト管理 B  
    ON A. テストNO = B. テストNO  
  INNER JOIN 受講生 C  
    ON B. 受講生NO = C. 受講生NO
```



ソート

ソートというのは、並び替えのことです。

SELECT文を用いてテーブルのデータを検索する際、テストの点数順で並べたり日付順に並べたりといった並び順を指定することができます。

ORDER BYという命令を使用して、任意のカラムで並び順を指定します。

右の図のように、

【ORDER BY カラム名】

を指定することによって、並び順を指定します。

ORDER BYで指定するカラムは複数指定も可能です。

カンマ区切りで指定すると、上から優先して並び順を指定する形になります。

値の小さい順に並べることを【昇順】、大きい順に並べることを【降順】と言います。

受験日の古い順

```
SELECT
    *
FROM
    テスト管理
ORDER BY
    受験日 ASC
```

-- ASCは省略可能

受験日の新しい順

```
SELECT
    *
FROM
    テスト管理
ORDER BY
    受験日 DESC
```



グルーピング

グルーピングとは、SELECTで検索したデータを指定のカラムでまとめることです。

指定カラムの値が同じものを1レコードとして取り出すことができます。

グルーピングには、GROUP BYというキーワードを使用します。

右の図のように、GROUP BYを使用することによって、指定された受験日カラムに登録されている値をまとめて表示しています。

GROUP BYも複数のカラムを組み合わせて行うことが可能です。

その場合は、カラム名をカンマ区切りで記述します。

通常のSQL

```
SELECT
  受験日
FROM
  テスト管理
```

受験日

4/20

4/20

4/20

4/21

4/25

4/25

4/25

グルーピングのSQL

```
SELECT
  受験日
FROM
  テスト管理
GROUP BY
  受験日
```

受験日

4/20

4/21

4/25



集約関数

集約関数は、複数のレコードから計算した結果を表示するSQLです。
データの件数、合計値、平均値、最大値などが計算結果にあたります。

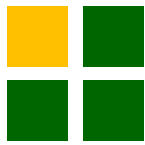
一番高い点数を取得する

```
SELECT  
    MAX(点数)  
FROM  
    テスト管理
```

テスト管理テーブルに登録されているデータから、最も高い点数を表示するSQL

上記の例は、MAX(カラム名)を指定することによって、カラムの中で最も大きい値を取得しています。

他にも多くの集約関数があり、取得したい計算結果によって使い分けます。



集約関数

● 集約関数の一覧(一部)

No	集約関数	説明
1	AVG()	平均値を取得します。
2	COUNT()	レコード件数を取得します。
3	MAX()	最大値を取得します。
4	MIN()	最小値を取得します。
5	SUM()	合計値を取得します。

上記にある一覧は、MySQLで使われる集約関数の一部です。

使用頻度が高く、業務でもよく使われるものになっていますので覚えておきましょう。

下記のURLは、MySQLの集約関数の公式ドキュメントです。

<https://dev.mysql.com/doc/refman/5.6/ja/group-by-functions.html>

※ バージョンが5.6となっていますが、日本語版のドキュメントは5.6以降提供していないようです



集約関数とグルーピング

集約関数は、GROUP BYと組み合わせて使用することが多いです。

例えば、【制御構文テストの平均点を知りたい】など、特定の範囲内で計算を行い、その結果を表示することができます。

右のSQL例がGROUP BYと集約関数を組み合わせた形になります。

WHERE句を外せば、制御構文に限らず各テストと平均点が表示される結果になります。

制御構文テストの平均点

```
SELECT
    T.小テスト
    , AVG(TM.点数)
FROM
    テスト T INNER JOIN テスト管理 TM
        ON T.テストNO = TM.テストNO
WHERE
    T.小テスト = '制御構文'
GROUP BY
    T.小テスト
```



あいまい検索

あいまい検索とは、検索条件が完全一致していない場合でも、一定のルールに基づいて抽出することです。

SQLでは、LIKE演算子とワイルドカード文字を使用することによって、条件が部分一致するようなレコードを抽出することが可能です。

右のSQL例がLIKEで検索を行った例になります。

比較演算子の部分にLIKEを記述し、検索条件には【%】というワイルドカード文字を加えた文字列が指定されています。

このSQLを実行すると、小テストカラムの値に、「構文」が含まれていれば検索対象になります。

【%】は任意の文字列を表しており、0文字以上となっています。

上記の例の場合「(0文字以上のなにか)構文(0文字以上のなにか)」で判定を行っているため、「構文」という文字がどこかにあれば検索対象になります。

LIKE+ワイルドカードを利用したSQL

```
SELECT
    *
FROM
    テスト
WHERE
    小テスト LIKE '%構文%'
```



ワイルドカード

● ワイルドカードの種類

文字	説明
% (パーセント)	0 個の文字も含めて、任意の数の文字に一致します
_ (アンダースコア)	正確に 1 つの文字に一致します

ワイルドカードは上記の2種類です。

このワイルドカードを使って、

1. 部分一致(文中のどこか(先頭・最後尾含む)に一致していれば良い)
2. 前方一致(先頭のみ一致していれば良い)
3. 後方一致(最後尾のみ一致していれば良い)

という検索条件の指定が可能になります。

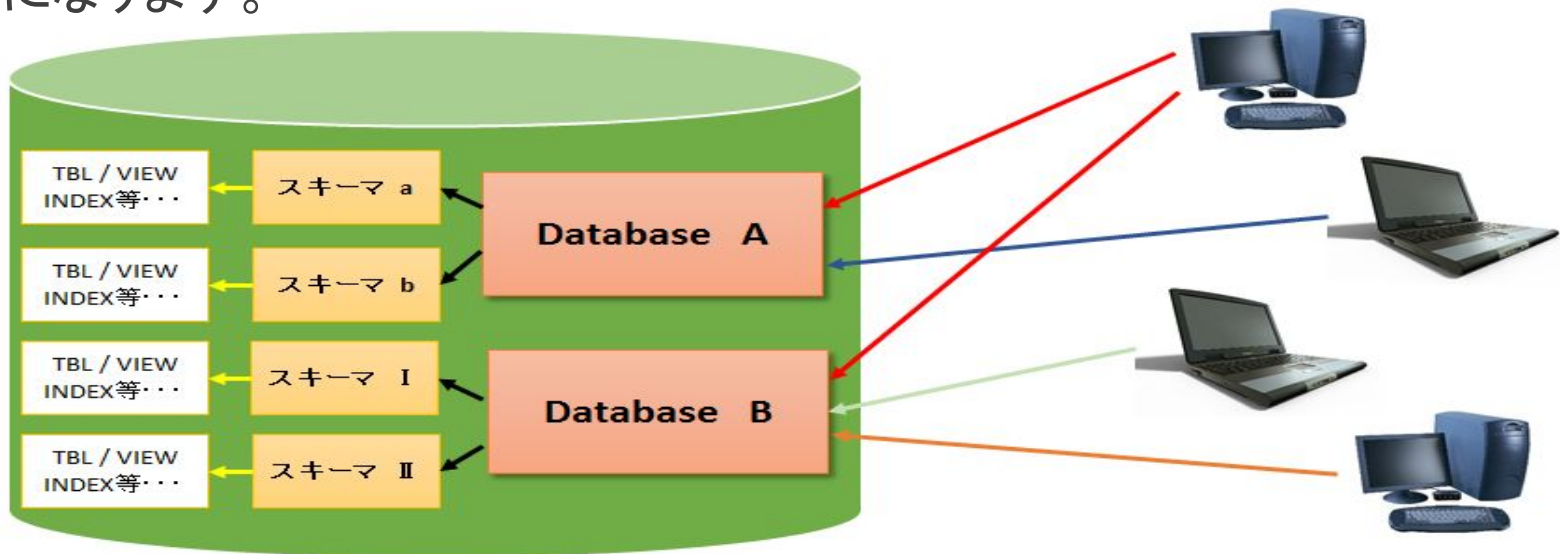


スキーマ

データベース上のオブジェクトを格納する名前空間のようなものです。

概念として、データベースインスタンス＞スキーマ＞テーブル、インデックス(等のオブジェクト)というような関係性があります。データベースを定義するということは、すなわちスキーマを定義することであり、データベースでいう設計図の役割を担っています。

ユーザはスキーマを扱う人の事であり、スキーマに対してのアクセスや権限を持つことによって、より詳細に役割や制限を設けることができます。





3層スキーマ

3層スキーマ構造とは、データベースの構造・形式(スキーマ)を3つの階層に分けてそれぞれ定義する方式で、現在ほとんどのDBMS製品で取り入れられています。

3つの層に分けてデータベースを定義することのメリットは、データの本質的な構造を概念スキーマで定義することで、ユーザ視点やデータベースの実装が変わっても、他に影響が無いデータベースを作ることができることです。

- **概念スキーマ(conceptual schema)**

データベースの全体基本構造を表現したものです。
RDBでは正規化されたテーブルに相当します。

- **内部スキーマ(physical schema)**

データベースを格納するストレージ(補助記憶装置)やデータファイル上でのデータの配置や格納方法を定義したものです。

ストレージにデータがどのように格納されているかといった具体的な部分は考慮していませんが、個別のデータベース製品におけるデータ格納は具体的に定義します。

- **外部スキーマ(external schema)**

ユーザやアプリケーションの立場からデータ構造やデータの関係を表現したものです。RDBではビューに該当します。複数の外部スキーマが定義されることになっても、概念スキーマには影響を与えないデータ構造となっていることを論理データ独立性といいます。



インデックス管理

インデックス管理はDBMSが持つ機能の一つで、検索を高速化する機能のことです。

インデックス(Index)とは索引のことで、データベースは膨大な量のデータを管理しますが、単純に検索を行うと時間がかかりすぎるため、あらかじめ小さな索引を作っておき、そこから検索を行うようにします。それによって膨大なデータを早く検索することができます。

- **パフォーマンス向上**

表の条件に使用される列に対して、インデックスを作成するとパフォーマンスが向上します。
値の分布が大きな列に対してインデックスを作成するとパフォーマンスが向上します。
値の分布が大きいとは異なる値が多いということです。

- **パフォーマンス低下**

値の分布が小さな列に対して、インデックスを作成するとパフォーマンスが低下します。
テーブルを更新すると、インデックスも更新されます。
テーブルが頻繁に更新されるような場合に、インデックスを利用するとパフォーマンスは低下します。

- **その他**

インデックスは表のデータとは別の領域に保存されるので、データベース設計時にはインデックスの領域も見込まなければなりません。



ストアドプロシージャ

SQLの問い合わせをサーバに持たせる技術をストアドプロシージャといいます。

クライアントサーバシステムにおいては、クライアントからDBに対して問い合わせ(SQL文の実行)が頻繁に行われると、ネットワークに負担がかかります。

これを軽減するのがストアドプロシージャです。

ストアドプロシージャはデータベース管理システムにある機能で、DBに対する一連の処理をまとめた手続きにして、リレーショナルデータベース管理システムに保存(永続化)したものです。

クライアントから引数を渡してそれに基づいて処理を行ったり、クライアントに処理結果を返したりすることもできます。

DB内部に保存してあるため、直接DB内で処理を行うことが可能で、処理速度もあがります。

システム構築を行う上では非常に重宝されます。



ストアドプロシージャ

- ストアドプロシージャを使用するメリット

メリット

ひとつの要求で複数のSQL文を実行できる(ネットワークに対する負荷を軽減できる)。

あらかじめ構文解析や内部中間コードへの変換をすませるため、処理時間が軽減される。

トリガ(イベントに反応して自動的に実行される操作)と組み合わせることで、複雑なルールによるデータの参照整合性保持が可能になる。

- ストアドプロシージャを多用することによるデメリット

デメリット

DB製品ごとに、記述する構文の規約がSQL/PSM規格との互換性が低いため、コード資産としての再利用性が悪い(各DBMSごとに書き方が異なる)。

ビジネスロジックの一部として利用する場合、業務の仕様変更に際して、外部のアプリケーションとともにストアドプロシージャの定義を変更する必要があるため、余計な手間や変更ミスによる障害を発生させる可能性がある。

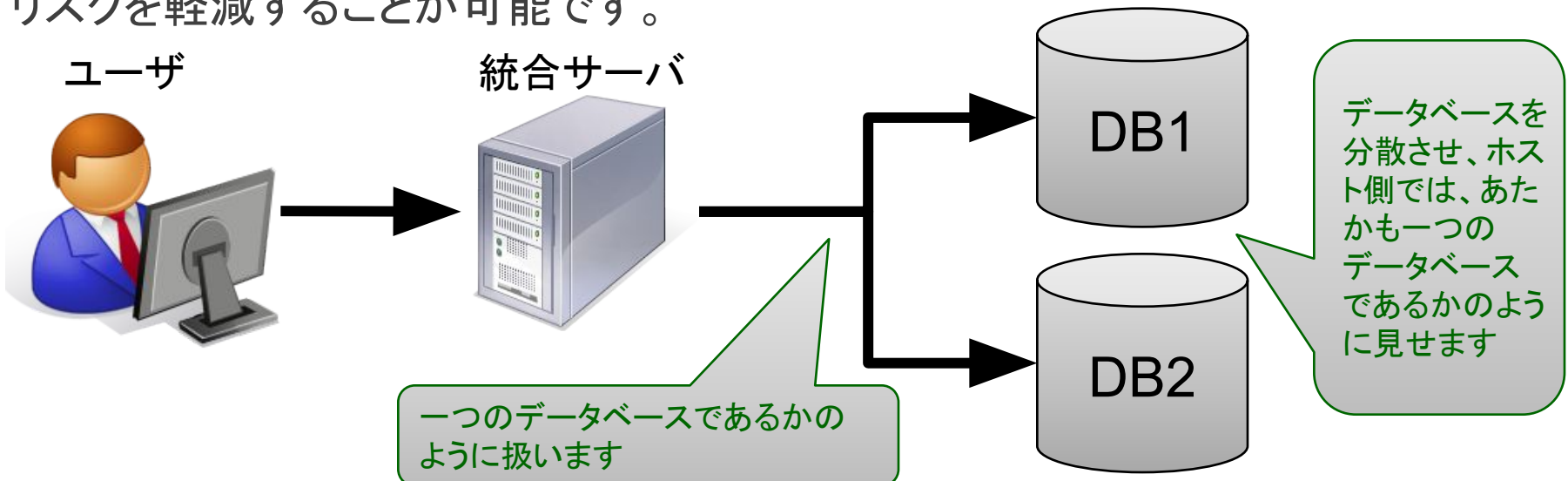


分散データベース

ネットワークに接続した複数のコンピュータが持っている複数のDBを、あたかも「単一のDB」であるかのように見せる技術です。

実際のDBは分散していますが、1台のコンピュータで集中管理しているように利用できます。物理的には分散、論理的には集中という利用方法です。

膨大な量の情報を1カ所に集中している場合、安全面や性能面での問題が生じる可能性があります。分散データベースでは、それらの問題へのリスクを軽減することが可能です。



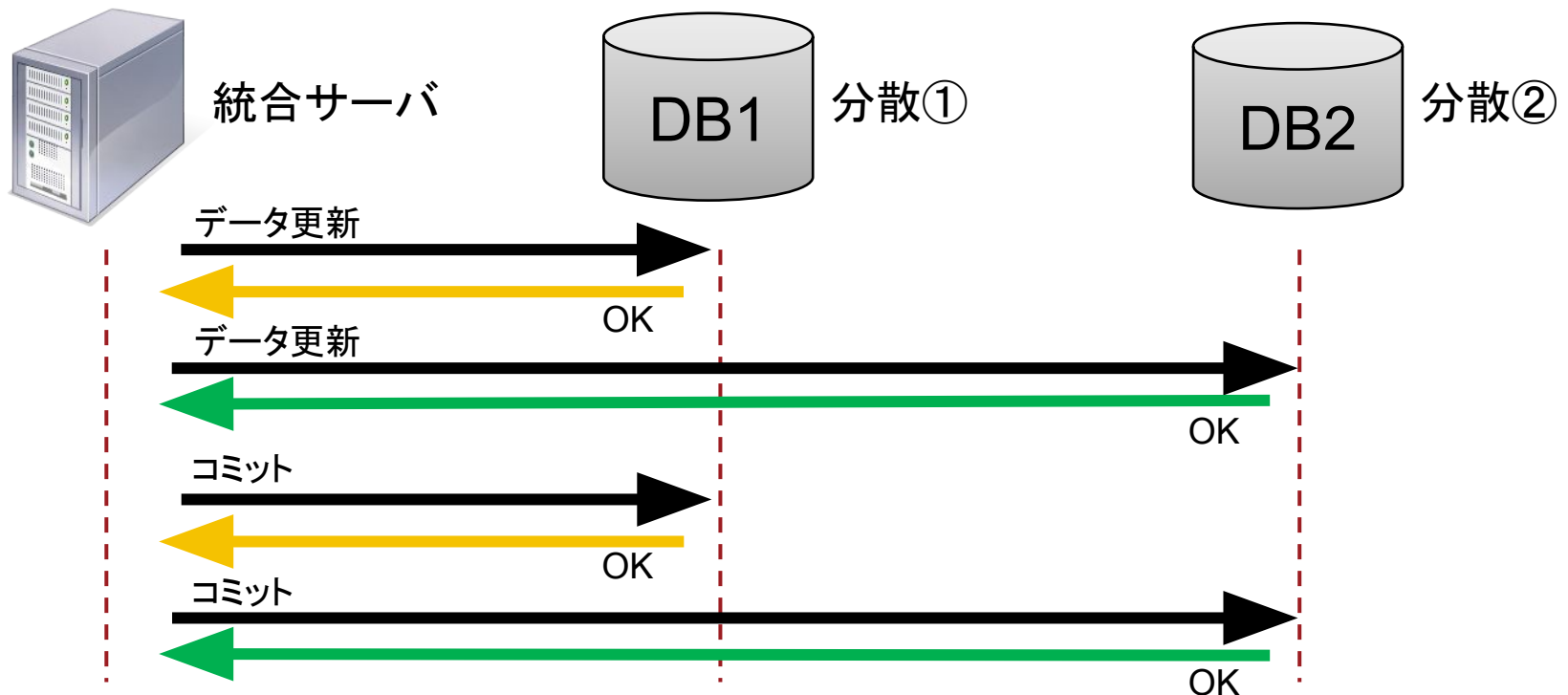


2相コミット

分散データベースはデータが分散しているため整合性に注意が必要です。

一つのサイトだけ更新され、別のサイトが更新されないことがあると、データの整合性が取れなくなってしまうのです。

データを操作する際に使われている方法が2相コミットです。

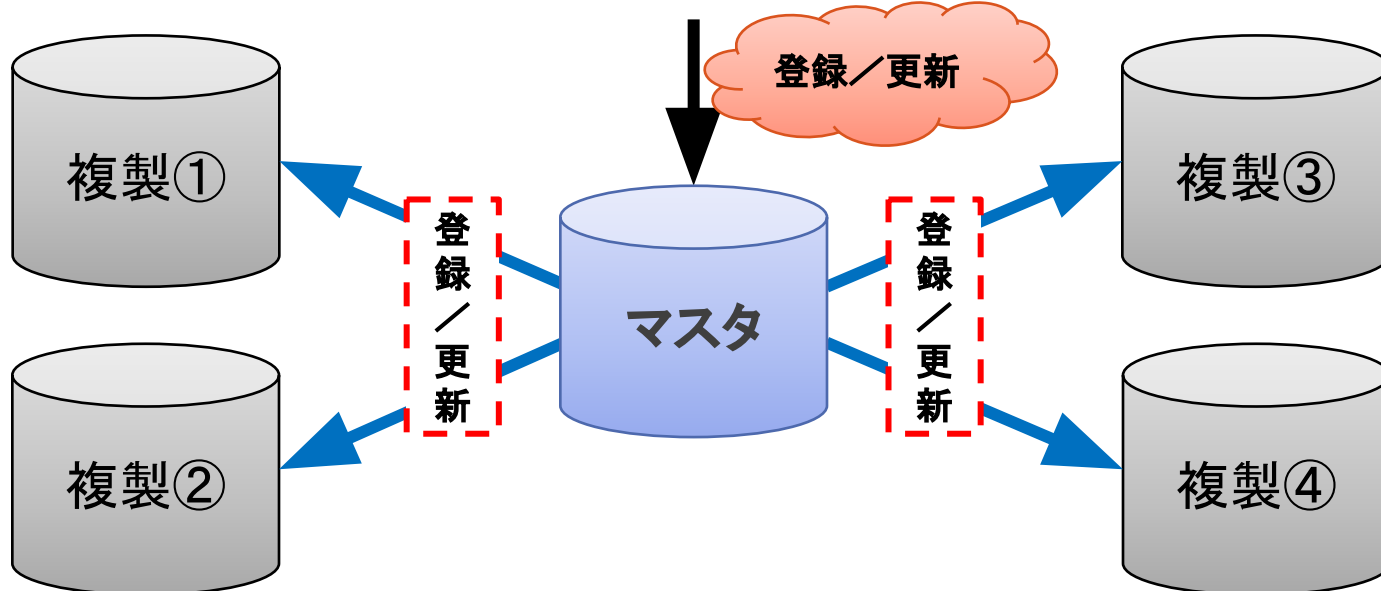




レプリケーション

レプリケーションは、DBMSが持つ機能の一つで、あるデータベースと全く同じ内容の複製(レプリカ)を別のコンピュータ上に作成し、常に内容を同期させる機能です。

負荷分散や耐障害性の向上などを目的に行われる機能です。マスターデータベースとレプリカは通信ネットワークなどを通じて互いにデータを交換しあい、常に内容が一致するようにできている為、一か所でデータを更新すると、マスターとすべてのレプリカに更新内容が伝播します。





トランザクション制御

ユーザによる検索や更新等の一続きの手続きをトランザクションと
いいます。

データベースは複数のユーザがデータを共有して同時に操作を行うことを前提と
しています。

DBMSには、そのトランザクションを制御する機能がついており、2つのトランザ
クションが並行して行われるとき、データの矛盾が起こることを防ぐため、データ
の更新中はアクセス制限(ロック)をかけて、別の
トランザクションが更新できないようにします。

- **占有ロック**

データベースを更新する際にかけるロックです。

他のユーザはロックをかけたり、データを参照したり、更新したりすることは
できません。

- **共有ロック**

データを参照する際にかけるロックです。

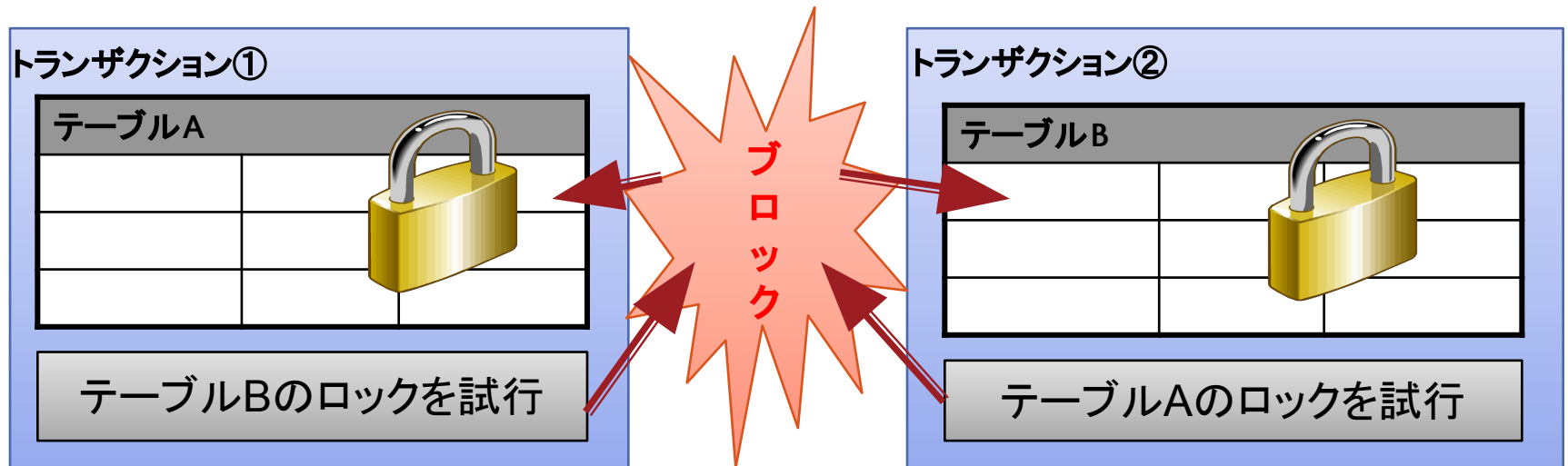
他のユーザは共有ロックをかけたり、データを参照したりすることができます。
ただし、占有ロックをかけたり、データを更新することはできません。



デッドロック

ロックを使う場合には、お互いにロック解除待ちを行うことで、処理が進まなくなるデッドロックが起こる可能性があります。

ロックを使う場合には、デッドロックを避ける処理も必要となります。



1. テーブルAの更新が完了
2. テーブルBの更新を行う為、ロックを実行
3. ロック中の為、テーブルAのロックを保持

1. テーブルBの更新が完了
2. テーブルAの更新を行う為、ロックを実行
3. ロック中の為、テーブルBのロックを保持



障害管理とバックアップ

ハードディスクの故障や停電等の障害によって、データベースに不整合が起こる場合があります。

DBMSはこうした障害に備えて、一定期間ごとにバックアップをとる機能を備えています。

また、更新の際にログと呼ばれる記録を取っています。

ログには更新前と更新後の値を記録します。障害が発生した場合には、ログを利用して障害回復を行います。

通常の業務においては長いスパン(1週間に一度程度)で全バックアップを行い、短いスパン(毎日)で差分バックアップを行うことで、日々の障害に対応しています



障害管理とバックアップ

● 全バックアップ

全データを別の媒体に作成し、作業データと全く同じ状態を保持しておきます。

全データをバックアップするため、データのバックアップ時間が長くなりますが、復旧の際にはそのままデータの移植を行えばいいため、短時間で復旧が可能です。

● 差分バックアップ

バックアップファイルを作成するとき、効率的な方法として、定期的にファイル全体のバックアップファイルを新しい媒体に作成し、毎日変更のあった部分のデータだけを別の媒体に作成する方法です。

バックアップの時間が短くて済みますが、復旧には全バックアップのデータを移植した後、さらに差分バックアップを反映させる必要があるため、復旧時間が長くなります。

日	月	火	水	木	金	土	日	月	火	...	日
フル	差分	増分	増分	差分	増分	増分	フル	差分	増分	...	フル



障害管理とバックアップ

- **ロールフォワード**

更新後にログを使って障害直前の状態にまで復旧させる処理です。
障害発生時までには手続きが完了しているトランザクションには、
ロールフォワードを行います。

物理的障害の対応に用いられます。

バックアップ復元後にバックアップ取得時点から最新のデータまで、ジャーナルファイルを使用して復旧する作業です。

- **ロールバック**

更新前のログを使い、トランザクション開始直前の状態にまでデータを復旧させる処理です。

障害発生時までには完了していないトランザクションにはロールバックを行い、
トランザクション開始前の状態に戻します。

論理的障害の対応に用いられます。



ジャーナルファイル

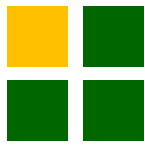
- ジャーナルファイル(ログファイル)

ロールフォワードやロールバックに使用される更新記録を残したファイルをジャーナルファイル又はログファイルといいます。

ログファイルは、「更新前情報」、「処理内容情報」及び「更新後情報」があり、処理前のデータベースの内容を更新前情報に記録し、更新した後のデータベースの内容を更新後情報に記録します。

近年では、WindowsやMacなどのOSファイルシステムもジャーナルファイルを取り入れており、実データとメタデータの不整合をなくすような仕組みになっています。

このジャーナルを用いた、ファイル管理を行っているシステムはジャーナルファイルシステムと呼ばれています。



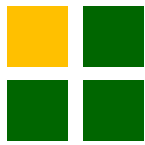
参考資料・DDL文

CREATE (DATABASE) 文

書式	CREATE DATABASE database_name database_definition;
引数	database_name — データベース名 database_definition — 作成するデータベースの内容
解説	CREATE DATABASE文を実行すると、汎用的なデータベースが作成される。 CREATE DATABASEに続いて、作成するデータベース名 (database_name)を記述する。 作成するデータベースの定義内容は、使用する文字、言語や制御ファイルの定義、 ログファイルの定義、アーカイブの指定、データファイルの定義、表領域などがある。

CREATE (TABLE) 文

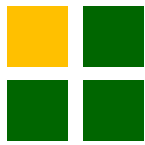
書式	CREATE TABLE table_name [(column_definition [, column_definition])];
引数	table_name — テーブル名 column_definition — 列定義
解説	CREATE TABLE文を実行すると、テーブルが作成される。 CREATE TABLEに続けて、テーブル名 (table_name)を、カッコ内に列名、型、制約などを 記述する。



参考資料・DDL文

ALTER TABLE文

書式	ALTER TABLE table_name ADD column_definition ALTER TABLE table_name ADD CONSTRAINT constraint_definition ALTER TABLE table_name DROP column_name ALTER TABLE table_name DROP CONSTRAINT constraint_name		
引数	table_name	—	表名
	column_definition	—	列定義
	column_name	—	列名
	constraint_definition	—	制約定義
解説	<p>列の定義を変更する。</p> <p>変更方法はおもにADD, DROPの2つに分けられる。</p> <p>ALTER TABLEに続き、表名を記述する。</p> <p>ADDに続き列定義 (column_name)を記述し、追加または修正する列定義を指定する。列定義は列名とデータ型を1つ以上指定可能である。</p> <p>複数の列を追加する場合は括弧でくる。</p> <p>また、CONSTRAINTにより制約を追加、修正することもできる。</p> <p>DROPに続き列名 (column_name)を記述し、列を削除する。</p> <p>列名は1つ以上指定可能であるが、複数の場合は括弧でくる。CONSTRAINTにより制約の削除も可能である。</p>		



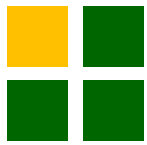
参考資料・DDL文

DROP TABLE文

書式	DROP TABLE <i>table_name</i> [CASCADE];
引数	<i>table_name</i> — 表名
解説	<p>表と表に含まれるすべてのデータをデータベースから削除する。 DROP TABLE に続けて、テーブル名 (<i>table_name</i>) を記述する。 CASCADE を指定した場合は、主キーや一意キーを参照しているすべての参照整合性制約を削除する。 * DROP TABLE文はロールバックできないので、注意が必要である。</p>

各SQLの詳細な構文については公式のリファレンスを参照してください。

<https://dev.mysql.com/doc/refman/5.6/ja/>



参考資料・DML応用

● SQLキーワード一覧(よく使うもの)

キーワード	概要	使用例
WHERE	欲しい情報の条件を指定する	<code>SELECT * FROM usertable WHERE userid = 111;</code>
AND/OR	条件を複数指定する場合に使用する	<code>SELECT * FROM usertable WHERE userid = 111 AND(OR) username = 'alice' ;</code>
LIKE	抽出条件に部分一致や前方一致を行いたい場合に使用する	<code>SELECT * FROM usertable WHERE username LIKE ' %c%' ;</code>
IN	列の値が複数の候補のいずれかと一致する場合の抽出条件	<code>SELECT * FROM usertable WHERE userid IN (111, 222);</code>
BETWEEN	列の値がある範囲に存在する場合の抽出条件	<code>SELECT * FROM usertable WHERE userid BETWEEN 333 AND 666;</code>
DISTINCT	重複しているデータを1つにまとめて表示する	<code>SELECT DISTINCT username FROM usertable;</code>
COUNT	レコードの合計数を表示する	<code>SELECT COUNT(*) FROM usertable;</code>
SUM	合計値を求める	<code>SELECT SUM(userid), username FROM usertable GROUP BY username;</code>
GROUP BY	集計関数をSELECTで指定した場合に使用する	<code>SELECT SUM(userid), username FROM usertable GROUP BY username;</code>
ORDER BY	問合せ結果を並び替える(昇順)	<code>SELECT * FROM usertable ORDER BY userid;</code>