01. データベースとは

◇ 関係データベース

現在、主流になっているデータベース

- **MariaDBのデータベース**
- **MySQLのデータベース**
- **PostgreSQLのデータベース**
- **Amazon Aurora**

◇ DBMS（データベース管理システム）

- **MariaDB**
- **MySQL**
- **PostgreSQL**
- **Amazon RDS：Amazon Relational Database Service**

◇ データベースエンジン

- **InnoDB**

02. データベースの設計

◇ UMLによる設計

16章を参照。Squidの設計図では、UMLとER図を組み合わせている。

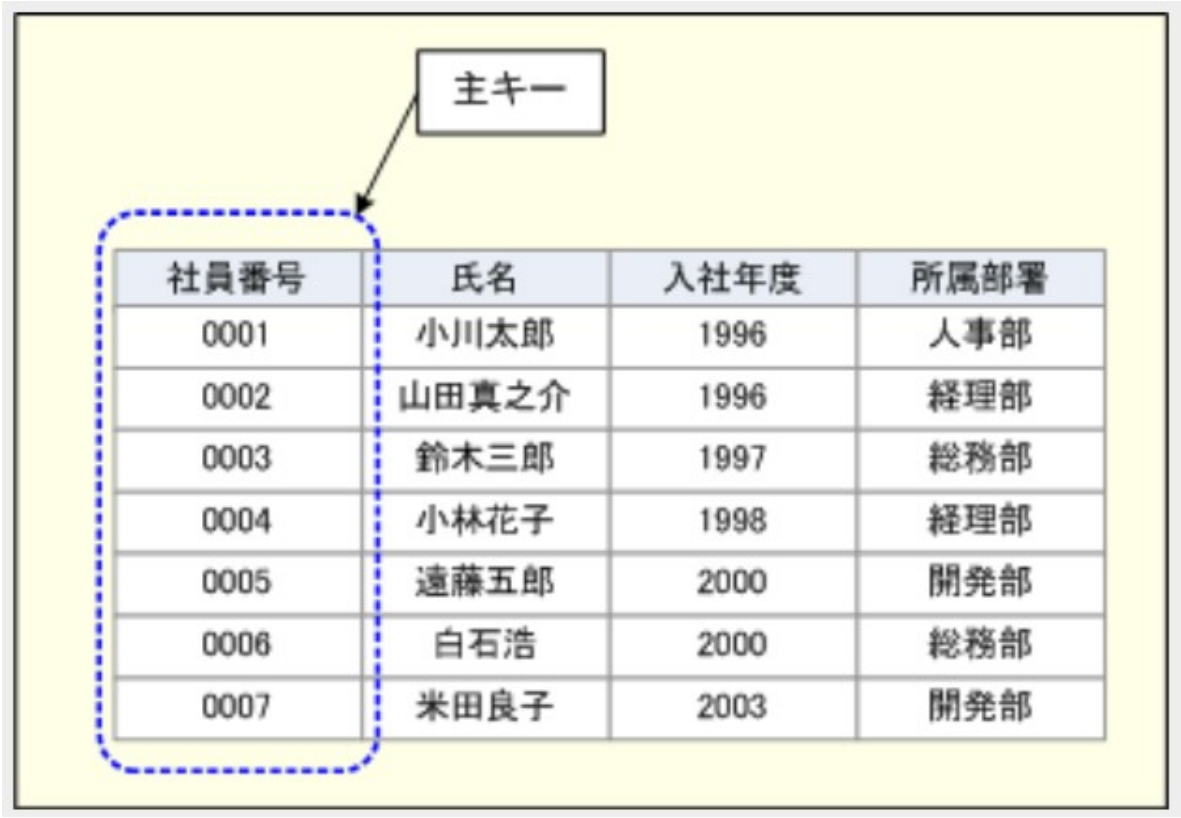
◇ ER図による設計：Entity Relation Diagram

16章を参照。Squidの設計図では、UMLとER図を組み合わせている。

03. テーブルの作成

◇ 主キー

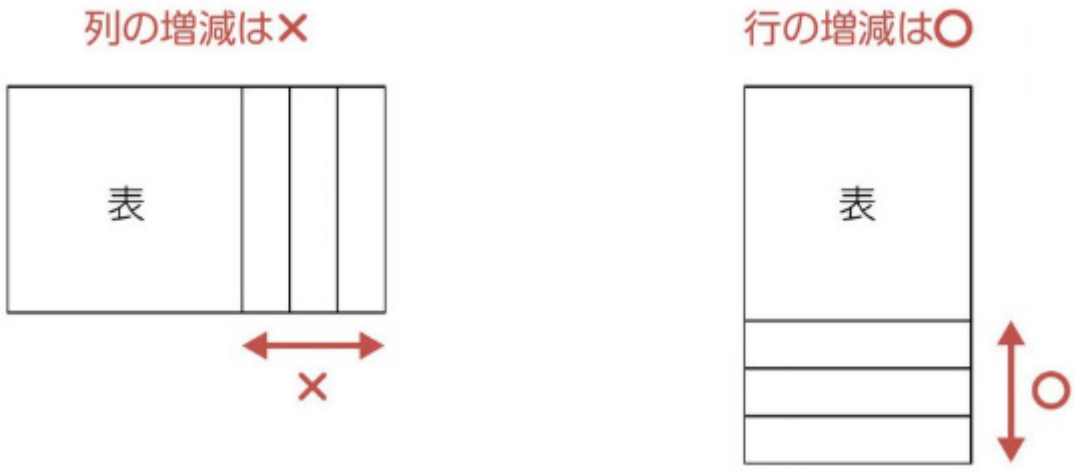
テーブルの中で、Rowデータを一意に特定できる値を『主キー』の値と呼ぶ。



社員番号	氏名	入社年度	所属部署
0001	小川太郎	1996	人事部
0002	山田真之介	1996	経理部
0003	鈴木三郎	1997	総務部
0004	小林花子	1998	経理部
0005	遠藤五郎	2000	開発部
0006	白石浩	2000	総務部
0007	米田良子	2003	開発部

◇ データを追加するあるいは削除する場合の注意点

データを追加するあるいは削除する場合、カラムではなく、レコードの増減を行う。カラムの増減の処理には時間がかかる。一方で、レコードの増減の処理には時間がかからない。



【具体例】

賞与を年1回から、2回・3回と変える場合、主キーを繰り返し、新しく賞与区分と金額区分を作る。

× [賞与3] 列を作り，
そこに金額を格納する

➡列を増やす

番号	…	賞与1	賞与2	賞与3
101	…	30万円	20万円	10万円

○ [区分] 列に「賞与3」を，
[金額] 列に金額を格納する

➡(行を増やすため) 列を増やさない

番号	…	区分	金額
101	…	賞与1	30万円
101	…	賞与2	20万円
101	…	賞与3	10万円

◇ 非正規形の正規化（最適なカラム設計）

『正規化』と『非正規形』：繰り返し要素のある表を正規形、その逆を非正規形という。

『正規化』：非正規形の表から、他と連動するカラムを独立させ、正規形の表に変更すること。

【具体例】

1. エクセル

   エクセルで作られた以下の表があると仮定。

繰返し要素
(行が結合している部分)

非正規形

受注No*4	日付	顧客ID	顧客名	商品ID	商品名	数量	単価
1001	8/30	002	B社	A	鉛筆	50	100
				B	定規	20	50
1002	3/7	001	A社	A	鉛筆	20	100
				C	ペン	10	200

2. 第一正規化：**繰り返し要素の排除**

   レコードを1つずつに分割。

第1正規形

受注No	日付	顧客ID	顧客名	商品ID	商品名	数量	単価
1001	8/30	002	B社	A	鉛筆	50	100
1001	8/30	002	B社	B	定規	20	50
1002	3/7	001	A社	A	鉛筆	20	100
1002	3/7	001	A社	C	ペン	10	200

3. 第二正規化：**主キーの関数従属性を排除**

   特定のカラムが連動している（関数従属性）場合、カラムを左表として独立させる

① 主キーの１列に値が連動する列（連動列）を探す。

主キーの１列[受注 No]列と値が連動する列(連動列)は，[日付]・[顧客 ID]・[顧客名]。

② 主キーの１列[受注 No]と，連動列[日付]・[顧客 ID]・[顧客名]を別表にする。

受注No	日付	顧客ID	顧客名
1001	8/30	002	B社
1001	8/30	002	B社
1002	3/7	001	A社
1002	3/7	001	A社

③ 重複行を削除する。

受注No	日付	顧客ID	顧客名
1001	8/30	002	B社
1002	3/7	001	A社

②’ 元の表から連動列[日付]・[顧客 ID]・[顧客名]を削除する。

受注No	商品ID	商品名	数量	単価
1001	A	鉛筆	50	100
1001	B	定規	20	50
1002	A	鉛筆	20	100
1002	C	ペン	10	200

③’ 重複行を削除する。

（重複行がないため，そのまま）

受注No	商品ID	商品名	数量	単価
1001	A	鉛筆	50	100
1001	B	定規	20	50
1002	A	鉛筆	20	100
1002	C	ペン	10	200

主キーの１列［受注 No］列は，両表ともに残す。

上で分割して生じた右表のカラムに関数従属性があるので、従属性のあるカラムを左表として独立させる。

① 主キーの１列に値が連動する列（連動列）を探す。

主キーの１列［商品 ID］に値が連動する列（連動列）は，［商品名］・［単価］。

② 主キーの１列[商品 ID]と，連動列[商品名]・[単価]を別表にする。

商品ID	商品名	単価
A	鉛筆	100
B	定規	50
A	鉛筆	100
C	ペン	200

③ 重複行を削除する。

商品ID	商品名	単価
A	鉛筆	100
B	定規	50
C	ペン	200

②’ 元の表から連動列[商品名]・[単価]を削除する。

受注No	商品ID	数量
1001	A	50
1001	B	20
1002	A	20
1002	C	10

③’ 重複行を削除する。

（重複行がないため，そのまま）

受注No	商品ID	数量
1001	A	50
1001	B	20
1002	A	20
1002	C	10

主キーの１列［商品 ID］列は，両表ともに残す。

第２正規形

受注No	日付	顧客ID	顧客名
1001	8/30	002	B社
1002	3/7	001	A社

商品ID	商品名	単価
A	鉛筆	100
B	定規	50
C	ペン	200

受注No	商品ID	数量
1001	A	50
1001	B	20
1002	A	20
1002	C	10

4. 第三正規化：**主キー以外のカラムの関数従属性を排除**

上で分割して生じた左表のカラムに関数従属性があるので、従属性のあるカラムを左表として独立させる。

① 主キー以外の列に値が連動する列を探す。

第２正規形のうち，左側の表に主キー以外の列に関数従属性がある。

主キー以外の列［顧客 ID］に値が連動する列（連動列）は，［顧客名］。

② 主キー以外の列［顧客 ID］と，連動列［顧客名］を別表にする。

顧客ID	顧客名
002	B社
001	A社

③ 重複行を削除する。

（重複行がないため，そのまま）

顧客ID	顧客名
002	B社
001	A社

②’ 元の表から連動列［顧客名］を削除する。

受注No	日付	顧客ID
1001	8/30	002
1002	3/7	001

③’ 重複行を削除する。

（重複行がないため，そのまま）

受注No	日付	顧客ID
1001	8/30	002
1002	3/7	001

第３正規形

顧客ID	顧客名
002	B社
001	A社

受注No	日付	顧客ID
1001	8/30	002
1002	3/7	001

商品ID	商品名	単価
A	鉛筆	100
B	定規	50
C	ペン	200

受注No	商品ID	数量
1001	A	50
1001	B	20
1002	A	20
1002	C	10

この表は主キー以外の関数従属性がなく，第３正規化の対象外のため，第２正規形からそのまま持ってくる。

04. データベースに必要な機能

◇ ACID特性

トランザクションを実現するためには、以下の４つの機能が必要である。

- **Atomicity**

  コミットメント制御によって実装される。

- **Consistency**

  トランザクションの前後でデータ排他制御によって実装される。

- **Isolation**

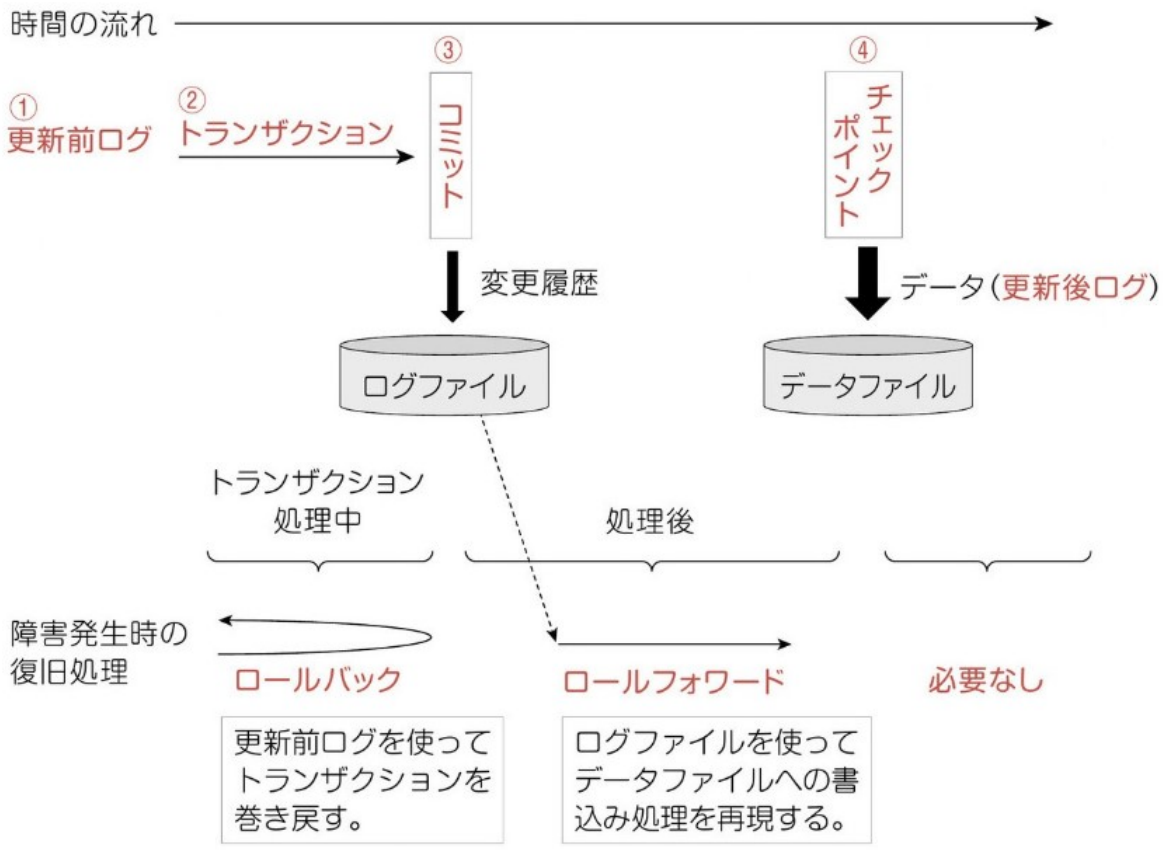
  排他制御によって実装される。

- **Durability**

  障害回復制御によって実装される。

05. コミットメント制御と障害回復制御

◇ データベース操作の全体像



【実装例】

```
try{
    // データベースと接続。
    $db = getDb();
```

```
    // 例外処理を有効化。
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // トランザクションを開始。
    $db->beginTransaction();
    // いくつかのSQLが実行される。※もし失敗した場合、ERRMODE_EXCEPTIONを実行。
    $db->exec("INSERT INTO movie(title, price) VALUES('ハリポタ', 2000)")
    $db->exec("INSERT INTO movie(title, price) VALUES('シスター', 2000)")

    // トランザクション内の一連のステートメントが成功したら、トランザクションをコミット。
    $db->commit();

} catch{
    // 例外が発生したらロールバックし、エラーメッセージを出力。
    $db->rollBack();
    print "失敗しました。：{$e->getMessage()}"
}
```

◇ 更新前ログのログファイルへの書き込み

◇ コミットによる更新後ログのログファイルへの書き込み

- **コミット**

トランザクション内の一連のステートメントをディスク上のログファイルに更新後ログを書き込む。

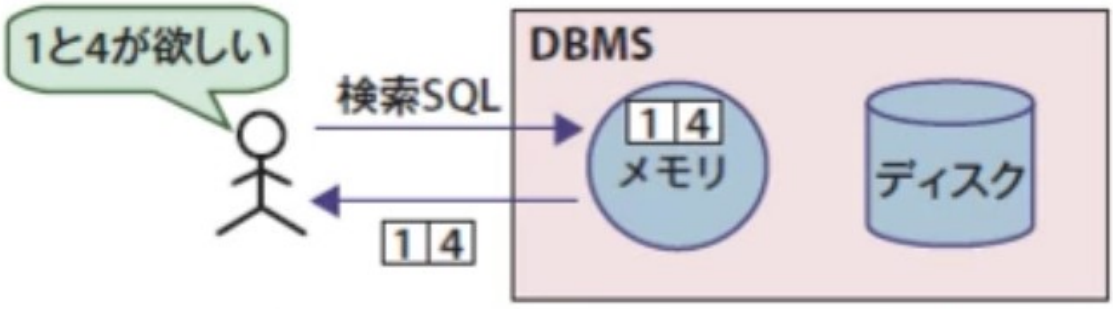
- **二相コミット**

コミットを以下の二つの段階に分けて行うこと。ACIDのうち、原子性と一貫性を実装している。

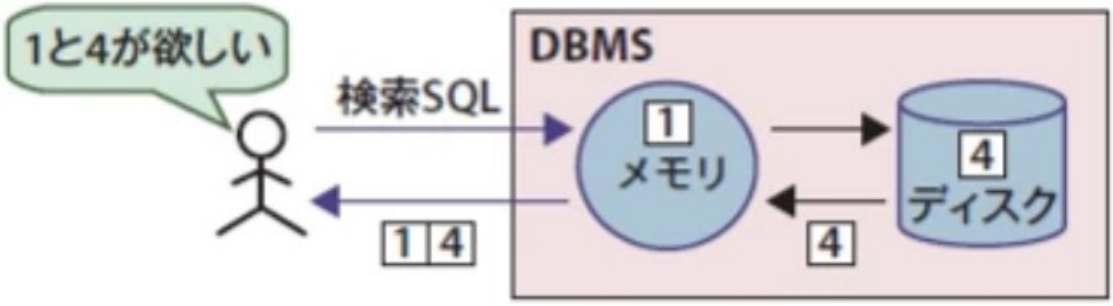
1. 他のサイトに更新可能かどうかを確認。
2. 全サイトからの合意が得られた場合に更新を確定。

◇ チェックポイントにおけるディスク上のデータファイルへの書き込み

トランザクションの終了後、DBMSは、処理速度を高めるために、ログファイルの更新後ログをいったんメモリ上で管理する。

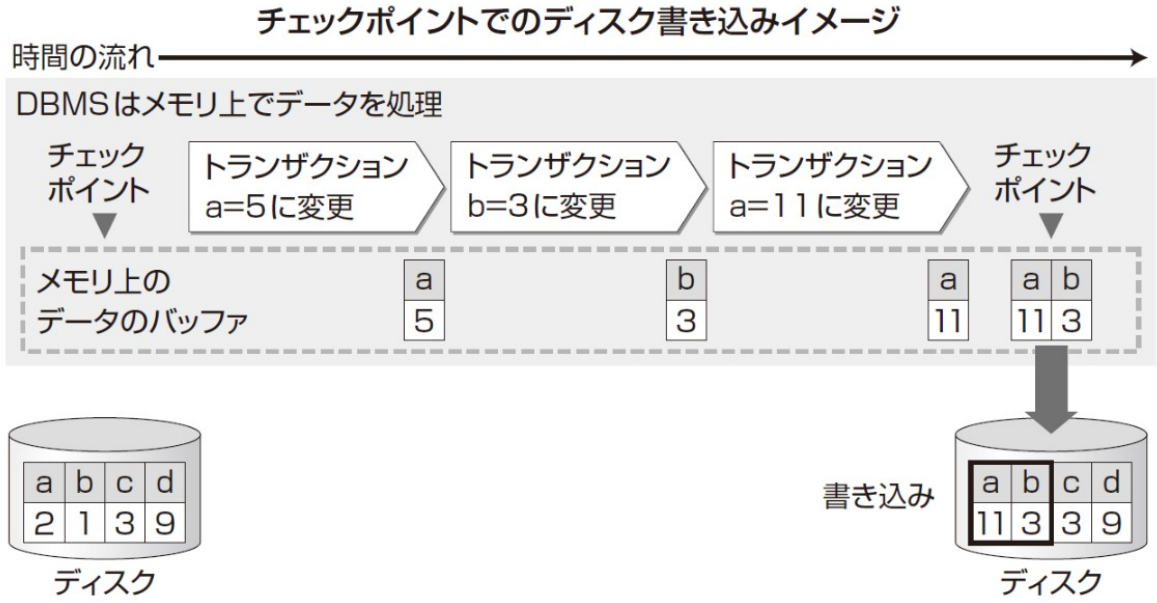


メモリに存在するデータだけで結果が返せると非常に速い



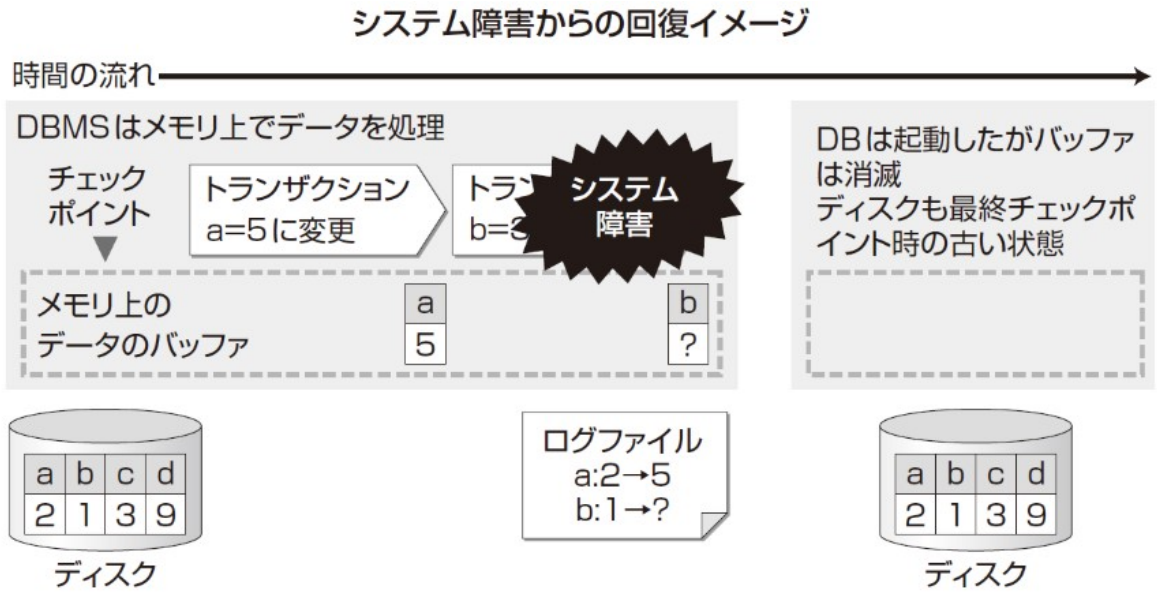
メモリにデータがなくて、ディスクまで検索しなければならないと遅い

そして、チェックポイントで、ログファイルの更新後ログをディスク上のデータファイルに反映させる。この時、チェックポイントは、自動実行または手動実行で作成する。



◇ システム障害からの回復

DBMSやOSのトラブル等によりシステム全体が停止する障害のこと。



ログファイルを利用して，可能な限り復旧
a：障害発生前にトランザクション完了（トランザクション後の値が５とわかっている）
　→ロールフォワードにより５に書き換え
b：トランザクション実行中に障害発生（何の値にすべきかログに残っていない）
　→ロールバックにより，障害発生前の値に戻す。

- **ロールバック**

障害によって、トランザクション内の一連のステートメントがすべて実行されなかった場合に、ログファイルの更新前ログを用いて、トランザクションの開始前の状態に戻す。

- **ロールフォワード**

障害によって、トランザクションの終了後に一連のステートメントの更新結果がディスクに反映されなかった場合に、ログファイルの更新後ログを用いて、ディスク上のデータファイルに更新結果を反映させる。

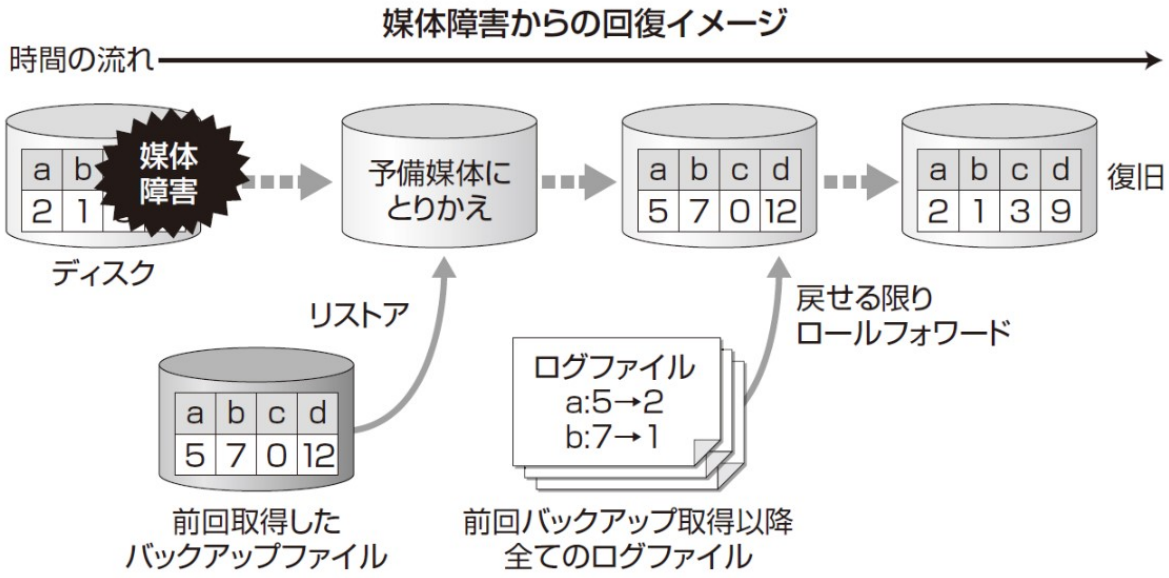
【具体例】

『a』の値を更新するステートメントを含むトランザクションの後に、システムが異常終了した場合、ログファイルの更新後ログ『a = 5』を用いて、ディスク上のデータファイルに更新結果を反映させる。（ロールフォワード）

『b』の値を更新するステートメントを含むトランザクションの途中に、システムが異常終了した場合、ログファイルの更新前ログ『b = 1』を用いて、障害発生前の状態に戻す。（ロールバック）

◇ 媒体障害からの回復

データベースの情報が格納された物理ディスクの障害のこと。ディスクを初期化／交換した後、バックアップファイルからデータベースを修復し、ログファイルの更新後ログ『a = 5』『b = 1』を用いて、修復できる限りロールフォワードを行う。



【具体例】

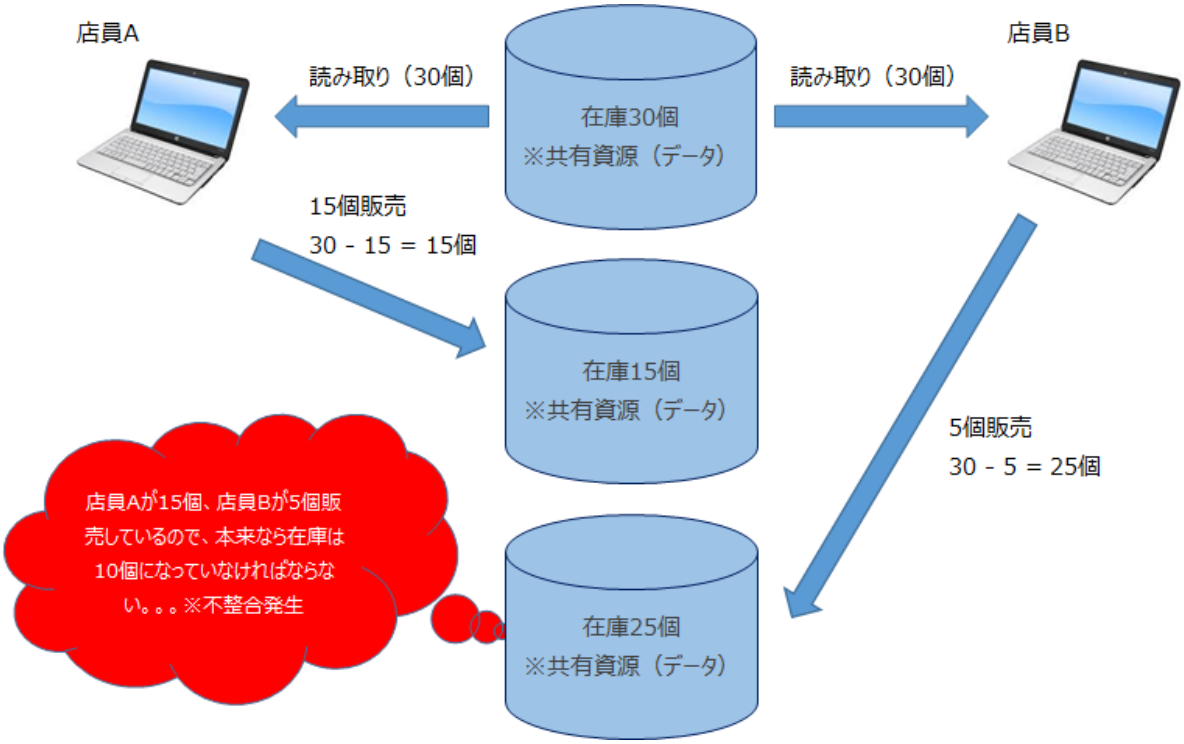
バックアップファイルの実際のコード

```
-- --------------------------------------------------------
-- Host:                         xxxxx
-- Server version:               10.1.38-MariaDB - mariadb.org binary
distribution
-- Server OS:                    Win64
-- HeidiSQL Version:             10.2.0.5611
-- --------------------------------------------------------

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET NAMES utf8 */;
/*!50503 SET NAMES utf8mb4 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0
*/;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

# データベース作成
-- Dumping database structure for kizukeba_pronami_php
CREATE DATABASE IF NOT EXISTS `kizukeba_pronami_php` /*!40100 DEFAULT CHARACTER
SET utf8 COLLATE utf8_unicode_ci */;
USE `kizukeba_pronami_php`;

# テーブルのデータ型を指定
-- Dumping structure for table kizukeba_pronami_php.mst_staff
CREATE TABLE IF NOT EXISTS `mst_staff` (
  `code` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(15) COLLATE utf8_unicode_ci NOT NULL,
  `password` varchar(32) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`code`)
) ENGINE=InnoDB AUTO_INCREMENT=22 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

# データを作成
-- Dumping data for table kizukeba_pronami_php.mst_staff: ~8 rows
(approximately)
/*!40000 ALTER TABLE `mst_staff` DISABLE KEYS */;
INSERT INTO `mst_staff` (`code`, `name`, `password`) VALUES
    (1, '秦基博', 'xxxxxxx'),
```

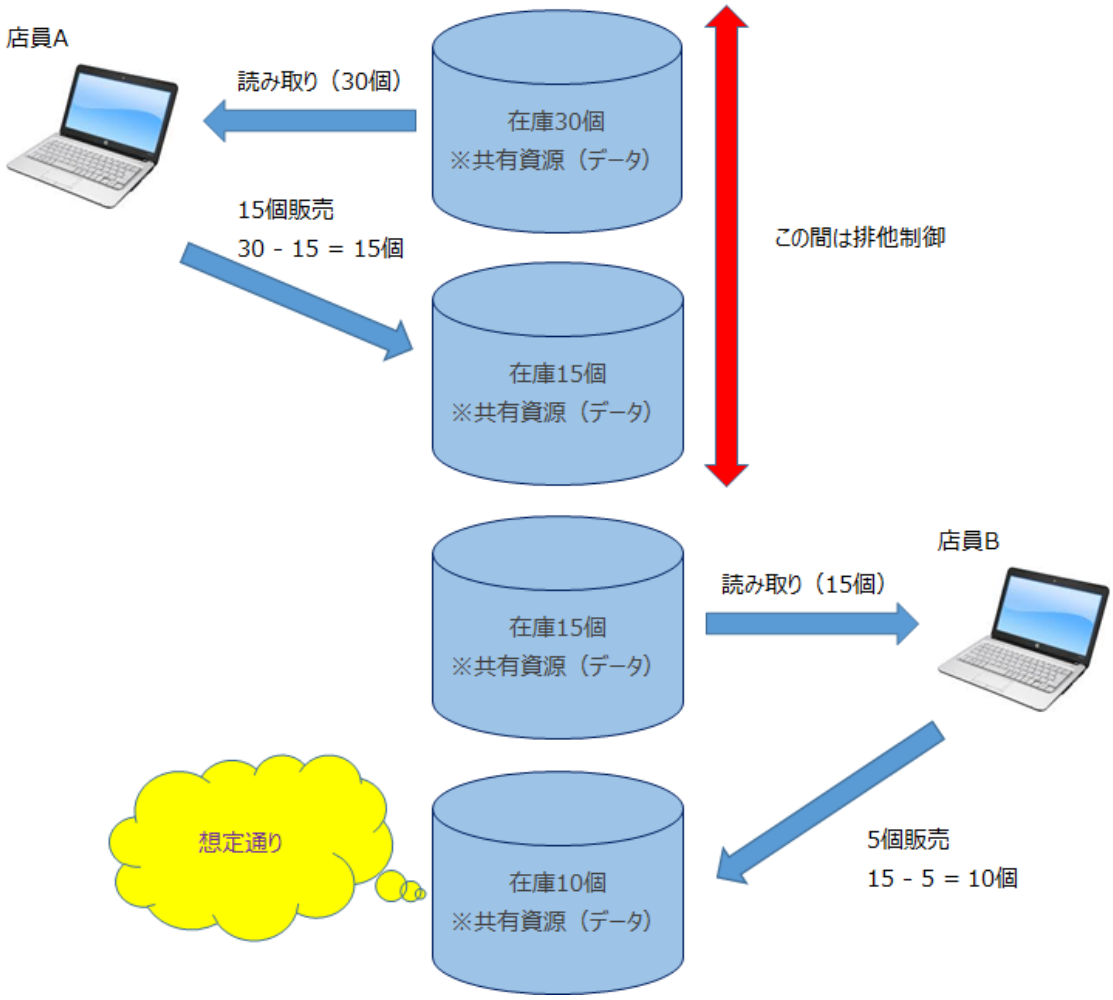
```
    (2, '藤原基央', 'xxxxxxx');
/*!40000 ALTER TABLE `mst_staff` ENABLE KEYS */;

/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
/*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1,
@OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

06. 排他制御

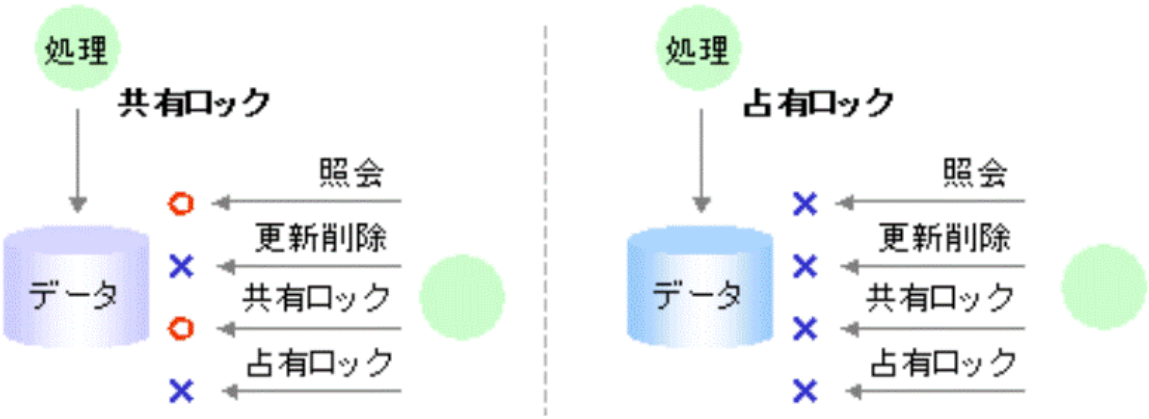
◇ なぜ排他制御が必要か



- **排他制御を行った結果**



◇ 排他制御におけるロックの種類



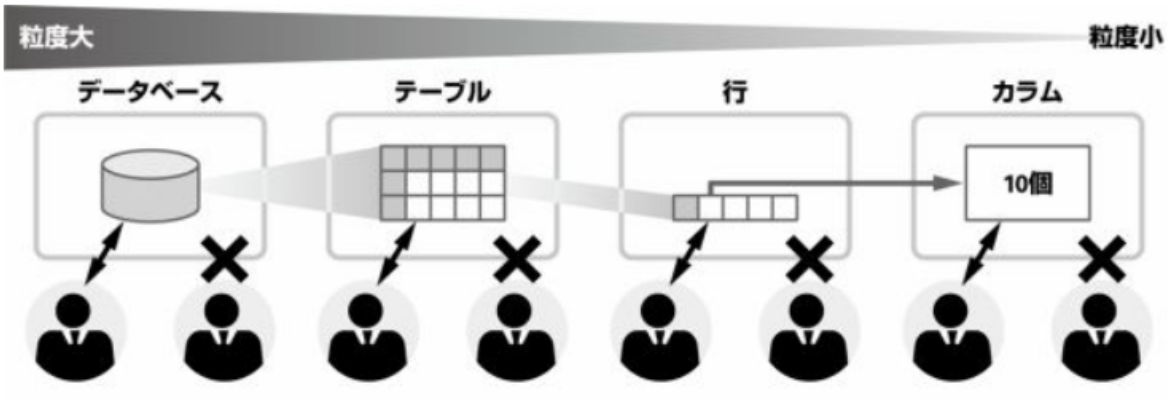
- **共有ロック**

CRUDのRead以外の操作を実行不可能にする。データのRead時に、他者によってUpdateされたくない場合に用いる。「共有」の名の通り、共有ロックされているデータに対して、他の人も共有ロックを行うことができる。

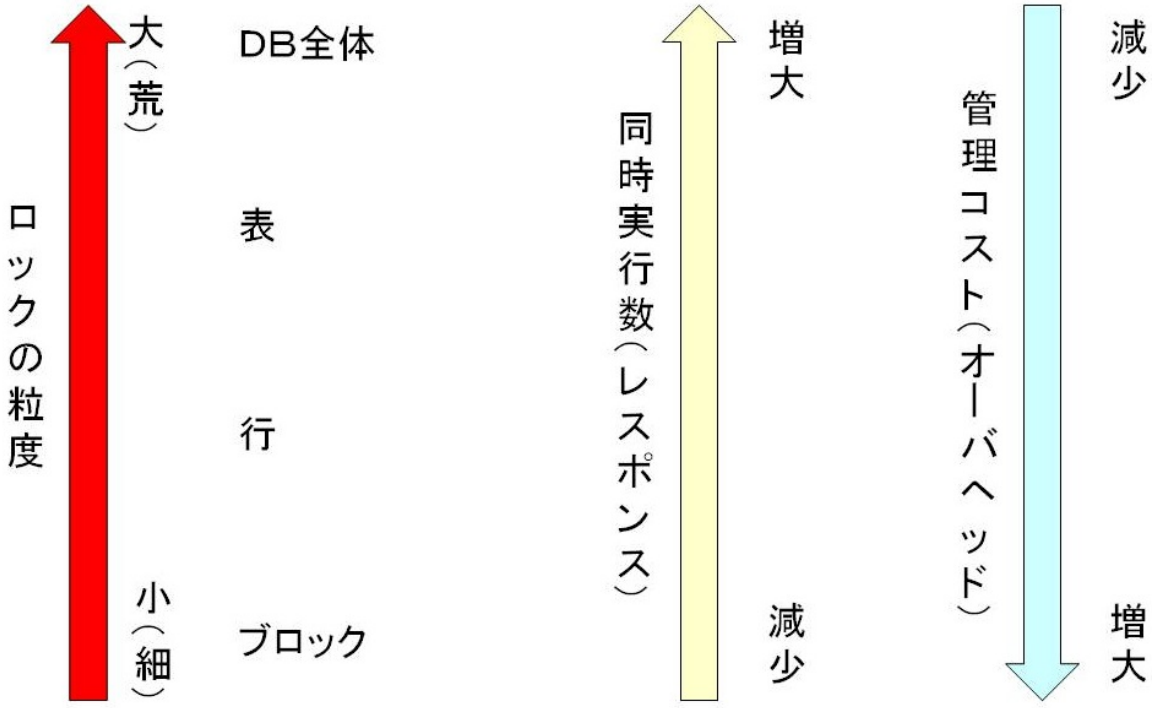
- **専有ロック**

CRUDの全ての操作を実行不可能にする。データのUpdate時に、他者によってUpdateもReadもされたくない場合に用いる。

◇ ロックの粒度



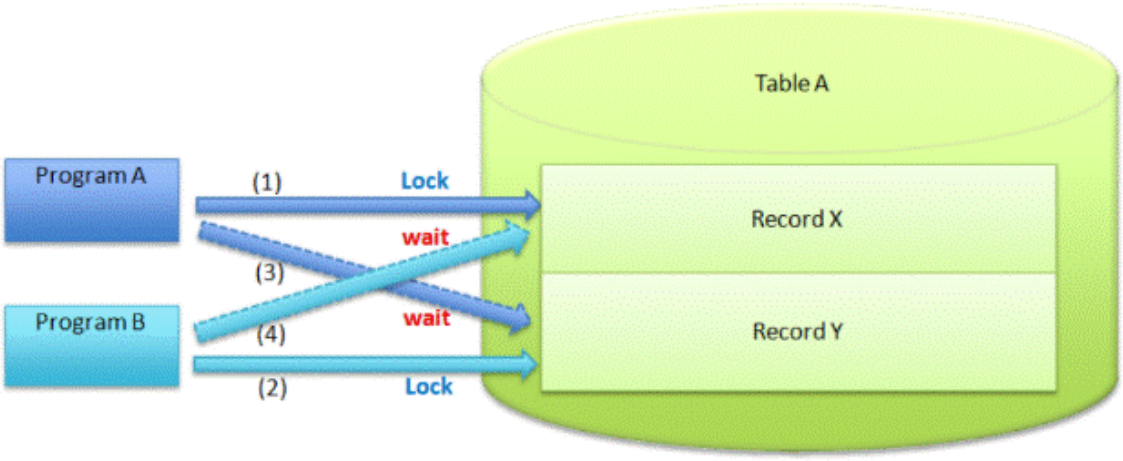
DB ＞ テーブル ＞ レコード ＞ カラム の順に、粒度は大きい。ロックの粒度が細かければ、トランザクションの同時実行性が高くなって効率は向上する（複数の人がDBに対して作業できる）。しかし、ロックの粒度を細かくすればするほど、それだけデータベース管理システムのCPU負荷は大きくなる。



◇ デッドロック現象

複数のトランザクションが、互いに他者が使いたいデータをロックしてしまい、お互いのロック解除を待ち続ける状態のこと。もう一方のレコードのロックが解除されないと、自身のレコードのロックを解除できない時、トランザクションが停止すること。

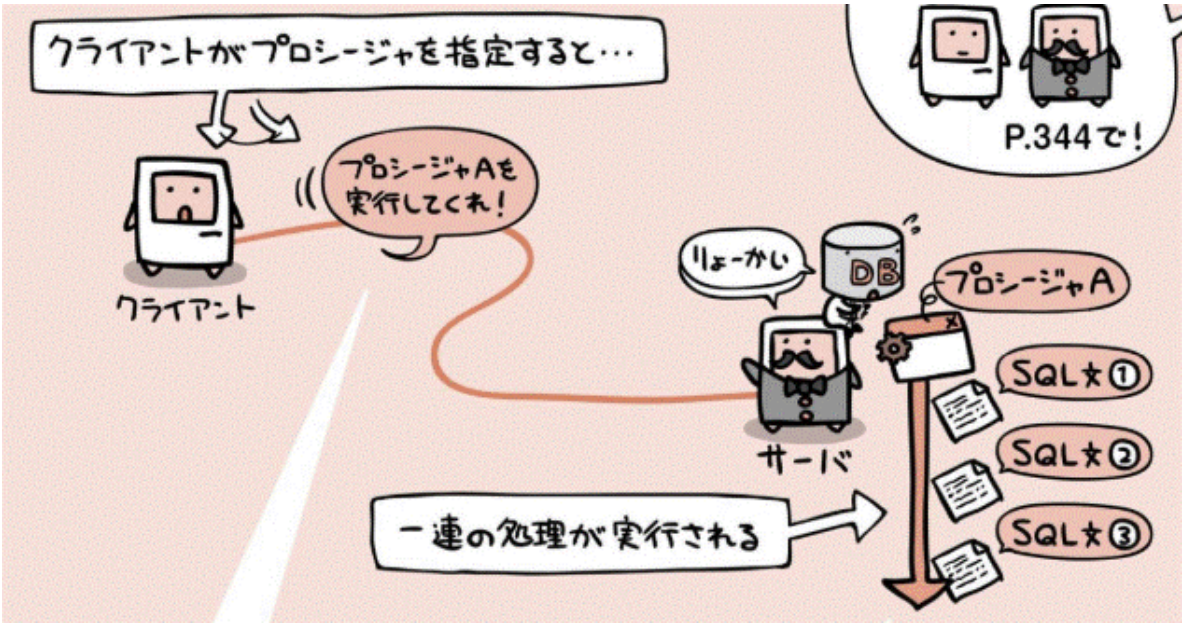
- **共有ロックしたデータを共有ロック**
- **共有ロックしたデータを専有ロック**
- **専有ロックしたデータを共有ロック**
- **専有ロックしたデータを専有ロック**



07. データベースの操作

◇ Stored procedure

あらかじめ一連のSQL文をデータベースに格納しておき、Call文で呼び出す方式。



【実装例】

```
# PROCEDUREを作成し、データベースへ格納しておく。
CREATE PROCEDURE SelectContact AS
  SELECT CustomerID, CompanyName, ContactName, Phone
  FROM Customers
```

```
# PROCEDUREを実行
EXEC SelectContact
```

◇ Migrationファイルによるテーブルの作成

マイグレーションファイルと呼ばれるスクリプトファイルを作成し、テーブルの新規作成やカラムの追加はこのスクリプトファイルに記述していく。

1. 誰かが以下のMigrationファイルをmasterにPush
2. Migrationファイルをローカル環境にPull
3. `db:reset`で、ローカル環境のDBスキーマとデータを更新

```
namespace Migration

class ItemQuery
{
    public static function insert()
    {
        return "INSERT INTO item_table VALUES(1, '商品A', 1000, '2019-07-24
07:07:07');"
    }
}
```

08. selectによるデータセットの取得

◇ 実装例で用いた略号

C：column（列）

R：record（行）

T：table

◇ SQLの処理の順番

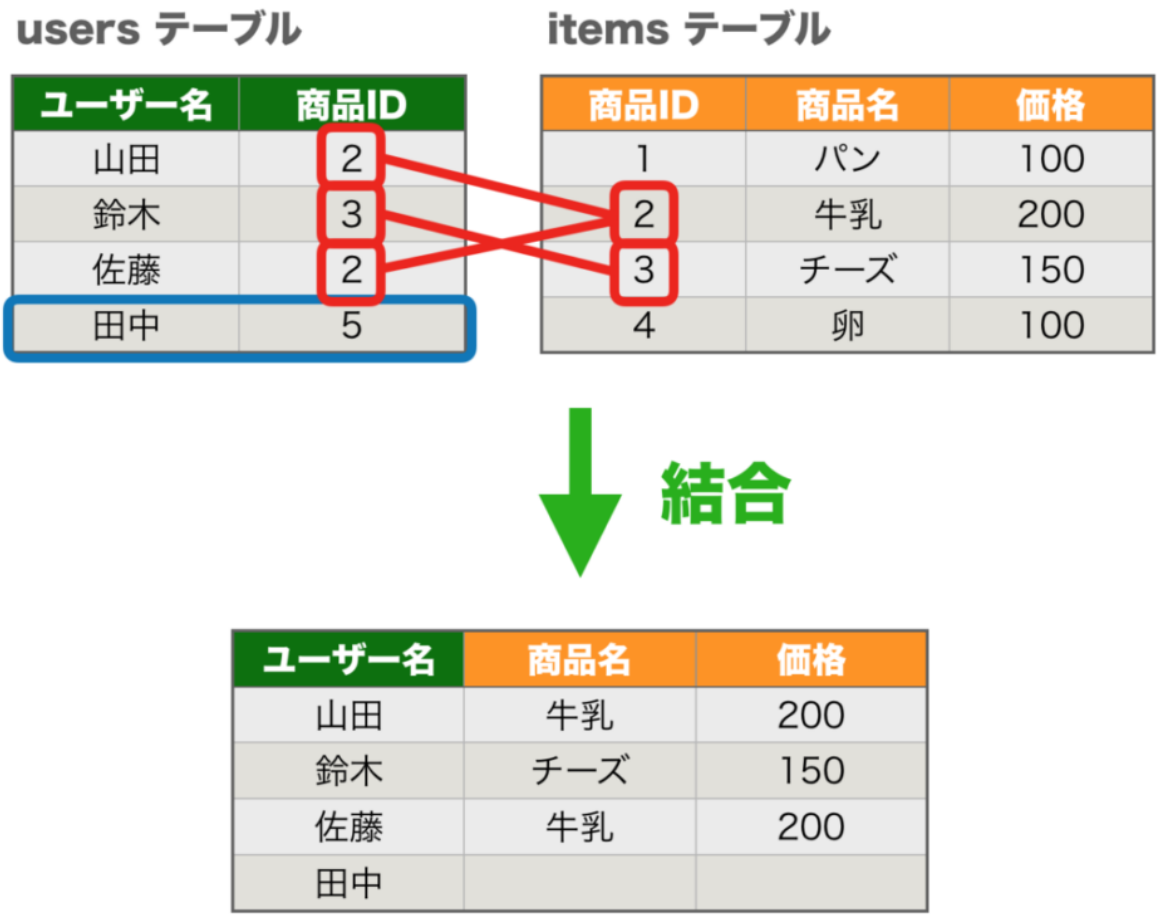
```
from ⇒ where ⇒ group by ⇒ having ⇒ select ⇒ order by
```

◇ 内部結合と外部結合

基本情報技術者試験では、内部結合（A∩B）しか出題されない。

- **`LEFT JOIN`で起こること**

『users』テーブルと『items』テーブルの商品IDが一致しているデータと、元となる『users』テーブルにしか存在しないデータが、セットで取得される。



ユーザー名	商品名	価格
山田	牛乳	200
鈴木	チーズ	150
佐藤	牛乳	200
田中		

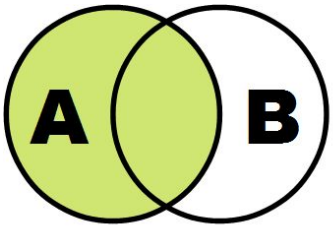
- **内部結合に`where`を用いる場合**

```
select C,  #『カラム』を指定
    from T1, T2, T3  #複数の表を指定
    where R1 = R2 and  #指定したカラムのうち、レコードと別の表のレコードを照合し、フィールドを取得
    R2 = R3  #3つ目の表のレコードとも照合
```

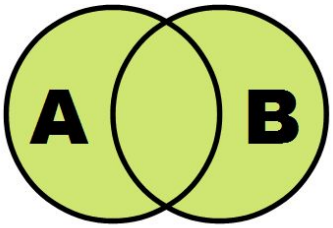
- **内部結合に`inner join on`を用いる場合（本試験では出題されない）**

```
select C,  #『カラム』を指定
    from T1  #複数の表を指定
    inner join T2
        on T1.C1 = T2.C2  #2つ目の表の『レコード』と照合
    inner join T3
        on T1.C1 = T3.C3  #3つ目の表の『レコード』と照合
```

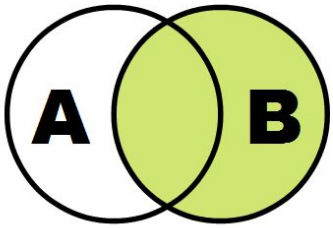
- **ベン図で内部結合と外部結合を理解しよう**



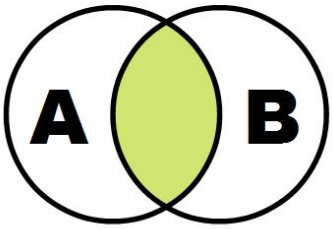

```
SELECT *
FROM A
LEFT JOIN B
ON A.id = B.id
```



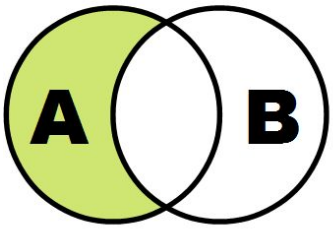

```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.id = B.id
```



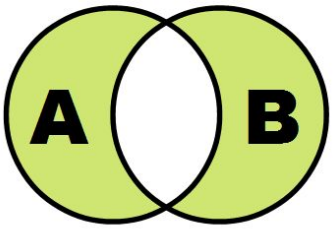

```
SELECT *
FROM A
RIGHT JOIN B
ON A.id = B.id
```



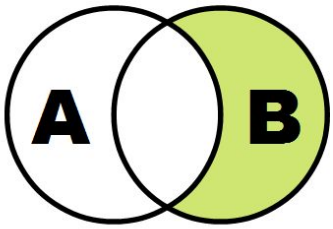

```
SELECT *
FROM A
INNER JOIN B
ON A.id = B.id
```


```
SELECT *
FROM A
LEFT JOIN B
ON A.id = B.id
WHERE B.id IS NULL
```




```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.id = B.id
WHERE A.id IS NULL
OR B.id IS NULL
```




```
SELECT *
FROM A
RIGHT JOIN B
ON A.id = B.id
WHERE A.id IS NULL
```

◇ 集合関数まとめ

- **sum()**

```
select sum(C)  #指定したカラムで、『フィールド』の合計を取得
    from T
```

- **avg()**

```
select avg(C)  #指定したカラムで、『フィールド』の平均値を取得
    from T
```

- **min()**

```
select min(C)  #指定したカラムで、『フィールド』の最小値を取得
    from T
```

- **max()**

```
select max(C)  #指定したカラムで、『フィールド』の最大値を取得
    from T
```

- **count()**

```
select count(C)  #指定したカラムで、『フィールド』の個数を取得
    from T
```

※消去法の小技：集合関数を入れ子状にはできない

```
select avg(sum(C))  #集合関数を集合関数の中に入れ子状にすることはできない。
    from T
```

```
select count(*)  #指定したカラムで、値無しも含む『フィールド』を取得
    from T
```

```
select count(C)  #指定したカラムで、値無しを除いた『フィールド』を取得

select count(all C)  #上に同じ
```

```
select count(distinct C)  #指定したカラムで、重複した『フィールド』を除く全ての『フィールド』を取得
```

◇ in

【IN句を使用しなかった場合】

```
SELECT * FROM fruit WHERE name = "みかん" OR name = "りんご";
```

【IN句を使用した場合】

```
SELECT * FROM fruit WHERE name IN("みかん","りんご");
```

◇ group by

※消去法の小技：`group by`のカラム引数は、`select`のカラムと同じでなければならない

```
select C
    from T
    group by C  #指定したカラムで、各グループのフィールドを集計
```

◇ having

`group by`で集計した結果から、`having`で『フィールド』を取得。

```
select C
    from T
    group by C
    having count(*) >=2  #集計値が２以上の『フィールド』を取得
```

※以下の場合、`group by + having`を使っても、`where`を使っても、同じ出力結果になる。

```
select C
    from T
    group by C
    having R
```

```
select C
    from T
    where R
    group by C
```

◇ sub-query

掛け算と同様に、括弧内から先に処理を行う。

```
select * from T  #Main-query
    where C != (select max(C) from T)  #Sub-query
```

◇ inとany

- **in**

  指定した値と同じ『フィールド』を取得

```
select * from T
    where C in (xxx, xxx, ...)  #指定したカラムで、指定した値の『フィールド』を取得
```

```
select * from T
    where C not in (R1, R2, ...)  #指定したカラムで、指定した値以外の『フィールド』を取得
```

```
select * from T
    where C not in ( #フィールドを指定の値として用いるｚ
        select C from T where R >= 160)  #指定したカラムで、『フィールド』を取得
```

- **any**

  書き方が異なるだけで、`in`と同じ出力

```
select * from T
    where C = any(xxx, xxx, xxx)
```

◇ view

ビューとはある表の特定のカラムや指定した条件に合致するレコードなどを取り出した仮想の表。また、複数の表を結合したビューを作成できる。ビューを作成することによりユーザに必要最小限のカラムやレコードのみにアクセスさせる事ができ、また結合条件を指定しなくても既に結合された表にアクセスできる。
⇒よくわからん...

```
create view T as
    select * from ...
```

◇ wildcard

```
select * from T
    where C like '%営業'  #任意の文字（文字無しも含まれる）
```

```
select * from T
    where C like '_営業'  #任意の一文字
```

◇ between

```
select * from T
    between 1 and 10  #指定したカラムで、1以上10以下の『フィールド』を取得
```

◇ Prepared statement

09. fetchによるデータ行の取り出し

DBで取得したデータをプログラムに一度に全て送信してしまうと、プログラム側のメモリを圧迫してしまう。そこで、fetchで少しずつ取得する。

【実装例】

`getConnection()`を起点として、返り値から繰り返しメソッドを取得し、`fetchAll()`で、テーブルのクエリ名をキーとした連想配列が返される。

```
getConnection()->executeQuery($query, $params, $types, $qcp)->fetchAll()

/// 結果
Array  = (
    [kbn_name] => レッド
    [kbn_value] => 2
    [quantity] => 50
)
```