

# データベース

## 第2回

土田 隼之

授業計画			
	週	授業内容・方法	週ごとの到達目標
後期	1週	データベースの概要	データベースの役割、データベースの学術利用、業務利用、その意義と用途を理解できる。
	2週	データベースのための基礎理論	集合とその演算、組（タプル）、組の集合としてのリレーションなど、データベースのための基礎理論を理解できる。
	3週	リレーショナルデータモデルとリレーショナル代数	RDBMSで利用されるデータモデルであるリレーショナルデータモデルとデータ操作のためのリレーショナル代数を理解できる。
	4週	SQL(1)	RDBMSの利用全般に用いられる言語SQLの基本を理解できる。リレーションへのデータ登録・削除・更新、簡単な問合せなど、基本的なSQLの使い方を理解できる。
	5週	SQL(2)	RDBMSの利用全般に用いられる言語SQLを作成できる。SQLにおける問合せを行うselect文を理解できる。
	6週	RDBMSの内部構成	RDBMSの内部構成、および大量のデータの中から目的とするデータに素早くアクセスする仕組みであるインデックスを理解できる。
	7週	問合せ最適化	RDBMSで、SQL問合せを実行するための実行プランを生成するための問合せ最適化が理解できる。
	8週	中間試験	中間試験
	9週	プログラムからのRDBMSの利用	汎用プログラミング言語で書かれたプログラムからRDBMSを利用する方法が理解できる。
	10週	正規化	リレーションの更新時に発生しうるデータの不整合、およびその解決策であるリレーションの正規化が理解できる。
	11週	データモデリング	実社会の中でデータベース化したい範囲を決定し、データ項目を抽出・整理して適切なデータ構造を決定する作業であるデータモデリングが理解できる。
	12週	SQL(3)	RDBMSの利用全般に用いられる言語SQLを作成できる。SQLにおける問合せを行う高度なselect文を理解できる。
	13週	トランザクションと同時実行制御	アプリケーションがデータベースにアクセスする単位であるトランザクションの概念、および複数のトランザクションを正常に実行するための基礎理論を理解できる。
	14週	NoSQLデータベースとビッグデータ(1)	ビッグデータを扱うため開発された新しいデータベースであるNoSQLの基礎を理解できる。主にNoSQLの概観と、ビッグデータを扱うためのデータモデルや実行制御理論を理解できる。

# タプルとリレーション

- ・データベースでは、「その要素の種類を表した属性」などと呼ばれる情報を付加して、扱われることが多い。
- ・同じ属性(料理名など)の要素の集合をドメインと呼ぶ。ドメイン  $D_1, D_2, \dots, D_n$  の直積集合を  $D_1 \times D_2 \times \dots \times D_n$  とすると、各ドメインから1つずつ選んだ(属性の)値の組  $(d_1, \dots, d_n)$  をタプル、その部分集合をリレーションと呼ぶ。

例:  $(d_1, d_2)$  だと(天津飯, 900)。  
d1はチャーハンなどが入る

料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
餃子	500

属性名: カラム名、列名  
#例: 「値段(円)」

タプル: 行、レコード  
#例: (チャーハン, 800)

リレーション: 表

# リレーショナルデータベースモデルと第1正規形

データベースにデータを蓄えるには、データやデータ同士の関係をモデル化して、蓄えるための枠組みをあらかじめ用意しなければならない。そのため、漠然と存在するデータ間の関係を把握する必要がある。リレーショナルデータベースモデルでは、下記のようにデータや関係を表現する。

- ・リレーション:表の各行は、共通の構造をしたタプル  
各行は、横に並んでいるそれぞれのデータが分解できない単純な値
- ・第一正規形:単純な値でできている表
- ・非第一正規形:表の中の項目が表やリストになっている

5, "明石" など

第一正規形

料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
餃子	500

非第一正規形

料理名		値段(円)
中華料理	チャーハン	800
	天津飯	900
イタリアン	パスタ	500
ミネラルウォーター		400

行が共通のタプル構造ではない

# リレーションと整合性制約

- ・現実世界のデータをデータベースに収める際に、データベース自身の整合性(データが矛盾しない)を保つためには、データやデータ間に存在している元々の条件や制約を、データベースにおいても再現するのが望ましい。

例えば、個人を識別している学生の学籍番号は各学校において重複することはない。電話番号も異なる契約に同じ番号が割り振られることは無い。生年月日の数字は、月ならば1から12までの整数である。

→このような制約を満たさないデータをデータベースの格納時にはじく事で、整合性がとれたデータのみを保存することができる。

データベースに課されるこのような制約のことを**整合性制約**と呼ぶ。

# リレーションと整合性制約(キー制約)

ある大学で使用されている下記のリレーションを考える。属性「学籍番号」では、重複した属性値(学籍番号の値)は存在しないはずである。

**空値:**一部の属性値が未決で、とりあえず空欄にしたい場合、空値(null value)とする。

**スーパーキー:**あるリレーションにおける任意のリレーションに対し「タプルを一意的に特定できる属性や属性の集合」を、スーパーキーという。

学籍番号が決まると、氏名や所属学部なども自動的にわかる

<u>学籍番号</u>	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde
120201	02	山田太郎	03-9012-fghi
120202	02	小鳥遊次郎	03-3456-jklm

上記でのスーパーキー:{学籍番号}、{学籍番号,所属学部}、{学籍番号,氏名}、{学籍番号,電話番号}、{学籍番号,所属学部,氏名}、{学籍番号,所属学部,電話番号}、{学籍番号,氏名,電話番号}、{学籍番号,所属学部,氏名,電話番号}の8つ。

#学籍番号以外の情報は無くても一意に定まるが、余分な情報があっても一意に定まる。

# リレーションと整合性制約(キー制約)2

一般に、リレーションでは空値にならない候補キーを1つ選択し、主キーと呼ぶ。

**スーパーキー:**あるリレーションにおける任意のリレーションに対し「タプルを一意的に特定できる属性や属性の集合」を、スーパーキーという。

**候補キー:**スーパーキーのうち、スーパーキーに他のスーパーキーが含まれていないものを、候補キー、あるいは単にキーと呼ぶ。

**空値:**一部の属性値が未決で、とりあえず空欄にしたい場合、空値(null value)とする。

主キーを構成する属性名の下にアンダーラインをひく

<u>学籍番号</u>	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde
120201	02	山田太郎	03-9012-fghi
120202	02	小鳥遊次郎	03-3456-jklm

上記でのスーパーキー:{学籍番号}、{学籍番号,所属学部}、{学籍番号,氏名}、{学籍番号,電話番号}、{学籍番号,所属学部,氏名}、{学籍番号,所属学部,電話番号}、{学籍番号,氏名,電話番号}、{学籍番号,所属学部,氏名,電話番号}の8つ。

上記での候補キー:{学籍番号}のみ 他は、余分な属性が含まれている。



# 表定義と主キー

テーブル内で一意の制約がかかる

テーブル定義の際に、primary keyと指定したカラムが主キーとなる。

```
create table テーブル名(  
  列名d1 データ型 [not null]  
  [, 列名 d2 データ型 [not null] ...]  
  [, primary key (列名p1 [,列名p2...])]  
);
```

[ ]は、オプションであり、  
指定しなくても文が実行可能

```
create table item(item_id char(3)not null,  
                  item_name varchar(20),  
                  price int,primary key (item_id));
```

item\_idカラムが主キー

item\_idが主キーなのに、重複  
した値を格納しようとする  
とエラーとなる

```
Runtime error(Exit status:1)  
1 create table item(item_id char(3) not null,  
2 item_name varchar(20),  
3 price int, primary key(item_id));  
4  
5 insert into item(item_id, item_name, price) values("A1", "pencil",200);  
6 select * from item;  
7 insert into item(item_id, item_name, price) values("A1", "pencil2",200);  
8  
9
```

実行 (Ctrl-Enter) MySQLを学ぶ | プログラミング力診断

出力 実行時エラー 入力 コメント 0

ERROR 1062 (23000) at line 8: Duplicate entry 'A1' for key 'item.PRIMARY'



# リレーションと整合性制約(参照整合性制約)

一方のリレーションの属性値が他方のリレーションの属性値を参照している事が有る。その場合、片方の属性値が対応する主キーの属性値のいずれかの値になっていなくてはならないという制約(参照整合性制約)がある。

外部キー:他リレーションの主キーにある属性値をとる属性(空値もとる)  
#外部キーは他リレーションの属性の集合からなる主キーの場合もある

「所属学部」の属性値は「学部コード」の属性値を参照している。  
→「学部コード」に無い値を「所属学部」の属性値にするのは不可  
存在しない学部には所属出来ない

学籍番号	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde
120201	02	山田太郎	03-9012-fghi
120202	02	小鳥遊次郎	03-3456-jklm

学部コード	学部名	学部長	住所
01	文学部	渡邊一郎	東京都千代田区丸の内A
02	理工学部	西園公平	東京都千代田区丸の内B
03	情報工学部	中田元	東京都千代田区丸の内C

# リレーションと整合性制約(参照整合性制約)

データベースでは、なんらかの事象(例:学生情報)を複数の表(例:学生表、学部表)に分割して管理する。

分割された表を結合する際に、関数の概念が用いられる。

#詳細は、また後で

「所属学部」の属性値は「学部コード」の属性値を参照している。  
→「学部コード」に無い値を「所属学部」の属性値にするのは不可  
存在しない学部には所属出来ない

学籍番号	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde
120201	02	山田太郎	03-9012-fghi
120202	02	小鳥遊次郎	03-3456-jklm

学部コード	学部名	学部長	住所
01	文学部	渡邊一郎	東京都千代田区丸の内A
02	理工学部	西園公平	東京都千代田区丸の内B
03	情報工学部	中田元	東京都千代田区丸の内C

# 表定義と外部キー

テーブル定義の際に、foreign keyと指定したカラムが外部キーとなる。

```
create table テーブル名(  
  列名d1 データ型 [not null]  
  [, 列名 d2 データ型 [not null] ...]  
  [, primary key (列名p1 [,列名p2...])]  
  [, foreign key (列名f1) references 参照テーブルf1 (参照列名f1)  
  [, foreign key (列名f2) references 参照テーブルf2 (参照列名f2) ...]  
);
```

[ ]は、オプションであり、  
指定しなくても文が実行可能

# 表定義と外部キー

## Schema SQL

```
1 create table dept(dept_id char(3) not null,  
2                 dept_name char(10),primary key(dept_id));  
3  
4 create table student(student_id char(3) not null,  
5                      dept_id char(3) not null,  
6                      student_name char(10),  
7                      foreign key(dept_id) references dept(dept_id));  
8  
9 -- insert into dept(dept_id,dept_name) values("D1","abc");  
10 insert into dept(dept_id,dept_name) values("D1","abc");  
11 insert into student(student_id,dept_id,student_name) values("S1","D1","name");
```

Text to DDL



## Query SQL

```
1 select * from dept;  
2 select * from student;
```

## Results

Copy as Markdown

Query #1 Execution time: 0ms

dept_id	dept_name
D1	abc

Query #2 Execution time: 0ms

student_id	dept_id	student_name
S1	D1	name

# 表定義と外部キー

## Schema SQL ●

```
1 create table dept(dept_id char(3) not null,  
2                 dept_name char(10),primary key(dept_id));  
3  
4 create table student(student_id char(3) not null,  
5                     dept_id char(3) not null,  
6                     student_name char(10),  
7                     foreign key(dept_id) references dept(dept_id));  
8  
9 -- insert into dept(dept_id,dept_name) values("D1","abc");  
10 --insert into dept(dept_id,dept_name) values("D1","abc");  
11 insert into student(student_id,dept_id,student_name) values("S1","D1","name");
```

Text to DDL

## Query SQL ●

```
1 select * from dept;  
2 select * from student;
```

## Results

**Schema Error:** Error: ER\_NO\_REFERENCED\_ROW\_2: Cannot add or update a child row: a foreign key constraint fails (`test`.`student`, CONSTRAINT `student\_ibfk\_1` FOREIGN KEY (`dept\_id`) REFERENCES `dept` (`dept\_id`))

# 基本SQL:データ挿入(格納)

・ create table文により作成されたテーブルにはデータを挿入されるまでは1行も中身は入っていない。データを挿入するにはinsert文を使う。insert文の構文は下記である。

[ ]は、オプションであり、  
指定しなくても文が実行可能

insert into テーブル名 [(列名1, ..., 列名m)] values (値1, ..., 値m)

item\_id, item\_name, priceにそれぞれ対応

```
insert into item values('A01', 'オフィス用紙A4', 2000);  
insert into item(item_id, item_name) values('A03', "オフィス用紙A4");
```

item_id	item_name	price
A01	オフィス用紙A4	2000
A03	オフィス用紙A4	NULL

priceはnullとしてデータ格納



# 基本SQL:データ参照

第一回の講義

・データベースに登録されているデータはselect文により参照することができる。このselect文では、いろいろな条件を与えて条件に一致するデータを検索することができるため、select文によるデータ参照のことを問合せ(クエリ,query)と呼ぶ。

#データ更新など、SQLによるデータ操作のための要求全般を問合せと呼ぶことも有る。

[ ]は、オプションであり、  
指定しなくても文が実行可能

select 列名1,...,列名m from テーブル名 [where 条件]  
select \* from テーブル名 [where 条件]

\*は、列名全てと等価

```
insert into item values('A01','オフィス用紙A4',2000);
insert into item(item_id, item_name) values('A03', "オフィス用紙A4");

select * from item;
select * from item where item_id = 'A01';
select * from item where price >1000;
```

item\_id='A01'

item_id	item_name	price
A01	オフィス用紙A4	2000
A03	オフィス用紙A4	NULL

item_id	item_name	price
A01	オフィス用紙A4	2000

item_id	item_name	price
A01	オフィス用紙A4	2000

price  
>1000

## 課題2-a締切:10/26

以下について、エラーとなっているSQLと実行結果のスクリーンショットを提出せよ。

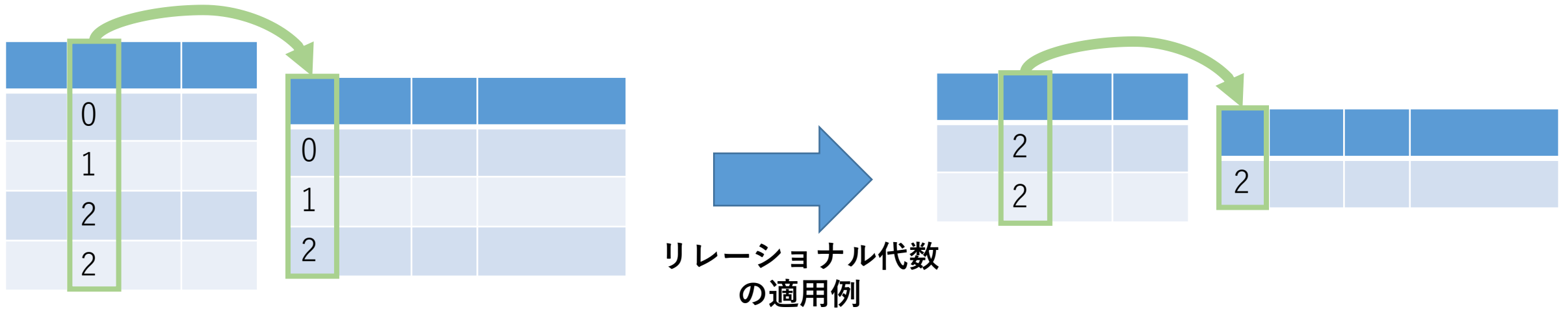
- 1)主キーを含むテーブルを作成し、主キーのカラムに重複したデータを格納しようとする、エラーとなることを確認せよ。
- 2)外部キーを含むテーブルを作成し、外部キーが原因でデータ格納エラーとなることを確認せよ。

<https://paiza.io/ja/projects/new>

<https://www.db-fiddle.com/> #外部キーはこちらのMySQLで実行

# リレーショナル代数

- ・リレーショナルデータベースでは、データモデルはリレーションの集合として表現される。複数のリレーションを外部キーと主キーによって関連付けることで、データベース内のデータ間の関係を管理する。
- ・ユーザがデータベースから目的のデータを取得するためには、格納されているリレーションから、目的とするデータを含む新たなリレーションを生成して、取得する。→この操作のための演算体系をリレーショナル代数という。



# 集合演算

- ・リレーショナル代数の演算は2種類(**集合演算**・関係演算)に分類できる。
- ・集合論で定義された集合演算には、和集合演算、差集合演算、共通集合演算、直積演算がある。

**a)和集合、共通集合、差集合は「属性が一致しているリレーション間」(和両立が成り立つリレーション間)のみに適用できる。**

b)直積演算は、リレーション間の属性が一致していなくても適用できる。

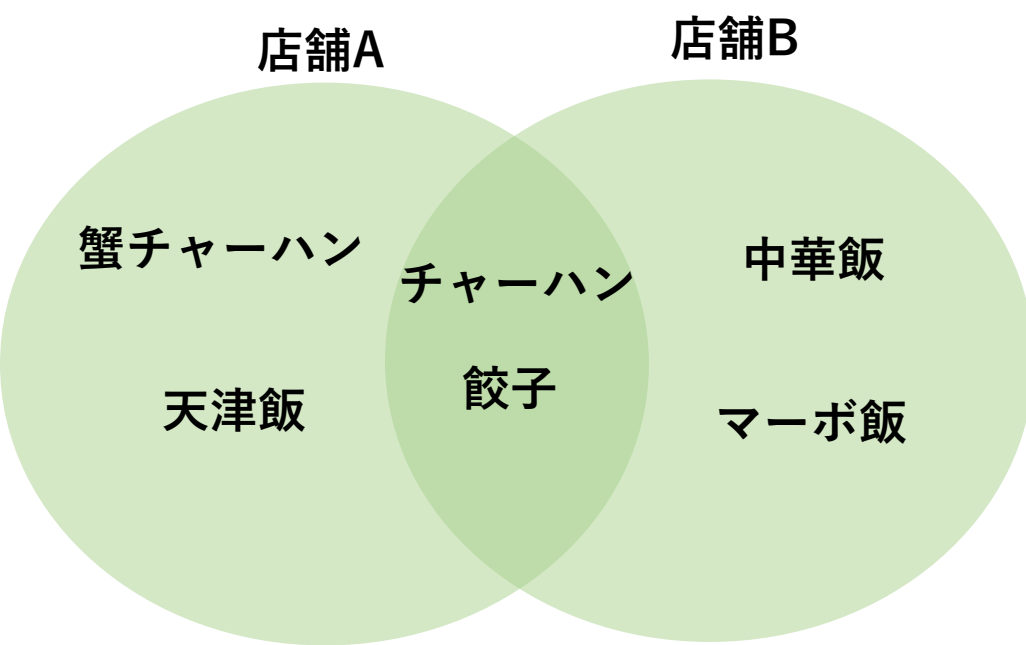
**和両立:**2つのリレーション $R, S$ が有る場合に、 $R$ と $S$ の属性が一致することを表す概念。以下のように定義される。

リレーション $R(A_1, A_2, \dots, A_n)$ と $S(B_1, B_2, \dots, B_n)$ が次の2条件を満たすとき、リレーション $R$ と $S$ は和両立である。

- 1) $R$ と $S$ の次数(属性の数)が等しい(つまり、 $n=m$ )
- 2)各 $i(1 \leq i \leq n)$ に対し、 $A_i$ と $B_i$ のドメイン(属性の取り得る範囲)が等しい

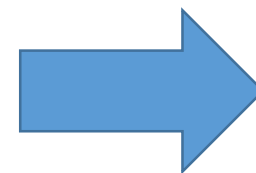
# 集合演算(和集合演算)

・ 和両立である2つのリレーションRとSに対して、RかSのどちらかに含まれるタプルからなる新たなリレーションを生成する演算を和集合演算と呼ぶ。その演算結果として得られるリレーションを和あるいは和集合と呼びRUSで表す。



店舗AのメニューRa	
料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
餃子	500

店舗BのメニューRb	
料理名	値段(円)
チャーハン	800
中華飯	750
マーボ飯	900
餃子	500



どちらかの店舗で扱っているメニュー	
料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
中華飯	750
マーボ飯	900
餃子	500

# 集合演算(和集合演算)

- MySQLでは、unionを使うことで和集合演算が行える。

```
1 create table item(item_id char(3) not null,  
2                   item_name varchar(20),  
3                   price int, primary key(item_id));  
4 create table item2(item_id char(3) not null,  
5                    item_name varchar(20),  
6                    price int, primary key(item_id));  
7 insert into item(item_id, item_name, price) values("A1", "pencil",200);  
8 insert into item(item_id, item_name, price) values("A2", "pencil2",200);  
9 insert into item2(item_id, item_name, price) values("A2", "pencil2",200);  
10  
11 select * from item union select * from item2;  
12 select * from item union all select * from item2;  
13
```

実行 (Ctrl-Enter)

MySQLを学ぶ | プログラミング力診断

出力 入力 コメント 0

item_id	item_name	price
A1	pencil	200
A2	pencil2	200

item_id	item_name	price
A1	pencil	200
A2	pencil2	200
A2	pencil2	200



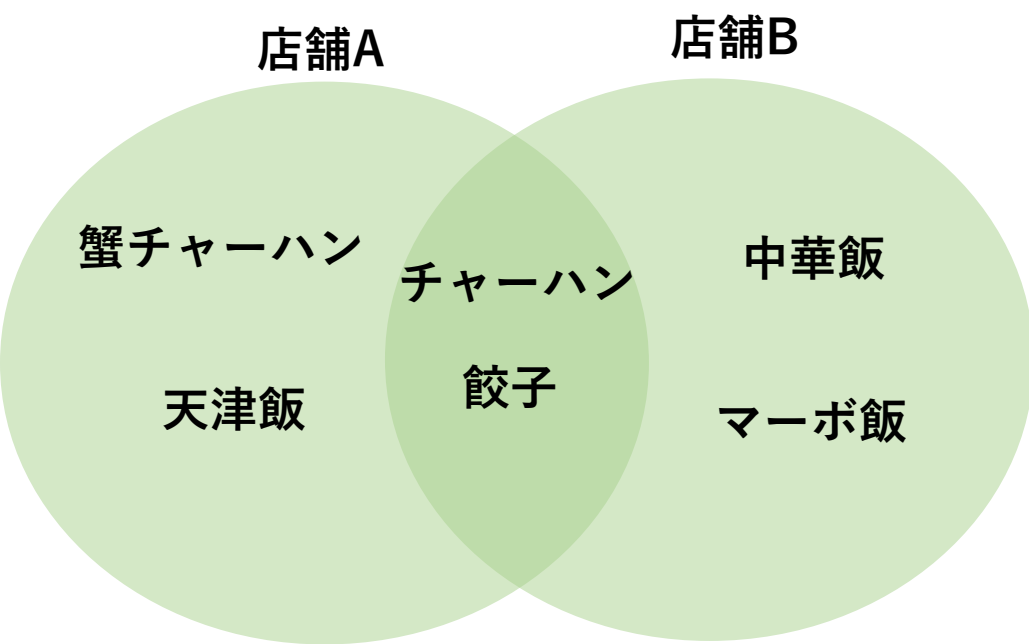
## 課題2-b 締切:10/26

MySQLでunionを用いたSQLを実行し、和集合演算となることを確認せよ。  
Unionとunion allで挙動が違う事も確認せよ。  
SQLと実行結果がわかるスクリーンショットを提出せよ。

# 集合演算(共通集合演算)

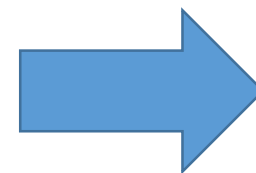
・ 和両立である2つのリレーションRとSに対して、RとSの両方に共通するタプルからなるリレーションを生成する演算を共通集合演算と呼ぶ。また、その演算結果として得られるリレーションを共通あるいは積集合と呼び、 $R \cap S$ で表す。

MySQLは共通集合演算の対応無し



店舗AのメニューRa	
料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
餃子	500

店舗BのメニューRb	
料理名	値段(円)
チャーハン	800
中華飯	750
マーボ飯	900
餃子	500

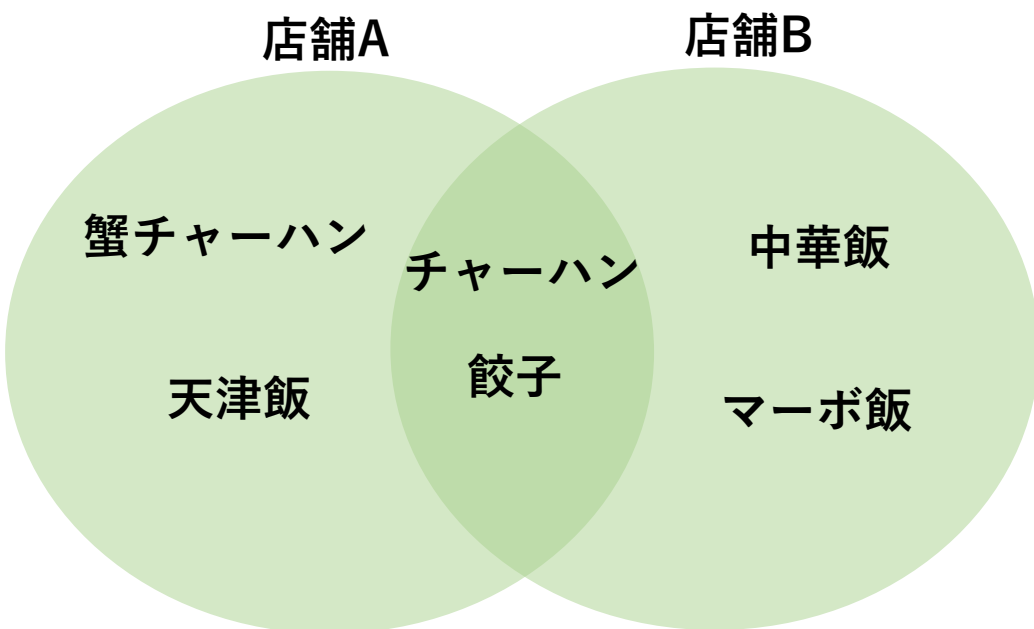


両方の店舗で扱っているメニュー	
料理名	値段(円)
チャーハン	800
餃子	500

# 集合演算(差集合演算)

・ 和両立である2つのリレーションRとSに対し、片方のリレーションのみに存在するタプルからなるリレーションを求める演算を差集合演算と呼ぶ。またその演算結果として得られるリレーションを差あるいは差集合と呼び、 $R-S$ (Rだけに存在するタプルのリレーション)および $S-R$ (Sだけに存在するタプルのリレーション)と表す。

MySQLは差集合演算の対応無し



店舗AのメニューRa	
料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
餃子	500

店舗BのメニューRb	
料理名	値段(円)
チャーハン	800
中華飯	750
マーボ飯	900
餃子	500

Ra-Rb  
店舗Raのみで  
扱っているメニュー

料理名	値段(円)
蟹チャーハン	1300
天津飯	900

Rb-Ra  
店舗Rbのみで  
扱っているメニュー

料理名	値段(円)
中華飯	1300
マーボ飯	900

# 集合演算(直積演算)

- ・ 2つの異なるリレーションRとSに対して、すべてのタプルの組合せを求める演算を直積(あるいはデカルト積)と呼び、その演算結果を $R \times S$ で表す。

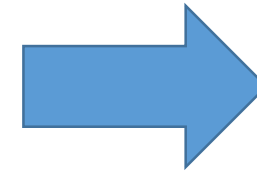
和両立でなくとも、  
直積演算は適用可能

販売店Ra

販売店	責任者
東京	太田
大阪	大津
福岡	小川

仕入れ商品Rb

仕入製品	製造工場
TV01	花巻
TV02	熊本
PV02	静岡

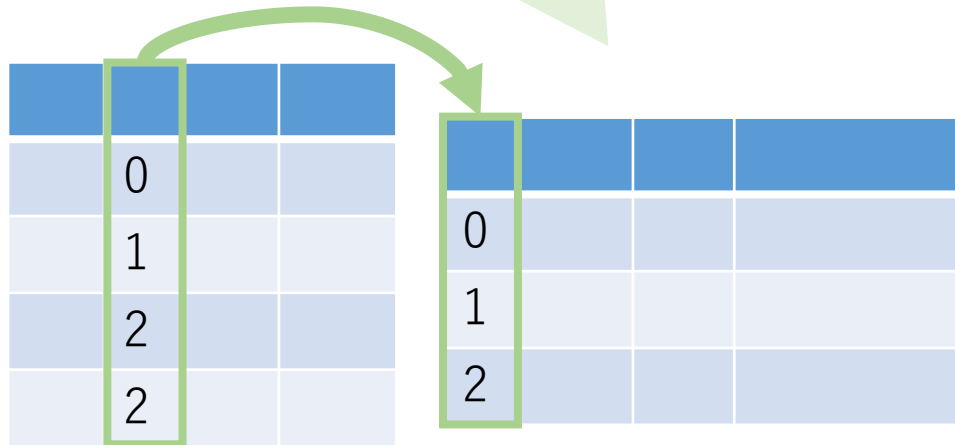


仕入製品	製造工場	販売店	責任者
TV01	花巻	東京	太田
TV01	花巻	大阪	大津
TV01	花巻	福岡	小川
TV02	熊本	東京	太田
TV02	熊本	大阪	大津
TV02	熊本	福岡	小川
PV02	静岡	東京	太田
PV02	静岡	大阪	大津
PV02	静岡	福岡	小川

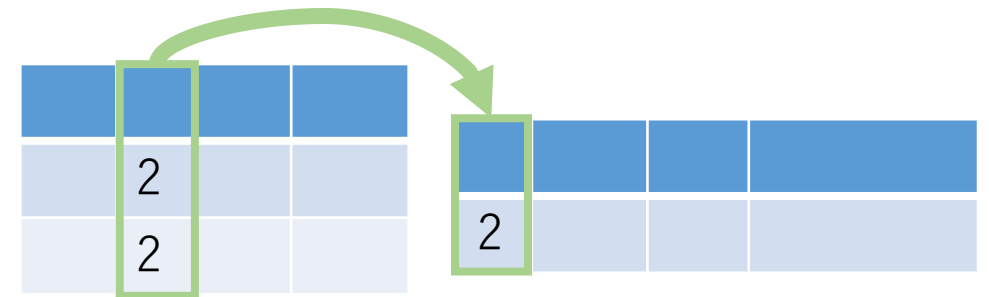
# 関係演算

- ・リレーショナル代数の演算は2種類(集合演算・**関係演算**)に分類できる。
- ・リレーショナルデータモデルに固有の主な関係演算としては、選択演算、射影演算、結合演算が有る。

選択演算の例



リレーショナル代数  
の適用例

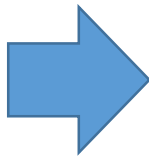


# 関係演算(選択演算)

- ・リレーションRの中から、条件式Fを満たすタプルを抽出する演算を選択(selection)と呼び、 $\sigma_F(R)$ で表す。条件式Fとしては、Rの属性 $A_i, A_j$ や定数値cに対して、比較演算子 $\theta (=, \neq, <, >, \geq, \leq)$ を用いて、 $A_i \theta A_j$ (例: $A_i = A_j$ )や $A_i \theta c$ (例: $A_i = 5$ )といった比較演算式を指定する。
- ・比較演算式で比較される2つの値は比較可能(ドメインが等しく、比較結果の真偽が常に定まる)でなければならない。
- $\theta$  比較可能:2つの値が比較可能であること
- ・条件式Fとして、比較演算を論理積( $\wedge$ )や論理和( $\vee$ )、否定( $\neg$ )で組合わせたものを指定できる。

学生一覧 Ra

学籍番号	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde
120201	02	山田太郎	03-9012-fghi
120202	02	小鳥遊次郎	03-3456-jklm



学籍番号	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde

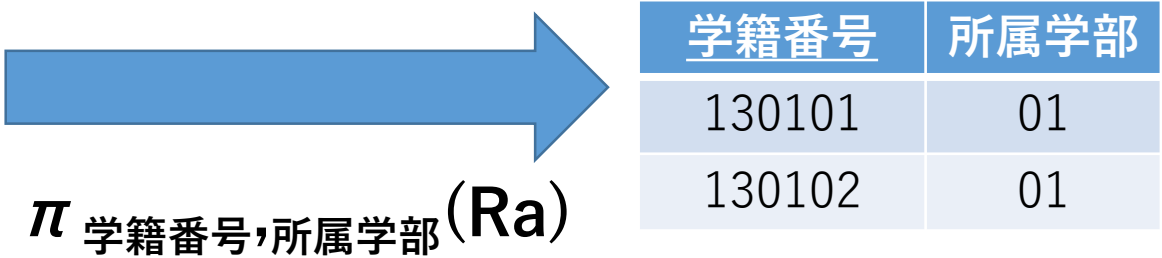
$\sigma_{\text{所属学部}=01}(Ra)$



# 関係演算(射影演算)

・リレーションRから、指定した属性集合  $\beta$  のみを持つリレーションを抽出する演算を射影(projection)と呼び、  $\pi_{\beta}(R)$  で表す。  $\beta$  はRの属性の部分集合である。射影は  $\beta$  で指定された属性集合以外を取り除き、テーブルを縦方向に切り出す演算である。

学生一覧 Ra			
学籍番号	所属学部	氏名	電話番号
130101	01	高橋一郎	03-1234-xyza
130102	01	鈴木花子	03-5678-bcde
120201	02	山田太郎	03-9012-fghi
120202	02	小鳥遊次郎	03-3456-jklm



# 関係演算(結合演算)

- ・ 結合(join)とは、2つのリレーションRとSを条件式Fに従って1つのリレーションに統合する演算であり、その演算結果を $R \bowtie_F S$ で表す。条件式Fは選択演算の場合と同様の条件式を指定する。その結果得られるリレーションは、RとSの直積 $R \times S$ に対して条件式Fを満足するタプルのみを残す選択演算を行った結果と等しい。
- ・ どちらか一方のリレーションのみに存在するタプルを演算結果に含めないことから、結合のことを内部結合(inner join)とも呼ぶ。

学籍番号	学部番号	氏名
130101	01	高橋一郎
120201	02	山田太郎

外部キーの属性間で結合する  
場合が多いですが、ドメイン  
があていれば外部キーでな  
くても結合可能

学部番号	学部名
01	文学部
02	理工学部
03	情報工学部

学籍番号	学部番号	氏名	学部番号	学部名
130101	01	高橋一郎	01	文学部
120201	02	山田太郎	02	理工学部

「学部コード」が03のタプル  
は、両方のリレーションに無  
いから結果に含まれない

# 関係演算(自然結合演算)

・等結合(「2つのリレーションの対応する属性の値が等しい結合条件」を用いる結合)で得られるリレーションでは、結合条件式に用いた属性が重複する。等結合の結果から重複する属性を射影で取り除いた結果を求める結合演算を、自然結合と呼ぶ。

学籍番号	学部番号	氏名
130101	01	高橋一郎
120201	02	山田太郎

学部番号	学部名
01	文学部
02	理工学部
03	情報工学部

結合演算

学籍番号	学部番号	氏名	学部番号	学部名
130101	01	高橋一郎	01	文学部
120201	02	山田太郎	02	理工学部

自然結合演算

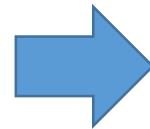
学籍番号	学部番号	氏名	学部名
130101	01	高橋一郎	文学部
120201	02	山田太郎	理工学部

# 関係演算(外部結合演算)

- ・ 結合演算(内部結合)の一種である等結合では、一方のリレーションRのタプルのうちで、もう一方のリレーションSに対応するタプルがないものは、演算結果であるリレーションに含まれない。これに対し、そのような場合でもRのタプルを演算結果に含め、対応するSの属性値を"null"とする等結合を外部結合(outer join)と呼ぶ。"null"は、その値が空値であることを表す。
- ・ 外部結合は、2つのリレーションを残すかにより、左外部結合、右外部結合に分類できる。両方のリレーションのタプルを全て残す完全外部結合もある。

学籍番号	学部番号	氏名
130101	01	高橋一郎
120201	02	山田太郎

学部番号	学部名
01	文学部
02	理工学部
03	情報工学部



右外部結合演算

学籍番号	学部番号	氏名	学部番号	学部名
130101	01	高橋一郎	01	文学部
120201	02	山田太郎	02	理工学部
null	null	null	03	情報工学部

## 課題2-1 締切:10/26

2-1)下記における、スーパーキー、候補キーをすべてあげよ

さらに、下記において、属性値間の外部キーの関係にあるものを1つあげよ

商品(商品ID,商品名,単価)

顧客(顧客ID,顧客名)

売上集計(顧客ID,商品ID,売上合計)

主キーを構成する属性名の下に  
アンダーラインをひく

2-2)下記に示すリレーションRaとRbに対して、和集合と共通集合を求めよ。

A店舗取り扱い商品リストRa

商品番号	商品名	値段
R001	プリンタ1	49,800
C001	パソコン1	148,000
H002	HDD2	9,800
M001	Camera1	38,800
M005	Camera5	49,800

B店舗取り扱い商品リストRb

商品番号	商品名	値段
R001	プリンタ1	49,800
C003	パソコン3	188,000
H001	HDD1	39,800
M002	Camera2	58,800
M005	Camera5	49,800

# (以降、時間有れば)基本SQL:データ挿入(格納)

・ create table文により作成されたテーブルにはデータを挿入されるまでは1行も中身は入っていない。データを挿入するにはinsert文を使う。insert文の構文は下記である。

[ ]は、オプションであり、  
指定しなくても文が実行可能

insert into テーブル名 [(列名1, ..., 列名m)] values (値1, ..., 値m)

item\_id, item\_name, priceにそれぞれ対応

```
insert into item values('A01', 'オフィス用紙A4', 2000);
insert into item(item_id, item_name) values('A03', "オフィス用紙A4");
```

priceはnullとしてデータ格納

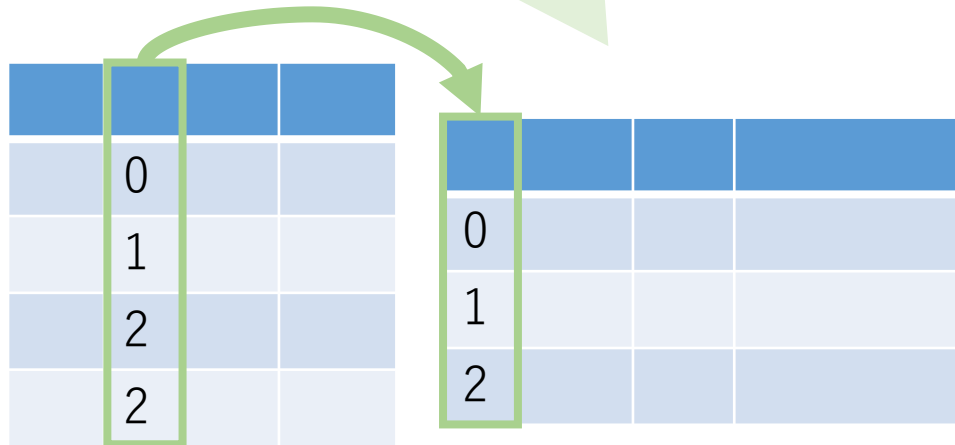
item_id	item_name	price
A01	オフィス用紙A4	2000
A03	オフィス用紙A4	NULL



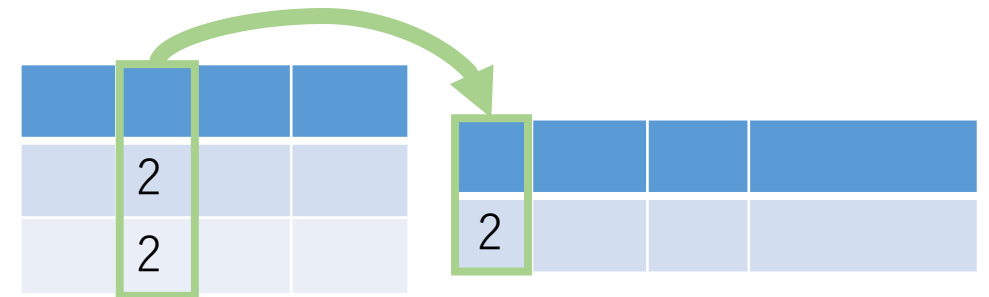
# 関係演算

- ・リレーショナル代数の演算は2種類(集合演算・**関係演算**)に分類できる。
- ・リレーショナルデータモデルに固有の主な関係演算としては、選択演算、射影演算、結合演算が有る。

選択演算の例



リレーショナル代数  
の適用例



# 基本SQL:データ参照

第一回の講義

・データベースに登録されているデータはselect文により参照することができる。このselect文では、いろいろな条件を与えて条件に一致するデータを検索することができるため、select文によるデータ参照のことを問合せ(クエリ,query)と呼ぶ。

#データ更新など、SQLによるデータ操作のための要求全般を問合せと呼ぶことも有る。

列名指定:射影演算

カラム値の定数絞込み:選択演算

select 列名1,...,列名m from テーブル名 [where 条件]  
select \* from テーブル名 [where 条件]

[ ]は、オプションであり、  
指定しなくても文が実行可能

\*は、列名全てと等価

```
insert into item values('A01','オフィス用紙A4',2000);  
insert into item(item_id, item_name) values('A03', "オフィス用紙A4");  
  
select * from item;  
select * from item where item_id = 'A01';  
select * from item where price >1000;
```

item\_id='A01'

item_id	item_name	price
A01	オフィス用紙A4	2000
A03	オフィス用紙A4	NULL

item_id	item_name	price
A01	オフィス用紙A4	2000

item_id	item_name	price
A01	オフィス用紙A4	2000

price  
>1000

# 基本SQL:データ更新

・データベースに登録されているデータの更新には次のような構文のupdate文を使う。これはテーブル名のテーブルに対し、条件を満たす行について列名1や列名2などの列を値1,値2のように更新する。値は、式を使った計算でも良い。条件を省略した場合は、すべての行が対象になる。

[ ]は、オプションであり、  
指定しなくても文が実行可能

```
update テーブル名 set 列名1 =値1 [,列名2 =値2 ...] [where 条件]
```

```
update item set price = price *1.1 where item_id='A01';
```

item_id	item_name	price
A01	オフィス用紙A4	2000
A03	オフィス用紙A4	NULL



item_id	item_name	price
A01	オフィス用紙A4	2200
A03	オフィス用紙A4	NULL

# 応用SQL:distinct句

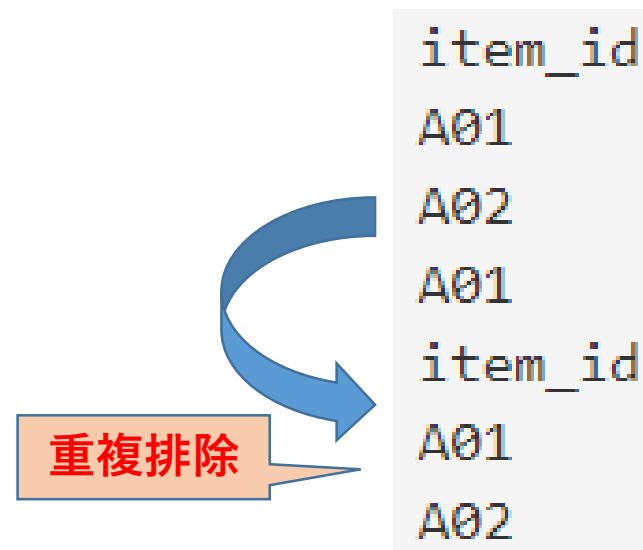
・元のテーブルにデータの重複がなくても、select文による問合せ結果に重複したデータが含まれることがある。問合せ結果から重複を取り除くためには、distinct句を選択する列名(カラムとも呼ぶ)の前につける。

```
select distinct 商品番号 from 注文
```

注文テーブル

注文番号	顧客番号	商品番号	商品名	個数	総額
001	C01	A01	オフィス用紙A4	1	2000
002	C01	A02	オフィス用紙A3	2	8000
003	C03	A01	オフィス用紙A3	3	6000

```
select item_id from order_t;  
select distinct item_id from order_t;
```



# 応用SQL:order by句

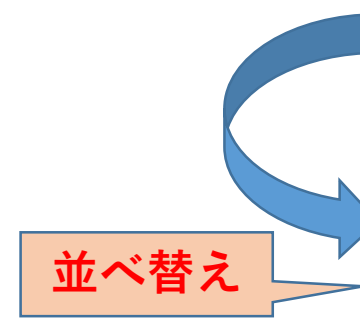
- ・テーブルは行(レコードとも呼ぶ)の集合であり、行の順番という概念は無い。しかし、数値や文字コードの順番で行を並べ替えたい場合がある。行を並べ替えるには、order by句を使う。  
#デフォルトでは昇順(ascending)である。降順(descending)を指定する場合には、列名の後にdescを指定する。

```
select 総額 from 注文 order by 総額 desc
```

注文テーブル

注文番号	顧客番号	商品番号	商品名	個数	総額
001	C01	A01	オフィス用紙A4	1	2000
002	C01	A02	オフィス用紙A3	2	8000
003	C03	A01	オフィス用紙A3	3	6000

```
select item_id, total from order_t;  
select item_id, total from order_t order by total;
```



item_id	total
A01	2000
A02	8000
A01	6000
item_id	total
A01	2000
A01	6000
A02	8000

# 応用SQL:複数テーブルからのデータ抽出

- ・ 複数のテーブルを結合するためには結合(join)演算を使う。結合演算の結果は、1つのテーブルとして扱える。結合演算では、データ型が同じか、もしくはデータ型の変換が可能である列を使って結合条件を指定する。
- ・ 内部結合:結合対象となるテーブルから結合条件を満たす行だけを取り出す演算であり、結合条件を満たさない行は欠落する。

商品テーブル

商品番号	商品名	価格
A01	オフィス用紙A4	2000
A02	オフィス用紙A3	4000
A03	オフィス用紙B5	1500

注文テーブル

注文番号	顧客番号	商品番号	商品名	個数	総額
001	C01	A01	オフィス用紙A4	1	2000
002	C01	A02	オフィス用紙A3	2	8000
003	C03	A01	オフィス用紙A3	3	6000

```
select * from item,order_t where item.item_id=order_t.item_id and order_t.total > 4000;
```

item_id	item_name	price	o_id	c_id	item_id	item_name	unit	total
A02	オフィス用紙A3	4000	002	C01	A02	オフィス用紙A3	2	8000
A01	オフィス用紙A4	2000	003	C03	A01	オフィス用紙A4	3	6000

## 課題3 締切:11/3

3-1) テーブル定義を作成し、distinct, order by, 内部結合を全て含んだSQLを実行し、処理の意味がどのようなになるか(例:注文があった商品のID)を記載せよ。「表定義と参照SQL, 実行結果」のスクリーンショットを提出ください。

なお、テーブル定義は次スライドと同じでもよいし、自分で作成しても良い。

```
create table order_t(o_id char(3)not null,  
                    c_id char(3) not null,  
                    item_id char(3) not null,  
                    item_name varchar(20),  
                    unit int,  
                    total int,primary key (o_id));  
create table item(item_id char(3) not null,  
                 item_name varchar(20),  
                 price int,primary key (item_id));  
insert into order_t values('O01','C01','A01','オフィス用紙A4',1,2000);  
insert into order_t values('O02','C01','A02','オフィス用紙A3',2,8000);  
insert into order_t values('O03','C03','A01','オフィス用紙A4',3,6000);  
insert into item values('A01','オフィス用紙A4',2000);  
insert into item values('A02','オフィス用紙A3',4000);  
insert into item values('A03','オフィス用紙B5',1500);  
select * from item,order_t where item.item_id=order_t.item_id and order_t.total > 4000;
```



# データベース管理システムの構成

SQL文を実行するための主な機能として下記がある。

- (1)SQL文解析:受信SQL文を解析し、使われているSQL句を明らかにする。
- (2)SQL文最適化:解析結果から、処理量減のため受信SQL文の変更やデータアクセスに索引を使うかの判断などを行い、クエリ実行プランを作成する。
- (3)クエリ実行エンジン:上記で作成されたクエリを実行する。

```
select *  
from tableA,tableB,tableC  
where tableA.a1=tableB.b1  
and tableB.b2=tableC.c2
```



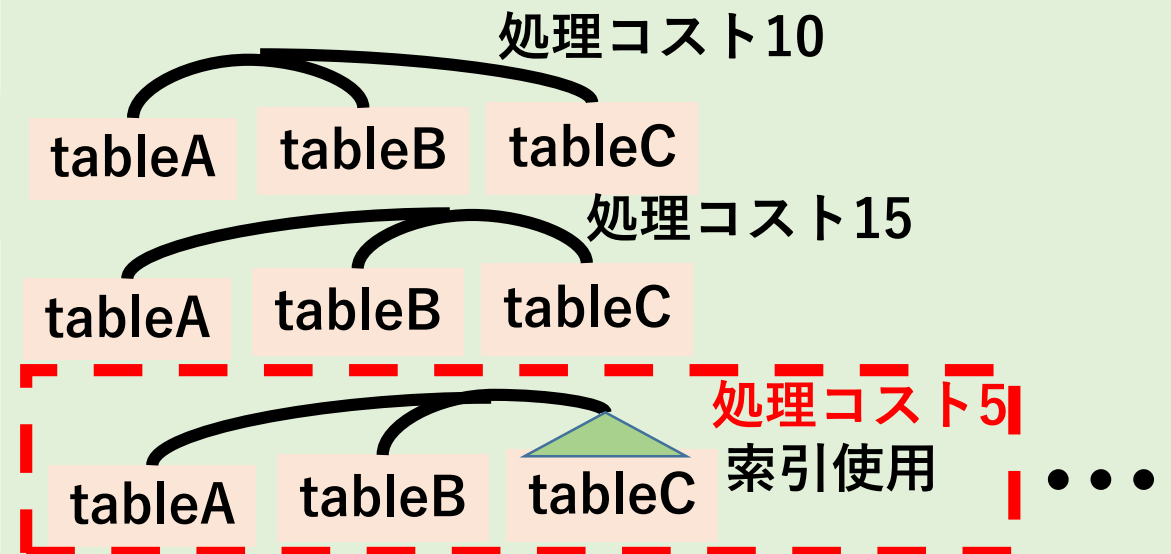
## データベース

SQLの解析

SQLの最適化

クエリ実行  
エンジン

抽出カラム: すべて  
操作対象表: tableA, tableB, tableC  
結合条件: tableA.a1=tableB.b1  
tableB.b2=tableC.c2



# インデックス(索引)

商用のデータベースでは大量データを扱っており、その中からある条件のレコードを抽出、演算して目的データを得る。何百万件とあるレコードを先頭から検索するのでは多大な時間を要し、実用に耐えられない。

**インデックス索引:**書籍巻末の索引のように、検索の手がかりとなる情報(あるカラムのカラム値)からレコード本体にアクセスできる仕組み

**インデックスファイル:**インデックスを構成するデータのまとめ

**select \* from ~ where A='A2'**

インデックス無し

すべてのレコードを調べて、カラムAの値がA2のレコードだけを抽出  
処理時間=表容量[GB]/シーケンシャルアクセス読み込み速度[GB/秒]

A			
A1			
A2			
A2			
...			

カラムAの値がA2のレコードだけを索引から抽出  
処理時間=「カラムAの値がA2」のレコード数[ランダムIO]/ランダムアクセス読み込み速度[IOPS(10 per sec)]

インデックス有り

A			
A1			
A2			
A2			
...			

一般に「全件をシーケンシャルアクセスの時間」 << 「全件をランダムアクセスの時間」であり、索引が早いのはアクセスレコード数の割合による

# テーブルに索引を作成

- create table文で索引を表に追加できる。

```
create table テーブル名(  
  列名d1 データ型 [not null]  
  [, 列名 d2 データ型 [not null] ...]  
  [, primary key (列名p1 [,列名p2...])]  
  [, foreign key (列名f1) references 参照テーブルf1 (参照列名f1)  
  [, foreign key (列名f2) references 参照テーブルf2 (参照列名f2) ...]  
  [,index 索引名(索引付与する列名)...]  
);
```

主キー  
#キー制約

外部キー  
#参照整合性制約

[ ]は、オプションであり、  
指定しなくても文が実行可能

索引作成

```
create table order_t(o_id char(3)not null,  
  c_id char(3) not null,  
  item_id char(3) not null,  
  item_name varchar(20),  
  unit int,  
  total int,primary key (o_id)  
  index index_cid(c_id));
```

c\_idカラムへ索引作成

# インデックス(索引)と実行プラン

詳細は次回以降

MySQLでは、explain命令により実行プランが確認できる

```
1 create table order_t(o_id char(3)not null,  
2                       c_id char(3) not null,  
3                       item_id char(3) not null,  
4                       item_name varchar(20),  
5                       unit int,  
6                       total int,primary key (o_id),  
7                       index index_cid(c_id));  
8 insert into order_t values('001','C01','A01','オフィス用紙A4',1,2000);  
9 insert into order_t values('002','C01','A02','オフィス用紙A3',2,8000);  
10 insert into order_t values('003','C03','A01','オフィス用紙A4',3,6000);  
11 explain select * from order_t where o_id= '001';  
12
```

primary key(=主キー)は  
自動的に索引作成される

実行 (Ctrl-Enter)

MySQLを学ぶ | プログラミング力診断

出力 入力 コメント 0

主キー索引を使用

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order_t	NULL	const	PRIMARY	PRIMARY	12	const	1	100.00	NULL

```
11 explain select * from order_t where c_id= 'C01';  
12
```

実行 (Ctrl-Enter)

MySQLを学ぶ | プログラミング力診断

出力 入力 コメント 0

cid索引を使用

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order_t	NULL	ref	index_cid	index_cid	12	const	1	100.00	Using index condition

## 課題3 締切:11/3

3-2)索引を含むテーブルを定義し、1表に対する問合せクエリ(結合クエリでなくても良い)の内容により「使われる索引が異なる場合がある事」を確認ください。そして、使用されている索引が異なるSQLの「表定義と参照SQL,実行結果」のスクリーンショットを提出ください。演習3-1と同じテーブル定義でも良いし、自分で作成してもよい。

索引が定義されている列に対して「選択演算」「結合演算」を行うと、索引が使用される可能性が有る  
→索引がある場合でも、処理が遅くなりそうなら索引が使われない。