データベース
第3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。

SQL:テーブルとテーブル定義

過去の講義

・テーブルはリレーションに相当する(列は属性に、行はタプルに対応)。テーブル定義では「データの格納型の種類や数、整合性制約」を定める。

・下記は、商品テーブルの例である。「item」はテーブル名であり、商品番号(item_id)、商品名(item_name)、価格の列(price)をもち、5行分のデータが格納されている。

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
B01	トナーカートリッジ黒	25000
C01	ホワイトボード	14000

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

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

SQL:データ挿入(格納)

過去の講義

・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");
```

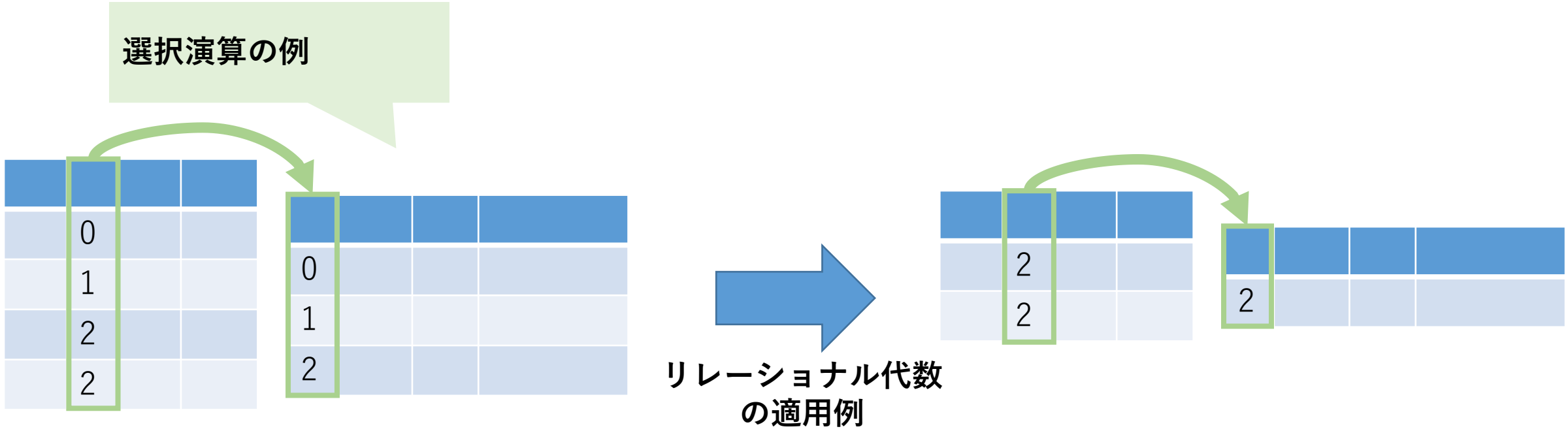
priceはnullとしてデータ格納

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

過去の講義

・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。



SQL:データ参照

過去の講義

・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

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

SQL:distinct句

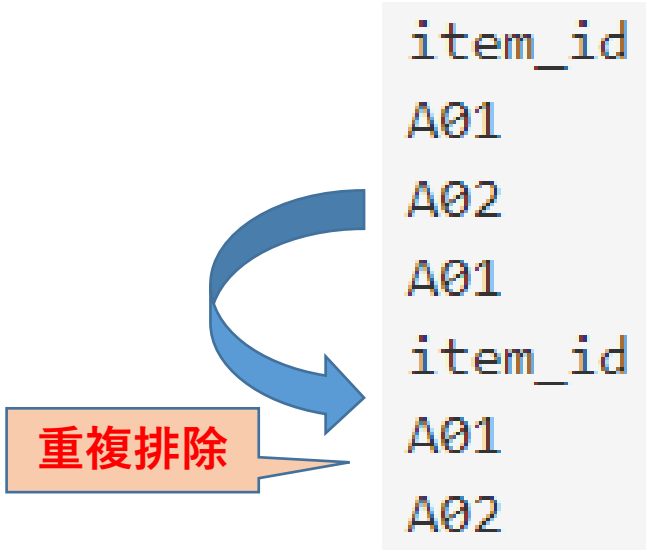
・元のテーブルにデータの重複がなくても、select文による問合せ結果に重複したデータが含まれることがある。問合せ結果から重複を取り除くためには、distinct句を選択する列名(カラムとも呼ぶ)の前につける。

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;
```



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]

[ ]は、オプションであり、指定しなくても文が実行可能

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

並べ替え

item_id	total
A01	2000
A01	6000
A02	8000

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:11/2

3-1) テーブル定義を作成し、あ)distinct,い)order by, う)内部結合を含んだSQLを実行し、処理の意味がどのようになるか(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、a),b),c)それぞれについて3つのクエリを作成すること

なお、テーブル定義は↓と同じでもよいし、自分で作成しても良い。
下記の表定義のSQLは次スライドに記載している。

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

```
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;
```

索引とデータベース管理システム

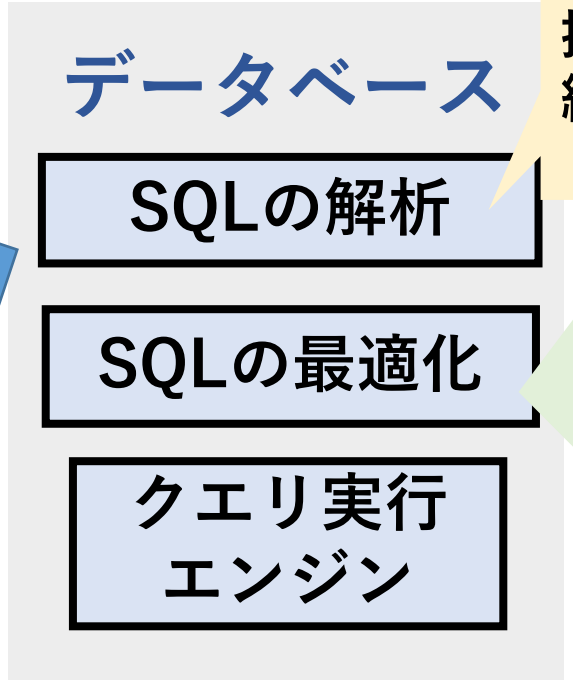
SQL文を実行するための主な機能として下記がある。

(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。

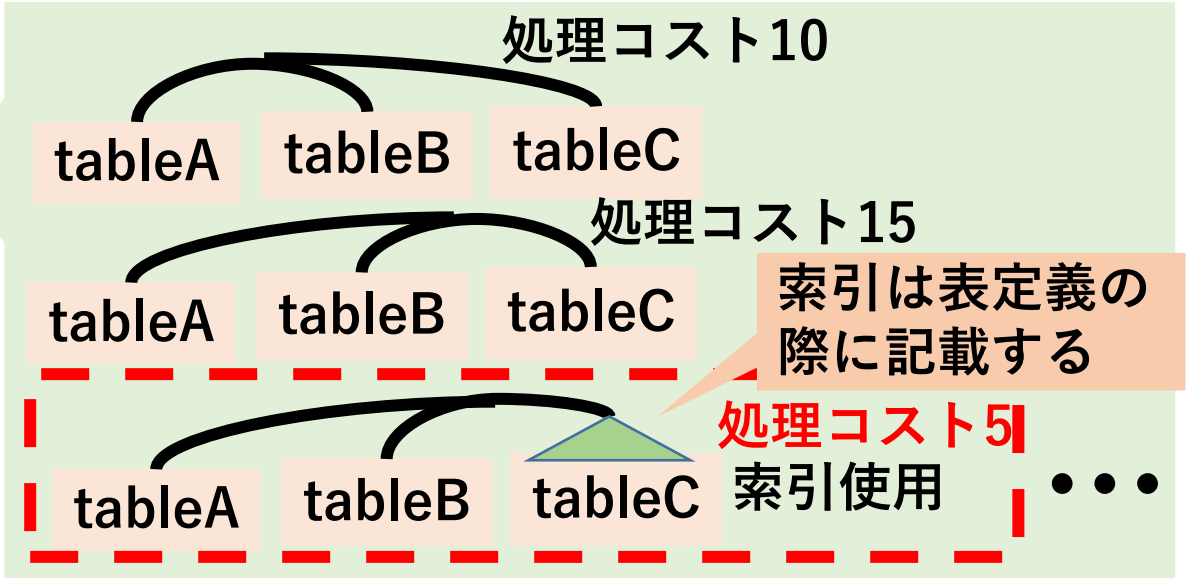
(2)SQL文最適化:解析結果から、処理量減のため受信SQL文の変更やデータアクセスに索引を使うかの判断などを行い、クエリ実行プランを作成する。

(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
```



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



実行プランとインデックス(索引)

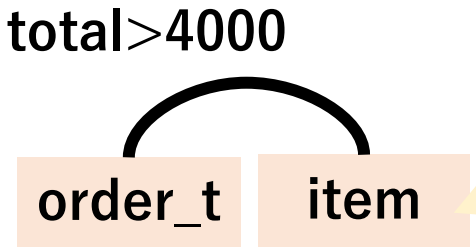
SQLは**「どのようなデータを取得するか」を指定するが、取得方法(テーブル間の結合順序などで実行プランと呼ばれる)は(一般的には)指定しない。**

下記SQLで指定された内容
1)item表とorder_t表をitem_idで結合し、2)totalが4000より大きいレコードを取得する。3)レコードの列は全て取得する。

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

下記の1)2)は、どちらを先にやっても良い
1)item表とorder_t表をitem_idで結合
2)totalが4000より大きいレコードをorder_t表から抽出

大容量データでは、実行プラン毎に処理時間が(例:100倍)大きく異なる。
MySQLでは、explain命令により実行プランが確認できる



SQL処理の例:
一般的に、(実行プランの図では)図の左側の表(外表や駆動表と呼ばれる)から処理→order_tに1)の絞込みをした後に、2)の結合を行う。

データベースでの表データの格納

・外部記憶(SSDなどの二次記憶)上にデータベースごとにディレクトリをつくり、テーブルのデータファイルや各テーブルの定義ファイルなどが格納される。

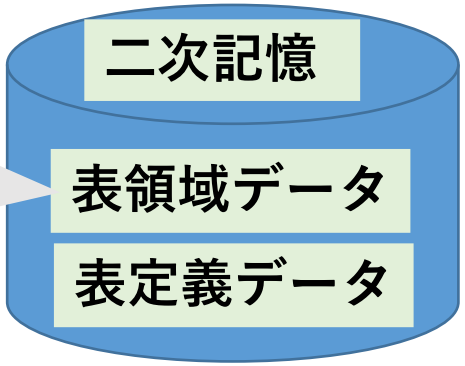
料理名	値段(円)
チャーハン	800
蟹チャーハン	1300
天津飯	900
餃子	500

バイナリデータとしてファイル書込み

チャーハン	800	蟹チャーハン
1300	…	

8KBなどの固定ページサイズ。→ページサイズでI/O性能が異なる

SQL処理では、全件がファイルから読みだされ、必要なデータを抽出する



・データベースソフト毎に、保存するデータ種類や保存形式が異なる。
・格納されるファイルは、すべて固定サイズ(通常8KBや16KBが多い)の「ページ」の集まりとして格納される。

表領域の特定レコードへ直接アクセス

表領域でレコードサイズは一定として一般的に格納されるため、取得したいレコードの先頭からの位置がわかれば、表領域全体を読込まなくてよい。

Execute | Share　main.c　STDIN

```c
#include <stdio.h>
struct Point{
  int x;
  int y;
};

int main()
{
    FILE *fp;
    struct Point p1,p2,p3;
    fp=fopen("test","wb");
    if(fp==NULL)printf("not open");
    else{
        p1.x=2;p1.y=-3;
        p3.x=4;p3.y=-5;
        fwrite(&p1,sizeof(struct Point),1,fp);
        fwrite(&p3,sizeof(struct Point),1,fp);
        p1.x=6;p1.y=-7;
        fwrite(&p1,sizeof(struct Point),1,fp);
        fclose(fp);
        fopen("test","rb");
        fseek(fp,sizeof(struct Point),SEEK_SET);
        fread(&p2,sizeof(struct Point),1,fp);
        printf("p2.x:%d p2.y:%d\n",p2.x,p2.y);
        fseek(fp,sizeof(struct Point)*2,SEEK_SET);
        fread(&p2,sizeof(struct Point),1,fp);
        printf("p2.x:%d p2.y:%d\n",p2.x,p2.y);
        fclose(fp);
    }
    return 0;
}
```

fwriteで順番に書き込まれる

2つめに書き込んだ構造体を読込み

Result

```
$gcc -o main *.c -lm
$main
p2.x:4 p2.y:-5
p2.x:6 p2.y:-7
```

int fseek(FILE *fp, long offset, int origin);
ファイル fp のファイル位置指示子を origin を基準として、offset バイト移動します。

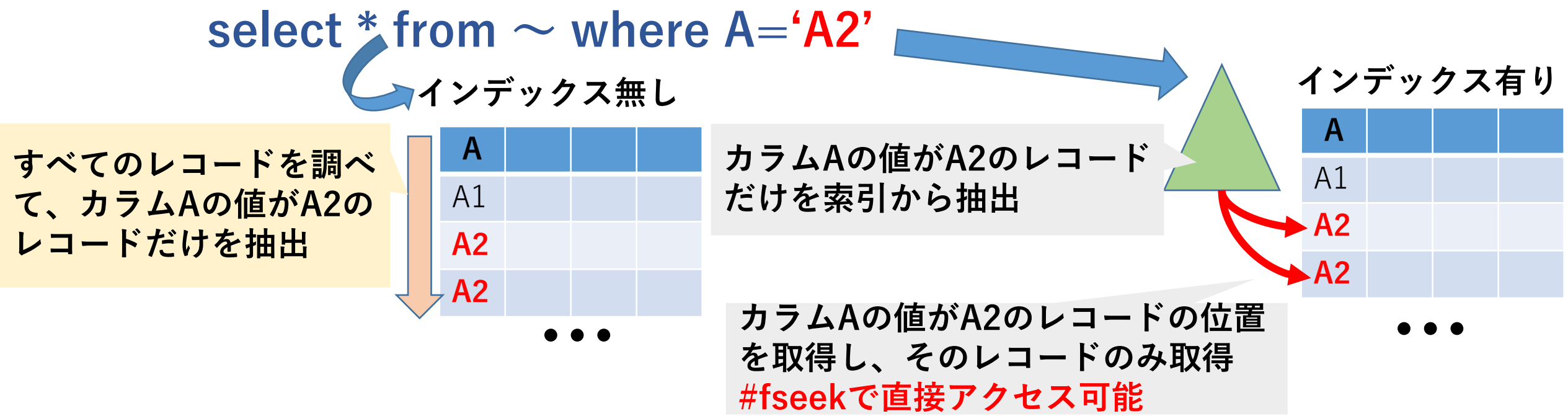
【引数】

FILE *fp	：	FILEポインタ
long offset	：	移動バイト数
int origin	：	SEEK_SET　（ファイルの先頭）
		SEEK_CUR　（ファイルの現在位置）
		SEEK_END　（ファイルの終端）

http://www9.plala.or.jp/sgwr-t/lib/fseek.html

インデックス(索引)

表領域でレコードサイズは一定として一般的に格納されるため、取得したいレコードの先頭からの位置がわかれば、表領域全体を読込まなくてよい。
→SQLで取得したいレコードは「あるカラムのカラム値がXのレコード」であるため、**「カラムとカラム値の組に該当するレコードのレコード位置」が高速に取得できる索引(一般にB木)が広く使われている**。



実行プラン

MySQLでは、explain命令により実行プランが確認できる

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

total>4000

order_t　item

(副問合せなどでない)シンプルなselect文は”SIMPLE”

表結合の型
→eq_refなら
結合処理を開始する表のレコード1行毎に1行読まれる

表条件で取り除かれるレコード数の推定割合
33.33は33%が残る

id	select type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order_t	NULL	ALL	NULL	NULL	NULL	NULL	3	33.33	Using where
1	SIMPLE	item	NULL	eq_ref	PRIMARY	PRIMARY	12	test.order_t.item_id	1	100.00	NULL

処理対象の表名

「使用可能な索引」と「実際に使われた索引」
itemの索引を使用

https://dev.mysql.com/doc/refman/8.0/ja/explain-output.html

実行プラン2

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

total>4000

totalには
索引は無い

order_t

item

主キー索引アクセス
→主キー索引(item_id索引)で1レコード取得

表スキャン
→表のレコード全てを先頭から順番に取得

filtered:レコード取得後に、where句条件適用後に(統計情報によれば)どれだけ残る想定か
→MySQLは、total>4000の絞込みで33.33%残る予定で、実行プランを作成している

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order_t	NULL	ALL	NULL	NULL	NULL	NULL	3	33.33	Using where
1	SIMPLE	item	NULL	eq_ref	PRIMARY	PRIMARY	12	test.order_t.item_id	1	100.00	NULL

type:アクセスタイプ(データへのアクセス方法)
ALL→テーブルスキャン
eq_ref→JOINにおいて、主キーやユニークキーによるアクセス
#最大一行だけ取得

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)…]
);
```

[ ]は、オプションであり、指定しなくても文が実行可能

主キー
#キー制約

外部キー
#参照整合性制約

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

```
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  create table item(item_id char(3) not null,
8                       item_name varchar(20),
9                       price int);
10 insert into order_t values('O01','C01','A01','オフィス用紙A4',1,2000);
11 insert into order_t values('O02','C01','A02','オフィス用紙A3',2,8000);
12 insert into order_t values('O03','C03','A01','オフィス用紙A4',3,6000);
13 insert into item values('A01','オフィス用紙A4',2000);
14 insert into item values('A02','オフィス用紙A3',4000);
15 insert into item values('A03','オフィス用紙B5',1500);
16 explain select * from item,order_t where item.item_id=order_t.item_id and order_t.total > 4000;
```

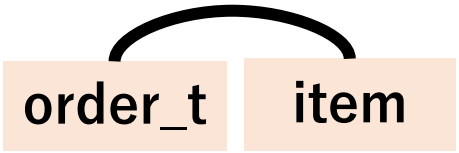
実行 (Ctrl-Enter)

両方、表スキャン

索引使用せず

出力 入力 コメント 0

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order_t	NULL	ALL	NULL	NULL	NULL	NULL	3	33.33	Using where
1	SIMPLE	item	NULL	ALL	NULL	NULL	NULL	NULL	3	33.33	Using where; Using join buffer (hash join)




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

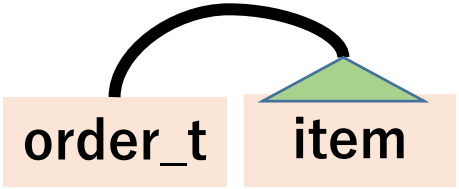
実行 (Ctrl-Enter)

order_t:表スキャン
item:最大1行取得

主キー索引を使用

出力 入力 コメント 0

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order_t	NULL	ALL	NULL	NULL	NULL	NULL	3	33.33	Using where
1	SIMPLE	item	NULL	eq_ref	PRIMARY	PRIMARY	12	test.order_t.item_id	1	100.00	NULL



課題3 締切:11/2

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。

```
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

keyの所に、何か索引名があればOK

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

内部結合とMySQL

・一般的に(MySQL含め)、where句で=条件でカラム同士を指定すると、内部結合となる。

SQLでは他表記での表結合もありますが、テスト回答ではMySQLで正しく出力されれば他表記でも正答とします。

- WHERE 句 (指定されている場合) は、選択されるために行が満たす必要のある 1 つまたは複数の条件を示します。***where_condition*** は、選択される各行に対して true に評価される式です。 WHERE 句がない場合、このステートメントはすべての行を選択します。

各行が条件を満たす必要がある
→片方の(表の)行しか条件を満たさない行は出ないので、内部結合となる

内部結合が一番大事なので、
とりあえずは内部結合だけ覚えておけば良い

```
1  select version();
```

実行 (Ctrl-Enter)

出力 入力 コメント 0

```
version()
8.0.26-0ubuntu0.20.04.2
```

https://dev.mysql.com/doc/refman/8.0/ja/select.html

SQL:複数テーブルからのデータ抽出

過去の講義

・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

where句に結合条件を書かない場合

・from句にて、join句を用いることで表結合が行える。
#where句で記載するのと、基本的に同じ効果

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

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	o_id	c_id	item_id	item_name	unit	total
A02	オフィス用紙A3	4000	O02	C01	A02	オフィス用紙A3	2	8000
A01	オフィス用紙A4	2000	O03	C03	A01	オフィス用紙A4	3	6000
item_id	item_name	price	o_id	c_id	item_id	item_name	unit	total
A02	オフィス用紙A3	4000	O02	C01	A02	オフィス用紙A3	2	8000
A01	オフィス用紙A4	2000	O03	C03	A01	オフィス用紙A4	3	6000

MySQLでの自然結合

・natural join処理をすると、自然結合してくれますが、「結合カラムのカラム名の語句」が完全に一致する必要があります。

```
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 order_t natural join item;
```

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	o_id	c_id	unit	total	price
A01	オフィス用紙A4	O01	C01	1	2000	2000
A02	オフィス用紙A3	O02	C01	2	8000	4000
A01	オフィス用紙A4	O03	C03	3	6000	2000

結合カラムの語句は、表毎にちょっと違う事も多いので、「内部結合して、射影処理(select のカラム選択)で、必要なカラムだけ抽出」することが多い。

SQL:集約関数(count)

DBMSでは組込み関数が用意されており、select文の中でこれらの関数を呼び出すことができる。集約関数は、複数のレコードに対する平均値や合計値などの集約値を得ることができる。

count関数)列名を引数とし、その列に含まれる行数をカウントする。

select count(*) from 注文;

```
select count(*) from order_t;
```

```
count(*)
3
```

レコード数3

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

SQL:集約関数(count as)

count (列名) as Aとすることで、結果出力の列名をAにできる。

```
select count(*) from order_t;
```

count(*)
3

select count(*) as 注文総数 from 注文;

```
select count(*) as total_record from order_t;
```

total_record
3

SQL:集約関数(distinct count)

count (distinct 列名)とすることで、ユニークな(重複していない)行数が得られる。

select count(distinct 商品名) from 注文;

```
select count(distinct item_name) from order_t;
```

```
count(distinct item_name)
2
```

列「商品名」で
ユニークなレコード数は2

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

SQL:集約関数(max,min)

max関数は、列名を引数とし、その列に含まれるデータの最大値を返す。
同様に、min関数は、最小値を返す。

select min(総額) from 注文;

```
select min(total) from order_t;
select max(total) from order_t;
```

```
min(total)
2000
max(total)
8000
```

列「総額」
→最小値2000,最大値8000

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

SQL:集約関数(avg,sum)

avg関数)列名を引数とし、その列に含まれるデータの平均値を返す。
sum関数)列名を引数とし、その列に含まれるデータの合計値を返す。

select avg(総額) from 注文;

```
select avg(total) from order_t;
select sum(total) from order_t;
```

```
avg(total)
5333.3333
sum(total)
16000
```

列「総額」
→平均値5333.3333,合計値16000

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

課題4 締切:11/10

4-1)集約関数を用いた例(表定義とクエリ)を作成せよ。
テーブル定義は下記でもよいし、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;
```

SQL:group by句

group by句:テーブルの特定の列で行をグループ化し、個々のグループに対して集約演算を適用するときに使用する。

select 顧客ID,sum(総額) from 注文 group by 顧客ID;

```
select c_id,sum(total) from order_t group by c_id;
```

c_id	sum(total)
C01	10000
C03	6000

顧客番号ごとの注文総額

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

SQL:group by句2

group by句:集約対象のレコードを(選択条件で)限定することは、where句で行える。

select 顧客ID,sum(総額) from 注文 where 個数>=2 group by 顧客ID;

```
select c_id,sum(total) from order_t where unit>=2 group by c_id;
```

c_id	sum(total)
C01	8000
C03	6000

顧客番号ごとの注文総額
ただし、注文個数が2以上の注文のみ集計

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

SQL:having句

having句:集約された値に対する選択条件はhaving句で指定する。

select 顧客ID,sum(総額) from 注文 group by 顧客ID having count(顧客ID)>=2;

```
select c_id,sum(total) from order_t group by c_id having count(c_id)>=2;
```

c_id	sum(total)
C01	10000

顧客番号ごとの注文総額を
「注文レコード数が2以上の顧客」についてだけ求める

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

課題4 締切:11/10

4-2)group by句を用いた例(表定義とクエリ)を作成せよ。
テーブル定義は下記でもよいし、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;
```

SQL:副問合せ

・select文中にselect文を記述する(入れ子にする)ことができ、入れ子の内側の問合せを副問合せ、外側の問合せを主問合せと呼ばれる。
・副問合せを用いたselect文では、まず内側の副問合せを実行して値を返し、その値を主問合せで受けて最終的な結果を生成する。
・副問合せの結果を主問合せに連携するために比較演算子(=,<,>など)およびin,exist,any,some,all演算子を使う。

『「総額の平均」以下の「総額」のレコード』についてのみ、顧客番号ごとに集約

例)比較演算子を使った副問合せ

```
select c_id,sum(total) from order_t where total <= (select avg(total) from order_t) group by c_id;
```

c_id	sum(total)
C01	2000

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

SQL:副問合せ(inとnot in演算子)

副問合せの結果が行の集合(1行だけもOK)の場合、inおよびnot in演算子を使って主問合せと連携する。

select 顧客番号,商品名,総額 from 注文 where 顧客番号 in (select 顧客番号 from 注文 where 総額>=7000);

not inと置き換え可能

『「総額」が7000以上の顧客番号』についてのみ、顧客番号・商品名・総額を出力

```
select c_id,item_name,total from order_t where c_id in (select c_id from order_t where total >=7000);
```

c_id	item_name	total
C01	オフィス用紙A4	2000
C01	オフィス用紙A3	8000

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

SQL:副問合せ(比較演算子)

副問合せの結果が1行(avg,sumなど)の場合、比較演算子(=,<,>など)を使って主問合せと連携する。

select 顧客ID,sum(総額) from 注文 where 総額<=(select avg(総額) from 注文) group by 顧客ID;

『「総額の平均」以下の「総額」のレコード』についてのみ、顧客番号ごとに集約

```
select c_id,sum(total) from order_t where total <= (select avg(total) from order_t) group by c_id;
```

c_id	sum(total)
C01	2000

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

課題4 締切:11/10

4-3)副問合せを用いた例(表定義とクエリ)を作成せよ。
テーブル定義は下記でもよいし、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;
```