データベース
第6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	
	10週	正規化	およる。
	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。

模擬試験:ポートフォリオ点に加算。課題だけで、ポートフォリオ点は満点になる。

(時間あれば)データベース管理システムの構成

過去の講義

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
```

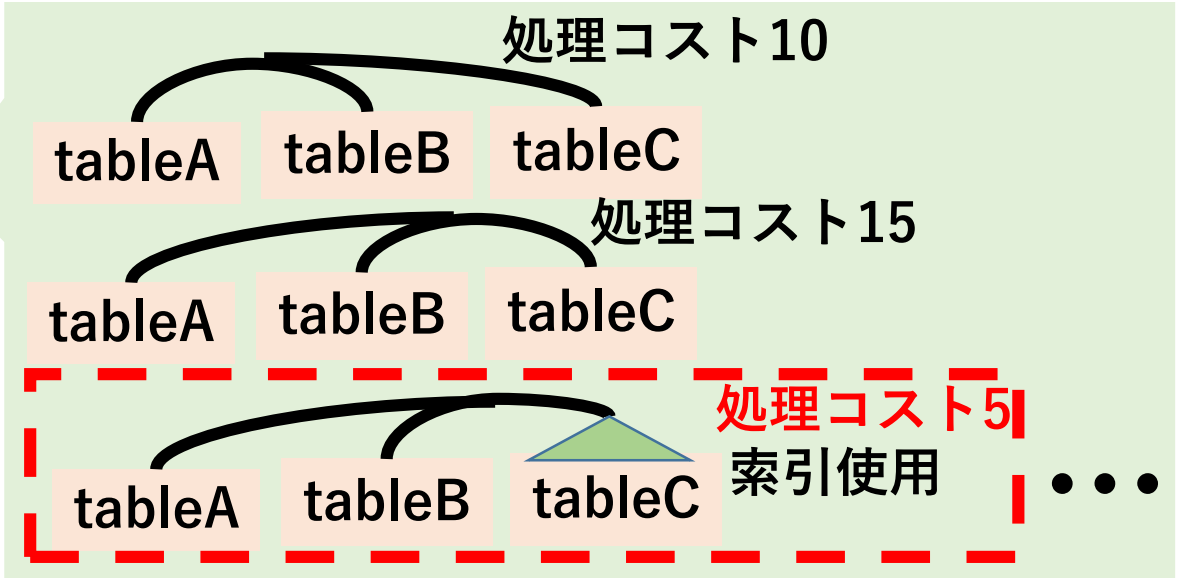
データベース

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



実行プランとインデックス(索引)

過去の講義

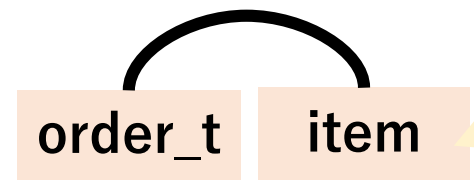
SQLは「どのようなデータを取得するか」を指定するが、取得方法(テーブル間の結合順序などで実行プランと呼ばれる)は(一般的には)指定しない。

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

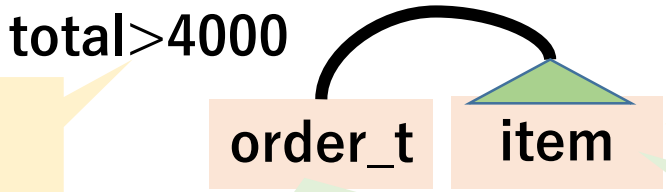


SQL処理の例:
一般的に、(実行プランの図では)図の左側の表(外表や駆動表と呼ばれる)から処理
→order_tに1)の絞込みをした後に、2)の結合を行う。

データベースのクエリ処理の実行時間

クエリ処理では表データを読込む(or書込む)必要がある。表アクセス時間(読込みor書込み)は各種情報(表サイズ、スループットなど)から計算できる

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



totalには
索引は無い

item表のitem_id
は索引がある

表スキャン
→表のレコード全てを先頭から順番に取得
処理時間[秒]=表サイズ[GB]/二次記憶の読込み
スループット[GB/秒]

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
処理時間[秒]=索引IO数/(IO処理時間≒IOPS[IO/秒])
#実際はIO待ち時間があるので、IOPSより遅い

データベースでの表データの格納

過去の講義

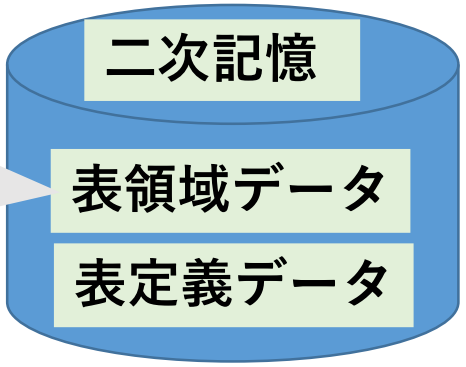
・外部記憶(SSDなどの二次記憶)上にデータベースごとにディレクトリをつくり、テーブルのデータファイルや各テーブルの定義ファイルなどが格納される。

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。

外部記憶装置の種類1

・記憶媒体の違い:HDD,SSD

HDD:磁性体を塗布した円盤(ディスク)の円周上にデータを書き込む。ディスク上のデータ読み書きでは、磁気ヘッドを読み書き位置に移動するという機械的な移動時間がかかるため、ランダムアクセスが比較的遅い。

SSD:半導体メモリを利用した外部記憶装置をSSD(Solid State Drive)と呼ぶ。HDDと異なりディスクを使っていないので、機械的な移動が無くランダムアクセスが高速に行える。

外部記憶装置の種類2

・サーバとの接続インタフェース:

DAS(Direct Attached Storage):1台のコンピュータに直接外部記憶装置を接続する形態。利点:接続が簡単で、導入コストが低い。欠点:複数のコンピュータでHDDを共有できない。例:外付けHDDをパソコンに接続

NAS(Network Attached Storage):複数のコンピュータがIPネットワークを介して複数のストレージに共有できるようにしたもの。データの共有単位はファイル。

高信頼・大容量(例1PB(=1024TB))で(SANなら)複数サーバから共有可能。Logical Volume(例:1つ10TB)という単位で各サーバに割り当て可能

SAN(Storage Area Network):サーバとストレージ間を独自の高速ネットワークで接続したネットワーク共有ストレージ。SANはサーバからはDASと認識され、HDDと同じように扱うことができ、ページ単位の共有が可能である。#ディスクアレイなどで、主にファイバーチャネルなど特別な接続方法が利用される。

1台1億円とかするものもある。→高すぎるのでクラウドで良いのではという流れ

IOPSとスループット

SQLクエリの問合せにDBMSが応答するために、各領域(例:表領域)のページを二次記憶から読書き(I/O)する必要がある。**一般に、I/O処理がSQL処理時間の大部分を占めるため、I/O処理時間の見積りが重要**である。

I/O処理時間の見積りは「処理に必要なI/O量÷システムのI/O処理能力」で算出される。過去、二次記憶媒体としてHDDが広く使われており、HDDの機械的構造を起因として「連続領域への読書き」と「読書き位置が小刻みに変わる場合」でI/O能力が大きく異なるため、下記の指標が用いられる。

IOPS(秒単位のIO処理能力 IO Per Second):ランダムアクセス時のI/O回数
→主に、索引ページへのアクセス性能
スループット:シーケンシャルアクセス時のI/Oデータ量
→主に、表領域へのアクセス性能

IOPSとスループット2

total>4000

order_t　item

中間後に、演習室PCでちょっと大きい
データで処理時間の違いを見てもらう予定

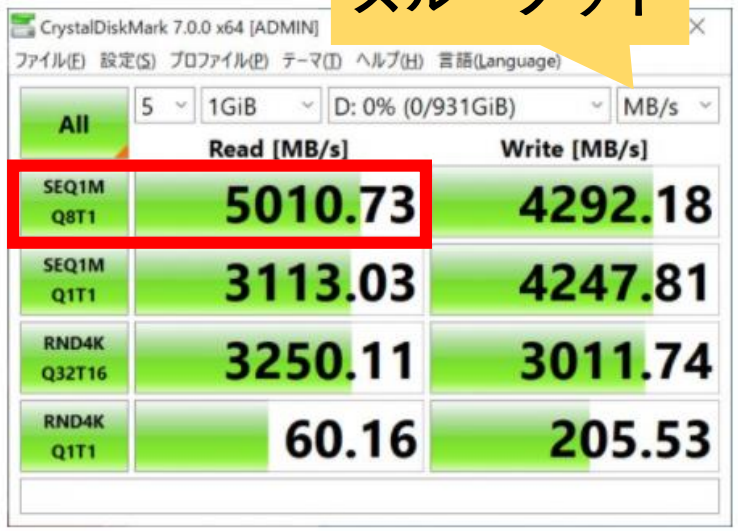
主キー索引アクセス
→主キー索引(item_id索引)で1レコード取得
処理時間[秒]=索引IO数/(IO処理時間≒IOPS[IO/秒])
#実際はIO待ち時間があるので、IOPSより遅い

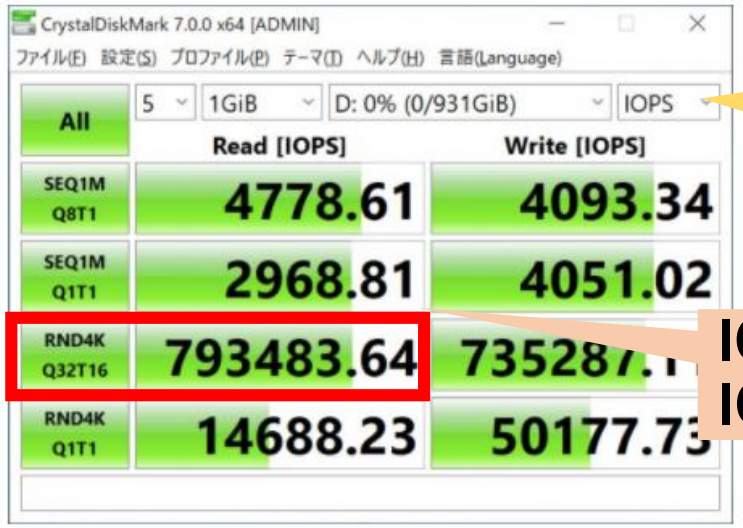
表スキャン
→表のレコード全てを先頭から順番に取得
処理時間[秒]=表サイズ[GB]/二次記憶の読込み
スループット[GB/秒]

CrystalDiskMark（ストレージ/BTO/ノートPC）

スループットとIOPSの例
→読書きするデータサイズ毎に性能が異なる

スループット





IOPS

スループットは
linuxのddコマンド
などで測定可能

IOサイズが小さいほど、
IOPSは高い

引用元https://www.gdm.or.jp/review/2019/1231/334382/3

データベースのクエリ処理の実行時間

「クエリ処理の時間見積り」には、表アクセス時間に加えて、表結合の処理時間が必要である。代表的な結合方式である、ネストループ結合(NL結合)の処理時間を考える。

ネストループ結合の処理時間＝
外部表の読込み時間+(外部表のレコード数×絞込み)×内部表の読込み処理

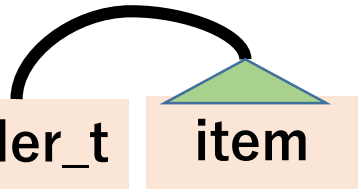
表スキャンor索引アクセス

表スキャンor索引アクセス

```
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



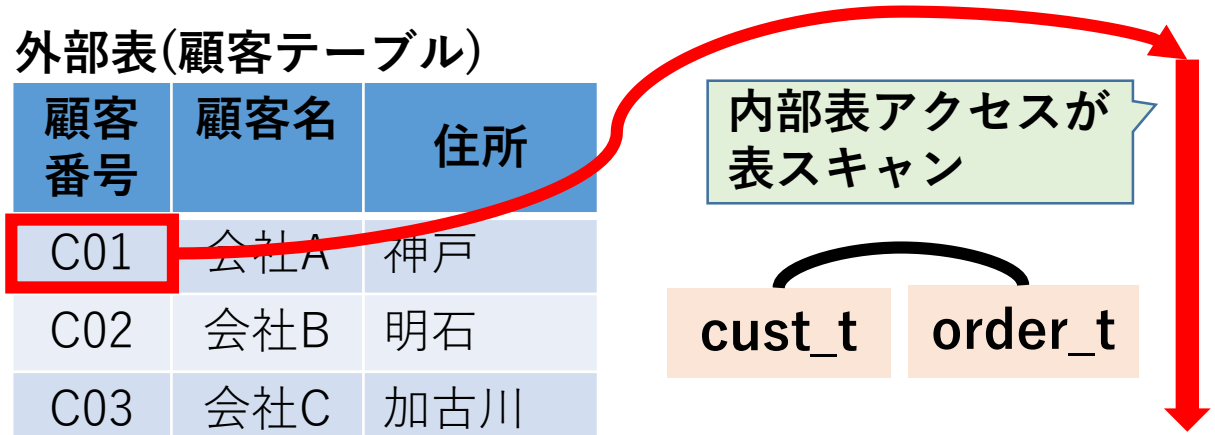
item表のitem_id
は索引がある

ネストループ結合(Nested Loops)

入れ子のループを使う結合であり、結合を開始するレコードが格納されている表(外部表)のレコードを1行毎ループしながら表スキャンし、もう一方の表(内部表)と結合する。具体的には、外部表から取り出されたレコードと結合条件が合致するレコードが内部表に存在するかを1レコード毎に行う。

外部表(顧客テーブル)

顧客番号	顧客名	住所
C01	会社A	神戸
C02	会社B	明石
C03	会社C	加古川

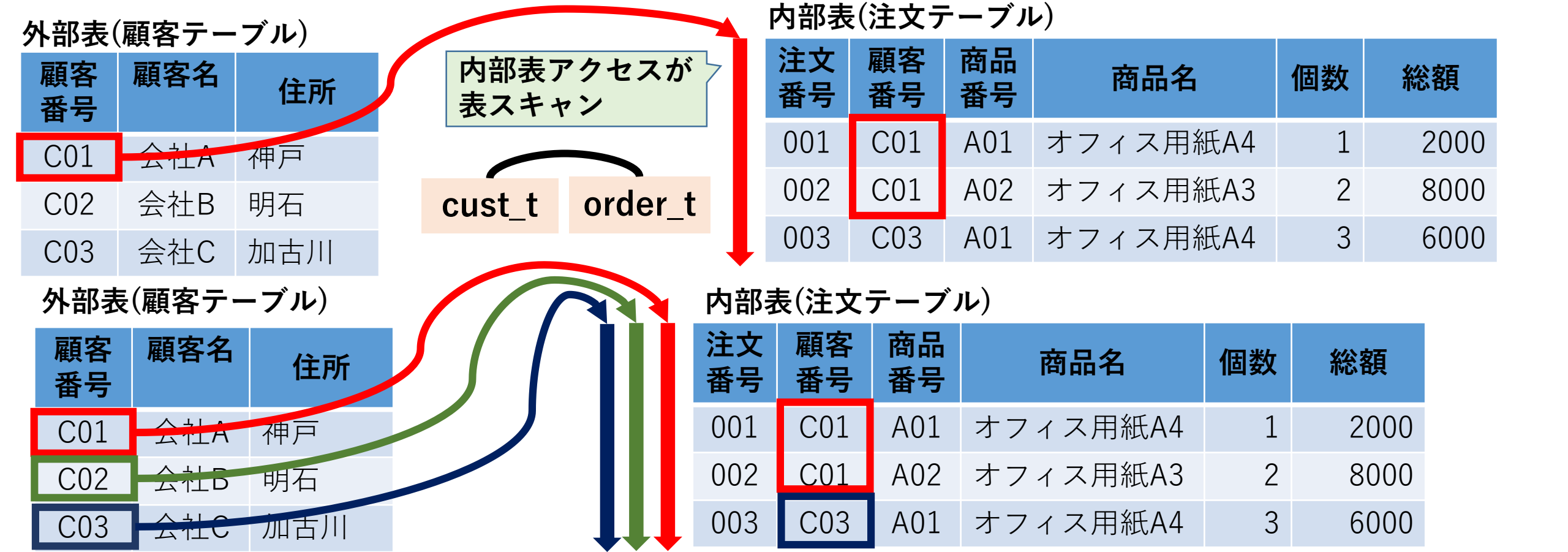


内部表(注文テーブル)

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A4	3	6000

ネストループ結合(Nested Loops)

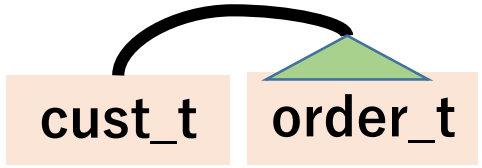
入れ子のループを使う結合であり、結合を開始するレコードが格納されている表(外部表)のレコードを1行毎ループしながら表スキャンし、もう一方の表(内部表)と結合する。具体的には、外部表から取り出されたレコードと結合条件が合致するレコードが内部表に存在するかを1レコード毎に行う。



ネストループ結合(Nested Loops)

外部表(顧客テーブル)

顧客番号	顧客名	住所
C01	会社A	神戸
C02	会社B	明石
C03	会社C	加古川



内部表アクセスが索引
例:「C01」のカラム値位置
を索引取得し、直接アクセス

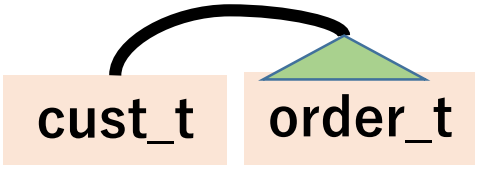
内部表(注文テーブル)

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A4	3	6000

ネストループ結合(Nested Loops)

外部表(顧客テーブル)

顧客番号	顧客名	住所
C01	会社A	神戸
C02	会社B	明石
C03	会社C	加古川



内部表アクセスが索引
例:「C03」のカラム値位置
を索引取得し、直接アクセス

内部表(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A4	3	6000

ネストループ結合の処理時間

「外部表の読込み時間」と「外部表から読込まれたクエリ結果を生成するために内部表と結合処理する必要があるレコード件数」×内部表の読込み時間の合計が、ネストループ結合の処理時間となる。

外部表(顧客テーブル)

顧客番号	顧客名	住所
C01	会社A	神戸
C02	会社B	明石
C03	会社C	加古川

内部表(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A4	3	6000

外部表から内部表へ渡されるレコード数=外部表のレコード数×絞込み
例:3行×0.33=1行 #C01のみ

内部表のレコード数

一般に、レコード数が小さい表を外部表とする

ネストループ結合の処理時間=
外部表の読込み時間+(外部表のレコード数×絞込み)×内部表の読込み処理

表スキャンor索引アクセス

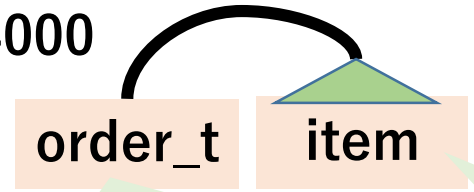
表スキャンor索引アクセス

インデックス(索引)と実行プラン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



表スキャン
→表のレコード全てを先頭から順番に取得
処理時間[秒]=表サイズ[GB]/二次記憶の読込み
スループット[GB/秒]

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
処理時間[秒]=索引IO数/(IO処理時間≒IOPS[IO/秒])
#実際はIO待ち時間があるのでIOPSより遅いが、処理時間見積りでは↑の式で良いとする。

ネストループ結合の処理時間=
外部表の読込み時間+(外部表のレコード数×絞込み)×内部表の読込み処理

表スキャンor索引アクセス

表スキャンor索引アクセス

表領域の特定レコードへ直接アクセス

過去の講義

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】

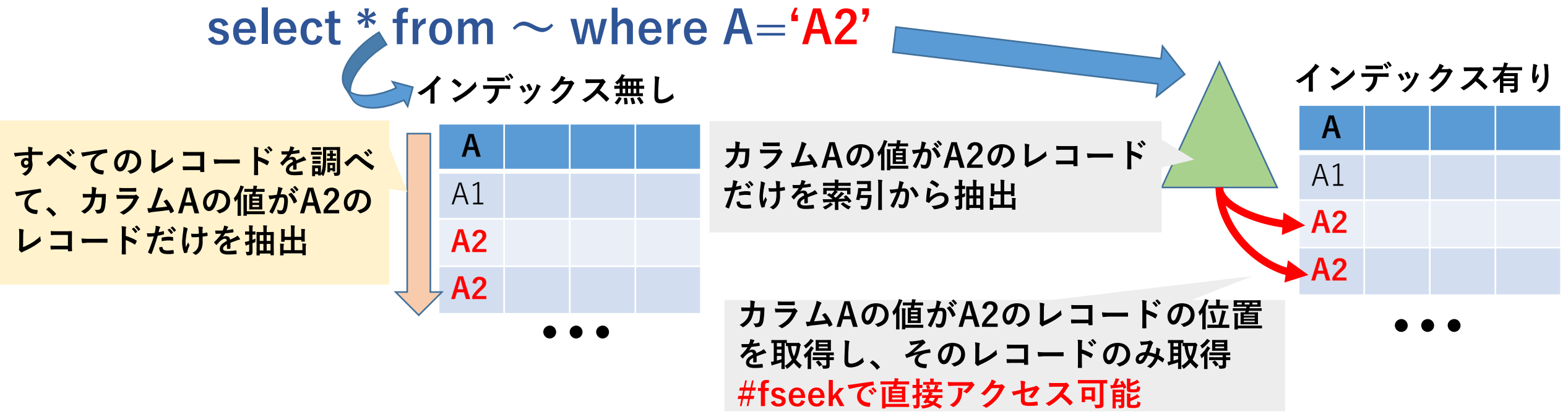
FILE *fp	：	FILEポインタ
long offset	：	移動バイト数
int origin	：	SEEK_SET　（ファイルの先頭）
		SEEK_CUR　（ファイルの現在位置）
		SEEK_END　（ファイルの終端）

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

インデックス(索引)

過去の講義

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

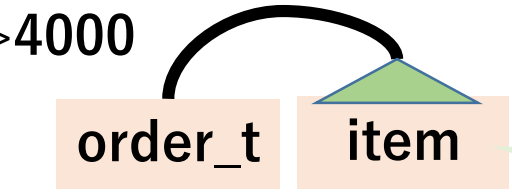
https://dev.mysql.com/doc/refman/8.0/ja/explain-output.html

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



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

インデックス(索引)と実行プラン3

```
explain format=json select cust.c_id,cust.city from cust,order_t where  order_t.c_id= cust.c_id;
```

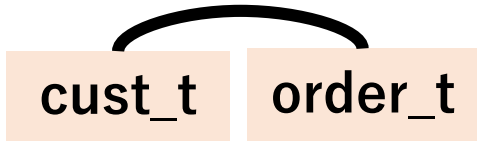
```
{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "2.21"
    },
    "nested_loop": [
      {
        "table": {
          "table_name": "cust",
          "access_type": "ALL",
          "rows_examined_per_scan": 3,
          "rows_produced_per_join": 3,
          "filtered": "100.00",
          "cost_info": {
            "read_cost": "0.50",
            "eval_cost": "0.30",
            "prefix_cost": "0.80",
            "data_read_per_join": "528"
          },
          "used_columns": [
            "c_id",
            "city"
          ]
        }
      },
      {
        "table": {
          "table_name": "order_t",
          "access_type": "ALL",
          "rows_examined_per_scan": 3,
          "rows_produced_per_join": 3,
          "filtered": "33.33",
          "using_join_buffer": "hash join",
          "cost_info": {
            "read_cost": "0.51",  "eval_cost": "0.30",
            "prefix_cost": "2.21",  "data_read_per_join": "384"
          },
          "used_columns": [
            "c_id"
          ],
          "attached_condition": "(`test`.`order_t`.`c_id` =
`test`.`cust`.`c_id`)"
        }
      }
    ]
  }
}
```

NL結合

表スキャン

表スキャン

コストベース最適化
→各プランの処理コストを算出し、最小の実行プランを採用

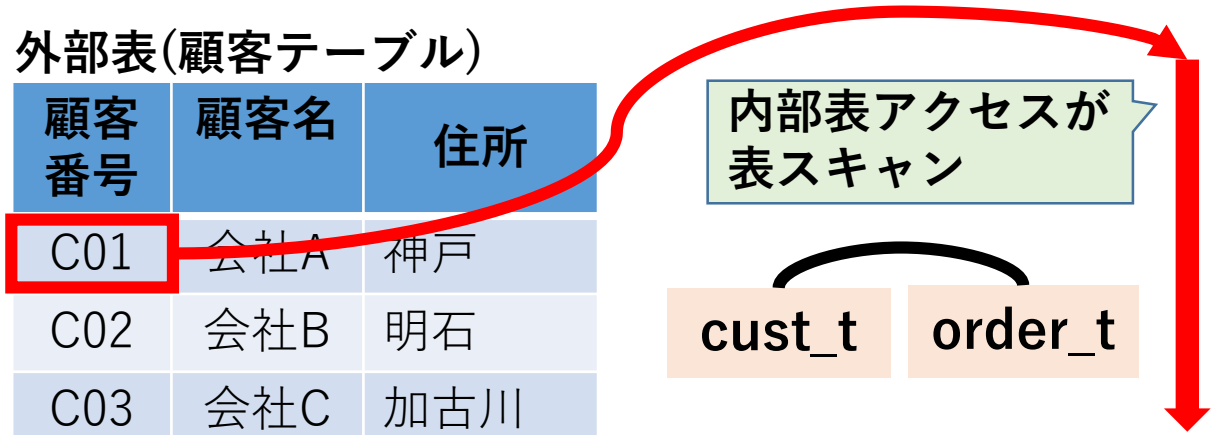


ネストループ結合(Nested Loops)

入れ子のループを使う結合であり、結合を開始するレコードが格納されている表(外部表)のレコードを1行毎ループしながら表スキャンし、もう一方の表(内部表)と結合する。具体的には、外部表から取り出されたレコードと結合条件が合致するレコードが内部表に存在するかを1レコード毎に行う。

外部表(顧客テーブル)

顧客番号	顧客名	住所
C01	会社A	神戸
C02	会社B	明石
C03	会社C	加古川



内部表(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A4	3	6000

ネストループ結合(Nested Loops)2

入れ子のループを使う結合であり、結合を開始するレコードが格納されている表(外部表)のレコードを1行毎ループしながら表スキャンし、もう一方の表(内部表)と結合する。具体的には、外部表から取り出されたレコードと結合条件が合致するレコードが内部表に存在するかを1レコード毎に行う。

外部表(顧客テーブル)

顧客番号	顧客名	住所
C01	会社A	神戸
C02	会社B	明石
C03	会社C	加古川

内部表アクセスが索引
例:「C01」のカラム値位置
を索引取得し、直接アクセス

内部表(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A4	3	6000

cust_t **order_t**

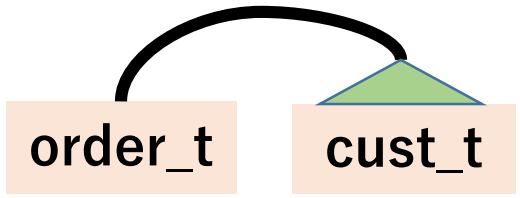
インデックス(索引)と実行プラン4

```
explain format=json select cust.c_id,cust.city from cust,order_t where  order_t.c_id= cust.c_id;
```

```
{  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "1.86“
    },
    "nested_loop": [
      {
        "table": {
          "table_name": "order_t",
          "access_type": "ALL",
          "rows_examined_per_scan": 3,
          "rows_produced_per_join": 3,
          "filtered": "100.00",
          "cost_info": {
            "read_cost": "0.51",
            "eval_cost": "0.30",
            "prefix_cost": "0.81",
            "data_read_per_join": "384“
          },
          "used_columns": [
            "c_id“
          ]
        }
      },
      {
        "table": {
          "table_name": "cust",
          "access_type": "eq_ref",
          "possible_keys": [ "PRIMARY“ ],
          "key": "PRIMARY",
          "used_key_parts": [    "c_id“  ],
          "key_length": "12",
          "ref": [    "test.order_t.c_id“  ],
          "rows_examined_per_scan": 1,
          "rows_produced_per_join": 3,
          "filtered": "100.00",
          "cost_info": {
            "read_cost": "0.75",    "eval_cost": "0.30",
            "prefix_cost": "1.86",  "data_read_per_join": "528“
          },          "used_columns": ["c_id",  "city" ]
        }
      }
    ]
  }
}
```

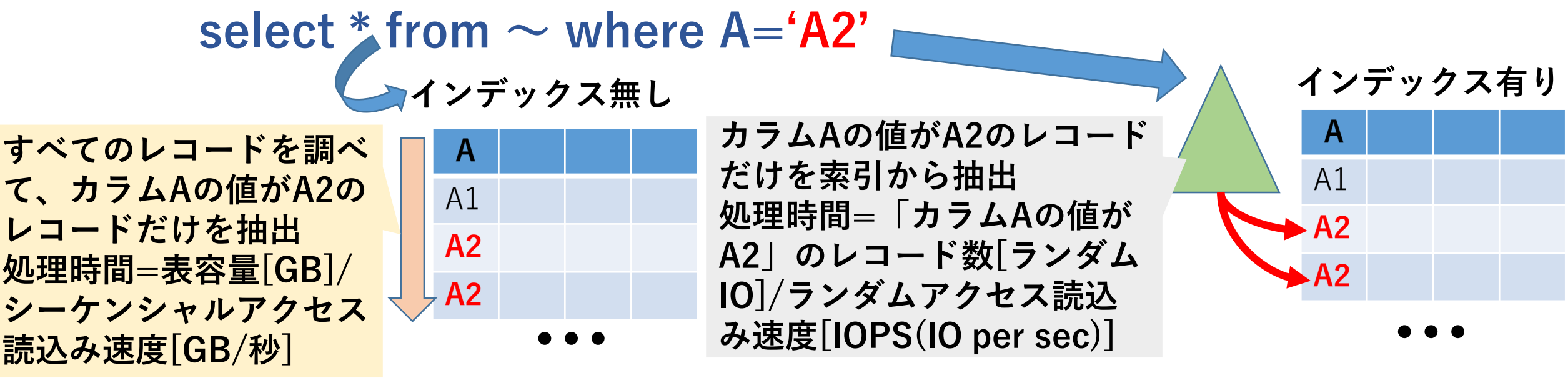
表スキャン

索引アクセス



インデックス(索引)のアクセス時間

クエリによっては、表スキャンが索引を使うよりも処理時間が小さい場合があるので、処理コストを見積って表アクセスの方法を決定する。



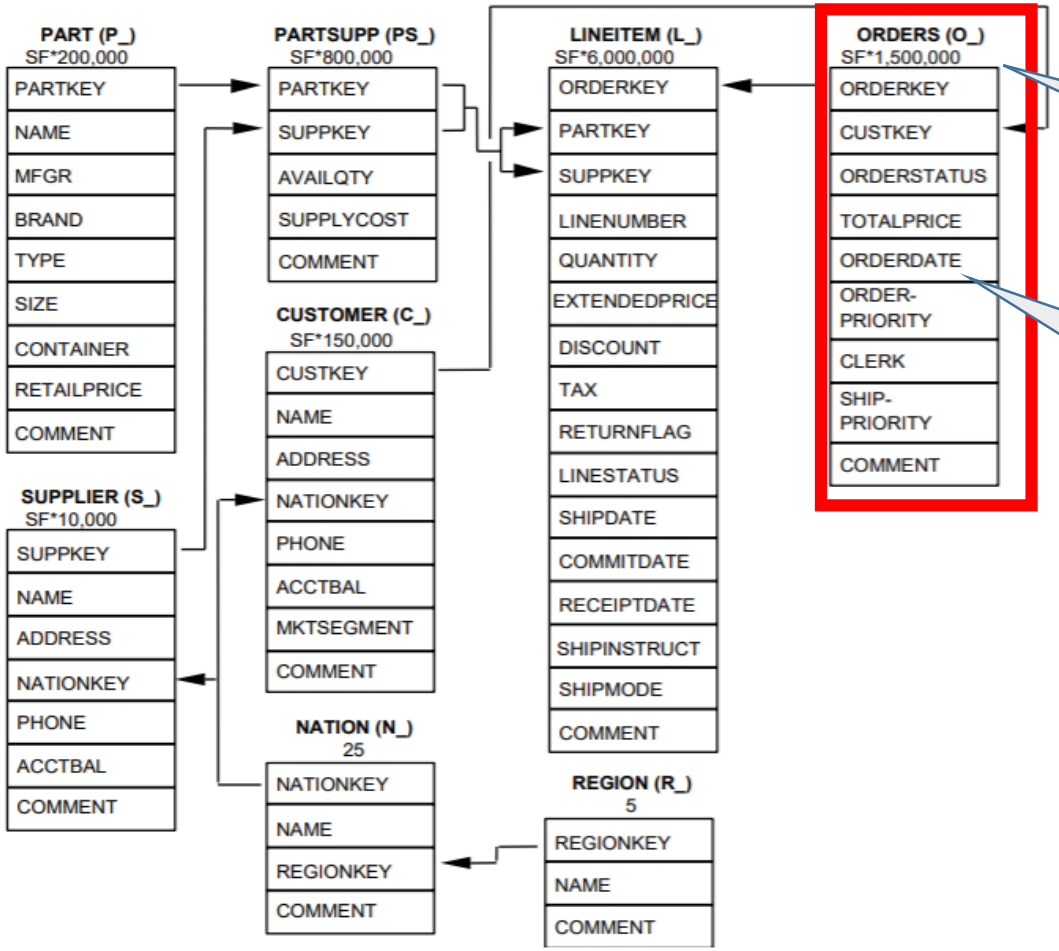
一般に「全件をシーケンシャルアクセスの時間」<<「全件をランダムアクセスの時間」であり、索引が早いかはアクセスレコード数の割合による

課題6-1
締切:11/23

下記データに対して、SF=1 の時に a)～c)の SQL に対する最短処理時間を記載せよ。小数点一桁まで記載すること。各表のレコード数は表名下記の数式(例えば、LINEITEM なら SF*6,000,000[行])で定まるとし、各テーブルは、カラム数やカラム型に関係なく 1,000,000[行]で 1[GB]とする。シーケンシャル READ は 0.5[GB/s]とし、索引アクセスには 1 レコード毎に 0.0001[s]かかるとする。なお、読込みディスク I/O 以外の処理(例えば、CPU 処理)は無視できるものとする。(4 問×5 点, SQL 問題:10 点)

a) select * from orders;で表スキャンを前提とした処理時間

b) select * from orders;で索引アクセスを前提とした処理時間(全件を 1 レコード毎に索引アクセス)と算出式を記載せよ。



ORDERS表はSF1で1,500,000行
→1,500,000[行]/1,000,000[行]=1.5[GB]

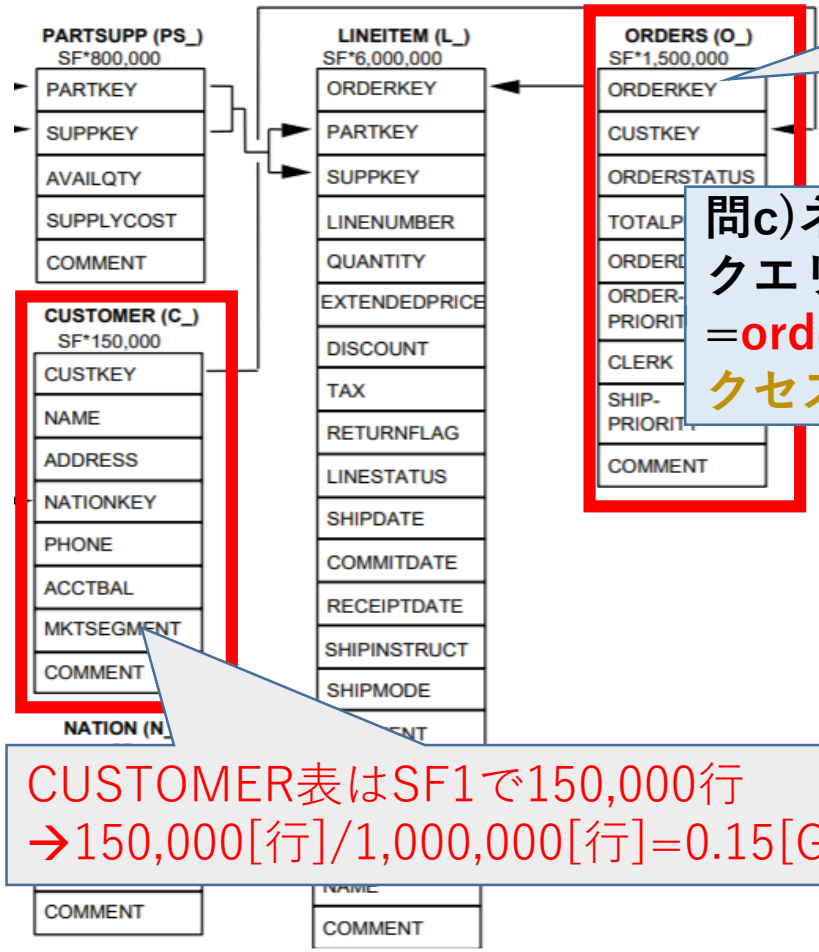
問a)1.5[GB]を0.5[GB/s]で表スキャンすると何秒か？
問b)1,500,000[行]を1レコード毎に0.0001[s]かけて索引アクセスすると何秒か？

課題6-2
締切:11/23

下記データに対して、SF=1 の時に a)～c)の SQL に対する最短処理時間を記載せよ。小数点一桁まで記載すること。各表のレコード数は表名下記の数式(例えば、LINEITEM なら SF*6,000,000[行])で定まるとし、各テーブルは、カラム数やカラム型に関係なく 1,000,000[行]で 1[GB]とする。シーケンシャル READ は 0.5[GB/s]とし、索引アクセスには 1 レコード毎に 0.0001[s]かかるとする。なお、読込みディスク I/O 以外の処理(例えば、CPU 処理)は無視できるものとする。(4 問×5 点, SQL 問題:10 点)

orders 表と customer 表にそれぞれ custkey 索引が備わっており、orders 表では 1 つの custkey に対して 10 レコード存在するときに、以下 SQL の処理時間と算出式を記載せよ。

c) select * from orders, customer where orders.custkey=customer.custkey 外表(駆動表)が orders 表



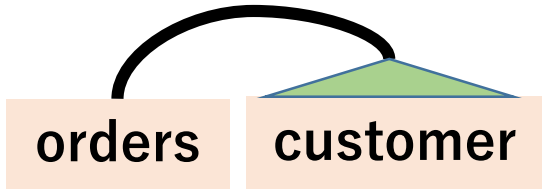
ORDERS表はSF1で1,500,000行
→1,500,000[行]/1,000,000[行]=1.5[GB]

絞込み(where句フィルタ)の指定は無いので全件処理対象

問c)ネストループ結合で、外部表がorders表なので
クエリ処理時間=外部表読込み時間+(外部表レコード×絞込み)×内部表読込み時間
=orders表の最短表読込み時間+(orders表のレコード数×1)×customer表の最短アクセス時間

1)ORDERS表の表全体の読込みは表スキャンが早い?索引アクセスが早い?
2)customer表の1レコードアクセスは表スキャンが早い?索引が早い?

CUSTOMER表はSF1で150,000行
→150,000[行]/1,000,000[行]=0.15[GB]

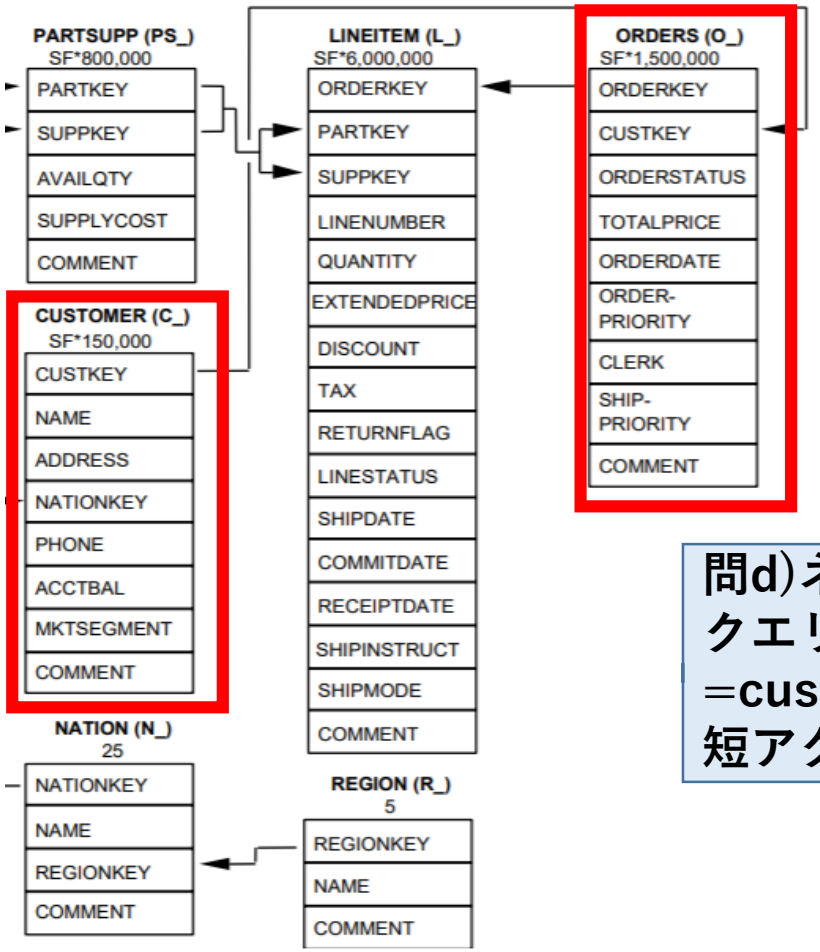


課題6-3
(余裕あれば)
締切:11/23

下記データに対して、SF=1 の時に a)～c)の SQL に対する最短処理時間を記載せよ。小数点一桁まで記載すること。各表のレコード数は表名下記の数式(例えば、LINEITEM なら SF*6,000,000[行])で定まるとし、各テーブルは、カラム数やカラム型に関係なく 1,000,000[行]で 1[GB]とする。シーケンシャル READ は 0.5[GB/s]とし、索引アクセスには 1 レコード毎に 0.0001[s]かかるとする。なお、読込みディスク I/O 以外の処理(例えば、CPU 処理)は無視できるものとする。(4 問×5 点, SQL 問題:10 点)

orders 表と customer 表にそれぞれ custkey 索引が備わっており、orders 表では 1 つの custkey に対して 10 レコード存在するときに、以下 SQL の処理時間と算出式を記載せよ。

d)select * from orders,customer where orders.custkey=customer.custkey 外表(駆動表)が customer 表



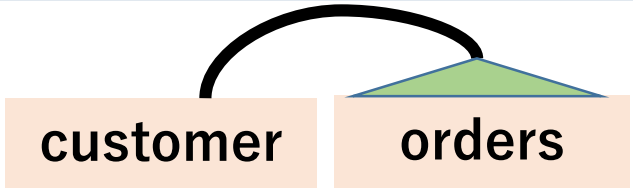
orders表のcustkey索引では、custkey1つに10レコード存在
→1つのcustkeyに対して「orders表のレコード10件」を読込む必要
→orders表の索引アクセスでは、アクセスするレコード数が10倍

質問番号	顧客番号
Q01	C01
Q02	C01
Q03	C02

注文番号	顧客番号	商品番号
001	C01	A01
002	C01	A02
003	C03	A01

例)1つの顧客番号に2つの注文番号
→Q01,Q02のそれぞれで「2件のC01」を読込む

問d)ネストループ結合で、外部表がcustomer表なので
クエリ処理時間=外部表読込み時間+(外部表レコード×絞込み)×内部表読込み時間
=customer表の最短表読込み時間+(customer表のレコード数×1)×orders表の最短アクセス時間



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

SQL:order by句

過去の講義

・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;
```

並べ替え

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

SQL:group by句

過去の講義

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

group byとorder byの組合せ

過去の講義

```
1  create table person_sales(year int not null,
2                    emp_id int not null,
3                    branch varchar(10),
4                   sale int,
5                   primary key (year,emp_id));
6  insert into person_sales values(2010,1,'支店1',50);
7  insert into person_sales values(2010,2,'支店1',21);
8  insert into person_sales values(2010,3,'支店1',30);
9  insert into person_sales values(2010,4,'支店2',21);
10 insert into person_sales values(2011,1,'支店1',60);
11 insert into person_sales values(2011,2,'支店1',41);
12 insert into person_sales values(2011,3,'支店1',20);
13 insert into person_sales values(2011,4,'支店2',31);
14
15 select branch, year,sum(sale) from person_sales group by branch,year order by sum(sale) desc;
```

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

branch	year	sum(sale)
支店1	2011	121
支店1	2010	101
支店2	2011	31
支店2	2010	21

課題6 締切:11/23

6-4)表1は、ある牛丼屋の注文状況を記録している注文履歴テーブルである。
表1に対して、次のデータを取得するためのSQLを作成してください。

```
create table order_t(o_date datetime,
                order_id int not null,
                detail_id int not null,
                item_name varchar(20),
                category char(1),
                size char(1),
                price int,
                unit int,
                total int, primary key (order_id,detail_id));
insert into order_t values('2022-11-01 10:10:10',1,1,'牛丼','F','S',2,400,800);
insert into order_t values('2022-11-01 10:10:10',1,2,'コーラ','D','S',1,200,200);
insert into order_t values('2022-11-01 10:10:10',1,3,'コーヒー','D','S',1,230,230);
insert into order_t values('2022-11-01 10:10:10',2,1,'牛丼','F','S',1,400,400);
 insert into order_t values('2022-11-01 10:10:10',2,2,'コーヒー','D','S',1,230,230);
select * from order_t;
```

注文日時

一つの注文の中に、複数商品がある可能性

注文番号(主キー)、注文枝番(主キー)

商品名

分類(F:フード、D:ドリンク、O:その他)

サイズ(S:スモール、L:ラージ)

a)注文順かつその明細順に、すべての注文データを取得する。
b)各注文について、ドリンク注文価格の総額を取得する。#注文を構成する全ての注文枝番について、分類Dなら集計
c)商品名毎の売り上げ集計を行い、売上額が大きい順に表示する。

(参考:テスト範囲外)集計値の最大値

「最も総額が高い注文」の注文番号を取得し、その注文を構成する商品名などを表示するSQL

最も総額が高い注文

```
insert into order_t values('2022-11-01 10:10:10',1,1,'牛丼','F','S',2,400,800);
insert into order_t values('2022-11-01 10:10:10',1,2,'コーラ','D','S',1,200,200);
insert into order_t values('2022-11-01 10:10:10',1,3,'コーヒー','D','S',1,230,230);
insert into order_t values('2022-11-01 10:10:10',2,1,'牛丼','F','S',1,400,400);
insert into order_t values('2022-11-01 10:10:10',2,2,'コーヒー','D','S',1,230,230);
```

「注文枝番を集約し、注文毎の総額を集計」した表をagg_tとする

```
WITH agg_t as(select order_id, sum(total) as sum_total from order_t group by order_id)
select * from order_t where order_t.order_id = (select order_id from agg_t where sum_total=(select
max(sum_total) from agg_t));
```

agg_tのsum_totalの最大値を取得

sum_totalの最大値を持つorder_idを取得

o_date	order_id	detail_id	item_name	category	size	price	unit	total
2022-11-01 10:10:10	1	1	牛丼	F	S	2	400	800
2022-11-01 10:10:10	1	2	コーラ	D	S	1	200	200
2022-11-01 10:10:10	1	3	コーヒー	D	S	1	230	230

SQL:副問合せ

過去の講義

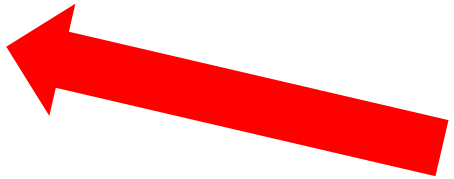
・select文中にselect文を記述する(入れ子にする)ことができ、入れ子の内側の問合せを副問合せ、外側の問合せを主問合せと呼ばれる。
・副問合せを用いたselect文では、まず内側の副問合せを実行して値を返し、その値を主問合せで受けて最終的な結果を生成する。
・副問合せの結果を主問合せに連携するために比較演算子(=,<,>など)およびin,exist,any,some,all演算子を使う。

『「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

副問合せ

『「一回の注文額が最も大きい注文」の商品名』など、テーブルのデータによりSQL処理の内容が異なる場合があり、副問合せを用いて対応する。
#「一回の注文額が最も大きい注文」は格納データ毎に異なる

副問合せを使わずに、2つのSQLの結果を人力で下記のように組み合わせて結果出力を得ることも非効率ではあるが可能

注文テーブル

注文番号	顧客番号	商品番号	商品名	個数	注文額
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A4	3	6000

手順1)「一回の注文額が最も大きい注文」の注文額は8000

select max(注文額) from 注文テーブル

手順2)「注文額が8000の注文」の商品名はオフィス用紙A3

select 商品名 from 注文テーブル where 注文額=8000

副問合せ2(結果一行の副問合せ)

副問合せでは「select文の中にselect文」を記載することで、複数のSQL出力を人力で組合せなくても、1つのSQLで必要な結果を得ることができる。

注文テーブル

注文番号	顧客番号	商品番号	商品名	個数	注文額
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A4	3	6000

2)select 商品名 from 注文テーブル where 注文額=8000

1)select 商品名 from 注文テーブル where 注文額
=(select max(注文額) from 注文テーブル)

1)select文の中にselect文がある場合、「入れ子の内側にあるselect文(注文額の最大を求めるselect文)」から処理が実行される。
2)入れ子の内側のselect文の実行結果が得られれば、実行結果の値がselect文と入れ替わって外側のselect文処理が行われる。

in演算子

in演算子を用いると複数の値と等しいかを判別できる。副問合せでよく使われる演算子であるが、副問合せでないSQLでも用いることができる。

```
select 顧客番号,商品名,注文額 from 注文 where 注文番号 in (‘O01’,’O03’);
```

注文番号=(‘O01’,’O03’)とはできない。
=演算子は1つのデータのみ比較

注文番号	顧客番号	商品番号	商品名	個数	注文額
O01	C01	A01	オフィス用紙A4	1	2000
O02	C01	A02	オフィス用紙A3	2	8000
O03	C03	A01	オフィス用紙A4	3	6000

副問合せ3(副問合せで複数結果)

入れ子の内側のselect文で得られる結果は、1つの値(例:8000)である場合もあるし、複数の値である場合もある。複数の値の比較ではin演算子を使う。

入れ子の内側のselect文で得られるのが、複数の値(例:'O01','O02')の場合は、in演算子を用いる。

注文テーブル

注文番号	顧客番号	商品番号	商品名	個数	注文額
O01	C01	A01	オフィス用紙A4	1	2000
O02	C01	A02	オフィス用紙A3	2	8000
O03	C03	A01	オフィス用紙A4	3	6000

副問合せにしなくても目的の結果は得られますが、例なので。

```
select 商品名 from 注文テーブル where 注文番号 in
(select 注文番号 from 注文テーブル where 注文額>5000)
```

('O01','O02')が実行結果として得られる

SQL:副問合せ(all演算子)

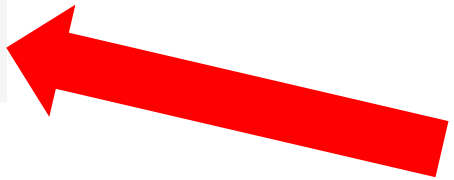
副問合せの結果のすべてが、指定された条件を満たすときにtrueになり、それ以外の場合はfalseになる。

```
select 顧客番号,商品名,総額 from 注文 where 総額 > (select 総額 from 注文 where 商品番号='A01');
```

『「商品番号がA01」の注文の最も大きい総額』以上の総額をもつレコードのみ、顧客番号・商品名・総額を出力

```
select c_id,item_name,total from order_t where total >= all (select total from order_t where item_id ='A01');
```

```
c_id    item_name       total
C01     オフィス用紙A3   8000
C03     オフィス用紙A4   6000
```



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(all演算子)2

all演算子は、min関数などの極値関数でも代用できそうだが、副問合せ結果が0件の場合の挙動が異なる。

```
select c_id,item_name,total from order_t where total >= (select max(total) from order_t where item_id ='A05');
select c_id,item_name,total from order_t where total >= all (select total from order_t where item_id ='A05');
```

all演算子の場合は、全レコードが出力
min関数の場合は、出力レコード無し→入力が空の場合はNULLを返すため

c_id	item_name	total
C01	オフィス用紙A4	2000
C01	オフィス用紙A3	8000
C03	オフィス用紙A4	6000

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:RANK関数

テーブルのカラム値の大小関係で順位を与えることも可能である。

```
create table person_sales(year int not null,
                          emp_id int not null,
                          branch varchar(10),
                         sale int,
                         primary key (year,emp_id));
```

「PARTITION BY カラム名」で、
カラム名毎に順位をつける。

```
select RANK() OVER (PARTITION BY year order by sale desc) AS num,year, emp_id,branch, sale from person_sales where year=2011;
```

2011年の売上順位を付加して出力する
#「順位の値」がカラムnumで出力

num	year	emp_id	branch	sale
1	2011	1	支店1	60
2	2011	2	支店1	41
3	2011	4	支店2	31
4	2011	3	支店1	20

個人売上テーブル

年度	従業員番号	支店	売上
2010	1	支店1	50
2010	2	支店1	21
2010	3	支店1	30
2010	4	支店2	21
2011	1	支店1	60
2011	2	支店1	41
2011	3	支店1	20
2011	4	支店2	30

SQL:RANK関数2

区切り(年度、支店毎)を設定して、大小関係で順位を与えることも可能。

```
create table person_sales(year int not null,
                          emp_id int not null,
                          branch varchar(10),
                         sale int,
                         primary key (year,emp_id));
```

```
select RANK() OVER (PARTITION BY year,branch order by sale desc) AS num,year, emp_id,branch, sale from person_sales;
```

年度、支店毎の売上順位を付加して出力する

num	year	emp_id	branch	sale
1	2010	1	支店1	50
2	2010	3	支店1	30
3	2010	2	支店1	21
1	2010	4	支店2	21
1	2011	1	支店1	60
2	2011	2	支店1	41
3	2011	3	支店1	20
1	2011	4	支店2	31

個人売上テーブル

年度	従業員番号	支店	売上
2010	1	支店1	50
2010	2	支店1	21
2010	3	支店1	30
2010	4	支店2	21
2011	1	支店1	60
2011	2	支店1	41
2011	3	支店1	20
2011	4	支店2	30

課題6 締切:11/23

余裕があれば6-5)RANK関数を用いたSQL(表定義とクエリ)を作成せよ。下記のSQLを用いても良いし、用いなくてもよい。

```
create table person_sales(year int not null,
                emp_id int not null,
                branch varchar(10),
               sale int,
               primary key (year,emp_id));
insert into person_sales values(2010,1,'支店1',50);
insert into person_sales values(2010,2,'支店1',21);
insert into person_sales values(2010,3,'支店1',30);
insert into person_sales values(2010,4,'支店2',21);
insert into person_sales values(2011,1,'支店1',60);
insert into person_sales values(2011,2,'支店1',41);
insert into person_sales values(2011,3,'支店1',20);
insert into person_sales values(2011,4,'支店2',31);
select RANK() OVER (PARTITION BY year,branch order by sale desc) AS num,year, emp_id,branch, sale from
person_sales;
```