

# データベース

## 第10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)	来週以降に答案返却と、(SQL以外の)問題解説をおこないます。 #SQLの問題の解説は今週実施
	6週	RDBMSの内部構成	
	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	データモデリングの中でデータベース化したい範囲を決定し、データモデリング作業で
	12週	SQL(3)	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。

データモデリングとSQLをやります

SQしは表にどのようなコードが入っている。  
正しい結果がどうか正しく書いて下さい。  
問題別に記載の表でだけ正しい結果が  
どうかSQしは不正解です。

問3 (c) L表, O表は表スキーム前提。  
→ 索引は使わない

問1 (a) 入来子のループを使う系論であり、

結合を開き出すコードが格納されている表へ

～ もう一方の表(表の呼び名は②)と結合処理を試みる。

問2 (a) ユーザ数は重複排除する。

→ 同じユーザIDで複数回入室にも1回として数える。

(b) すべてのユーザの入室回数の総数

→ 同じユーザIDで複数回入室は、複数回の入室として数える。

(c) 「所属コード, 入室回数の総数」を表示する。

問4

(e) 2冊以上借りていない、貸出の貸出返却枚数が2シート以上存在する。

貸出では処理コードが1になる。返却は処理コードが2になる。

問2 以下は、ある企業での入退室管理システムを構成するデータベースの表である。入退室するたびに、入退室履歴テーブルに入退室レコードが格納される。なお、入室を表すレコードの入退室コードは”In”となっているとする。ユーザ情報は、ユーザ情報テーブルに格納されているとする。表のカラム型はすべて varchar(100)とする。以下の a)～e)に該当する SQL を解答欄に記載せよ。SQL 内の表名やカラム名は表中の日本語をもちいること。 7点×5問(35点)

- a)過去に一度でも入室したことのあるユーザ数を求める SQL を記載せよ。なお、ユーザ数は重複排除することとする。
- b)すべてのユーザの入室回数の総数を求める SQL を記載せよ。なお、下記の例では入室回数 2 となる。
- c)過去の入室回数の総数をユーザ ID 毎に求める SQL を記載せよ。
- d)「過去の入室回数の総数が 2 回以上」の「ユーザ ID とその入室回数」を求める SQL を記載せよ。

入退室履歴テーブル				ユーザ情報テーブル			
入退室ID	入退室コード	入退室時刻	ユーザID	ユーザID	ユーザ名	所属コード	所属名
001	In	9:00	U01	U01	魚住 一郎	K1	開発一部
002	In	9:10	U02	U02	西岡 次郎	K2	開発二部
003	Out	9:35	U01	U03	兵庫 三郎	S	総務

NTid	NTcode	NTTime	UID
001	In	9:00	U01
002	In	9:10	U02
003	Out	9:35	U01

count(distinct UID)	a)
2	
count(UID)	b)
2	
UID	count(*)
U01	1
U02	1

d)は該当レコードが存在しないので出力無し

入退室履歴テーブル:Nyutai  
ユーザID:UID  
入退室コード:NTcode

```
select * from Nyutai;  
select count(distinct UID) from Nyutai;  
select count(UID) from Nyutai where NTcode="In";  
select UID, count(*) from Nyutai where NTcode="In" group by UID;  
select UID, count(*) from Nyutai where NTcode="In" group by UID having count(*)>=2;
```

a)ユーザIDでdistinctしてcount処理

b)入退室コードがInのレコードをcount処理

c)入退室コードがInのレコードをcount処理して、ユーザIDでgroup by

d)group byの結果をhavingでフィルタ



問2 以下は、ある企業での入退室管理システムを構成するデータベースの表である。入退室するたびに、入退室履歴テーブルに入退室レコードが格納される。なお、入室を表すレコードの入退室コードは”In”となっているとする。ユーザ情報は、ユーザ情報テーブルに格納されているとする。表のカラム型はすべて varchar(100)とする。以下の a)～e)に該当する SQL を解答欄に記載せよ。SQL 内の表名やカラム名は表中の日本語をもちいること。 7点×5問(35点)

e)過去の入室回数の総数を所属コード毎に求める SQL を記載せよ。

入退室履歴テーブル				ユーザ情報テーブル			
入退室ID	入退室コード	入退室時刻	ユーザID	ユーザID	ユーザ名	所属コード	所属名
001	In	9:00	U01	U01	魚住 一郎	K1	開発一部
002	In	9:10	U02	U02	西岡 次郎	K2	開発二部
003	Out	9:35	U01	U03	兵庫 三郎	S	総務

```
select Scode, count(*) from Nyutai, User where Nyutai.Uid=User.Uid and Nyutai.NTcode="In" group by Scode;
```

入退室履歴テーブル:Nyutai, ユーザ情報テーブル:User  
ユーザID:UID, 入退室コード:NTcode  
所属コード:Scode

e)入退室履歴テーブルとユーザ情報テーブルを結合させて、入退室コードがInのレコードのみを抽出して、所属コードでGroup by

Scode	count(*)
K1	1
K2	1

e)

e)は所属コードも記載するように、試験中に連絡している。Select Scodeがなければ、1点減点

問 4 以下は、ある図書館における貸出返却管理システムを構成するデータベースの表である。貸し出し返却が行われるたびに、レコードが格納される。なお、貸出返却 ID は小さい方が過去の貸出返却であるとする。表のカラム型はすべて varchar(100)とする。以下の a)～e)に該当する解答を解答欄に記載せよ。7 点×5 問(35 点)#合計 85 点

- a)図書館が管理している書籍の総数を求める SQL を記載せよ。なお、同じ書籍名の書籍が複数ある場合には重複して数えることとする。
- b)書籍分類が数学である書籍を借りたことのあるすべての利用者 ID を取得する SQL を記載せよ。なお、利用者 ID は重複排除すること。
- c)過去に貸し出された書籍について、書籍分類ごとに貸し出された冊数の総数を集計し、貸し出された冊数について降順に表示する SQL を記載せよ。「書籍分類と書籍分類の貸し出しの総数」を表示すること。

貸出返却履歴テーブル

貸出返却ID	貸出返却枝番ID	処理コード	日付	書籍ID	利用者ID
K01	E01	K	2022-12-06	S01	U01
K01	E02	K	2022-12-06	S02	U01
K02	E01	K	2022-12-07	S03	U02
H03	E01	H	2022-12-08	S01	U01

書籍情報テーブル

書籍ID	書籍名	書籍分類
S01	Book1	工学
S02	Book2	数学
S03	Book3	歴史
S04	Book4	経済

Bid	Bname	Bcode			
S01	B01	数学			
S02	B02	数学			
S03	B03	歴史			
S04	B04	経済			
count(*)					
4					
KHid	KHid_sub	Kcode	hizu	Bid	Uid
K01	E01	K	2022-12-06	S01	U01
K01	E02	K	2022-12-06	S02	U01
K02	E01	K	2022-12-07	S03	U02
H03	E01	H	2022-12-08	S01	U01
Uid					
U01					
Bcode	count(*)				
数学	2				
歴史	1				

書籍情報テーブル:Sho, 貸出返却履歴テーブル:KH  
書籍ID:Bid, 処理コード:Kcode  
書籍分類:Bcode

a)書籍情報テーブルをcount処理

b)書籍情報テーブルと貸出返却履歴テーブルを結合し、書籍分類が数学のみ抽出

c)表結合し、処理コードKのみ抽出して、書籍分類でgroup by

降順はdesc

```
select count(*) from Sho;
select * from KH;
select distinct Uid from KH,Sho where KH.Bid=Sho.Bid and Bcode="数学";
select Bcode, count(*) from KH,Sho where KH.Bid=Sho.Bid and Kcode="K" group by Bcode order by count(*) desc;
```

select Uid from KH where KH.Bid in (select KH.Bid from KH,Sho where Kcode='K' and Bcode='数学'and KH.Bid=Sho.Bid) group by Uid; #bはこれも正答



問 4 以下は、ある図書館における貸出返却管理システムを構成するデータベースの表である。貸し出し返却が行われるたびに、レコードが格納される。なお、貸出返却 ID は小さい方が過去の貸出返却であるとする。表のカラム型はすべて varchar(100)とする。以下の a)～e)に該当する解答を解答欄に記載せよ。7 点×5 問(35 点)#合計 85 点

d)「貸出返却 ID が K04 である貸出」について、その貸出で貸し出されたすべての書籍 ID を取得する SQL を記載せよ。以下の表には K04 は無いが、K04 が存在した場合に取得するような SQL を記載すること。なお、同一の貸出は複数書籍の貸出で構成されている可能性があり、その場合には同じ貸出返却 ID に対して、複数の貸出返却枝番 ID のレコードが存在することになる。例えば、下図では K01 の貸出は E01 と E02 の枝番 ID から構成されている。

e)「これまでに 2 冊以上本を借りているすべての利用者」について、一番最初に借りた貸出返却 ID を取得せよ。なお、2 冊以上借りているとは、貸出の貸出返却枝番が 5 レコード以上存在することを意味する。

Bid	Bname	Bcode
S01	B01	数学
S02	B02	数学
S03	B03	歴史
S04	B04	経済

KHid	KHid_sub	Kcode	hizu	Bid	Uid
K01	E01	K	2022-12-06	S01	U01
K01	E02	K	2022-12-06	S02	U01
K02	E01	K	2022-12-07	S03	U02
H03	E01	H	2022-12-08	S01	U01

貸出返却履歴テーブル

貸出返却ID	貸出返却枝番ID	処理コード	日付	書籍ID	利用者ID
K01	E01	K	2022-12-06	S01	U01
K01	E02	K	2022-12-06	S02	U01
K02	E01	K	2022-12-07	S03	U02
H03	E01	H	2022-12-08	S01	U01

試験中に2レコード以上と連絡

書籍情報テーブル:Sho, 貸出返却履歴テーブル:KH  
書籍ID:Bid, 処理コード:Kcode  
書籍分類:Bcode, 貸出返却ID:KHid,貸出返却枝番ID:KHid\_sub

書籍情報テーブル

書籍ID	書籍名	書籍分類
S01	Book1	工学
S02	Book2	数学
S03	Book3	歴史
S04	Book4	経済

d)は該当レコードが存在しないので出力無し

d)KhidがK04であるレコードを抽出し、書籍IDを表示

Uid	min(KHid)
U01	K01

e)

```
select KHid, KHid_sub, Bid from KH where KHid='K04';
```

「取得したUid」と合致するUidを持つレコードで、KcodeがKのものをUidでGroup byし、最小のKHidを表示

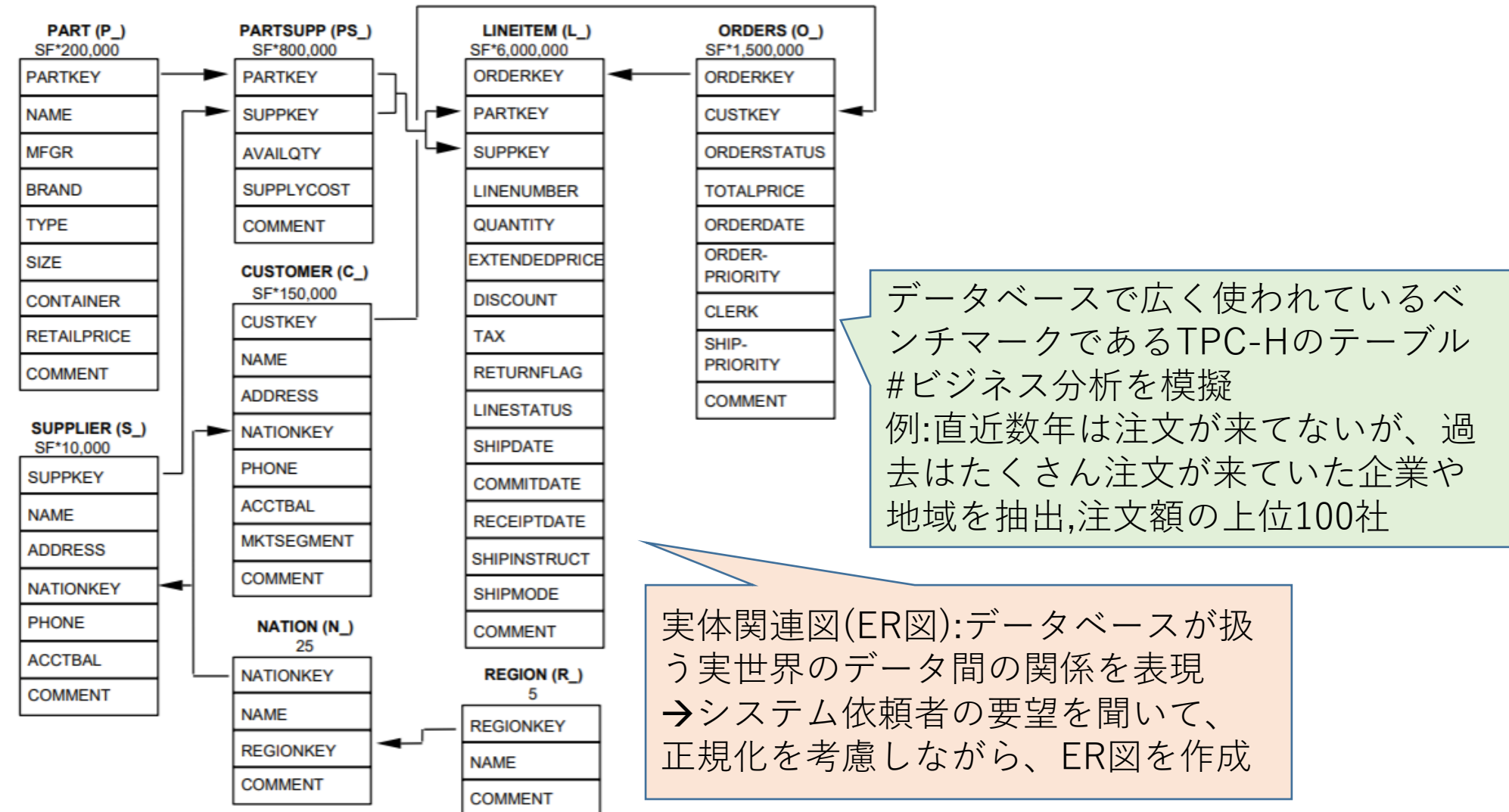
```
select Uid, min(KHid) from KH where Kcode="K" and KH.Uid in (select Uid from KH where Kcode="K" group by Uid having count(*)>=2) group by Uid;
```

e)

KcodeがKのレコードを抽出し、Uidでgroup byした後のcount処理結果が2以上のレコードのUidを取得

# 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からデータ項目を抽出・整理して、データベースの適切な構造を決定することを、データモデリングという。





# データベース設計

以下の手順により、一般的にデータベースの設計を行う

概念設計: システム構築対象となる実世界からデータ項目を抽出

データ間の関係を整理

データモデルに依存しない形でモデル化(ER図が一般的)

→ **例: 正規化する**

論理設計: 概念モデルを、対象とするデータモデルに適合させ、

データベースが十分な性能を発揮するようにデータ構造を調整

→ **例: 関係DBならリレーションの集合にデータを変形させ、索引や制約を設定**

**関係DBにするか、他(例: ドキュメント指向DB)にするか決める**

**どんな索引が必要か考える**

物理設計: システムに要求される性能要件を満たすように、

ハードディスクを含めたハードウェア構成や、

各リレーションのデータを記憶させる記憶装置・位置の割り当て

使用するDBMSの選定

→ **例: 索引や表の2次記憶上の位置を考える。**

**#大容量の表は、高スループットの領域に置くなど**

# データモデリングの方法1

例題:ある大学の履修管理システムの構築

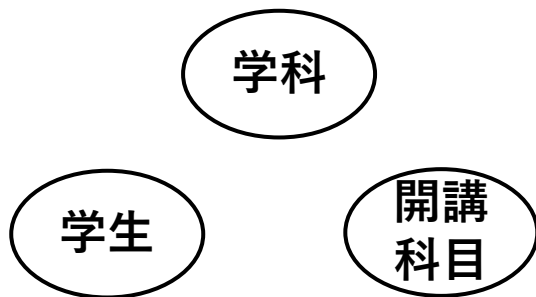
要求1:履修管理システムは大学のすべての学生と開講科目を管理し、どの学生がどの科目をどの学期に履修したかを検索表示できる。

要求2:履修管理システムは学生の履修履歴を管理し、学生ごと、科目ごとに検索表示できる。

手順1:実体の抽出

実体とは、実世界のデータをモデル化する際のデータの単位であり、リレーショナルデータベースではリレーションに対応する。実世界での物理的な実体を伴うもの(例:学生、学科、開講科目)に着目することを考える。

#学生個人名や学科名などの固有名詞ではなく、総称を挙げる



# データモデリングの方法2

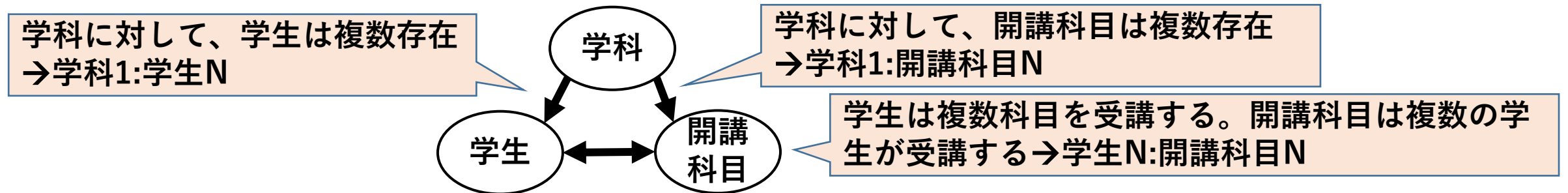
例題:ある大学の履修管理システムの構築

手順2:実体間の関連の設定

2つの実体間の関連を、リレーショナルデータベースでは外部キーで表現する。関連は2つの実体がどのような関係にあるかを考えると設定しやすい。

A):「学生」と「学科」は「所属する・所属される」の関係にある。1学生は必ず1学科に所属し、1学科には複数の学生が所属するとすると、学科と学生は「1対多」の関連になる。

B):「学生」と「開講科目」は「受講する・される」の関係にある。1学生は複数の開講科目を受講し、1開講科目は複数の学生に受講されるとすると、学生と開講科目の関連は「多対多」である。





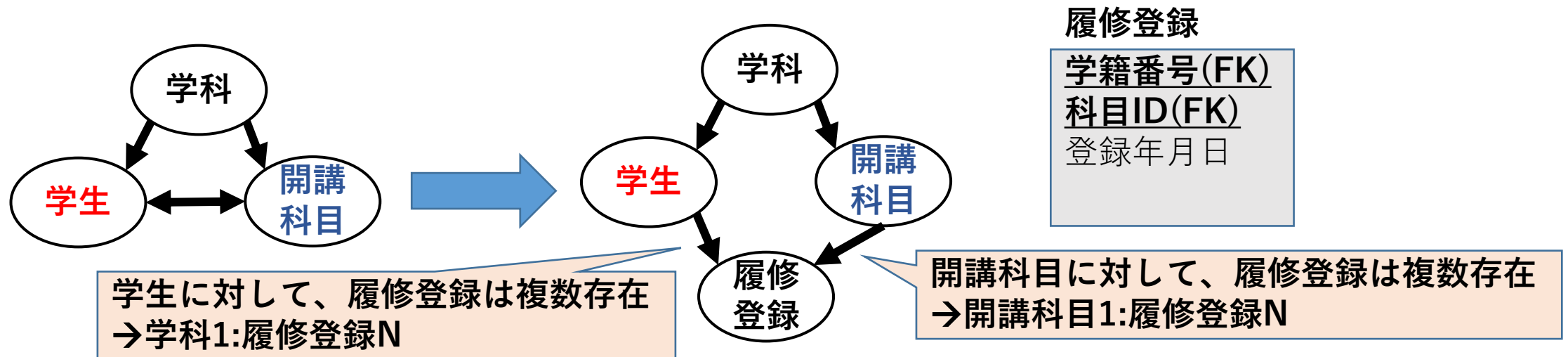
# データモデリングの方法3

例題:ある大学の履修管理システムの構築

手順3:多対多の関連の分割

実体間の関連に「多対多」がある場合には、実体の間に新たな実体を加えて複数の「1対多」に分解する。

例:「**学生**」と「**開講科目**」の間に「履修登録」を追加する。「学生」と「履修登録(ある学生がある科目を履修という情報)」の関連は「1対多」であり、「履修登録」と「開講科目」も同様に「1対多」である。

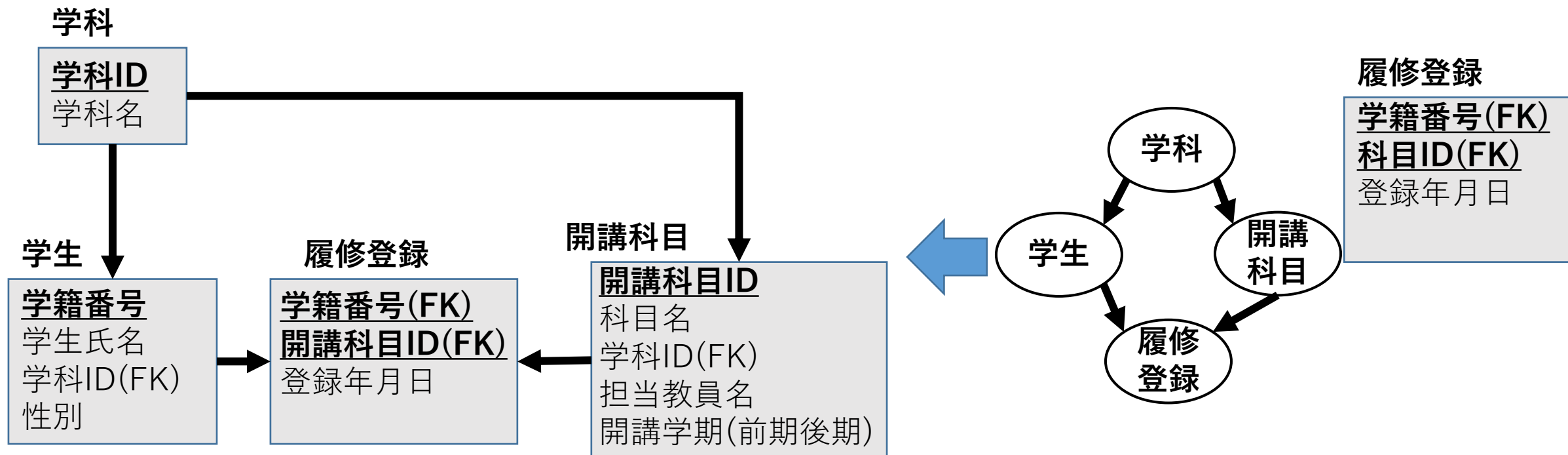


# データモデリングの方法4

例題:ある大学の履修管理システムの構築

手順4:キーと属性の設定

各実体に対し、キーと属性を設定する。キーはその実体を一意に識別するためのデータ項目であり、リレーショナルデータベースでは主キーとなる。  
例:「開講科目」に対し「開講科目ID」、「学生」に対し「学籍番号」等



# データモデリングの方法5

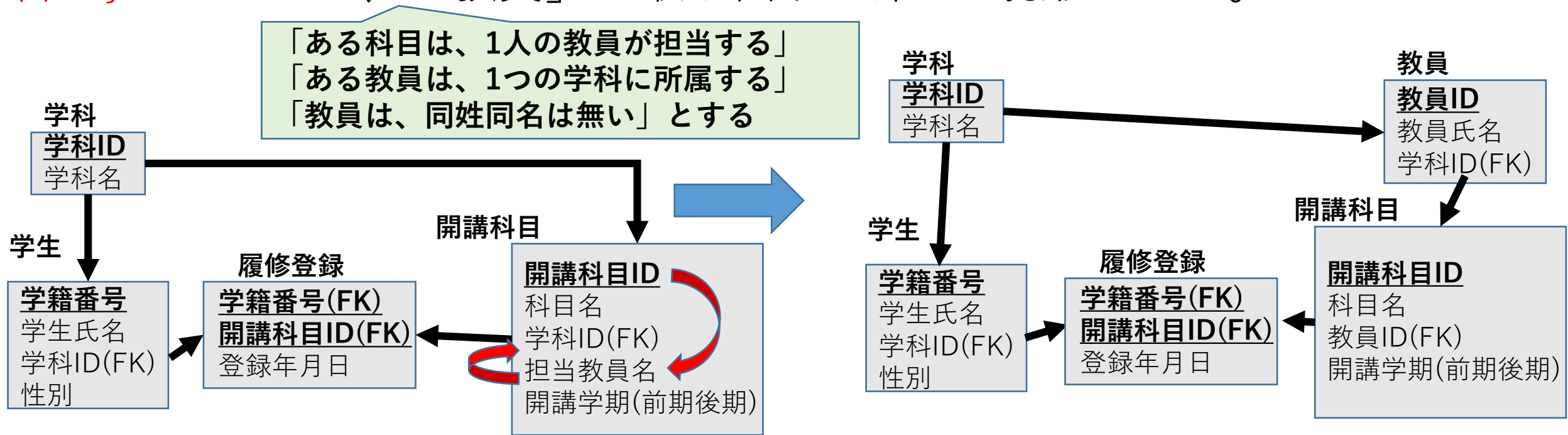
例題:ある大学の履修管理システムの構築

手順5:正規化

第3正規形まで正規化するのが一般的である。

#理由の一つとして、正規化を進めると表分割されていくが、データを復元するためには多数の(処理時間が大きい)表結合(JOIN)をする必要があるため。

例:「開講科目」の中に推移的関数従属性{開講科目ID}→{担当教員名}→{学科ID}があるので、「教員」を取り出すと第3正規形となる。





# データモデリングの例(眼鏡店の販売管理)

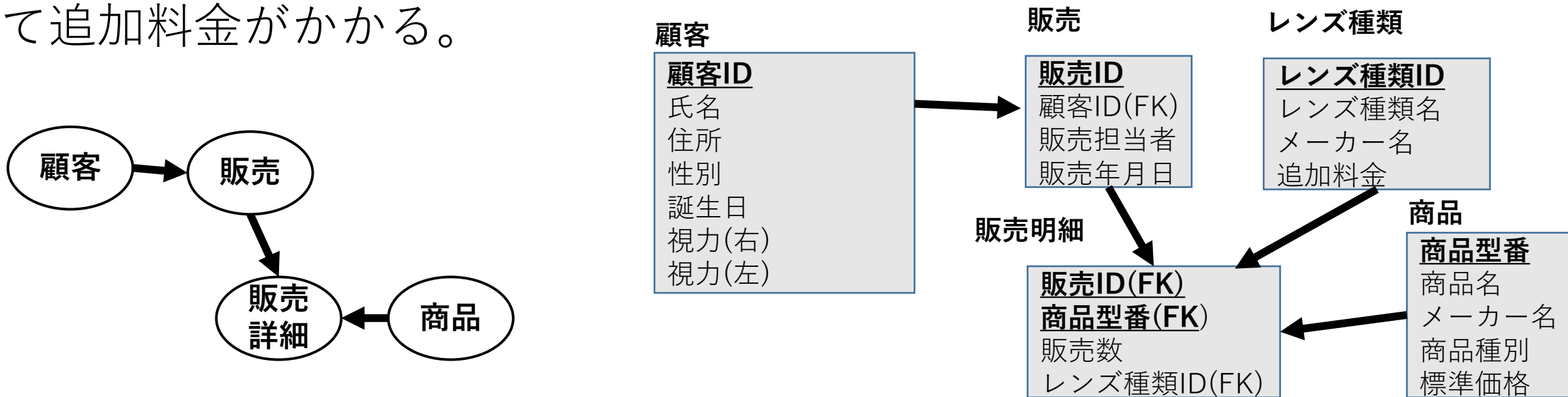
例題:眼鏡店の顧客管理と販売管理のシステム

要求(1):眼鏡と眼鏡用品を販売する。販売する商品は複数の製造メーカーから仕入れており、商品型番によって一意に決まる。

要求(2):購入客は、登録用紙に氏名、住所、性別、誕生日を記入し顧客登録

要求(3):顧客単位で購入履歴を管理。購入履歴や過去視力結果を参照したい

要求(4):眼鏡はフレームごとに値段が決まり、使用するレンズの種類によって追加料金がかかる。



# 課題10 締切:1/4

10-1)図書館の書籍貸出業務を支援するデータベースシステムを構築したい。

以下の要求を満たす実体関連図を作成せよ。

要求(1):図書館の利用者は、事前に利用者登録を行えば、受付で書籍を借りることができる。

要求(2):利用者は、書籍を最大2週間の期間借りることができる。ただし、同時に借りられる書籍は10冊以内とする。

要求(3):利用者の要望や、書籍の破損状態に応じ、書籍入荷や廃棄を行う。

要求(4):利用者は図書館内の端末で蔵書検索を行い、書籍情報と保管されている書架を知ることができる。

10-2)以下を抽出するSQLを記載せよ。なお、表名や属性名は自ら考案すること。

a)「2週間を超えて貸し出されている本」

b)「累積の貸出本数の多い利用者」

```
1 create table kashidashi(  
2 kashidashi_id char(5) not null,  
3 user_id char(5) not null,  
4 kashidashi_date date,  
5 primary key (kashidashi_id));  
6  
7 insert into kashidashi values('K01','U01','2021-12-01');  
8 insert into kashidashi values('K02','U01','2021-12-21');  
9  
10 select * from kashidashi where kashidashi_date >= (NOW() - INTERVAL 14 day);  
11 select * from kashidashi where kashidashi_date < (NOW() - INTERVAL 14 day);
```

日付型はdate

'YYYY-MM-DD'

貸し出し日 >= 今日-14日  
#2週間以内の貸し出し

貸し出し日 < 今日-14日  
#2週間超過の貸し出し

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

kashidashi_id	user_id	kashidashi_date
K02	U01	2021-12-21
kashidashi_id	user_id	kashidashi_date
K01	U01	2021-12-01



# 発展SQL:副問合せ

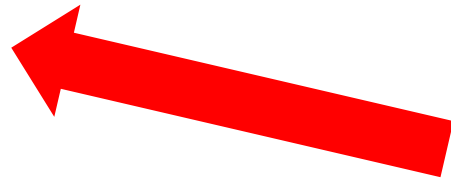
- ・ select文中にselect文を記述する(入れ子にする)ことができ、入れ子の内側の問合せを副問合せ、外側の問合せを主問合せと呼ばれる。
- ・ 副問合せを用いたselect文では、まず内側の副問合せを実行して値を返し、その値を主問合せで受けて最終的な結果を生成する。
- ・ 副問合せの結果を主問合せに連携するために比較演算子(=,<,>など)およびin,exist,any,some,all演算子を使う。

『「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

# 発展SQL:副問合せ

過去の講義

- ・ 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。

『「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

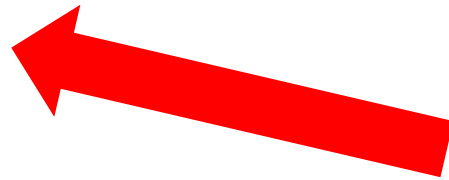
例)比較演算子を使った副問合せ

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

先に、`select avg(total) from order_t`が処理され  
5333.333...と置き換わる。

$\#(2000+8000+6000)/3=5333.333...$

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

# 発展SQL:副問合せ(inとnot in演算子)

過去の講義

副問合せの結果が行の集合(1行だけでも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);
```

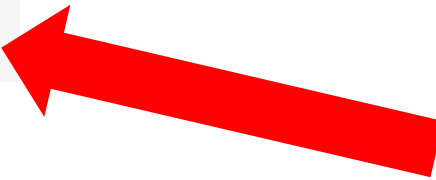
not inと置き換え可能

#処理結果はinとnot in で異なる

『「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



# 発展SQL:副問合せ(inとnot in演算子)

過去の講義

副問合せの結果が行の集合(1行だけでも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。

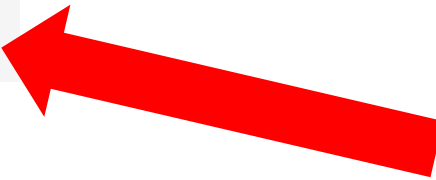
『「総額」が7000以上の顧客番号』についてのみ、顧客番号・商品名・総額を出力

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

先に、select c\_id from order\_t where total>=7000が処理されC01と置き換わる。

#総額7000以上の注文をしているのが顧客C01だけなので

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

# 発展SQL:副問合せ(相関有り)

「入れ子になった問合せ(副問合せ)を外側(主問合せ)から1レコード毎もらいながら処理する」 副問合せを「相関有り副問合せ」という

「商品番号'A01'の商品を注文した顧客」の顧客番号と住所を取得  
#cust.c\_idが主問合せから渡される

```
select c_id,city from cust where 'A01' in (select item_id from order_t where c_id= cust.c_id);
```

A01を購入しているのは、C01とC03

副問合せの中では、order\_t表しかfrom句で宣言されていない。  
→where c\_id=cust.c\_idのcust表は主問合せのfrom句

c_id	city
C01	神戸
C03	加古川

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

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

# 発展SQL:副問合せ(相関有り)2

「商品番号'A01'の商品を注文した顧客」の顧客番号と住所を取得  
#cust.c\_idが主問合せから渡される

```
select c_id,city from cust where 'A01' in (select item_id from order_t where c_id= cust.c_id);
```

処理1)主問合せからcust表が1行毎(例:顧客番号C01)に副問合せに渡される

処理2)副問合せは受け取った行毎に結果を生成する。

→顧客番号C01の注文レコードは注文番号001,002

該当レコードの顧客番号(c\_id),商品番号(item\_id)を保持

処理3)処理2)の「副問合せの生成結果」に対し、in処理して主問合せ側の結果(c\_id,city)をユーザに返す

#処理1)～3)をcust表全件に対して繰り返し行う

c_id	city
C01	神戸
C03	加古川

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

処理1



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

処理2

# 発展SQL:副問合せ(相関有り)3

「商品番号'A01'の商品を注文した顧客」の顧客番号と住所を取得  
#cust.c\_idが主問合せから渡される

```
select c_id,city from cust where 'A01' in (select item_id from order_t where c_id= cust.c_id);
```

処理1)主問合せからcust表が1行毎(例:顧客番号C01)に副問合せに渡される

処理2)副問合せは受け取った行毎に結果を生成する。

→顧客番号C01の注文レコードは注文番号001,002

該当レコードの顧客番号(c\_id),商品番号(item\_id)を保持

処理3)処理2)の「副問合せの生成結果」に対し、in処理して主問合せ側の結果  
(c\_id,city)をユーザに返す

#処理1)~3)をcust表全件に対して繰り返し行う

c_id	city
C01	神戸
C03	加古川

処理3

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

処理2

副問合せ結果  
(C01,A01),(C01,A03)

副問合せ結果  
(C03,A01)



# 課題10 締切:1/25

10-3) 相関有り副問合せを用いた例(表定義とクエリ)を作成せよ。

<https://paiza.io/ja/projects/new>で簡易なMySQLが実行可能である。下記表定義を用いても良いし、用いなくても良い。

```
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 cust(c_id char(3) not null,  
                 c_name varchar(20),  
                 city varchar(20),primary key (c_id));  
insert into order_t values('001','C01','A01','オフィス用紙A4',1,2000);  
insert into order_t values('002','C01','A02','オフィス用紙A3',2,8000);  
insert into order_t values('003','C03','A01','オフィス用紙A4',3,6000);  
insert into cust values('C01','会社A','神戸');  
insert into cust values('C02','会社B','明石');  
insert into cust values('C03','会社C','加古川');  
select c_id,city from cust where 'A01' in (select item_id from order_t where c_id= cust.c_id);
```

# 発展SQL:副問合せ(相関有り)4

相関有り副問合せを使って、時系列データを扱う事も可能である。

```
create table sales(year int not null,  
                  sale int,  
                  primary key (year));
```

主問合せ側から1件レコードが来るので、そのレコードの前年の売り上げを返す

```
select S1.year,S1.sale from sales S1 where sale > (select sale from sales S2 where S2.year=S1.year-1);
```

前年度と比べて、売上が上がった年を抽出する

year	sale
2011	51
2012	52

2011,2012だけが  
前年より売り上げが増えてる

総売り上げテーブル

年度	売上
2010	50
2011	51
2012	52
2013	52
2014	51

# 課題10 締切:1/25

10-4)下記の表定義にて、前年より売り上げが下がった年を抽出するSQLを作成せよ。

```
create table sales(year int not null,  
                  sale int,  
                  primary key (year));  
insert into sales values(2010,50);  
insert into sales values(2011,51);  
insert into sales values(2012,52);  
insert into sales values(2013,52);  
insert into sales values(2014,51);  
select S1.year,S1.sale from sales S1 where sale> (select sale from sales S2 where  
S2.year=S1.year-1);
```

# 発展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

# 発展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

2010の支店1内の順位

2011の支店1内の順位

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10 締切:1/25

10-5)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;
```

