SQLによるデータベース更新操作（Database Update）

(1) 挿入 (insert)

[一般形] insert into <表名> values (<挿入値リスト>|<問合せ指定>)

[例] insert into emp values ('E01', 'Smith', 'D01', 100)

insert into R1 values select X, Y from R2

(2) 削除 (delete)

[一般形] delete from <表名> where <探索条件>

[例] delete from emp where ename = 'Smith'

(3) 修正 (update)

[一般形] update <表名> set <設定> where <探索条件>

[例] update emp set deptno = 'D02' where empno = 'E01'

update emp set salary = salary-10 where deptno = 'D01'

表の定義
（リレーショナルスキーマの作成）

[一般形]

create table <表名> (<列定義>[, <列定義>...])

[例]

```
create table emp
   (empno char(6), ename varchar2(15),
     deptno char(6), salary int)
create table dept (deptno char(6) primary key,   dname
   varchar(15), manager char(6))
```

（参考）

作成した表を削除するにはdrop table、
表を修正するにはalter table
をそれぞれ使用する

リレーションスキーマとインスタンス

- リレーションスキーマ(relation schema):リレーションの骨組みだけの情報。時間に対して不変。太字で表記(例: リレーションスキーマ**R**)
- インスタンス(instance):リレーションの実現値(ある時点での値)。時間によって変化。厳密には時刻をつけて記述。

R(リレーションスキーマ)

商品番号	商品名

R_t(時刻tでのインスタンス)

商品番号	商品名
G1	冷蔵庫
G2	冷蔵庫
G3	テレビ

データの型と記述方法

データ型	説明	データ例
char(サイズ)	固定長の文字データ。サイズ不足分は空白で埋める。サイズは1～2000	'ABC　　'
varchar(サイズ)	可変長の文字データ。サイズは1～4000	'ABC'
number(p,s) integer（int)やfloatも可能	数値データ。精度pと位取りsを指定しない場合は浮動小数点（38けた）	123、123.45
date	日付データ。4けたの年、月、日、時間、分、秒を紀元前4712年1月1日から西暦9999年12月31日まで扱える	'2006-01-15 17:15:30'

キーの種類

- **超キー**(super key):その値だけ決まればタップルを一意に識別できる属性の集合
  リレーションスキーマRの任意のインスタンスRにおいて
  $$(\forall t, t' \in R)(t[K] = t'[K] \Rightarrow t = t')$$
- リレーションはタップルの集合で同じタップルは2個以上ないので、リレーション**R**の**全属性集合Ω_R**={A1, A2, …, An}は常に超キー
- **候補キー**(candidate key):超キーの中で**極小のもの**(これ以上属性を減らすと超キーとならないもの)
- **主キー**(primary key)は候補キーのうちの一つ(候補キーの中でそのリレーションの主要なキーとして適切なもの)

主キーの例

商品

商品番号	商品名
G1	冷蔵庫
G2	テレビ
G3	エアコン

顧客

顧客番号	顧客名
C1	竹中
C2	松田

注文

注文番号	商品番号	顧客番号	個数
O1	G1	C1	1
O2	G2	C1	3
O3	G2	C2	2
O4	G3	C2	1

商品番号、顧客番号、注文番号（属性名に下線）は主キー

SQLでのキーに関連した記述

create table 表名
(列名 ドメイン名 not null, ...) ←空値をとらない

create table 表名
(列名 ドメイン名 unique, ...) ←この属性が同じ値をとるタップルは1つしかない（候補キーに相当）

create table 表名
(列名 ドメイン名 primary key, ...) ←この属性が主キーである

ビュー（外部スキーマ）の定義

create view <ビュー名> as <問合せ指定>
[with check option]

[例] create view high_salary
as select * from emp where salary >= 100
[with check option]

（注意）with check option を付けると、<問合せ指定>の中にある探索条件に合わない、次のような更新操作を受け付けないようにすることができる。

update high_salary set salary = 50
where empno = 'E01'

（参考）3層スキーマ

ユーザ1

応用プログラム1　応用プログラム2

外部スキーマ（ビュー）
・ユーザやプログラマに見せるデータ形式（ユーザやプログラムごとに機密保持などの理由で異なるように設定できる）
・リレーショナルデータモデルでは、SQLのCREATE VIEW 文で作成

概念スキーマ
・実世界に対応してデータを記述したデータ形式
・リレーショナルデータモデルでは、SQLのCREATE TABLE 文で作成

内部スキーマ
・データベースの内容をディスクに格納するときの物理的な形式
・固定長のページ、B木やAVL木の構造、インデックスの作成など

問合せ処理と最適化 (query processing and optimization)

- データベースに対するデータ操作言語(SQLなど)により記述された問合せでは実行手順が示されていない

  →非手続き的問合せ記述

- データ操作言語の処理系が、問合せを解析し、基本データ操作からなる手続き的問合せ記述を生成する

  → これを問合せ実行プラン(または問合せプラン)と呼ぶ

- 一つの問合せに対する問合せ実行プランは何通りも考えられるので、その中で処理時間や必要なメモリ領域が最小なプランを選択する(これを問合せ最適化という)

問合せ実行プラン (query execution plan)

問合せ実行プランの生成は、次の2つのフェーズで行われる

フェーズ1：SQLの問合せを、リレーショナル代数演算子列へ変換

フェーズ2：各リレーショナル代数演算あるいはその組合せのデータ操作を、具体的に実行する基本データ操作列へ変換

フェーズ1：SQLからリレーショナル代数への変換

一般形

select 列リスト from 表リスト where 検索条件

- リレーショナル代数と対応させると次のようになる

例　select R.A from R,S where R.A=S.B

from R, S ・・・・・・・・・R×S（RとSの直積）

where R.A=S.B・・・・$\sigma_{R.A=S.B}$（選択演算）

select R.A ・・・・・・・・$\pi_{R.A}$(射影演算、ただし重複を除かない）

フェーズ1：生成されるリレーショナル演算子列の例

問合せ

```
select distinct emp.empno, emp.ename, emp.deptno, dept.dname
from emp, dept
where emp.deptno = dept.deptno and emp.empno = 'E01‘
```

この問合せに対する処理のフェーズ1で生成され得る演算子列としては、次のようなものが考えられる（以下では、empを**a**、deptを**b**という別名で表現している）

(1) $\pi_{a.empno,\ a.ename,\ a.deptno,\ b.dname}\ (\sigma_{a.deptno = b.deptno\ \wedge\ a.empno = 'E01'}\ (a \times b))$

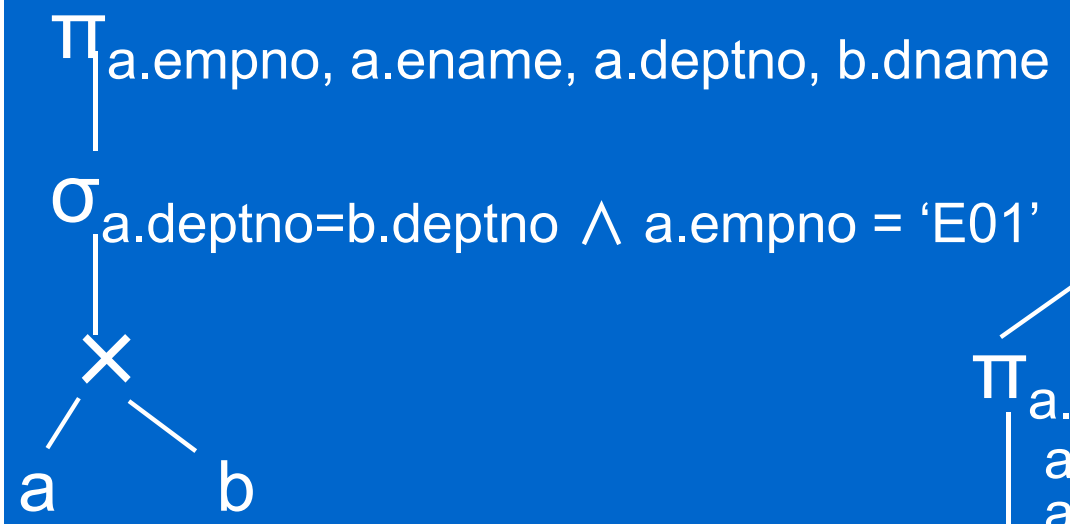
(2) $\pi_{a.empno,\ a.ename,\ a.deptno,\ b.dname}\ (\sigma_{a.empno = 'E01'}\ (a \bowtie_{a.deptno = b.deptno} b))$

(3) $\pi_{a.empno,\ a.ename,\ a.deptno,\ b.dname}\ (\pi_{a.empno,\ a.ename,\ a.deptno}\ (\sigma_{a.empno = 'E01'}\ (a)) \bowtie_{a.deptno = b.deptno} \pi_{b.deptno,\ b.dname}\ (b))$

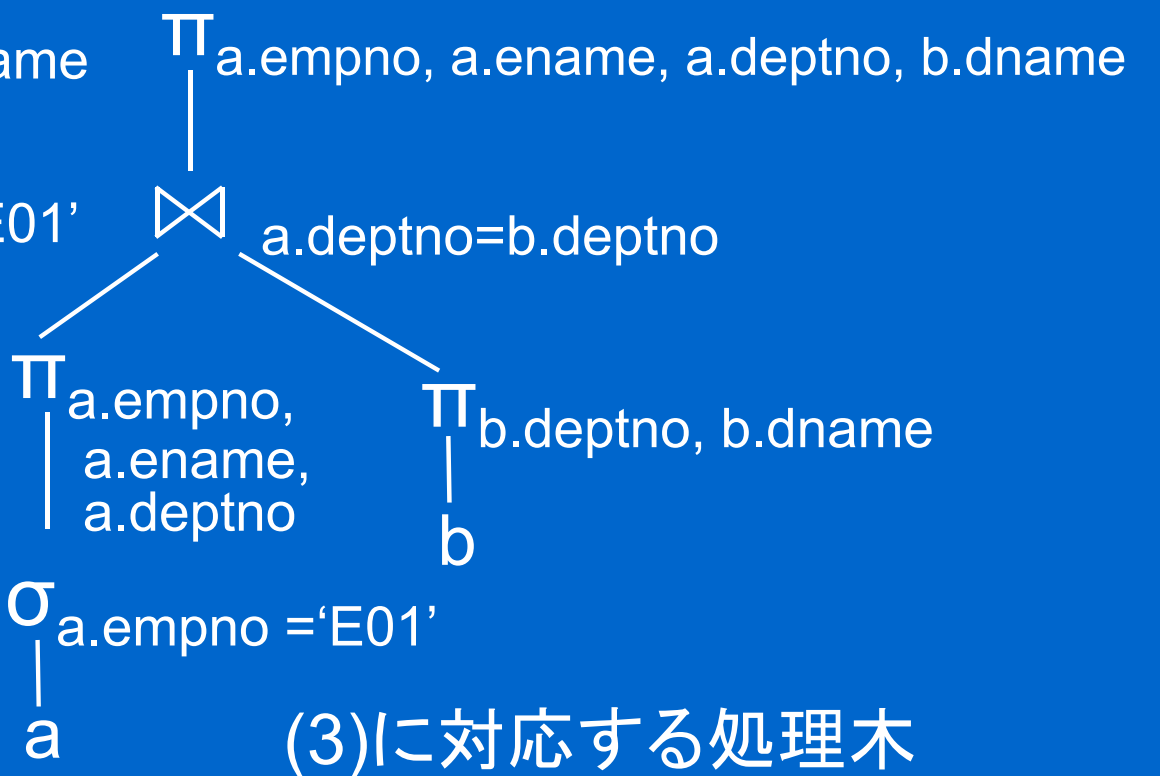
フェーズ2:問合せ実行プランの決定

- フェーズ2では、さらに各リレーショナル代数演算あるいはその組合せのデータ操作をどのような手順で実行するかが決定され、最終的な問合せ実行プランが生成される
- 上記の決定過程では、**処理木**を用いるのが便利である

フェーズ2：問合せ実行プランの例
処理木(processing tree)



(1)に対応する処理木



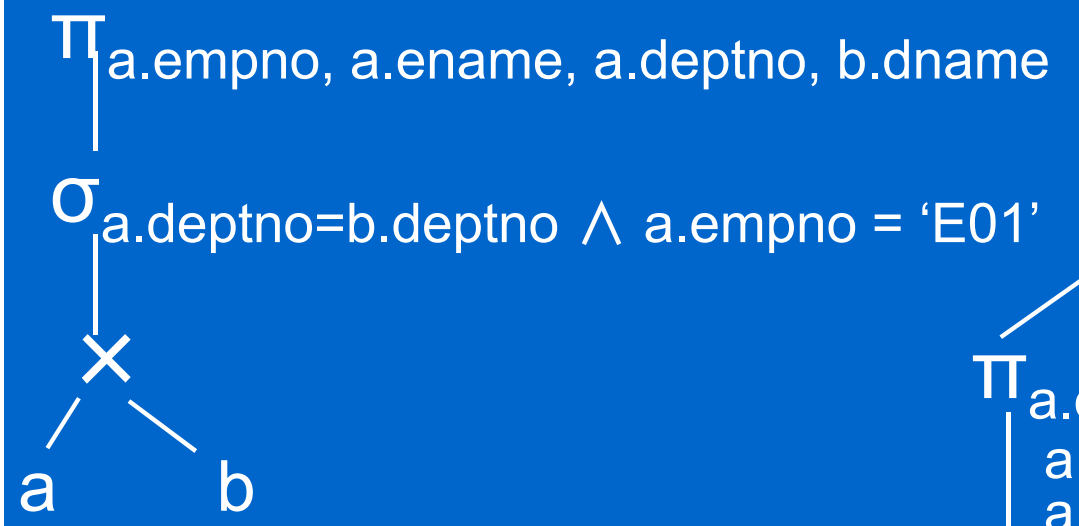
(3)に対応する処理木

どちらの処理木の方が、問合せを効率的に実行できると考えられるか？
その理由は何か？

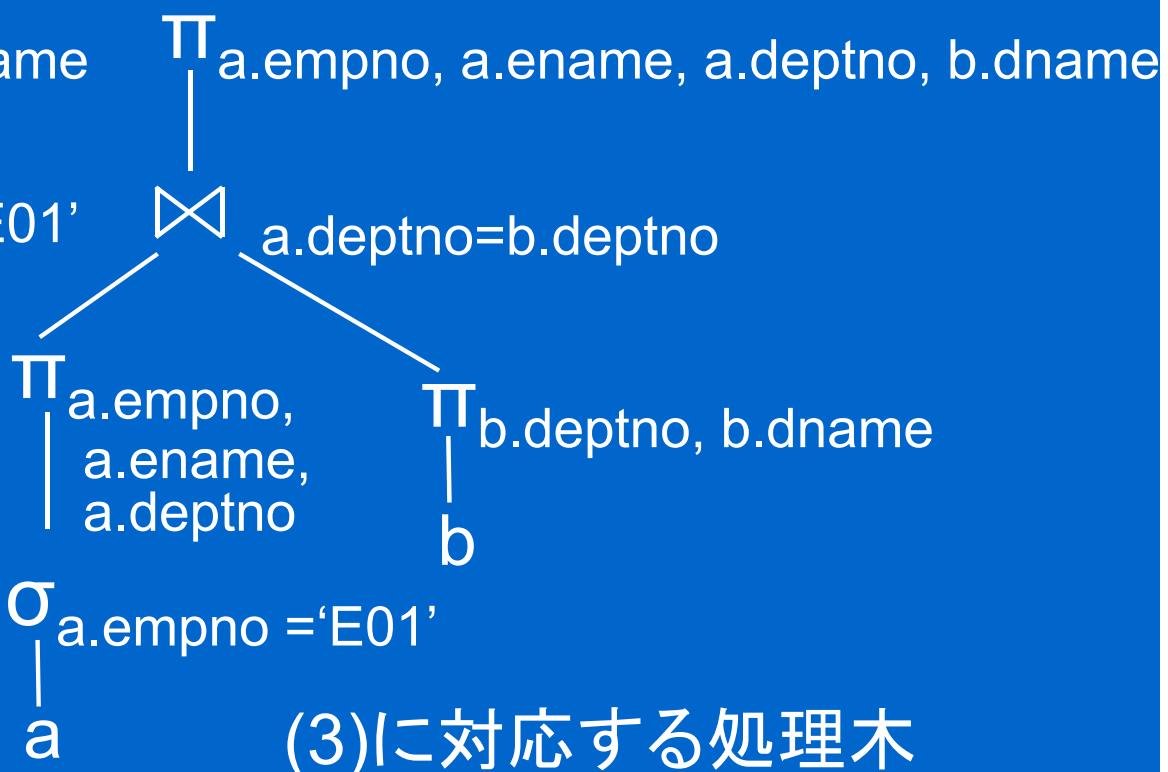
コスト見積りによる問合せ最適化 (Cost-based Query Optimization)

- 問合せを実行するコスト(問合せの結果を求めるのに必要な時間)を見積もる
- リレーションR, Sのタップルの数(濃度)を、それぞれN, Mとする
  - 直積演算R×Sのコスト　N･M、射影演算$\pi_{attribute}$(R)のコストN
- 選択演算によりタップルが絞り込まれる割合(選択演算をする前のタップル数と、演算をした後のタップル数の比)を選択率という
  - 選択演算$\sigma_{condition}$(R)のコスト:
    B$^+$木による索引が使える場合はH+1+$\lambda_{condition}$･N
    (HはB$^+$木の高さ、$\lambda_{condition}$は条件式conditionでの選択率≦1)
    索引部分でHページ、データページの中から1ページをアクセスする。選択されたデータの取り出しに$\lambda_{condition}$･Nページをアクセス

コスト見積りによる問合せ最適化の例



(1)に対応する処理木



(3)に対応する処理木

(1), (3)を実行するコストを見積もる
どちらのコストが小さいか？

規則を利用した問合せ最適化 (Rule-based, Heuristic-based Query Optimization)

問合せ最適化は、次の規則で行われる

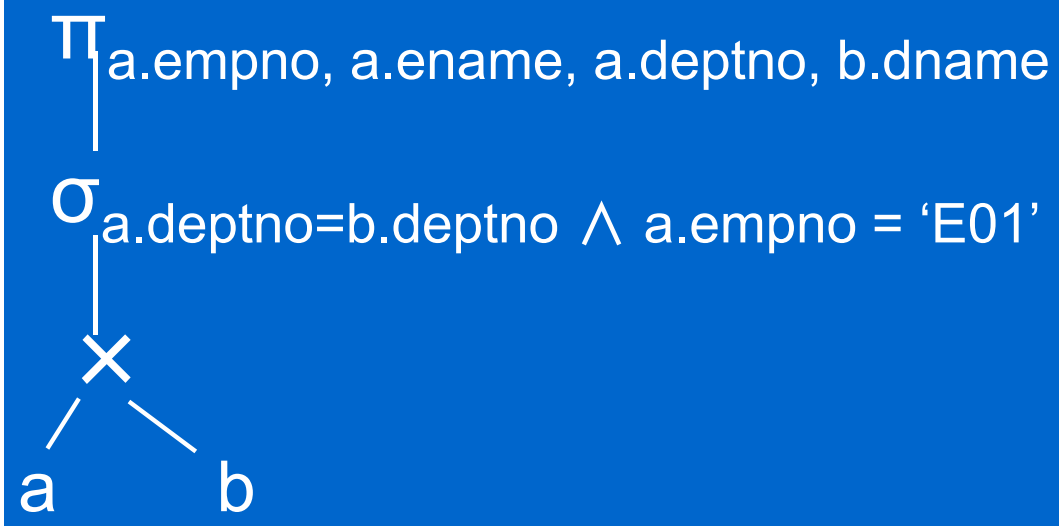
(1)**選択**(特に1つだけの表についての選択)をできるだけ早い段階で適用(ANDで結ばれた条件を持つ選択を、可能ならANDの前と後の条件で分けた複数の選択に分割)

(2)**射影**による不要な属性の削除をできるだけ早い段階で実行

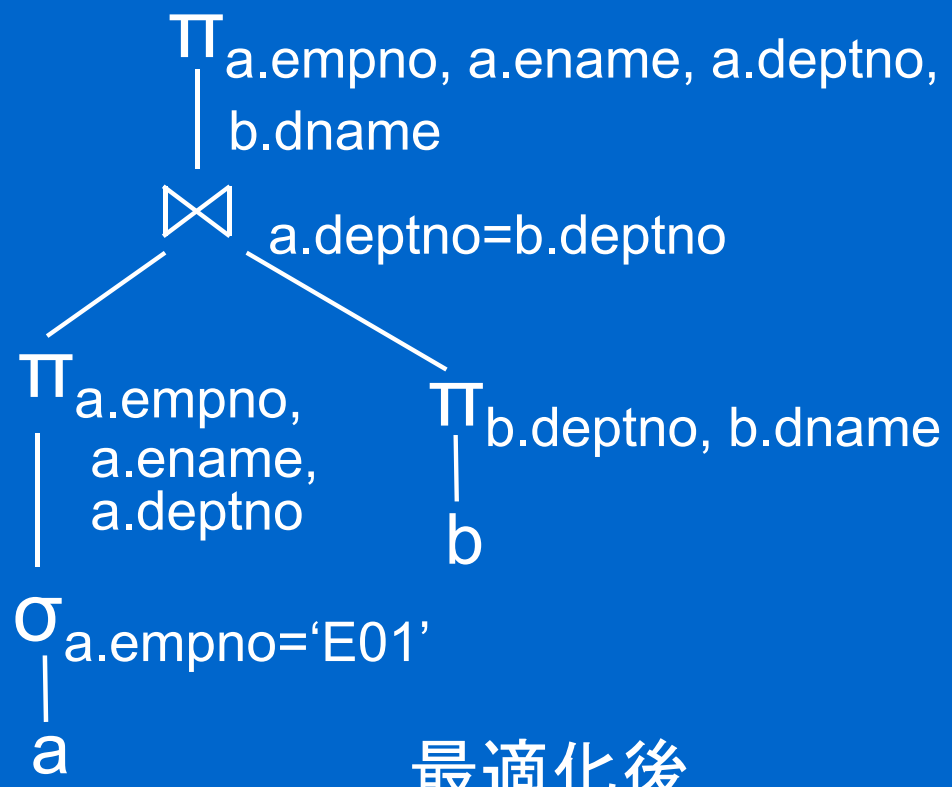
(3)直積の後に選択を行うときは，可能な限り**結合**にまとめる

(4)連続した選択および射影を、**単一の選択**、**単一の射影**、または単一の選択とそれに続く単一の射影にする

規則を利用した問合せ最適化の例



最適化前



最適化後

基本データ操作の実行方法

問合せ最適化により、問合せで処理すべき基本データ操作の系列が決まる

各基本データ操作列を実行することで、問合せが処理される

ここでは、次の2つの基本データ操作列の実行方法を示す

1. 選択操作
2. 結合操作

選択操作の実行方法

(A) 線形探索

- 表のデータファイルの全タップルを順次読み出し、選択条件を満たすものを抽出する

(B) 索引を用いた探索

- 表のデータファイルがハッシュファイル、索引付きファイル、B^+木などの索引をもつファイル編成で構成されており、選択条件で索引が利用できる場合には、索引を用いて選択条件を満たすタプルを探索する

結合操作の実行方法

結合操作の実行方法には次の3つがある

(1) **入れ子ループ結合(nested loop join)**

入れ子ループ結合は結合演算を行うための最も基本的な手順

(2)**索引を用いた方法**

索引を引くことで内側のループでの探索を省く

(3) **マージ結合(merge join)**

RとSがそれぞれXとYでソートされている場合にはマージ結合を適用できる

(4) **ハッシュ結合(hash join)**

ハッシュ表により、一方のファイル中の結合条件を満たすタプルを効率よく求める

以下では、2つの表RとSの$R \bowtie_{R.X=S.Y} S$という結合演算を考える

RとSのタップルを、それぞれ、r1, r2, ..., rm, s1, s2, ..., snで表す

(1) 入れ子ループ結合

次の擬似コードで表現される

for *i* := 1 to m do
  for *j* := 1 to n do
    if $r_i[X] = s_j[Y]$
      then r_iとs_jを結合したタプルを出力

(2) 索引を用いた結合

表Sの列S.Yに索引が作成されていると仮定すると、索引を用いた結合の実行方法は、次の擬似コードで表現される

for *i* := 1 to m do

S.Yに関する索引により$r_i[X] = s_j[Y]$を満たすSのタプルs_jを探索

r_iとs_jを結合したタプルを出力

(3) マージ結合

結合すべき2つの表RとSの各タップルが、結合条件の列R.XとS.Yの値でソートしてから、結合を行う

次の擬似コードで表現される

Rのタップルを R.Xの値でソート

Sのタップルを S.Yの値でソート

ソート後のRのタップル r1, …, rmとSのタップル s1, …, snについて結合条件を満たすタップルを結合

（次のページ参照）

(3) マージ結合（つづき）


```
i := 1
j := 1
while true do
  while ri[X] > sj[Y] do
    j := j+1
    if j>n then 終了
  while ri[X] < sj[Y] do
    i := i+1
    if i>m then 終了
  if ri[X] = sj[Y] then
    k := j
    repeat
      riとsjを結合したタプルを出力
      j := j+1
    until j>n またはri[X]≠sj[Y]
    i := i+1
    if i>m then 終了
    if ri[X]=sk[Y] then j := k
    else if j>n then 終了
```

(4)ハッシュ結合

ハッシュテーブルを利用して結合

for *i*:=1 to n do
  タプルr_iをT(hash(r_i[X]))に格納
 for *j*:=1 to m do
  foreach タプルr_i in T(hash(s_j[Y]))
   if r_i[X]=s_j[Y] then
    r_iとs_jを結合したタプルを出力

（注）hash(): ハッシュ関数、T(): ハッシュテーブル