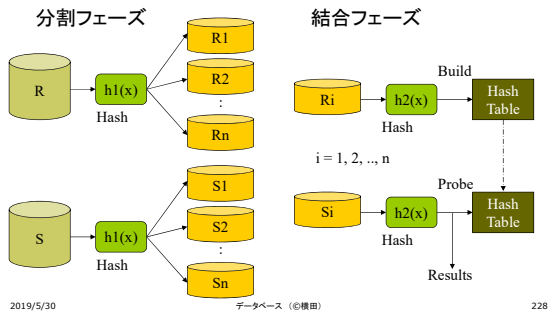
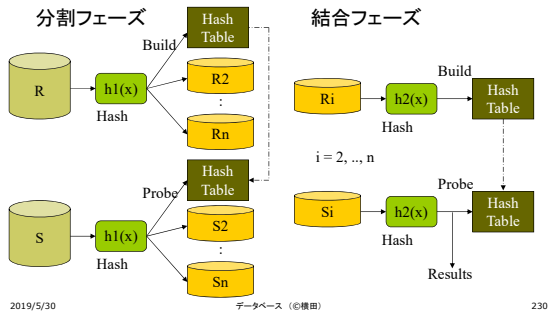
GRACEハッシュ結合（1）



GRACEハッシュ結合（2）

- コスト
  - ディスク I/O 回数: 3(|R| + |S|)
  - 比較回数: $\Sigma\{S_i\} = (\{S\}/n) \times n = \{S\}$
    - 結合フェーズでもハッシュ結合を使うとして、
  - ハッシュ適用回数:
    - 分割フェーズ: {R} + {S}
    - 結合フェーズ: $\Sigma(R_i + S_i) = \{R\} + \{S\}$
  - 全体: 2({R} + {S})

ハイブリッドハッシュ結合（1）



ハイブリッドハッシュ結合（2）

- コスト
  - ディスク I/O 回数:
    - 3(|R| + |S|) - 2 (|R| + |S|)/L= (3 - 2|M|/|R|) × (|R| + |S|)
  - 比較回数とハッシュ適用回数
    - GRACE と同じ

	ディスクI/O	ハッシュ適用	比較回数
Simple	|R|(|R| + |S|)/|M|	(|R|+|M|)({R}+{S})/2|M|	{S}
GRACE	3(|R| + |S|)	2({R} + {S})	{S}
Hybrid	(3-2|M|/|R|)(|R|+|S|)	2({R} + {S})	{S}

ディスク I/O の比較

- Simple Hash 結合と GRACE Hash 結合
  - |R|/|M| < 3 より
    - |R| < 3|M| の時は、Simple Hash が得
    - それ以外の時（普通）は、GRACE Hash が得
- Simple Hash 結合と Hybrid Hash 結合
  - |R|/|M| < (3-2 |M|/|R|) より
    - |R| < 2|M| の時は、Simple Hash が得
    - それ以外の時は、Hybrid Hash が得
- GRACE Hash 結合と Hybrid Hash 結合
  - 常に 3 > (3-2|M|/|R|) : Hybrid Hash が得
  - ただし実現方法は GRACE Hash の方が簡単

練習問題

- 製造元関係（商品番号、商品名、製造元）と商品分類関係（商品分類、商品名）を、商品名でハッシュ結合処理する場合を考える。製造元関係と商品分類関係のページ数をそれぞれ100、50とし、商品分類関係でビルドし、製造元関係でプローブする。
  1. ハッシュテーブルに使えるページ数を10とした場合の、Simpleハッシュ結合、GRACEハッシュ結合、Hybrid ハッシュ結合のディスク I/O数を見積もれ。
  2. GRACEハッシュ結合より Simple ハッシュ結合のI/Oが少ないのはハッシュテーブルに使えるメモリ量がどのような場合か
  3. Hybridハッシュ結合より Simple ハッシュ結合のI/Oが少ないのはハッシュテーブルに使えるメモリ量がどのような場合か

割り算 ($R_a[X] \div R_b[Y]$)

- 直接法
  - ソート利用: R_aのXをマイナー、X_cをメジャーでソートし、R_bも Xでソートする
    - 例: R_a[販売員] ÷ R_b[販売員]
      - 商品名が変る毎に販売員を付き合せ最後まで行ったら商に追加
  - ハッシュ利用: R_bに対応したハッシュテーブルH_1と商の候補のハッシュテーブルH_2 に R_bに対応した bit を用意
    - R_aをハッシュしH_1のエントリを調べ、H_2の bit を立てる
- 間接法
  - 集約演算を使う: R_aをGroup-ByしR_bと同じCountか調査
    - 予めR_aを R_b で Semi-Join しておく

ソートを利用した割り算の例

販売実績関係 [R_a]

販売員	商品名
田中	17'CRT
田中	15'TFT
田中	3.5' HDD
田中	600dpi LPR
鈴木	15'TFT
鈴木	3.5' HDD
鈴木	600dpi LPR
佐藤	17'CRT
佐藤	15'TFT
佐藤	3.5' HDD

ソート

マイナー　メジャー

販売員	商品名
佐藤	15'TFT
鈴木	15'TFT
田中	15'TFT
佐藤	17'CRT
田中	17'CRT
佐藤	3.5' HDD
鈴木	3.5' HDD
田中	3.5' HDD
鈴木	600dpi LPR
田中	600dpi LPR

販売員関係 [R_b]

販売員
佐藤
鈴木
田中

結果

商品名
15'TFT
3.5' HDD

ハッシュを利用した割り算の例

販売実績関係 [R_a]

販売員	商品名
田中	17'CRT
田中	15'TFT
田中	3.5' HDD
田中	600dpi LPR
鈴木	15'TFT
鈴木	3.5' HDD
鈴木	600dpi LPR
佐藤	17'CRT
佐藤	15'TFT
佐藤	3.5' HDD

H_1

佐藤	1
鈴木	2
田中	4

H_2

	佐藤	鈴木		田中
15'TFT	1	1		1
600dpi LPR		1		1
17'CRT	1			1
3.5' HDD	1	1		1

集約演算を利用した割り算の例

販売実績関係 [R_a]

販売員	商品名
田中	17'CRT
田中	15'TFT
田中	3.5' HDD
田中	600dpi LPR
鈴木	15'TFT
鈴木	3.5' HDD
鈴木	600dpi LPR
佐藤	17'CRT
佐藤	15'TFT
佐藤	3.5' HDD

SELECT 商品名, COUNT(販売員)
FROM 販売実績
GROUP BY 商品名

商品名	COUNT(販売員)
17'CRT	2
15'TFT	3
3.5' HDD	3
600dpi LPR	2

練習問題

- 下記の SQL は、semi-join と最終結果の商品名を選んでいない。

SELECT 商品名, COUNT(販売員)
FROM 販売実績
GROUP BY 商品名

semi-join を含み、最終結果を選ぶように書き換えよ

結合、割り算以外の 2 入力関係演算

- 準結合（Semi-Join）
  - 等結合の結果が一方の関係のみ
    - Nested Query に対応
  - 基本的なコストは等結合と同じ
    - ハッシュテーブルを小さくできる
  - 使われ方
    - 集合演算に利用
    - 分散データベース等で転送データを絞り込む時に用いられる

結合、割り算以外の 2 入力関係演算

- 集合積
  - 全ての属性に対する等結合または準結合
    - コストは等結合と同じ
- 集合和
  - 2 つの関係の Concatenation 後の重複除去
  - または一方の関係と集合差との Concatenation

結合、割り算以外の 2 入力関係演算

- 集合差
  - 全ての属性に対する Anti-Semi-Join
    - Anit-Semi-Join：
      - Semi-Join の結果に含まれないタプルを選択
    - Semi-Join の処理の中に組み込むことが可能
      - Semi-Join で捨てるタプルを残し、残すタプルを捨てる
  - コストは等結合と同じ

問い合わせ木のコスト最適化

- 演算適用の順番を変えることでコストが変る
- 関係代数式の変形
  - $\sigma_{c1 \wedge c2}(R) = \sigma_{c1}(\sigma_{c2}(R))$
  - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
  - $\{x1, x2, \ldots, xn\} \subseteq \{y1, y2, \ldots, ym\}$ なら
    $\pi_{x1, x2, \ldots, xn}(\pi_{y1, y2, \ldots, ym}(R)) = \pi_{x1, x2, \ldots, xn}(R)$
  - $\pi_{x1, x2, \ldots, xn}(\sigma_c(R)) = \sigma_c(\pi_{x1, x2, \ldots, xn}(R))$
  - 条件 c が属性 $\{y1, y2, \ldots, ym\}$ を参照するなら
    $\pi_{x1, x2, \ldots, xn}(\sigma_c(R)) = \sigma_c(\pi_{x1, x2, \ldots, xn, y1, y2, \ldots, ym}(R))$

問い合わせ木のコスト最適化

- 条件c1が R_1の属性のみを参照するなら
  - $\sigma_{c1}(R_1 \times R_2) = \sigma_{c1}(R_1) \times R_2$
  - $\sigma_{c1}(R_1 \bowtie_{c2} R_2) = \sigma_{c1}(R_1) \bowtie_{c2} R_2$
- 射影と集合演算
  - $\sigma_c(R_1 \cup R_2) = \sigma_c(R_1) \cup \sigma_c(R_2)$
  - $\sigma_c(R_1 \cap R2) = \sigma_c(R_1) \cap \sigma_c(R_2)$
  - $\sigma_c(R_1 - R2) = \sigma_c(R_1) - \sigma_c(R_2)$

問い合わせ木のコスト最適化

- $\Pi_{x1, x2, \ldots, xn}(R_1 \times R_2)$
  $= \Pi_{y1, y2, \ldots, ym}(R_1) \times \Pi_{z1, z2, \ldots, zk}(R_2)$
- $\Pi_{x1, x2, \ldots, xn}(R_1 \bowtie_c R_2)$
  $= \Pi_{y1, y2, \ldots, ym}(R_1) \bowtie_c \Pi_{z1, z2, \ldots, zk}(R_2)$
- 条件cが{u1, …, up}と{v1, …, vq}を参照するなら、
- $\Pi_{x1, x2, \ldots, xn}((R_1 \bowtie_c R_2)$
  $= \Pi_{y1, y2, \ldots, ym, u_1, \ldots, u_p}(R_1) \bowtie_c \Pi_{z1, \ldots, zk, v1, \ldots, vq}(R_2)$
- $\Pi_{x1, x2, \ldots, xn}((R_1 \cup R_2)$
  $= \Pi_{x1, x2, \ldots, xn}(R_1) \cup \Pi_{x1, x2, \ldots, xn}(R_2)$

問い合わせ木のコスト最適化の例

- 「商品分類が Computer に属する GELL の商品で、秋葉原支店での販売価格が 100,000 円以下の商品の商品名と秋葉原支店での在庫を出力せよ」
- SQLで記述

```
SELECT  製造元.商品名, 秋葉原支店.在庫
FROM    製造元, 商品分類, 秋葉原支店
WHERE   製造元.商品名 = 商品分類.商品名
AND     製造元.製造元 = 'GELL'
AND     商品分類.商品分類 = 'Computer'
AND     製造元.商品番号 = 秋葉原支店.商品番号
AND     秋葉原支店.販売価格 ≦ 100,000
```

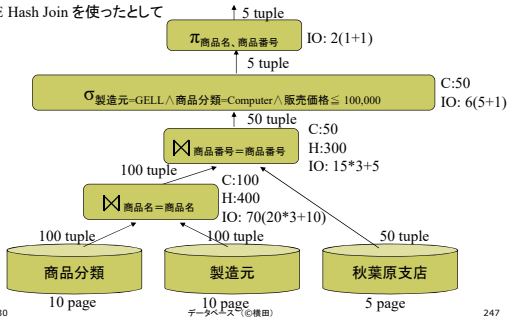
関係代数表現（実行スケジュール）の例

1. 先に結合してから絞り込む
   - $\Pi_{商品名,\ 在庫}(\sigma_{製造元=GELL \wedge 商品分類=Computer \wedge 販売価格 \leq 100,000}$
     $((商品分類 \bowtie_{商品名} 製造元) \bowtie_{商品番号} 秋葉原支店))$
2. 選択して絞り込んでから結合する
   - $\Pi_{商品名,\ 在庫}((\sigma_{商品分類\ =\ Computer}(商品分類)$
     $\bowtie_{商品名} (\sigma_{製造元\ =\ GELL}(製造元)))$
     $\bowtie_{商品番号} (\sigma_{販売価格\ \leq\ 100,000}(秋葉原支店)))$

- 絞込み（選択率）を予測した最適実行スケジュール
  - ディスク I/O、ハッシュ・比較回数が最小になるように
- 上の例だと 2 の方が良い

先に結合してから絞り込む

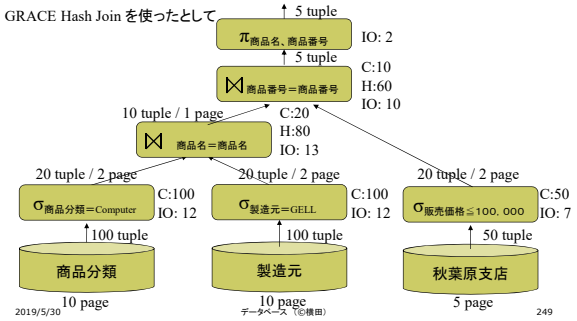
GRACE Hash Join を使ったとして



先に結合してから絞り込む

- 比較（C）:
  - 100 + 50 + 50 = 200
- ハッシュ（H）:
  - 400 + 300 = 700
- I/O
  - 70 + 50 + 6 + 2 = 128

選択して絞り込んでから結合する



選択して絞り込んでから結合する

- 比較(C):
  - 100 + 100 + 50 + 20 + 10 = 280 (vs. 200)
- ハッシュ(H):
  - 80 + 60 = 140 (vs. 700)
- I/O
  - 12 + 13 + 7 + 13 + 10 + 2 = 57 (vs. 128)