Introduction to Databases
《数据库引论》

Lecture 10: Query Optimization
第10讲：查询优化

周水庚 / Shuigeng Zhou

邮件: sgzhou@fudan.edu.cn 网址：admis.fudan.edu.cn/sgzhou

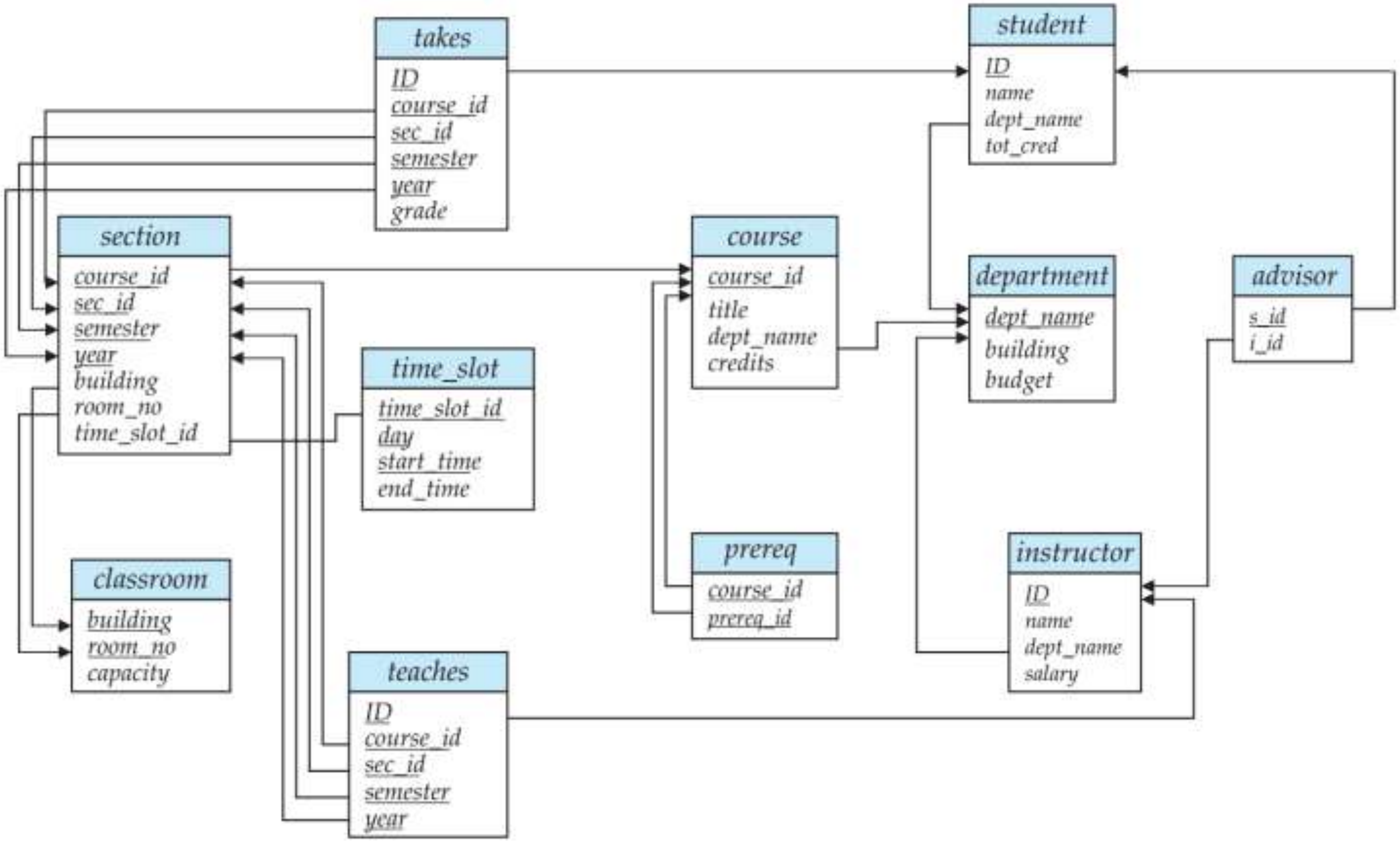
复旦大学计算机科学技术学院

Outline of the Course

- **Part 0: Overview**
  - Lect. 1 (Feb. 29) - Ch1: Introduction
- **Part 1 Relational Databases**
  - Lect. 2 (Mar. 7) - Ch2: Relational model (data model, relational algebra)
  - Lect. 3 (Mar. 14) - Ch3: SQL (Introduction)
  - Lect. 4 (Mar. 21) – Ch4/5: Intermediate and Advanced SQL
- **Part 2 Database Design**
  - Lect. 5 (Mar. 28) - Ch6: Database design based on E-R model
  - **Apr. 4 (Tomb-Sweeping Day): no course**
  - Lect. 6 (Apr. 11/18) - Ch7: Relational database design
- **Midterm exam: Apr. 25**
  - 13: 00-15: 00, H3109
- **Part 3 Data Storage & Indexing**
  - Lect. 7 (May 2 -> Apr. 28) - Ch12/13: Storage systems & structures
  - Lect. 8 (May 10) - Ch14: Indexing and Hashing
- ☞ **Part 4 Query Processing & Optimization**
  - Lect. 9 (May 17) - Ch15: Query processing
  - Lect. 10 (May 24) - Ch16: Query optimization
- **Part 5 Transaction Management**
  - Lect. 11 (May 31) - Ch17: Transactions
  - Lect. 12 (Jun. 7) - Ch18: Concurrency control
  - Lect. 13 (Jun. 14) - Ch19: Recovery system

Final exam: 13:00-15:00, Jun. 26

University Database



The Banking Schema

- branch = (branch_name, branch_city, assets)
- customer = (customer_id, customer_name, customer_street, customer_city)
- loan = (loan_number, amount)
- account = (account_number, balance)
- employee = (employee_id, employee_name, telephone_number, start_date)

- dependent_name = (employee_id, dname) (derived from a multivalued attribute)

- account_branch = (account_number, branch_name)
- loan_branch = (loan_number, branch_name)
- borrower = (customer_id, loan_number)
- depositor = (customer_id, account_number, access_date)
- cust_banker = (customer_id, employee_id, type)
- works_for = (worker_employee_id, manager_employee_id)

- payment =(loan_number,payment_number,payment_date,payment_amount)

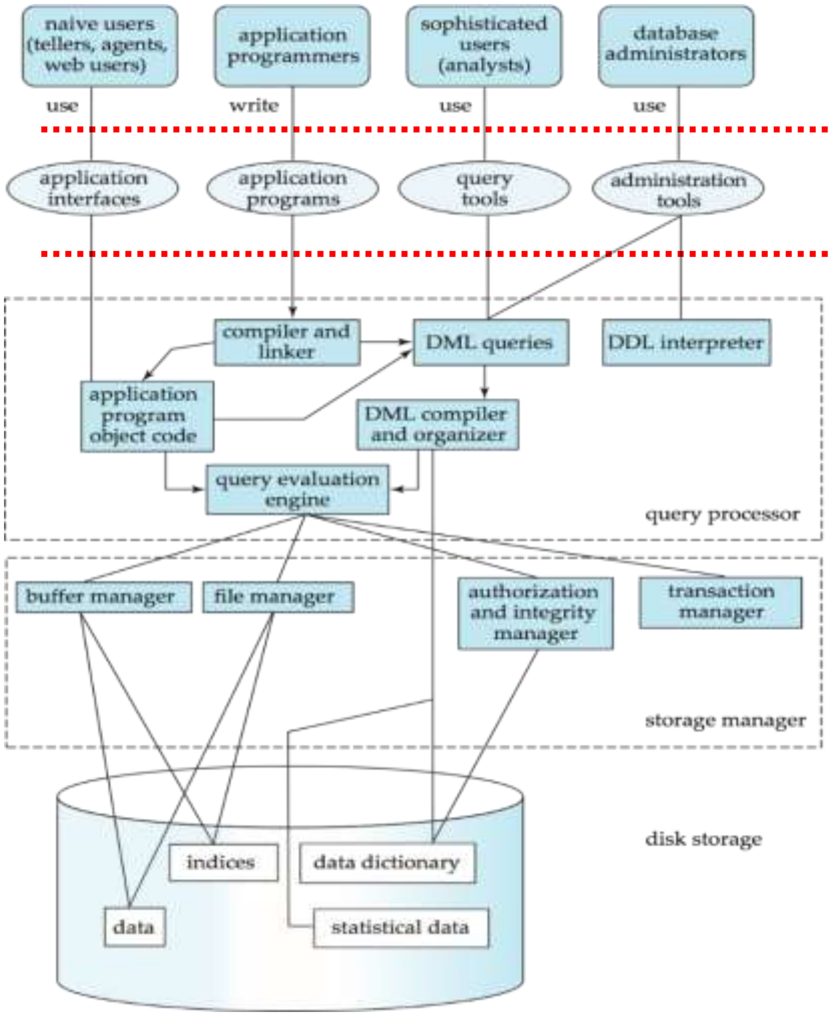
- savings_account = (account_number, interest_rate)
- checking_account = (account_number, overdraft_amount)

Outline

☞ Introduction

- Transformation of Relational Expressions
- Catalog Information for Cost Estimation
- Estimation of Statistics
- Dynamic Programming for Choosing Evaluation Plans

Database System Structure



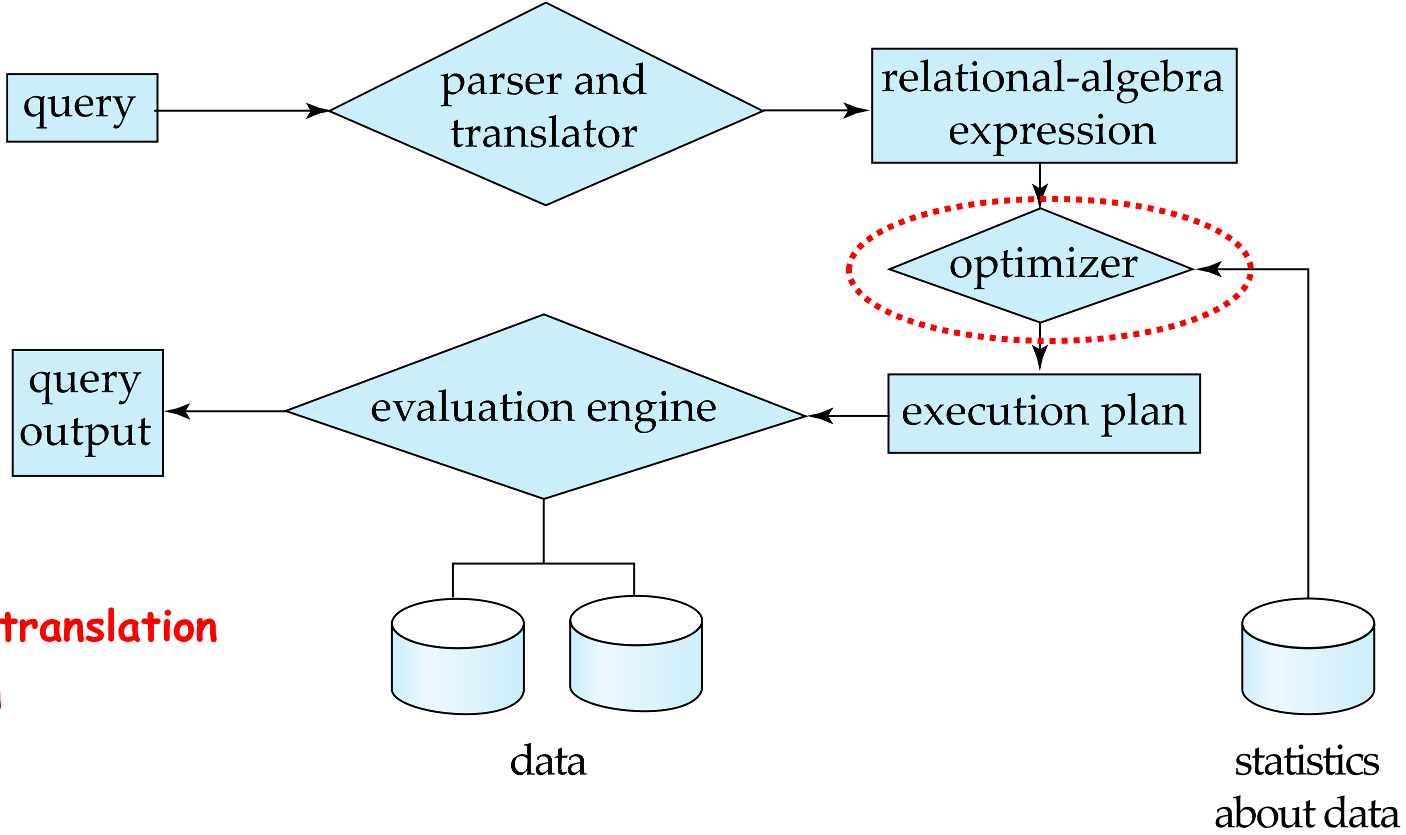
Users

Applications/tools

DBMS

Database

Basic Steps in Query Processing



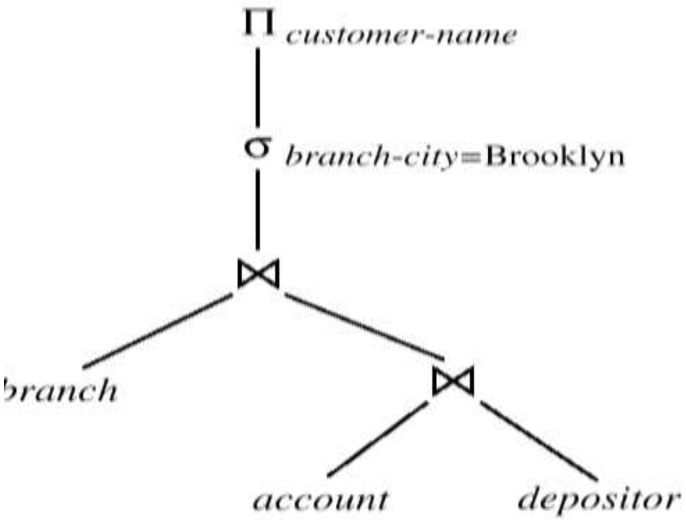
1. Parsing and translation
2. Optimization
3. Evaluation

Introduction

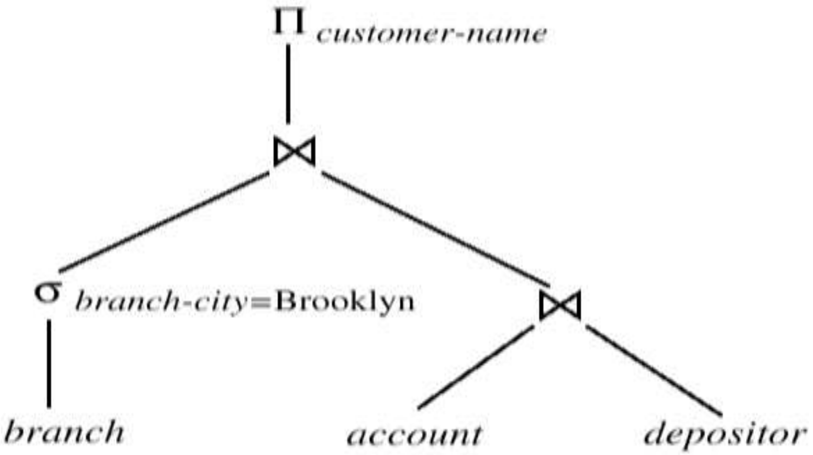
- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

Introduction (Cont.)

Relations generated by two **equivalent expressions** have **the same set of attributes** and contain the **same set of tuples**, although their attributes may be **ordered differently.**



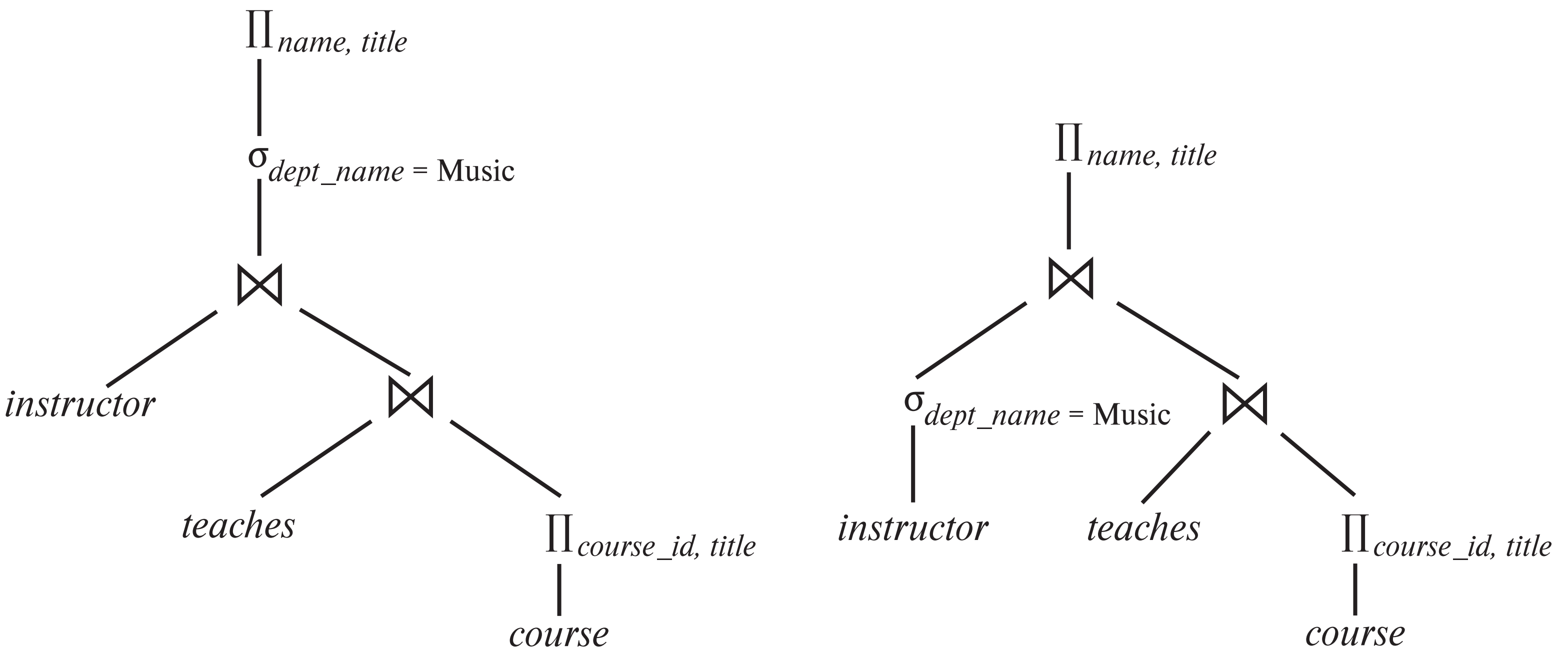
(a) Initial Expression Tree



(b) Transformed Expression Tree

Introduction (Cont.)

- Eg：查询找出Music系所有教师的名字以及每位教师所教授课程的名称

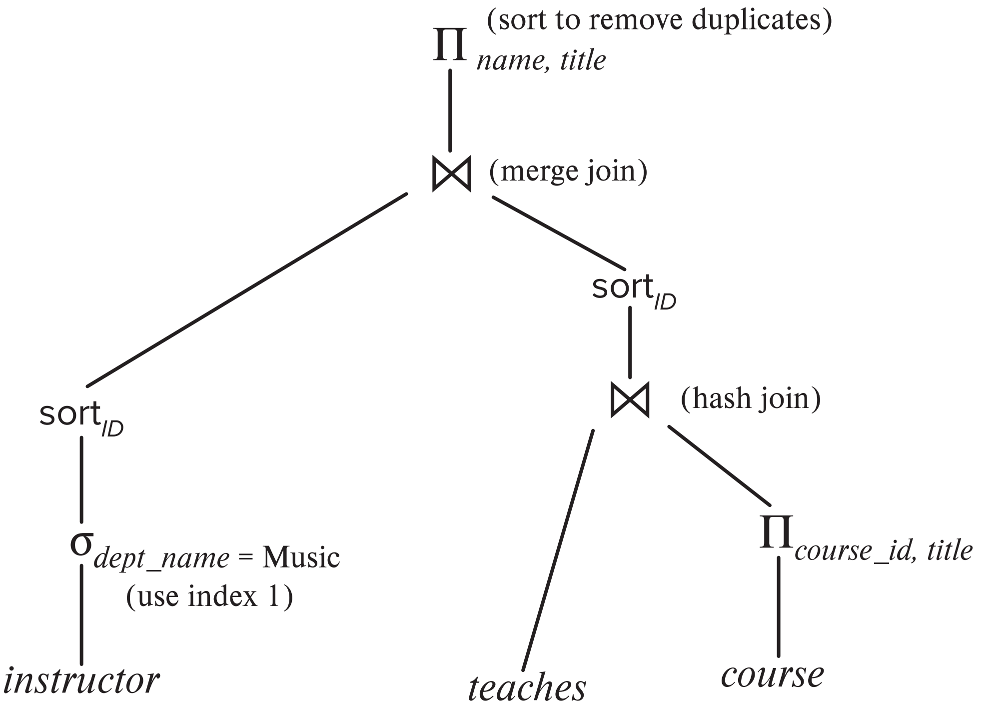


(a) Initial expression tree

(b) Transformed expression tree

Introduction (Cont.)

- **执行计划：**需明确每个运算应使用的算法以及运算之间的执行如何协调



Introduction (Cont.)

- **Generation of query-evaluation plans for an expression involves several steps:**
  1. Generating **logically equivalent expressions**(步骤1：产生逻辑上与给定表达式等价的表达式). Use **equivalence rules** to transform an expression into an equivalent one.
  2. Annotating resultant expressions to get **alternative query plans**(步骤2：对所产生的表达式以不同方式标注，产生不同的查询执行计划)
  3. Choosing **the cheapest plan based on estimated cost**(步骤3：估计每个执行计划的代价，选择估计代价最小的执行计划)
- **The overall process is called cost based optimization**

Outline

- Introduction
- ☞ Transformation of Relational Expressions
- Catalog Information for Cost Estimation
- Estimation of Statistics
- Dynamic Programming for Choosing Evaluation Plans

Relational Expression Transformation

- Two relational algebra expressions are said to be **equivalent** if on **every** legal database instance the two expressions generate the same set of tuples
  - Note: order of tuples is irrelevant
- In SQL, inputs and outputs are multisets of tuples
- An equivalence rule says that expressions of two forms are equivalent
  - Can replace expression of first form by second, or vice versa

Equivalence Rules

- **Conjunctive selection** operations can be deconstructed into a sequence of individual selections.(规则1：合取选择运算可分解为单个选择运算的序列)

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

- **Selection** operations are **commutative**.(规则2：选择运算满足交换律)

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

Equivalence Rules

- Only the last in a sequence of projection operations is needed, the others can be omitted.（规则3：多个连续投影中只有最后一个运算是必需的，其余可忽略）

$$\Pi_{t_1}(\Pi_{t_2}(\ldots(\Pi_{tn}(E))\ldots)) = \Pi_{t_1}(E)$$

- Selections can be combined with Cartesian products and theta joins.（规则4：选择操作可以与笛卡尔积以及θ连接相结合）

$$\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$$

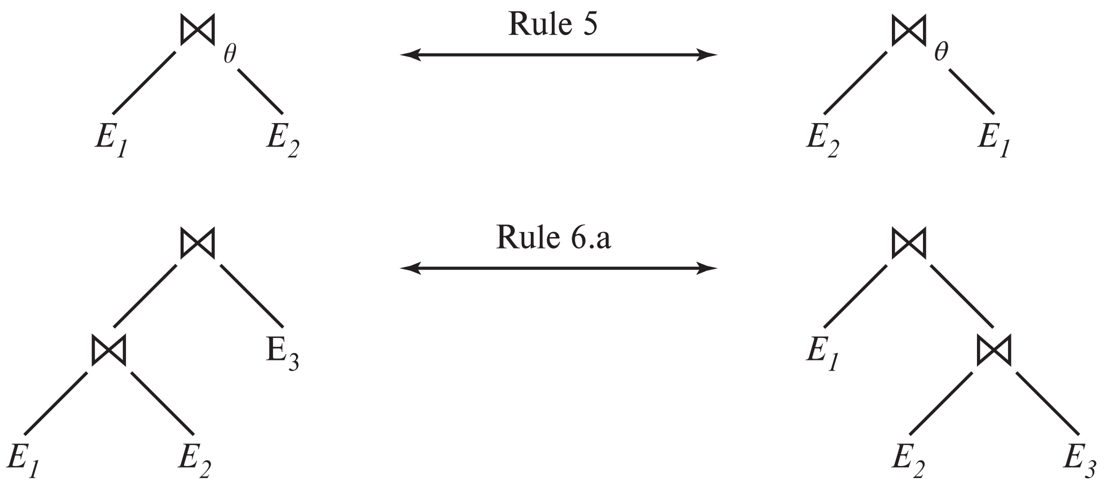
$$\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$$

Equivalence Rules (Cont.)

- **Theta-join** operations(and **natural joins**) are **commutative**.(规则5：θ 连接满足交换律)
  - $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
- **Natural join** operations are **associative** (规则6a：自然连接满足结合律)
  - $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
- **Theta joins** are **associative** in the following manner, where θ_2 involves attributes from only E_2 and E_3 (规则6b：θ连接满足下列方式的结合律，其中θ_2只涉及E_2和E_3的属性)
  - $(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$

Equivalence Rules (Cont.)

- **规则5**：θ连接满足交换律 $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
- **规则6a**：自然连接满足结合律 $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
- **规则6b**：θ连接满足下列方式的结合：$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$，其中$\theta_2$只涉及$E_2$和$E_3$的属性



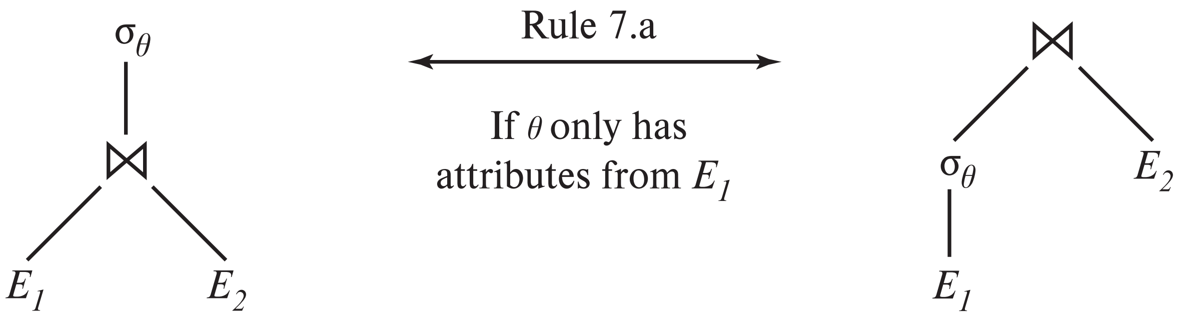
Equivalence Rules (Cont.)

- The **selection operation distributes over the theta join** operation under the following two conditions: (a)When all the attributes in θ_0 involve only the attributes of one of the expressions (E_1) being joined. (规则7：选择操作在下面两个条件下对θ连接满足分配律，a. 当选择条件θ_0中的所有属性只涉及参与连接的表达式之一(如E_1)时)

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

  - (b) When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.(b. 当选择条件θ_1只涉及E_1的属性，选择条件θ_2只涉及E_2的属性时)

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$



Equivalence Rules (Cont.)

- The projections operation distributes over the theta join operation as follows:(a) if θ involves only attributes from $L_1 \cup L_2$ （规则8：令L_1、L_2分别代表E_1、E_1的属性子集，投影操作在下列条件下对θ连接满足分配律: a. 如果连接条件θ只涉及$L_1 \cup L_2$中的属性)

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

  (b) Consider a join $E_1 \bowtie_\theta E_2$. Let L_1 and L_2 be sets of attributes from E_1 and E_2, respectively. Let L_3 be attributes of E_1 that are involved in join condition θ, but are not in $L_1 \cup L_2$, and let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$. （b. 针对连接$E_1 \bowtie_\theta E_2$，令L_3是E_1出现在连接条件θ中但不在$L_1 \cup L_2$中的属性，令L_4是E_2出现在连接条件θ中但不在$L_1 \cup L_2$中的属性)

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

Equivalence Rules (Cont.)

- Set union and intersection are commutative(set difference is not commutative).（规则9：集合的并和交满足交换律）
  - $E_1 \cup E_2 = E_2 \cup E_1$
  - $E_1 \cap E_2 = E_2 \cap E_1$
- Set union and intersection are associative.（规则10：集合的并和交满足结合律）
  - $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$
  - $(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$
- Selection distributes over ∪, ∩, –. （规则11：选择操作对并、交、差满足分配率）
  - $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - \sigma_\theta(E_2)$
    - similarly for ∪ and ∩ in place of –
  - $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$
    - similarly for ∩ in place of –, but not for ∪
- Projection distributes over union.（规则12：投影对并的分配律）
  - $\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$

Example 1: Pushing Selections

- Eg.: Find the names of all instructors in the Music department, along with the titles of the courses that they teach

$$\Pi_{name,\ title}(\sigma_{dept\_name=\ 'Music'}(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course))))$$

- Transformation using rule 7a

$$\Pi_{name,\ title}((\sigma_{dept\_name=\ 'Music'}(instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course)))$$

Example 2: Multiple Transformations

- Eg.: Find the names of all instructors in the Music department who have taught a course in 2017, along with the titles of the courses that they taught

  $\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"} \wedge year = 2017}(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}(course))))$

- Rule 6a:

  $\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"} \wedge year = 2017}((instructor \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course)))$

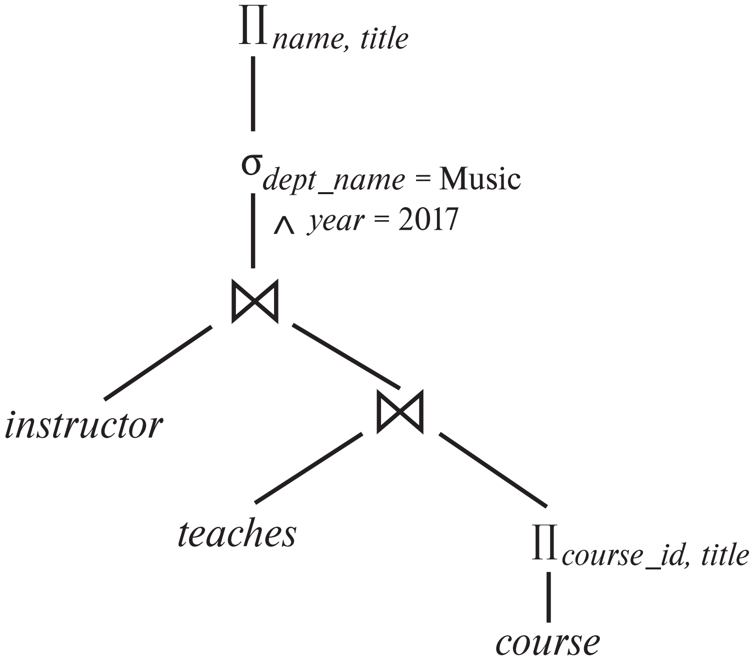
- Rule 7a:

  $\Pi_{name,\ title}((\sigma_{dept\_name=\text{"Music"} \wedge year = 2017}(instructor \bowtie teaches)) \bowtie \Pi_{course\_id,\ title}(course))$

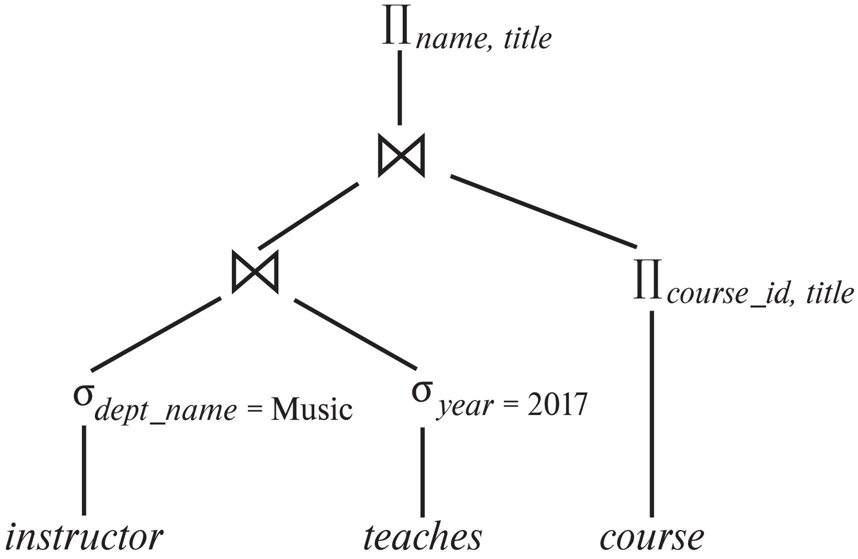
- Rule1 & 7a:

  $\Pi_{name,\ title}((\sigma_{dept\_name=\text{"Music"}}(instructor) \bowtie \sigma_{year = 2017}(teaches)) \bowtie \Pi_{course\_id,\ title}(course))$

Example 2: Multiple Transformations



(a) Initial expression tree



(b) Tree after multiple transformations

Example 3: Multiple Transformations

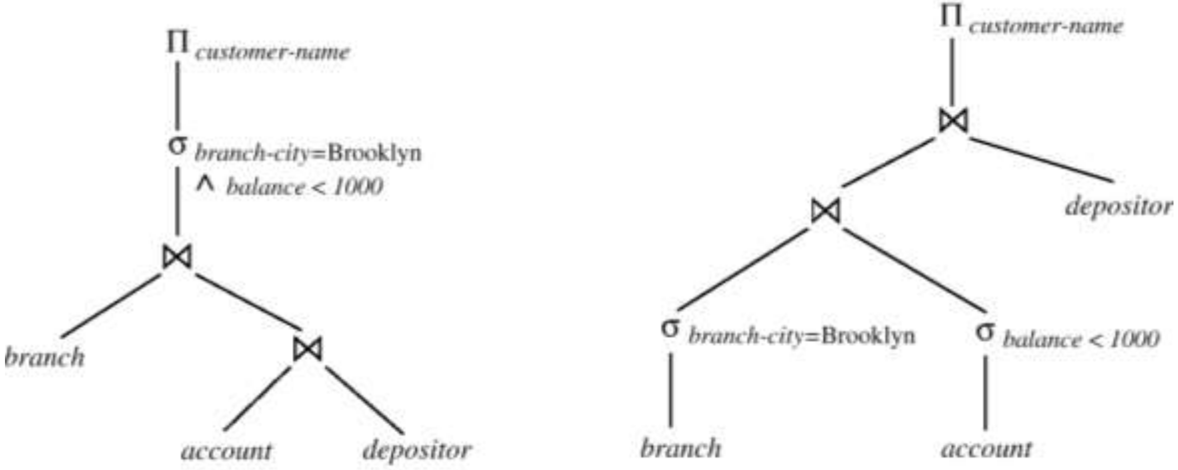
- **Eg.:** Find the names of all customers with an account at a Brooklyn branch whose account balance is over $1000
  - $\Pi_{CN}(\sigma_{BC=\text{“Brooklyn”} \wedge\, balance>1000}(branch \bowtie (account \bowtie depositor)))$
  - CN: customer name, BC: branch city
- **Task**: Give one equivalent expression with better execution performance

- **Performing the selection as early as possible reduces the size of the relation to be joined.**

Example 3: Multiple Transformations

- **Eg.**: Find the names of all customers with an account at a Brooklyn branch whose account balance is over $1000
  - $\Pi_{CN}(\sigma_{BC=\text{“Brooklyn”} \wedge\, balance>1000}(branch \bowtie (account \bowtie depositor)))$
- **Task**: Give one equivalent expression with better execution performance
- One solution: **Performing the selection as early as possible**

$$\Pi_{CN}((\boldsymbol{\sigma_{BC=\text{"Brooklyn"}}(branch)} \bowtie \boldsymbol{\sigma_{balance>1000}(account)}) \bowtie depositor)$$



(a) Initial Expression Tree

(b) Tree After Multiple Transformations

Example 4: Projection Operation

$\Pi_{customer\text{-}name}((\sigma_{branch\text{-}city\ =\ \text{"Brooklyn"}}\ (branch) \bowtie account) \bowtie depositor)$

- When we compute

  $(\sigma_{branch\text{-}city\ =\ \text{"Brooklyn"}}\ (branch) \bowtie account\ )$

  we obtain a relation whose schema is:

  (branch-name, branch-city, assets, account-number, balance)

- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:

  $\Pi_{customer\text{-}name}\ ($
  $\Pi_{account\text{-}number}\ (\sigma_{branch\text{-}city\ =\ \text{"Brooklyn"}}\ (branch) \bowtie account) \bowtie depositor))$

Join Ordering

- For three relations r_1, r_2, and r_3,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

- If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we choose

$$(r_1 \bowtie r_2) \bowtie r_3$$

so that we can compute and store a **smaller** temporary relation

Join Ordering (Cont.)

- Consider the expression

  $\Pi_{name,\ title}(\sigma_{dept\_name=\ \text{"Music"}}\ (instructor) \bowtie teaches) \bowtie \Pi_{course\_id,\ title}\ (course))))$

- **Solution A**
  - compute $(teaches \bowtie \Pi_{course\_id,\ title}\ (course))$ first, and join the result with $\sigma_{dept\_name=\ \text{"Music"}}\ (instructor)$
  - the result of the first join is likely to be a large relation
- **Solution B**
  - compute $(\sigma_{dept\_name=\ \text{"Music"}}\ (instructor) \bowtie teaches)$ first
  - only a small fraction of instructors are likely to be from the Music department

Outline

- Introduction
- Transformation of Relational Expressions
- ☞ Catalog Information for Cost Estimation
- Estimation of Statistics
- Dynamic Programming for Choosing Evaluation Plans

Statistical Information for Relation

- **关系(表)的统计信息**
  - n_r: the number of tuples in a relation r
  - b_r: the number of blocks of r
  - s_r: the size of a tuple of r
  - f_r: the blocking factor of r, i.e., the number of tuples that fit into one block
  - $V(A, r)$: the number of distinct values that appear in r for attribute A, i.e., the size of $\Pi_A(r)$
  - **SC(A, r)**: selection cardinality of attribute *A* of relation *r*; average number of records that satisfy equality on *A*.
  - If the tuples of r are stored together physically in a file, then: $b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$

- **Estimation**
  - **Size**
  - **Distinct Values**

Catalog Information about Indices

- F_i: the average fan-out(扇出) of internal nodes of index i
  - for tree-structured indices such as B+-tree
- HT_i: the number of levels in index i
  - i.e., the height of i
  - for a balanced tree index (such as B+-tree) on attribute A of relation r, $HT_i = \lceil log_{F_i}(V(A,r)) \rceil$ (其中$V(A,r)$: the number of distinct values)
  - for a hash index, HT_i is 1
- LB_i: the number of lowest-level index blocks in i
  - i.e., the number of blocks at the leaf level of the index

Measures of Query Cost

- **Recall that**
  - Typically, **disk access** is the predominant cost, and is also relatively easy to be estimated
  - **The number of block transfers from disk** is used as a measure of the actual cost of evaluation
  - It is assumed that all transfers of blocks have the same cost
- Usually do not include the cost to write output to disk
- We refer to the cost estimate of algorithm A as E_A

简单选择操作结果大小估计

- **Equality selection** $\sigma_{A=a}(r)$
  - 假设取值**均匀分布**，则可估计选择结果有$n_r/V(A,r)$个元组
  - $SC(A,r)$：number of records that will satisfy the selection
  - $\lceil SC(A,r)/f_r \rceil$：number of blocks that these records will occupy
  - E.g. Binary search cost estimate becomes

$$E_{a2} = \lceil \log_2(b_r) \rceil + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil - 1$$

  - Equality condition on a key attribute: $SC(A,r)=1$

Statistical Information for Examples

- $f_{account}$ = 20 (20 tuples of *account* fit in one block)
- *V(branch-name, account)* = 50 (50 branches)
- *V(balance, account)* = 500 (500 different *balance* values)
- $n_{account}$ = 10000 (*account* has 10,000 tuples)
- Assume the following indices exist on *account:*
  - A primary, B+-tree index for attribute *branch-name*
  - A secondary, B+-tree index for attribute *balance*

  - n_r: the number of tuples in a relation r
  - f_r: the number of tuples that fit into one block
  - $V(A, r)$: the number of distinct values that appear in r for attribute *A*

简单选择操作结果大小估计

- Equality selection $\sigma_{A=a}(r)$
  - 假设取值**均匀分布**，则可估计选择结果有$n_r/V(A,r)$个元组
- Selections of the form $\sigma_{A\leq v}(r)$， case of $\sigma_{A\geq v}(r)$ is symmetric
  - Let c denote the estimated number of tuples satisfying the condition. If $\min(A,r)$ and $\max(A,r)$ are available in database catalog and we assume that values are uniformly distributed (值均匀分布)
    - $C = 0$, if $v < min(A,r)$
    - $C = n_r \cdot \frac{v - min(A,r)}{\max(A,r) - min(A,r)}$
    - $C = n_r$, if $v \geq max(A,r)$
  - In absence of statistical information, c is assumed to be $n_r/2$

- n_r: the number of tuples in a relation r
- $V(A,r)$: the number of distinct values that appear in r for attribute A

Outline

- Introduction
- Transformation of Relational Expressions
- Catalog Information for Cost Estimation
- ☞ Estimation of Statistics
- Dynamic Programming for Choosing Evaluation Plans

复杂选择操作结果大小估计

- **Selectivity (中选率) of a condition θ_i**
  - The probability that a tuple in the relation r satisfies θ_i
  - If s_i is the number of tuples satisfying θ_i, the selectivity of θ_i is given by s_i/n_r
- **合取：** $\sigma_{\theta 1 \wedge \theta 2 \wedge \cdots \wedge \theta n}(r)$
  - Estimated number of tuples:

$$n_r * \frac{s_1 * s_2 * \cdots * s_n}{n_r^n}$$

- **析取：** $\sigma_{\theta 1 \vee \theta 2 \vee \cdots \vee \theta n}(r)$
  - Estimated number of tuples:

$$n_r * \left(1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \ldots * (1 - \frac{s_n}{n_r})\right)$$

- **取反：** $\sigma_{\neg\theta}(r)$
  - Estimated number of tuples: $n_r - size(\sigma_\theta(r))$

连接操作结果大小估计

- **Cartesian product**
  - $r \times s$ contains $n_r * n_s$ tuples
- **Natural join**
  - If $R \cap S = \emptyset$, then $r \bowtie s$ is the same as $r \times s$
  - If $R \cap S$ **is a key for** R, then a tuple of s will join with at most one tuple from r, and $\text{size}(r \bowtie s) \leq n_s$
  - If $R \cap S$ **is a foreign key in** S **referencing** R, the number of tuples in $r \bowtie s$ is exactly the same as the number of tuples in s

- Example: depositor ⋈ customer
  - *customer-name* in *depositor* is a foreign key of *customer*
  - the result has exactly $n_{depositor}$ tuples

连接操作结果大小估计(续)

- Catalog information for join examples:
  - $n_{customer}$ = 10,000, $f_{customer}$ = 25, $b_{customer}$ =10,000/25 = 400
  - $n_{depositor}$ = 5,000, $f_{depositor}$ = 50, $b_{depositor}$ = 5,000/50 = 100
  - **V(customer-name, depositor) = 2,500**, which implies that, on average, each customer has two accounts
- Example: $depositor \bowtie customer$
  - $n_{depositor}$ = 5000 (*customer-name* in *depositor* is a foreign key of *customer*, the result has exactly $n_{depositor}$ tuples)

  - n_r: the number of tuples in a relation r
  - f_r: the number of tuples that fit into one block
  - b_r: the number of blocks of r
  - $V(A, r)$: the number of distinct values that appear in r for attribute A

连接操作结果大小估计(续)

- If $R \cap S = \{A\}$ is not a key for R or S
  - If we assume that every tuple t in R produces tuples in $R \bowtie S$, the number of tuples in $R \bowtie S$ is estimated to be:

$$\frac{n_r * n_s}{V(A,s)}$$

  - If the reverse is true, the estimate obtained will be:

$$\frac{n_r * n_s}{V(A,r)}$$

  - The lower of these two estimates is probably the more accurate one
  - $V(A,r)$: the number of distinct values that appear in r for attribute A

连接操作结果大小估计(续)

- Estimate the size of *depositor* ⋈ *customer* without using the information about foreign keys:
  - V(customer-name, depositor) = 2500, $n_{depositor}$ = 5,000, and V(customer-name, customer) = 10000, $n_{customer}$ = 10,000
  - The two estimates are

    5000 * 10000/2500 = 20,000 and

    5000 * 10000/10000 = 5000
- Choose the lower estimate, which is the same as the computation using foreign keys

  - $V(A, r)$: the number of distinct values that appear in r for attribute A

其他操作结果集大小估计

- **投影**
  - estimated size of $\Pi_A(r) = V(A, r)$
- **聚集**
  - estimated size of $_A g_F(r) = V(A, r)$
- **集合操作**
  - For unions/intersections of selections on the same relation: rewrite and use size estimate for selections
    - E.g., $\sigma_{\theta 1}(r) \cup \sigma_{\theta 2}(r)$ can be rewritten as $\sigma_{\theta 1 \vee \theta 2}(r)$
  - For operations on different relations:
    - estimated size of $r \cup s$ = size of r + size of s
    - estimated size of $r \cap s$ = min{size of r, size of s}
    - estimated size of $r - s = r$
    - All the three estimates may be quite inaccurate, but provide upper bounds for the sizes

其他操作结果集大小估计（续）

- **Outer join**
  - Estimated size of $r ⟕ s$ = size of $r \bowtie s$ + size of r
    - Case of right outer join is symmetric
  - Estimated size of $r ⟗ s$ = size of $r \bowtie s$ + size of r + size of s

Estimation of Distinct Values

- **Selections: $\sigma_\theta(r)$**
  - If θ forces A to take a specified value:
    - If $A = 3$, $V(A, \sigma_\theta(r))$ = 1
  - If θ forces A to take on one of a specified set of values
    - $V(A, \sigma_\theta(r))$ = number of specified values
    - e.g., (A = 1 V A = 3 V A = 4)
  - If the selection condition θ is of the form A op v
    - Estimated $V(A, \sigma_\theta(r)) = V(A, r) * s$, where s is the selectivity of the selection.
  - In all the other cases: use approximate estimate of $\min(V(A, r), n_{\sigma_\theta(r)})$
    - More accurate estimate can be obtained using probability theory

Estimation of Distinct Values (Cont.)

- **Joins: $r \bowtie s$**
  - If all attributes in A are from r
    - Estimated size of $V(A, r \bowtie s) = \min(V(A,r), n_{r \bowtie s})$
  - If A contains attributes A_1 from r and A_2 from s, then
    - $V(A, r \bowtie s) = \min(V(A_1, r) * V(A_2 - A_1, s), V(A_1 - A_2, r) * V(A_2, s), n_{r \bowtie s})$
    - More accurate estimate can be obtained using probability theory
- **Projection**
  - Estimation of distinct values are straightforward for projections
  - They are the same in $\Pi_A(r)$ as in r
- **Aggregation**
  - For $\min(A)$ and $\max(A)$, the number of distinct values can be estimated as $\min(V(A,r), V(G,r))$ where G denotes grouping attributes
  - For other aggregates, assume all values are distinct, and use $V(G,r)$

Outline

- Introduction
- Transformation of Relational Expressions
- Catalog Information for Cost Estimation
- Estimation of Statistics

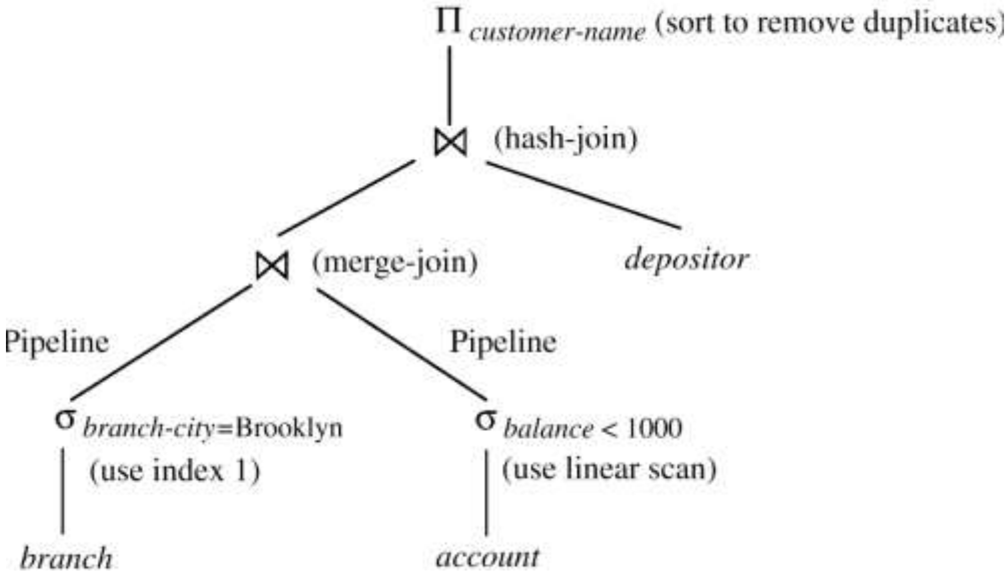
☞ Dynamic Programming for Choosing Evaluation Plans

Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to systematically generate expressions equivalent to the given expression
- Conceptually, generate all equivalent expressions by repeatedly executing the following step until no more expressions can be found
  - Given an expression E, if any sub-expression Es of E matches one side of an equivalence rule, the optimizer generates a new expression where Es is transformed to match the other side of the rule
- The above approach is very expensive in space and time.
  - Space requirements reduced by sharing common sub-expressions for equivalent expressions
  - Time requirements are reduced by not generating all expressions

Evaluation Plan

- **An evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated



Choice of Evaluation Plans

- Must consider the interaction of evaluation techniques when choosing evaluation plans.
  - choosing the cheapest algorithm for each operation independently may not yield best overall algorithm.
    - merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for an outer level aggregation
    - nested-loop join may provide opportunity for pipelining

- Practical query optimizers incorporate elements of the following two broad approaches:
  1. Search all the plans and choose the best plan in a cost-based fashion
  2. Uses heuristics to choose a plan

Cost-based Optimization

- **To find the best join-order for $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$**
  - There are $(2(n-1))!/(n-1)!$ (Refer to Practice Exercises 16.12) different join orders for above expression
  - With $n = 3$, the number is 12

    $r_1 \bowtie (r_2 \bowtie r_3)$, $r_1 \bowtie (r_3 \bowtie r_2)$, $(r_2 \bowtie r_3) \bowtie r_1$, $(r_3 \bowtie r_2) \bowtie r_1$

    $r_2 \bowtie (r_1 \bowtie r_3)$, $r_2 \bowtie (r_3 \bowtie r_1)$, $(r_1 \bowtie r_3) \bowtie r_2$, $(r_3 \bowtie r_1) \bowtie r_2$

    $r_3 \bowtie (r_1 \bowtie r_2)$, $r_3 \bowtie (r_2 \bowtie r_1)$, $(r_1 \bowtie r_2) \bowtie r_3$, $(r_2 \bowtie r_1) \bowtie r_3$

  - With $n = 7$, the number is 665280
  - With $n = 10$, the number is greater than 17.6 billion
- **No need to generate all the join orders**
  - Using dynamic programming
  - The least-cost join order for any subset of $\{r_1, r_2, \ldots, r_n\}$ is computed only once and stored for future use

Cost-based Optimization

- **Given an example** $(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$
- To compute $(r_1 \bowtie r_2 \bowtie r_3)$, there are 12 possible plans
- Join the result of $(r_1 \bowtie r_2 \bowtie r_3)$ with $r_4 \bowtie r_5$, there are 12 possible plans
- So there are totally 12*12=144 possible plans

- On the other hand, we first compute $(r_1 \bowtie r_2 \bowtie r_3)$, find the best plan, then join the result of the best plan of $(r_1 \bowtie r_2 \bowtie r_3)$ with $r_4 \bowtie r_5$
- Then the totaly plans are 12+12=24

Dynamic Programming in Optimization

- **To find best join tree for a set S of n relations**
  - Consider all possible plans of the form: $S_1 \bowtie (S - S_1)$ where S_1 is any non-empty subset of S
  - When the plan for any subset is computed, store it and reuse it when it is required again, instead of re-computing it
  - Recursively compute costs for joining subsets of S to find the cost of each plan. Choose the cheapest.

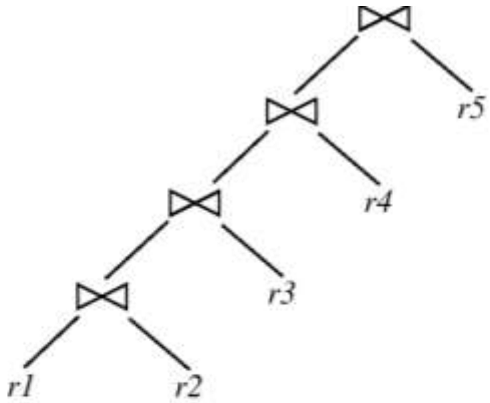
Join Order Optimization Algorithm

Dynamic-programming algorithm

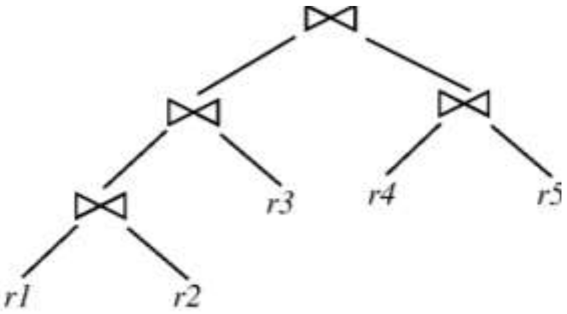
```
procedure findbestplan(S)
    if (bestplan[S].cost ≠ ∞)
            return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost
                    based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
            P1= findbestplan(S1)
            P2= findbestplan(S - S1)
            A = best algorithm for joining results of P1 and P2
            cost = P1.cost + P2.cost + cost of A
            if cost < bestplan[S].cost
                    bestplan[S].cost = cost
                    bestplan[S].plan = "execute P1.plan;
                      execute P2.plan;
                   join results of P1 and P2 using A"
    return bestplan[S]
```

Left Deep Join Trees

- Used by the System R optimizer
- In **left-deep join trees**, the right-hand-side input for each join is a relation, not the result of an intermediate join



(a) Left-deep Join Tree



(b) Non-left-deep Join Tree

Cost of Optimization

- **Complexity of dynamic programming**
  - The time complexity is $O(3^n)$. With **n = 10**, this number is **59000** instead of **17.6 billion**
  - Space complexity is $O(2^n)$
- **Complexity for finding the best left-deep join tree**
  - Consider **n** alternatives with one relation as right-hand side input and the other relations as left-hand side input.
  - Using (recursively computed and stored) least-cost join order for each alternative on left-hand-side, choose the cheapest of the **n** alternatives.
  - If only left-deep trees are considered, time complexity of finding best join order is $O(n2^n)$
  - Space complexity remains at $O(2^n)$
- Cost-based optimization is expensive, but worthwhile for queries on large datasets (typical queries have small n, generally < 10)

Interesting Orders in Cost-based Optimization

- Consider the expression $(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$
- An interesting sort order is a particular sort order of tuples that could be useful for a later operation.
  - Generating the result of $r_1 \bowtie r_2 \bowtie r_3$ sorted on the attributes common with r_4 or r_5 may be useful.
  - Using merge-join to compute $r_1 \bowtie r_2 \bowtie r_3$ may be costlier, but may provide an output sorted in an interesting order.
- Not sufficient to find the best join order for each subset of the set of n given relations; must find the best join order for each subset, for each interesting sort order of the join result for that subset
  - Simple extension of earlier dynamic programming algorithms
  - Usually, the number of interesting orders is quite small and doesn't affect time/space complexity significantly

Heuristic Optimization

- Cost-based optimization is expensive, even with dynamic programming.
- Systems may use *heuristics* to reduce the number of choices that must be made in a cost-based fashion.
- Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:
  - Perform selection early (reduces the number of tuples)
  - Perform projection early (reduces the number of attributes)
  - Perform most restrictive selection and join operations before other similar operations.
  - Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

Steps in Typical Heuristic Optimization

1. Deconstruct conjunctive selections into a sequence of single selection operations (Equiv. rule 1.).
   $$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$
2. Move selection operations down the query tree for the earliest possible execution (Equiv. rules 2, 7a, 7b, 11).
3. Execute first those selection and join operations that will produce the smallest relations (Equiv. rule 6).
4. Replace Cartesian product operations that are followed by a selection condition by join operations (Equiv. rule 4a).
5. Deconstruct and move as far down the tree as possible lists of projection attributes, creating new projections where needed (Equiv. rules 3, 8a, 8b, 12).
6. Identify those subtrees whose operations can be pipelined, and execute them using pipelining.

Assignments

- Practice Exercises
  - 16.5, 16.6, 16.7
- Exercises
  - 16.16
- DDL: 12:59pm, May 29, 2024

End of Lecture 16