



查询优化

Query Optimization

李文根/Wengen Li

Email: lwengen@tongji.edu.cn

先进数据与机器智能系统实验室

Advanced Data and Machine Intelligence Systems (ADMIS) Lab

<https://admis-tongji.github.io>

同济大学 计算机科学与技术学院

2026年05月

- **Part 0: Overview**
 - Ch1: Introduction
- **Part 1 Relational Languages**
 - Ch2: Relational model
 - Ch3: Introduction to SQL
 - Ch4: Intermediate SQL
 - Ch5: Advanced SQL
- **Part 2 Database Design**
 - Ch6: Database design via E-R model
 - Ch7: Relational database design
- **Part 3 Application Design & Development**
 - Ch8: Complex data types
 - Ch9: Application development
- **Part 4 Big Data Analytics**
 - Ch10: Big data
 - Ch11: Data analytics
- **Part 5 Storage Management & Indexing**
 - Ch12: Physical storage systems
 - Ch13: Data storage structures
 - Ch14: Indexing
- **Part 6 Query Processing & Optimization**
 - Ch15: Query processing
 - **Ch16: Query optimization**
- **Part 7 Transaction Management**
 - Ch17: Transactions
 - Ch18: Concurrency control
 - Ch19: Recovery system
- **Part 8 Parallel & Distributed Database**
 - Ch20: Database system architecture
 - Ch21-23: Parallel & distributed storage, query processing & transaction processing
- **Advanced topics**
 - DB Platform: **OceanBase**, MongoDB, Neo4J
 - RAG, Multimodal retrieval, ...

- **概述**
- **关系表达式的转换**
- **表达式执行代价估计**
- **执行计划选择**
- **物化视图***
- **查询优化中的高级话题***

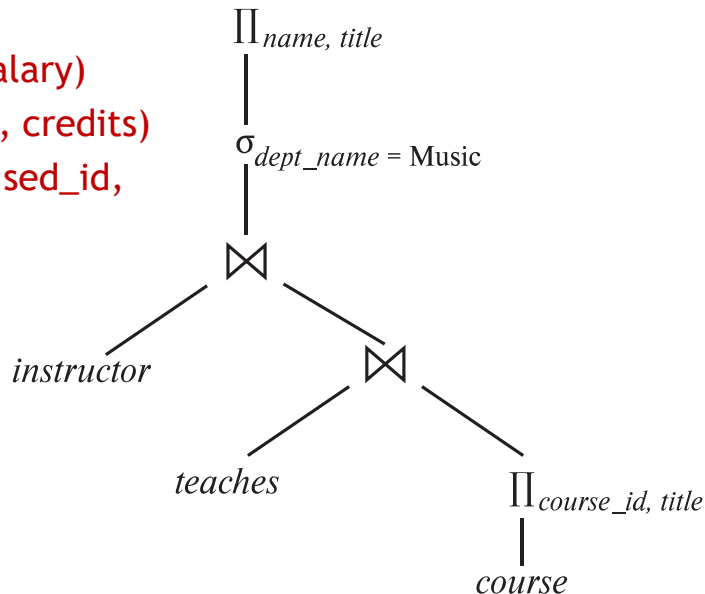
- Alternative ways to execute (执行) a given query
 - equivalent expressions
 - different algorithms for each operation
- The cost difference between a good way and a bad way of executing a query can be enormous
- **Estimate the cost of operations**
 - depend on the statistical information about **relations** which the database should maintain
 - need to estimate the statistics for **intermediate results** to compute the cost of complex expressions

▶ 查询优化概述 (续)

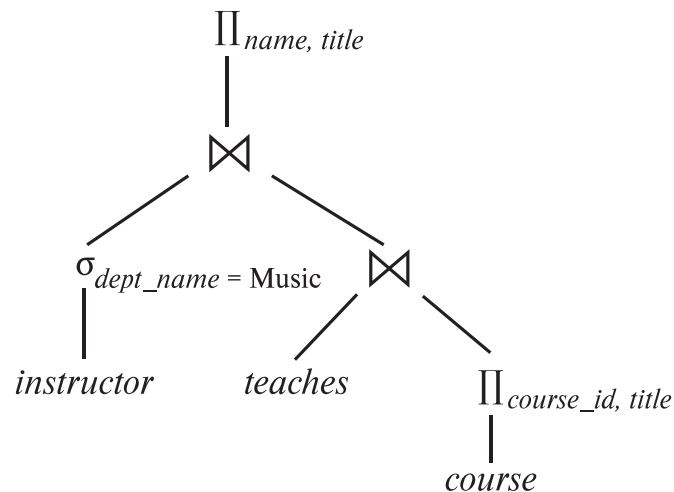


- **查询：** 找出Music系所有教师的名字以及每位教师所教授课程的名称

instructor (ID, name, salary)
course (course_id, title, credits)
teaches (ID, course_id, sed_id,
semester, year)



(a) Initial expression tree

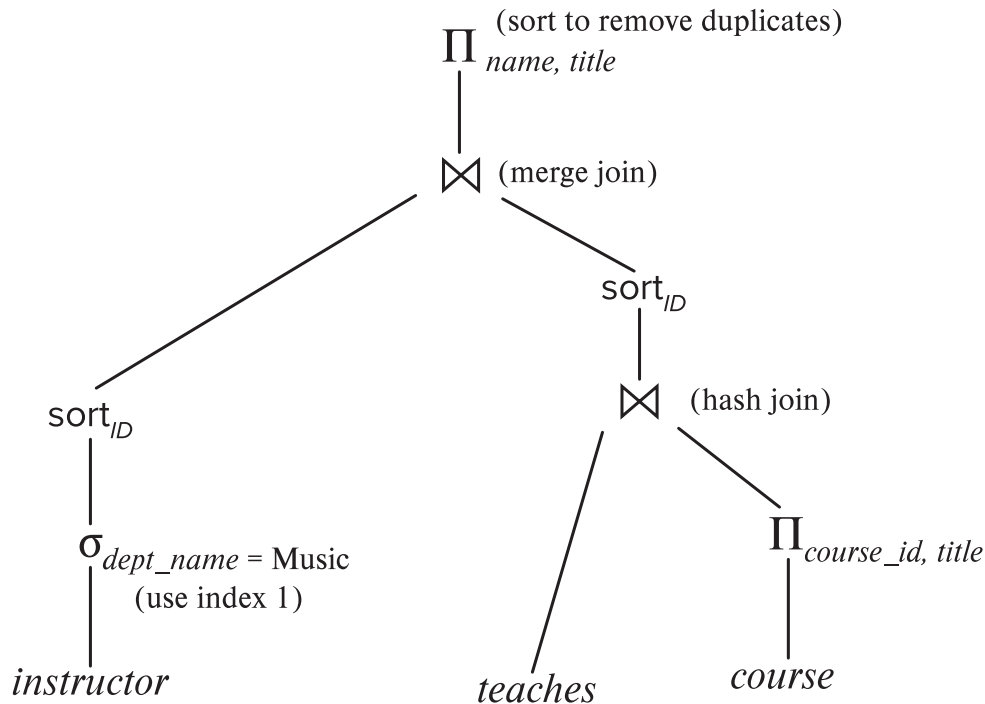


(b) Transformed expression tree

▶ 查询优化概述 (续)



- 执行计划需明确每个运算使用的算法以及运算之间的执行如何协调



- **Cost-based Optimization, 基于代价的优化**
 - **步骤1**: 产生逻辑上与给定表达式等价的表达式
 - 使用等价规则
 - **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
 - **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

- 概述
- **关系表达式的转换**
- 表达式执行代价估计
- 执行计划选择
- 物化视图*
- 查询优化中的高级话题*

- **Equivalence of two relational algebra expressions**
 - Generate the same set of tuples on every legal database instance
 - Note: the order of tuples is irrelevant
- **Equivalence rule (等价规则)**
 - The expressions of two forms are equivalent
 - Can replace expression of first form by the second, or vice versa

► 等价规则



- **规则1:** 合取选择运算可分解为单个选择运算的序列

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

- **规则2:** 选择运算满足交换律 (commutative)

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

- **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

$$\Pi_{t_1}(\Pi_{t_2}(\dots(\Pi_{t_n}(E))\dots)) = \Pi_{t_1}(E)$$

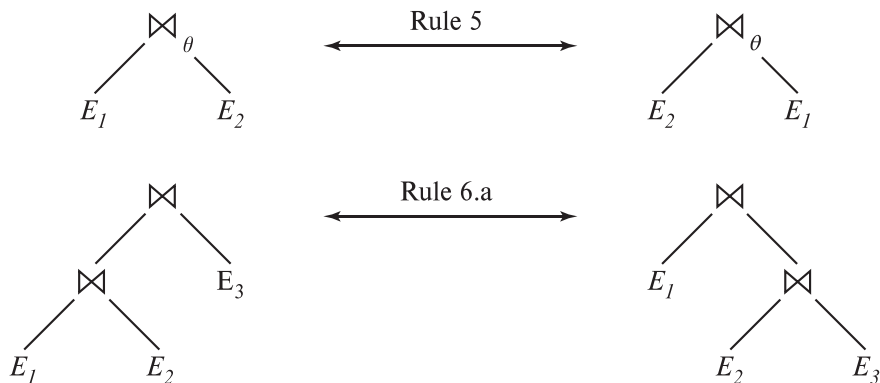
- **规则4:** 选择操作可以与笛卡尔积以及 θ 连接相结合

$$\begin{aligned}\sigma_{\theta}(E_1 \times E_2) &= E_1 \bowtie_{\theta} E_2 \\ \sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) &= E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2\end{aligned}$$

▶ 等价规则 (续)



- **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 连接满足下列形式的结合律:
 - $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$, 要求其中 θ_2 只涉及 E_2 和 E_3 的属性



▶ 等价规则 (续)



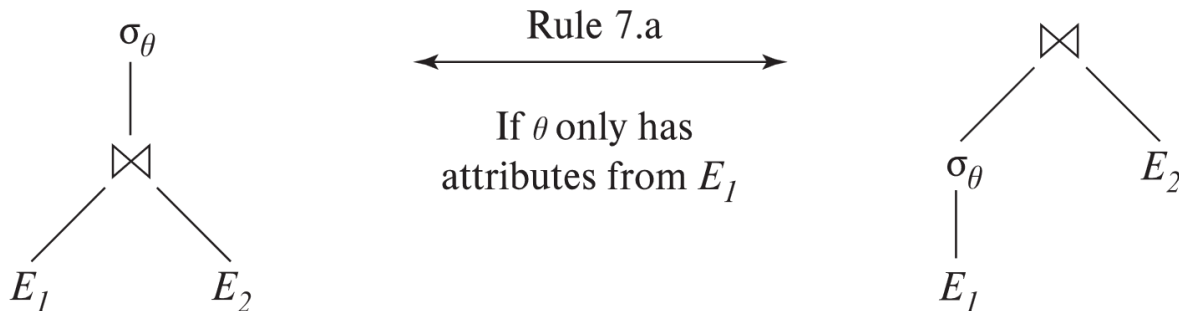
• 规则7: 选择操作在下面两个条件下对 θ 连接满足分配律

- 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. 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:

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



▶ 等价规则 (续)



- **规则8:** 令 L_1 、 L_2 分别代表 E_1 、 E_2 的**属性子集**，投影操作在下列条件下对 θ 连接满足分配率
 - 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. 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

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

- **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 difference is not commutative)
- **规则10:** 集合的并和交满足结合律
 - $(E_1 \cup E_2) \cup U_3 = E_1 \cup (E_2 \cup E_3)$
 - $(E_1 \cap E_2) \cap U_3 = E_1 \cap (E_2 \cap E_3)$
- **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 \cup and \cap in place of $-$
 - $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$
 - similarly for \cap in place of $-$, **but not for \cup ?**
- **规则12:** 投影对并的分配律
 - $\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$

▶ 例1: Selections First



- Find the names of all instructors in 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

$$\Pi_{name, title}((\sigma_{dept_name='Music'}(instructor)) \bowtie (teaches \bowtie \Pi_{course_id, title}(course)))$$

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:

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

▶ 例2: Multiple Transformations



- Find the names of all instructors in Music department who have taught a course in 2021, along with the titles of the courses that they taught

$$\Pi_{name, title}(\sigma_{dept_name= "Music" \wedge year= 2021}(instructor \bowtie (teaches \bowtie \Pi_{course_id, title} (course))))$$

- Rule 6a:

$$\Pi_{name, title}(\sigma_{dept_name= "Music" \wedge year= 2021}((instructor \bowtie teaches) \bowtie \Pi_{course_id, title} (course)))$$

- Rule 7a:

$$\Pi_{name, title}((\sigma_{dept_name= "Music" \wedge year= 2021}(instructor \bowtie teaches)) \bowtie \Pi_{course_id, title} (course))$$

- Rule1 & 7a:

$$\Pi_{name, title}((\sigma_{dept_name= "Music"}(instructor) \bowtie \sigma_{year= 2021}(teaches)) \bowtie \Pi_{course_id, title} (course))$$

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

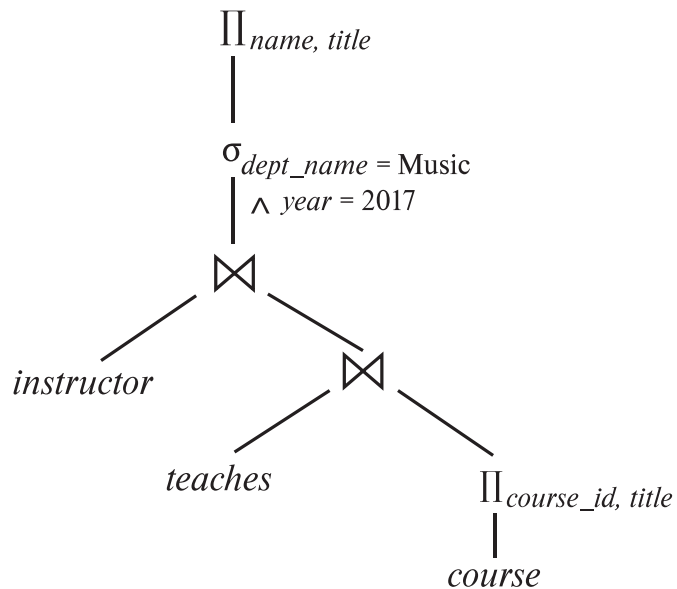
- a. 当选择条件 θ_0 中的所有属性只涉及参与连接的表达式之一(如 E_1)时:

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

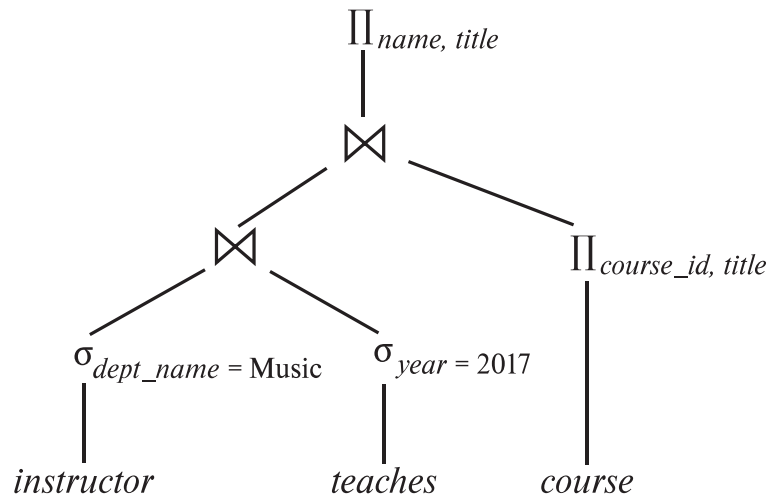
规则1: 合取选择运算可分解为单个选择运算的序列

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

▶ 例2: Multiple Transformations



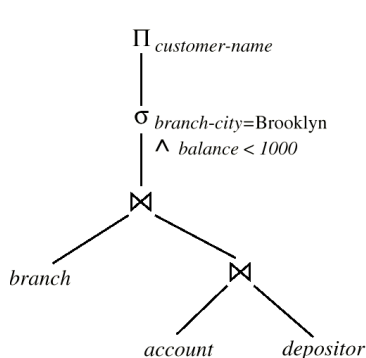
(a) Initial expression tree



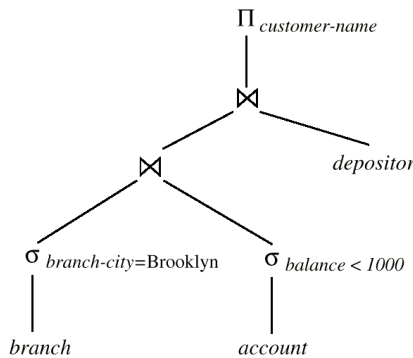
(b) Tree after multiple transformations

- **Query:** Find the names of all the customers who have an account with balance less than \$1000 at the Brooklyn branch
 - $\Pi_{CN}(\sigma_{BC="Brooklyn" \wedge balance < 1000}(branch \bowtie (account \bowtie depositor)))$
 - CN: customer name, BC: branch city
- **Task:** Give one equivalent expression with better execution performance
- **One solution:**

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



(a) Initial Expression Tree



(b) Tree After Multiple Transformations

▶ 连接顺序



- 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

- Consider the expression

$$\Pi_{name, title}(\sigma_{dept_name= 'Music'}(instructor) \bowtie teaches) \bowtie \Pi_{course_id, title}(course))$$

- **Solution A**

- compute $teaches \bowtie \Pi_{course_id, title}(course)$ first, and join the result with $\sigma_{dept_name= 'Music'}(instructor)$
- the result of the first join is likely to be a large relation

- **Solution B**

- compute $\sigma_{dept_name= 'Music'}(instructor) \bowtie teaches$ first
- only a small fraction of instructors are likely to be from the Music department

- 概述
- 关系表达式的转换
- **表达式执行代价估计**
- 执行计划选择
- 物化视图*
- 查询优化中的高级话题*

• 关系(表)的统计信息

- n_r : the number of tuples in relation r
- b_r : the number of blocks of r
- s_r : the size of a tuple of 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 , i.e., the size of $\Pi_A(r)$
- If the tuples of r are stored together physically in a file, then: $b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$

- 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 (e.g., B⁺-tree) on attribute A of relation r , $HT_i = \lceil \log_{F_i}(V(A, r)) \rceil$
 - 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

- **Recall that**
 - 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 execution
 - It is assumed that all block transfers have the same cost

▶ 简单选择操作结果大小估计



- $\sigma_{A=a}(r)$
 - 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
- $\sigma_{A \leq v}(r)$
 - 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 $n_r/2$

► 复杂选择操作结果大小估计



- **Selectivity (中选率) of the 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: $\sigma_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n}(r)$**

- Estimated number of tuples:

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

- **析取: $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$**

- Estimated number of tuples:

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

- **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 , i.e., n_s
 - E.g., for depositor \bowtie customer, customer-name in depositor is a foreign key of customer, and the result has exactly $n_{depositor}$ tuples

▶ 连接操作结果大小估计(续)



- Catalog information for join examples:
 - $n_{customer} = 10,000$, $f_{customer} = 25$, $b_{customer} = 10,000/25 = 400$
 - $n_{depositor} = 5,000$, $f_{depositor} = 50$, $b_{depositor} = 5,000/50 = 100$
 - $V(\text{customer-name, depositor}) = 2,500$, which implies that, on average, each customer has two accounts
- E.g., $depositor \bowtie customer$
 - $n_{depositor} = 5000$

▶ 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:

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

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

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

- The lower of these two estimates is probably the more accurate one

▶ 连接操作结果大小估计(续)



- Estimate the size of depositor \bowtie customer without using the information about foreign keys:
 - $V(\text{customer-name, depositor}) = 2500$, and
 $V(\text{customer-name, customer}) = 10000$
 - The two estimates are
 $5000 * 10000/2500 = 20,000$ and
 $5000 * 10000/10000 = 5000$
- Choose the lower estimate, which is the same as the earlier estimation using foreign keys

- 投影

- Estimated size of $\Pi_A(r) = V(A, r)$

- 聚集

- Estimated size of $Ag_F(r) = V(A, r)$

- 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 $r \cup s = \text{size of } r + \text{size of } s$
 - estimated size of $r \cap s = \min\{\text{size of } r, \text{size of } s\}$
 - estimated size of $r - s = \text{size of } r$
 - 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 $r \bowtie s = \text{size of } r \bowtie s + \text{size of } r$
 - Case of right outer join is symmetric
- Estimated size of $r \bowtie \sqsubset s = \text{size of } r \bowtie s + \text{size of } r + \text{size of } s$

► 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 one of a specified set of values
 - $V(A, \sigma_\theta(r)) =$ the number of specified values
 - e.g., $(A = 1 \vee A = 3 \vee A = 4)$
 - If the selection condition θ is of the form $A \text{ op } v$
 - 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 that all values are distinct, and use $V(G, r)$

- 概述
- 关系表达式的转换
- 表达式执行代价估计
- **执行计划选择**
- 物化视图*
- 查询优化中的高级话题*

- **Equivalent expression generation**

- Query optimizers use equivalence rules to systematically generate expressions equivalent to the given expression
- 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 E_s of E matches one side of an equivalence rule, the optimizer generates a new expression where E_s is transformed to match the other side of the rule

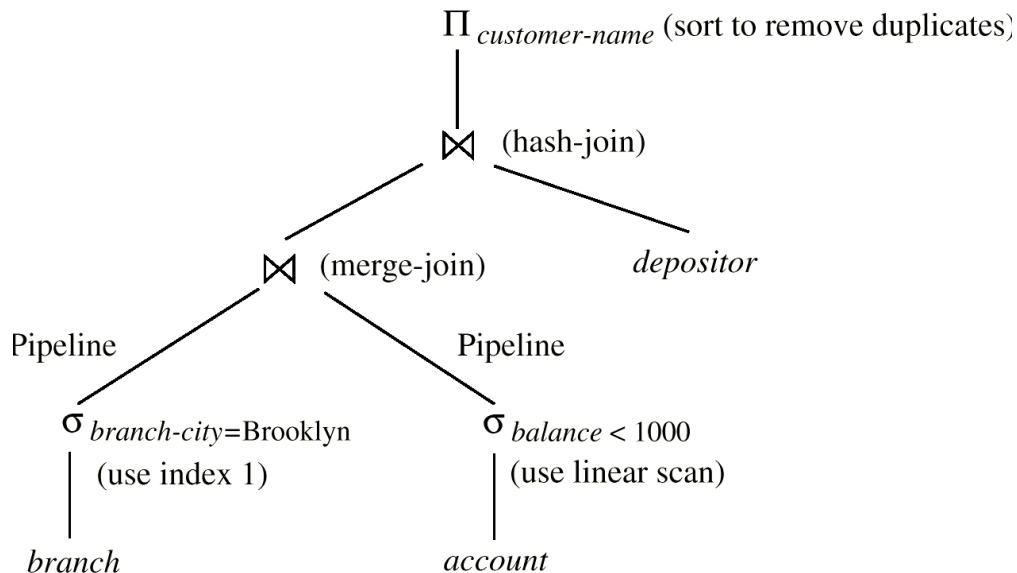
- **Issues**

- The above approach is very expensive in space and time
 - Space requirements can be reduced by sharing **common sub-expressions** for equivalent expressions
 - Time requirements can be reduced by not generating all expressions

▶ 执行计划



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



- **Two general approaches**
 - Search all the plans and choose the best plan in a cost-based fashion
 - Uses heuristics to choose a plan
- **Interaction of evaluation operations**
 - Choosing the cheapest algorithm for each operation independently may not yield the 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

- **To find the best join-order for $r_1 \bowtie r_2 \bowtie \dots \bowtie r_n$**
 - There are $(2(n-1))/(n-1)!$ different join orders for above expression (cf. Exercise 16.12)
 - E.g., 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 665,280
 - 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 $\{r_1, r_2, \dots, r_n\}$ is computed only once and stored for future use

- **To find the 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.

procedure findbestplan(*S*)

 if (*bestplan*[*S*].*cost* $\neq \infty$)

 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 *S*1 of *S* such that *S*1 $\neq S$

*P*1 = findbestplan(*S*1)

*P*2 = findbestplan(*S* - *S*1)

A = best algorithm for joining results of *P*1 and *P*2

cost = *P*1.*cost* + *P*2.*cost* + cost of *A*

 if *cost* < *bestplan*[*S*].*cost*

bestplan[*S*].*cost* = *cost*

bestplan[*S*].*plan* = “execute *P*1.*plan*;

 execute *P*2.*plan*;

 join results of *P*1 and *P*2 using *A*”

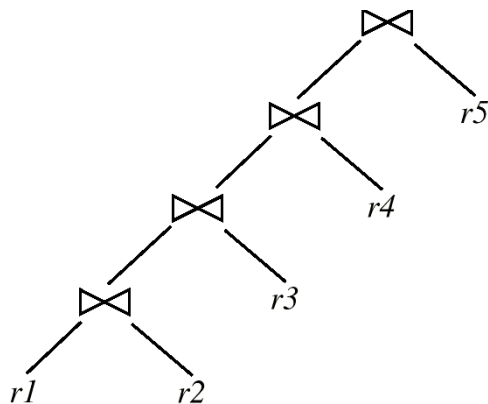
 return *bestplan*[*S*]

Dynamic-programming algorithm

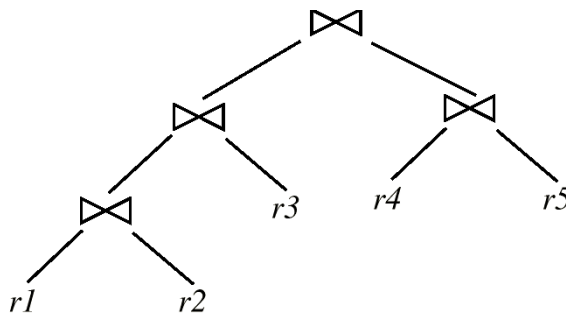
▶ 左深连接树



- 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

- **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)

▶ 启发式优化 (Heuristic Optimization)



- Cost-based optimization is expensive, even with dynamic programming
- DBMS 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 (reduce the number of tuples)
 - Perform projection early (reduce 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

▶ 启发式优化的主要步骤



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

- 16.5 (第7版)
 - Canvas上提交, 单个PDF文件

16.5 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A , C , and E , respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$, and give an efficient strategy for computing the join.