CS315: DATABASE SYSTEMS
QUERY OPTIMIZATION

Arnab Bhattacharya

arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur

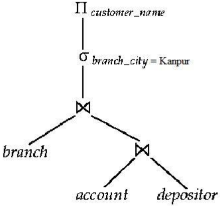
http://web.cse.iitk.ac.in/~cs315/

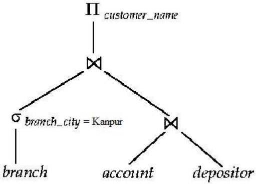
2nd semester, 2019-20

Tue, Wed 12:00-13:15

Evaluation Plan

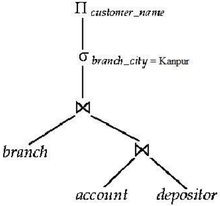
- Equivalent expressions provide alternate ways of executing a query

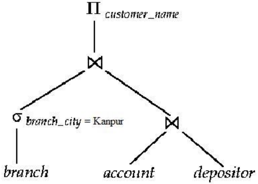




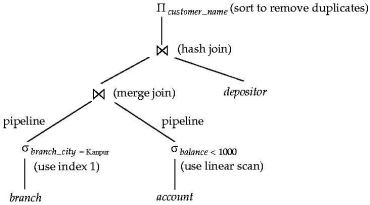
Evaluation Plan

- Equivalent expressions provide alternate ways of executing a query





- An evaluation plan also specifies the algorithms



- Cost-based query optimization

Equivalent Expressions

- Two relational algebra expressions are equivalent if they generate the same set of output tuples on every *legal* input relation
  - Order of tuples does not matter
  - For SQL, same multiset of output tuples

Equivalent Expressions

- Two relational algebra expressions are equivalent if they generate the same set of output tuples on every *legal* input relation
  - Order of tuples does not matter
  - For SQL, same multiset of output tuples
- Equivalence rule: specifies which expressions are equivalent
- Equivalent expressions are systematically generated by repeatedly applying equivalence rules and replacing one form by another
- Evaluation plans must account for all algorithms used
  - Merge join may be costlier than hash join, but since it provides a sorted output, a higher level aggregation will be faster

Equivalence Rules

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

Equivalence Rules

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

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\ldots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
6. $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\ldots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
6. $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$
7. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\ldots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
6. $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$
7. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
8. $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\ldots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
6. $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$
7. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
8. $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$
9. $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$
    - θ_2 involves attributes from only E_2 and E_3

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
6. $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$
7. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
8. $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$
9. $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$
   - θ_2 involves attributes from only E_2 and E_3
10. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_2} E_2$
    - θ_1 involves attributes from only E_1

Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
6. $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$
7. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
8. $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$
9. $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$
   - θ_2 involves attributes from only E_2 and E_3
10. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_2} E_2$
    - θ_1 involves attributes from only E_1
11. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2))$
    - θ_1 and θ_2 involve attributes from only E_1 and E_2 respectively

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2
13. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - L_3 involves attributes from E_1 but is disjoint with L_1
    - L_4 involves attributes from E_2 but is disjoint with L_2
    - θ involves L_3 and L_4

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2
13. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - L_3 involves attributes from E_1 but is disjoint with L_1
    - L_4 involves attributes from E_2 but is disjoint with L_2
    - θ involves L_3 and L_4
14. $E_1 \oplus E_2 = E_2 \oplus E_1$
    - $\oplus$ is any of $\cup$ and $\cap$

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2
13. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - L_3 involves attributes from E_1 but is disjoint with L_1
    - L_4 involves attributes from E_2 but is disjoint with L_2
    - θ involves L_3 and L_4
14. $E_1 \oplus E_2 = E_2 \oplus E_1$
    - $\oplus$ is any of $\cup$ and $\cap$
15. $(E_1 \oplus E_2) \oplus E_3 = E_1 \oplus (E_2 \oplus E_3)$
    - $\oplus$ is any of $\cup$ and $\cap$

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2
13. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - L_3 involves attributes from E_1 but is disjoint with L_1
    - L_4 involves attributes from E_2 but is disjoint with L_2
    - θ involves L_3 and L_4
14. $E_1 \oplus E_2 = E_2 \oplus E_1$
    - $\oplus$ is any of $\cup$ and $\cap$
15. $(E_1 \oplus E_2) \oplus E_3 = E_1 \oplus (E_2 \oplus E_3)$
    - $\oplus$ is any of $\cup$ and $\cap$
16. $\sigma_\theta(E_1 \oplus E_2) = \sigma_\theta(E_1) \oplus \sigma_\theta(E_2)$
    - $\oplus$ is any of $\cup$, $\cap$ and $-$

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2
13. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - L_3 involves attributes from E_1 but is disjoint with L_1
    - L_4 involves attributes from E_2 but is disjoint with L_2
    - θ involves L_3 and L_4
14. $E_1 \oplus E_2 = E_2 \oplus E_1$
    - $\oplus$ is any of $\cup$ and $\cap$
15. $(E_1 \oplus E_2) \oplus E_3 = E_1 \oplus (E_2 \oplus E_3)$
    - $\oplus$ is any of $\cup$ and $\cap$
16. $\sigma_\theta(E_1 \oplus E_2) = \sigma_\theta(E_1) \oplus \sigma_\theta(E_2)$
    - $\oplus$ is any of $\cup$, $\cap$ and $-$
17. $\sigma_\theta(E_1 \oplus E_2) = \sigma_\theta(E_1) \oplus E_2$
    - $\oplus$ is any of $\cap$ and $-$

Equivalence Rules (contd.)

12. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - θ involves only L_1 and L_2
13. $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$
    - L_1 and L_2 involve attributes from only E_1 and E_2 respectively
    - L_3 involves attributes from E_1 but is disjoint with L_1
    - L_4 involves attributes from E_2 but is disjoint with L_2
    - θ involves L_3 and L_4
14. $E_1 \oplus E_2 = E_2 \oplus E_1$
    - $\oplus$ is any of $\cup$ and $\cap$
15. $(E_1 \oplus E_2) \oplus E_3 = E_1 \oplus (E_2 \oplus E_3)$
    - $\oplus$ is any of $\cup$ and $\cap$
16. $\sigma_\theta(E_1 \oplus E_2) = \sigma_\theta(E_1) \oplus \sigma_\theta(E_2)$
    - $\oplus$ is any of $\cup$, $\cap$ and $-$
17. $\sigma_\theta(E_1 \oplus E_2) = \sigma_\theta(E_1) \oplus E_2$
    - $\oplus$ is any of $\cap$ and $-$
18. $\Pi_L(E_1 \cup E_2) = \Pi_L(E_1) \cup \Pi_L(E_2)$

Example Schema

- branch (bname, bcity, assets)
- customer (cname, cstreet, ccity)
- account (ano, bname, bal)
- loan (lno, bname, amt)
- depositor (cname, ano)
- borrower (cname, lno)

Example

- Find names of customers having an account at “Kanpur” city

Example

- Find names of customers having an account at “Kanpur” city
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch \bowtie (account \bowtie depositor)))$

Example

- Find names of customers having an account at “Kanpur” city
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch \bowtie (account \bowtie depositor)))$
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch) \bowtie (account \bowtie depositor)))$

Example

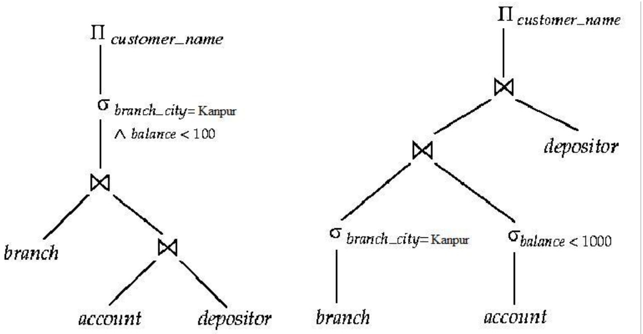
- Find names of customers having an account at "Kanpur" city
  - $\Pi_{cname}(\sigma_{bcity=\text{"Kanpur"}}(branch \bowtie (account \bowtie depositor)))$
  - $\Pi_{cname}(\sigma_{bcity=\text{"Kanpur"}}(branch) \bowtie (account \bowtie depositor)))$
- Find names of customers with an account in "Kanpur" with balance more than 1000

Example

- Find names of customers having an account at "Kanpur" city
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch \bowtie (account \bowtie depositor)))$
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch) \bowtie (account \bowtie depositor)))$
- Find names of customers with an account in "Kanpur" with balance more than 1000
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”} \wedge bal>1000}(branch \bowtie (account \bowtie depositor)))$

Example

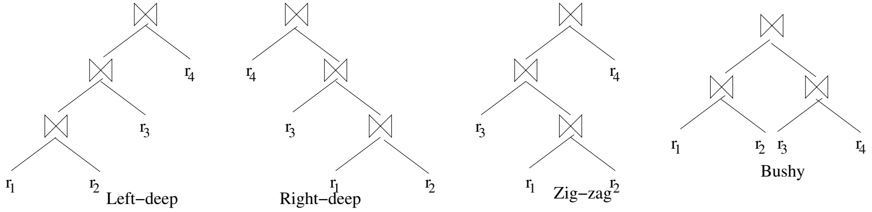
- Find names of customers having an account at “Kanpur” city
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch \bowtie (account \bowtie depositor)))$
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch) \bowtie (account \bowtie depositor)))$
- Find names of customers with an account in “Kanpur” with balance more than 1000
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”} \wedge bal>1000}(branch \bowtie (account \bowtie depositor)))$
  - $\Pi_{cname}(\sigma_{bcity=\text{“Kanpur”}}(branch) \bowtie \sigma_{bal>1000}(account)) \bowtie depositor)))$



(a) Initial expression tree (b) Tree after multiple transformations

Join Trees

- Left-deep join tree
  - Right side of each join is a single relation, and not an intermediate result of join of two or more relations
- Similarly, right-deep join trees can be defined
- A tree where at least one child of an internal node is a single relation is called a zig-zag tree
- A general tree is also called a bushy tree



Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations

Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations
- Number of ways leaves can be placed
  - If order does not matter, it is $n!$
  - If order matters, it is 1

Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations
- Number of ways leaves can be placed
  - If order does not matter, it is $n!$
  - If order matters, it is 1
- Number of tree configurations
  - Left-deep and right-deep trees:

Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations
- Number of ways leaves can be placed
  - If order does not matter, it is $n!$
  - If order matters, it is 1
- Number of tree configurations
  - Left-deep and right-deep trees: 1
  - Zig-zag trees:

Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations
- Number of ways leaves can be placed
  - If order does not matter, it is $n!$
  - If order matters, it is 1
- Number of tree configurations
  - Left-deep and right-deep trees: 1
  - Zig-zag trees: 2^{n-2}

Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations
- Number of ways leaves can be placed
  - If order does not matter, it is $n!$
  - If order matters, it is 1
- Number of tree configurations
  - Left-deep and right-deep trees: 1
  - Zig-zag trees: 2^{n-2}
    - Place 2 leaves; rest can be either right or left
  - Bushy trees:

Number of Join Orders

- Number of join orders is number of ways relations (leaves) can be placed times the number of different tree configurations
- Number of ways leaves can be placed
  - If order does not matter, it is $n!$
  - If order matters, it is 1
- Number of tree configurations
  - Left-deep and right-deep trees: 1
  - Zig-zag trees: 2^{n-2}
    - Place 2 leaves; rest can be either right or left
  - Bushy trees: $(n-1)^{\text{th}}$ Catalan number $\frac{(2n-2)!}{n!(n-1)!}$
    - $C(1) = 1$; $C(n) = \sum_{i=1}^{n-1} C(i).C(n-i)$

Algorithm for Join Order

- Consider set S as the join of n relations
- S can be represented as $S_1 \bowtie (S - S_1)$ for any non-empty proper subset $S_1 \subset S$
- Choose S_1 that minimizes the overall cost
- Recursively choose the best plan for S_1 and $(S - S_1)$

Algorithm for Join Order

- Consider set S as the join of n relations
- S can be represented as $S_1 \bowtie (S - S_1)$ for any non-empty proper subset $S_1 \subset S$
- Choose S_1 that minimizes the overall cost
- Recursively choose the best plan for S_1 and $(S - S_1)$
- Employ dynamic programming solution
- Store solutions of subsets

Algorithm for Join Order

- Consider set S as the join of n relations
- S can be represented as $S_1 \bowtie (S - S_1)$ for any non-empty proper subset $S_1 \subset S$
- Choose S_1 that minimizes the overall cost
- Recursively choose the best plan for S_1 and $(S - S_1)$
- Employ dynamic programming solution
- Store solutions of subsets
- Interesting sort order: Particular order of records that are useful later
  - Example: Merge join produces tuples in sorted order which makes later merge joins faster
  - Example: Sorted order makes later grouping and aggregation faster
- Algorithm should find the best subset for *each* interesting sort order

Heuristics in Query Optimization

- Perform selections early
  - Reduces size of relations

Heuristics in Query Optimization

- Perform selections early
  - Reduces size of relations
- Perform projections early
  - Reduces number of attributes

Heuristics in Query Optimization

- Perform selections early
  - Reduces size of relations
- Perform projections early
  - Reduces number of attributes
- Perform joins earlier than cross products
  - Produces smaller relations

Heuristics in Query Optimization

- Perform selections early
  - Reduces size of relations
- Perform projections early
  - Reduces number of attributes
- Perform joins earlier than cross products
  - Produces smaller relations
- Perform only left-deep (or right-deep) joins
  - Number of trees is less
  - Easier to incorporate other operations
  - Hybrid hash joins can be used

Heuristics in Query Optimization

- Perform selections early
    - Reduces size of relations
- Perform projections early
    - Reduces number of attributes
- Perform joins earlier than cross products
    - Produces smaller relations
- Perform only left-deep (or right-deep) joins
    - Number of trees is less
    - Easier to incorporate other operations
    - Hybrid hash joins can be used
- Perform semantic optimizations
    - Find all employees earning more than their manager
    - May use domain knowledge to return empty result directly

Statistics

- For each relation r
  - Number of tuples n_r
  - Number of blocks b_r
  - Blocking factor, i.e., number of tuples that fit in a block $f_r = \lfloor n_r / b_r \rfloor$
  - Size of a tuple l_r

Statistics

- For each relation r
  - Number of tuples n_r
  - Number of blocks b_r
  - Blocking factor, i.e., number of tuples that fit in a block $f_r = \lfloor n_r / b_r \rfloor$
  - Size of a tuple l_r
- For each attribute A of a relation r
  - Number of distinct values, i.e., size of $\Pi_A(r)$: $v_r(A)$
  - Minimum $\min_r(A)$ and maximum $\max_r(A)$
  - Histogram of values: equi-width and equi-depth

Statistics

- For each relation r
  - Number of tuples n_r
  - Number of blocks b_r
  - Blocking factor, i.e., number of tuples that fit in a block $f_r = \lfloor n_r / b_r \rfloor$
  - Size of a tuple l_r
- For each attribute A of a relation r
  - Number of distinct values, i.e., size of $\Pi_A(r)$: $v_r(A)$
  - Minimum $\mathsf{min}_r(A)$ and maximum $\mathsf{max}_r(A)$
  - Histogram of values: equi-width and equi-depth
- For each index
  - Number of levels of the index
  - Number of leaves

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key
- $\sigma_{A \leq x}(r)$

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key
- $\sigma_{A \leq x}(r)$
  - 0 if $x < \min_r(A)$ and n_r if $x > \max_r(A)$
  - $n_r . \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
  - Estimate is better when histograms are available

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key
- $\sigma_{A \leq x}(r)$
  - 0 if $x < \min_r(A)$ and n_r if $x > \max_r(A)$
  - $n_r . \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
  - Estimate is better when histograms are available
- $\sigma_{A<x}(r)$
  - $size(\sigma_{A \leq x}(r)) - size(\sigma_{A=x}(r))$

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key
- $\sigma_{A \leq x}(r)$
  - 0 if $x < \min_r(A)$ and n_r if $x > \max_r(A)$
  - $n_r . \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
  - Estimate is better when histograms are available
- $\sigma_{A<x}(r)$
  - $size(\sigma_{A \leq x}(r)) - size(\sigma_{A=x}(r))$
- $\sigma_{A \geq x}(r)$
  - $n_r - size(\sigma_{A<x}(r))$

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key
- $\sigma_{A \leq x}(r)$
  - 0 if $x < \min_r(A)$ and n_r if $x > \max_r(A)$
  - $n_r . \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
  - Estimate is better when histograms are available
- $\sigma_{A<x}(r)$
  - $size(\sigma_{A \leq x}(r)) - size(\sigma_{A=x}(r))$
- $\sigma_{A \geq x}(r)$
  - $n_r - size(\sigma_{A<x}(r))$
- $\sigma_{A>x}(r)$
  - $n_r - size(\sigma_{A \leq x}(r))$

Estimating Size of Selections

- $\sigma_{A=x}(r)$
  - $n_r / v_r(A)$
  - 1 if key
- $\sigma_{A \leq x}(r)$
  - 0 if $x < \min_r(A)$ and n_r if $x > \max_r(A)$
  - $n_r . \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
  - Estimate is better when histograms are available
- $\sigma_{A<x}(r)$
  - $size(\sigma_{A \leq x}(r)) - size(\sigma_{A=x}(r))$
- $\sigma_{A \geq x}(r)$
  - $n_r - size(\sigma_{A<x}(r))$
- $\sigma_{A>x}(r)$
  - $n_r - size(\sigma_{A \leq x}(r))$
- $\sigma_{A \neq x}(r)$
  - $n_r - size(\sigma_{A=x}(r))$

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r
- Conjunction: $\sigma_{\theta_1 \wedge \cdots \wedge \theta_k}(r)$
  - If conditions are independent, estimate is

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r
- Conjunction: $\sigma_{\theta_1 \wedge \cdots \wedge \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r \times \frac{s_1 \times \cdots \times s_k}{n_r^k}$$

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r
- Conjunction: $\sigma_{\theta_1 \wedge \cdots \wedge \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r \times \frac{s_1 \times \cdots \times s_k}{n_r^k}$$

- Disjunction: $\sigma_{\theta_1 \vee \cdots \vee \theta_k}(r)$
  - If conditions are independent, estimate is

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r
- Conjunction: $\sigma_{\theta_1 \wedge \cdots \wedge \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r \times \frac{s_1 \times \cdots \times s_k}{n_r^k}$$

- Disjunction: $\sigma_{\theta_1 \vee \cdots \vee \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r - n_r \times \frac{(n_r - s_1) \times \cdots \times (n_r - s_k)}{n_r^k}$$

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r
- Conjunction: $\sigma_{\theta_1 \wedge \cdots \wedge \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r \times \frac{s_1 \times \cdots \times s_k}{n_r^k}$$

- Disjunction: $\sigma_{\theta_1 \vee \cdots \vee \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r - n_r \times \frac{(n_r - s_1) \times \cdots \times (n_r - s_k)}{n_r^k}$$

- Negation: $\sigma_{\neg\theta}(r)$
  - Estimate is

Estimating Size of Complex Selections

- Selectivity of a condition θ is the probability that a tuple in r satisfies θ
- If s number of tuples satisfy, selectivity is s/n_r
- Conjunction: $\sigma_{\theta_1 \wedge \cdots \wedge \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r \times \frac{s_1 \times \cdots \times s_k}{n_r^k}$$

- Disjunction: $\sigma_{\theta_1 \vee \cdots \vee \theta_k}(r)$
  - If conditions are independent, estimate is

$$n_r - n_r \times \frac{(n_r - s_1) \times \cdots \times (n_r - s_k)}{n_r^k}$$

- Negation: $\sigma_{\neg\theta}(r)$
  - Estimate is

$$n_r - size(\sigma_\theta(r))$$

Estimating Size of Joins

- Size of Cartesian product is $n_r.n_s$ tuples
- If $R \cap S = \Phi$, then $size(r \bowtie_\theta s) =$

Estimating Size of Joins

- Size of Cartesian product is $n_r.n_s$ tuples
- If $R \cap S = \Phi$, then $size(r \bowtie_\theta s) = size(r \times s)$
- If $R \cap S$ is a key for R,

Estimating Size of Joins

- Size of Cartesian product is $n_r.n_s$ tuples
- If $R \cap S = \Phi$, then $size(r \bowtie_\theta s) = size(r \times s)$
- If $R \cap S$ is a key for R, then each tuple of s will (natural) join with at most one tuple of r, resulting in at most n_s joined tuples
- If $R \cap S$ is a foreign key in S referencing R,

Estimating Size of Joins

- Size of Cartesian product is $n_r.n_s$ tuples
- If $R \cap S = \Phi$, then $size(r \bowtie_\theta s) = size(r \times s)$
- If $R \cap S$ is a key for R, then each tuple of s will (natural) join with at most one tuple of r, resulting in at most n_s joined tuples
- If $R \cap S$ is a foreign key in S referencing R, then it is exactly n_s

Estimating Size of Joins

- Size of Cartesian product is $n_r.n_s$ tuples
- If $R \cap S = \Phi$, then $size(r \bowtie_\theta s) = size(r \times s)$
- If $R \cap S$ is a key for R, then each tuple of s will (natural) join with at most one tuple of r, resulting in at most n_s joined tuples
- If $R \cap S$ is a foreign key in S referencing R, then it is exactly n_s
- Every tuple of r can join with $n_s/v_s(A)$ when joining attribute is A, thereby prducing $n_r.n_s/v_s(A)$ joined tuples
- Reversing r and s, estimate becomes $n_s.n_r/v_r(A)$
- Lower size is the better estimate
- Histograms can improve the estimates

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) =$

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size({}_A\mathcal{G}_F(r)) =$

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $size(r \cup s) =$

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $size(r \cup s) = n_r + n_s$
  - Intersection: $size(r \cap s) =$

Estimating Size of Other Operations

- Projection: $\mathit{size}(\Pi_A(r)) = v_r(A)$
- Aggregation: $\mathit{size}({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $\mathit{size}(r \cup s) = n_r + n_s$
  - Intersection: $\mathit{size}(r \cap s) = \min\{n_r, n_s\}$
  - Set difference: $\mathit{size}(r - s) =$

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $size(r \cup s) = n_r + n_s$
  - Intersection: $size(r \cap s) = \min\{n_r, n_s\}$
  - Set difference: $size(r - s) = n_r$
- Set operation estimates are *upper bounds*

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $size(r \cup s) = n_r + n_s$
  - Intersection: $size(r \cap s) = \min\{n_r, n_s\}$
  - Set difference: $size(r - s) = n_r$
- Set operation estimates are *upper bounds*
- Outer join
  - Left outer join: $size(r ⟕ s) =$

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $size(r \cup s) = n_r + n_s$
  - Intersection: $size(r \cap s) = \min\{n_r, n_s\}$
  - Set difference: $size(r - s) = n_r$
- Set operation estimates are *upper bounds*
- Outer join
  - Left outer join: $size(r ⟕ s) = size(r \bowtie s) + size(r)$
  - Right outer join: $size(r ⟖ s) =$

Estimating Size of Other Operations

- Projection: $\textit{size}(\Pi_A(r)) = v_r(A)$
- Aggregation: $\textit{size}({}_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $\textit{size}(r \cup s) = n_r + n_s$
  - Intersection: $\textit{size}(r \cap s) = \min\{n_r, n_s\}$
  - Set difference: $\textit{size}(r - s) = n_r$
- Set operation estimates are *upper bounds*
- Outer join
  - Left outer join: $\textit{size}(r ⟕ s) = \textit{size}(r \bowtie s) + \textit{size}(r)$
  - Right outer join: $\textit{size}(r ⟖ s) = \textit{size}(r \bowtie s) + \textit{size}(s)$
  - Full outer join: $\textit{size}(r ⟗ s) =$

Estimating Size of Other Operations

- Projection: $size(\Pi_A(r)) = v_r(A)$
- Aggregation: $size(_A\mathcal{G}_F(r)) = v_r(A)$
- Set operations
  - Union: $size(r \cup s) = n_r + n_s$
  - Intersection: $size(r \cap s) = \min\{n_r, n_s\}$
  - Set difference: $size(r - s) = n_r$
- Set operation estimates are *upper bounds*
- Outer join
  - Left outer join: $size(r ⟕ s) = size(r \bowtie s) + size(r)$
  - Right outer join: $size(r ⟖ s) = size(r \bowtie s) + size(s)$
  - Full outer join: $size(r ⟗ s) = size(r \bowtie s) + size(r) + size(s)$
- Outer join estimates are *upper bounds*