

CS315: DATABASE SYSTEMS QUERY OPTIMIZATION

Arnab Bhattacharya

arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur

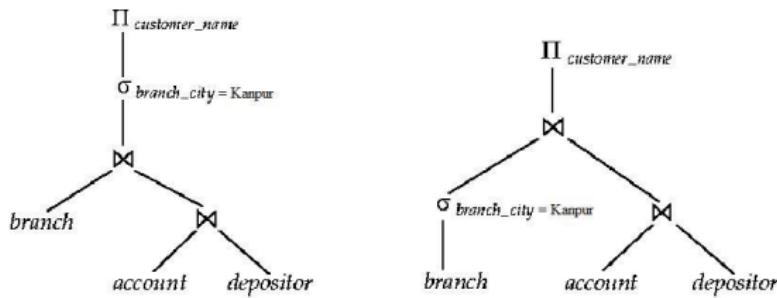
<http://web.cse.iitk.ac.in/~cs315/>

2nd semester, 2019-20

Tue, Wed 12:00-13:15

Evaluation Plan

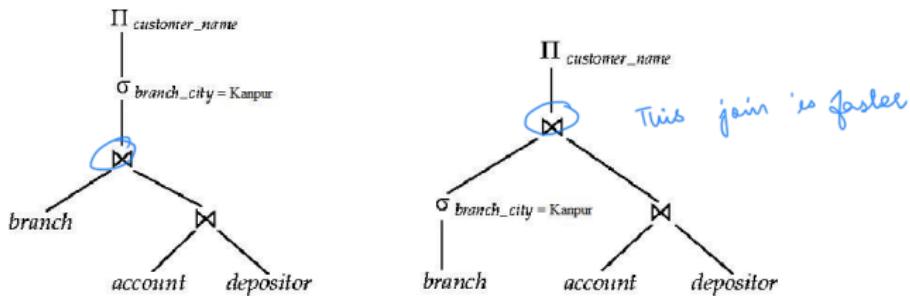
- Equivalent expressions provide alternate ways of executing a query



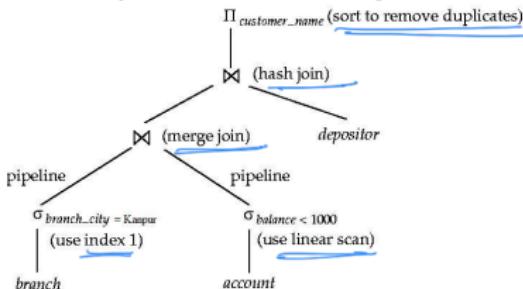
Expression-tree

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 $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$

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 $\Pi_{L_1}(\Pi_{L_2}(\dots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$
- 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 $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

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 $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$

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

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 $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$ $\theta_2 : E_2 \& E_3 \text{ attr only}$

θ_2 involves attributes from only E_2 and E_3

10 $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_2} E_2$ \Rightarrow contains only E_2 attr.

θ_1 involves attributes from only E_1

11 $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2))$ \Rightarrow faster

θ_1 and θ_2 involve attributes from only E_1 and E_2 respectively
only E_1 only E_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

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 $\sigma_\theta(E_1 \oplus E_2) = \sigma_\theta(E_1) \oplus \underline{E_2}$ $\left[\begin{matrix} \cup \rightarrow \text{extra tuples from } E_2 \\ \text{is being done} \end{matrix} \right]$ as no selection

- \oplus is any of \cup 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
 - o $(\text{depositor} \bowtie \text{acc} \bowtie \text{branch}) \Rightarrow \text{depositor} \bowtie \text{acc.} \bowtie (\text{o} \text{ Kanpur} \text{ branch})$
 b city = Kanpur

Example

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

Example

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

Example

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

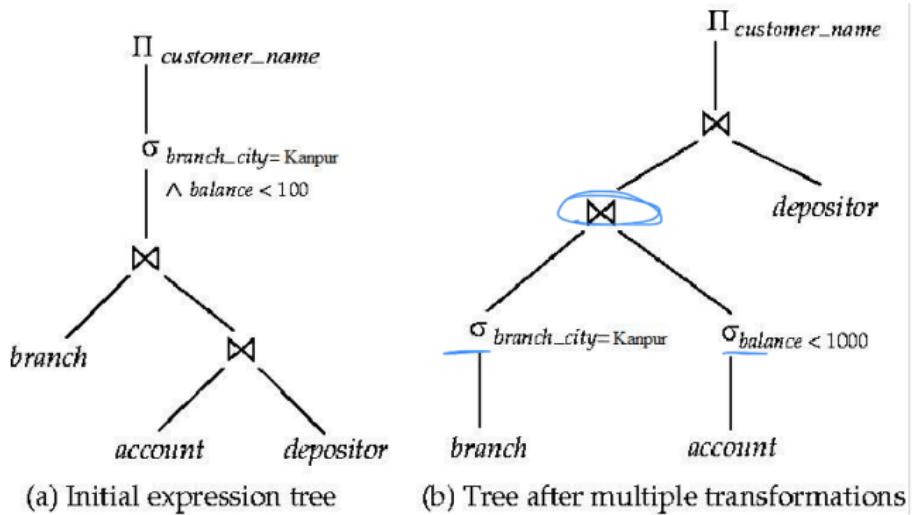
$\Rightarrow \Pi_{cname} (\text{depositor} \bowtie ((\sigma_{bal > 1000} \text{ account}) \bowtie (\sigma_{bcity = “Kanpur”} \text{ branch}))$

Example

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

Example

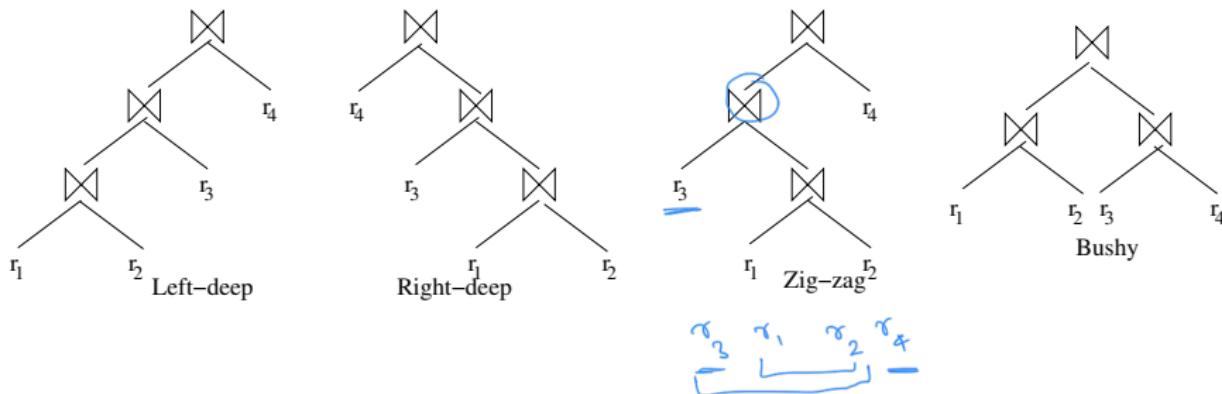
- Find names of customers having an account at “Kanpur” city
 - $\Pi_{cname}(\sigma_{bcity='Kanpur'}(branch \bowtie (account \bowtie depositor)))$
 - $\Pi_{cname}(\sigma_{bcity='Kanpur'}(branch) \bowtie (account \bowtie depositor))$
- Find names of customers with an account in “Kanpur” with balance more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



(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} [at least one-interval node child is single row]
 - Place 2 leaves; rest can be either right or left
 - Bushy trees: $(n - 1)^{\text{th}}$ **Catalan number** $\frac{(2n-2)!}{n!(n-1)!}$
 - $C(1) = 1$; $C(n) = \sum_{i=1}^{n-1} C(i).C(n-i)$

Commutativity is different $\Rightarrow C_n = \frac{(2n-2)!}{(n-1)!}$

No. of ways of
bracketing

$$\begin{matrix} r_1 \leftrightarrow r_2 \\ r_2 \leftrightarrow r_1 \end{matrix}$$

Commutativity is not different $\Rightarrow N_n = \frac{(2n-2)!}{2^{n-1}(n-1)!}$

Algorithm for Join Order

- Consider set \underline{S} as the join of n relations
- S can be represented as $S_1 \bowtie (\underline{S} - S_1)$ for any non-empty proper subset $S_1 \subset S$
- Choose \underline{S}_1 that minimizes the overall cost
- Recursively choose the best plan for \underline{S}_1 and $(\underline{S} - \underline{S}_1)$

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

[Dynamic Prog. Solution]

- 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 $S_1 \subset S$
$$S_1 = \min_{S_1 \subset S} \{S_1 \bowtie (S - S_1)\}$$
- Choose S_1 that minimizes the overall cost
- Recursively choose the best plan for $\underline{S_1}$ and $\underline{(S - S_1)}$
- 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

- 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 $\underline{l_r} = \frac{\text{block size}}{\text{blocking factor}}$

- 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

- 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 $l_r = \frac{\text{tuple size}}{\text{for}}$
- 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
- For each index
 - Number of levels of the index
 - Number of leaves

Index \Rightarrow Bt-tree

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 $x > \max_r(A)$
 - $n_r \cdot \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ 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 $x > \max_r(A)$
 - $n_r \cdot \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
 - Estimate is better when histograms are available
- $\sigma_{A < x}(r)$
 - $\text{size}(\sigma_{A \leq x}(r)) - \text{size}(\sigma_{A=x}(r))$

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 $x > \max_r(A)$
 - $n_r \cdot \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
 - Estimate is better when histograms are available
- $\sigma_{A < x}(r)$
 - $\text{size}(\sigma_{A \leq x}(r)) - \text{size}(\sigma_{A=x}(r))$
- $\sigma_{A \geq x}(r)$
 - $n_r - \text{size}(\sigma_{A < x}(r))$

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 $x > \max_r(A)$
 - $n_r \cdot \frac{x - \min_r(A)}{\max_r(A) - \min_r(A)}$ assuming uniform distribution
 - Estimate is better when histograms are available
- $\sigma_{A < x}(r)$
 - $\text{size}(\sigma_{A \leq x}(r)) - \text{size}(\sigma_{A=x}(r))$
- $\sigma_{A \geq x}(r)$
 - $n_r - \text{size}(\sigma_{A < x}(r))$
- $\sigma_{A > x}(r)$
 - $n_r - \text{size}(\sigma_{A \leq x}(r))$

Estimating Size of Selections

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

Estimating Size of Complex Selections

multiple attr.

- **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: $\sigma_{\theta_1 \wedge \dots \wedge \theta_k}(r)$
 - 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: $\sigma_{\theta_1 \wedge \dots \wedge \theta_k}(r)$
 - If conditions are independent, estimate is

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

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: $\sigma_{\theta_1 \wedge \dots \wedge \theta_k}(r)$
 - If conditions are independent, estimate is

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

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

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

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

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

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: $\sigma_{\theta_1 \wedge \dots \wedge \theta_k}(r)$
 - If conditions are independent, estimate is

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

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

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

- 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: $\sigma_{\theta_1 \wedge \dots \wedge \theta_k}(r)$
 - If conditions are independent, estimate is

$$n_r \times \frac{s_1 \times \dots \times s_k}{n_r^k} \quad ; \text{total no. of tuples going to be selected}$$

- Disjunction: $\sigma_{\theta_1 \vee \dots \vee \theta_k}(r) \rightarrow (\theta_1 \cup \theta_2 \cup \theta_3 \dots \cup \theta_k) = \overline{\theta}_1 \cap \overline{\theta}_2 \cap \overline{\theta}_3 \dots \cap \overline{\theta}_k$
 - If conditions are independent, estimate is

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

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

$$\underline{n_r} - \underline{\text{size}}(\sigma_\theta(r))$$

Estimating Size of Joins

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

Estimating Size of Joins

- Size of Cartesian product is $n_r.n_s$ tuples
- If $R \cap S = \Phi$, then $\text{size}(r \bowtie_\theta s) = \text{size}(r \times s)$
- 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 $\text{size}(r \bowtie_\theta s) = \text{size}(r \times s)$
- 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 $n_r \cdot n_s$ tuples
- If $R \cap S = \emptyset$, then $\text{size}(r \bowtie_\theta s) = \text{size}(r \times s)$
- 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 $\text{size}(r \bowtie_\theta s) = \text{size}(r \times s)$
- 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 producing $n_r.n_s/v_s(A)$ 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: $\text{size}(\Pi_A(r)) =$

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

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

Estimating Size of Other Operations

- Projection: $\text{size}(\Pi_A(r)) = \underline{v_r(A)}$
- Aggregation: $\text{size}({}_A\mathcal{G}_F(r)) = \underline{v_r(A)}$
- Set operations
 - Union: $\text{size}(r \cup s) = n_r + n_s$ [upper bound]
 - Intersection: $\text{size}(r \cap s) = \min\{\underline{n_r}, \underline{n_s}\}$ {upper bound}
 - Set difference: $\text{size}(r - s) = n_r \setminus \{ \dots \}$
- Set operation estimates are upper bounds
- Outer join
 - Left outer join: $\text{size}(r \Join s) = \text{size}(r \bowtie s) + \text{size}(r)$ ^{nr}
 - Right outer join: $\text{size}(r \ltimes s) = \text{size}(r \bowtie s) + \text{size}(s)$ ^{ns}
 - Full outer join: $\text{size}(r \Join s) = \text{size}(r \bowtie s) + \text{size}(r) + \text{size}(s)$ ^{nr} ^{ns}
- Outer join estimates are upper bounds