Join and Aggregation Algorithms

CSE 4/562: Database Systems | Lecture 5

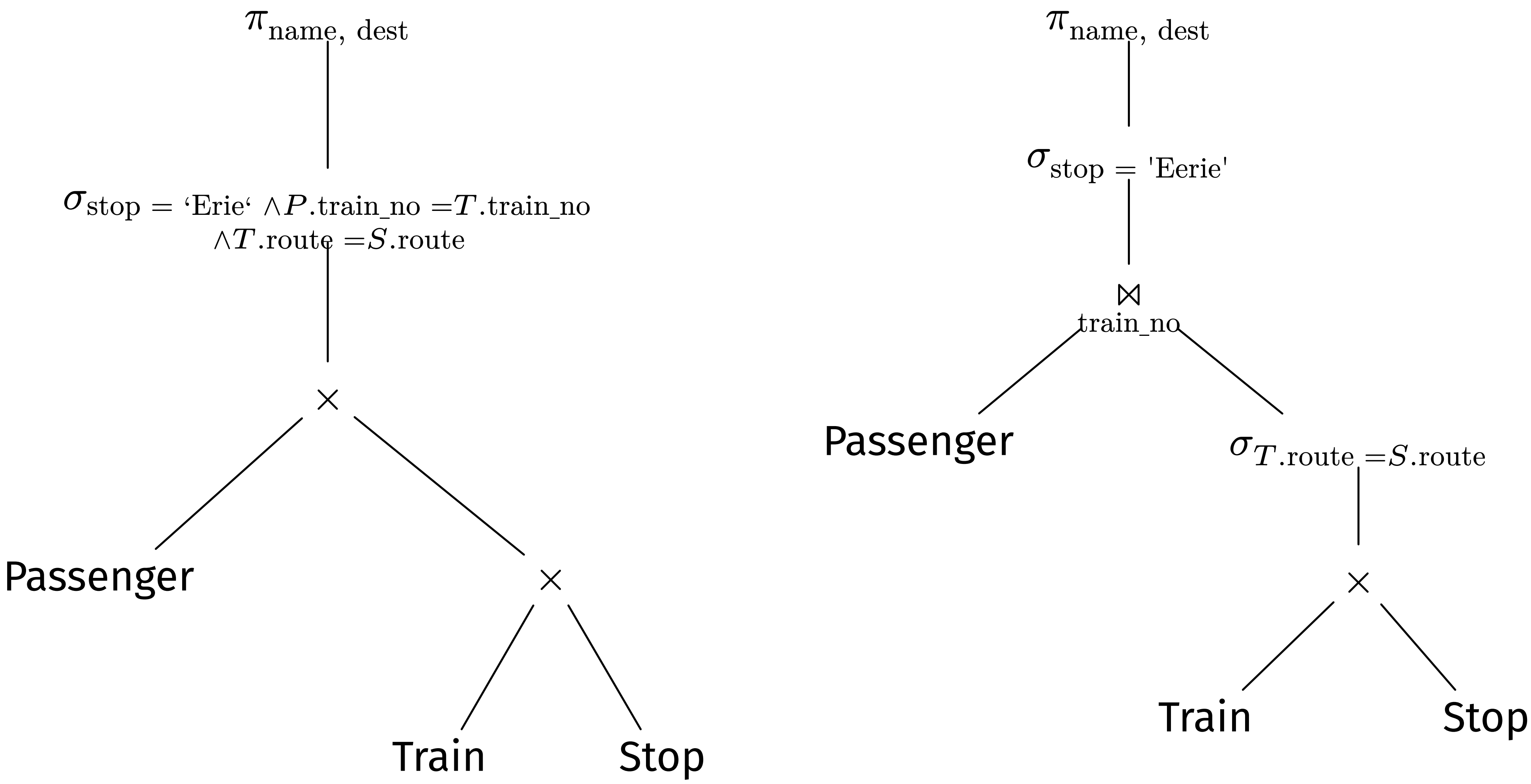
DB. Sys.: T.C.B.: 16.1-16.3

Recap

Query Evaluation

```sql
SELECT name, dest
FROM Passenger P, Train T, Stop S
WHERE stop = 'Erie'
  AND P.train_no = T.train_no
  AND T.route = S.route
```

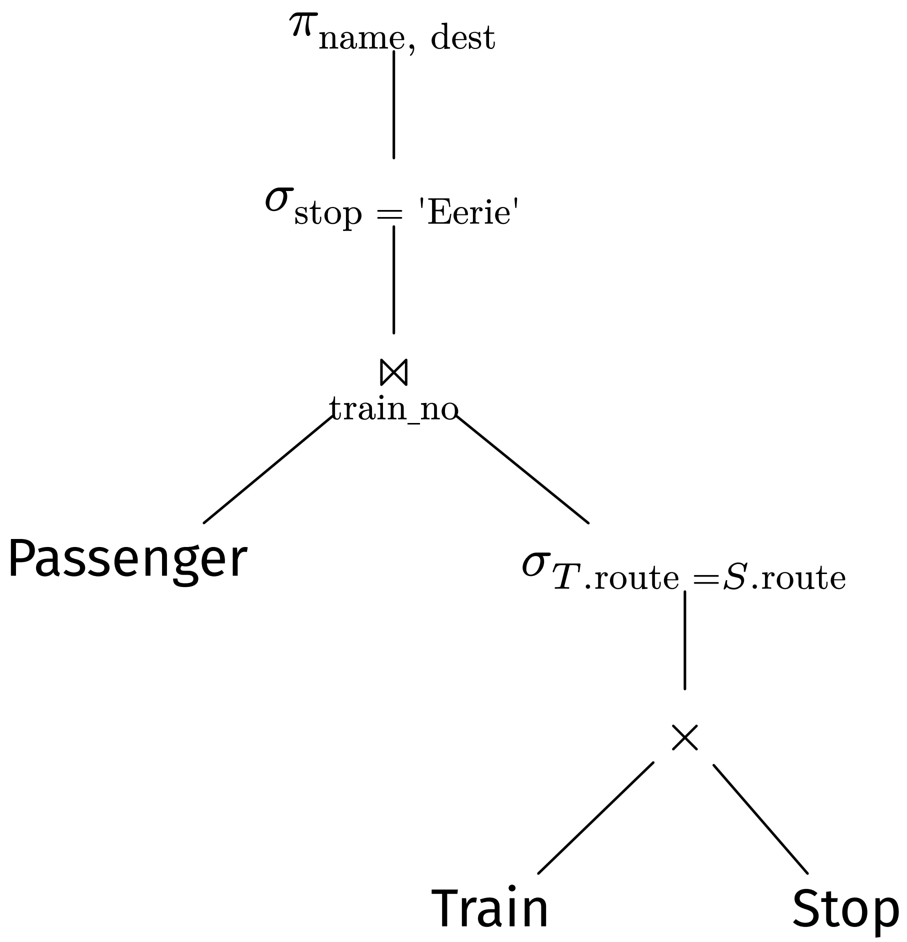
Query Evaluation



The Iterator Model

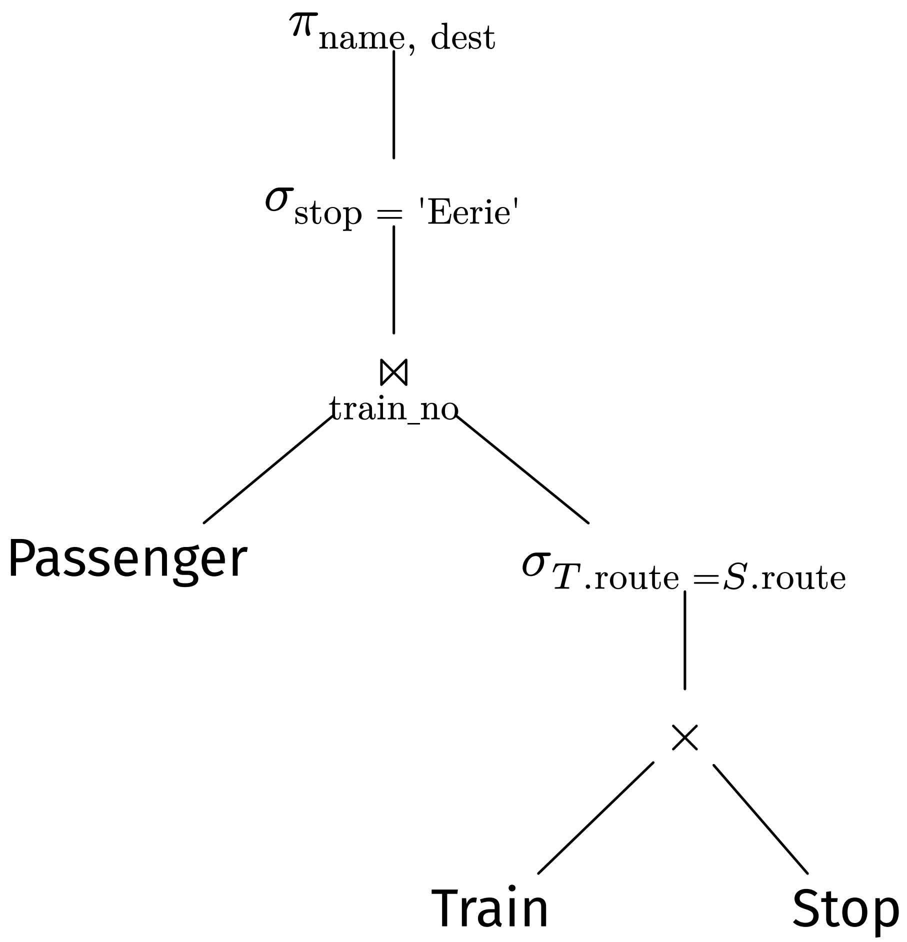
Operator	Memory	IO
Table	$O(1)$	$O(n)$
Project (π)	$O(1)$	$O(1)$
Filter (σ)	$O(1)$	$O(1)$
Union ($\cup$)	$O(1)$	$O(1)$
Nested Loop ($\times$)	$O(1)$	$O(n^2)$
Block Nested Loop ($\times$)	$O(\text{Buffer})$	$O\left(\frac{n^2}{\text{Buffer}}\right)$
2-Pass Hash Join ($\bowtie$)	$O\left(\frac{n}{\text{Buckets}}\right)$	$O(n)$
Sort Merge Join ($\bowtie$)	$O(1)$+sort	$O(1)$+sort
2-Pass Hash Aggregate (Σ)	$O\left(\frac{n}{\text{Buckets}}\right)$	$O(n)$
Sort Aggregate	$O(1)$+sort	$O(1)$+sort

Query Cost Estimation




```
Table(Train)

Table(Stop)

BlockNLoop

Filter

Table(Pa...)

2PHashJoin

Filter

Project
```

Query Cost Estimation



$O(|T|)$ `Table(Train)`

$+O(|S|)$ `Table(Stop)`

`BlockNLoop`

`Filter`

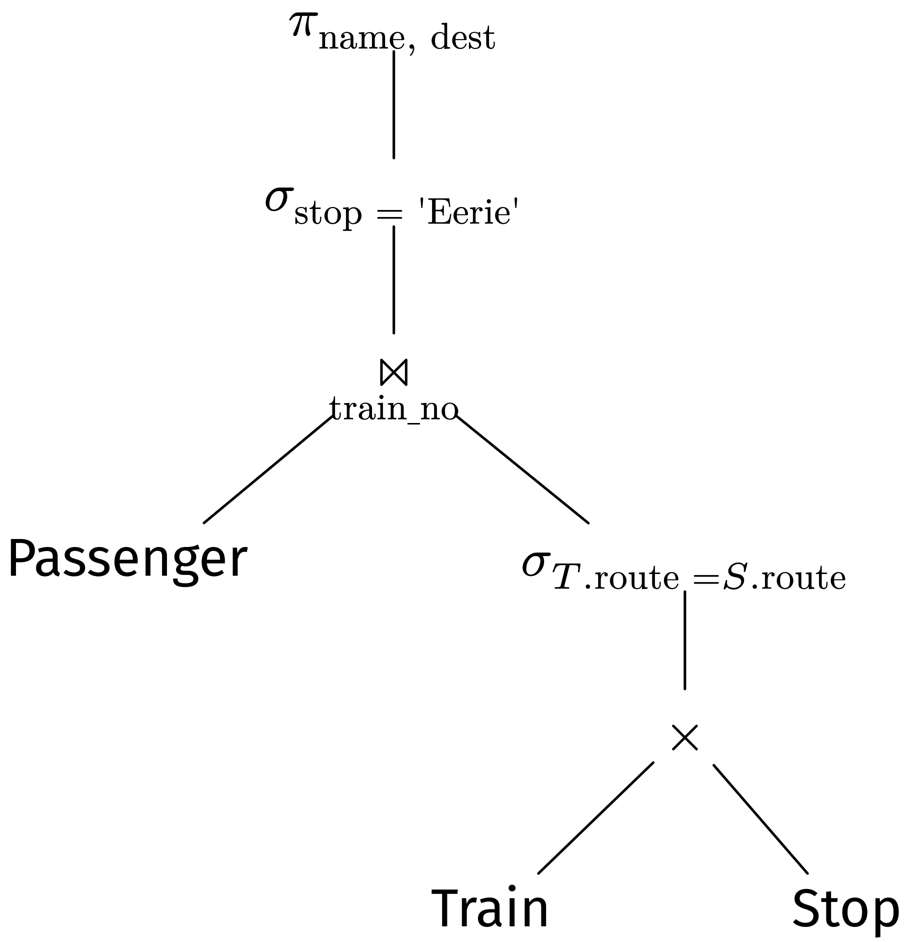
`Table(Pa...)`

`2PHashJoin`

`Filter`

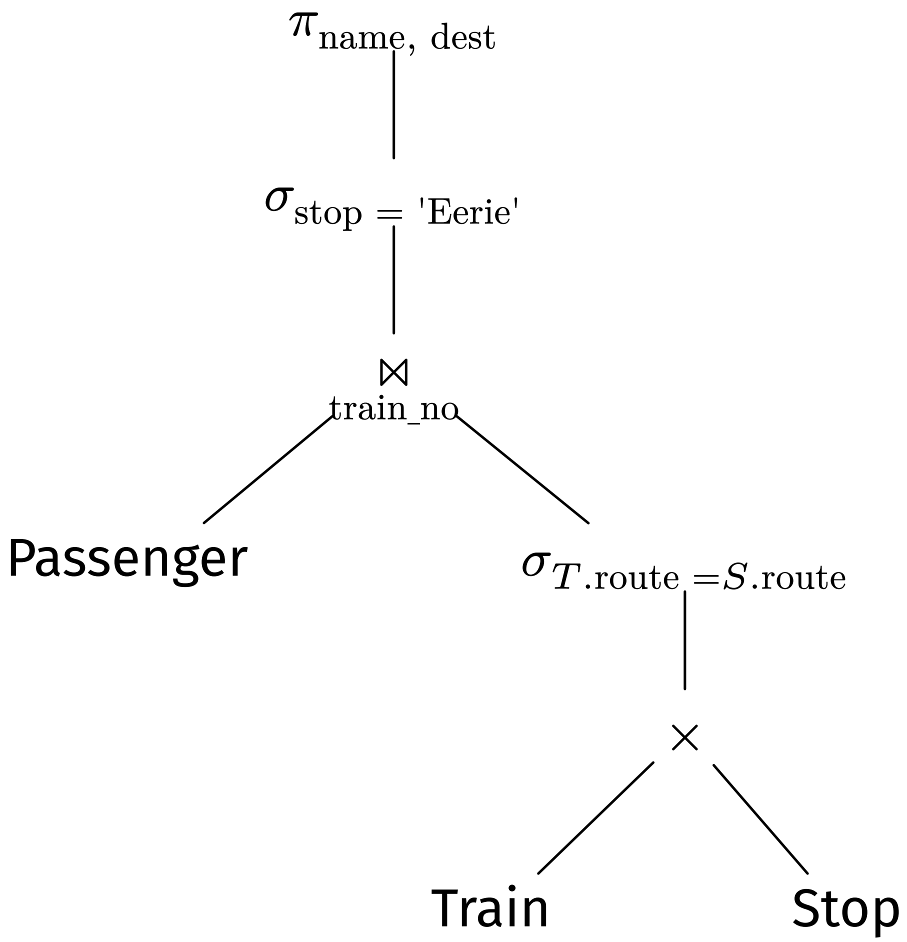
`Project`

Query Cost Estimation



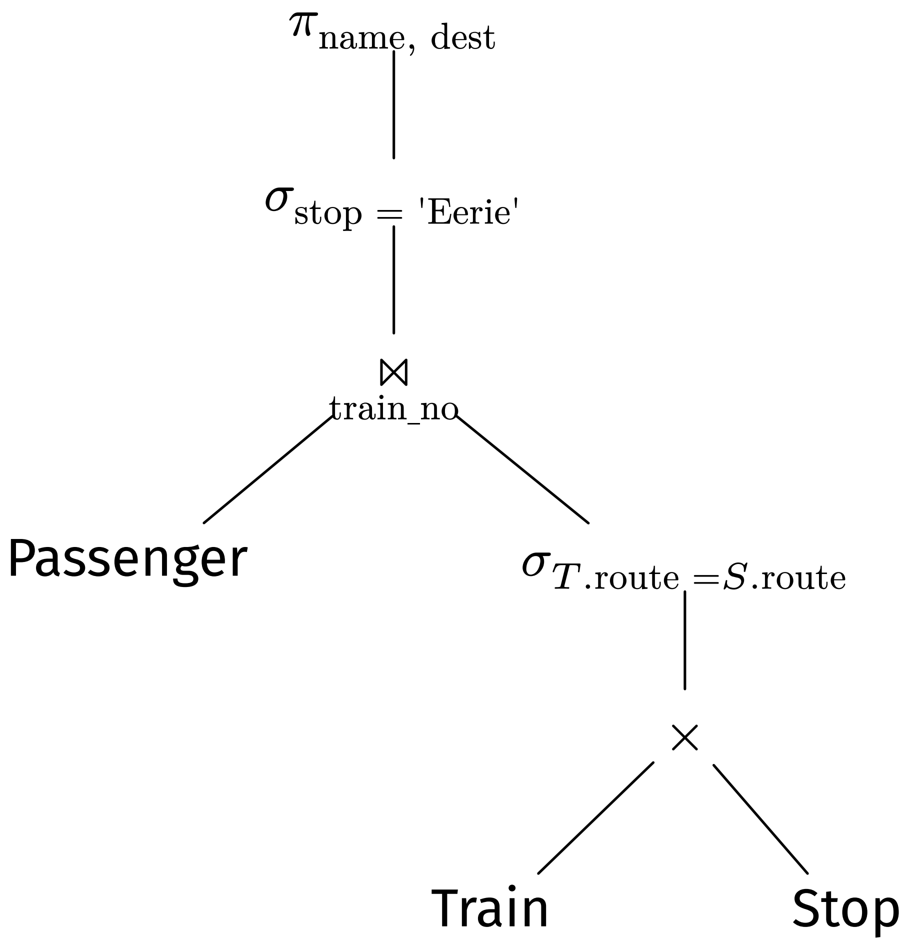
$O(\lvert T \rvert)$	`Table(Train)`
$+O(\lvert S \rvert)$	`Table(Stop)`
$+O\left(\frac{\lvert T \times S \rvert}{\text{Buffer}_1}\right)$	`BlockNLoop`
	`Filter`
	`Table(Pa...)`
	`2PHashJoin`
	`Filter`
	`Project`

Query Cost Estimation



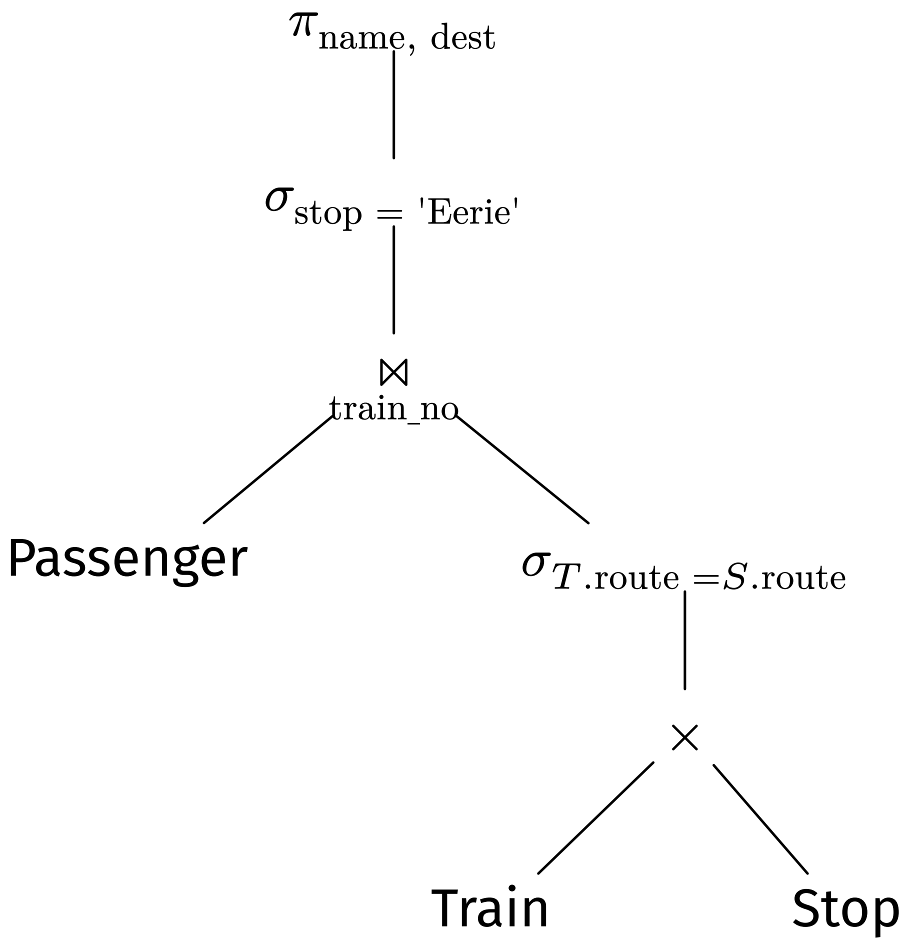
$O(\lvert T \rvert)$	`Table(Train)`
$+O(\lvert S \rvert)$	`Table(Stop)`
$+O\left(\frac{\lvert T \times S \rvert}{\text{Buffer}_1}\right)$	`BlockNLoop`
$+0$	`Filter`
	`Table(Pa...)`
	`2PHashJoin`
	`Filter`
	`Project`

Query Cost Estimation



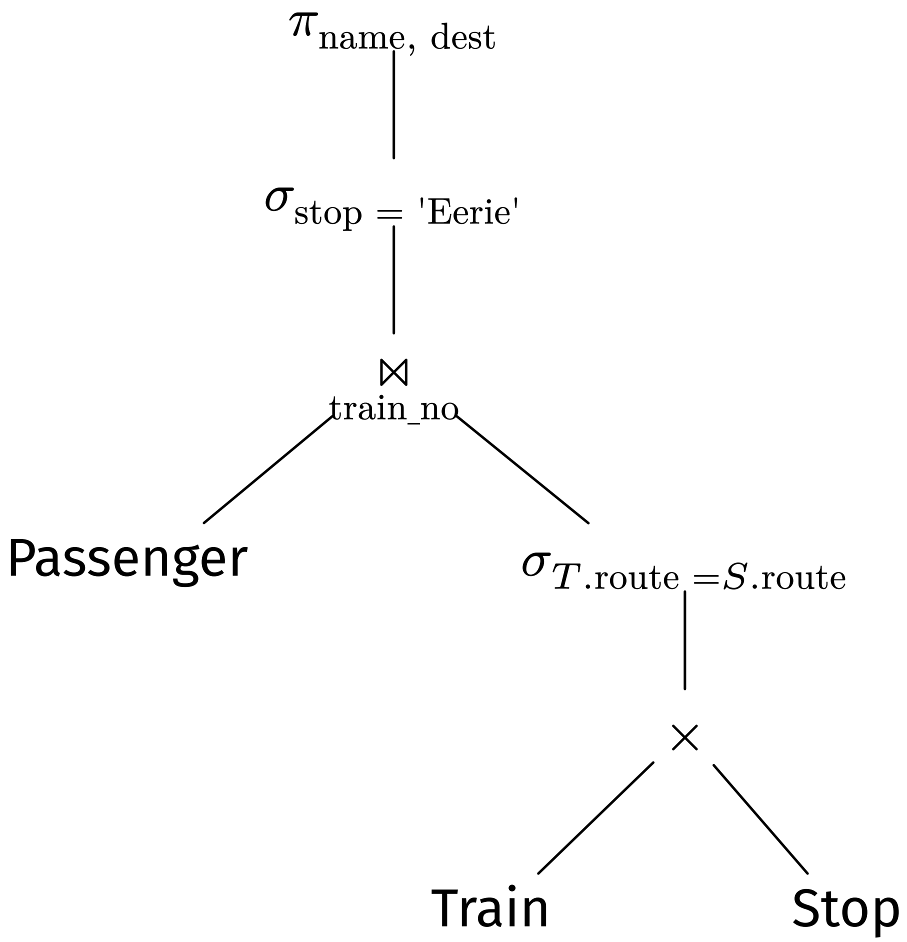
$O(\|T\|)$	`Table(Train)`
$+O(\|S\|)$	`Table(Stop)`
$+O\left(\frac{\|T \times S\|}{\text{Buffer}_1}\right)$	`BlockNLoop`
$+0$	`Filter`
$+O(\|P\|)$	`Table(Pa...)`
	`2PHashJoin`
	`Filter`
	`Project`

Query Cost Estimation



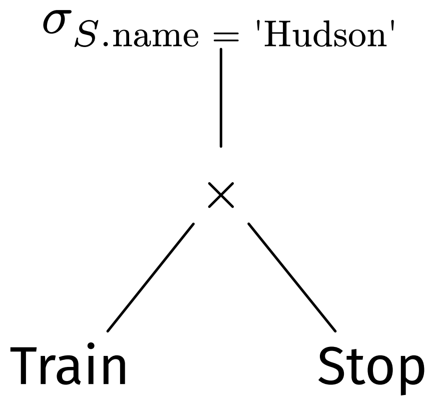
$O(\lvert T \rvert)$	`Table(Train)`
$+O(\lvert S \rvert)$	`Table(Stop)`
$+O\left(\frac{\lvert T \times S \rvert}{\text{Buffer}_1}\right)$	`BlockNLoop`
$+0$	`Filter`
$+O(\lvert P \rvert)$	`Table(Pa...)`
$+O(\lvert P \rvert + \lvert \sigma(T \times S) \rvert)$	`2PHashJoin`
	`Filter`
	`Project`

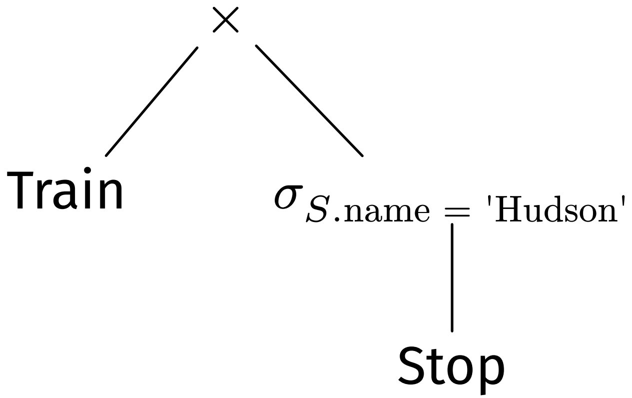
Query Cost Estimation



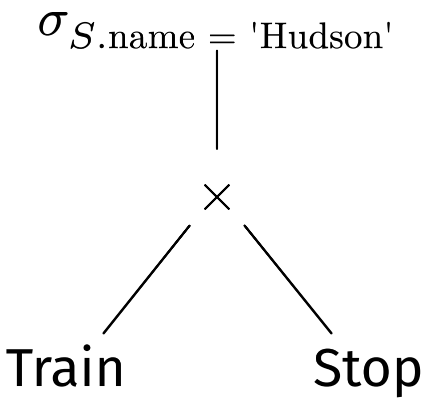
$O(\|T\|)$	`Table(Train)`
$+O(\|S\|)$	`Table(Stop)`
$+O\left(\frac{\|T \times S\|}{\text{Buffer}_1}\right)$	`BlockNLoop`
$+0$	`Filter`
$+O(\|P\|)$	`Table(Pa...)`
$+O(\|P\| + \|\sigma(T \times S)\|)$	`2PHashJoin`
$+0$	`Filter`
$+0$	`Project`

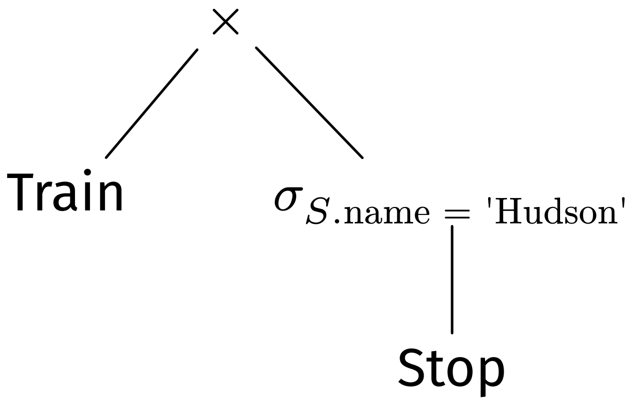
Options





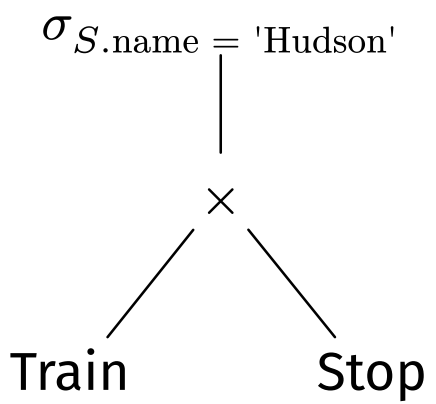
Options



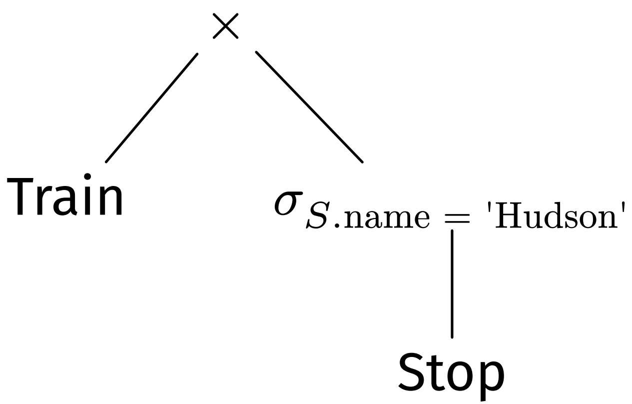


$$|T| + |S| + \frac{|T| \times |S|}{\text{Buffer}} + 0$$

Options

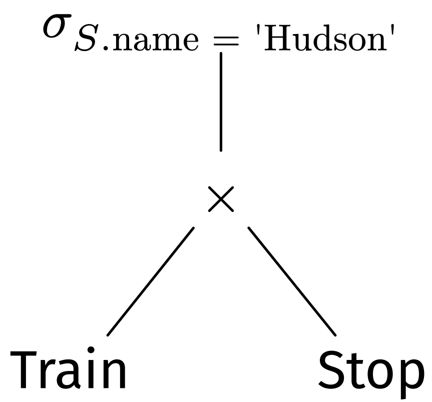


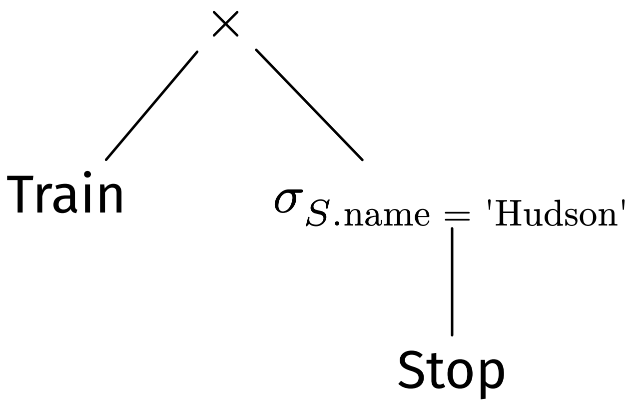
$$|T| + |S| + \frac{|T| \times |S|}{\text{Buffer}} + 0$$



$$|T| + |S| + 0 + \frac{|T| \times |\sigma S|}{\text{Buffer}}$$

Options





$$|T| + |S| + \frac{|T| \times |S|}{\text{Buffer}} + 0$$

$$|T| + |S| + 0 + \frac{|T| \times |\sigma S|}{\text{Buffer}}$$

$$|S| > |\sigma S|$$

Which query plans are the same?

$$a \times (b + (c \times (e + f)))$$

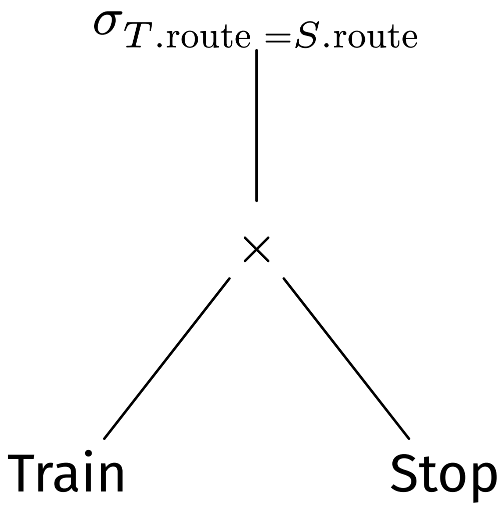
$$ace + ba + fac$$

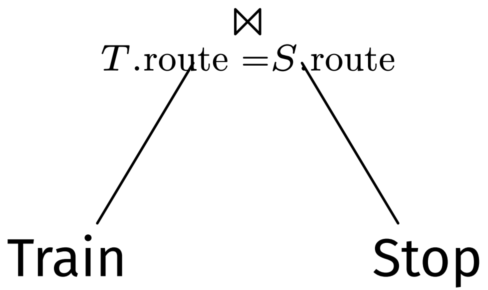
Are these the same?

$$a \times (b + (c \times (e + f)))$$

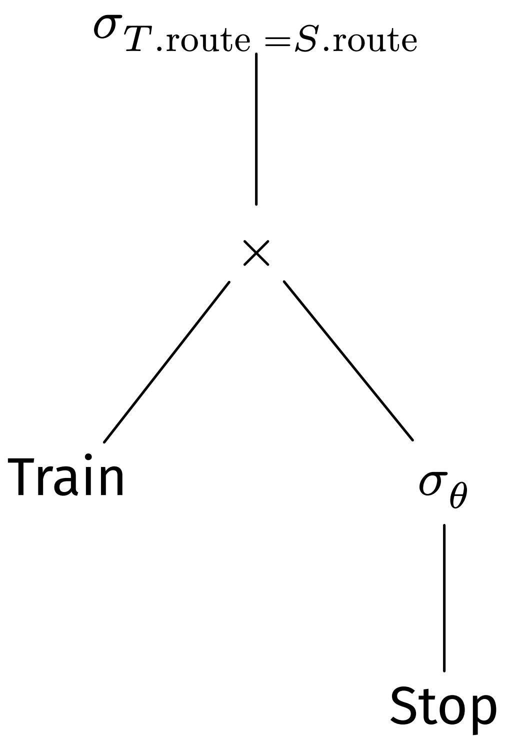
$$ace + ba + fac$$

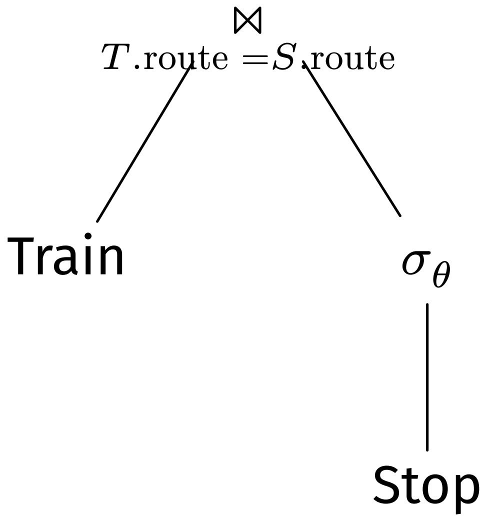
Is one easier to solve than the other?

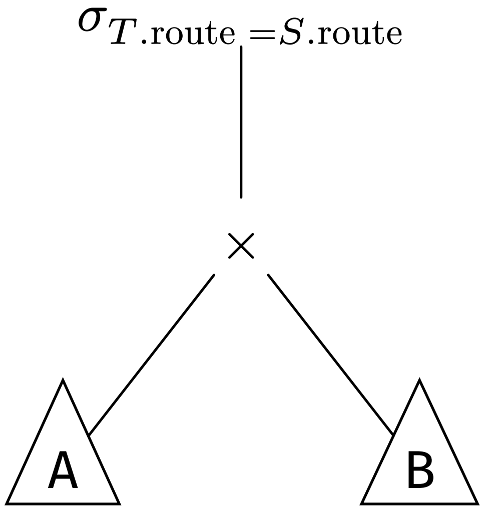

$\sigma_{T.\text{route} = S.\text{route}}$
×
Train
Stop

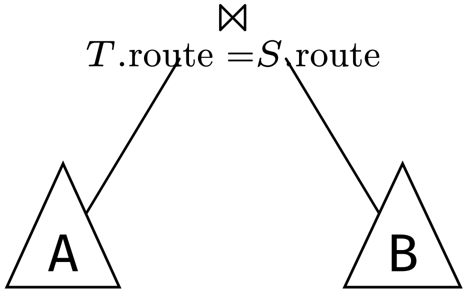

$\bowtie_{T.\text{route} = S.\text{route}}$
Train
Stop

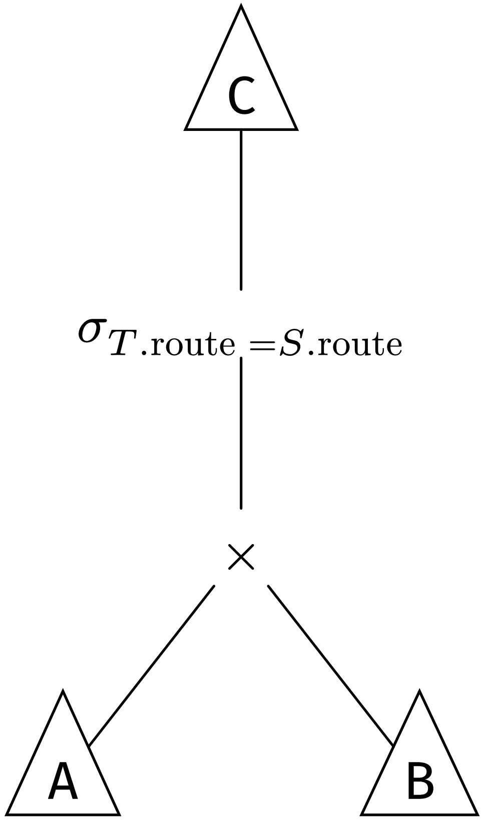
Relational Algebra

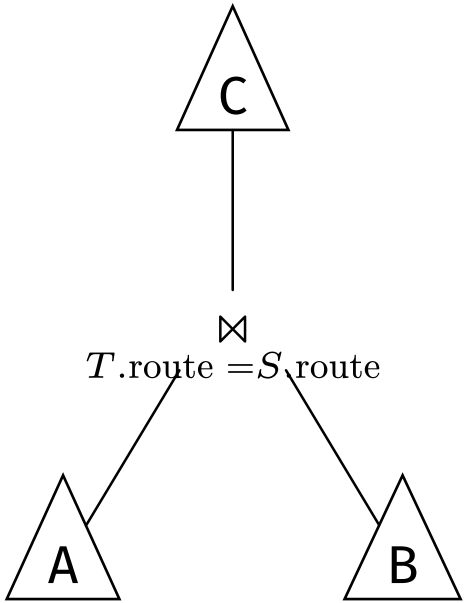





$\sigma_{T.\text{route} = S.\text{route}}$
$\times$
A
B


$\bowtie_{T.\text{route} = S.\text{route}}$
A
B


C
$\sigma_{T.\text{route} = S.\text{route}}$
$\times$
A
B


C
$\bowtie_{T.\text{route} = S.\text{route}}$
A
B

Rewrite Rules

- $\sigma_\theta(R \times S) \rightarrow R \bowtie_\theta S$
  - 🏃 Usually faster if θ is an equality test
  - Strategy 1 (Fixed Point; Spark)
- $R \times S \rightarrow S \times R$
  - 🤷 Sometimes a good idea?
  - Strategy 2 (Cost-Based; Postgres)

Fixed Point Optimization

- Search through the plan for an occurrence of $\sigma_\theta(Q_1 \times Q_2)$.
- Replace it with $Q_1 \underset{\theta}{\bowtie} Q_2$.
- Repeat until done.

Fixed Point Optimization

- For (**Pattern** $\rightarrow$ **Rewrite**) in **Rules**
  - Search through the plan for a subplan Q that matches **Pattern**[1].
  - Replace Q with **Rewrite**(Q).
- Repeat until done.

[1] We'll go into more detail when we talk about Checkpoint 3, and after we talk about IVM.

Cost-Based Optimization

- **Plans** $\leftarrow$ Naive Plan
- Loop until done (or threshold)
  - **Plans** $\leftarrow$ Apply a new rewrite to a plan in **Plans**
- Estimate the cost of all **Plans** and use the cheapest.

What are valid rewrites?

If, for any Q_1, Q_2 where:

- Q_1 matches **Pattern**
- $Q_2 = \textbf{Rewrite}\ (Q_1)$

...it is also true that, for any database D:

- $Q_1(D) = Q_2(D)$

...then we say that the rewrite rule (**Pattern**, **Rewrite**) is valid.

Ground Rules

$$Q_1(D) = Q_2(D)$$

Ground Rules

$$Q_1(D) = Q_2(D)$$

The relation obtained by evaluating Q_1 on **any** D is equivalent to that obtained by evaluating Q_2

- **Bag-Equivalence**: The same number of each record. Order does not matter.
- **Set-Equivalence**: The same records, ignoring duplicates. Order does not matter.
- **List-Equivalence**: The same records in the same order.

Ground Rules

$$Q_1(D) = Q_2(D)$$

The relation obtained by evaluating Q_1 on **any** D is equivalent to that obtained by evaluating Q_2

- **Bag-Equivalence**: The same number of each record. Order does not matter.
- **Set-Equivalence**: The same records, ignoring duplicates. Order does not matter.
- **List-Equivalence**: The same records in the same order.

For now, we use **Bag** equivalence

$$Q_1(D) = Q_2(D)$$

The relation obtained by evaluating Q_1 on **any** D is equivalent to that obtained by evaluating Q_2

- **Bag-Equivalence**: The same number of each record. Order does not matter.
- **Set-Equivalence**: The same records, ignoring duplicates. Order does not matter.
- **List-Equivalence**: The same records in the same order.

For now, we use **Bag** equivalence

We also ignore attribute order ($R(A, B, C)$ is the same as $R(C, B, A)$)

Query Equivalence...

$Q_1(D) \equiv Q_2(D)$ if and only if, for any database D:

- $Q_1(D)$ is a valid query if and only if $Q_2(D)$ is a valid query.
- The set of attributes of $Q_1(D)$ is the same as the set of attributes of $Q_2(D)$.
- The bag of records of $Q_1(D)$ is the same as the bag of records of $Q_2(D)$.

Rewrite Validity

(**Pattern**, **Rewrite**) is valid if for any database D and any valid query $Q(D)$ that matches **Pattern**, **Rewrite**:

- $(\textbf{Rewrite}\ (Q))(D) \equiv Q(D)$

Selection

$$\sigma_{\theta_1 \wedge \theta_2}(R) \equiv \sigma_{\theta_1}\left(\sigma_{\theta_2}(R)\right) \qquad \text{Decomposability}$$

Projection

$$\pi_A(R) \equiv \pi_A(\pi_{A \cup B}(R)) \qquad \text{Idempotence}$$

Cartesian Product

$$R \times (S \times T) \equiv (R \times S) \times T \qquad \text{Associativity}$$

$$R \times S \equiv S \times R \qquad \text{Commutativity}$$

$$\sigma_\theta(R \times S) \equiv R \underset{\theta}{\bowtie} S \qquad \text{Join Conversion}$$

Union

$$R \cup (S \cup T) \equiv (R \cup S) \cup T \qquad \text{Associativity}$$

$$R \cup S \equiv S \cup R \qquad \text{Commutativity}$$

Try it: Show that...

$$R \times (S \times T) \equiv T \times (S \times R)$$

Algebra!

Try it: Show that...

$$\sigma_{\theta_1}\big(\sigma_{\theta_2}(R)\big) \equiv \sigma_{\theta_2}\big(\sigma_{\theta_1}(R)\big)$$

Try it: Show that...

$$R \underset{\theta}{\bowtie} S \equiv S \underset{\theta}{\bowtie} R$$

Try it: Show that...

$$\sigma_{R.B=S.B \wedge R.A>3}(R \times S) \equiv \sigma_{R.A>3}\left(R \underset{B}{\bowtie} S\right)$$

Selection + Projection

$$\pi_A(\sigma_\theta(R)) \equiv \sigma_\theta(\pi_A(R)) \quad \text{Commutativity}$$

... but only if A and θ are compatible

A must include all columns referenced by θ ($\mathrm{cols}(\theta)$)

Try it: Show that...

$$\pi_A(\sigma_\theta(R)) \equiv \pi_A\Big(\sigma_\theta\Big(\pi_{A\cup\,\mathrm{cols}(\theta)}(R)\Big)\Big)$$

Selection + Product Rules

Selection+Cross Product

$$\sigma_\theta(R \times S) \equiv (\sigma_\theta(R)) \times S$$ Selection "Push Down"

... but only θ references only columns of R

$$\text{cols}(\theta) \subseteq \text{cols}(R)$$

Try it: Show that...

$$\sigma_{R.B=S.B \wedge R.A>3}(R \times S) \equiv (\sigma_{R.A>3}(R)) \underset{B}{\bowtie} S)$$

Projection+Cross Product

$$\pi_A(R \times S) \equiv \left(\pi_{A_R}(R)\right) \times \left(\pi_{A_S}(S)\right)$$ Projection “Push Down”

... where $A_R = A \cap \text{cols}(R)$ and $A_S = A \cap \text{cols}(S)$

Try it: Show that...

$$\pi_A(R \bowtie S) \equiv \left(\pi_{A_R}(R)\right) \bowtie \left(\pi_{A_S}(S)\right)$$

(Is this always true?)

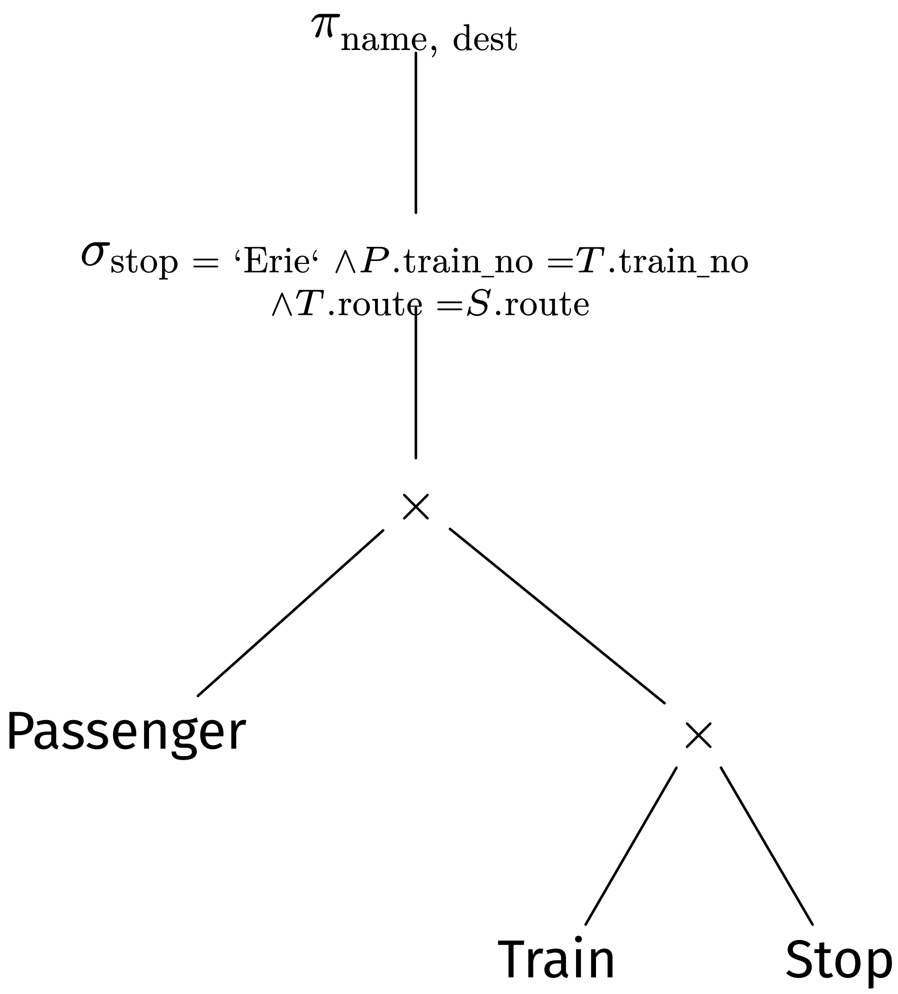
Union Rules

Union

$\sigma_\theta(R \cup S) \equiv (\sigma_\theta(R)) \cup (\sigma_\theta(S))$	Selection “Push Down”
$\pi_A(R \cup S) \equiv (\pi_A(R)) \cup (\pi_A(S))$	Projection “Push Down”
$R \times (S \cup T) \equiv (R \times S) \cup (R \times T)$	Distributivity

Example

```sql
SELECT p.name, t.id
FROM passenger p, train t, stop s
WHERE p.train_no = t.train_no
  AND t.route = s.route
  AND stop = 'Erie'
```



Rewrite Strategies

Selection Pushdown & Join Construction

Almost always a good idea

Projection Pushdown

May remove redundant columns (data copies), and may avoid loading data

Join Algorithm Selection

Remember that $\times$ and $\bowtie$ are actually several different algorithms

Join/Union Ordering

$R \times S$ and $S \times R$ may have different memory requirements.

$R \bowtie (S \bowtie T)$ and $(R \bowtie S) \bowtie T$ have *different* memory/IO requirements.

Access Paths

$\sigma_\theta(R)$ and $Q(\ldots) \underset{\theta}{\bowtie} R$ are special cases that we can sometimes optimize

Deliverables

- AI Quiz and Checkpoint 0 Past Due!
  - Reach out to me if you haven't finished it yet!
- Checkpoint 1 due Monday!
  - Don't forget to schedule a code review (times available tomorrow and next Friday).