Relational Algebra

Database Systems: The Complete Book

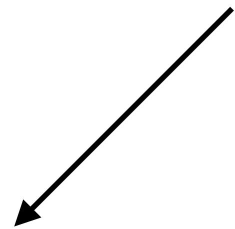
Ch 2.4 (plus preview of 15.1, 16.1)

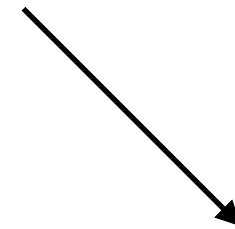
The running theme…

Replace **[thing]** with better, but equivalent **[thing]**.

The running theme…

Replace **[thing]** with better, but equivalent **[thing]**.





How can we tell if **[thing]** is better?

How can we tell if **[thing]** is equivalent?

First, a few definitions…

Relational Data

- Relation (Table): A collection of Tuples of Values
  - All tuples have the same set of attributes, or schema
- What constraints are present on the collection?

<u>Uniqueness</u> <u>Order Matters</u>

```
<Spock,    Lt.>
<Kirk,     Capt.>
<Redshirt, Ensign>
<McCoy,    Lt. Cmdr>
```

Set

```
<Redshirt, Ensign>
<Spock,    Lt.>
<Kirk,     Capt.>
<Redshirt, Ensign>
<Redshirt, Ensign>
<McCoy,    Lt. Cmdr>
```

Bag

```
<Kirk,     Capt.>
<Spock,    Lt.>
<McCoy,    Lt. Cmdr>
<Redshirt, Ensign>
<Redshirt, Ensign>
<Redshirt, Ensign>
```

List

Declarative Languages

Declarative	Imperative
Say **what** you want	Say **how** you want to get it
"Get me the TPS reports"	"Look at every T report, For each week, Sum up the sprocket count Find that week's S report etc…."
SQL, RA, R, …	C, Scala, Java, Ruby, Python, …

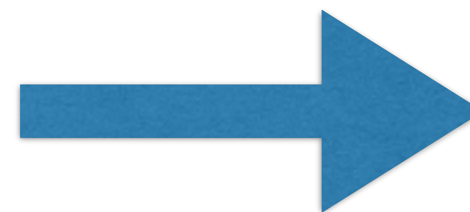
Declarative languages make it easier to explore equivalent computations to find the best one.

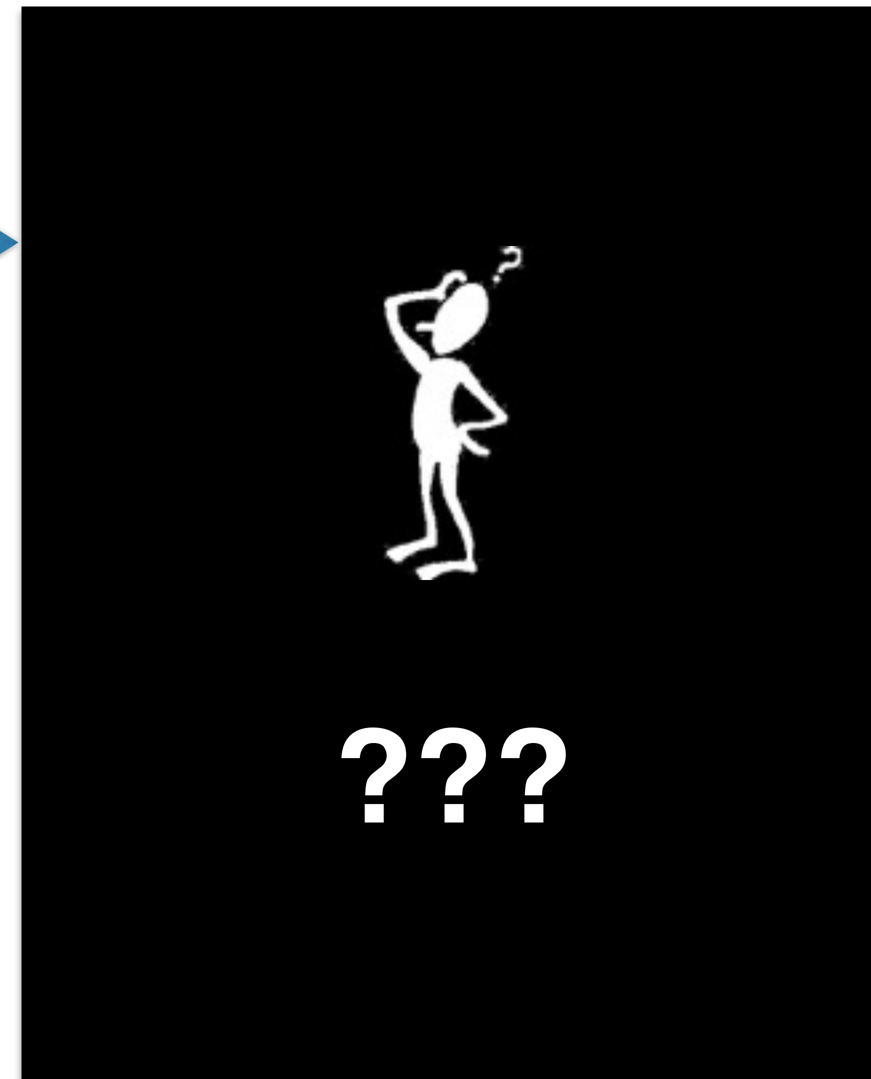
How do you build a query processor?

Project Outline

SQL Query

Parser & Translator

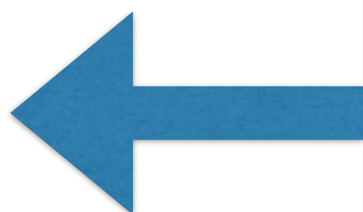




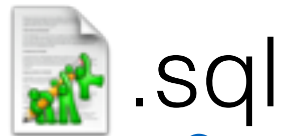
.sql

JSqlParser

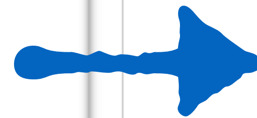
Query Result

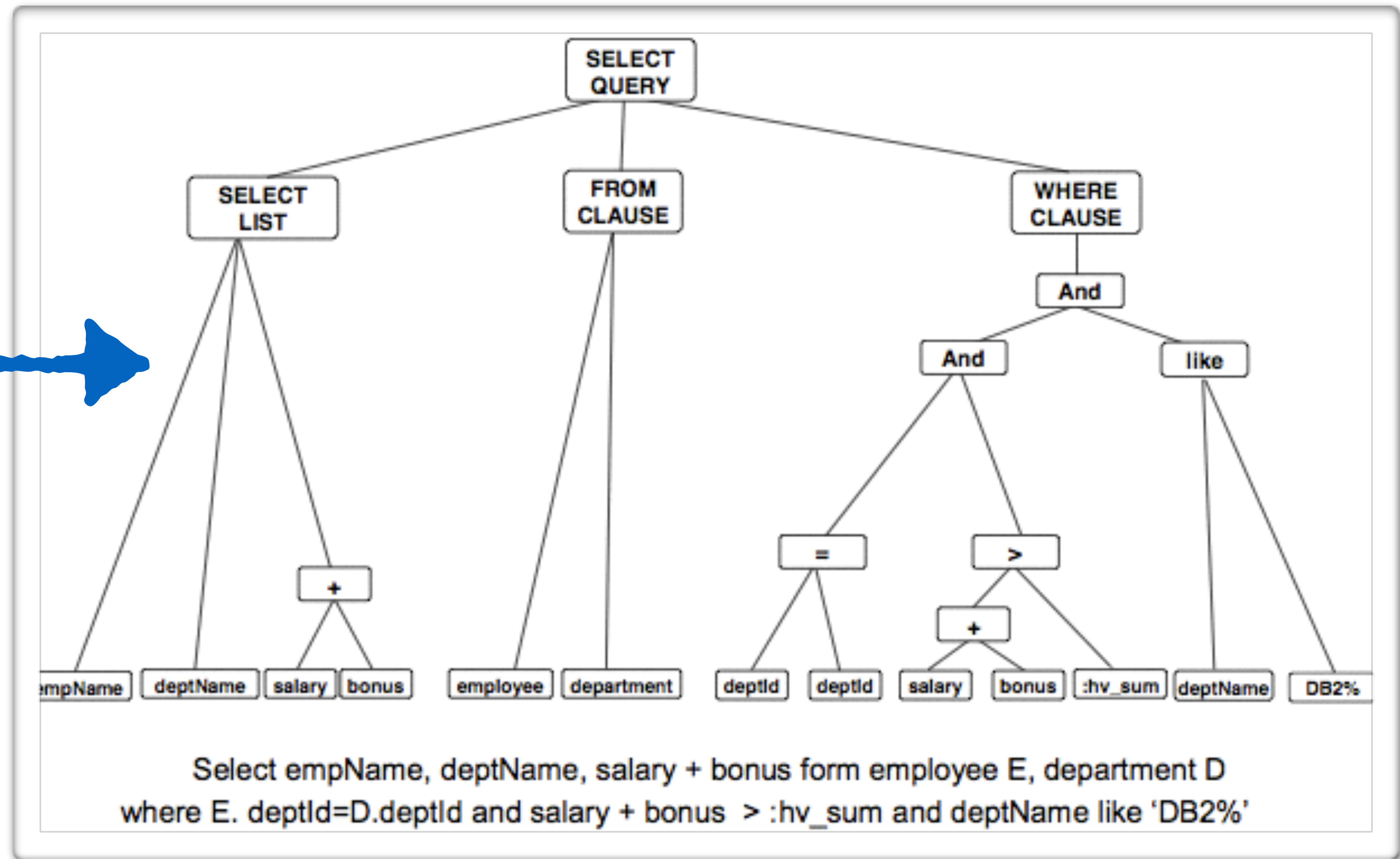


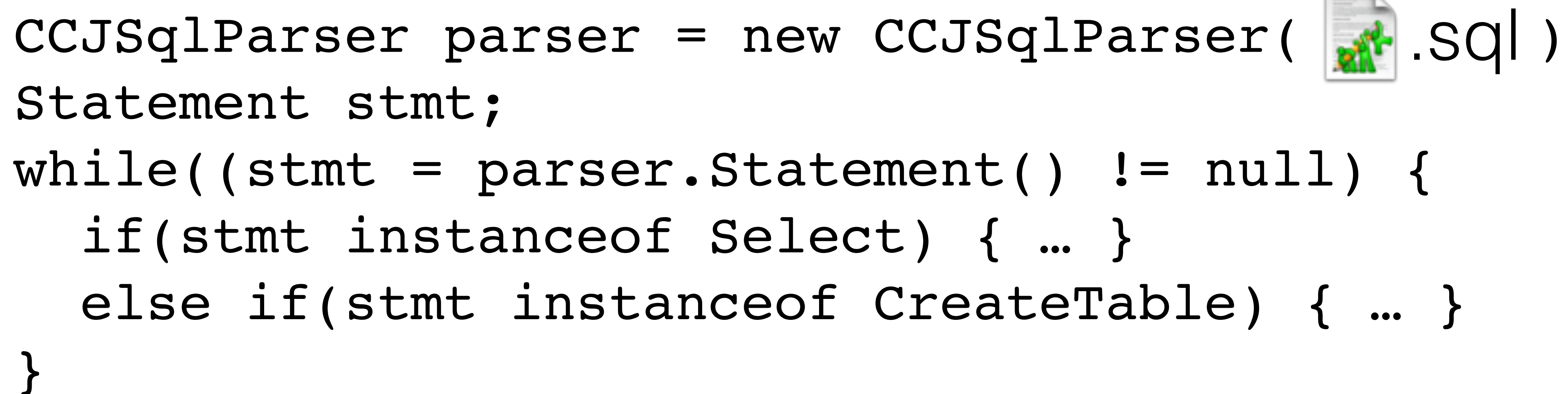
Trained Monkeys?


.sql

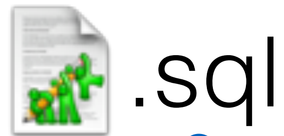
JSqlParser



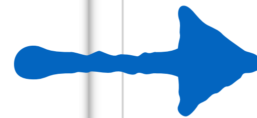


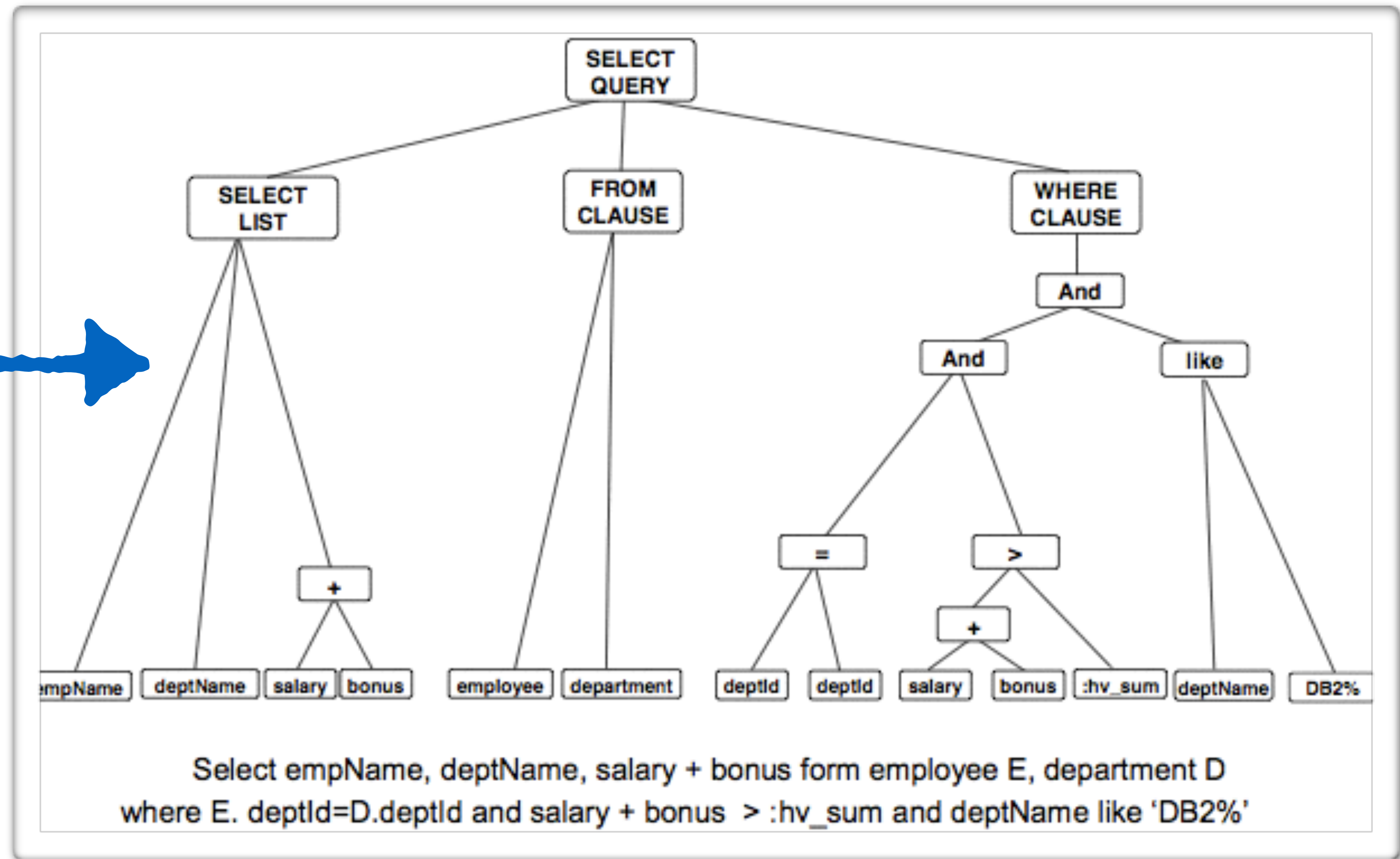


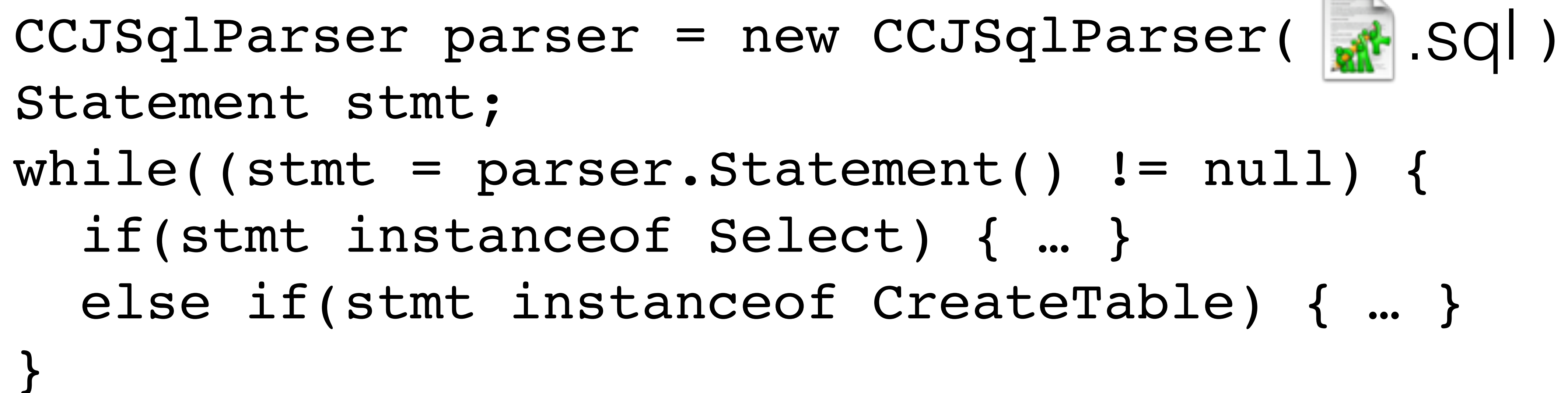
```
CCJSqlParser parser = new CCJSqlParser( .sql )
Statement stmt;
while((stmt = parser.Statement() != null) {
  if(stmt instanceof Select) { … }
  else if(stmt instanceof CreateTable) { … }
}
```


.sql

JSqlParser



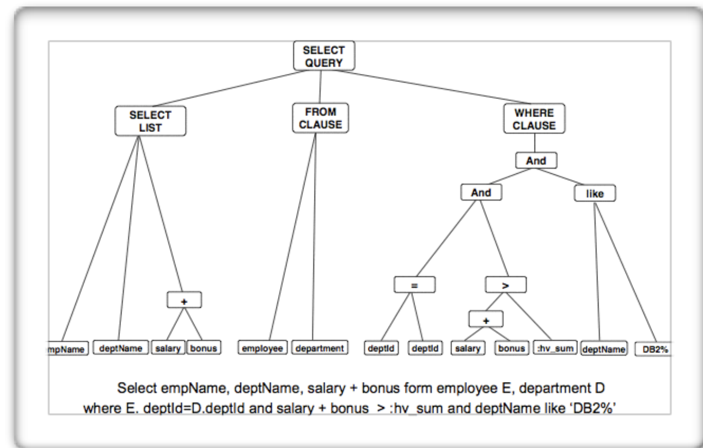




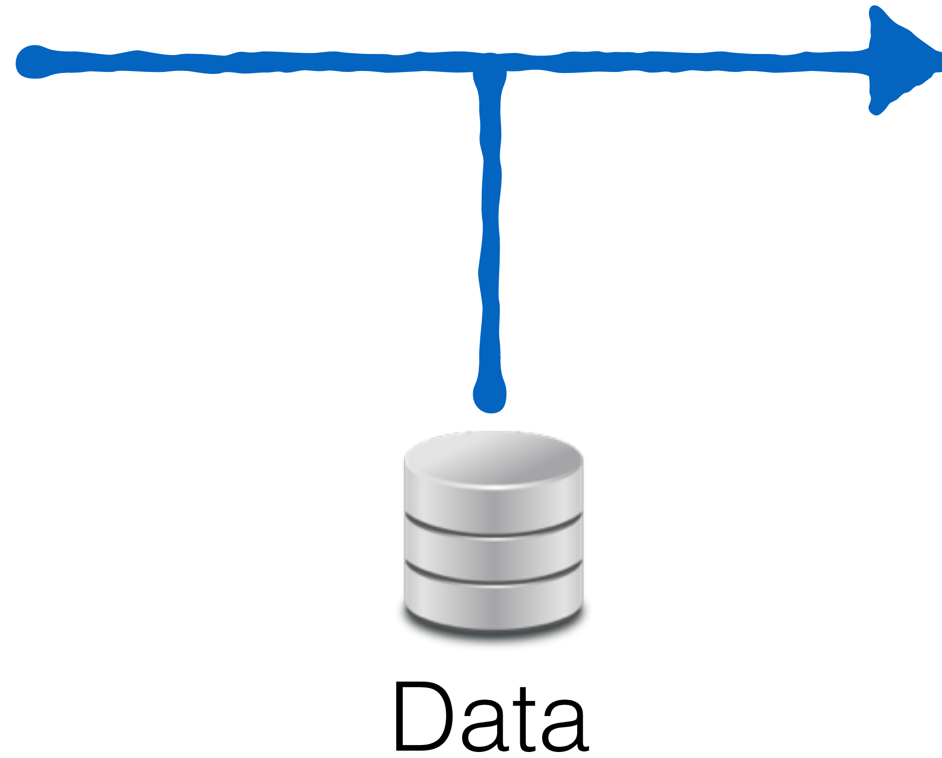
```
CCJSqlParser parser = new CCJSqlParser( .sql )
Statement stmt;
while((stmt = parser.Statement() != null) {
  if(stmt instanceof Select) { … }
  else if(stmt instanceof CreateTable) { … }
}
```

Now what?

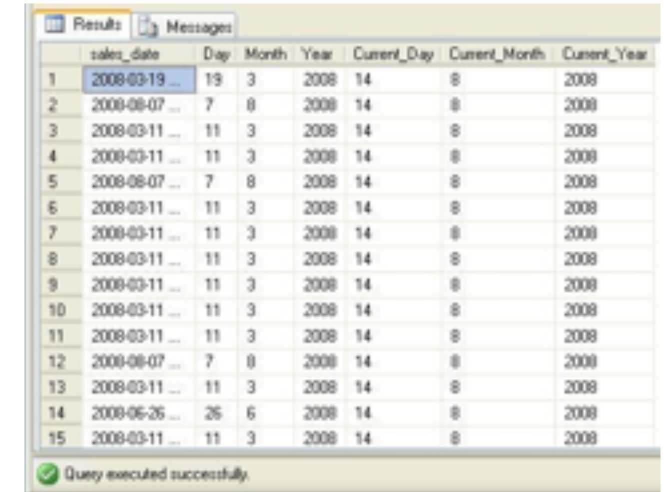
The Evaluation Pipeline



Parsed Query



Data



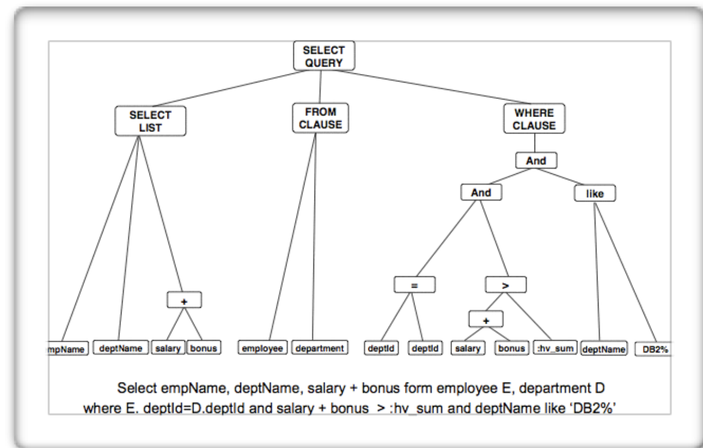
Results | Messages

	sales_date	Day	Month	Year	Current_Day	Current_Month	Current_Year
1	2008-03-19 ...	19	3	2008	14	8	2008
2	2008-08-07 ...	7	8	2008	14	8	2008
3	2008-03-11 ...	11	3	2008	14	8	2008
4	2008-03-11 ...	11	3	2008	14	8	2008
5	2008-08-07 ...	7	8	2008	14	8	2008
6	2008-03-11 ...	11	3	2008	14	8	2008
7	2008-03-11 ...	11	3	2008	14	8	2008
8	2008-03-11 ...	11	3	2008	14	8	2008
9	2008-03-11 ...	11	3	2008	14	8	2008
10	2008-03-11 ...	11	3	2008	14	8	2008
11	2008-03-11 ...	11	3	2008	14	8	2008
12	2008-08-07 ...	7	8	2008	14	8	2008
13	2008-03-11 ...	11	3	2008	14	8	2008
14	2008-06-26 ...	26	6	2008	14	8	2008
15	2008-03-11 ...	11	3	2008	14	8	2008

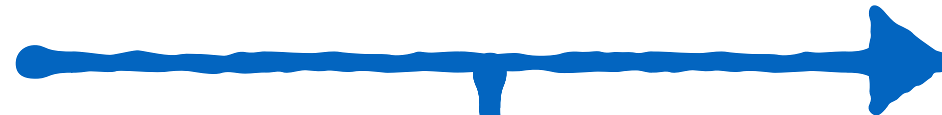
Query executed successfully.

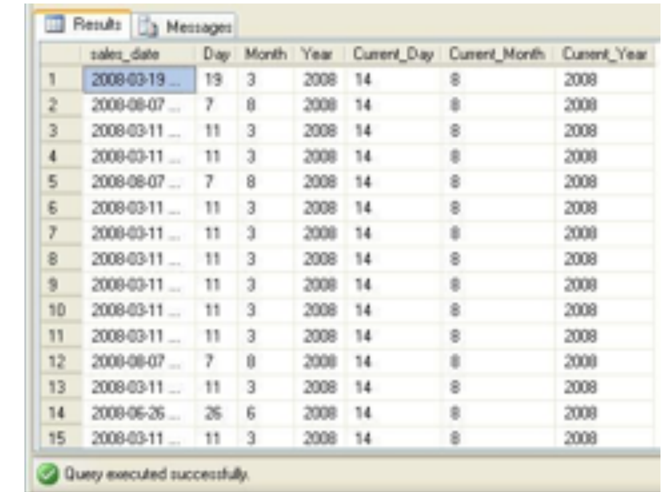
Results

The Evaluation Pipeline



Parsed Query





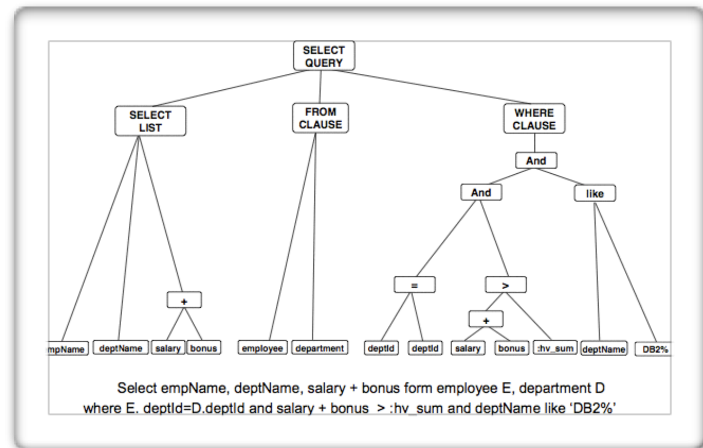
	sales_date	Day	Month	Year	Current_Day	Current_Month	Current_Year
1	2008-03-19 ...	19	3	2008	14	8	2008
2	2008-08-07 ...	7	8	2008	14	8	2008
3	2008-03-11 ...	11	3	2008	14	8	2008
4	2008-03-11 ...	11	3	2008	14	8	2008
5	2008-08-07 ...	7	8	2008	14	8	2008
6	2008-03-11 ...	11	3	2008	14	8	2008
7	2008-03-11 ...	11	3	2008	14	8	2008
8	2008-03-11 ...	11	3	2008	14	8	2008
9	2008-03-11 ...	11	3	2008	14	8	2008
10	2008-03-11 ...	11	3	2008	14	8	2008
11	2008-03-11 ...	11	3	2008	14	8	2008
12	2008-08-07 ...	7	8	2008	14	8	2008
13	2008-03-11 ...	11	3	2008	14	8	2008
14	2008-06-26 ...	26	6	2008	14	8	2008
15	2008-03-11 ...	11	3	2008	14	8	2008

Results

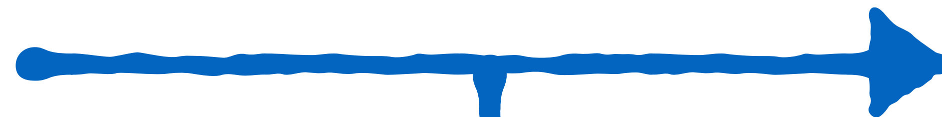
Data

Done?

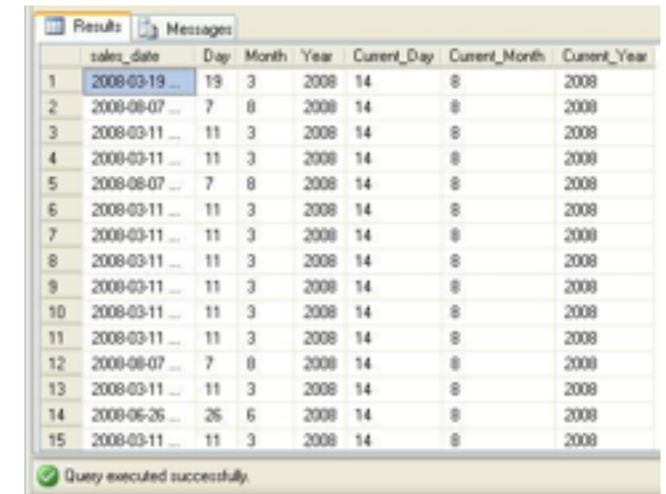
The Evaluation Pipeline



Parsed Query



Data



	sales_date	Day	Month	Year	Current_Day	Current_Month	Current_Year
1	2008-03-19 ...	19	3	2008	14	8	2008
2	2008-08-07 ...	7	8	2008	14	8	2008
3	2008-03-11 ...	11	3	2008	14	8	2008
4	2008-03-11 ...	11	3	2008	14	8	2008
5	2008-08-07 ...	7	8	2008	14	8	2008
6	2008-03-11 ...	11	3	2008	14	8	2008
7	2008-03-11 ...	11	3	2008	14	8	2008
8	2008-03-11 ...	11	3	2008	14	8	2008
9	2008-03-11 ...	11	3	2008	14	8	2008
10	2008-03-11 ...	11	3	2008	14	8	2008
11	2008-03-11 ...	11	3	2008	14	8	2008
12	2008-08-07 ...	7	8	2008	14	8	2008
13	2008-03-11 ...	11	3	2008	14	8	2008
14	2008-06-26 ...	26	6	2008	14	8	2008
15	2008-03-11 ...	11	3	2008	14	8	2008

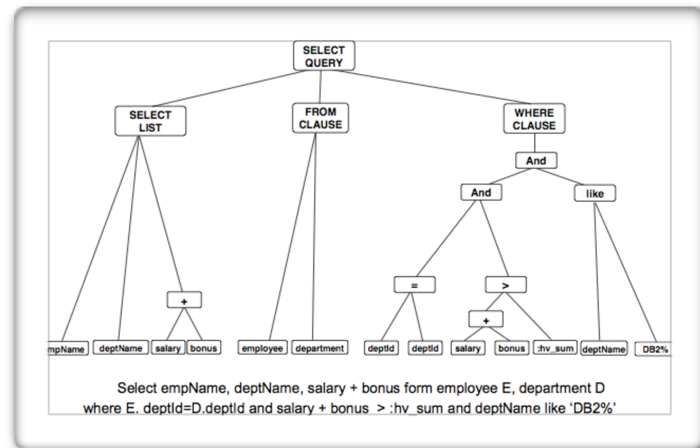
Query executed successfully.

Results

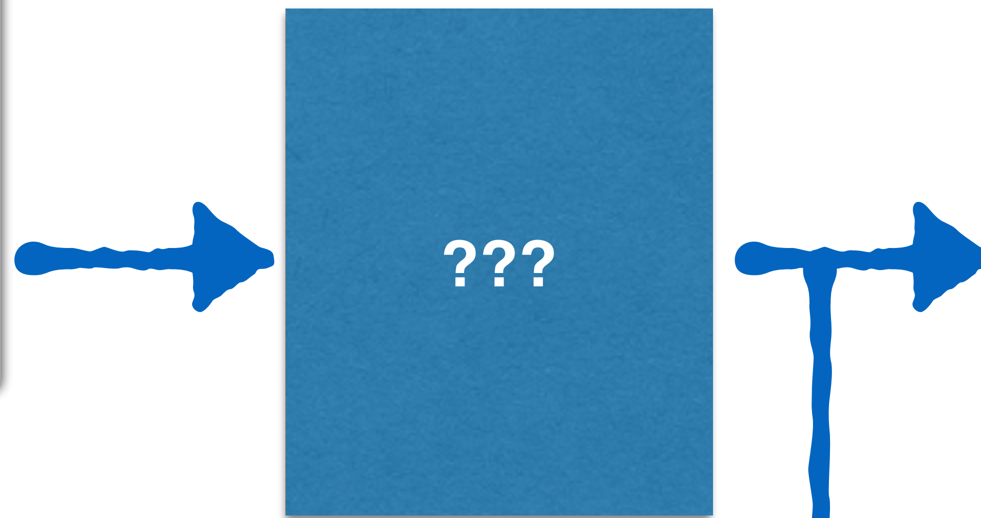
Done?

No! Evaluating SQL is HARD.

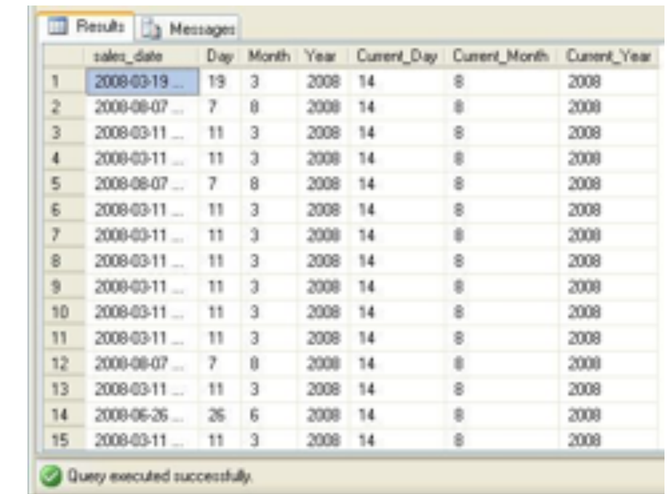
The Evaluation Pipeline



Parsed Query



Data



Results | Messages

	sales_date	Day	Month	Year	Current_Day	Current_Month	Current_Year
1	2008-03-19 ...	19	3	2008	14	8	2008
2	2008-08-07 ...	7	8	2008	14	8	2008
3	2008-03-11 ...	11	3	2008	14	8	2008
4	2008-03-11 ...	11	3	2008	14	8	2008
5	2008-08-07 ...	7	8	2008	14	8	2008
6	2008-03-11 ...	11	3	2008	14	8	2008
7	2008-03-11 ...	11	3	2008	14	8	2008
8	2008-03-11 ...	11	3	2008	14	8	2008
9	2008-03-11 ...	11	3	2008	14	8	2008
10	2008-03-11 ...	11	3	2008	14	8	2008
11	2008-03-11 ...	11	3	2008	14	8	2008
12	2008-08-07 ...	7	8	2008	14	8	2008
13	2008-03-11 ...	11	3	2008	14	8	2008
14	2008-06-26 ...	26	6	2008	14	8	2008
15	2008-03-11 ...	11	3	2008	14	8	2008

Query executed successfully.

Results

First, transform the query into something simpler.
(simpler, but equivalent)

What’s in the box?

Formal Query Languages

- Two mathematical query languages form the basis for user-facing languages (e.g., SQL):
  - Relational Algebra: Operational, useful for representing how queries are evaluated.
  - Relational Calculus: Declarative, useful for representing what a user wants rather than how to compute it.

Formal Query Languages

For Now

- Two mathematical query languages form the basis for user-facing languages (e.g., SQL):
  - Relational Algebra: Operational, useful for representing how queries are evaluated.

Preliminaries

Queries are applied to Relations

`Q(Officers, Ships, …)`

A Query works on **fixed** relation schemas.

… but runs on any relation instance

Preliminaries

Important: The result of a query is **also a relation**!

```
Q2(Officers, Q1(Ships))
```

Allows simple, **composable** query operators

Example Instances

Captains

FirstName,	LastName,	Rank,	Ship
[James,	Kirk,	4.0,	1701A]
[Jean Luc,	Picard,	4.0,	1701D]
[Benjamin,	Sisko,	3.0,	DS9]
[Kathryn,	Janeway,	4.0,	74656]
[Nerys,	Kira,	2.5,	75633]

FirstOfficers

FirstName,	LastName,	Rank,	Ship
[Spock,	NULL,	2.5,	1701A]
[William,	Riker,	2.5,	1701D]
[Nerys,	Kira,	2.5,	DS9]
[Chakotay,	NULL,	3.0,	74656]

Locations

Ship,	Location
[1701A,	Earth]
[1701D,	Risa]
[75633,	Bajor]
[DS9,	Bajor]

Relational Algebra

Operation	Sym	Meaning
Selection	σ	Select a subset of the input rows
Projection	π	Delete unwanted columns
Cross-product	x	Combine two relations
Set-difference	-	Tuples in Rel 1, but not Rel 2
Union	U	Tuples either in Rel 1 or in Rel 2

Also: Intersection, **Join**, Division, Renaming

(Not essential, but can be useful)

Relational Algebra

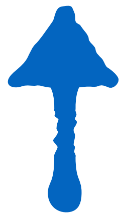
Each operation returns a relation!

Operations can be composed

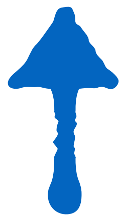
(Relational Algebra operators are **closed**)

Relational Algebra

Data



Relational Algebra

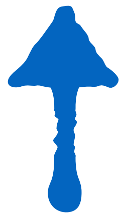


Data

Relational Algebra

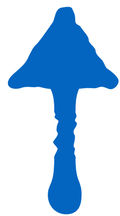
Data

A Set of Tuples



Relational Algebra

[Set] Relational Algebra



Data

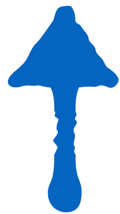
A Set of Tuples

Relational Algebra

Data

A Set of Tuples

A Bag of Tuples



Relational Algebra

[Set] Relational Algebra

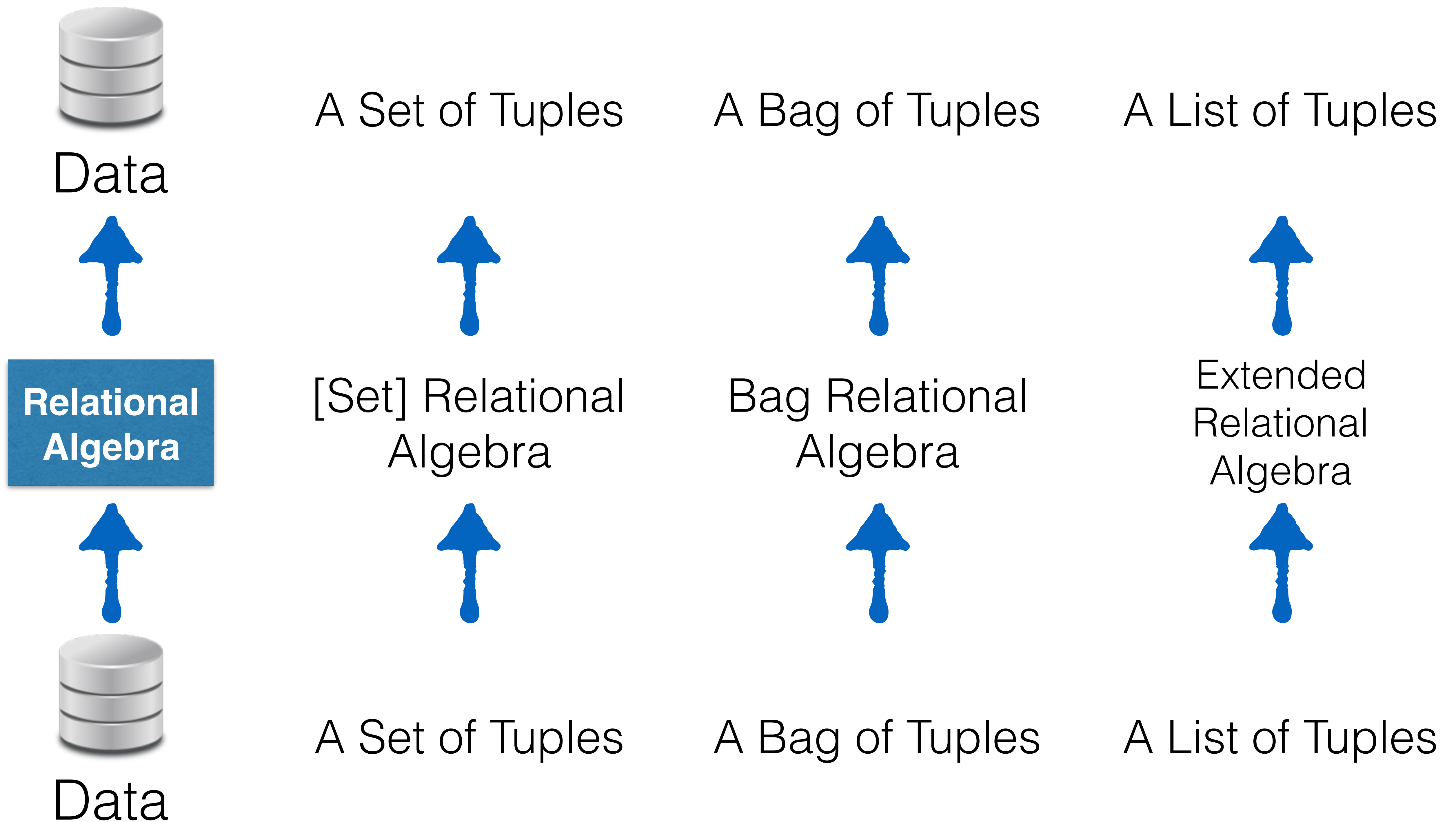
Bag Relational Algebra

Data

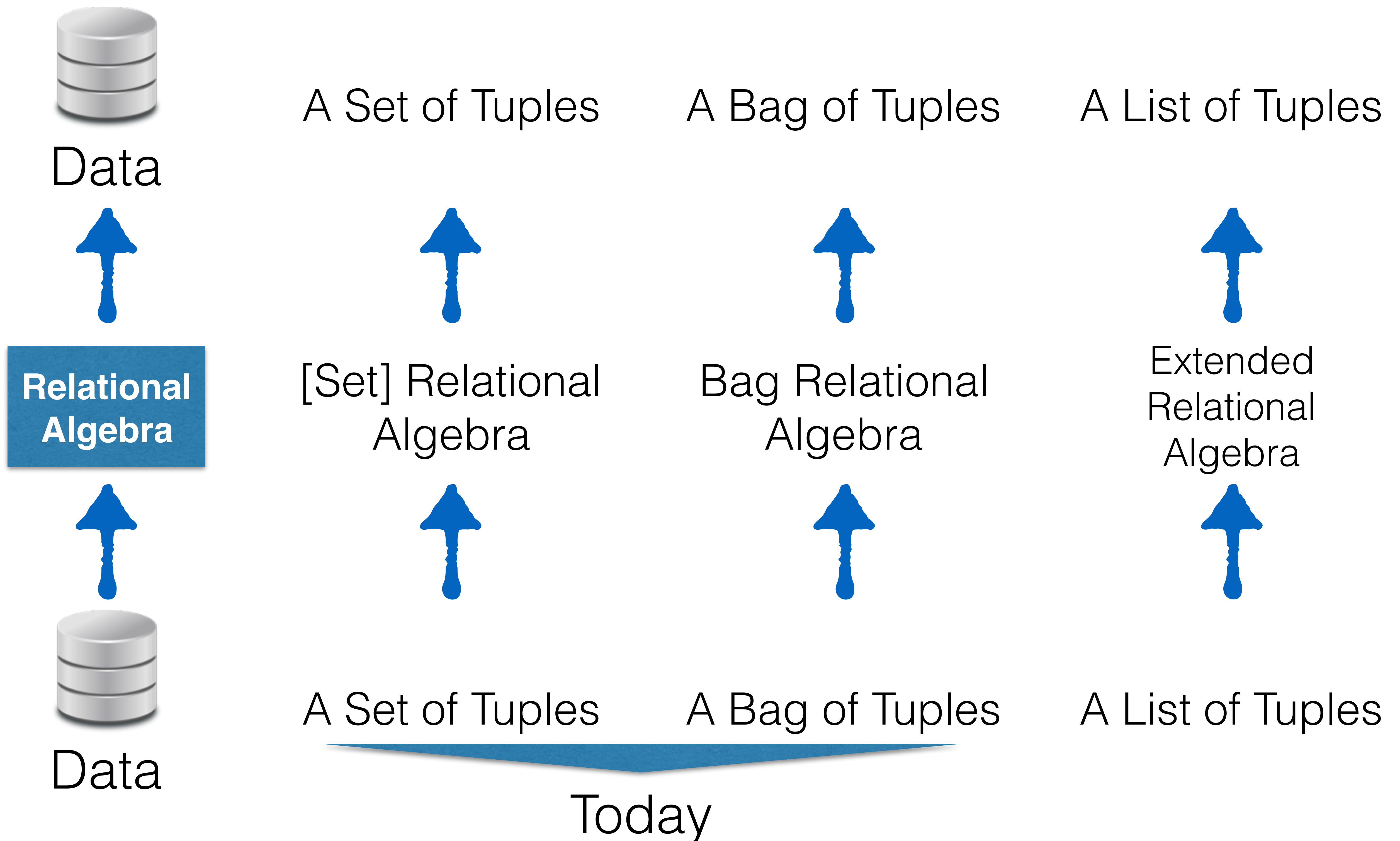
A Set of Tuples

A Bag of Tuples

Relational Algebra



Relational Algebra



Projection (π)

Delete attributes not in the **projection list**.

$\pi_{lastname, ship}(Captains)$

FirstName,	LastName,	Rank,	Ship
[Spock,	NULL,	2.5,	1701A]
[William,	Riker,	2.5,	1701D]
[Nerys,	Kira,	2.5,	DS9]
[Chakotay,	NULL,	3.0,	74656]

Projection (π)

Delete attributes not in the **projection list**.

$\pi_{lastname, ship}$(Captains)

LastName,	Ship
[Kirk,	1701A]
[Picard,	1701D]
[Sisko,	DS9]
[Janeway,	74656]
[Kira,	75633]

FirstName,	LastName,	Rank,	Ship
[Spock,	NULL,	2.5,	1701A]
[William,	Riker,	2.5,	1701D]
[Nerys,	Kira,	2.5,	DS9]
[Chakotay,	NULL,	3.0,	74656]

π_{rank}(FirstOfficers)

Projection (π)

Delete attributes not in the **projection list**.

$\pi_{lastname, ship}$(Captains)

LastName	Ship
Kirk	1701A
Picard	1701D
Sisko	DS9
Janeway	74656
Kira	75633

FirstName	LastName	Rank	Ship
Spock	NULL	2.5	1701A
William	Riker	2.5	1701D
Nerys	Kira	2.5	DS9
Chakotay	NULL	3.0	74656

π_{rank}(FirstOfficers)

Rank
2.5
3.0

Projection (π)

Delete attributes not in the **projection list**.

$\pi_{\text{lastname, ship}}(\text{Captains})$

LastName,	Ship
[Kirk,	1701A]
[Picard,	1701D]
[Sisko,	DS9]
[Janeway,	74656]
[Kira,	75633]

FirstName,	LastName,	Rank,	Ship
[Spock,	NULL,	2.5,	1701A]
[William,	Riker,	2.5,	1701D]
[Nerys,	Kira,	2.5,	DS9]
[Chakotay,	NULL,	3.0,	74656]

Why is this strange?

$\pi_{\text{rank}}(\text{FirstOfficers})$

Rank
[2.5]
[3.0]

Relational Algebra on Bags:

Bag Relational Algebra

Why?

Projection (π)

Queries are relations

What is this (query) relation's schema?

$\pi_{\text{lastname, ship}}(\text{Captains})$

Selection (σ)

Selects rows that satisfy the **selection condition**.

$\sigma_{rank < 3.5}(Captains)$

FirstName,	LastName,	Rank,	Ship
[Benjamin,	Sisko,	3.0,	DS9]
[Nerys,	Kira,	2.5,	75633]

When does selection need to eliminate duplicates?

$\pi_{lastname}(\sigma_{rank > 3.5}(Captains))$

LastName
[Kirk]
[Picard]
[Janeway]

What is the schema of these queries?

Union, Intersection, Set Difference

Each takes two relations that are **union-compatible**

(Both relations have the same number of fields with the same types)

Union: Return all tuples in **either relation**

$$\pi_{firstname,lastname}(\text{Captains}) \cup \pi_{firstname,lastname}(\text{FirstOfficers})$$

```
 FirstName, Lastname
[James,     Kirk    ]
[Jean Luc,  Picard  ]
[Benjamin,  Sisko   ]
[Kathryn,   Janeway ]
[Spock,     NULL    ]
[William,   Riker   ]
[Nerys,     Kira    ]
[Chakotay,  NULL    ]
```

Union, Intersection, Set Difference

Each takes two relations that are **union-compatible**
(Both relations have the same number of fields with the same types)

Intersection: Return all tuples in **both** relations

$\pi_{firstname,lastname}$ (Captains) ∩ $\pi_{firstname,lastname}$ (FirstOfficers)

```
FirstName, Lastname
[Nerys,     Kira    ]
```

Union, Intersection, Set Difference

Each takes two relations that are **union-compatible**
(Both relations have the same number of fields with the same types)

Set Difference: Return all tuples in the first but not the second relation

$$\pi_{firstname,lastname} (Captains) - \pi_{firstname,lastname} (FirstOfficers)$$

```
 FirstName,  LastName
[James,      Kirk    ]
[Jean Luc,   Picard  ]
[Benjamin,   Sisko   ]
[Kathryn,    Janeway ]
```

Union, Intersection, Set Difference

Each takes two relations that are **union-compatible**
(Both relations have the same number of fields with the same types)

What is the **schema** of the result
of any of these operators?

Cross Product

All pairs of tuples from both relations.

FirstOfficers x Locations

FirstName,	LastName,	Rank,	(Ship),	(Ship),	Location
[Spock,	NULL,	2.5,	1701A,	1701A,	Earth]
[Spock,	NULL,	2.5,	1701A,	1701D,	Risa]
[Spock,	NULL,	2.5,	1701A,	DS9,	Bajor]
[Spock,	NULL,	2.5,	1701A,	75633,	Bajor]
[William,	Riker,	2.5,	1701D,	1701A,	Earth]
[William,	Riker,	2.5,	1701D,	1701D,	Risa]
[William,	Riker,	2.5,	1701D,	DS9,	Bajor]
[William,	Riker,	2.5,	1701D,	75633,	Bajor]
[Nerys,	Kira,	2.5,	DS9,	1701A,	Earth]
[Nerys,	Kira,	2.5,	DS9,	1701D,	Risa]
[Nerys,	Kira,	2.5,	DS9,	DS9,	Bajor]
[Nerys,	Kira,	2.5,	DS9,	75633,	Bajor]
[Chakotay,	NULL,	3.0,	74656,	1701A,	Earth]
[Chakotay,	NULL,	3.0,	74656,	1701D,	Risa]
[Chakotay,	NULL,	3.0,	74656,	DS9,	Bajor]
[Chakotay,	NULL,	3.0,	74656,	75633,	Bajor]

Cross Product

All pairs of tuples from both relations.

FirstOfficers x Locations

What is the schema of this operator's result?

Cross Product

All pairs of tuples from both relations.

FirstOfficers x Locations

```
FirstName,  LastName, Rank, (Ship), (Ship), Location
                         ...
```

What is the schema of this operator's result?

Naming conflict: Both relations have a 'Ship' field

Renaming

$\rho_{\text{First, Last, Rank, OShip, LShip, Location}}$(FirstOfficers x Locations)

<u>First, Last, Rank, OShip, LShip, Location</u>

...

...

Cross Product

Can combine with selection

(FirstOfficers x Locations)

FirstName,	LastName,	Rank,	(Ship),	(Ship),	Location	
[Spock,	NULL,	2.5,	1701A,	1701A,	Earth	]
[Spock,	NULL,	2.5,	1701A,	1701D,	Risa	]
[Spock,	NULL,	2.5,	1701A,	DS9,	Bajor	]
[Spock,	NULL,	2.5,	1701A,	75633,	Bajor	]
[William,	Riker,	2.5,	1701D,	1701A,	Earth	]
[William,	Riker,	2.5,	1701D,	1701D,	Risa	]
[William,	Riker,	2.5,	1701D,	DS9,	Bajor	]
[William,	Riker,	2.5,	1701D,	75633,	Bajor	]
[Nerys,	Kira,	2.5,	DS9,	1701A,	Earth	]
[Nerys,	Kira,	2.5,	DS9,	1701D,	Risa	]
[Nerys,	Kira,	2.5,	DS9,	DS9,	Bajor	]
[Nerys,	Kira,	2.5,	DS9,	75633,	Bajor	]
[Chakotay,	NULL,	3.0,	74656,	1701A,	Earth	]
[Chakotay,	NULL,	3.0,	74656,	1701D,	Risa	]
[Chakotay,	NULL,	3.0,	74656,	DS9,	Bajor	]
[Chakotay,	NULL,	3.0,	74656,	75633,	Bajor	]

Cross Product

Can combine with selection

$\sigma_{[4] = [5]}$(FirstOfficers x Locations)

FirstName,	LastName,	Rank,	(Ship),	(Ship),	Location
[Spock,	NULL,	2.5,	1701A,	1701A,	Earth]
[William,	Riker,	2.5,	1701D,	1701D,	Risa]
[Nerys,	Kira,	2.5,	DS9,	DS9,	Bajor]
[Chakotay,	NULL,	3.0,	74656,	75633,	Bajor]

Join

Pair tuples according to a **join condition**.

$$\pi_{FirstName,Rank}(FO) \bowtie_{FO.Rank < C.Rank} \pi_{FirstName,Rank}(C)$$

```
 FirstName, Rank, FirstName, Rank
[Spock,      2.5,  James,      4.0 ]
[Spock,      2.5,  Jean Luc,   4.0 ]
[Spock,      2.5,  Benjamin,   3.0 ]
[Spock,      2.5,  Kathryn,    4.0 ]
[William,    2.5,  James,      4.0 ]
[William,    2.5,  Jean Luc,   4.0 ]
[William,    2.5,  Benjamin,   3.0 ]
[William,    2.5,  Kathryn,    4.0 ]
[Nerys,      2.5,  James,      4.0 ]
[Nerys,      2.5,  Jean Luc,   4.0 ]
[Nerys,      2.5,  Benjamin,   3.0 ]
[Nerys,      2.5,  Kathryn,    4.0 ]
[Chakotay,   3.0,  James,      4.0 ]
[Chakotay,   3.0,  Jean Luc,   4.0 ]
[Chakotay,   3.0,  Kathryn,    4.0 ]
```

Result schema is like the cross product

There are fewer tuples in the result than cross-products: we can often compute joins more efficiently

(these are sometimes called 'theta-joins')

Equi-Joins

A special case of joins where the condition contains *only equalities*.

$$FO \bowtie_{FO.Ship\ =\ Loc.Ship} Loc$$

```
 FirstName,  LastName, Rank, (Ship), (Ship), Location
[Spock,      NULL,     2.5,  1701A,  1701A,  Earth   ]
[William,    Riker,    2.5,  1701D,  1701D,  Risa    ]
[Nerys,      Kira,     2.5,  DS9,    DS9,    Bajor   ]
```

Result **schema** is like the cross product, but only one copy of each field with an equality

Equi-Joins

A special case of joins where the condition contains *only equalities*.

$$FO \bowtie_{Ship} Loc$$

FirstName,	LastName,	Rank,	(Ship),	(Ship),	Location
[Spock,	NULL,	2.5,	1701A,	1701A,	Earth]
[William,	Riker,	2.5,	1701D,	1701D,	Risa]
[Nerys,	Kira,	2.5,	DS9,	DS9,	Bajor]

Result **schema** is like the cross product, but only one copy of each field with an equality

Equi-Joins

A special case of joins where the condition contains *only equalities*.

$FO \bowtie_{Ship} Loc$

```
 FirstName,  LastName, Rank, (Ship), (Ship), Location
[Spock,      NULL,     2.5,  1701A,  1701A,  Earth   ]
[William,    Riker,    2.5,  1701D,  1701D,  Risa    ]
[Nerys,      Kira,     2.5,  DS9,    DS9,    Bajor   ]
```

Result **schema** is like the cross product, but only one copy of each field with an equality

Natural Joins: Equi-Joins on all fields with the same name

$FirstOfficers \bowtie_{Ship} Locations = FirstOfficers \bowtie Locations$

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	
Projection	π	
Cross-product	x	
Set-difference	-	
Union	U	
Join	$\bowtie$	

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	No
Projection	π	
Cross-product	x	
Set-difference	-	
Union	U	
Join	$\bowtie$	

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	No
Projection	π	Yes
Cross-product	x	
Set-difference	-	
Union	U	
Join	$\bowtie$	

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	No
Projection	π	Yes
Cross-product	x	No
Set-difference	-	
Union	U	
Join	$\bowtie$	

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	No
Projection	π	Yes
Cross-product	x	No
Set-difference	-	No
Union	U	
Join	$\bowtie$	

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	No
Projection	π	Yes
Cross-product	x	No
Set-difference	-	No
Union	U	Yes
Join	$\bowtie$	

Which operators can create duplicates?

(Which operators behave differently in Set- and Bag-RA?)

Selection	σ	No
Projection	π	Yes
Cross-product	x	No
Set-difference	-	No
Union	U	Yes
Join	$\bowtie$	No

Group Work

Find the Last Names of all Captains of a Ship located on 'Bajor'

Come up with at least 2 distinct queries that compute this. Which are the most efficient and why?

Captains

FirstName,	LastName,	Rank,	Ship
[James,	Kirk,	4.0,	1701A]
[Jean Luc,	Picard,	4.0,	1701D]
[Benjamin,	Sisko,	3.0,	DS9]
[Kathryn,	Janeway,	4.0,	74656]
[Nerys,	Kira,	2.5,	75633]

Locations

Ship,	Location
[1701A,	Earth]
[1701D,	Risa]
[75633,	Bajor]
[DS9,	Bajor]

Find the Last Names of all Captains of a Ship located on ‘Bajor’

Solution 1:

$$\pi_{LastName}(\sigma_{Location=\text{‘Bajor’}}(Locations) \bowtie Captains)$$

Solution 2:

$$Temp1 = \sigma_{Location=\text{‘Bajor’}}(Locations))$$

$$Temp2 = Temp1 \bowtie (\pi_{(LastName,Ship)} Captains)$$

$$\pi_{LastName}(Temp2)$$

Solution 3:

$$\pi_{LastName}(\sigma_{Location=\text{‘Bajor’}}(Captains \bowtie Locations))$$

Find the Last Names of all Captains of a Ship located on 'Bajor'

Solution 1:

$$\pi_{LastName}(\sigma_{Location= \text{'Bajor'}}(Locations) \bowtie Captains)$$

Solution 2:

$$Temp1 = \sigma_{Location= \text{'Bajor'}} (Locations))$$

$$Temp2 = Temp1 \bowtie (\pi_{(LastName,Ship)} Captains)$$

$$\pi_{LastName}(Temp2)$$

Solution 3:

$$\pi_{LastName}(\sigma_{Location= \text{'Bajor'}}(Captains \bowtie Locations))$$

These are all equivalent queries!

Division

Not typically supported as a primitive operator,
but useful for expressing queries like:

Find officers who have visited **all** planets

Relation V has fields Name, Planet

Relation P has field Planet

V / P = { Name | For each Planet in P, <Name, Planet> is in V }

All Names in the Visited table who have visited
every Planet in the Planets table

Division

```
 Name,    Planet
[Kirk,    Earth    ]
[Kirk,    Vulcan   ]
[Kirk,    Kronos   ]
[Spock,   Earth    ]
[Spock,   Vulcan   ]
[Spock,   Romulus  ]
[McCoy,   Earth    ]
[McCoy,   Vulcan   ]
[Scotty,  Earth    ]
```

V

```
 Planet
[Earth  ]
```

P1

```
 Planet
[Earth  ]
[Vulcan]
```

P2

```
 Planet
[Earth   ]
[Vulcan  ]
[Romulus]
```

P3

```
 Name
[Kirk  ]
[Spock ]
[McCoy ]
[Scotty]
```

V/P1

```
 Name
[Kirk  ]
[Spock ]
[McCoy ]
```

V/P2

```
 Name
[Spock ]
```

V/P2

Division

- Not an essential operation, but a useful shorthand.
  - Also true of joins, but joins are so common that most systems implement them specifically
- How do we implement division using other operators?
  - Try it! (Group Work)

Find the Last Names of all captains of a ship located in Federation Territories

Affiliation

```
 Location, Affiliation
[Earth,    Federation ]
[Risa,     Federation ]
[Bajor,    Bajor      ]
```

$\pi_{LastName}(\sigma_{Affiliation='Federation'}(Loc) \bowtie Affil \bowtie Cap)$

σ

Find the Last Names of all captains of a ship located in Federation Territories

Affiliation

Location,	Affiliation
[Earth,	Federation]
[Risa,	Federation]
[Bajor,	Bajor]

$$\pi_{LastName}(\sigma_{Affiliation=\text{`Federation'}}(Loc) \bowtie Affil \bowtie Cap)$$

But we can do this more efficiently:

$$\pi_{LastName}(\pi_{Ship}(\pi_{Location}(\sigma_{Affiliation=\text{`Federation'}}(Loc))) \bowtie Affil) \bowtie Cap)$$

A query optimizer can find this, given the first solution

Relational Algebra

- A simple way to think about and work with set-at-a-time computations.
  - … simple → easy to evaluate
  - … simple → easy to optimize
- Next time…
  - SQL