Relational Algebra

Ramakrishnan & Gehrke Ch 4

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

DISTINCT

Why do you explicitly indicate that you want duplicate elimination in SQL?

Review

What does the following query compute?

```
SELECT DISTINCT O.Rank
FROM   Officers O, Ships S
WHERE  O.Ship = S.ID
  AND  S.Name = 'Enterprise'
```

Review

Are these queries equivalent?

```
SELECT DISTINCT O.Ship
FROM   Officers O, Visited V
WHERE  O.ID = V.Officer
  AND  (V.Visited = ‘Vulcan’ OR
        V.Visited = ‘Andoria’)
```

```
SELECT O.Ship
FROM   Officers O,
WHERE  O.ID IN (
         SELECT V.Officer
         FROM Visited V
         WHERE V.Visited = ‘Vulcan’
            OR V.Visited = ‘Andoria’)
```

Homework 1

- To be posted tonight
- Due in 1 week
- Covers SQL and the Relational Model

Relational Query Languages

- **Recall:** Query Languages allow manipulation and **retrieval** of data from a database.
- The relational model supports many simple and powerful query languages.
  - A formal foundation based on logic supports many different optimizations
- Query Languages **are not** Programming Languages
  - … not generally 'Turing Complete'
  - … not generally intended for complex calculuations
  - … do support easy, efficient access to big data

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

As in SQL, the result of a query is **also a relation**!

`Q2(Officers, Q1(Ships))`

The schema of a query result is defined by query language constructs

Field Notation

- Positional Notation (e.g., Field #2)
  - Easier to work with for formal semantics
- Field-Name Notation (e.g., FirstName)
  - More readable.
- SQL supports both.

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 very useful)

Relational Algebra

Each operation returns a relation!

Operations can be composed

(Relational Algebra operators are **closed**)

Projection (π)

Delete attributes not in the *projection list.*

$\pi_{\text{lastname, ship}}$(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.*

$\pi_{\text{lastname, ship}}$(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*.

$\pi_{\text{lastname, ship}}$(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]

Why is this strange?

π_{rank}(FirstOfficers)

Rank
[2.5]
[3.0]

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]

Why is this strange?

π_{rank}(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*.

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

Selection (σ)

Selects rows that satisfy the *selection condition*.

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

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

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

Selection (σ)

Selects rows that satisfy the *selection condition*.

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

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

When does selection need to eliminate duplicates?

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

LastName
[Kirk]
[Picard]
[Janeway]

Selection (σ)

Selects rows that satisfy the *selection condition*.

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

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

When does selection need to eliminate duplicates?

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

```
 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

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

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

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

```
 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

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

```
First,  Last, Rank, OShip,  LShip,  Location
                     ...
                     ...
```

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*.

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

```
 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]
[Chakotay,	NULL,	3.0,	74656,	75633,	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*.

$$\text{FO} \bowtie_{\text{Ship}} \text{Loc}$$

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]
[Chakotay,	NULL,	3.0,	74656,	75633,	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]
[Chakotay,	NULL,	3.0,	74656,	75633,	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

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

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)

Division

Compute the set of tuples that are disqualified first!

$$\pi_{Name}(((\pi_{Name} V) \times P) - V)$$

Division

Compute the set of tuples that are disqualified first!

$$\pi_{Name}(\overbrace{((\pi_{Name} V) \times P)}^{\text{[All Names] x [All Planets]}} - V)$$

Division

Compute the set of tuples that are disqualified first!

$$\pi_{Name}(\overbrace{((\pi_{Name} V) \times P) - V}^{\text{All planets that each person hasn't visited}})$$

Division

Compute the set of tuples that are disqualified first!

All people that haven't visited a planet

$$\pi_{Name}(((\pi_{Name} V) \times P) - V)$$

Division

Compute the set of tuples that are disqualified first!

All people that haven't visited a planet

$$\pi_{Name}(((\pi_{Name} V) \times P) - V)$$

$$V / P = (\pi_{Name} V) - \pi_{Name}(((\pi_{Name} V) \times P) - V)$$

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:

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

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

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

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:

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

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

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

Solution 3:

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

These are all equivalent queries!

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

Affiliation

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

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

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

Affiliation

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

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

But we can do this more efficiently:

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

A query optimizer can find this, given the first solution

Summary

- The relational model has rigorously defined query languages that are simple and powerful
- Relational Algebra is operational
  - … and a useful internal representation of query evaluation plans.
- There are many ways of expressing any given query.
  - Query optimizers should pick the most efficient one.