Relational algebra

What's relational algebra?

- Defines basic operations on *relation instances*
  - composition of operations to form queries
  - basis for SQL
- Useful to represent *execution plans*
  - what are the operations needed to execute a query
  - what is the order of execution of these operations

Basic operations

- Selection σ (choose subset of rows)
- Projection Π (choose subset of columns)
- Cross product $\times$ ←
- Union $\cup$ ←
- Difference $-$ ←
- Rename ρ
- Join $\bowtie$

Relations = sets of tuples

Rename

$$\rho_{R(A1,A2,\ldots)}(S)$$

$$\rho_{Stars(Name,Age,City)}(Actors)$$

Actors (Name, Age, Addr)

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Stars

Name	Age	City
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Selection (1/2)

$$R1 = \sigma_C(R2)$$

C is a condition on attributes of $R2$

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return all actors living in Mumbai

$$R1 = \sigma_{Addr='Mumbai'}(Actors)$$

R1

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Abhishek Bachchan	45	Mumbai

Selection (2/2)

$$R1 = \sigma_C(R2)$$

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return all actors whose age is more than 42.

$\sigma_{Age>42}(Actors)$

Name	Age	Addr
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return all actors whose age is more than 42 and who live in Mumbai

$\sigma_{Age>42 \text{ and } Addr='Mumbai'}(Actors)$

Name	Age	Addr
Abhishek Bachchan	45	Mumbai

Projection (1/2)

$$R1 = \Pi_L(R2)$$

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return the name and age of all actors

$$\Pi_{Name,Age}(Actors)$$

Name	Age
Priyanka Chopra	38
Anthony Hopkins	81
Bill Nighy	69
Abhishek Bachchan	45

Projection (2/2)

$$R1 = \Pi_L(R2)$$

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return the addresses of the actors

$$\Pi_{Addr}(Actors)$$

Addr
Mumbai
LA

Relation = set of tuple

Duplicate elimination under set semantics

Cross product

$$R3 = R1 \times R2$$

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

$4 \times 4 = 16$

Movies

Name	Year	Title
Priyanka Chopra	2011	Don-II
Anthony Hopkins	2011	MI-IV
Bill Nighy	2009	Valkyrie
Abhishek Bachchan	2010	Raavan

Actor.name	Age	Addr	Movies.Name	Year	Title
Priyanka Chopra	38	Mumbai	Priyanka Chopra	2011	Don-II
…15 more rows…					

Joins (1/2)

$$R3 = R1 \bowtie_{C} R2$$

C is a condition on attributes of $R1$ and/or $R2$

Actors

Name	Age	Addr
PC	38	Mumbai
AH	81	LA
BN	69	LA
AB	45	Mumbai

Movies

Name	Year	Title
PC	2011	Don-II
AH	2011	Thor: R
BN	2009	Valkyrie
AB	2010	Raavan

Return all information about actors and their movies

$$Actors \bowtie_{A.Name=M.Name} Movies$$

Name	Age	Addr	Year	Title
PC	38	Mumbai	2011	Don-II
AH	81	LA	2011	Thor: R
BN	69	LA	2009	Valkyrie
AB	45	Mumbai	2010	Raavan

Joins (2/2)

- Natural joins
  - implicitly compares attributes of the same name for equality

$R1_{(\cdot F)} \bowtie_{=} R_2 (\cdot F \cdot)$

- Theta join
  - conditions not restricted to equality

$R1 \bowtie_{\boxed{A1<A2 \text{ AND } A3=A4 \ \vdots}} R2$

Equi-join

- Left-outer/right-outer/full-outer joins
  - non-matching tuples are still returned

$R1 \overset{L}{\bowtie} \overset{R}{\bowtie} R_2$

- Self-join
  - table joining with itself

$R1 \bowtie_c R1$

Left outer joins

$$R = X ⟕ Y$$

Actors

Name	Age	Addr
PC	38	Mumbai
AH	81	LA
BN	69	LA
AB	45	Mumbai

Movies

Name	Year	Title
PC	2011	Don-II
AH	2011	Thor: R
AB	2010	Raavan

Return all information about actors and their movies

$$Actors ⟕_{A.Name=M.Name} Movies$$

Name	Age	Addr	Year	Title
PC	38	Mumbai	2011	Don-II
AH	81	LA	2011	Thor: R
BN	69	LA	*null*	*null*
AB	45	Mumbai	2010	Raavan

Self Join

Return all grandparents and their grand children

Actors

Name	Age	Addr	Parent
PC	38	Mumbai	Madhu
AH	81	LA	Muriel
BN	69	LA	Catherine
AB	45	Mumbai	Jaya
Jaya	63	Mumbai	Indira

Actors_1

Name	Age	Addr	Parent
PC	38	Mumbai	Madhu
AH	81	LA	Muriel
BN	69	LA	Catherine
AB	45	Mumbai	Jaya
Jaya	63	Mumbai	Indira

Actors ⋈ Actors_1

Actors.Parent = Actors_1.Name

COMPOSITION OF OPERATORS AND AGGREGATES

Composition of operators (1/2)

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return the names and addresses of actors over 42

$$\Pi_{Name,Addr}\left(\sigma_{Age>42}(Actors)\right)$$

Return the names of actors over 42 who live in Mumbai

$$\Pi_{Name,Addr}\left(\sigma_{Age>42 \text{ AND } Addr='Mumbai'}(Actors)\right)$$

Composition of operators (2/2)

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Movies

Name	Year	Title
Priyanka Chopra	2011	Don-II
Anthony Hopkins	2011	MI-IV
Bill Nighy	2009	Valkyrie
Abhishek Bachchan	2010	Raavan

Relational algebra exp

Return the names of actors below the age of 50 who have acted in a movie in 2011

$$\Pi_{Name}(\sigma_{Age<50 \text{ AND } Year=2011}(Actors \bowtie_{A.Name=M.Name} Movies))$$

Actor

Movie

Intermediate result relation

$$Allmovies = Actors \bowtie_{A.Name=M.Name} Movies$$

$$Movies1 = \sigma_{Age<50 \text{ AND } Year=2011}(AllMovies)$$

$$\boxed{Result} = \Pi_{Name}(Movies1)$$

Notation (1/2)

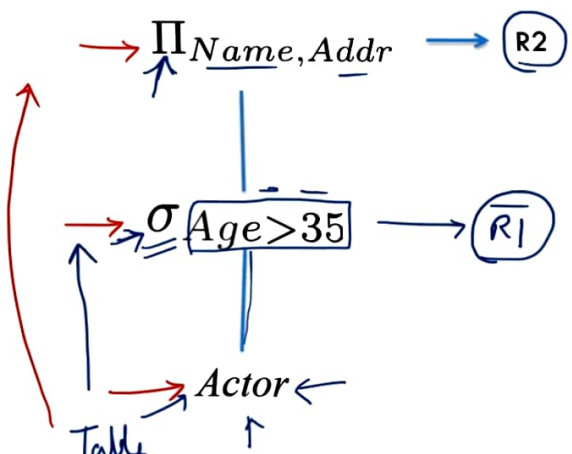
Return the names and addresses of actors over 35

$$\Pi_{Name,Addr}(\sigma_{Age>35}(Actor))$$

$$R1 = \sigma_{Age>35}(Actor)$$

$$R2 = \Pi_{Name,Addr}(R1)$$

Query plan

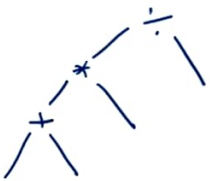


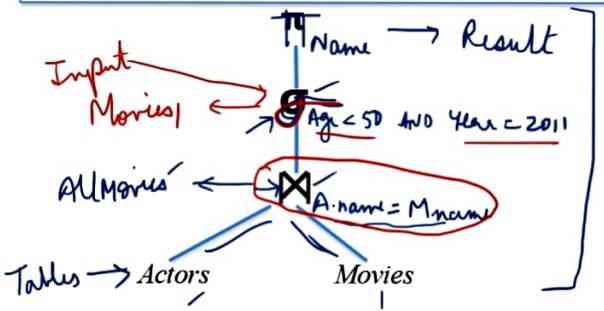
Notation (2/2)

Return the names of actors below the age of 50 who have acted in a movie in 2011

$$\Pi_{Name}(\sigma_{Age<50 \text{ AND } Year=2011}(Actors \bowtie_{A.Name=M.Name} Movies))$$

$$\begin{aligned} Allmovies &= Actors \bowtie_{A.Name=M.Name} Movies \\ Movies1 &= \sigma_{Age<50 \text{ AND } Year=2011}(AllMovies) \\ Result &= \Pi_{Name}(Movies1) \end{aligned}$$





Logical Query plan

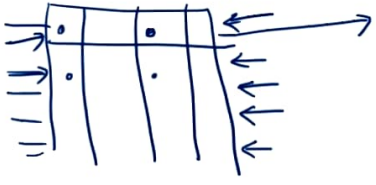
Relational algebra for bags

Set → no duplicates

Bag → multiplicities

{a, a, a, b, b, c, c}

- Efficiency issues if we consider relations as sets
  - Extra effort to eliminate duplicates
- Select, project, join (SPJ) work exactly the same
  - Applied to one tuple at a time
- Set operations become bag operations
  - Need to be careful about semantics
  - Union, Intersection, Difference



{a, a} ∪ {a} = {a, a, a}

More operators

- Duplicate elimination $\delta(R)$
- Aggregation
  - count, min, max, sum, avg
- Grouping γ
- Sorting τ

Aggregation and grouping (1/2)

- Grouping $\gamma_L(R)$
  - L is a list of grouping attributes and/or aggregate operators

Movies

Movie	City	Boxoffice
MI-IV	LA	2,000,000
Don-II	LA	500,000
MI-IV	NY	3,000,000

Return total boxoffice returns per movie

Aggregate

grouping attribute

$$\gamma_{Movie, Sum(Boxoffice)}(Movies)$$

Movie	City	Boxoffice
MI-IV	LA	2,000,000
MI-IV	NY	3,000,000
Don-II	LA	500,000

Movie	Boxoffice
MI-IV	5,000,000
Don-II	500,000

Aggregation and grouping (2/2)

- Grouping $\gamma_L(R)$
  - L is a list of grouping attributes and/or aggregate operators

Movie	City	Boxoffice
MI-IV	LA	2,000,000
Don-II	LA	500,000
MI-IV	NY	3,000,000

Return total boxoffice returns per city

per movie

aggregate

grouping attribute

$$\gamma_{City, Sum(Boxoffice)}(Movies)$$

Movie	City	Boxoffice
MI-IV	LA	2,000,000
MI-IV	NY	3,000,000
Don-II	LA	500,000

Movie	City	Boxoffice
MI-IV	LA	2,000,000
Don-II	LA	500,000
MI-IV	NY	3,000,000

City	Boxoffice
LA	2,500,000
NY	3,000,000

SQL

SQL – basic structure

```
SELECT L
FROM R
WHERE C
```

Attributes of the output relation

List of all relations involved

Conditions to be satisfied

Selection (1/2)

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return all actors living in Mumbai

$$\sigma_{Addr='Mumbai'}(Actors)$$

```
SELECT Name, Age, Addr
FROM Actors
WHERE Addr = 'Mumbai'
```

```
SELECT *
FROM Actors
WHERE Addr = 'Mumbai'
```

Selection (2/2)

Return all actors whose age is more than 35.

$$\sigma_{Age>35}(Actors)$$

```
SELECT *
FROM Actors
WHERE Age > 35
```

Return all actors whose age is more than 35 and who live in Mumbai

$$\sigma_{Age>35 \text{ and } Addr='Mumbai}(Actors)$$

```
SELECT *
FROM Actors
WHERE Age > 35 AND
      Addr = 'Mumbai'
```

Projection

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return the name and age of all actors

$$\Pi_{Name,Age}(Actors)$$

```
SELECT Name, Age
FROM Actors
```

Return the addresses of the actors

$$\Pi_{Addr}(Actors)$$

```
SELECT Addr
FROM Actors
```

```
SELECT DISTINCT (Addr)
FROM Actors
```

δ

Addr
Mumbai
LA

Equi-Joins

Actors

Name	Age	Addr
PC	38	Mumbai
AH	81	LA
BN	69	LA
AB	45	Mumbai

Movies

Name	Year	Title
PC	2011	Don-II
AH	2011	Thor: R
BN	2009	Valkyrie
AB	2010	Raavan

Return all information about actors and their movies

$$Actors \bowtie_{A.Name=M.Name} Movies$$

```
SELECT *
FROM Actors, Movies
WHERE Actors.Name = Movies.Name
```

Left outer joins

Actors

Name	Age	Addr
PC	38	Mumbai
AH	81	LA
BN	69	LA
AB	45	Mumbai
Null	75	XYZ

doesn't have match

Movies

Name	Year	Title
PC	2011	Don-II
AH	2011	Thor: R
AB	2010	Raavan
Null	2025	ABC

Return all information about actors and their movies

$$Actors \bowtie_{A.Name=M.Name} Movies$$

ROJ, FOJ

```
SELECT *
FROM Actors LEFT OUTER JOIN Movies
ON (Actors.Name = Movies.Name)
```

What happens when you compare something with a null value? Or when you compare a null with a null?

Self Join

Return all grandparents and their grand children

Actors

Name	Age	Addr	Parent
PC	38	Mumbai	Madhu
AH	81	LA	Muriel
BN	69	LA	Catherine
AB	45	Mumbai	Jaya
Jaya	63	Mumbai	Indira

Actors_1

Name	Age	Addr	Parent
PC	38	Mumbai	Madhu
AH	81	LA	Muriel
BN	69	LA	Catherine
AB	45	Mumbai	Jaya
Jaya	63	Mumbai	Indira

```
SELECT *
FROM Actors AS Actors1, Actors AS Actors2
WHERE Actors1.Parent = Actors2.Name
```

ρ rename

Composition of operators (1/2)

Actors

Name	Age	Addr
Priyanka Chopra	38	Mumbai
Anthony Hopkins	81	LA
Bill Nighy	69	LA
Abhishek Bachchan	45	Mumbai

Return the names and addresses of actors over 42

$$\Pi_{Name,Addr}\left(\sigma_{Age>42}(Actors)\right)$$

```
SELECT Name, Addr
FROM Actors
WHERE Age > 42
```

Return the names of actors over 42 who live in Mumbai

$$\Pi_{Name}\left(\sigma_{Age>42 \text{ AND } Addr='Mumbai'}(Actors)\right)$$

```
SELECT Name
FROM Actors
WHERE Age > 42
AND   Addr = 'Mumbai'
```

Composition of operators (2/2)

Return the names of actors below the age of 50 who have acted in a movie in 2011

$$\Pi_{Name}(\sigma_{Age<50 \text{ AND } Year=2011}(Actors \bowtie_{A.Name=M.Name} Movies))$$

$$\begin{aligned} Allmovies &= Actors \bowtie_{A.Name=M.Name} Movies \\ Movies1 &= \sigma_{Age<50 \text{ AND } Year=2011}(AllMovies) \\ Result &= \Pi_{Name}(Movies1) \end{aligned}$$

```
SELECT Actors.Name
FROM Actors, Movies
WHERE Age < 50
AND   Year = 2011
AND   Actors.Name = Movies.Name
```

Movies.Name

SQL – AGGREGATION, GROUPING AND SUBQUERIES

Aggregation and grouping

Movies

Movie	City	Boxoffice
Thor: R	LA	2,000,000
Don-II	LA	500,000
Thor: R	NY	3,000,000

Return total boxoffice returns per movie

Agg.

grouping attr

$$\gamma_{Movie, Sum(Boxoffice)}(Movies)$$

```
SELECT Movie, SUM (Boxoffice)
FROM Movies
GROUP BY Movie
```

Return movies for which total boxoffice returns were greater than 1,000,000

```
SELECT Movie, SUM (Boxoffice)
FROM Movies
GROUP BY Movie
HAVING SUM (Boxoffice) > 1000000
```

Movie	City	Boxoffice
Thor: R	LA	2,000,000
Don-II	LA	500,000
Thor: R	NY	3,000,000

order asc.

Sorting

- Sorting tuples by column

```
SELECT *
FROM Movies
ORDER BY City
```

```
SELECT Movie, SUM (Boxoffice)
FROM Movies
GROUP BY Movie
HAVING SUM (Boxoffice) > 1000000
ORDER BY Movie
```

```
SELECT Movie, Boxoffice
FROM Movies
ORDER BY Boxoffice DESC
```

```
SELECT Movie, SUM (Boxoffice) AS BO
FROM Movies
GROUP BY Movie
HAVING SUM (Boxoffice) > 1000000
ORDER BY BO DESC
```

Subqueries (1/2)

- Temporary relations, constants

```
SELECT L
FROM R1, R2, …, Rn
WHERE C
```

compute as needed temporary relation

Replace any of these relations with another query

```
SELECT L
FROM R
WHERE C1 op v1
```

need not be a specified const.

Replace the value with a query which returns a constant

Replace the value with a query which returns a relation

1 col.

row →

5

'Mumbai'

Subqueries (2/2)

Select * from Movies → Movies

```
SELECT *
FROM (SELECT *
      FROM MOVIES) AS DUMB
```

Movies

Actors

Name	Age	Addr
PC	38	Mumbai
AH	81	LA
BN	69	LA
AB	45	Mumbai

```
SELECT *
FROM Actors
WHERE Age <
      (SELECT AVG(Age)
       FROM Actors)
```

No grouping Attr = entire relation in a single group

↳ op V1

Conditions involving relations (1/2)

```
SELECT L
FROM R
WHERE C1 op v1
```

Replace the value with a query which returns a constant

Replace the value with a query which returns a relation

```
SELECT *
FROM Actors
WHERE Name IN
      (SELECT Name
       FROM Movies)
```

- C1 IN R, C1 NOT IN R
- C1 ≥ ALL R, C1 op ANY R
- C1 > ANY R, C1 op ANY R

Conditions involving relations (2/2)

```
SELECT *
FROM (SELECT *
      FROM MOVIES) AS DUMB
```

```
SELECT *
FROM Actors
WHERE Age < ALL
      (SELECT Age
       FROM Actors)
```

What is the output!

Actors

Name	Age	Addr
PC	38	Mumbai
AH	81	LA
BN	69	LA
AB	45	Mumbai

```
SELECT *
FROM Actors
WHERE Age <
      (SELECT AVG(Age)
       FROM Actors)
```

```
SELECT *
FROM Actors
WHERE Age < ANY
      (SELECT Age
       FROM Actors)
```

Reading and Practical HW

- Correlated subqueries
- Union, intersection, difference operations

UNION

SQL1

→ UNION → What is the semantics

SQL2

set or bag?