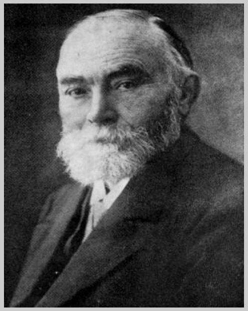
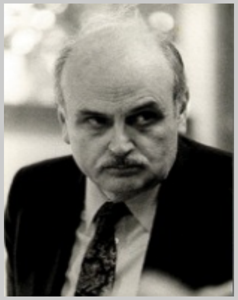
relational algebra & calculus

Relational DB: The Origins



Frege: FO logic

Tarski: Algebra for FO



Codd: Relational databases

relational calculus

Relational Calculus (aka FO)

- Models data manipulation core of SQL
  Idea: specify "what" not "how"
- General form:
  {t | property (t)}
- property (t) is described by a language based on predicate calculus (first-order logic)

Relational Calculus Example

Display the movie table

In SQL

SELECT *
FROM Movie

In words
(making answer tuple explicit)

The answer consists of tuples m such that m is a tuple in Movie

Need to say

"tuple m is in relation R": $\mathbf{m} \in \mathbf{R}$

Relational Calculus Example

Find the directors and actors of currently playing movies

In SQL

SELECT m.Director, m.Actor
FROM movie m, schedule s
WHERE m.Title = s.Title

In words *(making answer tuple explicit)*

"The answer consists of tuples t s.t.
there exist tuples m in movie and s in schedule for which
t.Director = m.Director and t.Actor = m.Actor and m.Title = s.Title"

Need to say

"there exists a tuple x in relation R": $\exists\, x \in R$
Refer to the value of attribute A of tuple x: **x(A)**
Boolean combinations

Relational Calculus Example

Find the directors and actors of currently playing movies

Need to say

"there exists a tuple x in relation R": $\exists\, x \in R$
Refer to the value of attribute A of tuple x: $x(A)$
Boolean combinations

In logic notation (tuple relational calculus)

$\{ t\text{: Director, Actor} \mid \exists\, m \in movie\ \exists\, s \in schedule$
$[\ t(Director) = m(Director) \wedge t(Actor) = m(Actor)$
$\wedge\ m(Title) = s(Title)\ ]\ \}$

Quantifiers

$\exists\ m \in R$: Existential quantification
"there exists some tuple m in relation R …."

Sometimes need to say:
"for every tuple m …."

e.g., "every director is also an actor"

Need to say:
"for every tuple m in movie there exists a tuple t in movie
Such that m.Director = t.Actor"

$\forall\ m \in \text{movie}\ \exists\ t \in \text{movie}\ [\ m(\text{Director}) = t(\text{Actor})\ ]$

(The answer to this query is true or false)

$\forall\ m \in R$: Universal quantification
"for every tuple m in relation R …."

Tuple Relational Calculus

- In the style of SQL: language talks about tuples
- What you can say:
  - Refer to tuples: tuple variables t, s, …
  - A tuple t belongs to a relation R: $t \in R$
  - Conditions on attributes of a tuple t and s:
    - t(A) = (≠)(≥) constant
    - t(A) = s(B)
    - t(A) ≠ s(B)
    - etc.
- Simple expressions above: atoms

Tuple Relational Calculus

- Combine properties using Boolean operators

  $\wedge$, $\vee$, $\neg$

  (abbreviation: $p \rightarrow q \equiv \neg p \vee q$)

- Quantifiers

  there exists: $\exists t \in R\ \varphi(t)$

  for every: $\forall t \in R\ \varphi(t)$

  where $\varphi(t)$ a formula in which t not quantified (it is “free”)

More on quantifiers

- Scope of quantifier:
  scope of $\exists t \in R\ \varphi(t)$ is φ
  scope of $\forall t \in R\ \varphi(t)$ is φ
- Free variable:
  not in scope of any quantifier
  free variables are the "parameters" of the formula
- Rule: in quantification $\exists t \in R\ \varphi(t)$, $\forall t \in R\ \varphi(t)$
  t must be free in φ

Quantifier Examples

{ t: Director, Actor | ∃ m ∈ movie ∃ s ∈ schedule
[t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title)] }

[t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title)]

free: t, m, s

∃ s ∈ schedule

[t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title)]

free: t, m

∃ m ∈ movie ∃ s ∈ schedule

[t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title)]

free: t

Example in predicate logic

A statement about numbers:

$\exists x \forall y \forall z [x = y * z \longrightarrow ((y = 1) \vee (z = 1))]$

"there exists at least one prime number x"

A "query" on numbers:

$\varphi(x)$: $\forall y \forall z [x = y * z \longrightarrow ((y = 1) \vee (z = 1))]$

This defines the set $\{x \mid \varphi(x)\}$ of prime numbers.
It consists of all x that make $\varphi(x)$ true.

Semantics of Tuple Calculus

- Active domain:
  A set of values in the database, or mentioned in the query result. Tuple variables range over the active domain

- Note:
  A query without free variables always evaluates to true or false

  e.g., "Sky is by Berto" is expressed without free variables:

  $\exists m \in$ movie [m(title) = "Sky" $\wedge$ m(director) = "Berto"]

  This statement is true or false

Tuple Calculus Query

{t: <att> | φ(t)}
where φ is a calculus formula with only one free variable t

produces as answer a table with attributes <att> consisting of all tuples v in active domain with make φ(v) true

Note:

φ(v) has no free variables so it evaluates to true or false

Movie Examples Revisited

Find titles of currently playing movies

```
select Title
from Schedule
```

Find the titles of all movies by “Berto”

```
select Title
from Movie
where Director=“Berto”
```

Find the titles and the directors of all currently playing movies

```
select Movie.Title, Director
from Movie, Schedule
where Movie.Title = Schedule.Title
```

Movie Examples Revisited

Find titles of currently playing movies

{t: title l ∃s ∈schedule [s(title) = t(title)]}

Find the titles of all movies by “Berto”

{t: titlel ∃m ∈ movie [m(director) = “Berto” ∧ t(title) = m(title)]}

Find the titles and the directors of all currently playing movies

{t: title, director l ∃s ∈schedule ∃m ∈ movie
[s(title) = m(title) ∧ t(title) = m(title) ∧ t(director) = m(director)]}

Movie Examples Revisited

- Find actors playing in every movie by Berto

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
∀m ∈ movie [m(director) = "Berto" → ∃t ∈ movie (m(title) =
t(title) ∧ t(actor) = y(actor))]]}

Is the following correct?

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
∀m ∈ movie [m(director) = "Berto" **∧** ∃t ∈ movie (m(title) =
t(title) ∧ t(actor) = y(actor))]]}

A: YES   B: ~~NO~~

Movie Examples Revisited

- Find actors playing in every movie by Berto

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
∀m ∈ movie [m(director) = “Berto” → ∃t ∈ movie (m(title) =
t(title) ∧ t(actor) = y(actor))]]}

Typical use of **∀**:

∀ m ∈ R [filter(**m**) → property(**m**)]

Intuition: check property(**m**) for those **m** that satisfy filter(**m**)
we don’t care about the **m**’s that do not satisfy filter(**m**)

Movie Examples Revisited

- Find actors playing in every movie by Berto

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
∀m ∈ movie [m(director) = “Berto” → ∃t ∈ movie (m(title) =
t(title) ∧ t(actor) = y(actor))]]}

Is this correct?

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
∀m ∈ movie **∃t ∈ movie** [m(director) = “Berto” → (m(title) =
t(title) ∧ t(actor) = y(actor))]]}

A: YES B: NO

Movie Examples Revisited

Is this correct?

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
∀m ∈ movie **∃t ∈ movie** [m(director) = "Berto" → (m(title) =
t(title) ∧ t(actor) = y(actor))]]}

A: ~~YES~~ B: NO

$\exists \mathbf{t}\ (\varphi \vee \psi) = \exists \mathbf{t}\ \varphi \vee \exists \mathbf{t}\ \psi$

$\exists \mathbf{t}\ \varphi = \varphi$ if **t** does not occur in φ

Is the following correct:

$\exists \mathbf{t}\ (\varphi \wedge \psi) = \exists \mathbf{t}\ \varphi \wedge \exists \mathbf{t}\ \psi$

A: YES B: NO

Movie Examples Revisited

Correct:

$\{a: actor \mid \exists y \in movie\ [a(actor) = y(actor) \land$
$\forall m \in movie\ \mathbf{\exists t \in movie}\ [m(director) = \text{"Berto"} \rightarrow (m(title) =$
$t(title) \land t(actor) = y(actor))]]\}$

$\mathbf{\exists t}\ (\varphi \lor \psi) = \mathbf{\exists t}\ \varphi \lor \mathbf{\exists t}\ \psi$
$\mathbf{\exists t}\ \varphi = \varphi$ if t does not occur in φ

$\mathbf{\exists t \in movie}\ [m(director) = \text{"Berto"} \rightarrow (m(title) =$
$t(title) \land t(actor) = y(actor))] =$
$\mathbf{\exists t \in movie}\ [\neg m(director) = \text{"Berto"} \lor (m(title) =$
$t(title) \land t(actor) = y(actor))] =$
$[\mathbf{\exists t \in movie}\ \mathbf{(}\neg m(director) = \text{"Berto"}\mathbf{)} \lor \mathbf{\exists t \in movie}\ (m(title) =$
$t(title) \land t(actor) = y(actor))] =$
$[\neg m(director) = \text{"Berto"} \lor \mathbf{\exists t \in movie}\ (m(title) =$
$t(title) \land t(actor) = y(actor))] =$
$[m(director) = \text{"Berto"} \rightarrow \mathbf{\exists t \in movie}\ (m(title) =$
$t(title) \land t(actor) = y(actor))]$

Movie Examples Revisited

Correct:

$$\{a: actor \mid \exists y \in movie\ [a(actor) = y(actor)\ \wedge\ \forall m \in movie\ \mathbf{\exists t \in movie}\ [m(director) = \text{“Berto”} \rightarrow (m(title) = t(title) \wedge t(actor) = y(actor))]]\}$$

Is this also correct (can we switch ∀ and ∃)?

$$\{a: actor \mid \exists y \in movie\ [a(actor) = y(actor)\ \wedge\ \mathbf{\exists t \in movie}\ \forall m \in movie\ [m(director) = \text{“Berto”} \rightarrow (m(title) = t(title) \wedge t(actor) = y(actor))]]\}$$

A: YES B: ~~NO~~

Tuple Calculus and SQL

- Example:
  "Find theaters showing movies by Bertolucci":

  SQL:

  SELECT s.theater
  FROM schedule s, movie m
  WHERE s.title = m.title AND m.director = "Bertolucci"

  tuple calculus:

  { t: theater | ∃ s ∈ schedule ∃ m ∈ movie [t(theater) = s(theater) ∧ s(title) = m(title) ∧ m(director) = Bertolucci] }

Basic SQL Query

SQL

- **SELECT** $A_1, \ldots, A_n$
FROM $R_1, \ldots, R_k$
WHERE $cond(R_1, \ldots, R_k)$

Tuple Calculus

- $\{t: A_1, \ldots, A_n \mid \exists r_1 \in R_1 \ldots \exists r_k \in R_k [\wedge_j\, t(A_j) = r_{ij}(A_j) \wedge cond(r_1, \ldots, r_k)]\}$
- Note:
  - Basic SQL query uses only ∃
  - No explicit construct for ∀

Using Tuple Calculus to Formulate SQL Queries

Example: “Find actors playing in every movie by Berto”

- Tuple calculus
  {a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
  ∀m ∈ movie [m(dir) = “Berto” → ∃t ∈ movie (m(title) =
  t(title) ∧ t(actor) = y(actor))]]}
- Eliminate ∀:
  {a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
  ¬∃m ∈ movie [m(dir) = “Berto” ∧ ¬∃t ∈ movie (m(title) =
  t(title) ∧ t(actor) = y(actor))]]}
- Rule: $\forall x \in R\ \varphi(x) \equiv \neg\exists x \in R\ \neg\varphi(x)$

“every x in R satisfies φ(x) iff
there is no x in R that violates φ(x)”

Convert to SQL query

- Basic rule: one level of nesting for each "¬∃"

{a: actor | ∃y ∈ movie [a(actor) = y(actor) ∧
¬∃m ∈ movie [m(dir) = "Berto" ∧ ¬∃t ∈ movie (m(title) = t(title)
∧ t(actor) = y(actor))]]}

↓

```
SELECT y.actor  FROM movie y
WHERE NOT EXISTS
        (SELECT * FROM movie  m
        WHERE m.dir = 'Berto' AND
        NOT EXISTS
            (SELECT *
            FROM movie t
            WHERE m.title = t.title  AND t.actor = y.actor ))
```

Another possibility (with similar nesting structure)

```
SELECT actor  FROM movie
WHERE actor  NOT IN
    (SELECT s.actor
    FROM movie s, movie m
    WHERE  m.dir  = 'Berto'
    AND s.actor  NOT IN
       (SELECT t.actor
       FROM movie t
       WHERE m.title = t.title ))
```

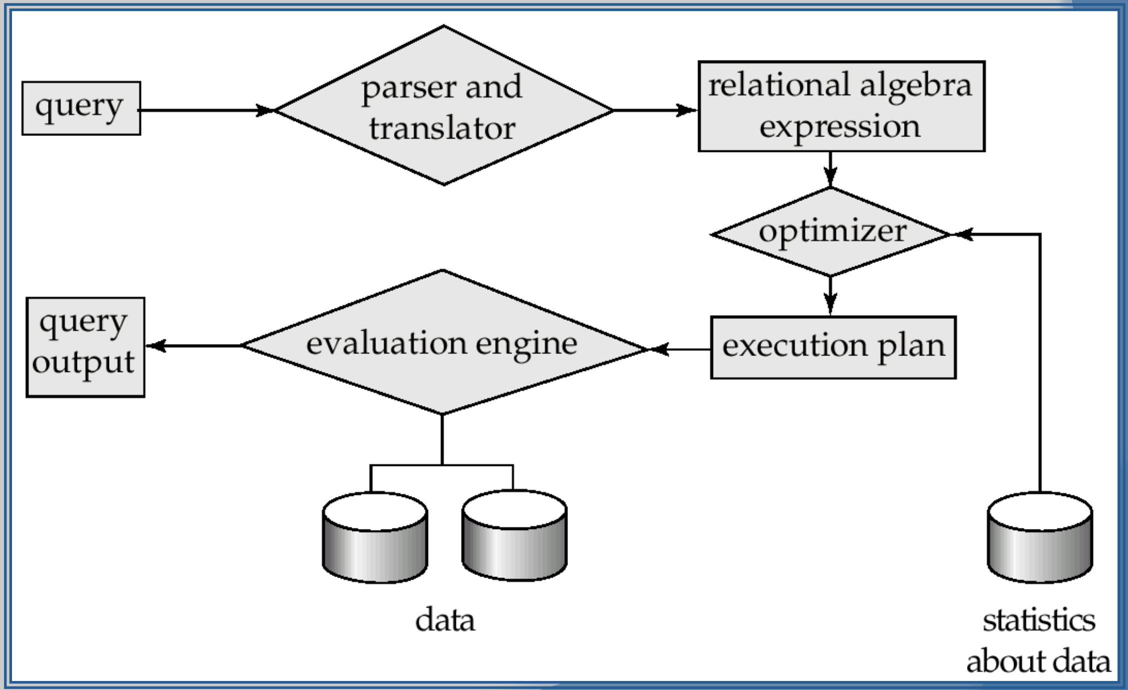
- Note: Calculus is more flexible than SQL because of the ability to mix ∃ and ∀ quantifiers

relational algebra

Query Processing

3 steps:

- Parsing & Translation
- Optimization
- Evaluation



Relational Algebra

- Simple set of algebraic operations on relations

Journey of a query

SQL	select … from…where
↓	↓
Relational algebra	$\pi_{13}(P \bowtie Q) \bowtie \ldots$
↓	↓
Query rewriting	$\pi_{14}(P \bowtie S) \bowtie Q \bowtie R$

- We use **set semantics** (no duplicates) and **no nulls**
- There are extensions with bag semantics and nulls

Relational Algebra
Projection

Eliminate some columns

$\pi_X(R)$	Display only attributes X of relation R
where R: table name & $X \subseteq$ attributes(R)	

Example:

Find titles of current movies

$\pi_{TITLE}(SCHEDULE)$

Relational Algebra
Projection

Eliminate some columns

$\pi_X(R)$	Display only attributes X of relation R
where R: table name & X ⊆ attributes(R)	

Example:

R

A	B	C
0	1	2
0	2	2
1	3	1
0	1	3

$\pi_A(R) =$

A
0
1

$\pi_{AB}(R) =$

A	B
0	1
0	2
1	3

No repetitions of tuples!

Relational Algebra

Selection

Compute set union

$\sigma_{cond}(R)$	Select tuples of R satisfying condition cond

where cond: condition involving only attributes of R
(e.g., attr = value, attr ≠ value, attr1 = attr2, attr1 ≠ attr2, etc.)

Example:

R

A	B	C
0	1	2
0	2	2
1	3	1
0	1	3

$\sigma_{A=0}(R) =$

A	B	C
0	1	2
0	2	2
0	1	3

$\sigma_{B=C}(R) =$

A	B	C
0	2	2

Relational Algebra

Selection

Compute set union

$\sigma_{cond}(R)$	Select tuples of R satisfying condition cond

where cond: condition involving only attributes of R
(e.g., attr = value, attr ≠ value, attr1 = attr2, attr1 ≠ attr2, etc.)

Example:

R

A	B	C
0	1	2
0	2	2
1	3	1
0	1	3

$\sigma_{A \neq 0}(R) =$

A	B	C
1	3	1

Union

Compute set union

$R \cup S$	Union of sets of tuples in R and S
where R, S: tables with same attributes	

Example:

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

$R \cup S$ =

A	B
α	1
α	2
β	1
β	3

Difference

Compute set difference

R - S	Difference of sets of tuples in R and S
where R, S: tables with same attributes	

Example:

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

R - S =

A	B
α	1
β	1

Relational Algebra
Join

Compute join

$R \bowtie S$	Natural Join of R, S
where R, S: tables	

Example:

R	A	B

S	B	C

$R \bowtie S$ =

A	B	C

Note: More than one common attributes allowed!

Relational Algebra
Join

Compute join

$R \bowtie S$	Natural Join of R, S
where R, S: tables	

Example:

R

A	B
0	1
0	2
5	3

S

B	C
1	2
1	3
2	2

$R \bowtie S$ =

A	B	C
0	1	2
0	1	3
0	2	2

Definition of Join

Let *r* and *s* be relations on schemas *R* and *S* respectively.
Then, $r \bowtie s$ is a relation with attributes att(*R*) ∪ att(*S*) obtained as follows:

Consider each pair of tuples t_r from *r* and t_s from *s*.

If t_r and t_s have the same value on each of the attributes in att(*R*) ∩ att(*S*),

add a tuple *t* to the result, where

- *t* has the same value as t_r on *r*
- *t* has the same value as t_s on *s*

Note: if *R ∩ S is empty, the join consists of all combinations of tuples from R and S, i.e. their cross-product*

Relational Algebra

Attribute Renaming

Rename attributes

$\delta_{A1 \rightarrow A2}(R)$	Change name of attribute A1 in rel. R to A2
where R: relation and A1: attribute in R	

Example:

R

A	B
α	1
α	2
β	1

$\delta_{A \rightarrow C}(R) =$

C	B
α	1
α	2
β	1

Contents remain unchanged!

Note: Can rename several attributes at once

Relational Algebra

- Basic set of operations:

  $$\pi, \sigma, \cup, -, \bowtie, \delta$$

- Back to movie example queries:

1. Titles of currently playing movies:

   $$\pi_{TITLE}(\text{schedule})$$

2. Titles of movies by Berto:

   $$\pi_{TITLE}(\sigma_{DIR=BERTO}(\text{movie}))$$

3. Titles and directors of currently playing movies:

   $$\pi_{TITLE,\ DIR}\ (\text{movie} \bowtie \text{schedule})$$

Relational Algebra

4. Find the pairs of actors acting together in some movie

$$\pi_{actor1,\ actor2} (\delta_{actor \rightarrow actor1} (movie) \bowtie \delta_{actor \rightarrow actor2} (movie))$$

5. Find the actors playing in every movie by Berto

$$\pi_{actor} (movie) -$$
$$\pi_{actor} [(\pi_{actor} (movie) \bowtie \pi_{title} (\sigma_{dir\ =BERTO}(movie))) - \pi_{actor,title} (movie)]$$

actor

title by Berto

actor acts in title

actors for which there is a movie by Berto in which they do not act

In this case (not in general): Same as cartesian product

Relational Algebra

Cartesian Product

Compute cartesian product

R×S	Cartesian Product of R, S
where R, S: tables	

Example:

R

A	B
0	1
0	2

S

C	D
1	2
1	3

R⋈S =

A	B	C	D
0	1	1	2
0	1	1	3
0	2	1	2
0	2	1	3

Same as R⋈S, when R and S have no common attributes

Relational Algebra

Cartesian Product

Compute cartesian product

R×S	Cartesian Product of R, S
where R, S: tables	

Example:

R

A	B
0	1
0	2

S

A	C
1	2
1	3

R⋈S =

R.A	B	S.A	C
0	1	1	2
0	1	1	3
0	2	1	2
0	2	1	3

If 2 attributes in R, S have the same name A, they are renamed to R.A and S.A in the output

Other useful operations

- Intersection R ∩ S
- Division (Quotient) R ÷ S

R	A	B

S	B

R ÷ S: {a | <a, b> ∈R for <u>every</u> b∈S}

Example:

R	A	B
	0	α
	0	β
	1	α
	1	β
	1	γ
	2	α

S	B
	α
	β

R ÷ S =

A
0
1

Another Division Example

- Find the actors playing in every movie by Berto

$$\pi_{TITLE,\ ACTOR}(movie) \div \pi_{TITLE}(\sigma_{DIR=BERTO}(movie))$$

Division by multiple attributes

- Relations *r, s*:

r

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

s

D	*E*
a	*1*
b	*1*

- $r \div s$:

A	*B*	*C*
α	a	γ
γ	a	γ

Relational Algebra

- Note:
  π is like $\exists$ "there exists"…
  $\div$ is like $\forall$ "for all"…
- Expressing $\div$ using other operators:

$$R \div S = \pi_A(R) - \pi_A((\pi_A(R) \bowtie S) - R)$$

R	A	B

S	B

Similar to: $\forall x\ \varphi(x) \equiv \neg \exists x\ \neg \varphi(x)$

Calculus Vs. Algebra

- Theorem: Calculus and Algebra are equivalent
- Basic Correspondence:

Algebra Operation		Calculus Operation
π	↔	∃
σ	↔	t(A) comp c
∪	↔	∨
⋈	↔	∧
-	↔	¬
÷	↔	∀

Example

- "Find theaters showing movies by Bertolucci":

  SQL:

  - **SELECT** s.theater
    FROM schedule s, movie m
    WHERE s.title = m.title **AND** m.director = 'Berto'

  tuple calculus:

  - { t: theater | $\exists$ s $\in$ schedule $\exists$ m $\in$ movie [t(theater) = s(theater) $\wedge$ s(title) = m(title) $\wedge$ m(director) = Berto] }

  relational algebra:

$$\pi_{theater}(\text{schedule} \bowtie \sigma_{dir\ =\ Berto}(\text{movie}))$$

Note: number of items in FROM clause = (number of joins + 1)