CS 133: Databases

Fall 2019
Lec 7 – 09/26
Relational Algebra

Prof. Beth Trushkowsky

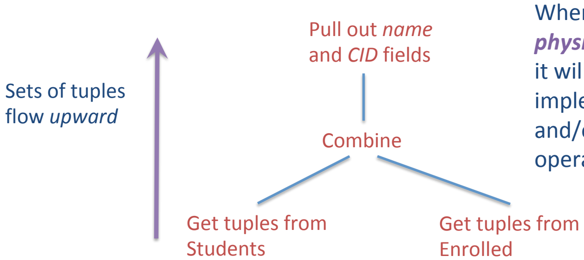
Goals for Today

- Learn about how relational algebra operates on sets of tuples
- Compose the basic relational algebra operators to form *queries* on relations
- Understand the goals for Lab 2
  – You're ready for Exercise 1 after class today!

Logical Query Plan Example

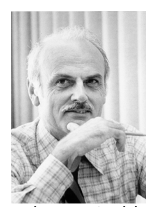
- Example: college database
  Students(SID, name, gpa)
  Enrolled(SID, CID, grade)

```
SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.SID=E.SID;
```



When query optimizer forms ***physical query plan***, it will consider available implementations and/or choices for the logical operators!

Operations on Sets of Tuples



Edgar F. Codd
Turing award, 1981

- **Relational model:** data represented as sets of tuples (i.e., *relations*)
- **Relational algebra**: an *algebra* on sets of tuples
  – Used to express *queries* about those relations
  – I.e., a **query language**

- **Note: Sets != Bags**
  – Sets: relations have no duplicate tuples
  – Bags, aka multi-sets: duplicate tuples possible

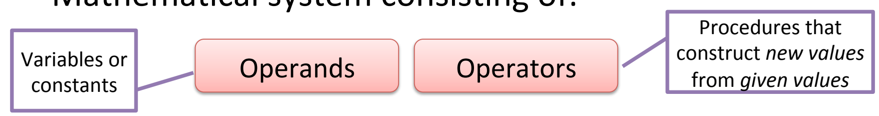
Formal Relational Query Languages

- *Query languages* allow manipulation and retrieval of data from a database
  - Query languages != programming languages!
- Two mathematical Query Languages form the basis for "real" languages (e.g., SQL), and for implementation:
  - *Relational Algebra*: More operational, useful for representing query execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

We'll see some differences between SQL and relational algebra

What is "an Algebra" ??

Mathematical system consisting of:

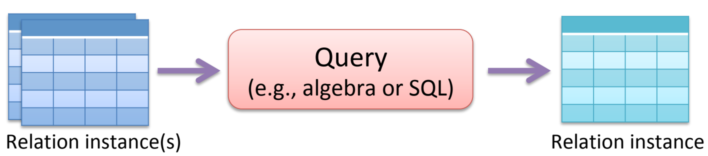


Example: Arithmetic	*x, y, 15*	*+, -, *, /*
Relational	Relations	*Let's see…*

An algebra allows us to **build expressions** by applying operators to operands and/or other expressions

Preliminaries

A query is applied to *relation instances*, and the result of a query is **also** a relation instance.



Depending on the query, output relation schema may be the same or different than input schema

SID	name	login	gpa
45	Alice	alicious	3.4
67	Bob	bobtastic	3.9
78	Carl	carl	2.5

→ Query →

name
Bob

Example Instances

Sailing Database:
Boats, Sailors, Reserves

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra: 5 Basic Operations

- *Selection* (σ) Selects a subset of *rows* from relation (horizontal).
- *Projection* (π) Retains only wanted *columns* from relation (vertical).
- *Cross-product* (x) Allows us to combine two relations.
- *Set-difference* (–) Tuples in relation1, but not in relation2.
- *Union* ($\cup$) Tuples in relation1 and/or in relation2.

Since each operation returns a relation,
operations can be *composed*!

Selection (σ) – Horizontal Restriction

- Selects rows that satisfy *selection condition*.
  - Note: **not** the same thing as SELECT in SQL
  - Can have several conditions, combined with ∨ (or), ∧ (and)
- ***Schema*** of result is same as that of the input relation.
- Example:

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$(S2)$

sid	sname	rating	age
31	lubber	8	55.5

$\sigma_{rating=8}(S2)$

Exercise 2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

Projection (π) – Vertical Restriction

- Examples: $\pi_{age}(S2)$ $\pi_{sname,rating}(S2)$
- Retains only attributes that are in the "***projection list***".
- *Schema* of result:
  - exactly the fields in the projection list, with the same names that they had in the input relation
- In relational algebra, projection operator ***eliminates duplicates***
  - How would duplicates arise?
  - *Note*: real systems typically don't do duplicate elimination in SQL unless the user explicitly asks for it (**why not?**)

Examples: Projection

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

age
35.0
55.5

$\pi_{age}(S2)$

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

Composing Operators

- Output of a Relational Algebra operator is a relation, so…
  - Can use result as input to another Relational Algebra operator

sid	sname	rating	age
28	yuppy	9	35.0
~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~44~~	~~guppy~~	~~5~~	~~35.0~~
58	rusty	10	35.0

S2

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Union and Set-Difference

- Take two input relations, which must be *union-compatible*:
  - Same number of fields
  - "Corresponding" fields have the same type

Will the Union operator have to do duplicate elimination? How about Set-Difference?

Union

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

Set-difference

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

$S2 - S1$

Cross-Product

- S1 x R1: Each tuple of S1 paired with each tuple of R1

  How many tuples will be in the result S1 x R1?

- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *May have a naming conflict*: When both S1 and R1 have a field with the same name.
  - In this case, can use the *renaming operator*:

    $\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

Rename the first and fifth fields in the relation that results from S1 x R1

Cross Product Example

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1) =$

sid1	sname	rating	age	sid2	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Compound Operators

- In addition to the five basic operators, there are several additional "Compound Operators"
  - These **add no computational power** to the language, but are useful short-hands
  - Can be expressed solely with the basic operators

- We'll look at
  - Intersection, Join
- See book for
  - Division

Intersection

Intersection takes two input relations, which must be *union-compatible*.

- *How to express it using only basic operators?*

  R ∩ S = R - (R - S)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Join (⋈)

- Joins are compound operators involving
  - **cross product**,
  - **selection**,
  - and (sometimes) **projection**.
- Most common type of join is a *natural join* (bowtie with no annotation)

R⋈S conceptually is:

- Compute the **cross product** R X S
- **Select** rows where attributes that appear in both relations have equal values
- **Project** all unique attributes and *one copy* of each of the common ones

Natural Join Example

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

R1⋈S1 =

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

How would join result change if R1 also contained the tuple:

sid	bid	day
22	105	12/13/96

Other Types of Joins

- *Condition Join (or "theta-join")*:

$$R \bowtie_c S = \sigma_c (R \times S)$$

  - *Result schema* same as that of cross-product.
  - (May have fewer tuples than cross-product)

- *Equi-Join*: Nickname for case when condition *c* contains only conjunction of *equalities*.

"Theta" Join Example

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S3

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \bowtie_{S1.rating<S3.rating} S3 =$$

not shown: renaming columns

sid1	sname1	rating1	age1	sid2	sname2	rating2	age2
22	dustin	7	45.0	31	lubber	8	55.5
22	dustin	7	45.0	58	rusty	10	35.0
31	lubber	8	55.5	58	rusty	10	35.0

Exercise 5-7: Relational Algebra using Joins

Find names of sailors who've reserved boat #103

$$\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$$

Find names of sailors who've reserved a red boat

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

Find sailors who've reserved a red and a green boat

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

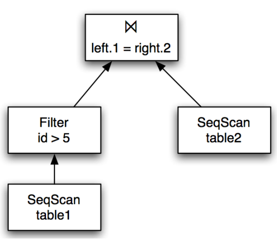
$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Lab 2: SimpleDb Operators

- *Goal*: building on Lab 1, be able to perform simple queries over **multiple relations**

```
SELECT *
FROM table1, table2
WHERE table1.field1 = table2.field2
   AND table1.id > 5;
```



- Three parts! *Submit **each** on Gradescope*
  - Part 1: Filter and Join: *out now, due next Wednesday*
  - Part 2: Aggregation, HeapFile Mutability
  - Final: Insert and Delete, Buffer Pool Eviction
- Operators will implement the interface ***DbIterator***

Operators are DbIterators

```
public abstract class Operator implements DbIterator {

    public boolean hasNext() throws DbException, TransactionAborted

    public Tuple next() throws DbException, TransactionAbortedExcep
        if (next == null) {
            next = fetchNext();
            if (next == null) throw new NoSuchElementException();
        }
        Tuple result = next;
        next = null;
        return result;
    }

    protected abstract Tuple fetchNext()
```

`hasNext()` and `next()` are already written

Calls `fetchNext()`

Your operators will have to implement this

Example: Filter

```java
/**
 * Filter is an operator that implements a relational select.
 */
public class Filter extends Operator {
  /**
   * Constructor accepts a predicate to apply
   * and a child operator to read tuples to filter from.
   *
   * @param p
   *           The predicate to filter tuples with
   * @param child
   *           The child operator
   */
  public Filter(Predicate p, DbIterator child) {
```

Each tuple will be filtered by `p.filter()`
Hint: *check out* `Field.compare()`

A child operator from which to read tuples!

You will need to call `super.open()` and `super.close()`

See `Project.java` *as an example*

Join

Takes a `JoinPredicate`:

```java
public JoinPredicate(int field1, Predicate.Op op, int field2) {
```

Which field from the tuple from **child1**

Which field from the tuple from **child2**

Recall nested-loop algorithm…

Relational Calculus

- Tuple Relational Calculus:
  - Variables range over (i.e., get bound to) tuples
  - **Answer tuples**: an assignment of constants to variables that **make an expression evaluate to true**

$$\{S \mid S \in Sailors \wedge S.rating > 7\}$$

$$\{P \mid \exists S \in Sailors(S.rating > 7 \wedge P.name = S.sname \wedge P.age = S.age)\}$$

Effectively the projected attributes

- Every relational algebra query can be expressed as a *safe* calculus query, and vice versa

Check out Section 4.3 in the book for more!

Logical Query Plan Example

- Example: college database
  Students(<u>SID</u>, name, gpa)
  Enrolled(<u>SID, CID</u>, grade)

```sql
SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.SID=E.SID;
```

Relational algebra expression?