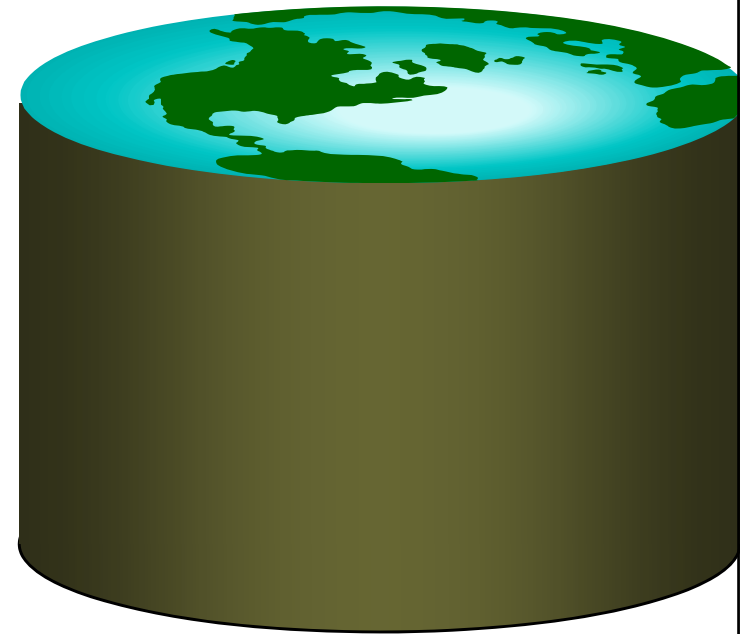
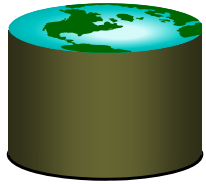


Database Management System

**Sumayyea Salahuddin (Lecturer)
Dept. of Computer Systems Eng.
UET Peshawar**





Objectives

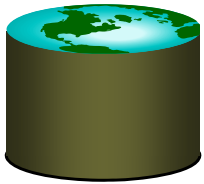
- **Relational Algebra**
- **Relational Calculus**

Relational Algebra

By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and, in effect, *increases the mental power of the race.*

-- Alfred North Whitehead (1861 - 1947)





Formal Relational Query Languages

Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:

Relational Algebra: More **operational**, very useful for representing execution plans.

Relational Calculus: Lets users describe what they want, rather than how to compute it. (even **more declarative**.)

□ *Understanding Algebra & Calculus is key to understanding SQL, query processing!*



Example: Joining Two Tables

Enrolled

cid	grade	sid
Carnatic101	C	53666
Reggae203	B	53666
Topology112	A	53650
History105	B	53666

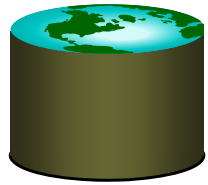
Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- **Pre-Relational:**
write a program
- **Relational SQL**
Select name, cid from students s, enrolled e where s.sid = e.sid
- **Relational Algebra**

$$\pi_{name, cid} (Students \bowtie_{sid} Enrolled)$$

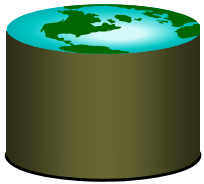
- **Relational Calculus**
 $\{S \mid S \in Students \wedge \exists E (E \in Enrolled \wedge E.sid = S.sid)\}$



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 (\times) Allows us to combine two relations.
- Set-difference ($-$) Tuples in r1, but not in r2.
- Union (\cup) Tuples in r1 and/or in r2.

Since each operation returns a relation, **operations can be composed!** (Algebra is “closed”.)



Example Instances

R1

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

S1

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

S2

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

<u>bid</u>	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Boats



Projection

- 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.
- Projection operator has to *eliminate duplicates* (How do they arise? Why remove them?)
 - Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it. (Why not?)



Projection

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

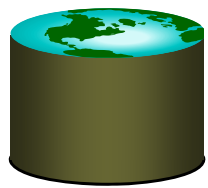
S2

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

age
35.0
55.5

$\pi_{age}(S2)$



Selection (σ)

- Selects rows that satisfy *selection condition*.
- Result is a relation.

Schema of result is same as that of the input relation.

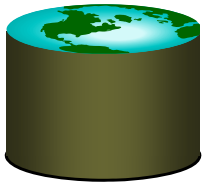
- Do we need to do duplicate elimination?

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

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

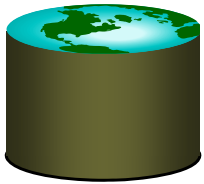
sname	rating
yuppy	9
rusty	10

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



Union and Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
 - Same number of fields.
 - `Corresponding' fields have the same type.
- For which, if any, is duplicate elimination required?



Union

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

S1

<u>sid</u>	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

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

S1

<u>sid</u>	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

$S1 - S2$

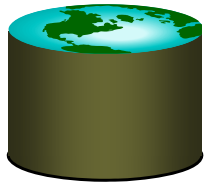
<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

$S2 - S1$



Cross-Product

- $S1 \times R1$: Each row of S1 paired with each row of R1.
- Q: How many rows in the result?
- *Result schema* has one field per field of S1 and R1, with field names 'inherited' if possible.
 - *May have a naming conflict*: Both S1 and R1 have a field with the same name.
 - In this case, can use the *renaming operator*:
$$\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$$



Cross Product Example

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

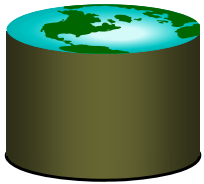
R1

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

S1

R1 X S1 =

(sid)	sname	rating	age	(sid)	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 Operator: Intersection

- In addition to the 5 basic operators, there are several additional “Compound Operators”
 - These add no computational power to the language, but are useful shorthands.
 - Can be expressed solely with the basic ops.
- Intersection takes two input relations, which must be **union-compatible**.
- Q: How to express it using basic operators?

$$R \cap S = R - (R - S)$$



Intersection

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

S1

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

S2

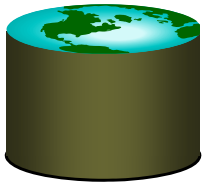
sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \cap S2$$



Compound Operator: Join

- Joins are compound operators involving cross product, selection, and (sometimes) projection.
- Most common type of join is a "natural join" (often just called "join"). $R \bowtie S$ conceptually is:
 - Compute $R \times S$
 - 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.
- Note: Usually done much more efficiently than this.
- Useful for putting "normalized" relations back together.



Natural Join Example

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

R1

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

S1

$R1 \bowtie S1 =$

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



Other Types of Joins

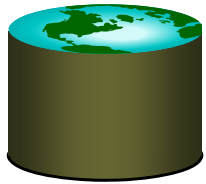
- **Condition Join (or "theta-join"):**

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

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- **Result schema** same as that of cross-product.
- May have fewer tuples than cross-product.
- **Equi-Join:** **Special case: condition c contains only conjunction of equalities.**

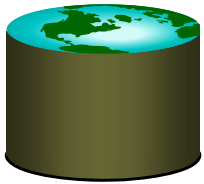


Compound Operator: Division

- Useful for expressing “for all” queries like:
Find sids of sailors who have reserved all boats.
- For A/B attributes of B are subset of attrs of A .
 - May need to “project” to make this happen.
- E.g., let A have 2 fields, x and y ; B have only field y :

$$A/B = \{ \langle x \rangle \mid \forall \langle y \rangle \in B (\exists \langle x, y \rangle \in A) \}$$

A/B contains all tuples (x) such that for every y tuple in B , there is an xy tuple in A .



Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

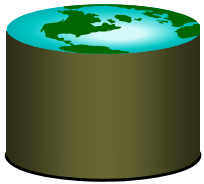
A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

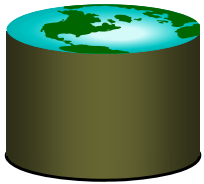


Expressing A/B Using Basic Operators

- **Division is not essential op; just a useful shorthand.**
 - (Also true of joins, but joins are so common that systems implement joins specially.)
- **Idea:** For A/B , compute all x values that are not 'disqualified' by some y value in B .
 - x value is *disqualified* if by attaching y value from B , we obtain an xy tuple that is not in A .

Disqualified x values: $\pi_x ((\pi_x(A) \times B) - A)$

A/B : $\pi_x(A) - \text{Disqualified } x \text{ values}$



Examples

Reserves

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

Sailors

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

Boats

<u>bid</u>	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red



E1: Find names of sailors who've reserved boat #103

- **Solution 1:** $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$
- **Solution 2:** $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$



E2: Find names of sailors who've reserved a red boat

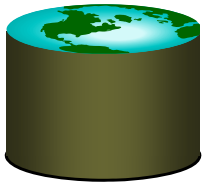
- **Information about boat color only available in Boats; so need an extra join:**

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

- A more efficient solution:

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

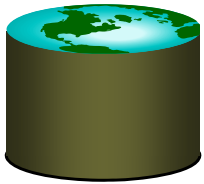
- *A query optimizer can find this given the first solution!*



E3: Find sailors who've reserved a red or a green boat

- **Can identify all red or green boats, then find sailors who've reserved one of these boats:**

$$\rho \text{ (Tempboats, } (\sigma_{color='red' \vee color='green'} \text{Boats}))$$
$$\pi_{sname}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})$$



E4: Find sailors who've reserved a red and a green boat

- **Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):**

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

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

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



E5: Find the names of sailors who've reserved all boats

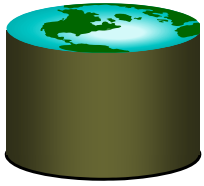
- **Uses division; schemas of the input relations to / must be carefully chosen:**

$$\rho (Tempsids, (\pi_{sid,bid} Reserves) / (\pi_{bid} Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$..... / \pi_{bid} (\sigma_{bname='Interlake'} Boats)$$



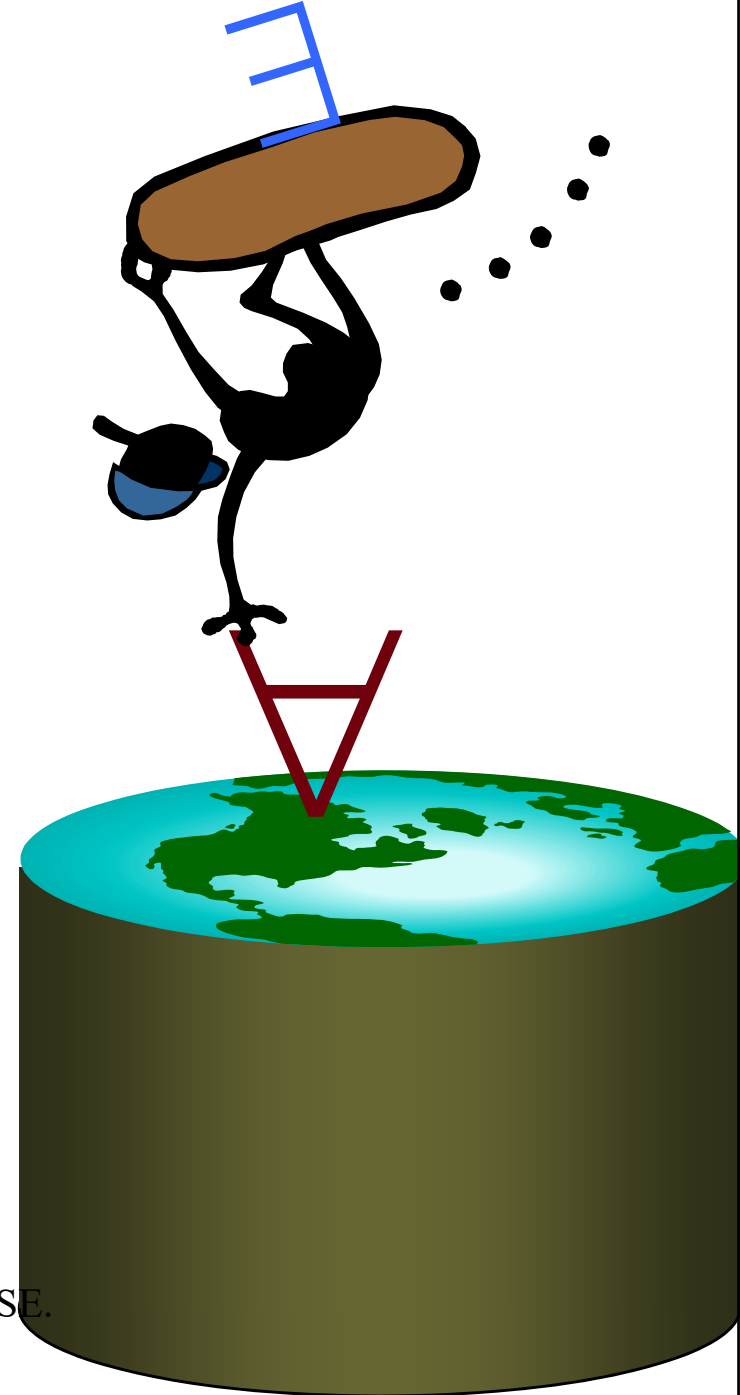
Summary

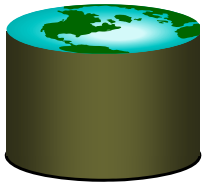
- **Relational Algebra: a small set of operators mapping relations to relations**
 - Operational, in the sense that you specify the explicit order of operations
 - A *closed* set of operators! Can mix and match.
- **Basic ops include:** σ , π , \times , \cup , $-$
- **Important compound ops:** \cap , \bowtie , $/$

Relational Calculus

We will occasionally use this arrow notation
unless there is danger of no confusion.

-- **Ronald Graham (Elements of Ramsey
Theory)**





Relational Calculus

- **High-level, first-order logic description**
- **English:**
Find all sailors with a rating above 7
- **More logical English:**
From the universe of all things, find me the set of things that happen to be tuples in the Sailors relation and whose rating field is a number greater than 7.
- **Relational Calculus (TRC)**
 $\{S \mid S \in \textit{Sailors} \wedge S.\textit{rating} > 7\}$



So what is Relational Calculus?

- **A formal, logical description, of what you want from the database.**



Relational Calculus

- **Comes in two flavors:**

- Tuple relational calculus (TRC)
- Domain relational calculus (DRC)

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

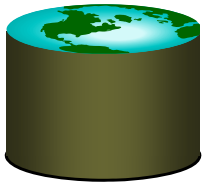
- **English Example: Find all sailors with a rating above 7**

- Tuple R.C.: From the universe of all things, find me the set of things that happen to be tuples in the Sailors relation and whose rating field is a number greater than 7.

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

- Domain R.C.: From the universe of all things, find me S, N, R, and A, where S is an integer, N is a string, R is an integer greater than 7, and A is a floating point number, and $\langle S, N, R, A \rangle$ is a tuple in the Sailors relation.

$\{\langle S, N, R, A \rangle \mid \langle S, N, R, A \rangle \in \text{Sailors} \wedge R > 7\}$



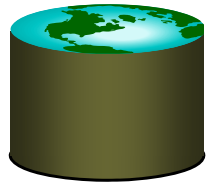
Relational Calculus (cont.)

- **Calculus has**
 - *variables*
 - TRC: Variables range over (i.e., get bound to) *tuples*. Like SQL.
 - DRC: Variables range over *domain elements* (= field values). Like Query-By-Example (QBE)
 - *constants*, e.g. 7, "Foo", 3.14159, etc.
 - *comparison ops*, e.g. =, <>, <, >, etc.
 - *logical connectives*
 - \neg - not
 - \wedge - and
 - \vee - or
 - \Rightarrow - implies
 - \in - is a member of
 - *quantifiers*
 - $\forall X(p(X))$ – $p(X)$ must be true for every X
 - $\exists X(p(X))$ – $p(X)$ is true for some X
- **Both TRC and DRC are simple subsets of first-order logic. We'll focus on TRC here**



Tuple Relational Calculus

- **Query** has the form: $\{T \mid p(T)\}$
 - $p(T)$ denotes a formula in which tuple variable T appears.
- **Answer** is the set of all tuples T for which the formula $p(T)$ evaluates to *true*.
- **Formula** is recursively defined:
 - start with simple *atomic formulas* (get tuples from relations or make comparisons of values)
 - build bigger and better formulas using the *logical connectives*.



TRC Formulas

- **An *Atomic formula* is one of the following:**

$R \in Rel$

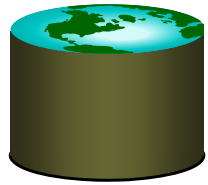
$R.a \text{ } op \text{ } S.b$

$R.a \text{ } op \text{ } constant$

op is one of $<, >, =, \leq, \geq, \neq$

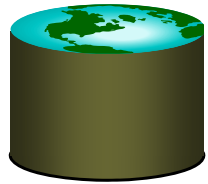
- **A *formula* can be:**

- an atomic formula
- $\neg p, p \wedge q, p \vee q$ where p and q are formulas
- $\exists R(p(R))$ where variable R is a tuple variable
- $\forall R(p(R))$ where variable R is a tuple variable



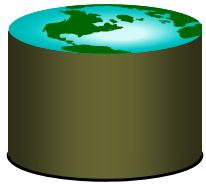
Free and Bound Variables

- The use of **quantifiers** $\forall X$ and $\exists X$ in a formula is said to **bind** X in the formula.
 - A variable that is **not bound** is free.
- Let us revisit the definition of a **query**:
 - $\{T \mid \rho(T)\}$
- There is an important restriction
 - the variable T that appears to the left of \mid must be the **only** free variable in the formula $\rho(T)$.
 - in other words, all other tuple variables must be bound using a quantifier.



Use of \forall

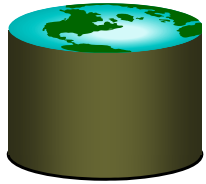
- $\forall x (P(x))$ - is only true if $P(x)$ is true for *every* x in the universe
- Usually:
$$\forall x ((x \in \text{Boats}) \Rightarrow (x.\text{color} = \text{"Red"}))$$
- \Rightarrow logical implication,
 $a \Rightarrow b$ means that if a is true, b must be true
 $a \Rightarrow b$ is the same as $\neg a \vee b$



$a \Rightarrow b$ is the same as $\neg a \vee b$

		b	
		T	F
a	T	T	F
	F	T	T

- **If a is true, b must be true!**
 - If a is true and b is false, the implication evaluates to false.
- **If a is not true, we don't care about b**
 - The expression is always true.



Quantifier Shortcuts

$$\forall x ((x \in \text{Boats}) \Rightarrow (x.\text{color} = \text{"Red"}))$$

can also be written as:

$$\forall x \in \text{Boats} (x.\text{color} = \text{"Red"})$$

$$\exists x ((x \in \text{Boats}) \wedge (x.\text{color} = \text{"Red"}))$$

can also be written as:

$$\exists x \in \text{Boats} (x.\text{color} = \text{"Red"})$$



Selection and Projection

- Find all sailors with rating above 8

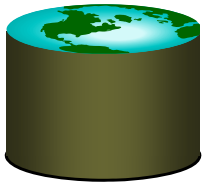
$$\{S \mid S \in \textit{Sailors} \wedge S.\textit{rating} > 8\}$$

- Find names and ages of sailors with rating above 8.

$$\{S \mid \exists S1 \in \textit{Sailors} (S1.\textit{rating} > 8 \\ \wedge S.\textit{sname} = S1.\textit{sname} \\ \wedge S.\textit{age} = S1.\textit{age})\}$$

- Note: S is a tuple variable of 2 fields (i.e. $\{S\}$ is a **projection of *Sailors***)

- only 2 fields are ever mentioned **and** S is never used to range over any relations in the query.

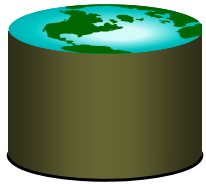


Joins

Find sailors rated > 8 who've reserved boat #103

$$\{S \mid S \in \text{Sailors} \wedge S.\text{rating} > 8 \wedge \\ \exists R(R \in \text{Reserves} \wedge R.\text{sid} = S.\text{sid} \\ \wedge R.\text{bid} = 103)\}$$

Note the use of \exists to find a tuple in Reserves that 'joins with' the Sailors tuple under consideration.



Joins (continued)

$$\{S \mid S \in \text{Sailors} \wedge S.\text{rating} > 8 \wedge \\ \exists R(R \in \text{Reserves} \wedge R.\text{sid} = S.\text{sid} \\ \wedge \exists B(B \in \text{Boats} \wedge B.\text{bid} = R.\text{bid} \\ \wedge B.\text{color} = \text{'red'}))\}$$

Find sailors rated > 8 who've reserved a red boat

- **Observe how the parentheses control the scope of each quantifier's binding.**
- **This may look cumbersome, but it's not so different from SQL!**



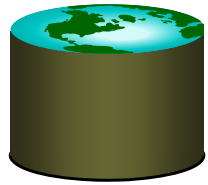
Division (makes more sense here???)

Find sailors who've reserved all boats

(hint, use \forall)

$$\{S \mid S \in \text{Sailors} \wedge \\ \forall B \in \text{Boats} (\exists R \in \text{Reserves} \\ (S.\text{sid} = R.\text{sid} \\ \wedge B.\text{bid} = R.\text{bid}))\}$$

- Find all sailors ***S*** such that for all tuples ***B*** in Boats there is a tuple in Reserves showing that sailor ***S*** has reserved ***B***.



Exercises

- Find the names of sailors who've reserved boat #103

$$\{N \mid \exists S \in \text{Sailors} (S.\text{name} = N.\text{name} \wedge \\ \exists R \in \text{Reserves} (S.\text{sid} = R.\text{sid} \\ \wedge R.\text{bid} = 103))\}$$



Exercises

- Find the names of sailors who've reserved any red boat

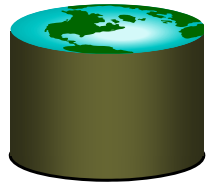
$$\{N \mid \exists S \in \text{Sailors} \\ (S.\text{name} = N.\text{name} \wedge \\ \exists R \in \text{Reserves} \\ (S.\text{sid} = R.\text{sid} \wedge \\ \exists B \in \text{Boats}(B.\text{color} = \text{"Red"} \wedge \\ B.\text{bid} = R.\text{bid}))))\}$$



Exercises

- Find sailors who've reserved a red boat or a green boat

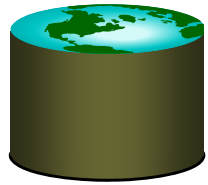
$$\{S \mid S \in \text{Sailors} \wedge$$
$$(\exists R \in \text{Reserves}$$
$$(S.\text{sid} = R.\text{sid} \wedge$$
$$\exists B \in \text{Boats}(\text{B.bid} = R.\text{bid} \wedge$$
$$(\text{B.color} = \text{"Red"} \vee$$
$$\text{B.color} = \text{"Green"}))))\}$$



Exercises

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

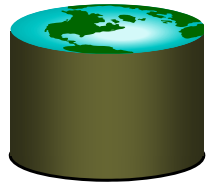
$$\{S \mid S \in \text{Sailors} \wedge$$
$$((\exists R \in \text{Reserves}$$
$$(S.\text{sid} = R.\text{sid} \wedge$$
$$\exists B \in \text{Boats}(B.\text{color} = \text{"Red"} \wedge B.\text{bid} = R.\text{bid})))$$
$$\wedge$$
$$(\exists R \in \text{Reserves}$$
$$(S.\text{sid} = R.\text{sid} \wedge$$
$$\exists B \in \text{Boats}(B.\text{color} = \text{"Green"} \wedge B.\text{bid} = R.\text{bid}))))\}$$



Exercises

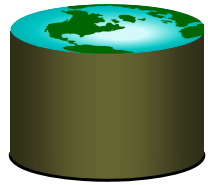
- Find sailors who've reserved all red boats

$$\{S \mid S \in \text{Sailors} \wedge \\ \forall B \in \text{Boats} (B.\text{color} = \text{'red'} \Rightarrow \\ \exists R (R \in \text{Reserves} \wedge S.\text{sid} = R.\text{sid} \\ \wedge B.\text{bid} = R.\text{bid}))\}$$
$$\{S \mid S \in \text{Sailors} \wedge \\ \forall B \in \text{Boats} (B.\text{color} \neq \text{'red'} \vee \\ \exists R (R \in \text{Reserves} \wedge S.\text{sid} = R.\text{sid} \\ \wedge B.\text{bid} = R.\text{bid}))\}$$



Unsafe Queries, Expressive Power

- \exists syntactically correct calculus queries that have an infinite number of answers! **Unsafe** queries.
 - e.g., $\{S \mid \neg(S \in Sailors)\}$
 - Solution???? Don't do that!
- **Expressive Power (Theorem due to Codd):**
 - every query that can be expressed in relational algebra can be expressed as a **safe** query in DRC / TRC; the converse is also true.
- **Relational Completeness:** Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus. (actually, SQL is more powerful, as we will see...)



Summary

- **The relational model has rigorously defined query languages — simple and powerful.**
- **Relational algebra is more operational**
 - useful as internal representation for query evaluation plans.
- **Relational calculus is non-operational**
 - users define queries in terms of what they want, not in terms of how to compute it. (*Declarative*)
- **Several ways of expressing a given query**
 - a *query optimizer* should choose the most efficient version.
- **Algebra and safe calculus have same *expressive power***
 - leads to the notion of *relational completeness*.