CompSci 516
Database Systems

Lecture 4
SQL

Instructor: Sudeepa Roy

Announcements

- Lab-1 makeup instructions sent on piazza
  - Please respond by 3 pm today if you have missed the lab
- Let me know if you are still not on piazza
- HW1 will be posted after the class
  - On sakai (data is already there)
  - Deadlines in stages
  - First deadline on 09/17

Today's topic

- Finish SQL
- RC

- Next week:
  - Tuesday: Guest Lecture by Junyang Gao: RA
  - Thursday: Lab on RA

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

Nulls and Views in SQL

Null Values

- Field values in a tuple are sometimes
  - unknown, e.g., a rating has not been assigned, or
  - inapplicable, e.g., no spouse’s name
  - SQL provides a special value null for such situations.

Standard Boolean 2-valued logic

- True = 1, False = 0
- Suppose X = 5
  - (X < 100) AND (X >= 1) is $T \wedge T = T$
  - (X > 100) OR (X >= 1) is $F \vee T = T$
  - (X > 100) AND (X >= 1) is $F \wedge T = F$
  - NOT(X = 5) is $\neg T = F$

- Intuitively,
  - T = 1, F = 0
  - For V1, V2 $\in$ {1, 0}
  - $V1 \wedge V2 = MIN(V1, V2)$
  - $V1 \vee V2 = MAX(V1, V2)$
  - $\neg(V1) = 1 - V1$

2-valued logic does not work for nulls

- Suppose rating = null, X = 5
- Is rating>8 true or false?
- What about AND, OR and NOT connectives?
  - (rating > 8) AND (X = 5)?
- What if we have such a condition in the WHERE clause?

End of Lecture 3

3-Valued Logic For Null

- TRUE (= 1), FALSE (= 0), UNKNOWN (= 0.5)
  - unknown is treated as 0.5
- Now you can apply rules from 2-valued logic!
  - For V1, V2 ∈ {1, 0, 0.5}
  - V1 ∧ V2 = MIN (V1, V2)
  - V1 ∨ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1
- Therefore,
  - NOT UNKNOWN = UNKNOWN
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN OR FALSE = UNKNOWN

New issues for Null

- The presence of null complicates many issues. E.g.:
  - Special operators needed to check if value IS/IS NOT NULL
  - Be careful!
  - "WHERE X = NULL" does not work!
  - Need to write "WHERE X IS NULL"

- Meaning of constructs must be defined carefully
  - e.g., WHERE clause eliminates rows that don't evaluate to true
  - So not only FALSE, but UNKNOWNs are eliminated too
  - very important to remember!

- But NULL allows new operators (e.g. outer joins)

- Can force "no nulls" while creating a table
  - sname char(20) NOT NULL
  - primary key is always not null

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?
- Ans: First 3, then 2

Aggregates with NULL

- COUNT, SUM, AVG, MIN, MAX (with or without DISTINCT)
  - Discards null values first
  - Then applies the aggregate
  - Except count(*)
- If only applied to null values, the result is null

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- SELECT sum(rating) from R2?
- Ans: 17

sid	sname	rating	age
22	dustin	null	45
31	lubber	null	55
58	rusty	null	35

R3

- SELECT sum(rating) from R3?
- Ans: null

Creating Relations in SQL

- Creates the "Students" relation
  - the type (domain) of each field is specified
  - enforced by the DBMS whenever tuples are added or modified

```
CREATE TABLE Students
  (sid CHAR(20),
   name CHAR(20),
   login CHAR(10),
   age INTEGER,
   gpa REAL)
```

- As another example, the "Enrolled" table holds information about courses that students take

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2))
```

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

Students

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

Enrolled

Destroying and Altering Relations

```
DROP TABLE Students
```

- Destroys the relation Students
  - The schema information *and* the tuples are deleted.

```
ALTER TABLE Students
    ADD COLUMN firstYear: integer
```

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a NULL value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO  Students (sid, name, login, age, gpa)
VALUES  (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

Integrity Constraints (ICs)

- IC: condition that must be true for any instance of the database
  - e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified
- A legal instance of a relation is one that satisfies all specified ICs
  - DBMS will not allow illegal instances
- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!

Keys in a Database

- Key / Candidate Key
- Primary Key
- Super Key
- Foreign Key

- Primary key attributes are underlined in a schema
  - Person(pid, address, name)
  - Person2(address, name, age, job)

Primary Key Constraints

- A set of fields is a key for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key

- Part 2 false? A superkey

- If there are > 1 keys for a relation, one of the keys is chosen (by DBA = DB admin) to be the primary key
  - E.g., sid is a key for Students
  - The set {sid, gpa} is a superkey.

- Any possible benefit to refer to a tuple using primary key (than any key)?

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  ???)
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )
```

- vs.

- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY ???,
   UNIQUE ??? )
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

- vs.

- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY sid,
   UNIQUE (cid, grade))
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."
- vs.
- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY sid,
   UNIQUE (cid, grade))
```

Foreign Keys, Referential Integrity

- Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation
  - Must correspond to primary key of the second relation
  - Like a `logical pointer'

- E.g. sid is a foreign key referring to Students:
  - Enrolled(sid: string, cid: string, grade: string)
  - If all foreign key constraints are enforced, referential integrity is achieved
  - i.e., no dangling references

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

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

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

Enforcing Referential Integrity

- Consider Students and Enrolled
  - sid in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted?
  - Reject it!
- What should be done if a Students tuple is deleted?
  - Three semantics allowed by SQL
  1. Also delete all Enrolled tuples that refer to it (cascade delete)
  2. Disallow deletion of a Students tuple that is referred to
  3. Set sid in Enrolled tuples that refer to it to a default sid
  4. (in addition in SQL): Set sid in Enrolled tuples that refer to it to a special value null, denoting `unknown' or `inapplicable'
- Similar if primary key of Students tuple is updated

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '000',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
       ON DELETE CASCADE
       ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations

- Can we infer ICs from an instance?
  - We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.

Example Instances

- What does the key (sid, bid, day) in Reserves mean?
- If the key for the Reserves relation contained only the attributes *(sid, bid)*, how would the semantics differ?

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

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

Views

Students(sid, name)
Enrolled(sid, cid, grade)

- A view is just a relation, but we store a definition, rather than a set of tuples

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21
```

- Views can be dropped using the DROP VIEW command

- Views and Security: Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s)
  - the above view hides courses “cid” from E

Can create a new table from a query on other tables too

SELECT… <u>INTO</u>…. FROM…. WHERE

```
SELECT S.name, E.grade
INTO YoungActiveStudents
 FROM  Students S, Enrolled E
WHERE  S.sid = E.sid and S.age<21
```

"WITH" clause – very useful!

- You will find "WITH" clause very useful!

```
WITH Temp1 AS
     (SELECT ..... ..),
     Temp2 AS
     (SELECT ..... ..)
SELECT X, Y
FROM TEMP1, TEMP2
WHERE....
```

- Can simplify complex nested queries

Overview: General Constraints

- Useful when more general ICs than keys are involved
- There are also ASSERTIONS to specify constraints that span across multiple tables
- There are TRIGGERS too : procedure that starts automatically if specified changes occur to the DBMS

```
CREATE TABLE  Sailors
    ( sid  INTEGER,
    sname  CHAR(10),
    rating  INTEGER,
    age  REAL,
    PRIMARY KEY  (sid),
    CHECK  ( rating >= 1
        AND rating <= 10 )
```

```
CREATE TABLE  Reserves
    ( sname  CHAR(10),
    bid  INTEGER,
    day  DATE,
    PRIMARY KEY  (bid,day),
    CONSTRAINT  noInterlakeRes
    CHECK  (`Interlake' <>
            ( SELECT  B.bname
            FROM  Boats B
            WHERE  B.bid=bid)))
```

Summary: SQL

- SQL has a huge number of constructs and possibilities
  - You need to learn and practice it on your own
- Can limit answers using "LIMIT" or "TOP" clauses
  - e.g. to output TOP 20 results according to an aggregate
  - also can sort using ASC or DESC keywords
- We learnt
  - Creating/modifying relations
  - Specifying integrity constraints
  - Key/candidate key, superkey, primary key, foreign key
  - Conceptual evaluation of SQL queries
  - Joins
  - Group bys and aggregates
  - Nested queries
  - NULLs
  - Views

Relational Query Languages

Relational Query Languages

- Query languages: Allow manipulation and retrieval of data from a database
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for much optimization
- Query Languages != programming languages
  - QLs not intended to be used for complex calculations
  - QLs support easy, efficient access to large data sets

Formal Relational Query Languages

- Two "mathematical" Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - Relational Calculus: Lets users describe what they want, rather than how to compute it (Non-operational, declarative, or procedural)
  - Relational Algebra: More operational, very useful for representing execution plans
- Note: Declarative (RC, SQL) vs. Operational (RA)

Relational Calculus (RC)

Please see updated Lecture notes in Lecture 7

Relational Algebra (RA)

Relational Algebra

- Takes one or more relations as input, and produces a relation as output
  - operator
  - operand
  - semantic
  - so an algebra!
- Since each operation returns a relation, operations can be composed
  - Algebra is "closed"

Relational Algebra

- Basic operations:
  - Selection (σ) Selects a subset of rows from relation
  - Projection (π) Deletes unwanted columns from relation.
  - Cross-product (x) Allows us to combine two relations.
  - Set-difference (-) Tuples in reln. 1, but not in reln. 2.
  - Union ($\cup$) Tuples in reln. 1 or in reln. 2.
- Additional operations:
  - Intersection ($\cap$)
  - join $\bowtie$
  - division(/)
  - renaming (ρ)
  - Not essential, but (very) useful.

Example Schema and Instances

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

S1

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

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

R1

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

Projection

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

- Deletes attributes that are not in projection list.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate duplicates (Why)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it (performance)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection

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

- Selects rows that satisfy selection condition
- No duplicates in result. Why?
- Schema of result identical to schema of (only) input relation

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

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

Composition of Operators

- Result relation can be the input for another relational algebra operation
  - Operator composition

sid	sname	rating	age
28	yuppy	9	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, Intersection, Set-Difference

S1

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

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

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - `Corresponding' fields have the same type
  - same schema as the inputs

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

$S1 \cup S2$

Union, Intersection, Set-Difference

S1

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

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

- Note: no duplicate
  - "Set semantic"
  - SQL: UNION
  - SQL allows "bag semantic" as well: UNION ALL

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

$S1 \cup S2$

Union, Intersection, Set-Difference

S1

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

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

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

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

$S1 \cap S2$

Cross-Product

- Each row of S1 is paired with each row of R.
- Result schema has one field per field of S1 and R, with field names `inherited' if possible.
  – Conflict: Both S1 and R have a field called sid.

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

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

(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

Renaming Operator ρ

$$(\rho_{sid \rightarrow sid1}\, S1) \times (\rho_{sid \rightarrow sid1}\, R1)$$

or

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

C is the new relation name

(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

- In general, can use ρ(<Temp>, <RA-expression>)

Joins

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

(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.
- Fewer tuples than cross-product, might be able to compute more efficiently

Find names of sailors who've reserved boat #103

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Find names of sailors who've reserved boat #103

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

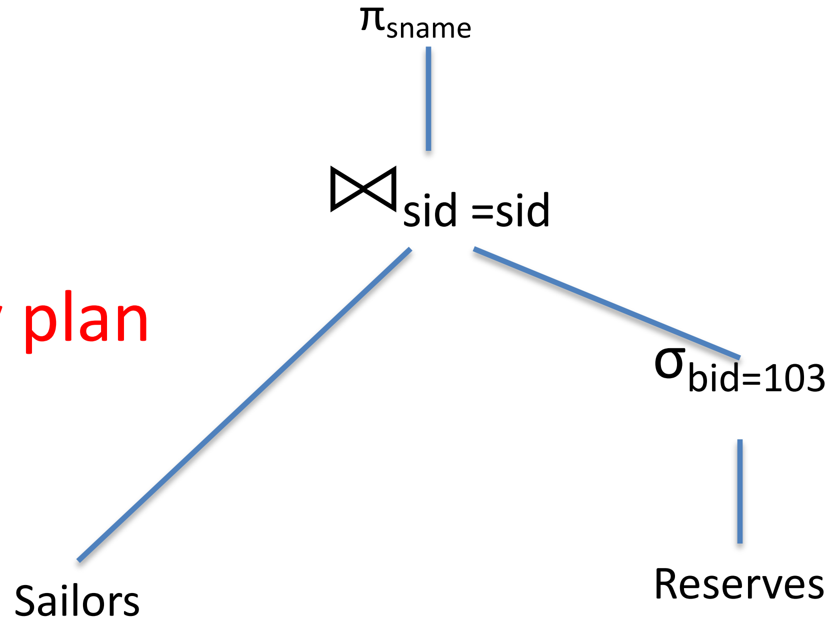
- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$
- Solution 2: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Expressing an RA expression as a Tree

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)



Also called a logical query plan

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

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

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

Use of rename operation

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

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie \text{Re}serves \bowtie Sailors)$$

Can also define Tempboats using union
Try the "AND" version yourself

What about aggregates?

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- Extended relational algebra
- $\gamma_{\text{age, avg(rating)} \rightarrow \text{avgr}}$ Sailors
- Also extended to "bag semantic": allow duplicates
  - Take into account cardinality
  - R and S have tuple t resp. m and n times
  - $R \cup S$ has t m+n times
  - $R \cap S$ has t min(m, n) times
  - R – S has t max(0, m-n) times
  - sorting(τ), duplicate removal (δ) operators