Database Systems
(資料庫系統)

September 27, 2003

Lecture #3

By Hao-hua Chu (朱浩華)

1

Course Administration

- Assignment #1 is due today.
- Assignment #2 is out on the home webpage.
  - It is due one week from today.
- Next week reading:
  - Chapter 8: Overview of Storage and Indexing

2

Ubicomp Project of the Week: House_n Project (MIT)

- Elders will not feel that they "live alone."
- Virtually connect elder's home to caregiver family members.

3

Relational Algebra

Chapter 4.1 – 4.2

4

Relational Query Languages

- What are query languages?
  - For asking questions about the database
- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - *Relational Algebra*:
    - A query is composed of a collection of step-by-step operators
    - More operational
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it.
    - Non-operational, *declarative*.

5

Preliminaries

- A query is applied to *relation instances (tables)*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed.
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in SQL

6

Example Instances

R1

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

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation,
  - Assume that names of fields in query results are `inherited' from names of fields in query input relations.

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

7

Relational Algebra

- Basic relational algebra operators:
  - *Selection* (σ , Greek letter pronounced sigma) 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* (U) Tuples in reln. 1 and in reln. 2.
- Additional relational algebra operators:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Each operator can take one or two input relation(s), and returns one relation.
- Since each operation returns a relation, operations can be *composed* to form a very complex query (very powerful).

8

Projection

- Deletes attributes that are not in *projection list*.
- Projection operator eliminates *duplicates*.

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

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

age
35.0
55.5

$\pi_{age}(S2)$

9

Selection

- Selects rows that satisfy *selection condition*.
  - No duplicates in result!
- *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

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

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

sname	rating
yuppy	9
rusty	10

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

10

Union, Intersection, 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.
- What is the *schema* of result?

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

$S1 \cup S2$

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

11

Intersection

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

$S1 \cap S2$

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

12

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

$S1 - S2$

13

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

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

S1 x R1

(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*: $\rho\,(C(1 \to sid1, 5 \to sid2),\ S1 \times R1)$

14

Condition 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

R

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

S

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

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

- Cross-product, followed by a selection
- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, reduce tuples not meeting the condition.

15

Equi-Joins

- *Equi-Join*: A special case of condition join where the condition *c* contains only ***equalities*.**
- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join (⋈)*: Equi-join on *all* common fields.

R

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

S

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

$S1 \bowtie_{sid} R1$

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

16

Division

- Not supported as a primitive operator, but useful for expressing queries like:
  - *Find sailors who have reserved **all** boats.*
- Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
  - $A/B = \{\langle x \rangle \mid \exists \langle x, y \rangle \in A \ \ \forall \langle y \rangle \in B\}$
  - i.e., ***A/B* contains all *x* tuples (sailors) such that for *every y* tuple (boat) in *B*, there is an *xy* tuple in *A*.**
  - *Or*: If the set of *y* values (boats) associated with an *x* value (sailor) in *A* contains all *y* values in *B*, the *x* value is in *A/B*.

17

Examples of Division A/B

X	Y
X1	Y1
X1	Y2
X1	Y3
X1	Y4
X2	Y1
X2	Y2
X3	Y2
X4	Y2
X4	Y4

A

Y
Y2

B1

Y
Y2
Y4

B2

Y
Y1
Y2
Y4

B3

X
X1
X2
X3
X4

A/B1

X
X1
X4

A/B2

X
X1

A/B3

18

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: $\pi_x(A)$ − all disqualified tuples

19

Find names of sailors who've reserved boat #103

Reserves(sid, bid, day) Sailors(sid, sname, rating, age)

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

❖ Solution 2:

$$\rho(Temp1, \sigma_{bid=103}\text{Re}serves)$$

$$\rho(Temp2, Temp1 \bowtie Sailors)$$

$$\pi_{sname}(Temp2)$$

❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Re}serves \bowtie Sailors))$

20

Find names of sailors who've reserved a red boat

Reserves(sid, bid, day)

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

- 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!

21

Find the sids of sailors with age over 20 who have not reserved a red boat

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

- Solution:
  - Find all sailors (sids) with age over 20
  - Find all sailors (sids) who have reserved a red boat
  - Take their set differences
- π_{sid} ($\sigma_{age>20}$ Sailors) –
  π_{sid} (($\sigma_{color='red'}$ Boats) ⋈ Reserves ⋈ Sailors)

22

SQL: Queries, Constraints, Triggers

Chapter 5

23

Lecture Outline

- Basic Query
  - **select**
- Set Constructs
  - **union, intersect, except, in, any, all, exists**
- Nested Queries
- Aggregate Operators
  - **count, sum, avg, max, min, group by, having**
- Null Values
- Integrity Constraints
  - **check, create assertion**
- Triggers
  - **create trigger, for each row**

24

Example Table Definitions

- Sailors(sid: integer, sname: string, rating: integer, age: real)
- Boats(bid: integer, bname: string, color: string)
- Reserves(sid: integer, bid: integer, day: date)

25

Basic SQL Query

SELECT	[DISTINCT] *target-list*
FROM	*relation-list*
WHERE	*qualification*

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, >, =, ≤, ≥, ≠) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. **Default** is that duplicates are *not* eliminated!

26

Conceptual Evaluation Strategy

SELECT	[DISTINCT] *target-list*
FROM	*relation-list*
WHERE	*qualification*

- A query can be evaluated using following "conceptual evaluation strategy":
  1. Compute the cross-product of *relation-list*.
  2. Discard resulting tuples if they fail *qualifications*.
  3. Delete attributes that are not in *target-list*. (called column-list)
  4. If DISTINCT is specified, eliminate duplicate rows.
- This is called conceptual evaluation strategy because it is probably the least efficient way to compute a query!
  - An optimizer will find more efficient strategies to compute **the same answers**.

27

Example of Conceptual Evaluation (1)

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

(1) Compute the cross-product of *relation-list*.

Sailors

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

×

Reserves

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

28

Example of Conceptual Evaluation (2)

SELECT S.sname
FROM Sailors S, Reserves R
WHERE **S.sid=R.sid AND R.bid=103**

(2) Discard resulting tuples if they fail *qualifications*.

(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

29

Example of Conceptual Evaluation (3)

SELECT **S.sname**
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103

(3) Delete attributes that are not in *target-list*. (or column-list).

(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

30

A Note on Range Variables

```
SELECT  S.sname
FROM    Sailors as S, Reserves R
WHERE   S.sid=R.sid AND bid=103
```

OR

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
        AND bid=103
```

range variables.

- Really needed range variables only if the same relation appears twice in the FROM clause.

```
SELECT  sname
FROM    Sailors S, Reserves R1,
        Reserves R2
WHERE   S.sid = R1.sid AND
        S.sid = R2.sid AND
        R1.bid <> R2.bid
```

31

Find sailors who've reserved at least one boat

```
SELECT  S.sid
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid
```

Sailors X Reserves

(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

32

DISTINCT

- Find the names and ages of all sailors

```
SELECT  S.names, S.ages
FROM    Sailors S
```

- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing S.sid by S.sname in the SELECT clause?
- Would adding DISTINCT to this variant of the query make a difference?

Sid	Sname	Rating	Age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

33

Expressions and Strings

```
SELECT  S.age,
        age1=S.age-5,
        2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find sailors whose names begin and end with B and contain at least three characters.*
- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.
  - WHERE S.name LIKE '_O%'

Sid	Sname	Rating	Age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	20

Age	Age1	Age2
20	15	40

34

SET Construct: UNION

- Find sid's of sailors who've reserved a red or a green boat:

```
SELECT  DISTINCT S.sid
FROM    Sailors S, Boats B, Reserves R
WHERE   S.sid=R.sid AND R.bid=B.bid
        AND (B.color='red' OR B.color='green')
```

- UNION: compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).

```
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
WHERE   S.sid=R.sid AND R.bid=B.bid AND B.color='red'
UNION
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
WHERE   S.sid=R.sid AND R.bid=B.bid AND B.color='green'
```

- If we replace OR by AND in the first version, what do we get?

35

SET Construct: INTERSECT

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

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
          AND B.color='red'
INTERSECT
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
          AND B.color='green'
```

- What do we get if we replace INTERSECT by EXCEPT?
  - (A Except B) returns tuples in A but not in B.
  - Find sids of all sailors who have reserved a red boat but not a green boat.

36

SET Construct: UNION ALL

- UNION, INTERSECT, and EXCEPT delete duplicate by default.
- To retain duplicates, use UNION ALL, INTERSECT ALL, or EXCEPT ALL.

Sid	Sname
71	Zorba
74	Horatio
85	Art
95	Bob

UNION ALL

Sid	Sname
22	Dustin
29	Brutus
71	Zorba
74	Horatio

=

Sid	Sname
71	Zorba
74	Horatio
71	Zorba
74	Horatio

37

Nested Queries

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL **subquery**! (Actually, so can FROM and HAVING clauses.)
- Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN  (SELECT  R.sid
                   FROM    Reserves R
                   WHERE   R.bid=103)
```

Subquery: finds sids who have reserved bid 103

- (x IN B) returns true when x is in set B.
  - To find sailors who've not reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*
- Subquery result does not change with each new Sailor row.

38

Nested Queries with Correlation

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS  (SELECT  *
                 FROM  Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```

Correlation: subquery finds all reservations for bid 103 from current sid

- EXISTS is another set comparison operator, like IN.
  - (EXISTS S) returns true when S is not empty.
- What is the above query in English?
  - To find sailors who have not reserved boat #103
- In case of correlation, subquery must be re-computed for each Sailors tuple.

39

Nested Queries with Unique

- (UNIQUE S) returns true if S has no duplicate tuples or S is empty.

```
SELECT  S.sname
FROM    Sailors S
WHERE   UNIQUE  (SELECT R.bid
                 FROM    Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```

- What is the above query in English?
  - Finds sailors with at most one reservation for boat #103.
- What happens when R.bid is replaced with *?
  - Finds sailors with at most one reservation for boat #103 in a given day.

40

More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, where *op* can be >, <, =, ≠, ≤, ≥
  - (a > ANY B) returns true when a is greater than any one element in set B.
  - (a > ALL B) returns true when a is greater than all elements in set B.

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY  (SELECT  S2.rating
                        FROM   Sailors S2
                        WHERE S2.sname='Horatio')
```

- What is the above query in English?
  - Find sailors whose rating is greater than that of some sailor called Horatio:

41

Rewriting INTERSECT Queries Using IN

Find sid's of sailors who've reserved both a red and a green boat:

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color='red'
       AND S.sid IN  (SELECT  S2.sid
                      FROM  Sailors S2, Boats B2, Reserves R2
                      WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
                             AND  B2.color='green')
```

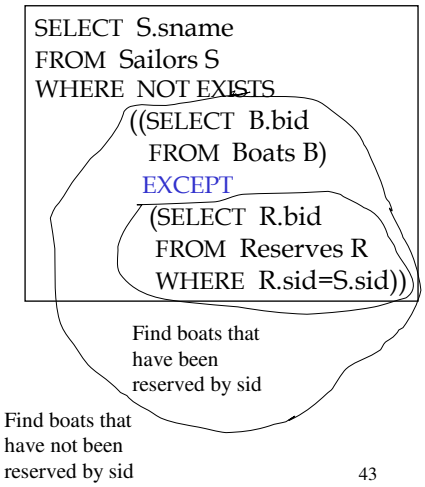
Subquery: find sids who've reserved a green boat

- Find *sid*'s of Sailors who've reserved red but not green boats.
  - Replace IN with NOT IN.
- Similarly, EXCEPT queries can be re-written using NOT IN.

42

Division in SQL

- Find sailors who've reserved all boats.
  - Find all boats that have been reserved by a sailor
  - Compare with all boats
  - Do the sailor's reserved boats include all boats?
    - Yes -> include this sailor
    - No -> exclude this sailor
- Can you do it the hard way, without EXCEPT & with NOT EXISTS?




```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
        (SELECT  B.bid
          FROM  Boats B
          WHERE NOT EXISTS
                  (SELECT  R.bid
                  FROM  Reserves R
                  WHERE  R.sid=S.sid AND
                          R.bid = B.bid))
```

Return the set of boats that have not been reserved by this sailor

Return if this boat has been reserved by this sailor or not

Aggregate Operators

- COUNT (*)
- COUNT ([DISTINCT] A)
  - A is a column
- SUM ([DISTINCT] A)
- AVG ([DISTINCT] A)
- MAX (A)
- MIN (A)
- Count the number of sailors

```
SELECT  COUNT (*)
FROM    Sailors S
```

- Find the average age of sailors with rating = 10

```
SELECT  AVG (S.age)
FROM    Sailors S
WHERE   S.rating=10
```

- Count the number of different sailor names

```
SELECT  COUNT (DISTINCT
   S.sname)
FROM    Sailors S
```

- Find the age of the oldest sailor:

```
SELECT  MAX(S.AGE)
FROM    Sailors S
```

45

Find name and age of the oldest sailor(s)

Cannot combine a column with a value (unless we use GROUP BY)

- The first query is illegal! (We'll look into the reason a bit later, when we discuss GROUP BY.)

```
SELECT S.sname, MAX (S.age)
FROM Sailors S
```

- The second query is allowed in the SQL/92 standard, but is not supported in some systems.

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age =
      (SELECT MAX (S2.age)
       FROM Sailors S2)
```

Convert a table (of a single aggregate value) into a single value for comparison

46

GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.
- Consider: Find the age of the youngest sailor for each rating level.
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

For i = 1, 2, ... , 10:

```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

47

Find the age of the youngest sailor for each rating level

```
SELECT     S.rating, MIN (S.age) as age
FROM       Sailors S
GROUP BY  S.rating
```

(1) The sailors tuples are put into "same rating" groups.

(2) Compute the Minimum age for each rating group.

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

Rating	Age
3	25.5
3	63.5
7	45.0
8	55.5
8	25.5

(1)

(2)

Rating	Age
3	25.5
7	45.0
8	25.5

48

Find the age of the youngest sailor for each rating level that has at least 2 members

```
SELECT    S.rating, MIN (S.age) as age
FROM      Sailors S
GROUP BY  S.rating
HAVING    COUNT(*) > 1
```

1. The sailors tuples are put into "same rating" groups.
2. Eliminate groups that have < 2 members.
3. Compute the Minimum age for each rating group.

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

Rating	Age
3	25.5
3	63.5
7	45.0
8	55.5
8	25.5

(1) (2)

Rating	Age
3	25.5
8	25.5

(3)

49

Queries With GROUP BY and HAVING

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
```

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., AVG (*S.age*)).
  - The attribute list (i) in target-list must be in *grouping-list*.
  - The attributes in group-qualification must be in *grouping-list*.
  - Why? See next two slides …

50

Attribute list is not in grouping-list

```
SELECT    S.sname, S.rating, AVG (S.age)
  as age
FROM      Sailors S
GROUP BY  S.rating
HAVING    COUNT(*) > 1
```

Not in group-list

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

Sname	Rating	Age
Dustin	3	25.5
Lubber	3	63.5
~~Art~~	~~7~~	~~45.0~~
Andy	8	55.5
Bob	8	25.5

Sname	Rating	Age
?	3	44.5
?	8	40.5

51

Group list is not in grouping-list

```
SELECT    S.rating, AVG (S.age) as age
FROM      Sailors S
GROUP BY  S.rating
HAVING    S.sname ≠ 'Dustin'
```

Not in group-list

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

Sname	Rating	Age
Dustin	3	25.5
Lubber	3	63.5
Art	7	45.0
Andy	8	55.5
Bob	8	25.5

?

Rating	Age

52

Conceptual Evaluation

- Without GROUP BY and HAVING:
  - The cross-product of *relation-list* is computed
  - Tuples that fail *qualification* are discarded
  - *unnecessary* fields are deleted
- With GROUP BY and HAVING, continue with
  - The remaining tuples are partitioned into groups by the value of attributes in *grouping-list* (in the GROUP BY clause).
  - The *group-qualification* (in the HAVING clause) is then applied to eliminate some groups.
  - One answer tuple is generated per qualifying group.

53

For each red boat, find the number of reservations for this boat

```
SELECT    B.bid, COUNT (*) AS scount
FROM      Boats B, Reserves R
WHERE     R.bid=B.bid AND B.color='red'
GROUP BY  B.bid
```

- What do we get if we remove *B.color='red'* from the WHERE clause and add a HAVING clause with this condition?
  - Illegal! because B.color column does not appear in group-list.

```
SELECT    B.bid, COUNT (*) AS scount
FROM      Boats B, Reserves R
WHERE     R.bid=B.bid
GROUP BY  B.bid
HAVING    B.color='red'
```

54

Find the age of the youngest sailor with age > 18 for each rating with at least 2 sailors (of any age)

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age > 18
GROUP BY  S.rating
HAVING  COUNT(*) > 1
```

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age > 18
GROUP BY  S.rating
HAVING  1  <  (SELECT  COUNT (*)
                FROM  Sailors S2
                WHERE  S.rating=S2.rating)
```

- Is the left query correct? Why not?
  - COUNT(*) is counting tuples after the qualification (S.age > 18).
    - Eliminate groups that have multiple sailors but only one sailor with age > 18.
- Use subquery in the HAVING clause.

55

Find rating(s) for which the average age is the minimum over all rating groups

```
SELECT  S.rating
FROM  Sailors S
WHERE  S.age =  (SELECT  MIN (AVG (S2.age))
                 FROM  Sailors S2
                 GROUP BY S2.rating)
```

- Is the above query correct?
  - Aggregate operations cannot be nested

```
SELECT  Temp.rating
FROM  (SELECT  S.rating, AVG (S.age) AS avgage
       FROM  Sailors S
       GROUP BY  S.rating) AS Temp
WHERE  Temp.avgage = (SELECT  MIN (Temp.avgage)
                      FROM  Temp)
```

A temp table (rating, avg age)

56

Table Constraints

- Specify constraints over a single table
  - Useful when more general ICs than keys are involved.
- Constraints can be named.

```
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)))
```

The boat 'Interlake' cannot be reserved

57

Assertions: Constraints Over Multiple Tables

- Awkward and wrong!
  - If Sailors is empty, the number of Boats tuples can be anything!
- ASSERTION is the right solution; not associated with either table.

```
CREATE TABLE Sailors
        ( sid INTEGER,
        sname CHAR(10),
        rating INTEGER,
        age REAL,
        PRIMARY KEY (sid),
        CHECK
        ( (SELECT COUNT (S.sid) FROM Sailors S)
        + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

Number of boats plus number of sailors is < 100

```
CREATE ASSERTION smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

58

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- A trigger has three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)

```
CREATE TRIGGER incr_count AFTER INSERT ON Students // Event
WHEN (new.age < 18)  // Condition
FOR EACH ROW
    BEGIN   // ACTION: a procedure in Oracle's PL/SQL syntax
        count := count + 1
    END
```

59

Summary

- How to specify queries using relational algebra?
- How to specify queries in SQL?
- What is grouping?
- What are nested queries?
- How to specify constraints, assertions, and triggers?

60