

CS432 Fall 2007

Example Instan	R1 ICES	<u>sid</u> 22 58	<u>bic</u> 10 10	1	10/1	<u>ay</u> 10/96 12/96
♦ We will use these S1	<u>sid</u>	snan	ne	rat	ing	age
instances of the Sailors and	22	dust	in	1	7	45.0
Reserves relations in our examples. If the key for the Reserves relation S2	31	lubb	er	8	3	55.5
	58	rusty	/	1	10	35.0
	<u>sid</u>	snan	ne	rat	ing	age
contained only the attributes <i>sid</i> and	28	yupp	у	9	9	35.0
<i>bid,</i> how would the	31	lubb	er	8	8	55.5
semantics differ?	44	gupp	у	4	5	35.0
	58	rusty	y		10	35.0
CS432 Fall 2007						

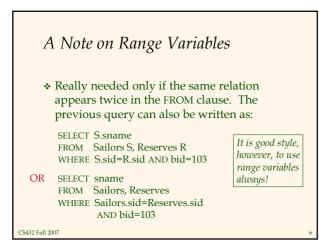
-	

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). * <u>target-list</u> A list of attributes of relations in *relation-list* * *qualification* Comparisons (Attr op const or Attr1 op Attr2, where *op* is one of $\langle , \rangle, =, \leq, \geq, \neq$) combined using AND, OR and NOT. * **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are <u>not</u> eliminated! CS432 Fall 2007

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
 - Compute the cross-product of *relation-list*.
 - Discard resulting tuples if they fail *qualifications*.
 - Delete attributes that are not in *target-list*.
 - If **DISTINCT** is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

	FROM	S.snar Sailors S.sid=	5 S, Re			=103
(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



Find sailors who've reserved at least one boat

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

- 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?

CS432 Fall 2007

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 triples (of ages of sailors and two fields defined by expressions) for 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.

CS432 Fall 2007

Find sid's of sailors who've reserved a red <u>or</u> a green boat

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- If we replace OR by AND in the first version, what do we get?

 Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

CS432 Fall 2007

SELECT 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')

SELECT S.sid

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

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

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

SELECT S.sid FROM Sailors S, Boats B1, Reserves R1,

- * INTERSECT: Can be used to of any two union*compatible* sets of tuples.
- ✤ Included in the SQL/92 standard, but some systems don't support it.
- * Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

CS432 Fall 2007

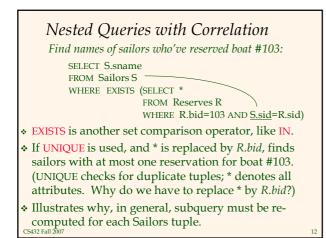
Boats B2, Reserves R2 compute the intersection of any two union AND S.sid=R1.sid AND R1.bid=B1.bid AND S.sid=R2.sid AND R2.bid=B2.bid AND (B1.color='red' AND B2.color='green') Key field! SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red' INTERSECT SELECT S.sid FROM Sailors S, Boats B, Reserves R

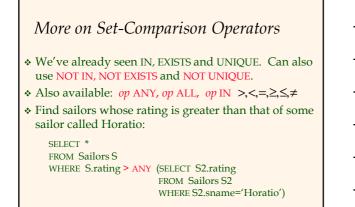
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'

Nested Queries

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)

- ✤ A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses.)
- ✤ 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.* CS432 Fall 2007





CS432 Fall 2007

Rewriting INTERSECT Queries Using IN

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

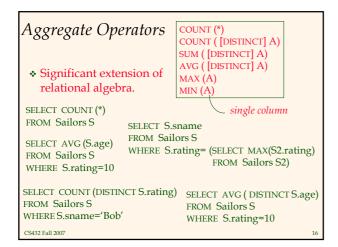
SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red' AND S.sid IN (SELECT S2.sid FROM Sailors S2, Boats B2, Reserves R2 WHERE S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color='green')

Similarly, EXCEPT queries re-written using NOT IN.

 To find *names* (not *sid*'s) of Sailors who've reserved both red and green boats, just replace *S.sid* by *S.sname* in SELECT clause. (What about INTERSECT query?)

(1)	SELECT S.sname			
	FROM Sailors S			
Division in SQL	WHERE NOT EXISTS			
	((SELECT B.bid			
	FROM Boats B)			
Find sailors who've reserved all boats.				
	(SELECT R.bid			
✤ Let's do it the hard	FROM Reserves R			
way, without EXCEPT:	WHERE R.sid=S.sid))			
(2) SELECT S.sname				
FROM Sailors S				
WHERE NOT EXISTS (SELECT B.bid				
FROM Boats B				
Sailors S such that WHERE NOT EX	ISTS (SELECT R.bid			
5uitors 5 such that	FROM Reserves R			
there is no boat B without	WHERE R.bid=B.bid			
mere is no bout D without	AND R.sid=S.sid))			
a Reserves tuple showing S reserved B				
CS432 Fall 2007	15			







Find name and age of the oldest sailor(s)

- The first query is illegal! (We'll look into the reason a bit later, when we discuss GROUP BY.)
- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.

CS432 Fall 2007

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

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

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

Motivation for Grouping

- 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 (!):

SELECT MIN (S.age)

FROM Sailors S WHERE S.rating = i

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

Queries With GROUP BY and HAVING

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

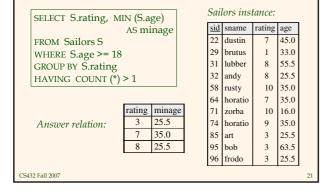
- The target-list contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (S.age)).
 - The <u>attribute list (i)</u> must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

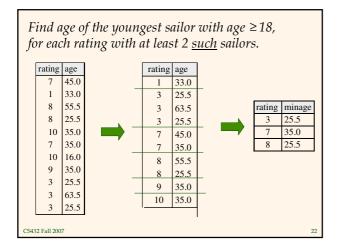
CS432 Fall 2007

Conceptual Evaluation

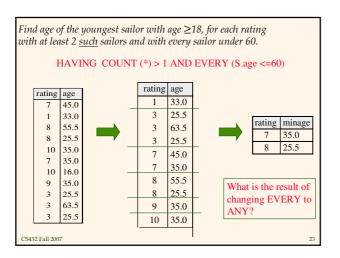
- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary*' fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The group-qualification is then applied to eliminate some groups. Expressions in group-qualification must have a <u>single value per group</u>!
 - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group. CS432 Fall 2007

Find age of the youngest sailor with age ≥ 18 , for each rating with at least 2 <u>such</u> sailors

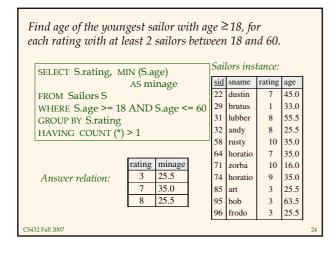














For each red boat, find the number of reservations for this boat

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

- * Grouping over a join of three relations.
- What do we get if we remove B.color='red' from the WHERE clause and add a HAVING clause with this condition?
- What if we drop Sailors and the condition involving S.sid?

CS432 Fall 2007

Find 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 1 < (SELECT COUNT (*) FROM Sailors S2 WHERE S.rating=S2.rating)

- Shows HAVING clause can also contain a subquery.
- Compare this with the query where we considered only ratings with 2 sailors over 18!
- What if HAVING clause is replaced by:

HAVING COUNT(*) >1
 CS432 Fall 2007

Find those ratings for which the average age is the minimum over all ratings

* Aggregate operations cannot be nested! WRONG:

SELECT Temp.rating, Temp.avgage 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)

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 <u>null</u> for such situations.

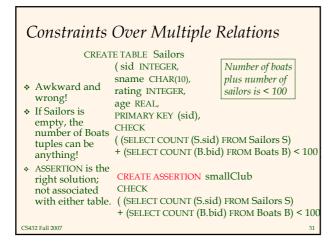
- * The presence of *null* complicates many issues. E.g.:
 - Special operators needed to check if value is/is not *null*.
 - Is *rating*>8 true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
 - We need a <u>3-valued logic</u> (true, false and *unknown*).
 - Meaning of constructs must be defined carefully. (e.g.,
 - WHERE clause eliminates rows that don't evaluate to true.)New operators (in particular, *outer joins*) possible/needed.
- 28432 Fall 2007

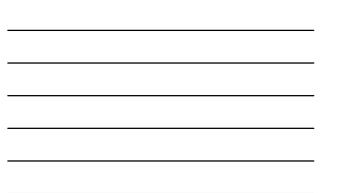
Integrity Constraints (Review)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
 - Inserts/deletes/updates that violate IC's are disallowed.
 - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- <u>Types of IC's</u>: Domain constraints, primary key constraints, foreign key constraints, general constraints.
 - *Domain constraints*: Field values must be of right type. Always enforced.

CS432 Fall 2007

CREATE TABLE Sailors General Constraints (sid INTEGER, sname CHAR(10), rating INTEGER, age REAL, ✤ Useful when PRIMARY KEY (sid), more general CHECK (rating >= 1 AND rating ≤ 10) ICs than keys CREATE TABLE Reserves are involved. (sname CHAR(10), Can use queries bid INTEGER, to express day DATE, constraint. PRIMARY KEY (bid,day), * Constraints can **CONSTRAINT** noInterlakeRes be named. CHECK (`Interlake' <> (SELECT B.bname FROM Boats B WHERE B.bid=bid))) CS432 Fall 2007





Triggers

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

CS432 Fall 2007

Triggers: Example (SQL:1999)

CREATE TRIGGER youngSailorUpdate AFTER INSERT ON SAILORS REFERENCING NEW TABLE NewSailors FOR EACH STATEMENT INSERT INTO YoungSailors(sid, name, age, rating) SELECT sid, name, age, rating FROM NewSailors N WHERE N.age <= 18

Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
 - In practice, users need to be aware of how queries are optimized and evaluated for best results.

CS432 Fall 2007

Summary (Contd.)

- NULL for unknown field values brings many complications
- SQL allows specification of rich integrity constraints
- * Triggers respond to changes in the database