Database Management Systems

Ehsan Noei
e.noei@utoronto.ca

UNIVERSITY OF TORONTO

Example

Sailors (sid: integer, sname: string, rating: integer, age: real)

Boats (bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer, day: date)

R1

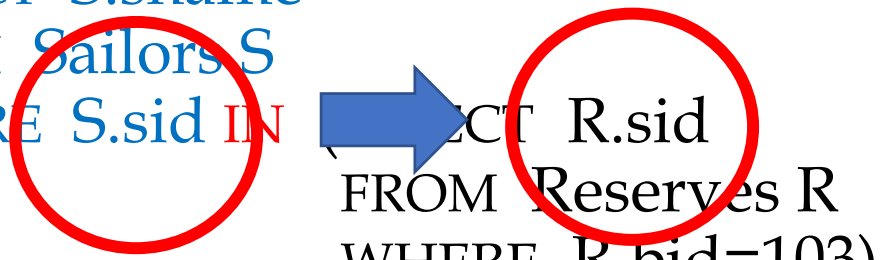
sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

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

IN - NOT IN

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



R1

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

EXISTS - NOT EXISTS

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
              FROM AnotherTable A
              WHERE A.age>20)
```

EXISTS - NOT EXISTS

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

- EXISTS is another set comparison operator, like IN.
- Allows test whether a set is nonempty.

Sort

SELECT sname, age

FROM student

ORDER BY age asc/desc

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.

Find the age of the youngest sailor for each rating level.

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

Example

store	product	date	sale
1	1	1	10
1	1	2	15
1	2	1	20
1	2	2	25
1	3	1	5
1	3	2	10
2	1	1	100
2	1	2	150
2	2	1	200
2	2	2	250
2	3	1	50
2	3	2	100

Select store, product, sum(sale)

from R

group by store

Example

store	product	date	sale
1	1	1	10
1	1	2	15
1	2	1	20
1	2	2	25
1	3	1	5
1	3	2	10
2	1	1	100
2	1	2	150
2	2	1	200
2	2	2	250
2	3	1	50
2	3	2	100

Select store, product, sum(sale)

from R

group by store, product

Queries With GROUP BY and HAVING

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

Conceptual Evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, *unnecessary* fields are deleted.
- 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.
- One answer tuple is generated per qualifying group.

Find the age of the youngest sailor with age 18, for each rating with at least 2 such sailors.

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

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

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

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

Find the average age of sailors for each rating level that has at least two sailors.

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

Find the average age of sailors for each rating level that has at least two sailors (ALTERNATIVE SOLUTION).

```
SELECT S.rating, AVG ( S.age ) AS average
FROM Sailors S
GROUP BY S.rating
HAVING 1 < ( SELECT COUNT (*)
                FROM Sailors S2
                WHERE S.rating = S2.rating )
```

HAVING clause can also contain a subquery.

HAVING clause can also contain a subquery.

- HAVING COUNT(*)>10
- HAVING EVERY (color='green')
- HAVING ANY (age < 18)

```
SELECT Temp.rating, Temp.average
FROM (SELECT S.rating, AVG (S.age) AS average
      FROM Sailors S
      GROUP BY S.rating) AS Temp
```

Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned).
  - IS NULL
  - IN NOT NULL

```
SELECT S.rating
FROM Sailors S
WHERE S.rating IS NOT NULL
```

EXAMPLE

```
MySQL  localhost:3306  dbbook  SQL > select * from emp;
+-----+-------+------+--------+
| eid | ename | age  | salary |
+-----+-------+------+--------+
|   1 | a     |    2 |      5 |
|   2 | b     |    3 |   NULL |
|   3 | c     | NULL |      6 |
|   4 | d     | NULL |   NULL |
+-----+-------+------+--------+
4 rows in set (0.0004 sec)
```

EXAMPLE

```
MySQL  localhost:3306  dbbook  SQL > select * from emp
                                   -> where age>1;
+-----+-------+-----+--------+
| eid | ename | age | salary |
+-----+-------+-----+--------+
|   1 | a     |   2 |      5 |
|   2 | b     |   3 |   NULL |
+-----+-------+-----+--------+
2 rows in set (0.0004 sec)
```

EXAMPLE

```
MySQL  localhost:3306  dbbook  SQL > select * from emp
                                   -> where age>1 and salary>1;
+-----+-------+-----+--------+
| eid | ename | age | salary |
+-----+-------+-----+--------+
|   1 | a     |   2 |      5 |
+-----+-------+-----+--------+
1 row in set (0.0004 sec)
```

EXAMPLE

```
MySQL  localhost:3306  dbbook  SQL > select * from emp
                                   -> where age>1 and
                                   -> (salary>1 or salary is null);
+-----+-------+-----+--------+
| eid | ename | age | salary |
+-----+-------+-----+--------+
|   1 | a     |   2 |      5 |
|   2 | b     |   3 |   NULL |
+-----+-------+-----+--------+
2 rows in set (0.0004 sec)
```

Null Values

- The arithmetic operations +, -, *, and return *null* if one of their arguments is *null*.
- COUNT(*) handles *null* values just like other values.
- All the other aggregate operations (COUNT, SUM, AVG, MIN, MAX, and DISTINCT) simply discard *null* values.

Joins

emp

eid	ename	age	salary
1	a	2	5
2	b	3	NULL
3	c	NULL	6
4	d	NULL	NULL

new_table

	id	age	book
▸	1	2	x
	2	4	NULL
	3	NULL	y
	4	NULL	z

Inner Join

```
1 • SELECT *
2   FROM emp e, new_table n
3   where e.age=n.age
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

eid	ename	age	salary	id	age	book
1	a	2	5	1	2	x

Inner Join

```
1  SELECT *
2  FROM emp e join new_table n
3  ON e.age=n.age
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

eid	ename	age	salary	id	age	book
1	a	2	5	1	2	x

Left Join

```
1 • SELECT *
2   FROM emp e left join new_table n
3   ON e.age=n.age
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

eid	ename	age	salary	id	age	book
1	a	2	5	1	2	x
2	b	3	NULL	NULL	NULL	NULL
3	c	NULL	6	NULL	NULL	NULL
4	d	NULL	NULL	NULL	NULL	NULL

Right Join

```
1 • SELECT *
2   FROM emp e right join new_table n
3   ON e.age=n.age
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

eid	ename	age	salary	id	age	book
1	a	2	5	1	2	x
NULL	NULL	NULL	NULL	2	4	NULL
NULL	NULL	NULL	NULL	3	NULL	y
NULL	NULL	NULL	NULL	4	NULL	z

Outer Join

```
1  SELECT *
2  FROM emp e left join new_table n
3  ON e.age=n.age
4  union
5  SELECT *
6  FROM emp e right join new_table n
7  ON e.age=n.age
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

eid	ename	age	salary	id	age	book
1	a	2	5	1	2	x
2	b	3	NULL	NULL	NULL	NULL
3	c	NULL	6	NULL	NULL	NULL
4	d	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	2	4	NULL
NULL	NULL	NULL	NULL	3	NULL	y
NULL	NULL	NULL	NULL	4	NULL	z

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 than relational algebra.
- 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.

Summary

- NULL for unknown field values brings many complications.
- Embedded SQL allows execution within a host language; cursor mechanism allows retrieval of one record at a time.
- SQL allows specification of rich integrity constraints.
- Triggers respond to changes in the database.