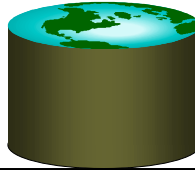


SQL: The Query Language Part II

15-415, Spring 2003, Lecture 12
R & G Chapter 5

The important thing is not to stop
questioning.

Albert Einstein



Example Instances

sid	bid	day
22	101	10/10/96
95	103	11/12/96

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Queries With GROUP BY

- To generate values for a column based on groups of rows, use **aggregate** functions in SELECT statements with the GROUP BY clause

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

The *target-list* contains (i) list of column names & (ii) terms with aggregate operations (e.g., MIN (S.age)).
- column name list (i) can contain only attributes from the *grouping-list*.

Group By Examples

For each rating, find the average age of the sailors

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

For each rating find the age of the youngest sailor with age ≥ 18

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

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*.
- One answer tuple is generated per qualifying group.

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

Answer Table

rating	age
1	33.0
7	35.0
8	55.0
10	35.0

3. Perform Aggregation

- Form cross product
- Delete unneeded columns, rows; form groups



Find sailors who've reserved all boats.

- Can you do this using Group By and Having?

```
SELECT S.name
FROM Sailors S, reserves R
WHERE S.sid = R.sid
GROUP BY S.name, S.sid
HAVING
COUNT(DISTINCT R.bid) =
  (Select COUNT (*) FROM Boats)
```

Note: must have both sid and name in the GROUP BY clause. Why?



```
SELECT S.name, S.sid
FROM Sailors S, reserves R
WHERE S.sid = r.sid
GROUP BY S.name, S.sid
HAVING
COUNT(DISTINCT R.bid) =
  Select COUNT (*) FROM Boats
```

s.name	s.sid	r.sid	r.bid
Dustin	22	22	101
Lubber	31	22	101
Bob	95	22	101
Dustin	22	95	102
Lubber	31	95	102
Bob	95	95	102

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Count (*) from boats = 4

s.name	s.sid	bcount
Dustin	22	1
Bob	95	1

Apply having clause to groups

s.name	s.sid



Sorting the Results of a Query

- ORDER BY *column* [ASC | DESC] [, ...]
 - SELECT S.rating, S.sname, S.age
 FROM Sailors S, Boats B, Reserves R
 WHERE S.sid=R.sid
 AND R.bid=B.bid AND B.color='red'
 ORDER BY S.rating, S.sname;
- Extra reporting power obtained by combining with aggregation.

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



INSERT

```
INSERT [INTO] table_name [(column_list)]
VALUES (value_list)
```

```
INSERT [INTO] table_name [(column_list)]
<select statement>
```

```
INSERT INTO Boats VALUES ( 105, 'Clipper', 'purple')
INSERT INTO Boats (bid, color) VALUES (99, 'yellow')
```

You can also do a "bulk insert" of values from one table into another:

```
INSERT INTO TEMP(bid)
SELECT r.bid FROM Reserves R WHERE r.sid = 22;
(must be type compatible)
```



DELETE & UPDATE

```
DELETE [FROM] table_name
[WHERE qualification]
```

```
DELETE FROM Boats WHERE color = 'red'
```

```
DELETE FROM Boats b
WHERE b. bid =
  (SELECT r.bid FROM Reserves R WHERE r.sid = 22)
```

Can also modify tuples using UPDATE statement.

```
UPDATE Boats
SET Color = "green"
WHERE bid = 103;
```



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.
- 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 *3-valued logic* (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.



Joins

```
SELECT (column_list)
FROM table_name
[INNER | {LEFT | RIGHT | FULL} OUTER] JOIN table_name
ON qualification_list
WHERE ...
```

Explicit join semantics needed unless it is an INNER join (INNER is default)



Inner Join

Only the rows that match the search conditions are returned.

```
SELECT s.sid, s.name, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid
```

Returns only those sailors who have reserved boats
SQL-92 also allows:

```
SELECT s.sid, s.name, r.bid
FROM Sailors s NATURAL JOIN Reserves r
```

“NATURAL” means equi-join for each pair of attributes with the same name (may need to rename with “AS”)



```
SELECT s.sid, s.name, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid
```

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45.0	22	101	10/10/96
31	Lubber	8	55.5	95	103	11/12/96
95	Bob	3	63.5			

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103



Left Outer Join

Left Outer Join returns all matched rows, plus all unmatched rows from the table on the left of the join clause
(use nulls in fields of non-matching tuples)

```
SELECT s.sid, s.name, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```

Returns all sailors & information on whether they have reserved boats



```
SELECT s.sid, s.name, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45.0	22	101	10/10/96
31	Lubber	8	55.5	95	103	11/12/96
95	Bob	3	63.5			

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103
31	Lubber	



Right Outer Join

Right Outer Join returns all matched rows, plus all unmatched rows from the table on the right of the join clause

```
SELECT r.sid, b.bid, b.name
FROM Reserves r RIGHT OUTER JOIN Boats b
ON r.bid = b.bid
```

Returns all boats & information on which ones are reserved.



```
SELECT r.sid, b.bid, b.name
FROM Reserves r RIGHT OUTER JOIN Boats b
ON r.bid = b.bid
```

sid	bid	day	bid	bname	color
22	101	10/10/96	101	Interlake	blue
95	103	11/12/96	102	Interlake	red
			103	Clipper	green
			104	Marine	red

r.sid	b.bid	b.name
22	101	Interlake
	102	Interlake
95	103	Clipper
	104	Marine



Full Outer Join

Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
SELECT r.sid, b.bid, b.name
FROM Reserves r FULL OUTER JOIN Boats b
ON r.bid = b.bid
```

Returns all boats & all information on reservations



```
SELECT r.sid, b.bid, b.name
FROM Reserves r FULL OUTER JOIN Boats b
ON r.bid = b.bid
```

sid	bid	day	bid	bname	color
22	101	10/10/96	101	Interlake	blue
95	103	11/12/96	102	Interlake	red
			103	Clipper	green
			104	Marine	red

r.sid	b.bid	b.name
22	101	Interlake
	102	Interlake
95	103	Clipper
	104	Marine

Note: in this case it is the same as the ROJ because bid is a foreign key in reserves, so all reservations must have a corresponding tuple in boats.



DDL – Create Table

- **CREATE TABLE** *table_name* (*{ column_name data_type [DEFAULT default_expr] [column_constraint [...]] | table_constraint } [...]*)
- **Data Types (PostgreSQL) include:**
 character(n) – fixed-length character string
 character varying(n) – variable-length character string
 smallint, integer, bigint, numeric, real, double precision
 date, time, timestamp, ...
 serial - unique ID for indexing and cross reference
 ...
- PostgreSQL also allows OIDs, arrays, inheritance, rules...
 conformance to the SQL-1999 standard is variable so we won't use these in the project.



Create Table (w/column constraints)

- **CREATE TABLE** *table_name* (*{ column_name data_type [DEFAULT default_expr] [column_constraint [...]] | table_constraint } [...]*)

Column Constraints:

- [CONSTRAINT *constraint_name*]
 { NOT NULL | NULL | UNIQUE | PRIMARY KEY |
 CHECK (*expression*) |
 REFERENCES *reftable* [(*refcolumn*)] [ON DELETE
action] [ON UPDATE *action*] }

action is one of:

- NO ACTION, CASCADE, SET NULL, SET DEFAULT
- expression* for column constraint must produce a boolean result and reference that column's value only.



Create Table (w/table constraints)

- **CREATE TABLE** *table_name* (*{ column_name data_type [DEFAULT default_expr] [column_constraint [...]] | table_constraint } [...]*)

Table Constraints:

- [CONSTRAINT *constraint_name*]
 { UNIQUE (*column_name* [, ...]) |
 PRIMARY KEY (*column_name* [, ...]) |
 CHECK (*expression*) |
 FOREIGN KEY (*column_name* [, ...]) REFERENCES
reftable [(*refcolumn* [, ...])] [ON DELETE *action*]
 [ON UPDATE *action*] }

Here, *expressions, keys, etc* can include multiple columns



Create Table (Examples)

```

CREATE TABLE films (
  code      CHAR(5) PRIMARY KEY,
  title     VARCHAR(40),
  did      DECIMAL(3),
  date_prod DATE,
  kind     VARCHAR(10),
  CONSTRAINT production UNIQUE(date_prod)
  FOREIGN KEY did REFERENCES distributors
  ON DELETE NO ACTION
);
CREATE TABLE distributors (
  did      DECIMAL(3) PRIMARY KEY,
  name     VARCHAR(40)
  CONSTRAINT con1 CHECK (did > 100 AND name <> '')
);

```



Views

```

CREATE VIEW view_name
AS select_statement

```

Makes development simpler
Often used for security
Not instantiated - makes updates tricky

```

CREATE VIEW Reds
AS SELECT B.bid, COUNT(*) AS scout
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
GROUP BY B.bid

```



Views Instead of Relations in Queries

```

CREATE VIEW Reds
AS SELECT B.bid, COUNT(*) AS scout
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
GROUP BY B.bid

```

bid	scount	Reds
102	1	

```

SELECT bname, scout
FROM Reds R, Boats B
WHERE R.bid=B.bid
AND scout < 10

```



Discretionary Access Control

```

GRANT privileges ON object TO users
[WITH GRANT OPTION]

```

- Object can be a **Table** or a **View**
- Privileges can be:
 - Select
 - Insert
 - Delete
 - References (cols) – allow to create a foreign key that references the specified column(s)
 - All
- Can later be **REVOKE**d
- Users can be single users or groups
- See Chapter 17 for more details.