The important thing is not to stop questioning.

Albert Einstein

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.

```sql
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

```sql
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

```sql
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.

Answer Table

1. Form cross product
2. Delete unneeded columns, rows; form groups
3. Perform Aggregation
Find the number of reservations for each red boat.

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

• Grouping over a join of two relations.

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

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

Queries With GROUP BY and HAVING

- Use the HAVING clause with the GROUP BY clause to restrict which group-rows are returned in the result set.

Conceptual Evaluation

- Form groups as before.
- The group-qualification is then applied to eliminate some groups.
  - Expressions in group-qualification must have a single value per group!
  - That is, attributes in group-qualification must be arguments of an aggregate op or must also appear in the grouping-list. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

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

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

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

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

Find sailors who've reserved all boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
SELECT B.bid, COUNT(*) AS numres
FROM Boats B, Reserves R
WHERE R.bid = B.bid
AND B.color = 'red'
GROUP BY B.bid
```

• Grouping over a join of two relations.

Find the number of reservations for each red boat.

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

Queries With GROUP BY and HAVING

- Use the HAVING clause with the GROUP BY clause to restrict which group-rows are returned in the result set.

Conceptual Evaluation

- Form groups as before.
- The group-qualification is then applied to eliminate some groups.
  - Expressions in group-qualification must have a single value per group!
  - That is, attributes in group-qualification must be arguments of an aggregate op or must also appear in the grouping-list. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

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

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

Find sailors who've reserved all boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
SELECT B.bid, COUNT(*) AS numres
FROM Boats B, Reserves R
WHERE R.bid = B.bid
AND B.color = 'red'
GROUP BY B.bid
```

• Grouping over a join of two relations.

Find the number of reservations for each red boat.

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

Queries With GROUP BY and HAVING

- Use the HAVING clause with the GROUP BY clause to restrict which group-rows are returned in the result set.

Conceptual Evaluation

- Form groups as before.
- The group-qualification is then applied to eliminate some groups.
  - Expressions in group-qualification must have a single value per group!
  - That is, attributes in group-qualification must be arguments of an aggregate op or must also appear in the grouping-list. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

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

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

Find sailors who've reserved all boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
SELECT B.bid, COUNT(*) AS numres
FROM Boats B, Reserves R
WHERE R.bid = B.bid
AND B.color = 'red'
GROUP BY B.bid
```

• Grouping over a join of two relations.
### Find sailors who have reserved all boats

- Can you do this using Group By and Having?

```sql
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?

### Insert

**INSERT**

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

**INSERT**

```sql
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)**

### 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.

### Delete & Update

**DELETE**

```sql
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**

```sql
SET Color = ”green”
WHERE bid = 103;
```

### Joins

**SELECT**

```sql
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.

```sql
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:

```sql
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.

**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)

```sql
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.

**Right Outer Join**

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

```sql
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.
**Full Outer Join**

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

```sql
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.
```

**Views**

```sql
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 scount
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
GROUP BY B.bid
```

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>95</td>
<td>102</td>
<td>10/11/96</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>11/12/96</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td></td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

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.