SQL: The Query Language

CS 186, Spring 2006,
Lectures 11&12
R &G - Chapter 5

Life is just a bowl of queries.

-Anon

Administrivia

- **Midterm1 was a bit easier than I wanted it to be.**
  - Mean was 80
  - Three people got 100(!)
  - I'm actually quite pleased.
  - But, I do plan to "kick it up a notch" for the future exams.
- **Be sure to register your name with your cs186 login if you haven't already --- else, you risk not getting grades.**
- **Homework 2 is being released today.**
  - Today and Tuesday's lectures provide background.
  - Hw 2 is due Tuesday 3/14
  - It's more involved than HW 1.

Relational Query Languages

- **A major strength of the relational model: supports simple, powerful *querying* of data.**
- **Two sublanguages:**
- **DDL – Data Defn Language**
  - define and modify schema (at all 3 levels)
- **DML – Data Manipulation Language**
  - Queries can be written intuitively.
- **The DBMS is responsible for efficient evaluation.**
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
  - Internal cost model drives use of indexes and choice of access paths and physical operators.

The SQL Query Language

- The most widely used relational query language.
- Originally IBM, then ANSI in 1986
- **Current standard is SQL-2003**
  - Introduced XML features, window functions, sequences, auto-generated IDs.
  - Not fully supported yet
- SQL-1999 Introduced "Object-Relational" concepts. Also not fully suppored yet.
- **SQL92 is a basic subset**
  - Most systems support a medium
- PostgreSQL has some "unique" aspects (as do most systems).

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 and other "system types", arrays, inheritance, rules…**
  conformance to the SQL-1999 standard is variable.

Constraints

- **Recall that the schema defines the legal instances of the relations.**
- **Data types are a way to limit the kind of data that can be stored in a table, but they are often insufficient.**
  - e.g., prices must be positive values
  - uniqueness, referential integrity, etc.
- **Can specify constraints on individual columns or on tables.**

Column constraints

[CONSTRAINT *constraint_name*]
{ NOT NULL | NULL | UNIQUE | PRIMARY KEY |

CHECK (*expression*) |

REFERENCES *reftable* [(*refcolumn*)] [ON DELETE *action*] [ON UPDATE *action*] }

primary key = unique + not null; also used as default target for references. (can have at most 1)

***expression* must produce a boolean result and reference that column's value only.**

***references* is for foreign keys; *action* is one of:**
NO ACTION, CASCADE, SET NULL, SET DEFAULT

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, etc can include multilple 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 <> ' ')
);
```

Other DDL Statements

- **Alter Table**
  - use to add/remove columns, constraints, rename things ...
- **Drop Table**
  - Compare to "Delete * From Table"
- **Create/Drop View**
- **Create/Drop Index**
- **Grant/Revoke privileges**
  - SQL has an authorization model for saying who can read/modify/delete etc. data and who can grant and revoke privileges!

The SQL DML

- Single-table queries are straightforward.
- To find all 18 year old students, we can write:

```
SELECT *
 FROM Students S
 WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:
  SELECT S.name, S.login

Querying Multiple Relations

- Can specify a join over two tables as follows:

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid AND E.grade='B'
```

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

result =

S.name	E.cid
Jones	History105

Note: obviously no referential integrity constraints have been used here.

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 tables in *relation-list*
- *qualification* : Comparisons combined using AND, OR and NOT.
  - Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of =≠<>≤≥
- *DISTINCT*: optional keyword indicating that the answer should not contain duplicates.
  - In SQL SELECT, the default is that duplicates are *not* eliminated! (Result is called a "multiset")

Query Semantics

- Semantics of an SQL query are defined in terms of the following conceptual evaluation strategy:
  1. do FROM clause: compute *cross-product* of tables (e.g., Students and Enrolled).
  2. do WHERE clause: Check conditions, discard tuples that fail. (i.e., "selection").
  3. do SELECT clause: Delete unwanted fields. (i.e., "projection").
  4. If DISTINCT specified, eliminate duplicate rows.

Probably the least efficient way to compute a query!

- An optimizer will find more efficient strategies to get the *same answer*.

Cross Product

SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

Step 2) Discard tuples that fail predicate

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'

Step 3) Discard Unwanted Columns

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'

Now the Details

We will use these instances of relations in our examples.

Reserves

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

Sailors

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

Boats

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

Example Schemas (in SQL DDL)

```
CREATE TABLE Sailors (sid INTEGER,
   sname CHAR(20),rating INTEGER,
   age REAL,
  PRIMARY KEY sid)

CREATE TABLE Boats (bid INTEGER,
   bname CHAR (20), color CHAR(10)
  PRIMARY KEY bid)

CREATE TABLE Reserves (sid INTEGER,
   bid INTEGER, day DATE,
  PRIMARY KEY (sid, bid, date),
  FOREIGN KEY sid REFERENCES Sailors,
  FOREIGN KEY bid REFERENCES Boats)
```

Another Join Query

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
        AND bid=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
95	Bob	3	63.5	22	101	10/10/96
95	Bob	3	63.5	95	103	11/12/96

Some Notes on Range Variables

- Can associate "range variables" with the tables in the FROM clause.
  - saves writing, makes queries easier to understand
- Needed when ambiguity could arise.
  - for example, if same table used multiple times in same FROM (called a "self-join")

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

Can be rewritten using range variables as:

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

More Notes

- Here's an example where range variables are required (self-join example):

```
SELECT  x.sname, x.age, y.sname, y.age
FROM Sailors x, Sailors y
WHERE  x.age > y.age
```

- Note that target list can be replaced by "*" if you don't want to do a projection:

```
SELECT  *
FROM Sailors x
WHERE  x.age > 20
```

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?

Expressions

- Can use arithmetic expressions in SELECT clause (plus other operations we'll discuss later)
- Use AS to provide column names

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname = 'dustin'
```

```
SELECT  S1.sname AS name1, S2.sname AS name2
FROM  Sailors S1, Sailors S2
WHERE  2*S1.rating = S2.rating - 1
```

String operations

- SQL also supports some string operations
- "LIKE" is used for string matching.

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

`_' stands for any one character and `%' stands for 0 or more arbitrary characters.

Find sid's of sailors who've reserved a red **or** 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).

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

Vs.

(note: UNION eliminates duplicates by default. Override w/ UNION ALL)

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

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

- If we simply replace OR by AND in the previous query, we get the wrong answer. (Why?)

```
SELECT R1.sid
FROM Boats B1, Reserves R1,
          Boats B2, Reserves R2
WHERE R1.sid=R2.sid
  AND R1.bid=B1.bid
  AND R2.bid=B2.bid
  AND (B1.color='red' AND B2.color='green')
```

AND Continued...

- **INTERSECT: discussed in book. Can be used to compute the intersection of any two *union-compatible* sets of tuples.**
- **Also in text: EXCEPT (sometimes called MINUS)**
- **Included in the SQL/92 standard, but many systems don't support them.**

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

- **Powerful feature of SQL: WHERE clause can itself contain an SQL query!**
  - Actually, so can FROM and HAVING clauses.

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)
```

- **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.*

Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

```
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.**
- **Can also specify NOT EXISTS**
- **If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103.**
  - UNIQUE checks for duplicate tuples in a subquery;
- **Subquery must be recomputed for each Sailors tuple.**
  - Think of subquery as a function call that runs a query!

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**
- **Find sailors whose rating is greater than that of some sailor called Horatio:**

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

Rewriting INTERSECT Queries Using IN

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

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

- **Similarly, EXCEPT queries re-written using NOT IN.**
- **How would you change this to find *names* (not *sid*'s) of Sailors who've reserved both red and green boats?**

Division in SQL

Find names of sailors who've reserved all boats.

- **Example in book, not using EXCEPT:**

```
SELECT S.sname          Sailors S such that ...
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
there is no boat B        FROM Boats B
                          WHERE NOT EXISTS (SELECT R.bid
that doesn't have ...                       FROM Reserves R
                                            WHERE R.bid=B.bid
    a Reserves tuple showing S reserved B       AND R.sid=S.sid))
```

Basic SQL Queries - Summary

- An advantage of the relational model is its well-defined query semantics.
- SQL provides functionality close to that of the basic relational model.
  - some differences in duplicate handling, null values, set operators, etc.
- Typically, many ways to write a query
  - the system is responsible for figuring a fast way to actually execute a query regardless of how it is written.
- Lots more functionality beyond these basic features.

Example Instances

Reserves

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

Sailors

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

Boats

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

Aggregate Operators

- **Significant extension of relational algebra.**

```
COUNT (*)
COUNT ( [DISTINCT] A)
SUM ( [DISTINCT] A)
AVG ( [DISTINCT] A)
MAX (A)
MIN (A)
```

single column

```
SELECT COUNT (*)
FROM Sailors S

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

SELECT COUNT (DISTINCT S.rating)
FROM Sailors S
WHERE S.sname='Bob'
```

Aggregate Operators (continued)

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

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

Find name and age of the oldest sailor(s)

- **The first query is incorrect!**

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

- **Third query equivalent to second query**
  - allowed in SQL/92 standard, but not supported in some systems.

```
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
```

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
```

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
```

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

1. Form cross product

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

2. Delete unneeded columns, rows; form groups

Answer Table

3. Perform Aggregation

rating	age
1	33.0
7	35.0
8	55.0
10	35.0

Find the number of reservations for each **red** 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
```

- Grouping over a join of two relations.

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

b.bid	b.color	r.bid
101	blue	101
102	red	101
103	green	101
104	red	101
101	blue	102
102	red	102
103	green	102
104	red	102

1

b.bid	b.color	r.bid
102	red	102

2

b.bid	scount
102	1

answer

Queries With GROUP BY and HAVING

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

- **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 ≥ 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	m-age	count
1	33.0	1
7	35.0	2
8	55.0	1
10	35.0	1

rating	
7	35.0

Answer relation

Find names of sailors who've reserved all boats.

- **Example in book, not using EXCEPT:**

```
SELECT  S.sname
FROM  Sailors S          Sailors S such that ...
WHERE  NOT EXISTS (SELECT  B.bid          there is no boat B without
                   FROM  Boats B          ...
                   WHERE  NOT EXISTS (SELECT  R.bid
  a Reserves tuple showing S reserved B      FROM  Reserves R
                                             WHERE  R.bid=B.bid
                                                AND R.sid=S.sid))
```

Find names of 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

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
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

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

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
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

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

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
22	101	10/10/96
95	103	11/12/96

bid	bname	color
101	Interlake	blue
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
22	101	10/10/96
95	103	11/12/96

bid	bname	color
101	Interlake	blue
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.

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

```
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
```

b.bid	scount
102	1

Reds