Life is just a bowl of queries.
-Anon (not Forrest Gump)

Relational Query Languages

- Two sublanguages:
  - DDL – Data Definition Language
    - Define and modify schema (at all 3 levels)
  - DML – Data Manipulation Language
    - Queries can be written intuitively.

- DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Optimizer can re-order operations, without affecting query answer.
  - Choices driven by "cost model"

The SQL Query Language

- The most widely used relational query language.
- Standardized
  (although most systems add their own "special sauce" -- including PostgreSQL)
- We will study SQL92 -- a basic subset

Example Database

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>1</td>
<td>Fred</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
</tr>
<tr>
<td>3</td>
<td>Nancy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserves</th>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

The 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)

The SQL DML

- Find all 18-year-old sailors:
  
  ```sql
  SELECT * FROM Sailors S WHERE S.age=18
  ```

- To find just names and ratings, replace the first line:
  
  ```sql
  SELECT S.sname, S.rating
  ```
Querying Multiple Relations

SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=102

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>sid</td>
<td>bid</td>
</tr>
<tr>
<td>1</td>
<td>Fred</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
</tr>
<tr>
<td>3</td>
<td>Nancy</td>
</tr>
</tbody>
</table>

Basic SQL Query

SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification

- relation-list: List of relation names
  - possibly with a range variable after each name
- target-list: List of attributes of tables in relation-list
- qualification: Comparisons combined using AND, OR and NOT.
- DISTINCT: optional keyword indicating that the answer should not contain duplicates.

Query Semantics

1. FROM: compute cross product of tables.
2. WHERE: Check conditions, discard tuples that fail.
3. SELECT: Delete unwanted fields.
4. DISTINCT (optional): eliminate duplicate rows.

Note: Probably the least efficient way to compute a query!
- Query optimizer will find more efficient ways to get the same answer.

About Range Variables

- Needed when ambiguity could arise.
  - e.g., same table used multiple times in FROM ("self-join")

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

Arithmetic Expressions

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 Comparisons

`_` 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

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

... or:

```sql
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

```sql
SELECT R.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')
```

Find sid's of sailors who have not reserved a boat

```sql
SELECT S.sid
FROM Sailors S
EXCEPT
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
```

- Could use a self-join:
Nested Queries: IN

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)

Names of sailors who’ve not reserved boat #103:

SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
FROM Reserves R
WHERE R.bid=103)

Nested Queries with Correlation

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)

Subquery must be recomputed for each Sailors tuple.
- Think of subquery as a function call that runs a query!
- Also: NOT EXISTS.

UNIQUE

Names of sailors who’ve reserved boat #103 exactly once:

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

More on Set-Comparison Operators

- we’ve seen: IN, EXISTS, UNIQUE
- can also have: NOT IN, NOT EXISTS, NOT UNIQUE
- other forms: 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')

A Tough One

Find sailors who’ve reserved all boats.

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

- Relational model has well-defined query semantics
- SQL provides functionality close to basic relational model (some differences in duplicate handling, null values, set operators, ...)
- Typically, many ways to write a query
  - DBMS figures out a fast way to execute a query, regardless of how it is written.