

SQL: The Query Language Part 1

R &G - Chapter 5

The important thing is not to
stop questioning.

Albert Einstein

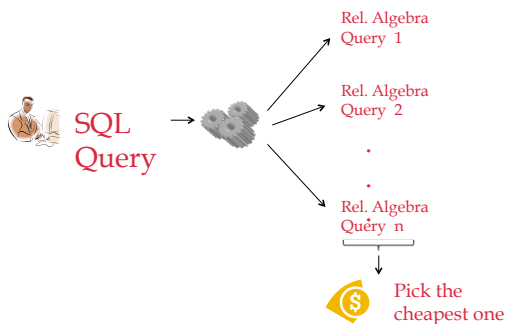


Review

- Relational Algebra (Operational Semantics)
 - Given a query, how to mix and match the relational algebra operators to answer it
 - Used for query optimization
- Relational Calculus (Declarative Semantics)
 - Given a query, what do I want my answer set to include?
- Algebra and safe calculus are simple and powerful models for query languages for relational model
 - Have same expressive power
- SQL can express every query that is expressible in relational algebra/calculus. (and more)



Relational Query Languages



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
 - Won't affect 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

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Boats

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

Reserves

sid	bid	day
1	102	9/12
2	102	9/13



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, day),
  FOREIGN KEY sid REFERENCES Sailors,
  FOREIGN KEY bid REFERENCES Boats);
```

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

sid	bid	day
1	102	9/12
2	102	9/13



The SQL DML

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

- Find all 18-year-old sailors:

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

- To find just names and ratings, replace the first line:

```
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
```

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13



Basic SQL Query

DISTINCT: optional. Answer should not contain duplicates.

SQL default: duplicates are *not* eliminated! (Result a "multiset")

target-list: List of expressions over attributes of tables in *relation-list*

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

qualification: Comparisons combined using AND, OR and NOT. Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of =, <, >, <=, >=, etc.

relation-list: List of relation names, possibly with a *range-variable* after each name



Query Semantics

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

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*.



Find sailors who've reserved at least one boat

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

- Would DISTINCT make a difference here?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?
 - Would DISTINCT make a diff to this variant of the query?



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
```

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27



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

```
SELECT S.sname
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

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

... OR:

```
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

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



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

```
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'
```



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

- Could use a self-join:

```
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')
```



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

```
SELECT S.sid
FROM   Sailors S

EXCEPT

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



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)
```



Nested Queries: NOT IN

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

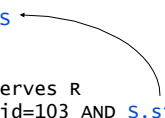
```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid NOT IN
       (SELECT R.sid
        FROM   Reserves R
        WHERE  R.bid=103)
```



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.**



More on Set-Comparison Operators

- we've seen: **IN, EXISTS**
- can also have: **NOT IN, NOT EXISTS**
- 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')
```



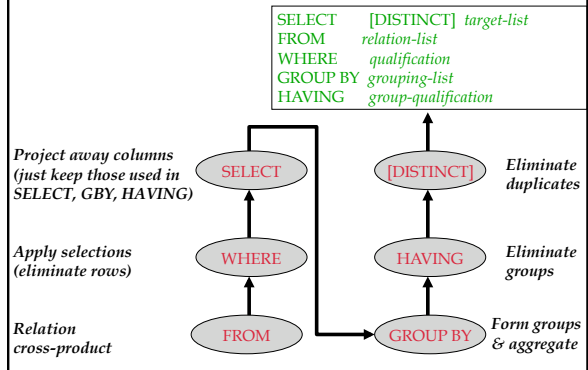
A Tough One

Find sailors who've reserved all boats.

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



Conceptual SQL Evaluation



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;
```

- **Can order by any column in SELECT list, including expressions or aggs:**

```
SELECT S.sid, COUNT(*) AS redrescnt
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 redrescnt DESC;
```



Null Values

- Field values 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 rows that match the qualification 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



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

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

<u>sid</u>	<u>bid</u>	<u>day</u>
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

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
```

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

<u>sid</u>	<u>bid</u>	<u>day</u>
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
```

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

<u>bid</u>	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!
bid is a foreign key in reserves, so all reservations must have a corresponding tuple in boats.

 **Views: Defining External DB Schemas**

CREATE VIEW *view_name*
AS *select_statement*

Makes development simpler
Often used for security
Not “materialized”

```
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	scout
102	1


Reds

```
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 **REVOKEd**
- Users can be single users or groups
- See Chapter 17 for more details.

 **Two more important topics**

- **Constraints**
- **SQL embedded in other languages**

 **Integrity Constraints (Review)**

- **An IC describes conditions that every *legal instance* of a relation must satisfy.**
 - Inserts/deletes/updates that violate IC's are disallowed.
 - Can ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- **Types of IC's: Domain constraints, primary key constraints, foreign key constraints, general constraints.**
 - *Domain constraints*: Field values must be of right type. Always enforced.
 - *Primary key and foreign key constraints*: you know them.



General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Checked on insert or update.
- Constraints can be named.

```
CREATE TABLE Sailors
(sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK (rating >= 1
AND rating <= 10))
```

```
CREATE TABLE Reserves
(sname CHAR(10),
bid*INTEGER,
day DATE,
PRIMARY KEY (bid,day),
CONSTRAINT noInterlakeRes
CHECK ('Interlake' <>
(SELECT B.bname
FROM Boats B
WHERE B.bid=bid)))
```



Constraints Over Multiple Relations

- Awkward and wrong!
- Only checks sailors!
- Only required to hold if the associated table is non-empty.

```
CREATE TABLE Sailors
(sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK
((SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM
Boats B) < 100))
```

Number of boats
plus number of
sailors is < 100

- ASSERTION is the right solution; not associated with either table.
- Unfortunately, not supported in many DBMS.
- Triggers are another solution.

```
CREATE ASSERTION smallClub
CHECK
((SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid)
FROM Boats B) < 100))
```



Writing Applications with SQL

- SQL is not a general purpose programming language.

- + Tailored for data retrieval and manipulation
- + Relatively easy to optimize and parallelize
- Can't write entire apps in SQL alone

Options:

Make the query language "Turing complete"

Avoids the "impedance mismatch"

but, loses advantages of relational language simplicity

Allow SQL to be embedded in regular programming languages.

Q: What needs to be solved to make the latter approach work?



Embedded SQL

- DBMS vendors traditionally provided "host language bindings"
 - E.g. for C or COBOL
 - Allow SQL statements to be called from within a program
 - Typically you preprocess your programs
 - Preprocessor generates calls to a proprietary DB connectivity library
- General pattern
 - One call to *connect* to the right database (login, etc.)
 - SQL statements can refer to *host variables* from the language
- Typically vendor-specific
 - We won't look at any in detail, we'll look at standard stuff
- Problem
 - SQL relations are (multi)-sets, no *a priori* bound on the number of records. No such data structure in C.
 - SQL supports a mechanism called a *cursor* to handle this.



Just to give you a flavor

```
EXEC SQL SELECT S.sname, S.age
INTO :c_sname,:c_age
FROM Sailors S
WHERE S.sid = :c_sid
```



Cursors

- Can declare a cursor on a relation or query
- Can *open* a cursor
- Can repeatedly *fetch* a tuple (moving the cursor)
- Special return value when all tuples have been retrieved.
- ORDER BY allows control over the order tuples are returned.
 - Fields in ORDER BY clause must also appear in SELECT clause.
- LIMIT controls the number of rows returned (good fit w/ORDER BY)
- Can also modify/delete tuple pointed to by a cursor
 - A "non-relational" way to get a handle to a particular tuple
- There's an Embedded SQL syntax for cursors
 - DECLARE <cursorname> CURSOR FOR <select stmt>
 - FETCH FROM <cursorname> INTO <variable names>
 - But we'll peek at JDBC instead

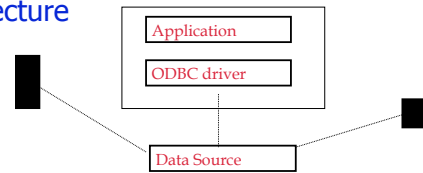


Database APIs: Alternative to embedding

- **Rather than modify compiler, add a library with database calls (API)**
 - special objects/methods
 - passes SQL strings from language, presents **result sets** in a language-friendly way
 - *ODBC* a C/C++ standard started on Windows
 - *JDBC* a Java equivalent
 - Most scripting languages have similar things
 - E.g. For Perl there is DBI, "oraPerl", other packages
- **Mostly DBMS-neutral**
 - at least try to hide distinctions across different DBMSs



Architecture



- A lookup service maps "data source names" ("DSNs") to drivers
 - Typically handled by OS
- Based on the DSN used, a "driver" is linked into the app at runtime
- The driver traps calls, translates them into DBMS-specific code
- Database can be across a network
- ODBC is standard, so the same program can be used (in principle) to access multiple database systems
- Data source may not even be an SQL database!



ODBC/JDBC

- **Various vendors provide drivers**
 - MS bundles a bunch into Windows
 - Vendors like DataDirect and OpenLink sell drivers for multiple OSes
- **Drivers for various data sources**
 - Relational DBMSs (Oracle, DB2, SQL Server, etc.)
 - "Desktop" DBMSs (Access, dBase, Paradox, FoxPro, etc.)
 - Spreadsheets (MS Excel, Lotus 1-2-3, etc.)
 - Delimited text files (.CSV, .TXT, etc.)
- **You can use JDBC/ODBC clients over many data sources**
 - E.g. MS Query comes with many versions of MS Office (msqry32.exe)
- **Can write your own Java or C++ programs against xDBC**



JDBC

- **Part of Java, easy to use**
- **Java comes with a JDBC-to-ODBC bridge**
 - So JDBC code can talk to any ODBC data source
 - E.g. look in your Windows Control Panel or MacOS Utilities folder for JDBC/ODBC drivers!
- **JDBC tutorial online**
 - <http://developer.java.sun.com/developer/Books/JDBCTutorial/>



Ruby on Rails

- **Rails' find method gives a simple rowset interface**
 - Just an array of records
 - Unfortunately slurps entire result set into memory.
- **Rails' ORM (Object Relational Mapping) goes beyond queries and cursors**
 - Data modeling and implicit query construction
 - The ActiveRecord.find method sometimes generates key/foreign-key joins, for example
- **Can also do:**
 - find_by_sql
 - ActiveRecord::Base.connection.execute(



API Summary

APIs are needed to interface DBMSs to programming languages

- Embedded SQL uses "native drivers" and is usually faster but less standard
- ODBC (used to be Microsoft-specific) for C/C++
- JDBC the standard for Java
- Scripting languages (PHP, Perl, JSP) are becoming the preferred technique for web-based systems



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.**