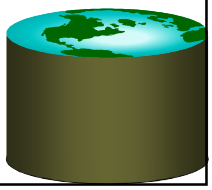
SQL: The Query Language Part 2

R & G - Chapter 5

The important thing is not to stop questioning.

Albert Einstein



Example Database

Sailors

sid	sname	rating	age
22	Dustin	7	45
31	Lubber	8	55
95	Bob	3	63

Boats

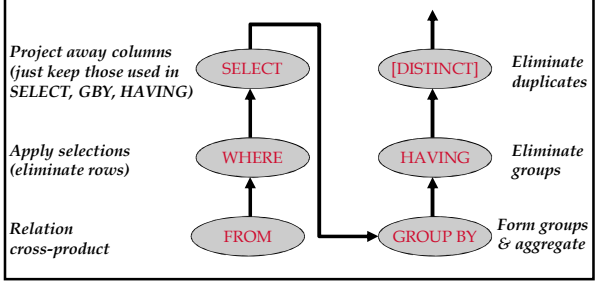
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Reserves

sid	bid	day
22	101	10/10/06
95	103	11/12/06

Conceptual SQL Evaluation

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



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 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

---

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

bid	scount
102	1

Reds

```
SELECT  bname, scount
   FROM Reds R, Boats B
   WHERE  R.bid=B.bid
      AND scount < 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 be used to 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.
- ASSERTION is the right solution; not associated with either table.
- Unfortunately, not supported in many DBMS.
- *Triggers* are another solution.

```
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

```
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
```

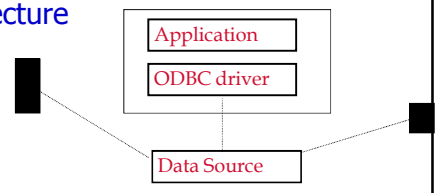
Cursors

- **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 in which tuples are returned.**
  - Fields in ORDER BY clause must also appear in SELECT clause.
- **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 *client*s 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, very 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