SQL: The Query Language (Part 1)

Fall 2016

Life is just a bowl of queries.

-Anon

The SQL Query Language

- The most widely used relational query language.
- Originally IBM, then ANSI in 1986
- Current standard is SQL-2008
 - 2003 was last major update: XML, window functions, sequences, auto-generated IDs.
 - Not fully supported yet
- SQL-1999 Introduced "Object-Relational" concepts.
 - · Also not fully supported yet.
- SQL92 is a basic subset
 - Most systems support at least this
- PostgreSQL has some "unique" aspects (as do most systems).
- SQL is not synonymous with Microsoft's "SQL Server"

Relational Query Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.
- Query languages can be divided into two parts
 - Data Manipulation Language (DML)
 - Allows for gueries and updates
 - Data Definition Language (DDL)
 - Define and modify schema (at all 3 levels)
 - Permits database tables to be created or deleted. It also defines indexes (keys), specifies links between tables, and imposes constraints between tables
- 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 DML

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

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

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

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

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
- <u>target-list</u>: 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 $= \neq <> \leq \geq$
- <u>DISTINCT</u>: optional keyword indicating that the answer should not contain duplicates.
 - In SQL SELECT, the default is that duplicates are <u>not</u> eliminated! (Result is called a "multiset")

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 | С |
| 53831 | Reggae203 | В |
| 53650 | Topology112 | Α |
| 53666 | History105 | В |

| | sid | name | login | age | gpa |
|---|-------|-------|----------|-----|-----|
| Ī | 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.

Query Semantics

- Semantics of an SQL query are defined in terms of the following conceptual evaluation strategy:
 - 1. do FROM clause: compute <u>cross-product</u> 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.

Step 1 – Cross Product

| 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 | В |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53650 | Topology112 | Α |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53666 | History105 | В |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53831 | Carnatic101 | C |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53831 | Reggae203 | В |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53650 | Topology112 | A |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53666 | History105 | В |

| sid | 1 | cid | grade |
|------|--------|-----------|-------|
| 5383 | 31 Ca | rnatic101 | C |
| 5383 | Re Re | ggae203 | В |
| 5365 | | pology112 | Α |
| 5366 | 66 His | story105 | В |

| sic | d | name | login | age | gpa |
|-----|----|-------|----------|-----|-----|
| 536 | 66 | Jones | jones@cs | 18 | 3.4 |
| 536 | 88 | Smith | smith@ee | 18 | 3.2 |

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 | С |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53832 | Reggae203 | В |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53650 | Topology112 | A |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53666 | History105 | В |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53831 | Carnatic 101 | C |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53831 | Reggae203 | В |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53650 | Topology112 | A |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53666 | History 105 | В |
| | | | | | | | |

SELECT S.name, E.cid FROM Students S, Enrolled E WHERE S.sid=E.sid AND E.grade='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 | Carnatic 101 | 5 |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53832 | Reggae203 | (B) |
| 53666 | Jones | jones@cs | 18 | 3.4 | 53650 | Topology112 | Ă |
| (53666) | Jones | jones@cs | 18 | 3.4 | 53666 | History105 | (B) |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53831 | Carnatic 101 | |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53831 | Reggae203 | (B) |
| 53688 | Smith | smith@ee | 18 | 3.2 | 53650 | Topology112 | Ă |
| 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'

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 <u>null</u> 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 <u>3-valued logic</u> (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.

Null Values – 3 Valued Logic

(null > 0) is null (null + 1) is null (null = 0)is null null AND true is null

| AND | T | F | Null |
|------|------|---|------|
| Т | Т | F | Null |
| F | F | F | F |
| NULL | Null | F | Null |

| OR | T | F | Null |
|------|---|------|------|
| Т | Т | Т | Т |
| F | Т | F | Null |
| NULL | Т | Null | Null |

Example Schemas (in SQL DDL)

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

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

Consider the use Of VARCHAR instead

Now the Details

We will use these instances of relations in our Sailors examples.

Reserves

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

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

Boats

| <u>bid</u> | bname | color |
|------------|-----------|-------|
| 101 | Interlake | blue |
| 102 | Interlake | red |
| 103 | Clipper | green |
| 104 | Marine | red |

Example Schemas (in SQL DDL)

| <u>bid</u> | bname | color |
|------------|-----------|-------|
| 101 | Interlake | blue |
| 102 | Interlake | red |
| 103 | Clipper | green |
| 104 | Marine | red |

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

Example Schemas (in SQL DDL)

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

CREATE TABLE Reserves (sid INTEGER, bid INTEGER, day DATE,

PRIMARY KEY (sid, bid, day),

FOREIGN KEY sid REFERENCES Sailors.

FOREIGN KEY bid REFERENCES Boats)

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

SELECT S.sname FROM Sailors S. Reserves R range variables as: | WHERE S.sid=R.sid AND bid=103

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 |

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?

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'
```

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'
```

• Can also have expressions in WHERE clause:

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

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

```
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 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'
UNION ALL)
```

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

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

 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')
```

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 <u>nested loops</u> evaluation: For each Sailors tuple, check the qualification by computing the subquery.

AND Continued...

• INTERSECT:

 Can be used to compute the intersection of any two unioncompatible sets of tuples.

EXCEPT

- (sometimes called MINUS)
- Included in the SQL/92 standard, but many systems (including MySQL) don't support them.

```
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 with Correlation

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

```
SELECT S.sname
FROM Sailors S is nonempty
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 subguery as a function call that runs a guery!

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')
```

What does this query return?

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
FROM Boats B
WHERE B.color = "Green"
AND
NOT EXISTS (SELECT R.bid
FROM Reserves R
WHERE R.bid=B.bid
AND R.sid=S.sid))
```

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.

```
SELECT S.sname Sailors S such that ...

FROM Sailors S

WHERE NOT EXISTS (SELECT B.bid
there is no boat B

WHERE NOT EXISTS (SELECT R.bid
that doesn't have ...

FROM Reserves R

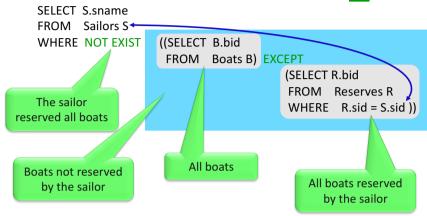
WHERE R.bid=B.bid
a Reserves tuple showing S reserved B

AND R.sid=S.sid))
```

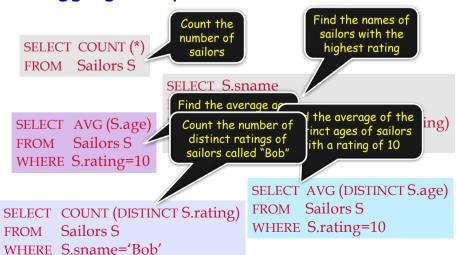
Recall Exists Tests whether the set is nonempty

Division Operations in SQL (1)

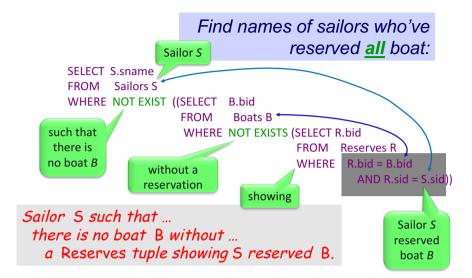
Find names of sailors who've reserved **all** boat:



Aggregate Operators



Division Operations in SQL (2)



More about this next time!

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.