CS150A Database

Wenjie Wang
School of Information Science and Technology
ShanghaiTech University
Sept. 23, 2024

Today:
- Introduction to SQL
- DDL & DML
  - DML in a Single Table
  - DML in Multiple Tables

Readings:
- Database Management Systems (DBMS), Chapter 5
- Lecture note SQL I
- Lecture note SQL II

Acknowledgement: Joe Hellerstein@UCBerkeley's scourse notes

SQL Roots

- Developed @IBM Research in the 1970s
  - System R project
  - Vs. Berkeley’s Quel language (Ingres project)
- Commercialized/Popularized in the 1980s
  - "Intergalactic Dataspeak”
  - IBM beaten to market by a startup called Oracle

SQL’s Persistence

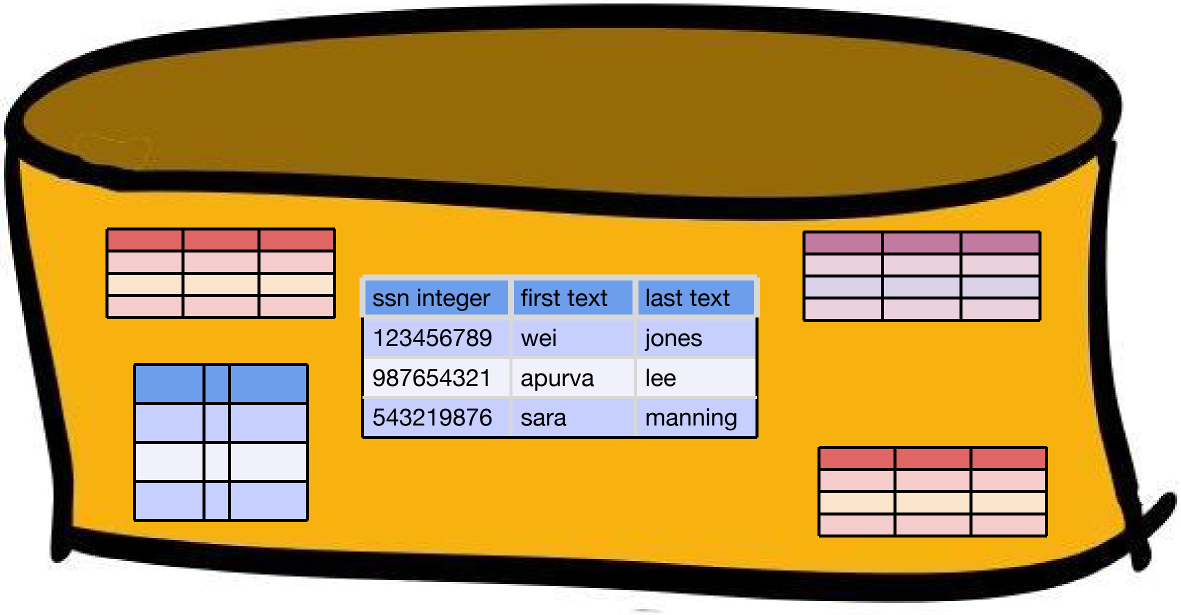
- Over 40 years old!
- Questioned repeatedly
  - 90’s: Object-Oriented DBMS (OQL, etc.)
  - 2000’s: XML (Xquery, Xpath, XSLT)
  - 2010’s: NoSQL & MapReduce
- SQL keeps re-emerging as the standard
  - Even Hadoop, Spark etc. mostly used via SQL
  - May not be perfect, but it is useful

SQL Pros and Cons

- Declarative!
  - Say *what* you want, not *how* to get it
- Implemented widely
  - With varying levels of efficiency, completeness
- Constrained
  - Not targeted at Turing-complete tasks
- General-purpose and feature-rich
  - many years of added features
  - extensible: callouts to other languages, data sources

Relational Terminology

- ***Database***: Set of named Relations



ssn integer	first text	last text
123456789	wei	jones
987654321	apurva	lee
543219876	sara	manning

Relational Terminology, Pt 2.

- ***Database***: Set of named Relations
- ***Relation*** (*Table*):
  - ***Schema:*** description ("metadata")
  - ***Instance:*** set of data satisfying the schema

ssn integer	first text	last text
123456789	wei	jones
987654321	apurva	lee
543219876	sara	manning

Relational Terminology, Pt. 3

- ***Database***: Set of named Relations
- ***Relation*** (*Table*):
  - ***Schema:*** description (“metadata”)
  - ***Instance:*** set of data satisfying the schema
- ***Attribute*** (*Column, Field*)

first text
wei
apurva
sara

Relational Terminology, Pt. 4

- ***Database***: Set of named Relations
- ***Relation*** (Table):
  - ***Schema***: description (“metadata”)
  - ***Instance***: set of data satisfying the schema
- ***Attribute*** (Column, Field)
- ***Tuple*** (Record, Row)

543219876	sara	manning

Relational Tables

- *Schema* is fixed:
  - unique attribute names, *atomic* types
  - folks (ssn integer, first text, last text)
- *Instance* can change often
  - a *multi*set of “rows” (“tuples”)

{(123456789, ’wei’, ’jones’),
(987654321, ’apurva’, ’lee’),
(543219876, ‘sara’, ‘manning’),
(987654321, ’apurva’, ’lee’)}

Quick Check 1

- Is this a relation?

num integer	street text	zip integer	
84	Maple Ave	54704	
22	High	Street	76425
75	Hearst Ave	94720	

Quick Check 2

- Is this a relation?

num integer	street text	num integer
84	Maple Ave	54704
22	High Street	76425
75	Hearst Ave	94720

Quick Check 3

- Is this a relation?

first text	last text	addr address
wei	jones	(84, ‘Maple’, 54704)
apurva	lee	(22, ‘High’, 76425)
sara	manning	(75, ‘Hearst’, 94720)

SQL Language

- Two sublanguages:
  - DDL – Data Definition Language
    - Define and modify schema
  - DML – Data Manipulation Language
    - Queries can be written intuitively.
- RDBMS responsible for efficient evaluation.
  - Choose and run algorithms for declarative queries
    - Choice of algorithm must not affect query answer.

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	9/12/2015
2	102	9/13/2015

The SQL DDL: Sailors

```
CREATE TABLE Sailors (
        sid INTEGER,
        sname CHAR(20),
        rating INTEGER,
        age FLOAT
```

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

The SQL DDL: Sailors, Pt. 2

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

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

The SQL DDL: Primary Keys

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

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

- Primary Key column(s)
  - Provides a unique “lookup key” for the relation
  - Cannot have any duplicate values
  - Can be made up of >1 column
    - E.g. (firstname, lastname)

The SQL DDL: Boats

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

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

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

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

The SQL DDL: Reserves

```
CREATE TABLE Sailors (
        sid INTEGER,
        sname CHAR(20),
        rating INTEGER,
        age FLOAT,
        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);
```

<u>sid</u>	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

<u>bid</u>	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

<u>sid</u>	<u>bid</u>	<u>day</u>
1	102	9/12
2	102	9/13

The SQL DDL: Reserves Pt. 2

```
CREATE TABLE Sailors (
        sid INTEGER,
        sname CHAR(20),
        rating INTEGER,
        age FLOAT,
        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,
```

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 DDL: Foreign Keys

```
CREATE TABLE Sailors (
        sid INTEGER,
        sname CHAR(20),
        rating INTEGER,
        age FLOAT,
        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 DDL: Foreign Keys Pt. 2

- Foreign key references a table
  - Via the primary key of that table
- Need not share the name of the referenced primary key

```
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

- Find all 27-year-old sailors:

```
SELECT *
FROM  Sailors AS S
WHERE S.age=27;
```

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

```
SELECT S.sname, S.rating
```

Sailors

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

Basic Single-Table Queries

- **SELECT** [*DISTINCT*] *<column expression list>*
  FROM *<single table>*
  [**WHERE** *<predicate>*]

- Simplest version is straightforward
  - Produce all tuples in the table that satisfy the predicate
  - Output the expressions in the SELECT list
  - Expression can be a column reference, or an arithmetic expression over column refs

SELECT DISTINCT

SELECT DISTINCT S.name, S.gpa
FROM students S
WHERE S.dept = 'CS'

- DISTINCT specifies removal of duplicate rows before output
- Can refer to the students table as “S”, this is called an alias

ORDER BY

- **SELECT** S.name, S.gpa, S.age*2 AS a2
  FROM Students S
  WHERE S.dept = 'CS'
  ORDER BY S.gpa, S.name, a2;

- ORDER BY clause specifies output to be sorted
  - ***Lexicographic*** ordering
- Obviously must refer to columns in the output
  - Note the AS clause for naming output columns!

ORDER BY, Pt. 2

- **SELECT** S.name, S.gpa, S.age*2 AS a2
  FROM Students S
  WHERE S.dept = 'CS'
  ORDER BY S.gpa DESC, S.name **ASC**, a2;

- Ascending order by default, but can be overridden
  - DESC flag for descending, ASC for ascending
  - Can mix and match, lexicographically

LIMIT

- ```
  SELECT  S.name, S.gpa, S.age*2 AS a2
  FROM Students S
  WHERE S.dept = 'CS'
  ORDER BY S.gpa DESC, S.name ASC, a2;
  LIMIT 3 ;
  ```

- Only produces the first <integer> output rows
- Typically used with ORDER BY
  - Otherwise the output is *non-deterministic*
  - Not a “pure” declarative construct in that case – output set depends on algorithm for query processing
  ```

Aggregates

- **SELECT** [DISTINCT] **AVG**(S.gpa)
  FROM Students S
  WHERE S.dept = 'CS'

- Before producing output, compute a summary (a.k.a. an *aggregate*) of some arithmetic expression
- Produces 1 row of output
  - with one column in this case
- Other aggregates: SUM, COUNT, MAX, MIN

GROUP BY

SELECT [DISTINCT] **AVG**(S.gpa), S.dept
FROM Students S
GROUP BY S.dept

- Partition table into groups with same GROUP BY column values
  - Can group by a list of columns
- Produce an aggregate result per group
  - Cardinality of output = # of distinct group values
- Note: can put grouping columns in SELECT list

HAVING

SELECT [DISTINCT] **AVG**(S.gpa), S.dept
FROM Students S
GROUP BY S.dept
HAVING COUNT(*) > 2

- The HAVING predicate filters groups
- HAVING is applied *after* grouping and aggregation
  - Hence can contain anything that could go in the SELECT list
  - I.e. aggs or GROUP BY columns
- HAVING can only be used in aggregate queries
- It’s an optional clause

Putting it all together

```
SELECT S.dept, AVG(S.gpa), COUNT(*)
FROM Students S
WHERE S.gender = 'F'
GROUP BY S.dept
HAVING COUNT(*) >= 2
ORDER BY S.dept;
```

name	dept	gpa	gender
Alice	CS	3.5	F
Bob	CS	3.7	M
Carol	CS	3.8	F
Dave	Math	3.6	M
Eve	Math	3.9	F
Grace	Math	3.8	F
Heidi	Physics	3.4	F
Ivan	Physics	3.5	M
Judy	Physics	3.9	F

DISTINCT Aggregates

Are these the same or different?

```
SELECT COUNT(DISTINCT S.name)
FROM Students S
WHERE S.dept = 'CS';
```

```
SELECT DISTINCT COUNT(S.name)
FROM Students S
WHERE S.dept = 'CS';
```

What Is This Asking For?

```
SELECT S.name, AVG(S.gpa)
FROM Students S
GROUP BY S.dept;
```

name	dept	gpa
Alice	CS	3.5
Bob	CS	3.7
Carol	Math	3.8
Dave	Math	3.6
Eve	CS	3.9

What Is This Asking For?

SELECT S.name, **AVG**(S.gpa)
FROM Students S
GROUP BY S.dept;

When using GROUP BY, any non-aggregated columns in the SELECT statement (in this case, S.name) must either appear in the GROUP BY clause or be aggregated.

```
SELECT S.dept, AVG(S.gpa)
FROM Students S
GROUPBY S.dept;
```

```
SELECT S.name, AVG(S.gpa)
FROM Students S
GROUPBY S.name;
```

```
SELECT  AVG(S.gpa)
FROM Students S
GROUPBY S.dept, S.name;
```

```
SELECT S.dept, S.name, AVG(S.gpa)
FROM Students S
GROUPBY S.dept, S.name;
```

SQL DML:
General Single-Table Queries

- **SELECT [DISTINCT]** *<column expression list>*
  FROM *<single table>*
  [**WHERE** *<predicate>*]
  [**GROUP BY** *<column list>*
  [**HAVING** *<predicate>*]]
  [**ORDER BY** *<column list>*]
  [**LIMIT** <integer>];

Summary

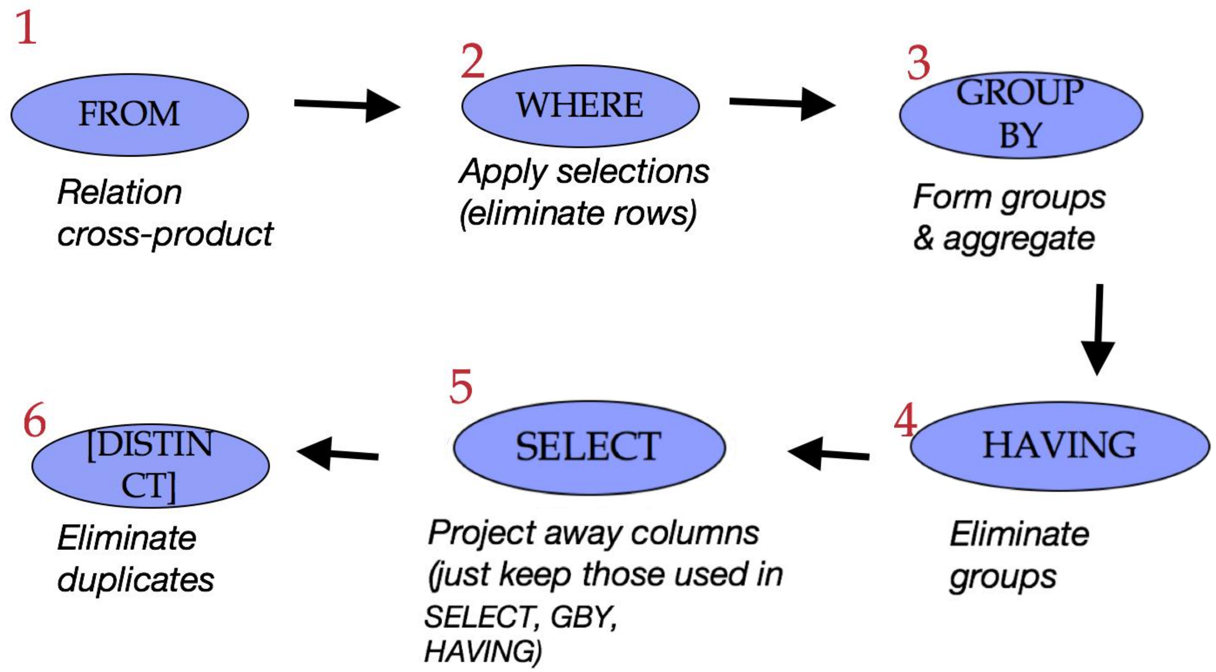
- Relational model has **well-defined query semantics**
- Modern SQL extends “pure” relational model

  (some extra goodies for duplicate row, non-atomic types… more in next lecture)
- Typically, many ways to write a query
  - DBMS figures out a fast way to execute a query, regardless of how it is written.

SQL DML 1:
Basic Single-Table Queries

- **SELECT** [**DISTINCT**] *<column expression list>*
  FROM *<single table>*
  [**WHERE** *<predicate>*]
  [**GROUP BY** *<column list>*
  [**HAVING** *<predicate>*]]
  [**ORDER BY** *<column list>*]
  [**LIMIT** <integer>];

Conceptual SQL Evaluation

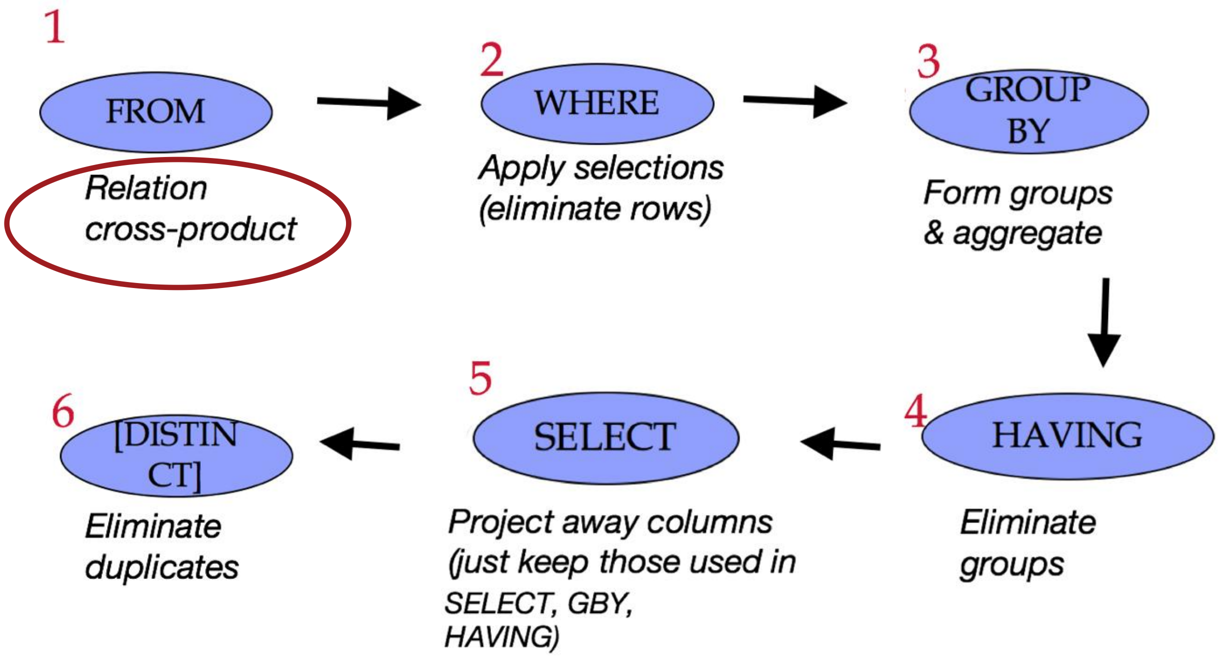


SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*
HAVING *group-qualificati*

Join Queries

- SELECT [DISTINCT] *<column expression list>*
  FROM *<table1 [AS t1], ... , tableN [AS tn]>*
  [WHERE *<predicate>*]
  [GROUP BY *<column list>*[HAVING *<predicate>*]]
  [ORDER BY *<column list>*];

Conceptual SQL Evaluation, cont



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

Cross (Cartesian) Product

- All pairs of tuples, concatenated

Sailors

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	rating	age	sid	bid	day
1	Popeye	10	22	1	102	9/12
1	Popeye	10	22	2	102	9/13
1	Popeye	10	22	1	101	10/01
2	OliveOyl	11	39	1	102	9/12
…	…	…	…	…	…	…

Find sailors who've reserved a boat

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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	rating	age	sid	bid	day
1	Popeye	10	22	1	102	9/12
1	Popeye	10	22	2	102	9/13
1	Popeye	10	22	1	101	10/01
2	OliveOyl	11	39	1	102	9/12
…	…	…	…	…	…	…

Find sailors who’ve reserved a boat cont

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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	bid
1	Popeye	102
1	Popeye	101
2	OliveOyl	102

Column Names and Table Aliases

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

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

More Aliases

```
SELECT x.sname, x.age,
       y.sname AS sname2,
       y.age AS age2
FROM Sailors AS x, Sailors AS y
WHERE x.age > y.age
```

sname	age
Popeye	22
OliveOyl	39
Garfield	27
Bob	19

More Aliases

sname	age
Popeye	22
OliveOyl	39
Garfield	27
Bob	19

```
SELECT x.sname, x.age,
      y.sname AS sname2,
      y.age AS age2
FROM Sailors AS x, Sailors AS y
WHERE x.age > y.age
```

sname	age	sname2	age2
Popeye	22	Bob	19
OliveOyl	39	Popeye	22
OliveOyl	39	Garfield	27
OliveOyl	39	Bob	19
Garfield	27	Popeye	22
Garfield	27	Bob	19

- Table aliases in the FROM clause
  - Needed when the same table used multiple times ("self-join")
- Column aliases in the SELECT clause

Arithmetic Expressions

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

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

SQL Calculator!

```
SELECT
        log(1000) as three,
        exp(ln(2)) as two,
        cos(0) as one,
        ln(2*3) = ln(2) + ln(3) as sanity;
```

String Comparisons

- Old School SQL

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname LIKE 'B_%’
```

- Standard Regular Expressions

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname ~ 'B.*’
```

Combining Predicates

- Subtle connections between:
  - Boolean logic in WHERE (i.e., AND, OR)
  - Traditional Set operations (i.e. INTERSECT, UNION)
- Let’s see some examples…

Sid's of sailors who 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')
```

Sid’s of sailors who reserved a red **OR** a green boat Pt 2

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

VS…

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

UNION ALL

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

Sid’s of sailors who reserved a red **AND** a green boat Pt 3

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

VS…

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

INTERSECT

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

Find 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
```

Set Semantics

- Set: a collection of *distinct* elements
- Standard ways of manipulating/combining sets
  - Union
  - Intersect
  - Except
- Treat tuples within a relation as elements of a set

Default: Set Semantics

Note: R and S are relations. They are not sets, since they have duplicates.

R = {A, A, A, A, B, B, C, D}
S = {A, A, B, B, B, C, E}

- UNION

  {A, B, C, D, E}
- INTERSECT

  {A, B, C}
- EXCEPT

  {D}

Note: Think of each letter as being a **tuple** in **relation.**

ex:
A: (Jim, 18, English, 4.0)
B: (Marcela , 20, CS, 3.8)
C: (Gail, 19, Statistics, 3.74)
D: (Goddard, 20, Math, 3.8

“ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

"UNION ALL": Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- UNION ALL: sum of cardinalities

  {A(4+2), B(2+3), C(1+1), D(1+0), E(0+1)}

  = {A, A, A, A, A, A, B, B, B, B, B, C, C, D, E}

“INTERSECT ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- INTERSECT ALL: min of cardinalities

  {A(min(4,2)), B(min(2,3)), C(min(1,1)),
  D(min(1,0)), E(min(0,1))}
  = {A, A, B, B, C}

"EXCEPT ALL": Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- EXCEPT ALL: difference of cardinalities

  {A(4-2), B(2-3), C(1-1), D(1-0), E(0-1)}
  = {A, A, D, }

Conclusion

- **Relational Terminology**
  - Database
  - Relation:
    - Schema:
    - Instance:
  - Attribute

- **DDL**
  - CREATE TABLE
  - Primary Key
  - Foreign Key

- **DML in a Single Table**

  SELECT [DISTINCT] <column expression list>
  FROM <single table>
  [**WHERE** <predicate>]
  [**GROUP BY** <column list>
  [**HAVING** <predicate>]]
  [**ORDER BY** <column list>]
  [**LIMIT** <integer>];

- **DML in Multiple Tables**
  - Aliases
  - Boolean logic vs Traditional Set

Next Lecture

- DML in Multiple Tables
  - Set Operations
  - Nested Queries
  - Join