

SQL I

R & G Chapter 5



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

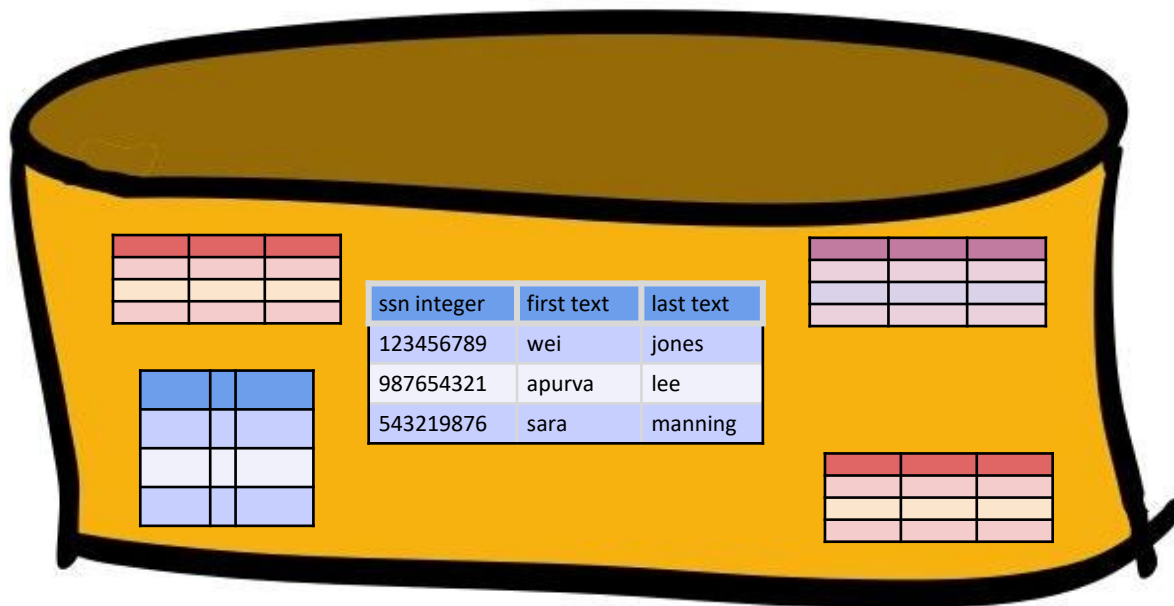


- Declarative!
 - Say *what* you want, not *how* to get it
- Implemented widely
 - With varying levels of efficiency, completeness
- Constrained
 - Not targeted at Turing-complete tasks
- General-purpose and feature-rich
 - many years of added features
 - extensible: callouts to other languages, data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

Relational Terminology



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



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 *multiset* of “rows” (“tuples”)

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

Quick Check 1

- Why is this not a relation?

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

76425

Quick Check 2

- Why is this not a relation?

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

Quick Check 3

- Why is this not 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

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

Boats

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

Reserves

<u>sid</u>	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
```

<u>sid</u>	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));
```

<u>sid</u>	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
```

<u>sid</u>	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
```

<u>sid</u>	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));
```

<u>bid</u>	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
```

<u>sid</u>	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));
```

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

```
CREATE TABLE Reserves (  
  sid INTEGER,  
  bid INTEGER,  
  day DATE,  
  PRIMARY KEY (sid, bid, day);
```

<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
```

<u>sid</u>	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));
```

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

```
CREATE TABLE Reserves (  
  sid INTEGER,  
  bid INTEGER,  
  day DATE,  
  PRIMARY KEY (sid, bid, day),  
  FOREIGN KEY (sid) REFERENCES Sailors,
```

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

<u>sid</u>	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));
```

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

```
CREATE TABLE Reserves (  
  sid INTEGER,  
  bid INTEGER,  
  day DATE,  
  PRIMARY KEY (sid, bid, day),  
  FOREIGN KEY (sid) REFERENCES Sailors,  
  FOREIGN KEY (bid) REFERENCES Boats);
```

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

<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 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

<u>sid</u>	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;
```


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;
```

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.