

Relational Model and SQL

Concepts

Syntax

Basic Queries

Alvin Cheung
Fall 2022

Reading: R & G Chapter 5



This Lecture

- The Relational Model
- SQL Basics



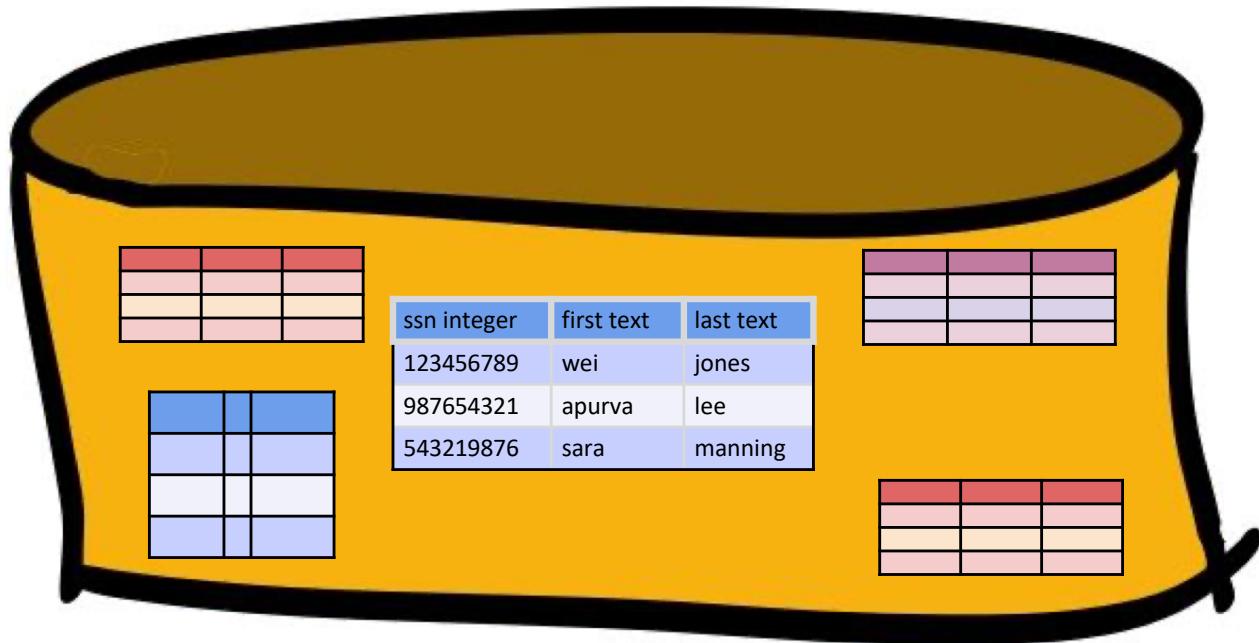
~~House~~ Zoom Rules



- Please turn on video if you feel comfortable
- Unmute for questions or comments
 - Raise hand / type in chat window works too

Relational Terminology

- **Database:** Set of named Relations



Relational Terminology, Pt 2.



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



Attribute

↓

first: text
wei
apurva
sara

Relational Terminology, Pt. 4



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

A diagram showing a tuple as a horizontal row of three cells. The first cell contains the value "543219876". The second cell contains the value "sara". The third cell contains the value "manning". A purple bracket underneath the first two cells spans from the start of the first cell to the end of the second cell. A purple arrow points from the word "Tuple" at the bottom right to the right edge of the third cell.

543219876	sara	manning
-----------	------	---------

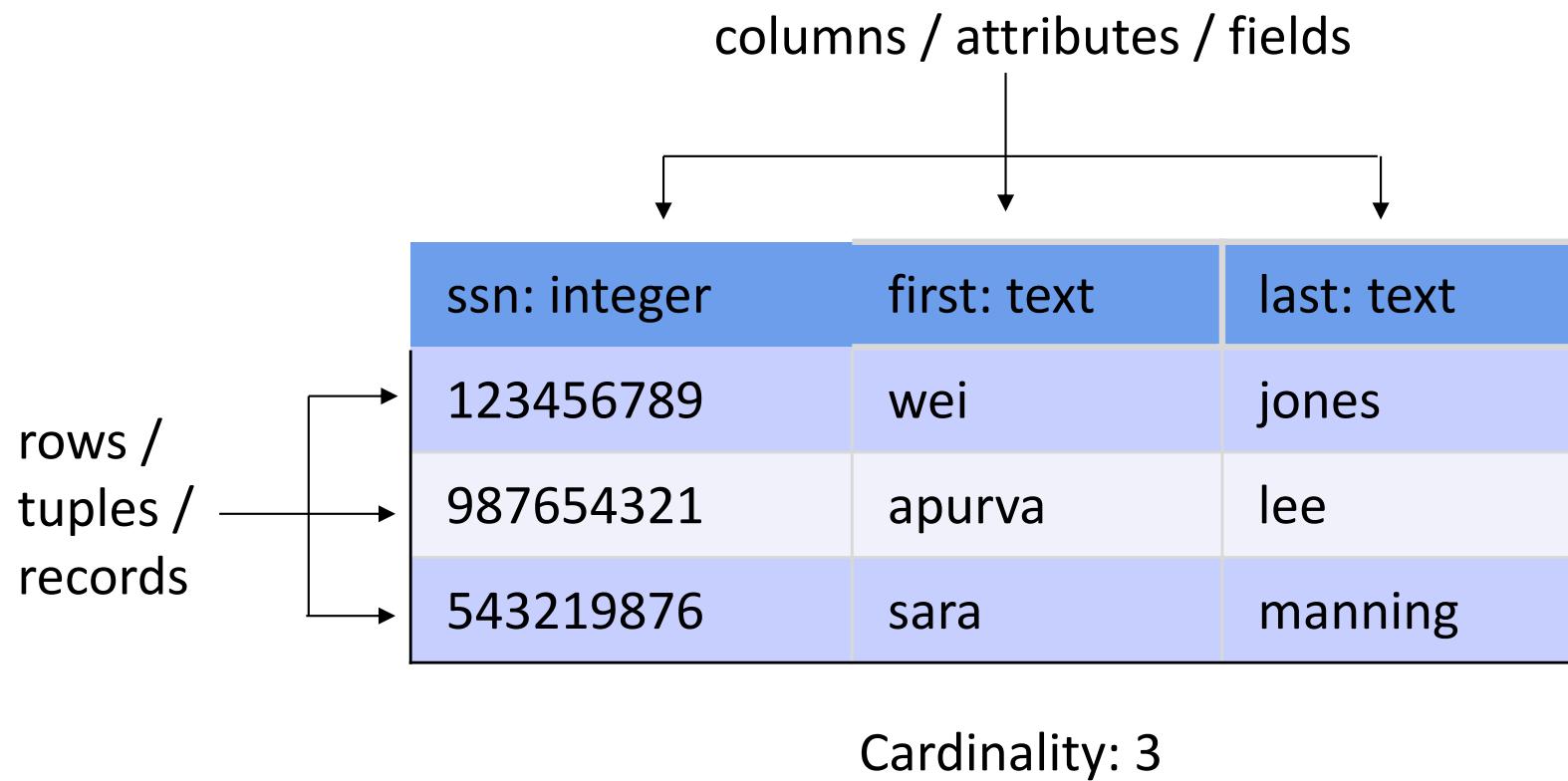
Tuple

Relational Terminology, Pt. 5



- **Database**: Set of named Relations
- **Relation** (aka *Table*):
 - **Schema**: description (“metadata”)
 - **Instance**: set of data satisfying the schema
- **Attribute** (aka *Column, Field*)
- **Tuple** (aka *Record, Row*)
- **Cardinality**:
 - # of tuples in a relation
entries

To summarize



Relational Tables



- Schema is fixed:
 - unique attribute names, atomic (aka primitive) types
- Tables are NOT ordered
 - they are sets or multisets (bags)
- Tables are FLAT
 - No nested attributes
- Tables DO NOT prescribe how they are implemented / stored on disk
 - This is called **physical data independence**

Table Implementation



- How would you store this on the disk?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Table Implementation



- How would you store this on the disk?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Row major: as an array of objects

GizmoWorks	Canon	Hitachi	HappyCam
USA	Japan	Japan	Canada
20000	50000	30000	500
True	True	True	False

Table Implementation



- How would you store this on the disk?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Column major: as one array per attribute

GizmoWorks	Canon	Hitachi	HappyCam
USA	Japan	Japan	Canada
20000	50000	30000	500
True	True	True	False

Table Implementation



- How would you store this on the disk?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Physical data independence

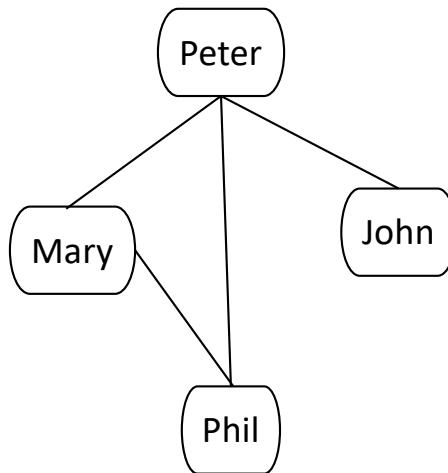
The logical definition of the data remains
unchanged, even when we make changes to the
actual implementation

row vs. col major

Relation is not the only data model



Example: storing FB friends



As a graph

OR

Person1: text	Person2: text	is_friend: int
Peter	John	1
John	Mary	0
Mary	Phil	1
Phil	Peter	1
...

As a relation

We will learn the tradeoffs of different
data models in the semester

Quick Check 1

- Why is this not a relation?

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

Schema fixed; does not conform
to integer

Quick Check 2

- Why is this not a relation?

Not unique attributes

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)



type must be primitive

First Normal Form



- All relations must be flat: we say that the relation is in *first normal form*

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

First Normal Form



- All relations must be flat: we say that the relation is in *first normal form*
- E.g., we want to add products manufactured by each company:

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

First Normal Form



- All relations must be flat: we say that the relation is in *first normal form*
- E.g., we want to add products manufactured by each company:

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

cname	country	no_employees	for_profit	products									
Canon	Japan	50000	Y	<table border="1"><thead><tr><th>pname</th><th>price</th><th>category</th></tr></thead><tbody><tr><td>SingleTouch</td><td>149.99</td><td>Photography</td></tr><tr><td>Gadget</td><td>200</td><td>Toy</td></tr></tbody></table>	pname	price	category	SingleTouch	149.99	Photography	Gadget	200	Toy
pname	price	category											
SingleTouch	149.99	Photography											
Gadget	200	Toy											
Hitachi	Japan	30000	Y	<table border="1"><thead><tr><th>pname</th><th>price</th><th>category</th></tr></thead><tbody><tr><td>AC</td><td>300</td><td>Appliance</td></tr></tbody></table>	pname	price	category	AC	300	Appliance			
pname	price	category											
AC	300	Appliance											

First Normal Form



- All relations must be flat: we say that the relation is in *first normal form*
- E.g., we want to add products manufactured by each company:

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

Non-1NF!

cname	country	no_employees	for_profit	products									
Canon	Japan	50000	Y	<table border="1"><thead><tr><th>pname</th><th>price</th><th>category</th></tr></thead><tbody><tr><td>SingleTouch</td><td>149.99</td><td>Photography</td></tr><tr><td>Gadget</td><td>200</td><td>Toy</td></tr></tbody></table>	pname	price	category	SingleTouch	149.99	Photography	Gadget	200	Toy
pname	price	category											
SingleTouch	149.99	Photography											
Gadget	200	Toy											
Hitachi	Japan	30000	Y	<table border="1"><thead><tr><th>pname</th><th>price</th><th>category</th></tr></thead><tbody><tr><td>AC</td><td>300</td><td>Appliance</td></tr></tbody></table>	pname	price	category	AC	300	Appliance			
pname	price	category											
AC	300	Appliance											

Use 2 tables!

First Normal Form



Now it's in 1NF

Company

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

Products

pname	price	category	manufacturer
SingleTouch	149.99	Photography	Canon
AC	300	Appliance	Hitachi
Gadget	200	Toy	Canon

SQL Roots



- Developed @IBM Research in the 1970s
 - System R project
 - Vs. Berkeley's Quel language **INGRES**
- Commercialized/Popularized in the 1980s
 - IBM started the db2 product line
 - IBM beaten to market by a startup called Oracle

SQL's Persistence



- Over 40 years old!
 - Not the only language for querying relations
- 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 for data computation and feature-rich
 - many years of added features
 - extensible: callouts to other languages, data sources

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)
```

SQL syntax

<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,
    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));
```

<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,
    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

```
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,
    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,
```

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

<u>bid</u>	<u>bname</u>	<u>color</u>
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: 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);
```

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

<u>bid</u>	<u>bname</u>	<u>color</u>
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: Foreign Keys Pt. 2



- Foreign key references a table
 - Via the primary key of that table
- Doesn't need to have the same name as 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

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

Basic Single-Table Queries

optional

- **SELECT** [*DISTINCT*] <*column expression list*>
 FROM <*single table*>
 [**WHERE** <*predicate*>]
- In this simple version:
 - 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



Distinct and Alias



```
SELECT DISTINCT S.name, S.gpa  
FROM students [AS] S  
WHERE S.dept = 'CS'
```

- Return all unique (name, GPA) pairs from students
- **DISTINCT** specifies removal of duplicate rows before output
- Can refer to the students table as **S**, this is called an **alias**

Ordering



- ```
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
  - Numeric ordering for “number-like” attributes (int, real, etc)
  - **Lexicographic** ordering otherwise (!!)(varchar, blob, etc)
- Obviously must refer to columns in the output
  - Note the AS clause for naming output columns!

# Ordering



- ```
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;
```

For tie-breaking S.gpa
- Ascending order by default, but can be overridden
 - DESC flag for descending, ASC for ascending
 - Can mix and match, lexicographically

Setting limits



- ```
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 (aka an *aggregate*) of some arithmetic expression
- Produces 1 row of output
 - with one column in this case
- Other aggregates: SUM, COUNT, MAX, MIN (and others)

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';
```



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: only grouping columns or aggregated values can appear in the SELECT list

Need to be Careful with GROUP BY...

```
SELECT product,  
       max(quantity)  
FROM   Purchase  
GROUP  BY product
```

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10



Need to be Careful with GROUP BY...



```
SELECT product,  
       max(quantity)  
FROM   Purchase  
GROUP  BY product
```

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

Product	Max(quantity)
Bagel	20
Banana	50

Need to be Careful with GROUP BY...



```
SELECT product,  
       max(quantity)  
FROM   Purchase  
GROUP  BY product
```

```
SELECT product, quantity  
FROM   Purchase  
GROUP  BY product  
-- what does this mean?
```

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

Product	Max(quantity)
Bagel	20
Banana	50

Need to be Careful with GROUP BY...



```
SELECT product,  
       max(quantity)  
FROM   Purchase  
GROUP  BY product
```

```
SELECT product, quantity  
FROM   Purchase  
GROUP  BY product  
-- what does this mean?
```

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

Product	Max(quantity)
Bagel	20
Banana	50

Product	Quantity
Bagel	20
Banana	??

Need to be Careful with GROUP BY...



```
SELECT product,  
       max(quantity)  
FROM   Purchase  
GROUP  BY product
```

```
SELECT product, quantity  
FROM   Purchase  
GROUP  BY product  
-- what does this mean?
```

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

Product	Max(quantity)
Bagel	20
Banana	50

Product	Quantity
Bagel	20
Banana	??



Need to be careful with GROUP BY...

Everything in SELECT must be either a GROUP-BY attribute, or an aggregate



```
SELECT product,  
       max(quantity)  
FROM Purchase  
GROUP BY product
```

```
SELECT product, quantity  
FROM Purchase  
GROUP BY product  
-- what does this mean?
```

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

Product	Max(quantity)
Bagel	20
Banana	50

Product	Quantity
Bagel	20
Banana	??



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

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



- Many query languages available for the relational data model
 - SQL is one of them that we will focus in this class
- Modern SQL extends set-based relational model
 - some extra goodies for duplicate row (bags), non-atomic types...
- Typically, many ways to write a query
 - DBMS figures out a fast way to execute a query, regardless of how it is written