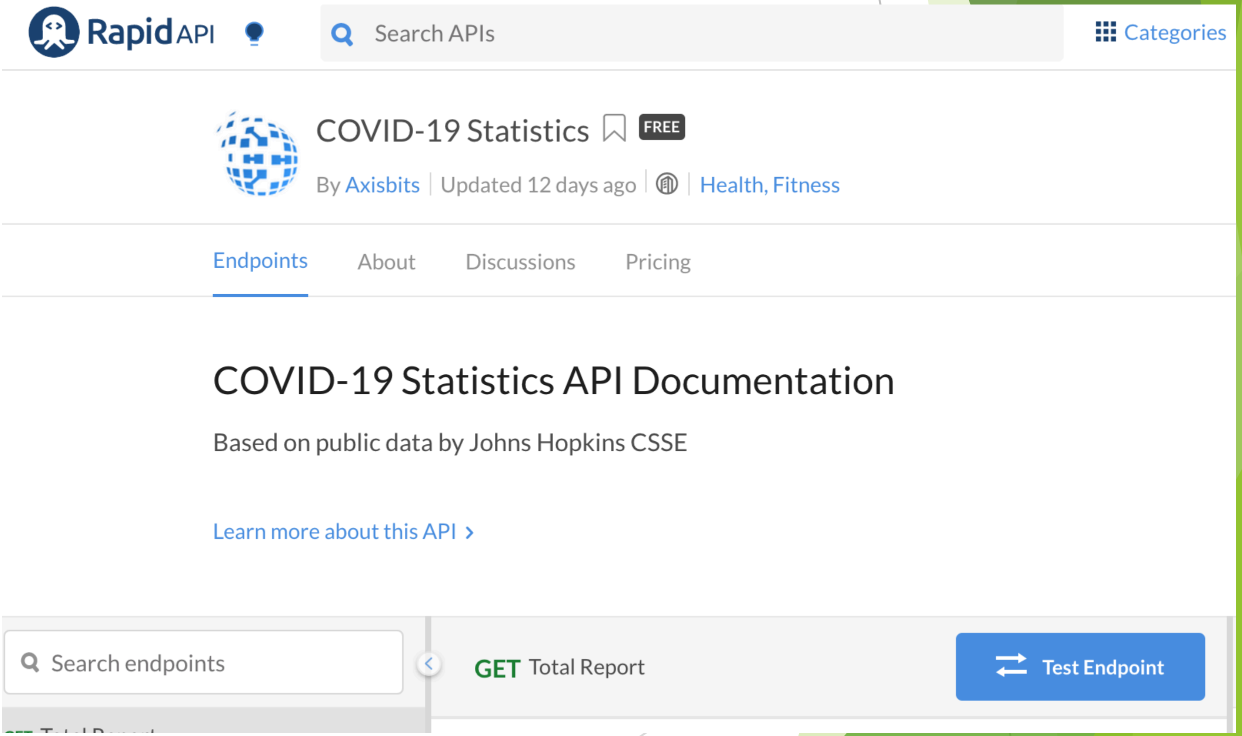
CS 217 Data Management and Information Processing

Structured Query Language

Last Week: JSON and Pandas

- We have learnt how to process data and extract useful information from semi-structured database using pandas.

- Why that's not sufficient?



COVID-19 Statistics

By Axisbits | Updated 12 days ago | Health, Fitness

COVID-19 Statistics API Documentation

Based on public data by Johns Hopkins CSSE

Learn more about this API ›

GET Total Report

Things You **Cannot** Do with Simple Tabular Data

- Model complex data relationships
  - Every row has a fixed number of attributes (columns)
  - Can’t model one-to-many and many-to-many relationships
  - You can try using multiple spreadsheet tabs or multiple matrices for different types of data, but linking them is difficult
- Enforce data integrity constraints
- Processing large volume of data efficiently
  - Pandas usually require loading all data to RAM
  - Not needed for SQL

Database Management Systems (DBMSs)

- A DBMS is a data management software that allows users to define databases, load them with data, and query them.
- Often run on a remote, multi-user server
  - Typically you need to know the hostname and have a username and password.
  - May be connected to one or more software applications or may stand alone.
- Client libraries exist for every common programming language
  - But you usually query the database using the SQL language

Why Use a Relational Database?

- **Scalability** – work with data larger than computer's RAM
- **Indexing** – efficiently sort & search along various dimensions *(don't be confused with index in pandas)*
- **Integrity** – restrict data type, ensure consistency across multiple tables
- **Deduplication** – save space, normalization
- **Concurrency** – multiple users or applications can query/update concurrently

Whether you know it or not,
you’re using a database everyday!

Basic Concepts in Databases

Table is the Main Concept in a Relational DB

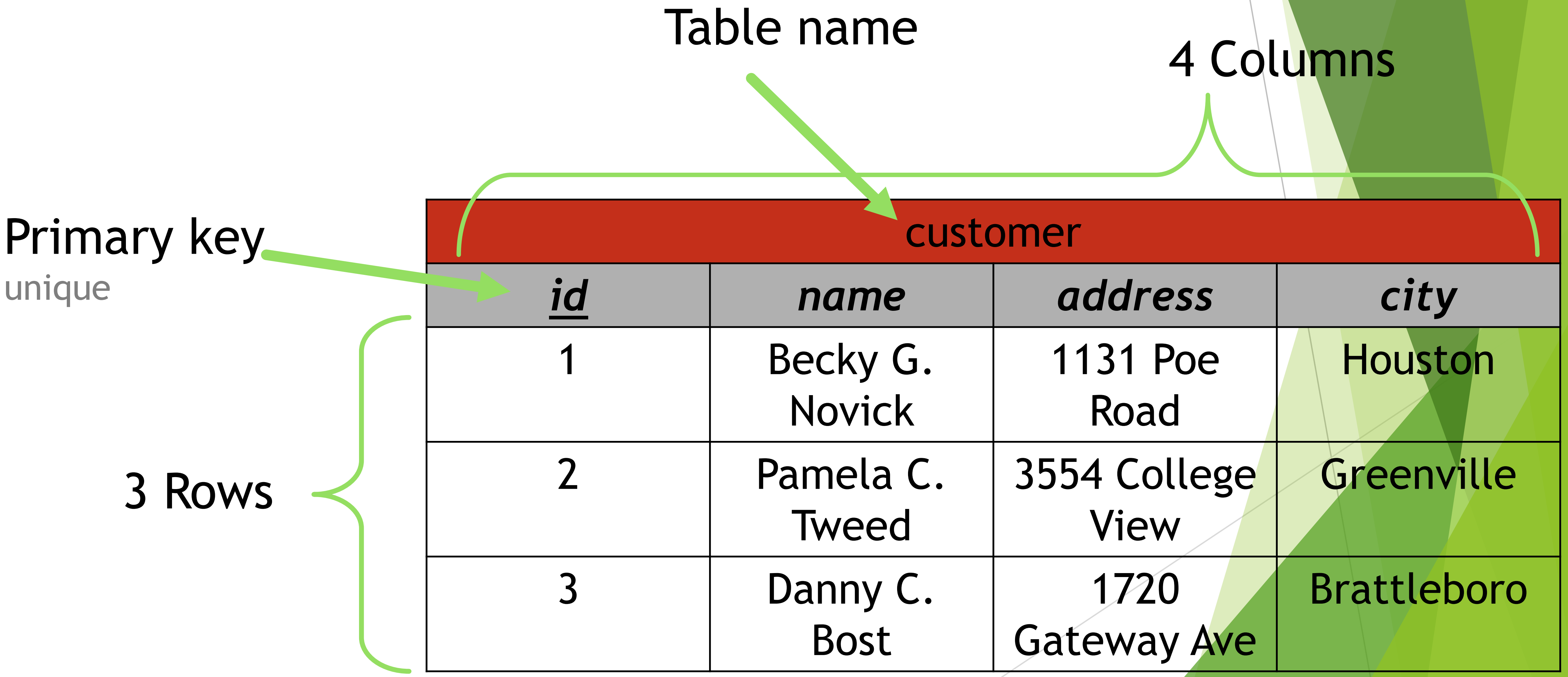


Table name

4 Columns

Primary key

unique

3 Rows

customer			
id	***name***	***address***	***city***
1	Becky G. Novick	1131 Poe Road	Houston
2	Pamela C. Tweed	3554 College View	Greenville
3	Danny C. Bost	1720 Gateway Ave	Brattleboro

DB Design Process Answers These Questions:

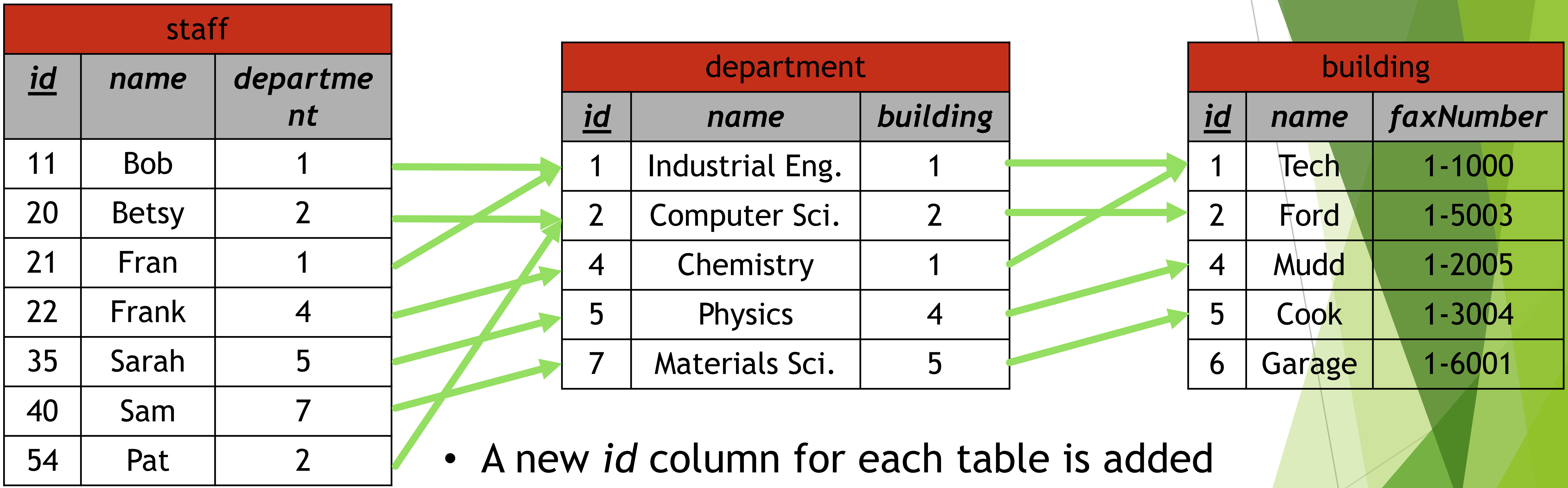
- What tables do we need?
  - How to logically separate the data?
- What columns?
  - Data types for columns?
  - How will rows be uniquely identified?
  - Are some columns optional?
- How will tables be linked?

customer			
id	***name***	***address***	***city***
1	Becky G. Novick	1131 Poe Road	Houston
2	Pamela C. Tweed	3554 College View	Greenville
3	Danny C. Bost	1720 Gateway Ave	Brattleboro

Sometimes We Start with One Redundant Table and Break it Down to Reflect the Logical Components

staff					
id	***name***	***department***	***building***	***room***	***faxNumber***
11	Bob	Industrial Eng.	Tech	100	1-1000
20	Betsy	Computer Sci.	Ford	100	1-5003
21	Fran	Industrial Eng.	Tech	101	1-1000
22	Frank	Chemistry	Tech	102	1-1000
35	Sarah	Physics	Mudd	200	1-2005
40	Sam	Materials Sci.	Cook	10	1-3004
54	Pat	Computer Sci.	Ford	102	1-5003

This is Called *Normalization*



staff		
id	***name***	***department***
11	Bob	1
20	Betsy	2
21	Fran	1
22	Frank	4
35	Sarah	5
40	Sam	7
54	Pat	2

department		
id	***name***	***building***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

building		
id	***name***	***faxNumber***
1	Tech	1-1000
2	Ford	1-5003
4	Mudd	1-2005
5	Cook	1-3004
6	Garage	1-6001

- A new *id* column for each table is added
- Removes redundancy
    - Save space
    - Edit values in one place, so duplicates don’t become inconsistent
- Tables can be populated separately

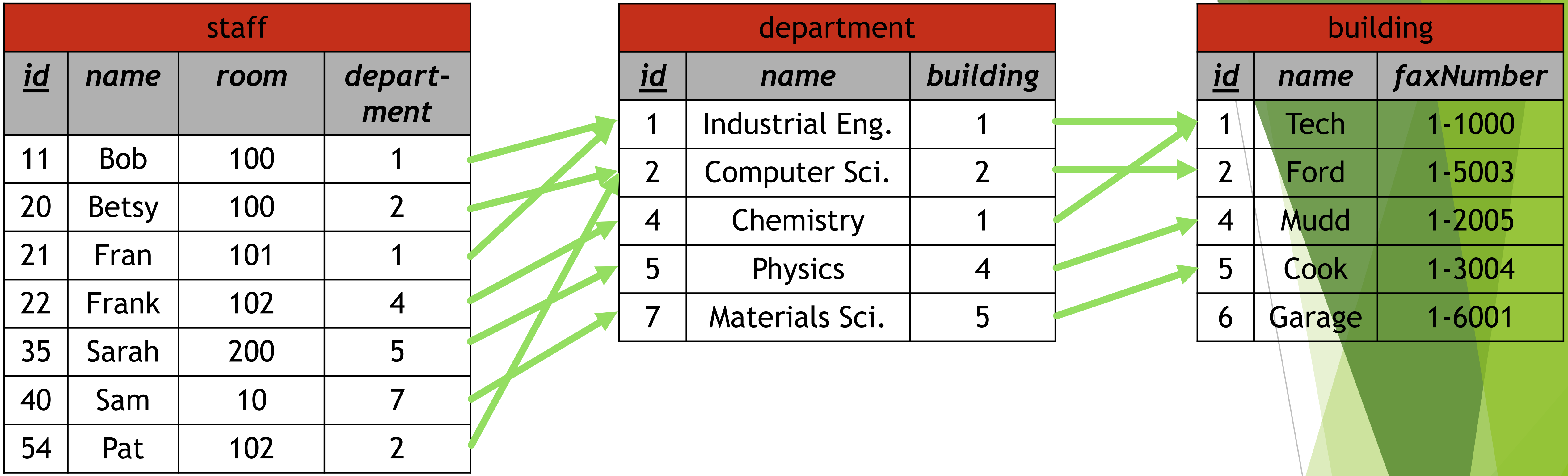
Tables

- Represent objects, events, or relationships
  - Each of its rows must be uniquely identifiable
  - Has attributes that the DB will store in columns
  - Can refer to rows in other tables
- *Objects*: people, places, or things
- *Events*: usually associated with a specific time. Can recur.
- *Relationships*: associations

Designing a set of tables is called *data modelling*, and it’s best learned by example.

Database Schema Defines Data’s Structure

- Also called a data model
- It’s *metadata* – data about data
- Defines the tables, including:
  - Columns in each table (both the name and *type*)
  - Primary Key for each table
  - Foreign Keys that link tables

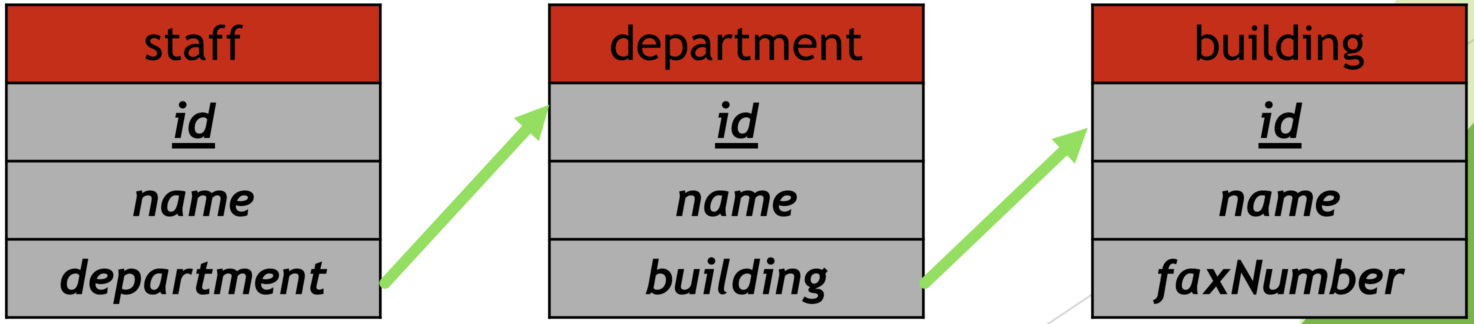


staff			
id	***name***	***room***	***department***
11	Bob	100	1
20	Betsy	100	2
21	Fran	101	1
22	Frank	102	4
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***building***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

building		
id	***name***	***faxNumber***
1	Tech	1-1000
2	Ford	1-5003
4	Mudd	1-2005
5	Cook	1-3004
6	Garage	1-6001

DB Design Diagram:

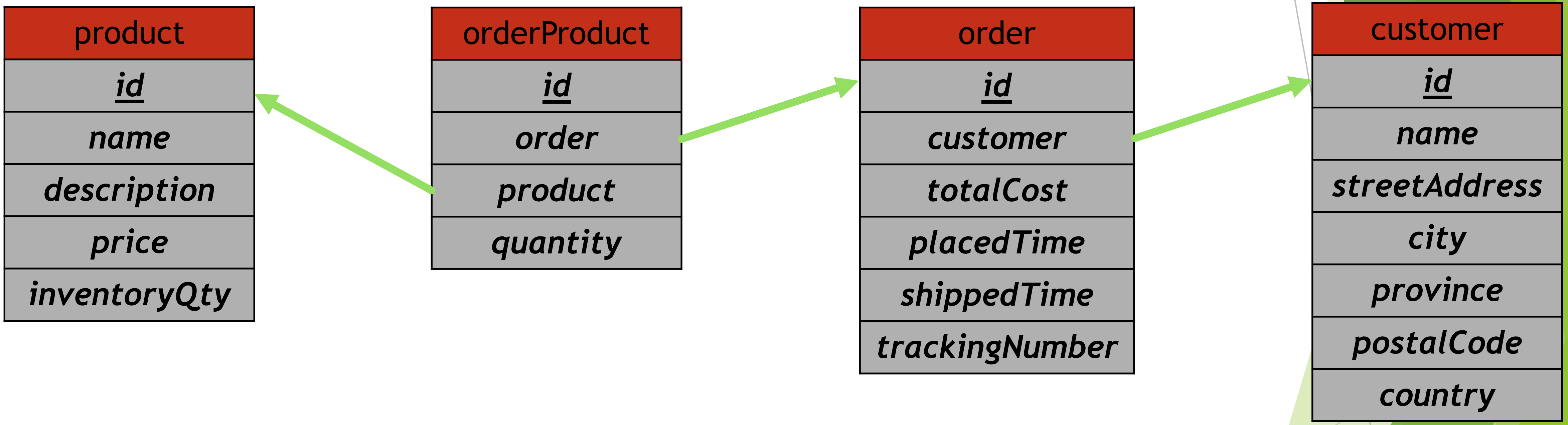


staff
id
name
department

department
id
name
building

building
id
name
faxNumber

Online Retail Example



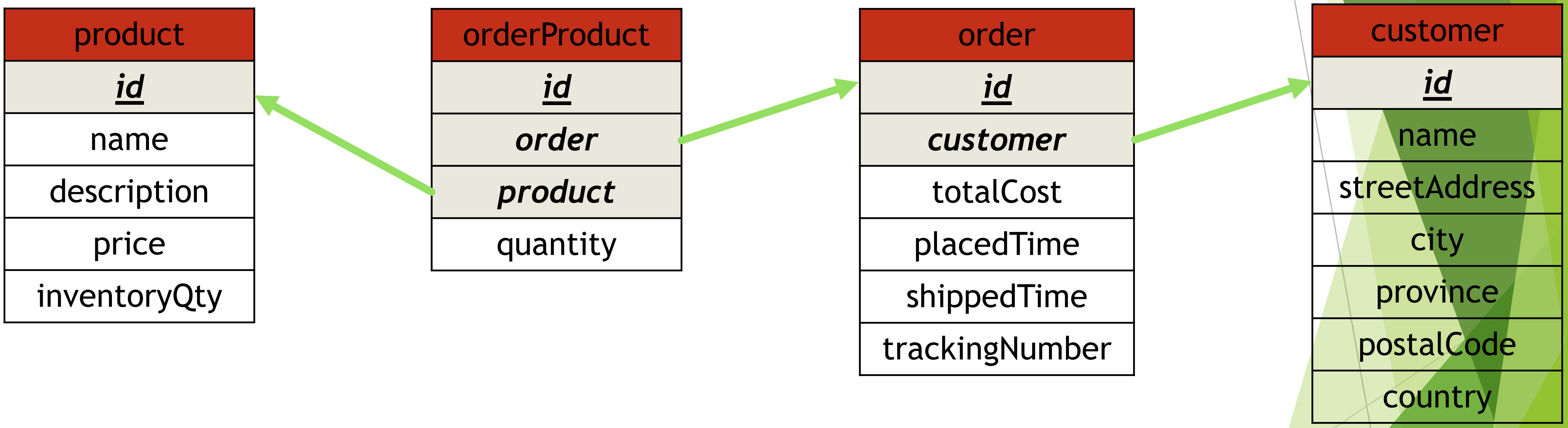
product
id
name
description
price
inventoryQty

orderProduct
id
order
product
quantity

order
id
customer
totalCost
placedTime
shippedTime
trackingNumber

customer
id
name
streetAddress
city
province
postalCode
country

Some Columns are Just Internal References



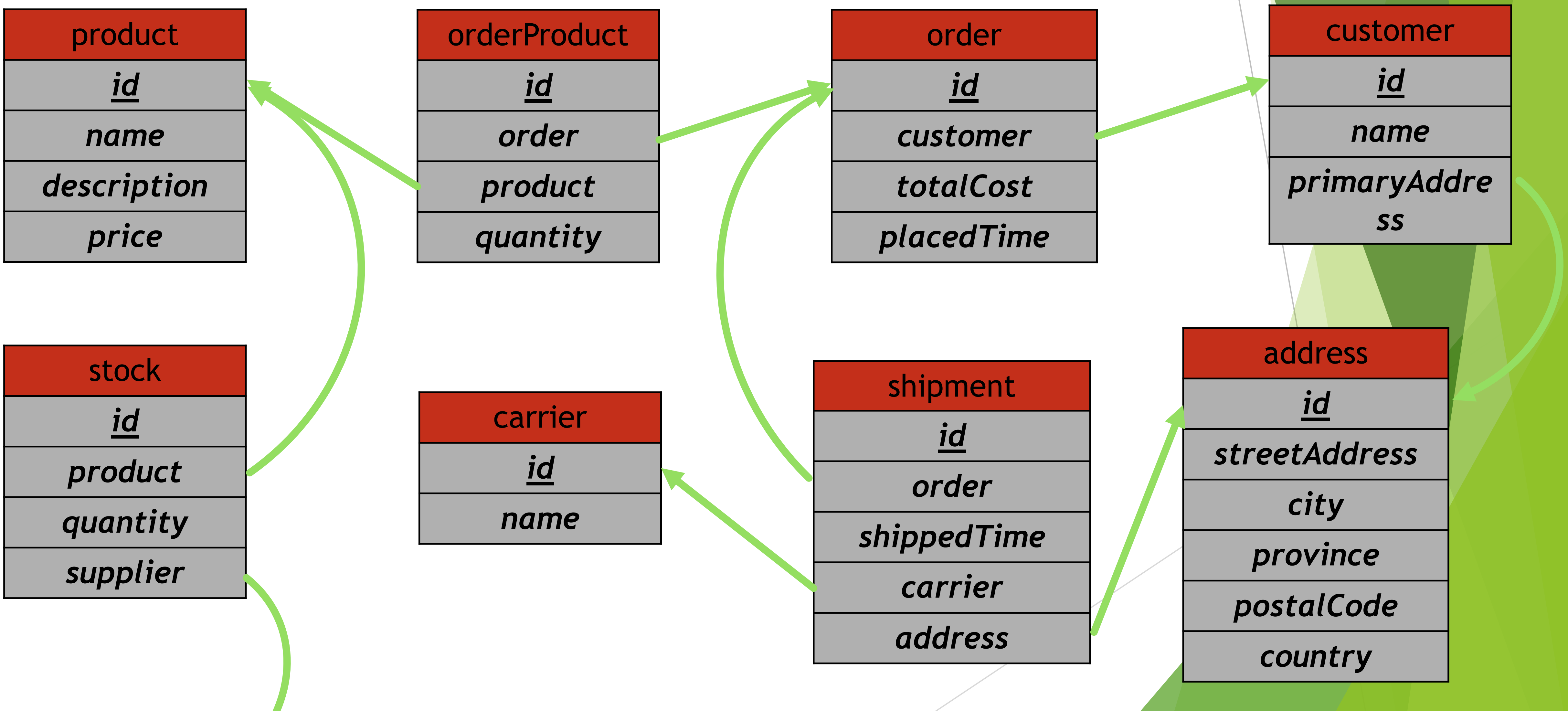
product
id
name
description
price
inventoryQty

orderProduct
id
order
product
quantity

order
id
customer
totalCost
placedTime
shippedTime
trackingNumber

customer
id
name
streetAddress
city
province
postalCode
country

Can make the model more complex



product
id
name
description
price

orderProduct
id
order
product
quantity

order
id
customer
totalCost
placedTime

customer
id
name
primaryAddress

stock
id
product
quantity
supplier

carrier
id
name

shipment
id
order
shippedTime
carrier
address

address
id
streetAddress
city
province
postalCode
country

Basic Steps

- Create table:
  - Table has a name
  - Table has certain named & *typed* columns.
- Add rows to table
  - Each row gives exactly one value to each column (except optional columns can take a null or empty value in a row).
- Write queries to fetch data from the table.

staff			
id	***name***	***room***	***department***

SQL Syntax Overview

Structured Query Language (SQL)

- The standard programming language for relational databases
- SQL is a **declarative** language (most other languages are imperative)
  - You describe the results you want to see
  - You do not describe the detailed steps necessary to gather those results
  - The DBMS cleverly determines an **execution plan** behind the scenes to carry out your requested analysis.
- We can use a client program to connect to the DBMS and running SQL statements **interactively**:
  - run one statement and look at the results before running another one

SQL Dialect

- There are many SQL dialects. However, they share almost all syntax with very minor differences.

- We will cover SQLite, but almost all of what will be covered in this course can be used for other SQL dialects.

- Major SQL dialects:
    - MYSQL, SQLite, Oracle DB, PostgreSQL, …

SELECT-FROM-WHERE

$$\underbrace{\texttt{SELECT FirstName, LastName}}_{\text{Columns to print}}\ \underbrace{\texttt{FROM customers}}_{\text{Table to examine.}}\ \underbrace{\texttt{WHERE City = "Paris";}}_{\text{Filter}}$$

Result is a table with two rows:

FirstName	***LastName***
Camille	Bernard
Dominique	Lefebvre

Filtering, Sorting, and Limiting

We can use more complex filters:

```
SELECT FirstName, LastName FROM customers
  WHERE City = "Chicago"
       AND (State = "Illinois"
          OR State = "IL");
```

Get all columns, sort the results (descending) and limit the results to just the first ten rows:

```
SELECT * FROM tracks ORDER BY UnitPrice DESC
LIMIT 10;
```

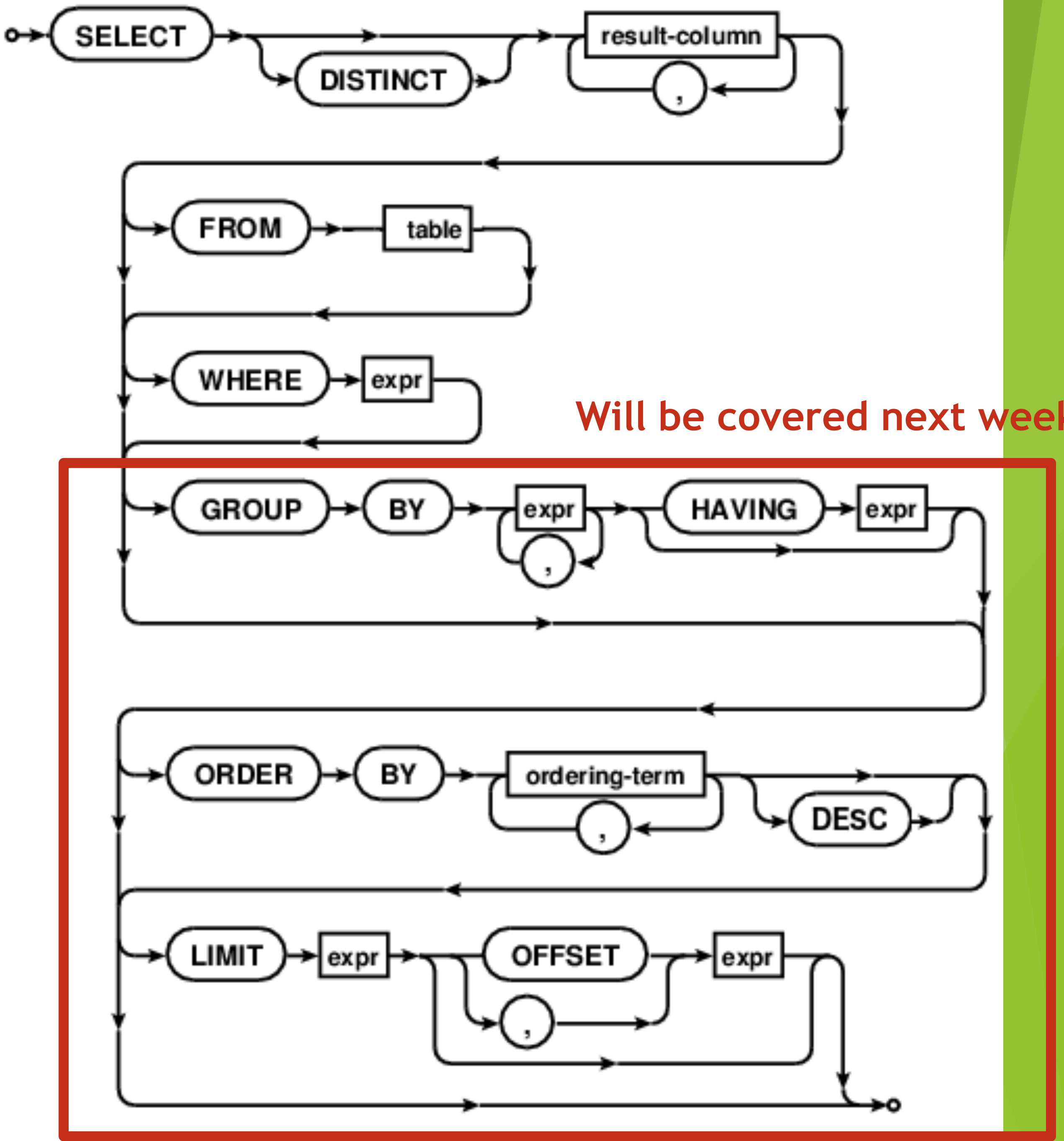
Arithmetic

Your SELECT statements can include arithmetic operation

```
SELECT 1+1;
SELECT ABS(COS(PI()));
SELECT Name, UnitPrice
/(Milliseconds/1000/60)
    AS PricePerMinute FROM tracks;
```

Syntax diagrams

- Any path from start to end is a valid statement.
- Choose which arrows to follow
- The rectangles refer to other diagrams.
- Used by SQLite online docs: https://sqlite.org/lang.html



Will be covered next week

SQL queries are series of *filtering* & *manipulation* steps

1. The FROM expression gives the starting point – a full table.

   The final result will be a subset or aggregation of this.

2. The WHERE expression keeps only those rows passing some test

   This expression can be very complex, but it must be something than can be evaluated on each row, one at a time.

3. GROUP BY combines rows if something about them is the same

4. The SELECT result-columns are computed, including aggregation.

   At this point we have thrown out the columns we don't need.

5. HAVING expression keeps only the aggregated rows passing a test.

6. ORDER BY sorts what's left.

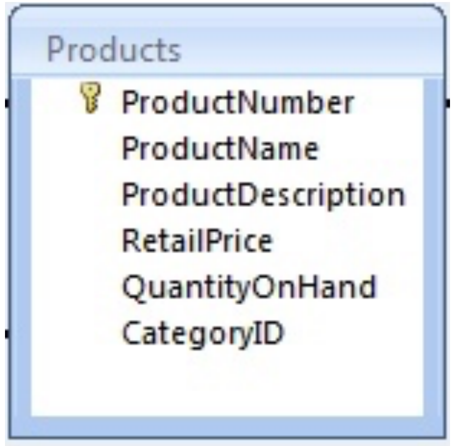
7. LIMIT truncates the results to just a certain number of rows.

Will be covered next week

An Example

What's the average retail price of a bike car rack (categoryID = 5)?

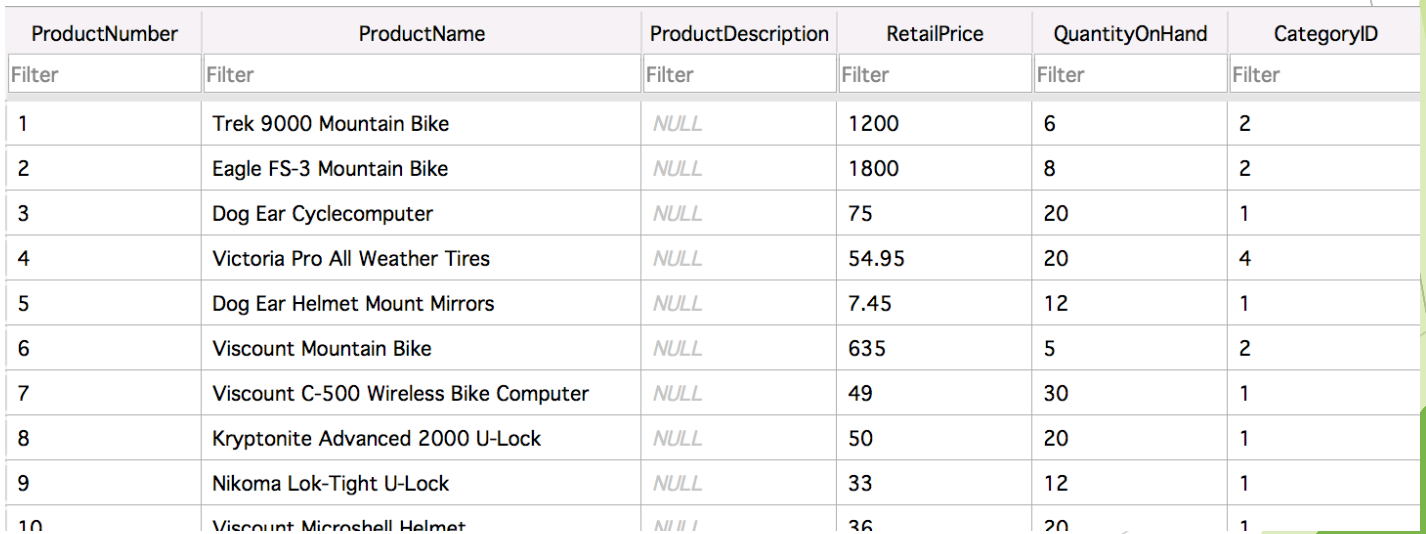
Schema



Products

- ProductNumber
- ProductName
- ProductDescription
- RetailPrice
- QuantityOnHand
- CategoryID

Content of the database



ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
Filter	Filter	Filter	Filter	Filter	Filter
1	Trek 9000 Mountain Bike	NULL	1200	6	2
2	Eagle FS-3 Mountain Bike	NULL	1800	8	2
3	Dog Ear Cyclecomputer	NULL	75	20	1
4	Victoria Pro All Weather Tires	NULL	54.95	20	4
5	Dog Ear Helmet Mount Mirrors	NULL	7.45	12	1
6	Viscount Mountain Bike	NULL	635	5	2
7	Viscount C-500 Wireless Bike Computer	NULL	49	30	1
8	Kryptonite Advanced 2000 U-Lock	NULL	50	20	1
9	Nikoma Lok-Tight U-Lock	NULL	33	12	1
10	Viscount Microshell Helmet	NULL	36	20	1

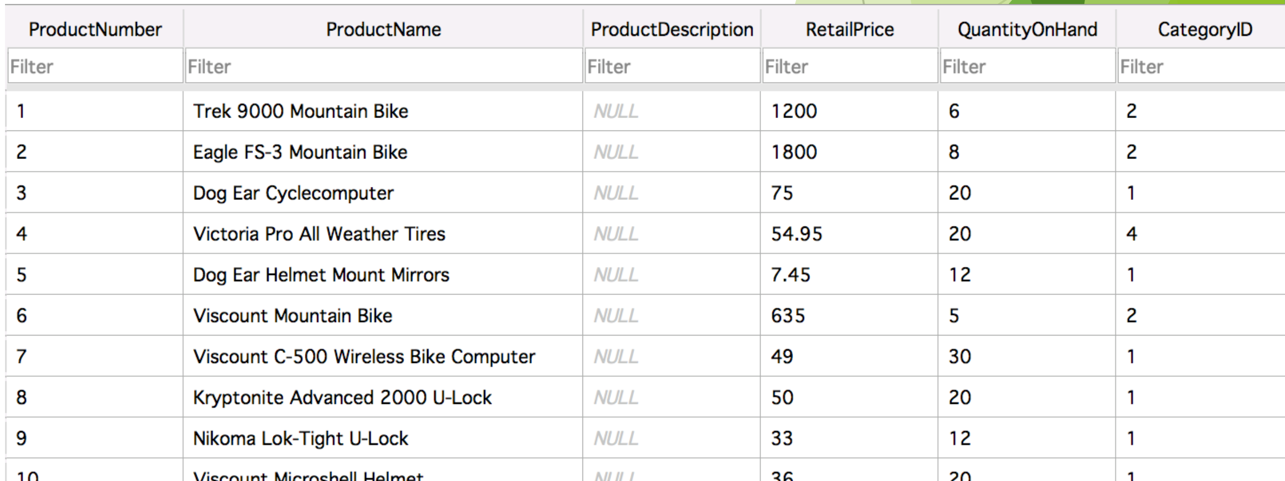
What's the average retail price of a bike car rack?

1. FROM chooses the table of interest
2. WHERE throws out irrelevant rows
3. GROUP BY identifies rows to combine
4. SELECT tells what values to return (allowing math and aggregation)
5. HAVING throws out irrelevant rows (after aggregation)
6. ORDER BY sorts
7. LIMIT throws out rows based on their position in the results

Products table has the price info, so we start there:

```
SELECT * FROM Products
```

This placeholder will be expanded later.



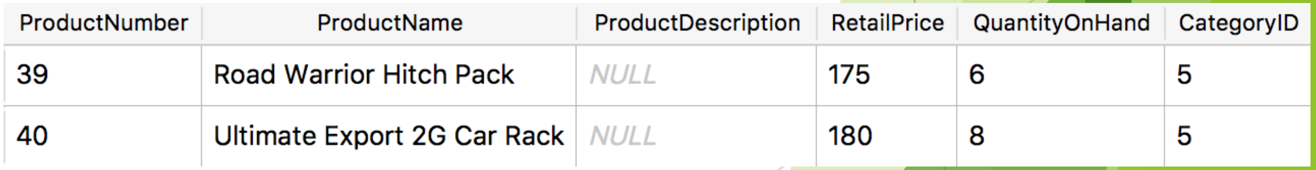
ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
Filter	Filter	Filter	Filter	Filter	Filter
1	Trek 9000 Mountain Bike	NULL	1200	6	2
2	Eagle FS-3 Mountain Bike	NULL	1800	8	2
3	Dog Ear Cyclecomputer	NULL	75	20	1
4	Victoria Pro All Weather Tires	NULL	54.95	20	4
5	Dog Ear Helmet Mount Mirrors	NULL	7.45	12	1
6	Viscount Mountain Bike	NULL	635	5	2
7	Viscount C-500 Wireless Bike Computer	NULL	49	30	1
8	Kryptonite Advanced 2000 U-Lock	NULL	50	20	1
9	Nikoma Lok-Tight U-Lock	NULL	33	12	1
10	Viscount Microshell Helmet	NULL	36	20	1

What’s the average retail price of a bike car rack?

1. FROM chooses the table of interest
2. WHERE throws out irrelevant rows
3. GROUP BY identifies rows to combine
4. SELECT tells what values to return (allowing math and aggregation)
5. HAVING throws out irrelevant rows (after aggregation)
6. ORDER BY sorts
7. LIMIT throws out rows based on their position in the results

We only need the bike rack products, so we filter on CategoryID = 5

```
SELECT * FROM Products
    WHERE CategoryID = 5;
```



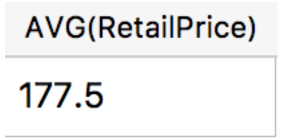
ProductNumber	ProductName	ProductDescription	RetailPrice	QuantityOnHand	CategoryID
39	Road Warrior Hitch Pack	*NULL*	175	6	5
40	Ultimate Export 2G Car Rack	*NULL*	180	8	5

What’s the average retail price of a bike car rack?

1. FROM chooses the table of interest
2. WHERE throws out irrelevant rows
3. GROUP BY identifies rows to combine
4. SELECT tells what values to return (allowing math and aggregation)
5. HAVING throws out irrelevant rows (after aggregation)
6. ORDER BY sorts
7. LIMIT throws out rows based on their position in the results

We want the RetailPrice column, and we want to aggregate all the rows with the average function

```
SELECT AVG(RetailPrice) FROM
   Products WHERE CategoryID = 5
```



AVG(RetailPrice)
177.5

Next Lecture

- More Examples
- How to visualize database content?
- How to execute SQL queries on databases?