Learning Objectives

- Understand the data-representation terminology underlying relational databases
- Understand core SQL concepts that stem from relational algebra
- Cover all the fundamental operators other than divide
- Gain familiarity with SQL syntax

Relational Databases and Tables

- A Relational Database is a database management system (DBMS) that holds a set of tables, like Excel worksheets, each filled with data
- A table = a set of rows or “records”
  - Like a list...
  - ...but rows have no assumed order
  - Even columns have no assumed order

A sample database table

<table>
<thead>
<tr>
<th>Product</th>
<th>Table name</th>
<th>Attribute names</th>
</tr>
</thead>
<tbody>
<tr>
<td>FName</td>
<td>Price</td>
<td>Category</td>
</tr>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
</tr>
</tbody>
</table>

DB Terminology overview

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation (also, Entity)</td>
<td>A table.</td>
</tr>
<tr>
<td>Tuple, (also, record or “entity instance”)</td>
<td>A row.</td>
</tr>
<tr>
<td>Attribute (also, field)</td>
<td>A column.</td>
</tr>
<tr>
<td>Attribute value</td>
<td>The value in a particular table cell (i.e., a row/column intersection).</td>
</tr>
<tr>
<td>Primary key</td>
<td>Specified column(s) providing a unique value for every row.</td>
</tr>
<tr>
<td>Table scheme</td>
<td>The set of attributes (not their values) defining the structure of a table.</td>
</tr>
<tr>
<td>Database schema</td>
<td>The full set of named tables and their schemes, in the full database (which could contain many tables).</td>
</tr>
</tbody>
</table>

SQL – what is it?

- All SQL query take some rectangular input table(s) and, using that data as you nicely ask it to, generates your desirable output table
  - You must ask nicely, i.e., properly
- A simple language
  - Not a true programming language
  - Much easier to learn
  - Still very hard to master
- Why was SQL developed?
  - As a user-friendly means to define and manipulating data in relational databases
  - Incredibly valuable for addressing managerial ad hoc query questions
SQL Query

Basic form:

```
SELECT attributes
FROM table
WHERE conditions
```

(or multiple tables, often “joined”) (“row restrictions”)

### 1-table query with row filtering

```
Product
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
```

```
SELECT *
FROM Product
WHERE category='Gadgets'
```

### A query projecting specific columns

```
Product
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
```

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 100
```

### Selections

What goes in the WHERE clause:
- \( x = y \), \( x < y \), \( x \leq y \), etc
- For numbers, they have the usual meanings
- For characters: lexicographic ordering
- For dates and times, earlier is less than later
- Pattern matching on strings: s LIKE p  (next slide)
- You can combine selections using AND / OR type logic
  - E.g., WHERE \( x < y \) AND \( x < z \) OR \( y > z \)

### The LIKE operator

- s LIKE p: pattern matching on strings
- p may contain two special "wildcard" symbols:
  - \( % \): any sequence of characters
  - \( _ \): any single character
- Note: these two special characters are system dependent...
- "Google" the LIKE wildcard symbols for any system you use

Product(Name, Price, Category, Manufacturer)
Find all products whose name mentions ‘gizmo’:

```
SELECT *
FROM Products
WHERE PName LIKE '%gizmo%'
```

### How to eliminate identical output rows?

Compare to:

```
SELECT DISTINCT category
FROM Product
```

```
Category: Gadgets
Photography
Household
```

```
SELECT category
FROM Product
```
Ordering the Results

```
SELECT  pname, price, manufacturer
FROM    Product
WHERE   category='Gadgets' AND price > 50
ORDER BY price, pname
```

Ordering is ascending, unless you specify ORDER BY DESC
(DESC means you want descending order)

Ties are broken by the second attribute on the ORDER BY list, etc.

**“Joining tables” in SQL**

- Connect two or more tables:

```
Product
\|  PName \|  Price \|  Category \|  Manufacturer \\
---\|---\|---\|---\|---
\| Gizmo \|  $19.99 \|  Gadgets \|  GizmoWorks \\
\| Powergizmo \|  $29.99 \|  Gadgets \|  GizmoWorks \\
\| SingleTouch \|  $149.99 \|  Photography \|  Canon \\
\| MultiTouch \|  $203.99 \|  Household \|  Hitachi \\
```

Company

```
\| CName \| StockPrice \| Country \\
---\|---\|---
\| GizmoWorks \| 25 \| USA \\
\| Canon \| 65 \| Japan \\
\| Hitachi \| 15 \| Japan \\
```

What is the connection? Do you think the DBMS sees that?

**Joins using SQL**

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT  PName, Price
FROM    Product, Company
WHERE   Manufacturer=CName AND Country='Japan'
        AND Price <= 200
```

Example: Find all countries that manufacture some product in the ‘Gadgets’ category

```
Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

SELECT Country
FROM    Product, Company
WHERE   Manufacturer=CName AND Category='Gadgets'
```

Joins using SQL

```
```

Joins and unexpected “duplicated” results

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
Joins in SQL

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

```
SELECT Country
FROM Product, Company
WHERE Manufacturer=Company.Cname AND Category='Gadgets'
```

What will the two output lines be? Do we recall how to avoid showing identical output rows?

Disambiguating Attributes

- Sometimes two relations have same attribute
  - Person(pname, address, worksfor)
  - Company(cname, address)

```
SELECT DISTINCT pname, address
FROM Person, Company
WHERE worksfor = Company.cname
```

Renaming Columns

```
SELECT Product.pName AS prodName, Price AS askPrice
FROM Product
WHERE Price > 100
```

Two tables for our example

```
SELECT * from driver;
NAME | DOB
Jim Smith | 11 Jan 1980
Bob Smith | 23 Mar 1981
Bob Jones | 3 Dec 1986
```

```
SELECT * from car;
REGNO | MAKE | COLOUR | PRICE | OWNER
F611 AAA | FORD | RED | 12000 | Jim Smith
J111 BBB | SKODA | BLUE | 11000 | Jim Smith
A155 BDE | MERCEDES | BLUE | 22000 | Bob Smith
K555 GHT | FIAT | GREEN | 6000 | Bob Jones
SC04 BFE | SMART | BLUE | 13000 |
What if you forget the "join condition?"

- The following SQL is missing the join condition for the two tables:
  - SELECT *
    FROM car, driver

- What does it do? It takes all combinations of rows from the two tables.

<table>
<thead>
<tr>
<th>REGNO</th>
<th>MAKE</th>
<th>COLOUR</th>
<th>PRICE</th>
<th>OWNER</th>
<th>NAME</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>F611AAA</td>
<td>FORD</td>
<td>RED</td>
<td>6000</td>
<td>Bob Jones</td>
<td>Bob Jones</td>
<td>3 Dec 1986</td>
</tr>
<tr>
<td>A155BDE</td>
<td>MERCEDES</td>
<td>BLUE</td>
<td>22000</td>
<td>Bob Jones</td>
<td>Bob Jones</td>
<td>23 Mar 1981</td>
</tr>
<tr>
<td>K555GHT</td>
<td>FIAT</td>
<td>GREEN</td>
<td>6000</td>
<td>Bob Jones</td>
<td>Bob Jones</td>
<td>3 Dec 1986</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>BLUE</td>
<td>13000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>RED</td>
<td>11000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>GREEN</td>
<td>12000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>BLUE</td>
<td>6000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>RED</td>
<td>22000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>GREEN</td>
<td>13000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
</tbody>
</table>

Adding the Join condition

- To make the prior query a proper "join" of the two tables, we add the following WHERE clause row restriction:
  - WHERE Car.Owner = Driver.Name

<table>
<thead>
<tr>
<th>REGNO</th>
<th>MAKE</th>
<th>COLOUR</th>
<th>PRICE</th>
<th>OWNER</th>
<th>NAME</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>F611AAA</td>
<td>FORD</td>
<td>RED</td>
<td>6000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>3 Dec 1986</td>
</tr>
<tr>
<td>A155BDE</td>
<td>MERCEDES</td>
<td>BLUE</td>
<td>22000</td>
<td>Bob Jones</td>
<td>Bob Jones</td>
<td>23 Mar 1981</td>
</tr>
<tr>
<td>K555GHT</td>
<td>FIAT</td>
<td>GREEN</td>
<td>6000</td>
<td>Bob Jones</td>
<td>Bob Jones</td>
<td>3 Dec 1986</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>BLUE</td>
<td>13000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>RED</td>
<td>11000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>GREEN</td>
<td>12000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>BLUE</td>
<td>6000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>RED</td>
<td>22000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>GREEN</td>
<td>13000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>11 Jan 1980</td>
</tr>
</tbody>
</table>

Traditional join syntax

- SELECT * FROM car, driver
  WHERE owner = name

An alternative syntax for the join in SQL

- SELECT *
  FROM car JOIN driver ON ( owner = name )

Outer joins

- Now let’s learn about the “outer join”
  - The 3 variations on the outer join are the...
  - Left outer join
  - Right outer join
  - Full outer join

OUTER JOIN

- Consider the last row shown in the product table we looked at...

<table>
<thead>
<tr>
<th>REGNO</th>
<th>MAKE</th>
<th>COLOUR</th>
<th>PRICE</th>
<th>OWNER</th>
<th>NAME</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC04BFE</td>
<td>SMART</td>
<td>BLUE</td>
<td>13000</td>
<td>Jim Smith</td>
<td>Jim Smith</td>
<td>23 Mar 1981</td>
</tr>
</tbody>
</table>

- This is a car without an owner (due to the "blank")
- Sometimes we want to see the rows that fail the join condition due to NULL (blank) values.
  - This idea is referred to as performing an “outer join.”
  - There are "left" and "right" outer joins, depending on which rows you want to keep.
- The join operation we have discussed up to this point is known as a standard "inner" join.
Consider this: SELECT * FROM car JOIN driver on (driver = name)

To the LEFT of the JOIN
To the RIGHT of the JOIN

If you want all the rows in CAR to always be in the answer (whether matched or unmatched), you need a "left outer join"

If you want all the rows in DRIVER to always be in the answer (whether matched or unmatched), you need a "right outer join"

What if you want to keep all the rows from both sides?

You need a "full outer join," known as simply a FULL JOIN.

SELECT *
FROM car LEFT JOIN driver ON ( owner = name )

Example: Full outer join

SELECT *
FROM car FULL JOIN driver ON ( owner = name )