SQL: joins

Recap: Tables for Plug-in Cars

<table>
<thead>
<tr>
<th>CAR</th>
<th>MODEL</th>
<th>MAN</th>
<th>PRICE</th>
<th>BATTERY</th>
<th>MOTORKW</th>
<th>MPGE</th>
<th>MILES</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>I3</td>
<td>BMW</td>
<td>42375</td>
<td>22</td>
<td>125</td>
<td>123</td>
<td>80</td>
<td>2700</td>
<td></td>
</tr>
<tr>
<td>BCE</td>
<td>MB</td>
<td>42375</td>
<td>28</td>
<td>130</td>
<td>84</td>
<td>80</td>
<td>3935</td>
<td></td>
</tr>
<tr>
<td>SPARKEV</td>
<td>CHEVY</td>
<td>27495</td>
<td>20</td>
<td>110</td>
<td>119</td>
<td>82</td>
<td>2989</td>
<td></td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td></td>
</tr>
<tr>
<td>S60</td>
<td>TESLA</td>
<td>71070</td>
<td>60</td>
<td>222</td>
<td>89</td>
<td>208</td>
<td>4647</td>
<td></td>
</tr>
<tr>
<td>500E</td>
<td>FIAT</td>
<td>32650</td>
<td>24</td>
<td>83</td>
<td>116</td>
<td>87</td>
<td>2980</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATE</th>
<th>CAR</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>LEAF</td>
<td>99</td>
</tr>
<tr>
<td>WA</td>
<td>I3</td>
<td>32</td>
</tr>
<tr>
<td>HI</td>
<td>LEAF</td>
<td>26</td>
</tr>
<tr>
<td>CA</td>
<td>LEAF</td>
<td>578</td>
</tr>
<tr>
<td>CA</td>
<td>I3</td>
<td>186</td>
</tr>
<tr>
<td>CA</td>
<td>S60</td>
<td>18</td>
</tr>
<tr>
<td>CA</td>
<td>SPARKEV</td>
<td>15</td>
</tr>
<tr>
<td>CA</td>
<td>S60E</td>
<td>30</td>
</tr>
<tr>
<td>GA</td>
<td>LEAF</td>
<td>118</td>
</tr>
<tr>
<td>GA</td>
<td>I3</td>
<td>38</td>
</tr>
<tr>
<td>OR</td>
<td>LEAF</td>
<td>39</td>
</tr>
<tr>
<td>OR</td>
<td>I3</td>
<td>13</td>
</tr>
<tr>
<td>UT</td>
<td>LEAF</td>
<td>12</td>
</tr>
<tr>
<td>CO</td>
<td>LEAF</td>
<td>25</td>
</tr>
</tbody>
</table>

Recap: the SQL Select Command

```
SELECT [DISTINCT] sel_expression [, sel_expression …]
FROM table_references
[WHERE condition]
[GROUPBY column [,column...]]
[[HAVING condition]]
[ORDER BY columns [ASC|DESC]]
[LIMIT num];
```

Practices...

- How many cars are sold (in total) at each state? What are the top 3 states?

```
SELECT st, SUM(qty)
FROM sales
GROUP BY st
ORDER BY SUM(qty) DESC
LIMIT 3;
```
A shorthand

• can use numbers instead of column names as a shorthand— the number corresponds to the order that appears in the SELECT clause
• These two queries are equivalent:

```sql
SELECT st, SUM(qty)
FROM sale
GROUP BY st
ORDER BY SUM(qty) DESC
LIMIT 3;
```

```sql
SELECT st, SUM(qty)
FROM sales
GROUP BY 1
ORDER BY 2 DESC
LIMIT 3;
```

Practices…

• What states have sales of at least two different models of cars?

```sql
SELECT st
FROM sales
GROUP BY st
HAVING COUNT(*) >= 2
ORDER BY SUM(qty) DESC;
```

Practices…

• filter out duplicate emails

```sql
SELECT DISTINCT Email
FROM Person
```

```sql
SELECT Email
FROM Person
GROUP BY Email
HAVING COUNT(*) = 1
```

Conditional logic

```sql
CASE WHEN ...
    THEN ...
    [ELSE ...
END
```

• SQL's way of handling if/else
• This statement defines a new column
Conditional logic: example

SELECT model, man, price,
  CASE WHEN price > '35000' THEN 'no'
       ELSE 'yes' END AS can_afford
FROM car

• I want to have another column that identifies whether I can afford that car

Conditional logic: example

SELECT model,
  man,
  price,
  CASE WHEN price > '50000' THEN 'over 50000'
       WHEN price > '35000' THEN 'over 35000'
       ELSE NULL
  END AS price_range
FROM car

• multiple outcomes are evaluated from left to right
• can specify null values as well — the ELSE is optional here

Conditional logic: example

• CASE and aggregations

SELECT
  CASE WHEN price > '50000' THEN 'over 50000'
       WHEN price > '35000' THEN 'over 35000'
       ELSE 'below 35000'
  END AS price_range,
  COUNT(1) AS count
FROM car
GROUP BY 1

<table>
<thead>
<tr>
<th>price_range</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>below 35000</td>
<td>3</td>
</tr>
<tr>
<td>over 35000</td>
<td>2</td>
</tr>
<tr>
<td>over 50000</td>
<td>1</td>
</tr>
</tbody>
</table>

Conditional logic: example

• Pivoting the table — display counts horizontally
• Note the case conditions!

SELECT
  COUNT(CASE WHEN price > '50000' THEN 1 END) AS gt_50000_count,
  COUNT(CASE WHEN price > '35000' AND price < '50000' THEN 1 END) AS gt_35000_count,
  COUNT(CASE WHEN price < '35000' THEN 1 END) AS lt_35000_count,
  COUNT(*) AS total
FROM car

<table>
<thead>
<tr>
<th>gt_50000_count</th>
<th>gt_35000_count</th>
<th>lt_35000_count</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
Administrivia

- Blackboard up and running
  - check out the slides
  - let me know if you’re not on Blackboard
- Slides also available at
  - http://www.cs.rochester.edu/~yjin18/

Join

- clarification: joins stitches data together horizontally — after the join, the table is wider than before
- to be able to combine two table schemas, we’ll have to find something in common

```
SELECT (column_list)
FROM table_name
INNER | LEFT | RIGHT | OUTER] JOIN table_name
ON qualification_list
WHERE ...
```

Join: an example

- two equivalent ways to write a join expression

```
select *
from car, sales
where car.model = sales.model;
```

```
select *
from car
join sales
on car.model = sales.model;
```
• Join all rows from the car table on to rows in the sales table, for which the model field in the car table is equal to the model field in the sales table
• 11 attributes, 14 records

<table>
<thead>
<tr>
<th>MODEL</th>
<th>MAN</th>
<th>PRICE</th>
<th>BATTERY</th>
<th>MOTOPW</th>
<th>MPGE</th>
<th>MILES</th>
<th>WEIGHT</th>
<th>ST</th>
<th>MODEL</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>BMW</td>
<td>42275</td>
<td>22</td>
<td>123</td>
<td>123</td>
<td>60</td>
<td>2700</td>
<td>WA</td>
<td>O2</td>
<td>32</td>
</tr>
<tr>
<td>O1</td>
<td>BMW</td>
<td>42375</td>
<td>12</td>
<td>123</td>
<td>123</td>
<td>60</td>
<td>2700</td>
<td>CA</td>
<td>O3</td>
<td>38</td>
</tr>
<tr>
<td>O1</td>
<td>BMW</td>
<td>42375</td>
<td>22</td>
<td>123</td>
<td>123</td>
<td>60</td>
<td>2700</td>
<td>OR</td>
<td>O3</td>
<td>38</td>
</tr>
<tr>
<td>SPARCEV</td>
<td>CHEVY</td>
<td>27495</td>
<td>26</td>
<td>110</td>
<td>119</td>
<td>82</td>
<td>2980</td>
<td>CA</td>
<td>SPARCEV</td>
<td>15</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>WA</td>
<td>LEAF</td>
<td>99</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>HI</td>
<td>LEAF</td>
<td>99</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>CA</td>
<td>LEAF</td>
<td>578</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>CA</td>
<td>LEAF</td>
<td>118</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>OR</td>
<td>LEAF</td>
<td>39</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>UT</td>
<td>LEAF</td>
<td>12</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>80</td>
<td>115</td>
<td>75</td>
<td>3340</td>
<td>CO</td>
<td>LEAF</td>
<td>25</td>
</tr>
<tr>
<td>S60</td>
<td>TESLA</td>
<td>71070</td>
<td>66</td>
<td>222</td>
<td>89</td>
<td>208</td>
<td>4647</td>
<td>CA</td>
<td>S60</td>
<td>18</td>
</tr>
<tr>
<td>S50E</td>
<td>FIAT</td>
<td>32050</td>
<td>24</td>
<td>83</td>
<td>116</td>
<td>87</td>
<td>2940</td>
<td>CA</td>
<td>S50E</td>
<td>30</td>
</tr>
</tbody>
</table>

Types of joins
• inner join – the default
  • JOIN or INNER JOIN
• left (outer) join
  • LEFT JOIN or LEFT OUTER JOIN
• right (outer) join
  • RIGHT JOIN or RIGHT OUTER JOIN
• full (outer) join
  • not implemented/available in MySQL
  • [http://joins.spathon.com/](http://joins.spathon.com/)

Join: multiple tables
• For each sale, find the full names of states and manufacturer of cars, and the number sold in descending order
  select name, man, qty
  from state
  join sales
  join car
  on state.st = sales.st and sales.model = car.model
  order by qty desc;

Practices...
• Retrieve FirstName, LastName, City, and State for each person, regardless of whether that person has an address.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Column Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonId</td>
<td>int</td>
<td>AddressId</td>
<td>int</td>
</tr>
<tr>
<td>FirstName</td>
<td>varchar</td>
<td>PersonId</td>
<td>varchar</td>
</tr>
<tr>
<td>LastName</td>
<td>varchar</td>
<td>City</td>
<td>varchar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>State</td>
<td>varchar</td>
</tr>
</tbody>
</table>
Practices...

- Retrieve FirstName, LastName, City, and State for each person, regardless of whether that person has an address

```sql
SELECT FirstName, LastName, City, State
FROM Person
LEFT JOIN Address
ON Person.personId = Address.personId
```

Practices...

- Find customers who never order anything.

```sql
SELECT C.Name from Customers C
LEFT JOIN Orders O
ON C.Id = O.CustomerId
WHERE O.CustomerId is NULL
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Henry</td>
</tr>
<tr>
<td>3</td>
<td>Sam</td>
</tr>
<tr>
<td>4</td>
<td>Max</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>CustomerId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Different flavors of SQL

- Multiple standard revisions and multiple flavors/implementations exist
- Revisions of standard: successively introduced additional features, mostly backwards-compatible with older standards
- Common implementations: Oracle, MySQL, SQLite, Postgres, MS SQL (Microsoft), Microsoft Access
- Implementations comply with the standard, but people almost never completely implement everything in the standard

Market share: Source: JoinVision E-Services GmbH, July 2006
UNIONs

• stack a dataset on top of another
• Restrictions
  • Both tables must have the same number of columns
  • The columns must have the same data types in the same order as the first table
• UNION appends distinct values
• UNION ALL appends all values (probably used more often)

UNIONs: example

• can simulate a FULL OUTER JOIN using unions
• example: the User and Likes database
  SELECT users.name, likes.like
  FROM users
  LEFT OUTER JOIN likes
  ON users.id = likes.user_id
  UNION ALL
  SELECT users.name, likes.like
  FROM users
  RIGHT OUTER JOIN likes
  ON users.id = likes.user_id

JOIN tricks — self joins

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Salary</th>
<th>ManagerId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>70000</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Henry</td>
<td>80000</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Sam</td>
<td>60000</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>Max</td>
<td>90000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

• Find people who earn more than their managers. In our example the query should return

Employee
Joe

JOIN tricks — self joins

• Find people who earn more than their managers.
  SELECT employee.Name AS 'Employee'
  FROM Employee employee
  JOIN Employee manager
  ON employee.ManagerId = manager.Id
  WHERE employee.Salary > manager.Salary

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Salary</th>
<th>ManagerId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>70000</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Henry</td>
<td>80000</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Sam</td>
<td>60000</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>Max</td>
<td>90000</td>
<td>NULL</td>
</tr>
</tbody>
</table>
JOIN tricks — inequality joins

• find pairs of car model names, in which the first name is shorter than the next

```
SELECT car1.model, car2.model
FROM car AS car1
JOIN car AS car2
ON length(car1.model) < length(car2.model)
```

JOIN tricks — multiple keys

```
SELECT *
FROM car
JOIN sales
ON car.model = sales.model
AND car.man = sales.man
```

Datatypes

• why datatypes?
  • easier for data manipulation
  • easier to store, and saves space on disk!
  • depending on the implementation, datatypes vary, but more or less the same
  • functions usually act on just one type of data, e.g. SUM() for numerical data
  • in MySQL 5.7, there are 5 major categories (numeric, text, date/time, spatial, JSON) — each has a handful of datatypes

• Why?
  • accuracy
  • speed
    • has to do with implementations — “indexes”
    • might hit one of the indexes and get a better speed

```
<table>
<thead>
<tr>
<th>MODEL</th>
<th>MAN</th>
<th>PRICE</th>
<th>BATTERY</th>
<th>MPGE</th>
<th>MILES</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>I3</td>
<td>BMW</td>
<td>42375</td>
<td>22</td>
<td>123</td>
<td>80</td>
<td>2700</td>
</tr>
<tr>
<td>BCE</td>
<td>HB</td>
<td>42375</td>
<td>28</td>
<td>84</td>
<td>80</td>
<td>3935</td>
</tr>
<tr>
<td>SPARKEV</td>
<td>CHEVY</td>
<td>27995</td>
<td>19</td>
<td>119</td>
<td>82</td>
<td>2989</td>
</tr>
<tr>
<td>LEAF</td>
<td>NISSAN</td>
<td>31820</td>
<td>24</td>
<td>115</td>
<td>75</td>
<td>3340</td>
</tr>
<tr>
<td>S60</td>
<td>TESLA</td>
<td>71870</td>
<td>60</td>
<td>89</td>
<td>208</td>
<td>4647</td>
</tr>
<tr>
<td>500E</td>
<td>FIAT</td>
<td>32650</td>
<td>24</td>
<td>116</td>
<td>87</td>
<td>2980</td>
</tr>
</tbody>
</table>
```
Datatypes: numeric

- Integer Types (Exact Value): INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
- Fixed-Point Types (Exact Value): DECIMAL, NUMERIC
- Floating-Point Types (Approximate Value): FLOAT, DOUBLE
- Bit-Value Type: BIT

  the difference is in the size. e.g. TINYINT ranges from -128 to 127, while INT ranges from -2147483648 to 2147483647

Datatypes: string

- CHAR, VARCHAR
- fixed and variable length string
- BINARY, VARBINARY
- fixed and variable length binary
- BLOB (Binary Large OBjects), TEXT
- binary and character of bigger size
- ENUM
  - a list of possible values (e.g. ‘XS’, ‘S’, ‘M’, ‘L’, ‘XL’)
- SET
  - like enum but stores more than 1 choice

Datatypes: date and time

- DATE
  - YYYY-MM-DD
- DATETIME
  - YYYY-MM-DD HH:MI:SS
- TIMESTAMP
  - Also YYYY-MM-DD HH:MI:SS, but stored as the number of seconds since ‘1970-01-01 00:00:00’ UTC (i.e. the Unix epoch)
- TIME
  - HH:MI:SS
- YEAR
  - A year in two-digit or four-digit format.

Datatypes: date and time

Date and time are tricky to get right! How do you...

  compare dates?
  take differences?
  group into weeks or months?
  convert string to dates?
Datatypes: others

• Spatial
  • GEOMETRY, POINT, LINESTRING, POLYGON etc
• JSON


More resources: learning

• SQLSchool: interactive PostgreSQL tutorial with examples
  • https://sqlschool.modeanalytics.com/
• SQLZoo: interactive SQL tutorial with exercises
  • http://sqlzoo.net/
• W3Schools:
  • http://www.w3schools.com/sql/

More resources: troubleshooting

• Googling...
• The database’s reference manual
  • e.g. MySQL 5.7 Reference Manual is at http://dev.mysql.com/doc/refman/5.7/en/
• Ask questions at http://stackoverflow.com/

More resources: data

http://alpha.data.gov/

A collection of open data from government, companies, and non-profits including:
• Commerce - GPS, Weather, Census Data
• Energy - Outlook and Analysis, Energy Usage, Energy Star, Your Car’s Data Stream, Renewable Energy + See all energy datasets and tools
• Health - Recalls and Safety Alerts, Medicare Plans, Hospital Quality Data, Public Health Data, Clinical Trials, Health Indicators, Your Health Record + See all health datasets & tools
• Education - Open Badges, Academic Data, Learning Registry, Education Data Standards, Higher Education Datasets + See all education datasets & tools
Health Data for Epidemiology

[http://chir.asu.edu/data](http://chir.asu.edu/data)

A key activity of CHiR from its outset has been obtaining health data being collected routinely by various health care and public health entities and making these data available for use for

Data for Urban Planning


City Forward data catalog is a listing of the data sets currently available for use from publicly available data sources about cities and metropolitan areas around the world. Please use the links back to the data source and their terms of use.

Database of Technology Companies


CrunchBase is the free database of technology companies, people, and investors that anyone can edit.

Google DB Search Engine

[www.google.com/publicdata/directory](www.google.com/publicdata/directory)

- Useful search engine
- You can save to “my data”
- Google maintains over 100 million curated pages of data
Amazon Public Data Sets

http://aws.amazon.com/datasets

- AWS Amazon Web Services
- Hadoop friendly files
- Can use seamlessly with Amazon cloud computing utility ‘EC2 Amazon ‘Elastic Cloud Compute’

In conclusion…

- Can use conditional logic to create new columns and pivot tables.
- There are 4 types of joins to integrate information from two tables.
- UNION and UNION ALL stitches two parts of the dataset together.
- Common data types include numerics, strings and date/times.

Stay tuned for next week

- subqueries
- window functions
- (maybe) query optimization

- and the week after next...
  - inserting, deleting and updating records
  - constraints for tables
  - what else do you want to see?