Lecture 2
Introduction to SQL

Part I

Today's Lecture

1. SQL introduction & schema definitions
2. Basic single-table queries
3. Multi-table queries

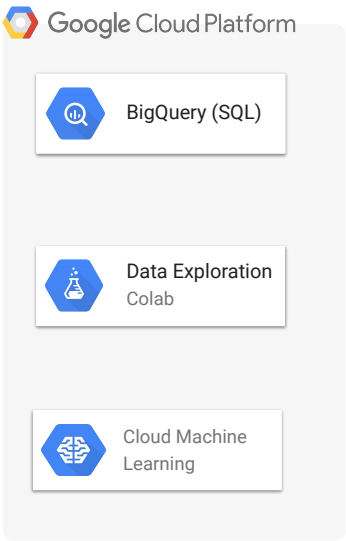
Project

Full data cycle on cloud

1 Run queries on public datasets

2 Explore/Visualize public datasets

3 Predict using Machine Learning



Example 1

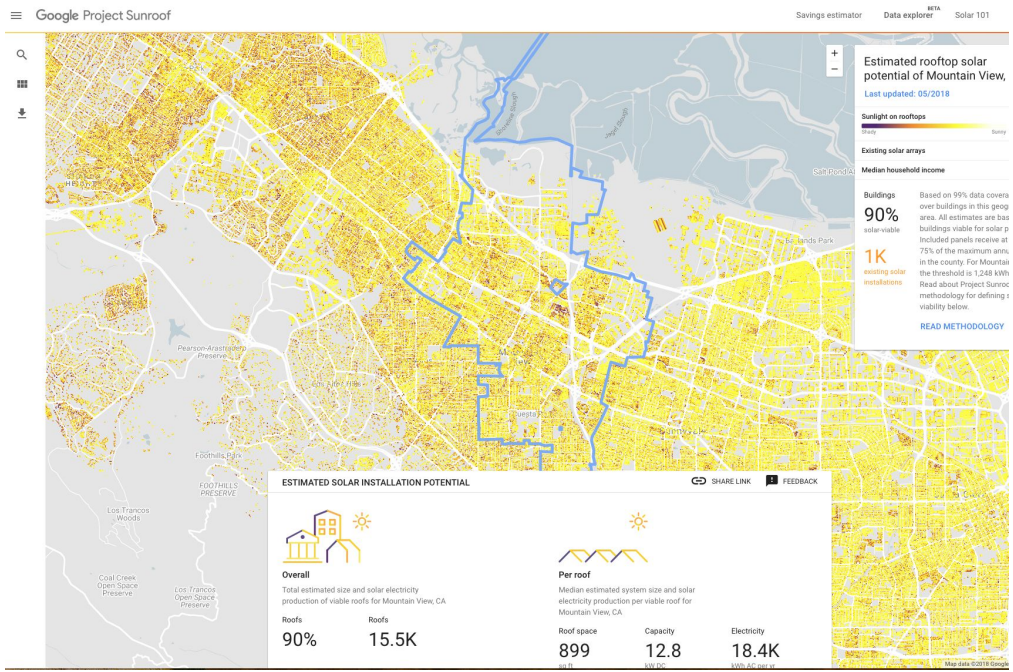
Sun Roof potential

from Satellite images

Example 1

SunRoof potential

[SunRoof](#) explorer



Example 1

Public dataset

Public Dataset: Solar_potential_by_postal_code.

Schema (+ sample records)

region_name	percent_covered	kw_total	carbon_offset_metric_tons
94043	97.79146031321109	215612.5	84929.00985071347
94041	99.05200433369447	56704.25	22189.34823862318

Example 1

SunRoof

On BigQueryPublic dataset

What is the solar potential of Mountain View, CA? [Run query]

Saved Query: MTV sunroof [edited]

```
#StandardSQL
SELECT
  region_name,
  percent_covered,
  kw_total,
  carbon_offset_metric_tons
FROM `bigquery-public-data.sunroof_solar.solar_potential_by_postal_code`
WHERE
  region_name = '94040'
  OR region_name = '94041'
  OR region_name = '94043'
```

Standard SQL Dialect

Ctrl + Enter: ru

RUN QUERY | Save Query | Save View | Format Query | Schedule Query | Show Options

Query complete (1.6s elapsed, 346 KB processed)

Results | Details | Download as CSV | Download as JSON | Save

Row	region_name	percent_covered	kw_total	carbon_offset_metric_tons
1	94043	97.79146031321109	215612.5	84929.00985071347
2	94041	99.05200433369447	56704.25	22189.34823862318
3	94040	98.9440337909187	139745.5	55039.74974407879

Example 1

Public dataset

Public Dataset: census_bureau_usa.population_by_zip_2010

Schema (+ sample records)

zipcode	population
99776	124
38305	49808
37086	31513
41667	720
67001	1676

Example 2

SunRoof

Public dataset
On BigQuery

How many metric tons of carbon would we offset, if building in communities with 100% coverage all had solar roofs? [Run query]

Saved Query: CO2 offset in 100percent zips

```
#StandardSQL
SELECT
  ROUND(SUM(s.carbon_offset_metric_tons),2) total_carbon_offset_possible_metric_tons
FROM `bigquery-public-data.sunroof_solar.solar_potential_by_postal_code` s
JOIN `bigquery-public-data.census_bureau_usa.population_by_zip_2010` c
ON s.region_name = c.zipcode
WHERE
  percent_covered = 100.0
  AND c.population > 0
```

Standard SQL Dialect

Ctrl + Enter: run qu

RUN QUERY | Save Query | Save View | Format Query | Schedule Query | Show Options | Query com

Results | Details | Download as CSV | Download as JSON | Save as

Row	total_carbon_offset_possible_metric_tons
1	3689508.33

Example 2

SunRoof

Public dataset
On BigQuery

How many metric tons of carbon would we offset, per zipcode?

Saved Query: CO2 offset in 100percent zips [edited]

```
#StandardSQL
SELECT
  zipcode, ROUND(SUM(s.carbon_offset_metric_tons),2) total_carbon_offset_possible_metri
FROM `bigquery-public-data.sunroof_solar.solar_potential_by_postal_code` s
JOIN `bigquery-public-data.census_bureau_usa.population_by_zip_2010` c
ON s.region_name = c.zipcode
WHERE
  percent_covered = 100.0
  AND c.population > 0
GROUP BY c.zipcode
```

Ctrl + Enter: run c

Standard SQL Dialect

RUN QUERY | Save Query | Save View | Format Query | Schedule Query | Show Options

Query complete (2.5s elapsed, 23.5 MB processed)

Results | Details | Download as CSV | Download as JSON | Save a

Row	zipcode	total_carbon_offset_possible_metric_tons
1	35119	3417.26
2	10165	162.1
3	21810	6650.09
4	74078	61515.66
5	47876	5544.3
6	10170	831.22

Example 2

SunRoof

How many metric tons of carbon would we offset, per zipcode sorted?

Saved Query: CO2 offset in 100percent zips [edited]

Query Editor

```
#StandardSQL
SELECT
  zipcode, ROUND(SUM(s.carbon_offset_metric_tons),2) total_carbon_offset_possible_metric_tons
FROM `bigquery-public-data.sunroof_solar.solar_potential_by_postal_code` s
JOIN `bigquery-public-data.census_bureau_usa.population_by_zip_2010` c
ON s.region_name = c.zipcode
WHERE
  percent_covered = 100.0
  AND c.population > 0
GROUP BY c.zipcode
ORDER BY total_carbon_offset_possible_metric_tons
DESC
```

Standard SQL Dialect

Ctrl + Enter: run query, Tab or Ctrl + Space

RUN QUERY | Save Query | Save View | Format Query | Schedule Query | Show Options

Query complete (2.8s elapsed, 23.5 MB processed)

Results | Details | Download as CSV | Download as JSON | Save as Table | Save to

Row	zipcode	total_carbon_offset_possible_metric_tons
1	18503	715700.55
2	44243	271861.55
3	38677	266787.12
4	96860	225850.35
5	47809	141087.91

Signup for GCP

SunRoof

Dear Students,

Here is the URL you will need to access in order to request a Google Cloud Platform coupon. You will be asked to provide your school email address and name. An email will be sent to you to confirm these details before a coupon is sent to you.

Student Coupon Retrieval Link

- You will be asked for a name and email address, which needs to match the domain. A confirmation email will be sent to you with a coupon code.
- You can request a coupon from the URL and redeem it until: 1/21/2019
- Coupon valid through: 9/21/2019
- You can only request ONE code per unique email address

1. Signup with stanford.edu email
2. You'll get a 50$ coupon in 1-2 minutes
3. Apply to your GMAIL account (signup if you don't have one), **NOT** Stanford account
4. RECOMMEND: separate browser profile

Signup for GCP

Google Cloud Platform Education Grants <cloudedugrants@google.com>

Reply all

Today, 8:49 PM

Shivakumar, Narayanan

To help protect your privacy, some content in this message has been blocked. To re-enable the blocked features, click here.

To always show content from this sender, click here.

Dear Shiva,

Here is your Google Cloud Platform Coupon Code: 0T8L-UFTY-8N7C-DYMP

Click [here] to redeem.

Course/Project Information
Instructor Name: Shiva Shivakumar
Email Address: shiva@cs.stanford.edu
School: Stanford
Course/project: CS 145
Activation Date: 9/21/2018
Redeem By: 1/21/2019
Coupon Valid Through: 9/21/2019

If you have any questions, please contact your course instructor as listed above.

Thanks,
Google Cloud Platform Education Grants Team

SQL Introduction & Definitions

What you will learn about in this section

1. What is SQL?
2. Basic schema definitions
3. Keys & constraints intro

SQL Motivation

Dark times 5 years ago.

- Are databases dead?

Now, as before: everyone sells SQL

- Pig, Hive, Impala

“Not-Yet-SQL?”

Basic SQL

SQL Introduction

- SQL is a standard language for querying and manipulating data

- SQL is a **very high-level** programming language
  This works because it is optimized well!
- Many standards out there:
  ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….

SQL stands for
Structured
Query
Language

NB: Probably the world's most successful **parallel** programming language (multicore?)

SQL is a...

- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables
- Data Definition Language (DDL)
  - Define relational schemata
  - Create/alter/delete tables and their attributes

Set algebra (reminder)

List: [1, 1, 2, 3]
Set: {1, 2, 3}
<u>Multiset</u>: {1, 1, 2, 3}

A **<u>multiset</u>** is an unordered list (or: a set with multiple duplicate instances allowed)

UNIONs

Set: {1, 2, 3} U { 2 } = { 1, 2, 3 }
<u>Multiset</u>: {1, 1, 2, 3} U { 2 } = { 1, 1, 2, 2, 3 }

Cross-product

{1, 1, 2, 3} * { y, z } =
{ <1, y>, <1, y>, <2, y>, <3, y>
<1, z>, <1, z>, <2, z>, <3, z>
}

i.e. no *next()*, etc. methods!

Tables in SQL

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

A **relation** or **table** is a multiset of tuples having the attributes specified by the schema

Let's break this definition down

Tables in SQL

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation

*NB: Attributes must have an **atomic** type in standard SQL, i.e. not a list, set, etc.*

Tables in SQL

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

A **tuple** or **row** is a single entry in the table having the attributes specified by the schema

*Also referred to sometimes as a **Record***

Tables in SQL

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

The number of tuples is the **cardinality** of the relation

The number of attributes is the **arity** of the relation

Data Types in SQL

Atomic types:

- Characters: CHAR(20), VARCHAR(50)
- Numbers: INT, BIGINT, SMALLINT, FLOAT
- Others: MONEY, DATETIME…

Every attribute must have an atomic type

- Hence tables are flat

Table Schemas

The **schema** of a table is the table name, its attributes, and their types:

Product(Pname: *string*, Price: *float*, Category: *string*, Manufacturer: *string*)

A **key** is an attribute whose values are unique; we underline a key

Product(<u>Pname</u>: *string*, Price: *float*, Category: *string*, <u>Manufacturer</u>: *string*)

Key constraints

A **key** is a **minimal subset of attributes** that acts as a unique identifier for tuples in a relation

- A key is an implicit constraint on which tuples can be in the relation
- i.e. if two tuples agree on the values of the key, then they must be the same tuple!

Students(sid:string, name:string, gpa: float)

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?

Declaring Schema

Students(sid: *string,* name: *string,* gpa: *float*)

```
CREATE TABLE Students (
  sid CHAR(20),
  name  VARCHAR(50),
  gpa  float,
  PRIMARY KEY (sid),
)
```

NULL and NOT NULL

- To say “don’t know the value” we use NULL
  NULL has (sometimes painful) semantics, more detail later

Students(sid:string, name:string, gpa: float)

sid	name	gpa
123	Bob	3.9
143	Jim	NULL

Say, Jim just enrolled in his first class.

In SQL, we may constrain a column to be NOT NULL,
e.g., “name” in this table

General Constraints

- We can actually specify arbitrary assertions

  E.g. "*There cannot be 25 people in the DB class*"

- In practice, we don't specify many such constraints. Why?

  <u>Performance!</u>

Whenever we do something ugly (or avoid doing something convenient) it's for the sake of performance

Summary of Schema Information

- Schema and Constraints are how databases understand the semantics (meaning) of data
- SQL supports general constraints:
  - Keys and foreign keys are most important
  - We'll give you a chance to write the others

2.Single - table queries

What you will learn about in this section

1. The SFW query
2. Other useful operators: LIKE, DISTINCT, ORDER BY

SQL Query

- Basic form (there are many many more bells and whistles)

```
SELECT <attributes>

FROM   <one or more relations>

WHERE  <conditions>
```

Call this a **SFW** query.

Simple SQL Query: Selection

Selection is the operation of filtering a relation's tuples on some condition

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

PName	Price	Category	Manuf
Gizmo	$19.99	Gadgets	GWorks
Powergizmo	$29.99	Gadgets	GWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

PName	Price	Category	Manuf
Gizmo	$19.99	Gadgets	GWorks
Powergizmo	$29.99	Gadgets	GWorks

Simple SQL Query: Projection

Projection is the operation of producing an output table with tuples that have a subset of their prior attributes

PName	Price	Category	Manuf
Gizmo	$19.99	Gadgets	GWorks
Powergizmo	$29.99	Gadgets	GWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT Pname, Price, Manufacturer
FROM   Product
WHERE  Category = 'Gadgets'
```

PName	Price	Manuf
Gizmo	$19.99	GWorks
Powergizmo	$29.99	GWorks

Notation

Input Schema

Product(PName, Price, Category, Manufacturer)

```
SELECT Pname, Price, Manufacturer
FROM   Product
WHERE  Category = 'Gadgets'
```

Output Schema

Answer(PName, Price, Manfacturer)

A Few Details

- SQL **commands** are case insensitive:

  Same: SELECT, Select, select

  Same: Product, product

- **Values** are **not:**

  <u>Different:</u> ‘Seattle’, ‘seattle’

- Use single quotes for constants:

  ‘abc’ - yes

  “abc” - no

LIKE: Simple String Pattern Matching

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

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category
FROM   Product
```

Category
Gadgets
Photography
Household

Versus

```
SELECT Category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

ORDER BY: Sorting the Results

```
SELECT      PName, Price, Manufacturer
FROM        Product
WHERE       Category='gizmo' AND Price > 50
ORDER BY    Price, PName
```

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

Ordering is ascending, unless you specify the DESC keyword.

Multi-table queries

What you will learn about in this section

1. Foreign key constraints
2. Joins: basics
3. Joins: SQL semantics

Foreign Key constraints

- Suppose we have the following schema :

Students(sid: *string*, name: *string*, gpa: *float*)

Enrolled(student_id: *string*, cid: *string*, grade: *string*)

- And we want to impose the following constraint:

Only bona fide students may enroll in courses' i.e. a student must appear in the Students table to enroll in a class

Students

sid	name	gpa
102	**Bob**	**3.9**
123	**Mary**	**3.8**

Enrolled

Student _id	cid	grade
123	564	A
123	537	A+

We say that student_id is a foreign key that refers to Students

Declaring Foreign Keys

Students(sid: *string,* name: *string,* gpa: *float*)
Enrolled(student_id: *string*, cid: *string,* grade: *string*)

```
CREATE TABLE Enrolled (
  student_id CHAR(20),
  cid  CHAR(20),
  grade  CHAR(10),
  PRIMARY KEY (student_id, cid),
  FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

Foreign Keys and update operations

Students(sid: *string*, name: *string*, gpa: *float*)

Enrolled(student_id: *string*, cid: *string*, grade: *string*)

- What if we insert a tuple into Enrolled, but no corresponding student?
  INSERT is rejected (foreign keys are constraints)!

- What if we delete a student?
  1. Disallow the delete
  2. Remove all of the courses for that student
  3. *SQL allows a third via NULL (not yet covered)*

DBA chooses

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is a foreign key vs. a key here?

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

THANK YOU!