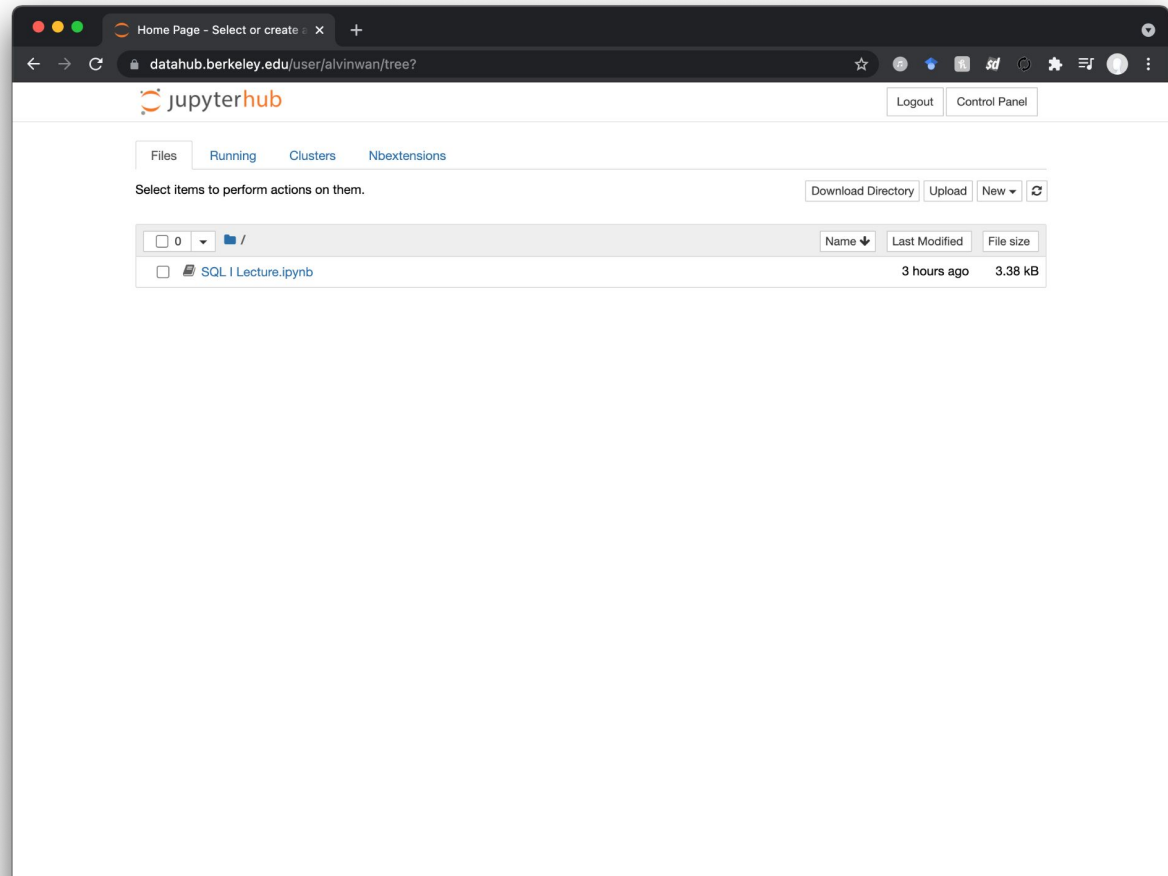
LECTURE 8

Create Tables with SQL Queries

Creating your first Table

Data 100/Data 200, Fall 2021 @ UC Berkeley
Fernando Pérez and Alvin Wan
(content by Alvin Wan, Anthony D. Joseph, Allen Shen, Josh Hug, John DeNero, Joseph Gonzalez)

datahub.berkeley.edu



SQL I Lecture - Jupyter Notebo

datahub.berkeley.edu/user/alvinwan/notebooks/SQL%20I%

jupyterhub SQL I Lecture Last Checkpoint: La

File Edit View Insert Cell Kernel Widgets

Run Cod

```
In [2]: %load_ext sql

In [ ]:
```

Constraints

Composite Key

More Constraints

Animal

type TEXT, PK	legs INTEGER, >=0	weight INTEGER, >=0
Corgi	4	10
T-Rex	2	12000
Penguin	2	10

```
CREATE TABLE Animal (
  type      TEXT        PRIMARY KEY,
  legs      INTEGER     CHECK (legs >= 0),
  weight    INTEGER     CHECK (weight >= 0)
);
DROP TABLE Animal;
```

PRACTICAL TIP

Put each column schema in a new line.

Not mandatory but improves readability.

Constraints Example

Composite Key Example

More Constraints

Member

athlete TEXT, PK	esport TEXT, PK	skill INTEGER
Danny	Warzone	10
Jane	Starcraft	1000
Jane	Warzone	100000

```
CREATE TABLE Member (
  athlete  TEXT,
  esport   TEXT,
  skill    INTEGER,
  PRIMARY KEY(athlete, esport)
);
```

PRACTICAL TIP

Use `IF EXISTS` when testing table creation queries.

Member

athlete TEXT, PK	esport TEXT, PK	skill INTEGER
Danny	Warzone	10
Jane	Starcraft	1000
Jane	Warzone	100000

```
DROP TABLE IF EXISTS Member;
CREATE TABLE Member (
  athlete  TEXT,
  esport   TEXT,
  skill    INTEGER,
  PRIMARY KEY(athlete, esport)
);
```

Member

athlete TEXT, PK	esport TEXT, PK	skill INTEGER
Danny	Warzone	10
Jane	Starcraft	1000
Jane	Warzone	100000

```
CREATE TABLE IF EXISTS Member (
  athlete  TEXT,
  esport   TEXT,
  skill    INTEGER,
  PRIMARY KEY(athlete, esport)
);
```

Constraints Example

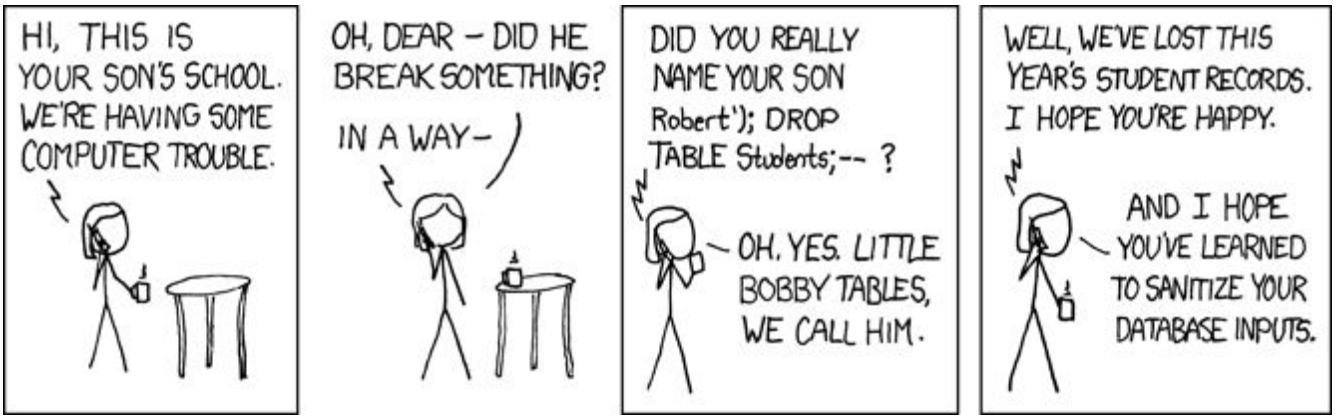
Composite Key Example

More Constraints

Clothing

id INT, PK, AUTOINC	sku TEXT, UNIQUE	name TEXT, NOT NULL
1	92183	blouse
2	23012	jeans
3	57603	polo

```
DROP TABLE IF EXISTS Clothing;
CREATE TABLE Clothing (
  id    INT     PRIMARY KEY AUTOINCREMENT,
  sku   TEXT    UNIQUE,
  name  TEXT    NOT NULL
);
```



xkcd.com/327/

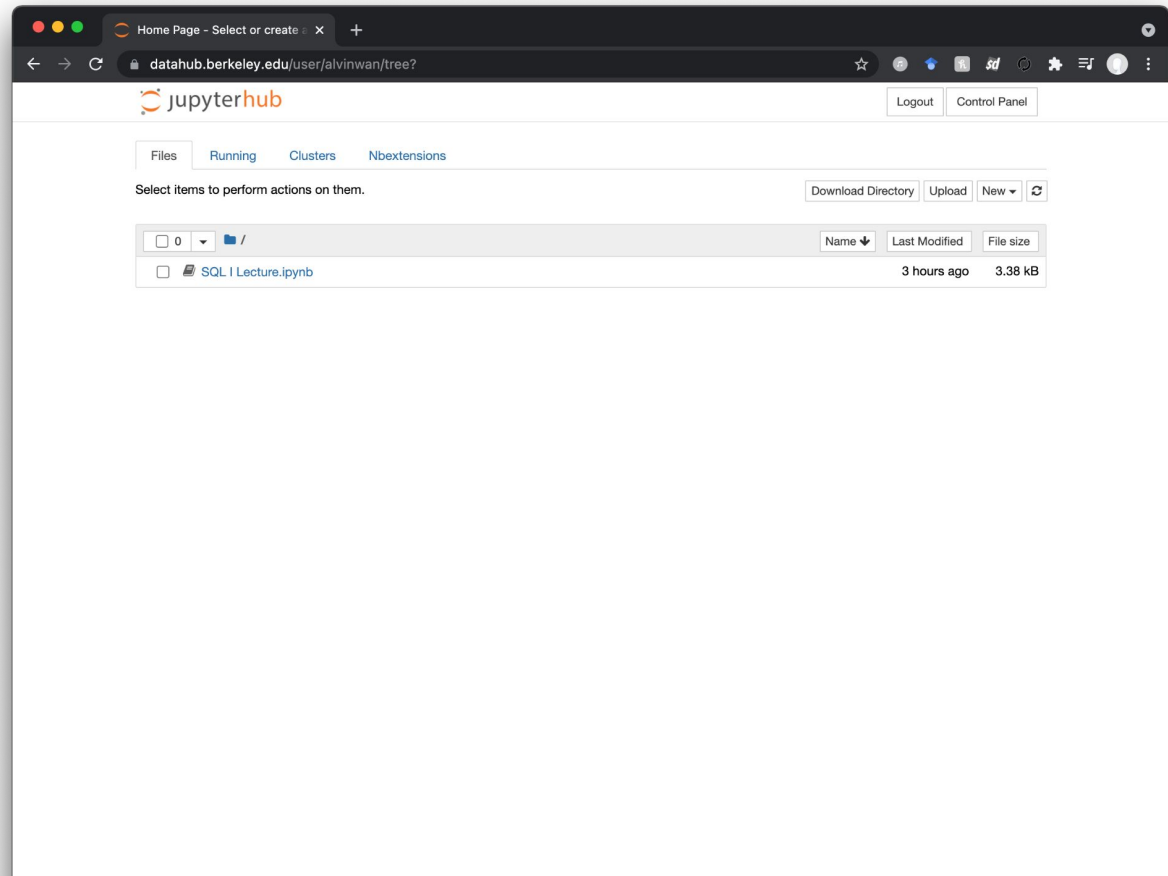
LECTURE 8

Query Rows with SQL

How to filter, sort, paginate, and subsample your data

Data 100/Data 200, Fall 2021 @ UC Berkeley
Fernando Pérez and Alvin Wan
(content by Alvin Wan, Anthony D. Joseph, Allen Shen, Josh Hug, John DeNero, Joseph Gonzalez)

datahub.berkeley.edu
notebook



SQL I Lecture - Jupyter Notebo

datahub.berkeley.edu/user/alvinwan/notebooks/SQL%20I%

jupyterhub SQL I Lecture Last Checkpoint: La

File Edit View Insert Cell Kernel Widgets

Run Cod

```
In [2]: %load_ext sql

In [ ]:
```

Dragon

name TEXT, PK	year INT, >=2000	cute INT, NOT NULL
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

```
DROP TABLE IF EXISTS Dragon;
CREATE TABLE Dragon (
  name TEXT PRIMARY KEY,
  year INTEGER CHECK(year >= 2000),
  cute INTEGER,
);
```

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	**year** INTEGER, >=2000	**cute** INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

table

```
INSERT INTO Dragon
VALUES ('hiccup',2010,10);
```

values to insert. 1 per column

PRACTICAL TIP

`INSERT INTO` can insert multiple rows at once.

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

all columns

```
SELECT * FROM Dragon;
```

table

PRACTICAL TIP

Put each clause (e.g., `FROM`) on a new line.

Not mandatory but improves readability.

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

columns

```
SELECT name, year
FROM Dragon;
```

table

NOTE

`AS` allows you to rename columns and tables.

Not just for prettification.
(We will see later)

Dragon

name TEXT, PK	**year** INTEGER, >=2000	**cute** INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

column alias

```
SELECT year AS birth
FROM Dragon AS Bird;
```

table alias

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

```
SELECT *
FROM
    Dragon AS Dragon1,
    Dragon AS Dragon2;
```

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

```
SELECT
    Dragon1.name,
    Dragon2.name
FROM
    Dragon AS Dragon1,
    Dragon AS Dragon2;
```

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

columns

```
SELECT name, year
FROM Dragon
WHERE cute > 0;
```

table

condition

QUICK CHECK

Design a table and query to find all pairs of whole numbers that sum to 5.

n

id
0
1
2
3
4
5
6

```
SELECT *
FROM n AS a, n AS b
WHERE a + b = 5;
```

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

all columns

```
SELECT *
FROM Dragon
ORDER BY cute desc;
```

table

column or asc

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

all columns

```
SELECT *
FROM Dragon
LIMIT 2;
```

table

number of result rows

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

all columns

```
SELECT *
FROM Dragon
LIMIT 2
OFFSET 1;
```

table

where to start

Insert Into

Select From

Pairs

Filter

Sort

Paginate

Subsample

Dragon

name TEXT, PK	year INTEGER, >=2000	cute INTEGER
hiccup	2010	10
drogon	2011	-100
Dragon 2	2019	0

all columns

```
SELECT *
FROM Dragon
ORDER BY RANDOM()
LIMIT 2;
```

table

shuffle

subsample size

WARNING: Could be slow! Revisit later.

Insert Into
Select From
Pairs
Filter
Sort
Paginate
Subsample

TAKEAWAY

SQL is powerful. With SQL, you can filter, sort, paginate, and subsample.

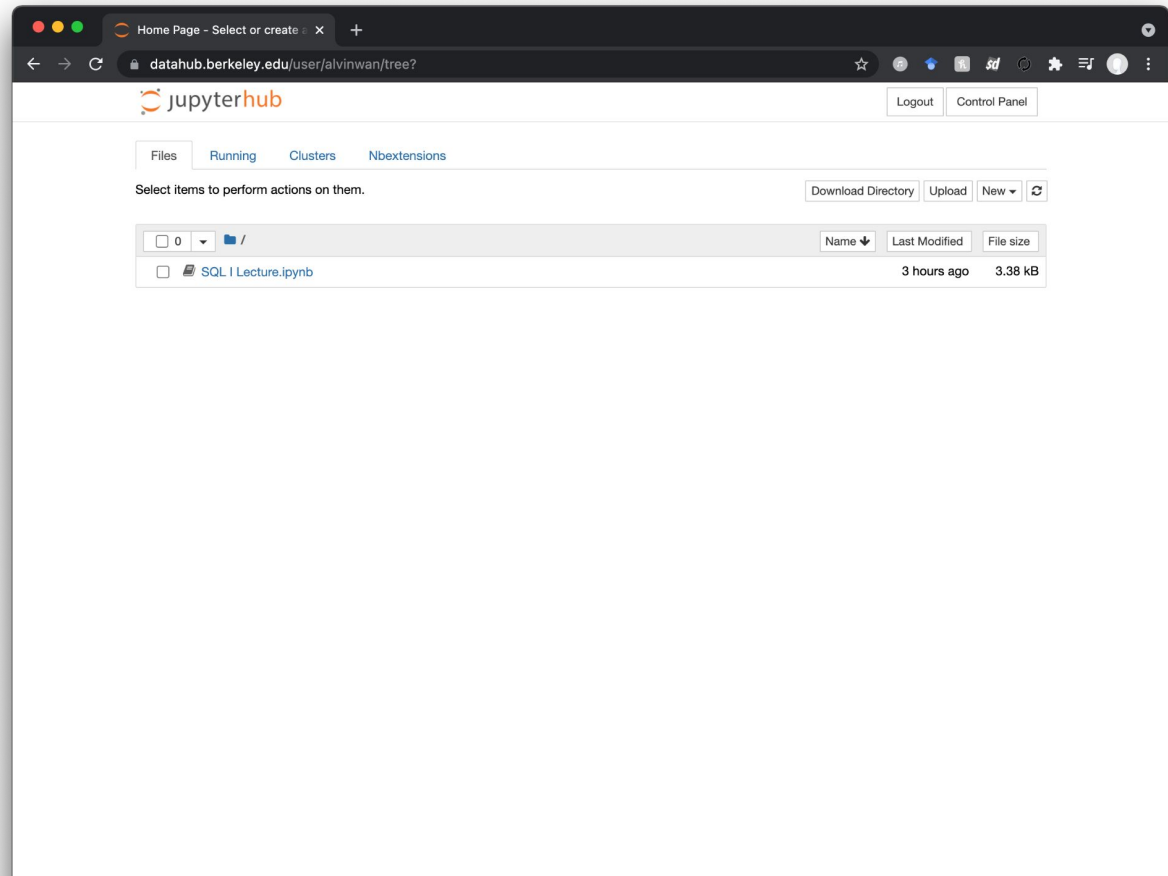
LECTURE 8

Query Groups with SQL

Organizing your data

Data 100/Data 200, Fall 2021 @ UC Berkeley
Fernando Pérez and Alvin Wan
(content by Alvin Wan, Anthony D. Joseph, Allen Shen, Josh Hug, John DeNero, Joseph Gonzalez)

datahub.berkeley.edu
notebook



SQL I Lecture - Jupyter Notebo

datahub.berkeley.edu/user/alvinwan/notebooks/SQL%20I%

jupyterhub SQL I Lecture Last Checkpoint: La

File Edit View Insert Cell Kernel Widgets

Run Cod

```
In [2]: %load_ext sql

In [ ]:
```

Dish

name TEXT, PK	type TEXT	cost INTEGER, >=0
ravioli	entree	10
pork bun	entree	7
taco	entree	7
edamame	appetizer	4
fries	appetizer	4
potsticker	appetizer	4
ice cream	dessert	5

```
CREATE TABLE Dish (
  name TEXT PRIMARY KEY,
  type TEXT,
  cost INTEGER CHECK (cost >= 0),
);
```

Dish

name **TEXT, PK**	**type** **TEXT**	**cost** **INTEGER, >=0**
ravioli	entree	10
pork bun	entree	7
taco	entree	7
edamame	appetizer	4
fries	appetizer	4
potsticker	appetizer	4
ice cream	dessert	5

```
INSERT INTO Dish
VALUES
    ('ravioli', 'entree', 10),
    ('pork bun', 'entree', 7),
    …
    ('ice cream', 'dessert', 5);
```

Group By

Aggregate Groups

Distinct

Filter Groups

Dish

name **TEXT, PK**	**type** **TEXT**	**cost** **INTEGER, >=0**
ravioli	entree	10
pork bun	entree	7
taco	entree	7
edamame	appetizer	4
fries	appetizer	4
potsticker	appetizer	4
ice cream	dessert	5

```
SELECT type
  FROM Dish
  GROUP BY type;
```

Group By

Aggregate Groups

Distinct

Filter Groups

```
SELECT COUNT(*)
FROM Dish;
```

~~SELECT type, cost~~
~~FROM Dish~~
~~GROUP BY type;~~

PRACTICAL TIP

Make sure group queries reference only either the grouped column or aggregated data.

```
SELECT type, SUM(cost)
FROM Dish
GROUP BY type;
```

```
SELECT type, SUM(cost)
COUNT(cost)
FROM dish
GROUP BY type
```

```
SELECT type, AVG(cost)
FROM dish
GROUP BY type
```

Group By

Aggregate Groups

Distinct

Filter Groups

Dish

name **TEXT, PK**	**type** **TEXT**	**cost** **INTEGER, >=0**
ravioli	entree	10
pork bun	entree	7
taco	entree	7
edamame	appetizer	4
fries	appetizer	4
potsticker	appetizer	4
ice cream	dessert	5

```
SELECT DISTINCT prices
  FROM Dish;
```

```
SELECT COUNT(DISTINCT
prices)
  FROM Dish;
```

Group By

Aggregate Groups

Distinct

Filter Groups

Dish

name **TEXT, PK**	**type** **TEXT**	**cost** **INTEGER, >=0**
ravioli	entree	10
pork bun	entree	7
taco	entree	7
edamame	appetizer	4
fries	appetizer	4
potsticker	appetizer	4
ice cream	dessert	5

```
SELECT COUNT(*)
  FROM Dish
  GROUP BY type
  HAVING cost > 5;
```

PRACTICAL TIP

HAVING filters groups.
WHERE filters rows.

Group By
Aggregate Groups
Distinct
Filter Groups

TAKEAWAY

With SQL, you can extract, aggregate, and filter groups.

LECTURE 8

Practice

Write and Debug SQL Queries

Data 100/Data 200, Fall 2021 @ UC Berkeley
Fernando Pérez and Alvin Wan
(content by Alvin Wan, Anthony D. Joseph, Allen Shen, Josh Hug, John DeNero, Joseph Gonzalez)

Basic Queries

Advanced Queries

Debug Queries

Create schema for this table.

Scene

id	biome	city	visitors	created_at
INT, PK	TEXT, NOT NULL	TEXT, NOT NULL	INT, >=0	DATETIME

Populate table with the following data.

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0	**created_at** DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Count the number of rows in the table.

Scene

id INT, PK	biome TEXT, NOT NULL	city TEXT, NOT NULL	visitors INT, >=0	created_at DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Basic Queries

Advanced Queries

Debug Queries

Find the most popular city for each biome.

Scene

id INT, PK	biome TEXT, NOT NULL	city TEXT, NOT NULL	visitors INT, >=0	created_at DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Find the biome with the most cities.

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0	**created_at** DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Find the pair of scenes with the most visitors.

Scene

id INT, PK	biome TEXT, NOT NULL	city TEXT, NOT NULL	visitors INT, >=0	created_at DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Basic Queries

Useful Queries

Debug Queries

Find all scenes with at least 125 visitors. Where's the bug?

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0	**created_at** DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Find all cities with at least 125 visitors. Where's the bug?

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0	**created_at** DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

Find the least populated city.
Where's the bug?

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0	**created_at** DATETIME
0	desert	Las Vegas	100	2021-01-01
1	marine	Honolulu	1000	2021-01-02
2	freshwater	Paris	50	2021-01-05
3	marine	Taipei	100	2021-01-07
4	desert	Austin	25	2021-01-12
5	freshwater	Austin	240	2021-01-15
6	freshwater	Las Vegas	100	2021-01-15

TAKEAWAY

Get familiar with SQL so you know what questions you can answer, and how to do it.

LECTURE 8

Practical Demo

Search NYC menus

Data 100/Data 200, Fall 2021 @ UC Berkeley
Fernando Pérez and Alvin Wan
(content by Alvin Wan, Anthony D. Joseph, Allen Shen, Josh Hug, John DeNero, Joseph Gonzalez)

Predicates

Casting

String Concatenation

Pandas