LECTURE 9

SQL II

Relational Databases and Querying Multiple Tables

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)

LECTURE 9

Motivation

Normalization Definition and Takeaways

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)

What & Why

Takeaways

How to Try SQL

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0
1	desert	Las Vegas	100
2	freshwater	Honolulu	1000
3	freshwater	Las Vegas	100

Scene

id INT, PK	biome TEXT, NOT NULL	city TEXT, NOT NULL	city radius INT, >= 0	visitors INT, >=0
1	desert	Las Vegas	15	100
2	freshwater	Honolulu	12	1000
3	freshwater	Las Vegas	15	100

Why **redundancy** is a **bad** idea.

More Storage

Slower **`INSERT`**

Possible incongruities

Scene

id INT, PK	biome TEXT, NOT NULL	city TEXT, NOT NULL	city radius INT, >= 0	visitors INT, >=0
1	desert	Las Vegas	15	100
2	freshwater	Honolulu	12	1000
3	freshwater	Las Vegas	15	100

Scene

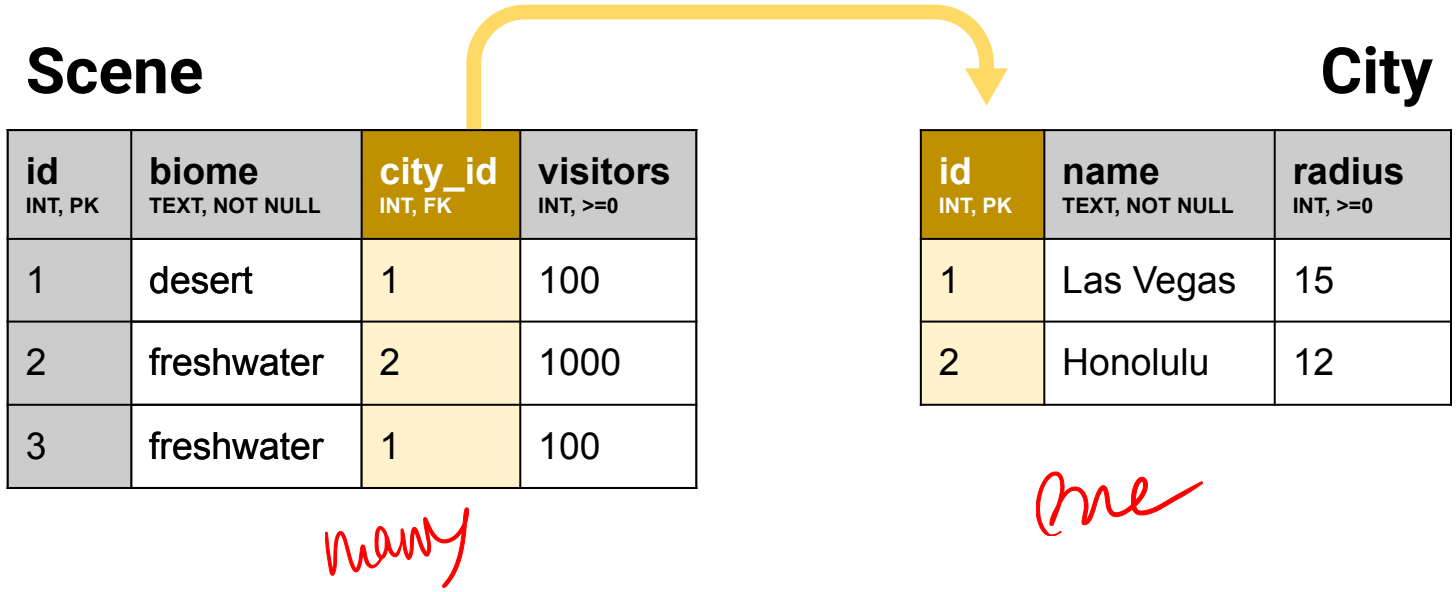
id INT, PK	biome TEXT, NOT NULL	city TEXT, NOT NULL	city radius INT, >= 0	visitors INT, >=0
1	desert	Las Vegas	15	100
2	freshwater	Honolulu	12	1000
3	freshwater	Las Vegas	15	100

Scene

id INT, PK	biome TEXT, NOT NULL	city_id INT, FK	visitors INT, >=0
1	desert	1	100
2	freshwater	2	1000
3	freshwater	1	100

City

id INT, PK	name TEXT, NOT NULL	radius INT, >=0
1	Las Vegas	15
2	Honolulu	12


Scene
id
INT, PK
biome
TEXT, NOT NULL
city_id
INT, FK
visitors
INT, >=0
1
desert
1
100
2
freshwater
2
1000
3
freshwater
1
100
many
City
id
INT, PK
name
TEXT, NOT NULL
radius
INT, >=0
1
Las Vegas
15
2
Honolulu
12
one

Normalization is a database design technique to remove redundancy and anomalies.

Why **normalization** is a **good** idea.

More Compact

Faster **`INSERT`**

Impossible to have incongruities

Minimize redesign need

What & Why

Takeaways

How to Try SQL

What you should focus on.

Not memorize *SQL syntax*

Understand what JOINs can do

Write clean SQL JOINs

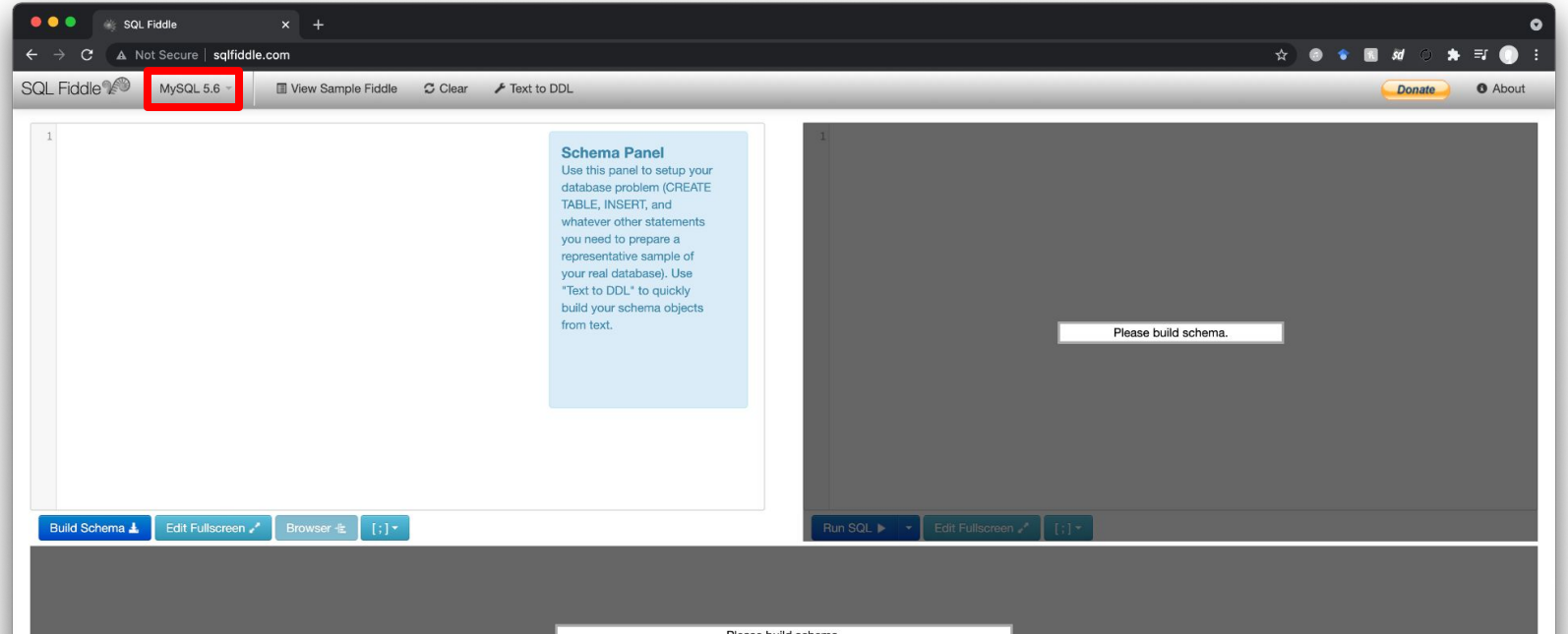
Debug SQL JOINs

What & Why

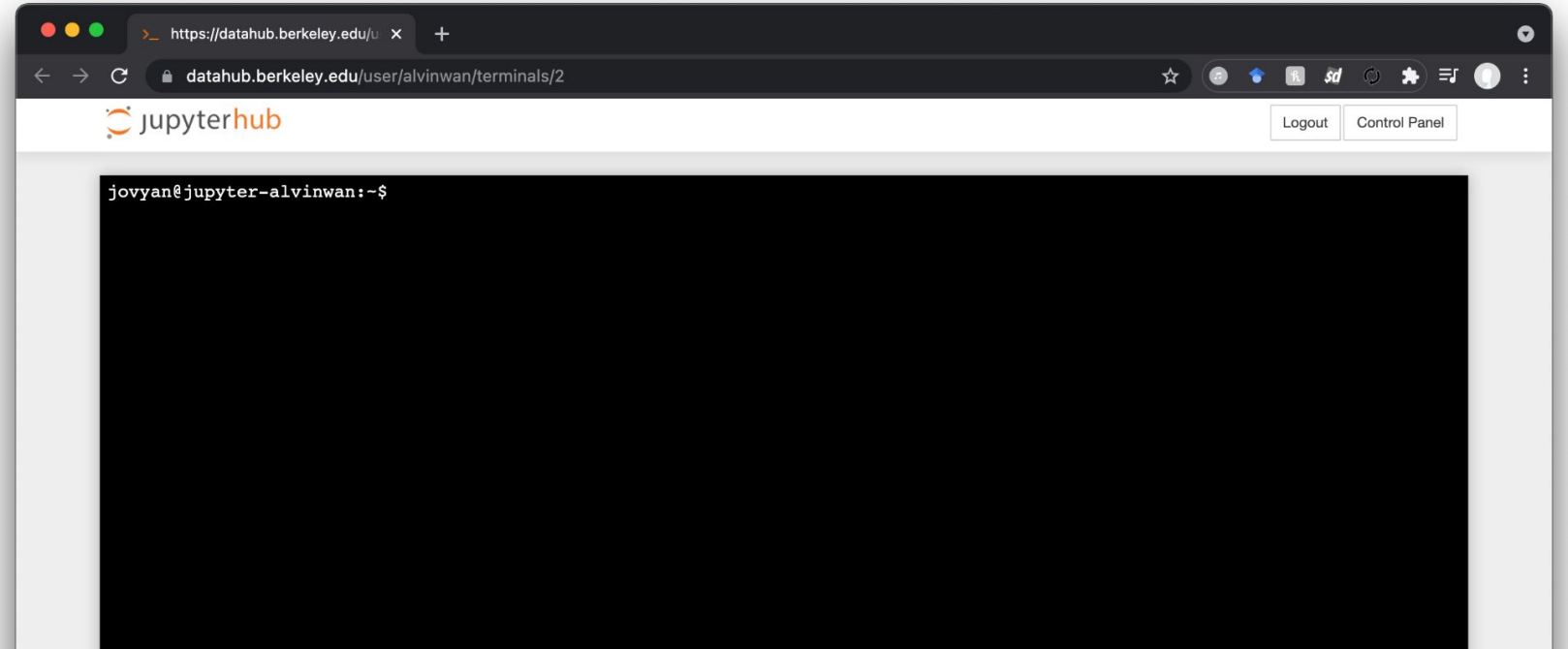
Takeaways

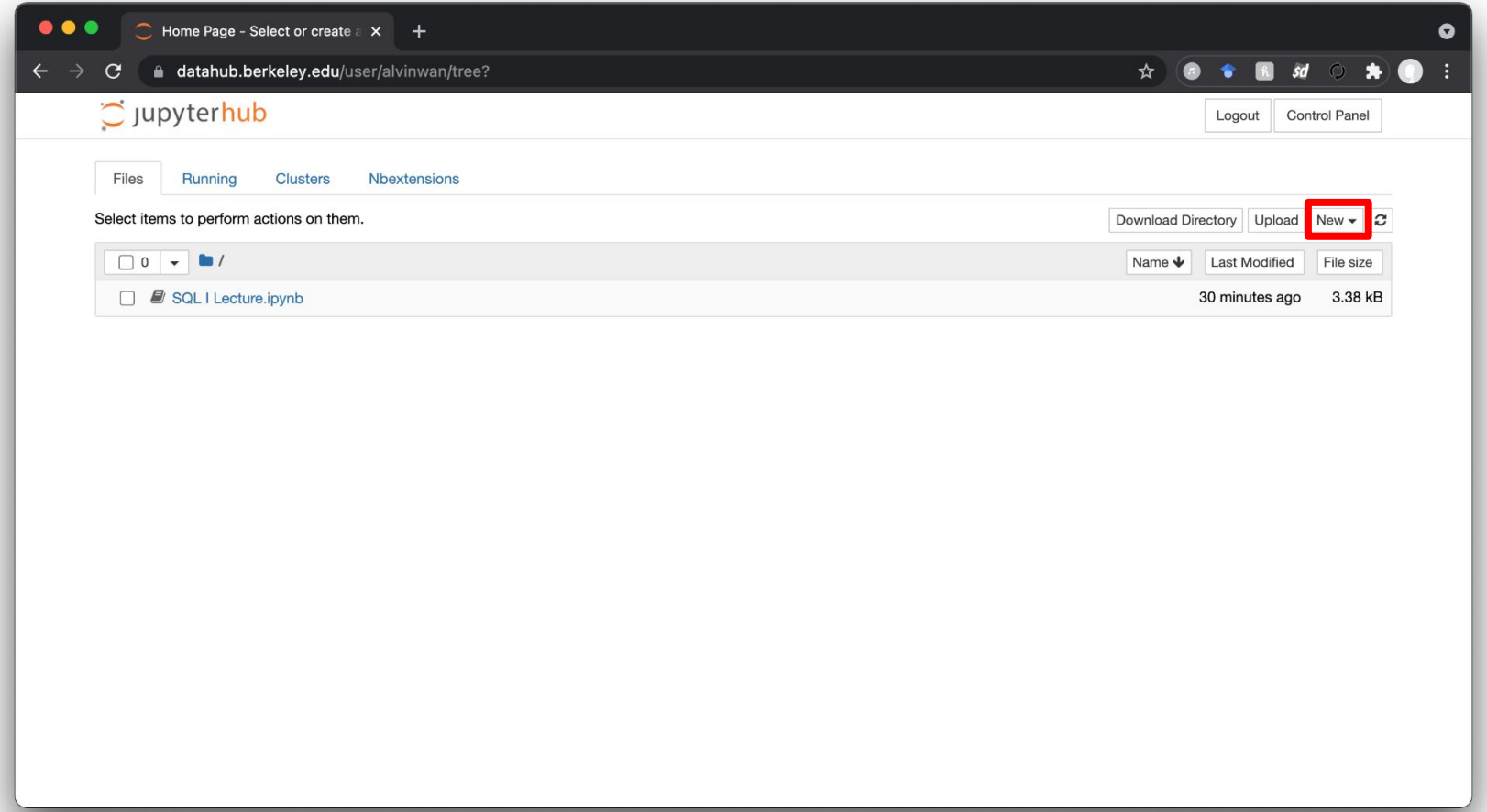
How to Try SQL

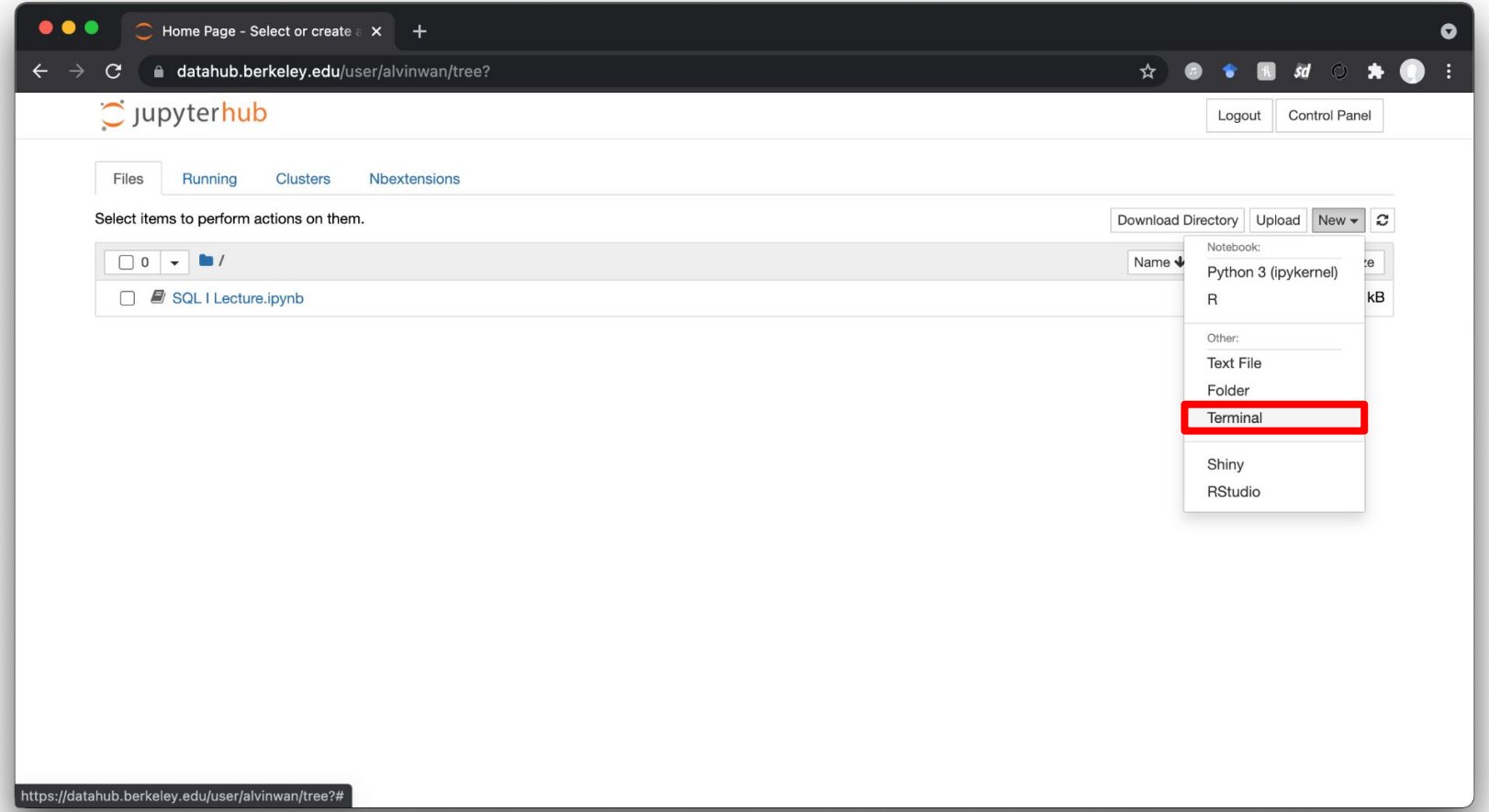
sqlfiddle.com

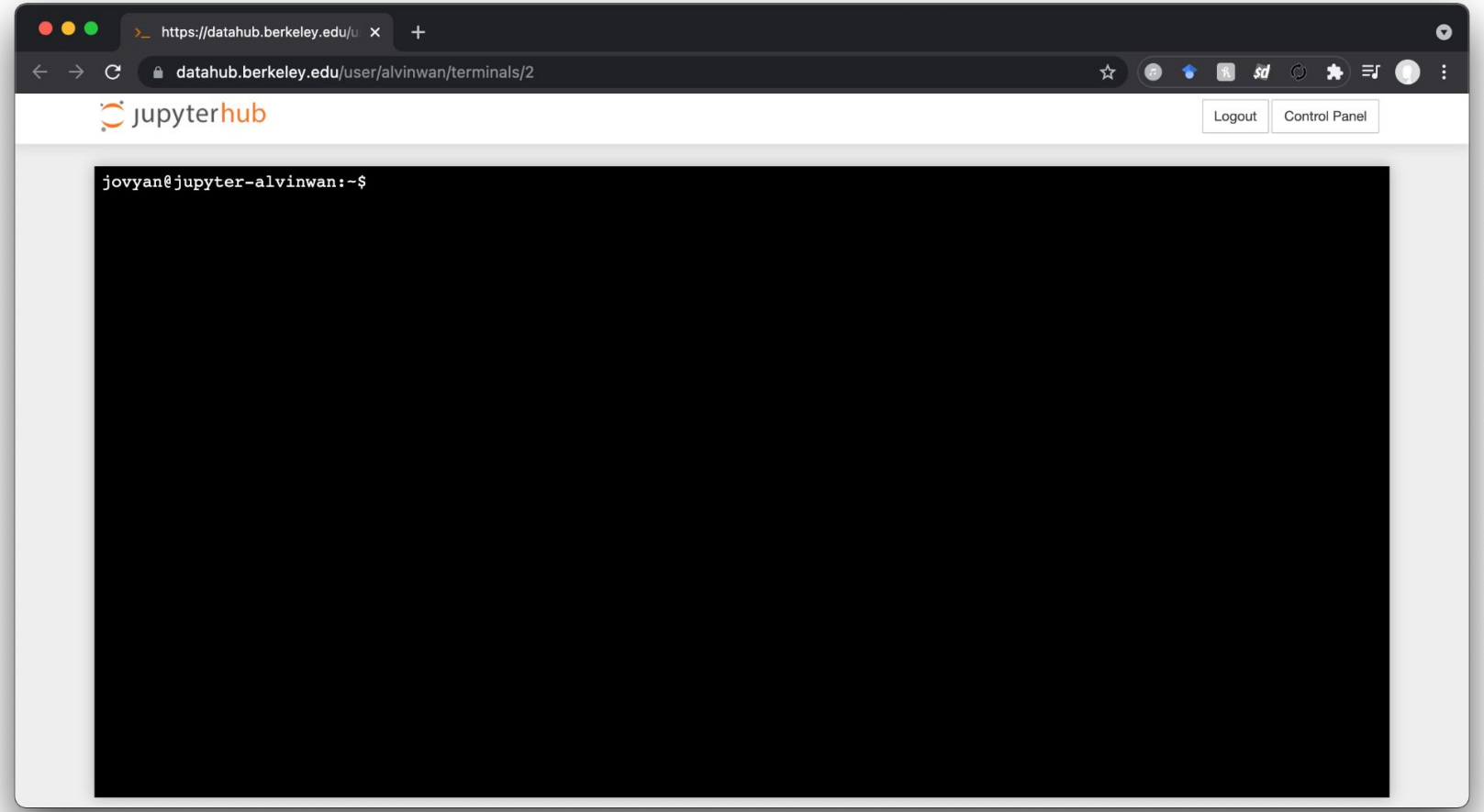


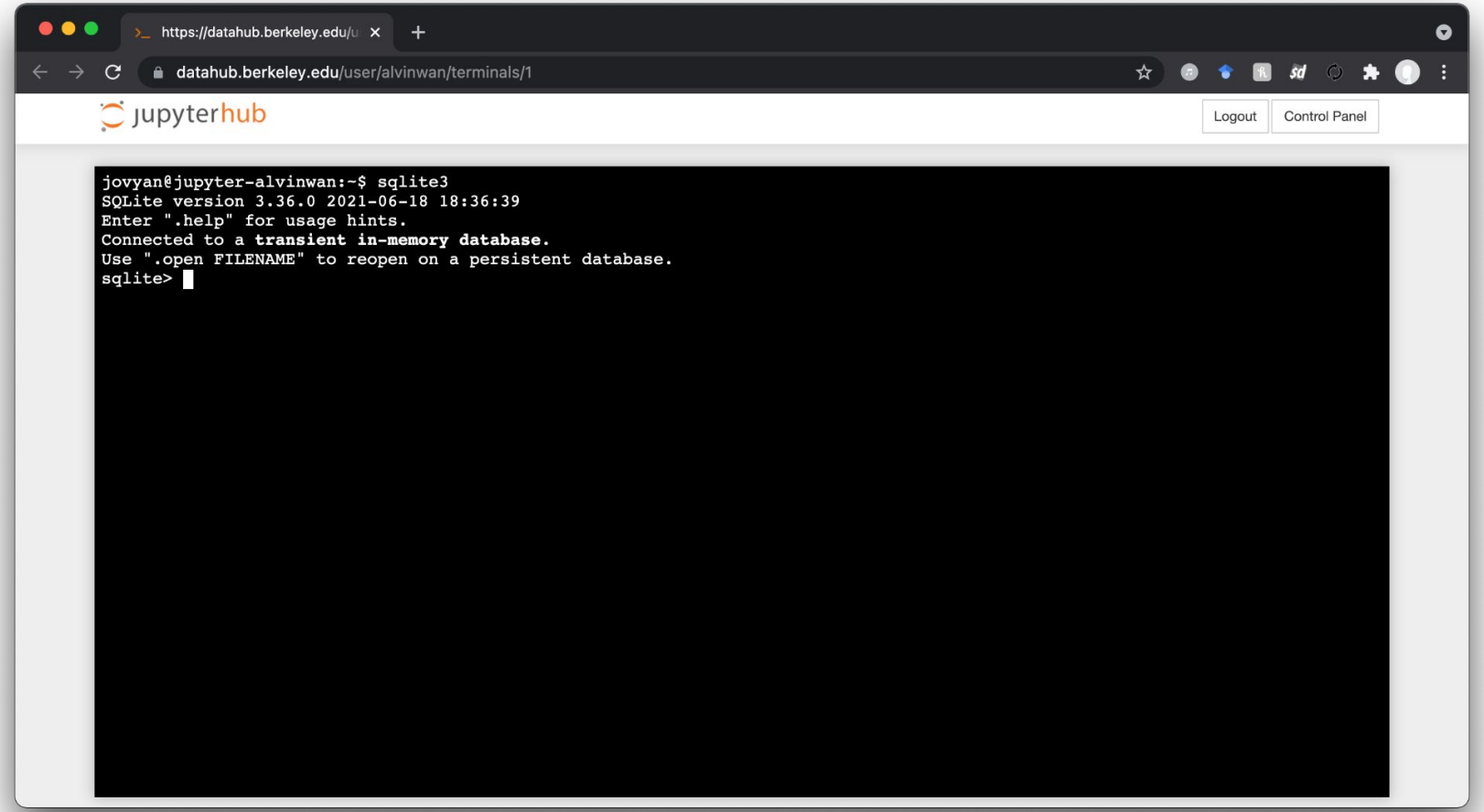
datahub.berkeley.edu terminal



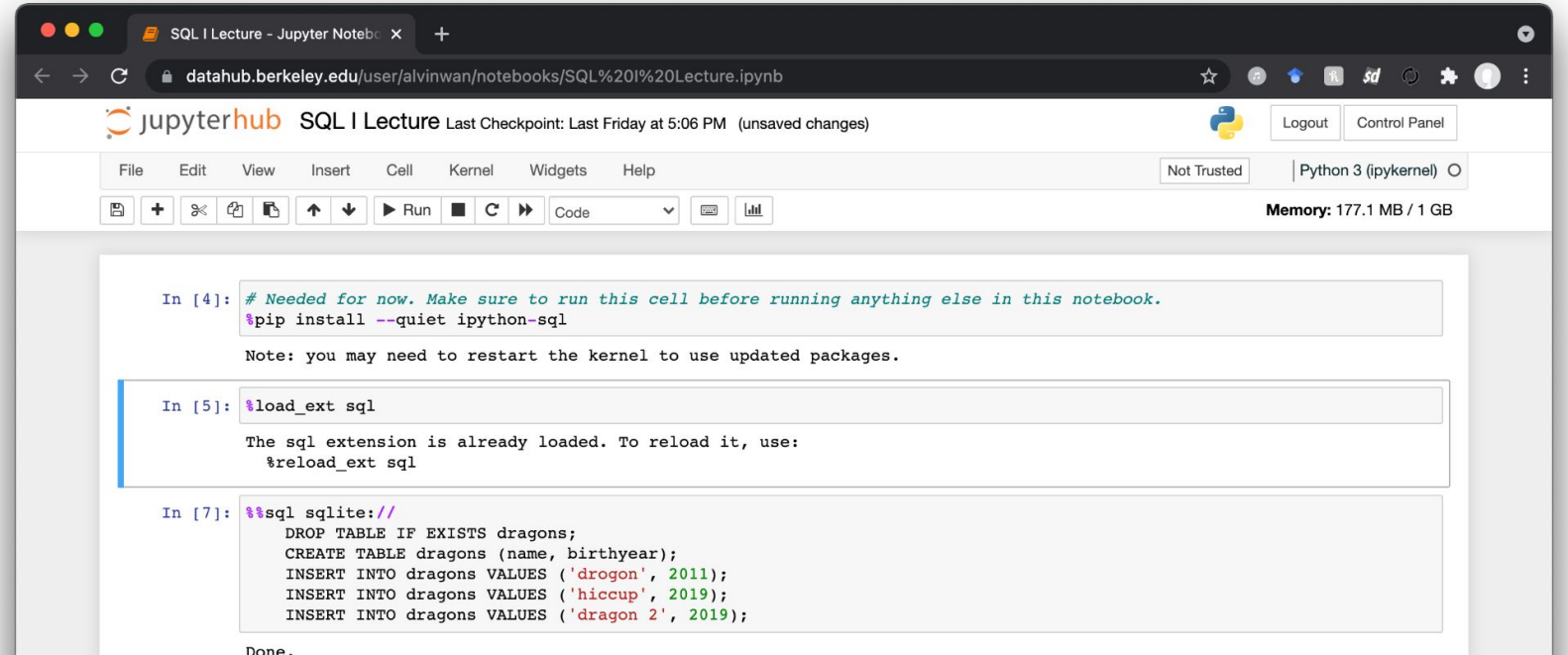

Home Page - Select or create
datahub.berkeley.edu/user/alvinwan/tree?
jupyterhub
Logout
Control Panel
Files
Running
Clusters
Nbextensions
Select items to perform actions on them.
Download Directory
Upload
New
0
/
Name
Last Modified
File size
SQL I Lecture.ipynb
30 minutes ago
3.38 kB

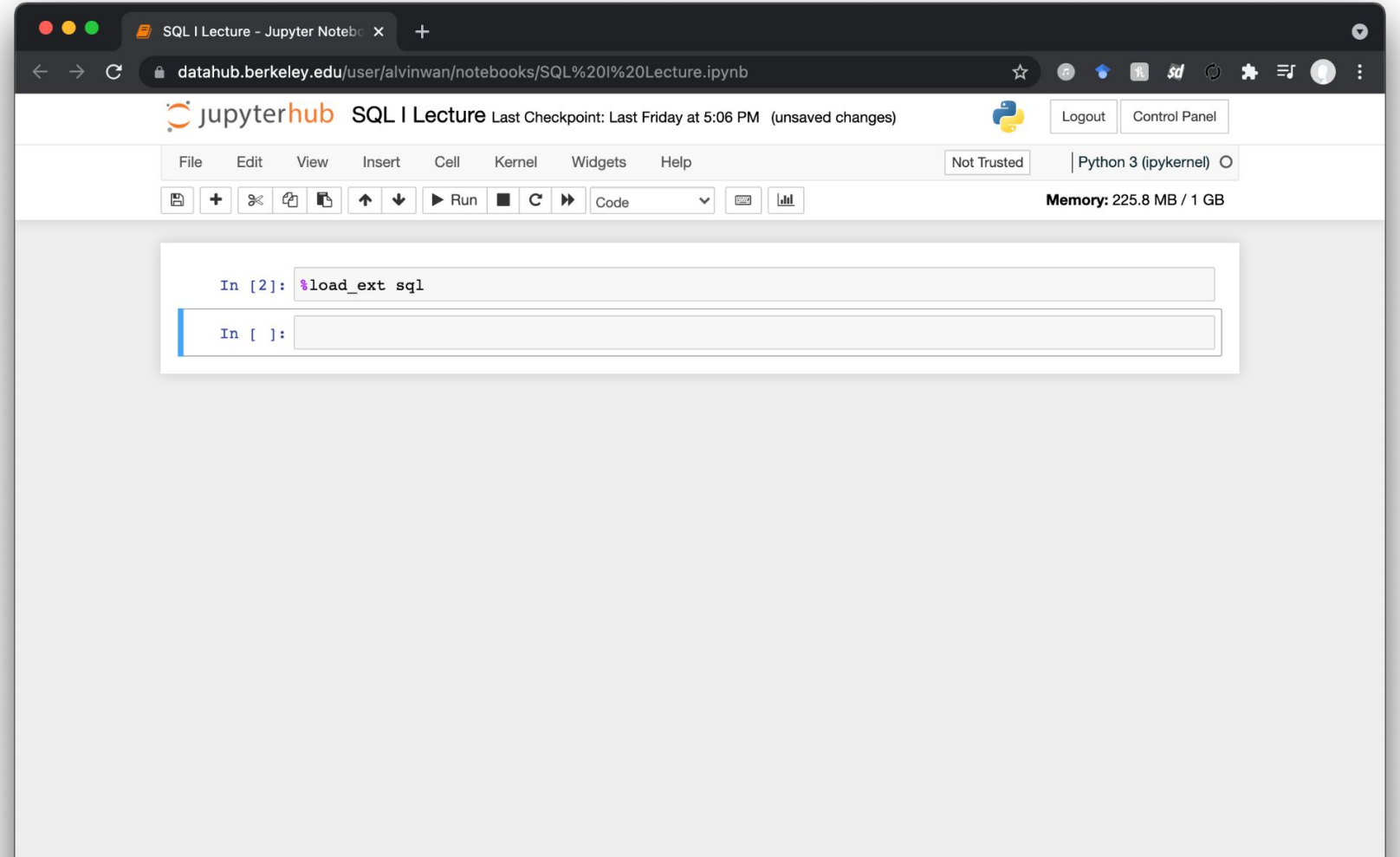

Home Page - Select or create a
datahub.berkeley.edu/user/alvinwan/tree?
jupyterhub
Logout
Control Panel
Files
Running
Clusters
Nbextensions
Select items to perform actions on them.
Download Directory
Upload
New
0
/
Name
SQL I Lecture.ipynb
kB
Notebook:
Python 3 (ipykernel)
R
Other:
Text File
Folder
Terminal
Shiny
RStudio
https://datahub.berkeley.edu/user/alvinwan/tree?#


https://datahub.berkeley.edu/u
datahub.berkeley.edu/user/alvinwan/terminals/2
jupyterhub
Logout
Control Panel
jovyan@jupyter-alvinwan:~$


https://datahub.berkeley.edu/u
datahub.berkeley.edu/user/alvinwan/terminals/1
jupyterhub
Logout
Control Panel
jovyan@jupyter-alvinwan:~$ sqlite3
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

datahub.berkeley.edu notebook




SQL I Lecture - Jupyter Notebo
datahub.berkeley.edu/user/alvinwan/notebooks/SQL%20I%20Lecture.ipynb
jupyterhub
SQL I Lecture
Last Checkpoint: Last Friday at 5:06 PM (unsaved changes)
Logout
Control Panel
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Memory: 225.8 MB / 1 GB
In [2]: %load_ext sql
In [ ]:

TAKEAWAY

Normalization **removes redundancy** and **anomalies** in your data. **Foreign keys** refer to **private keys** in a different table.

LECTURE 9

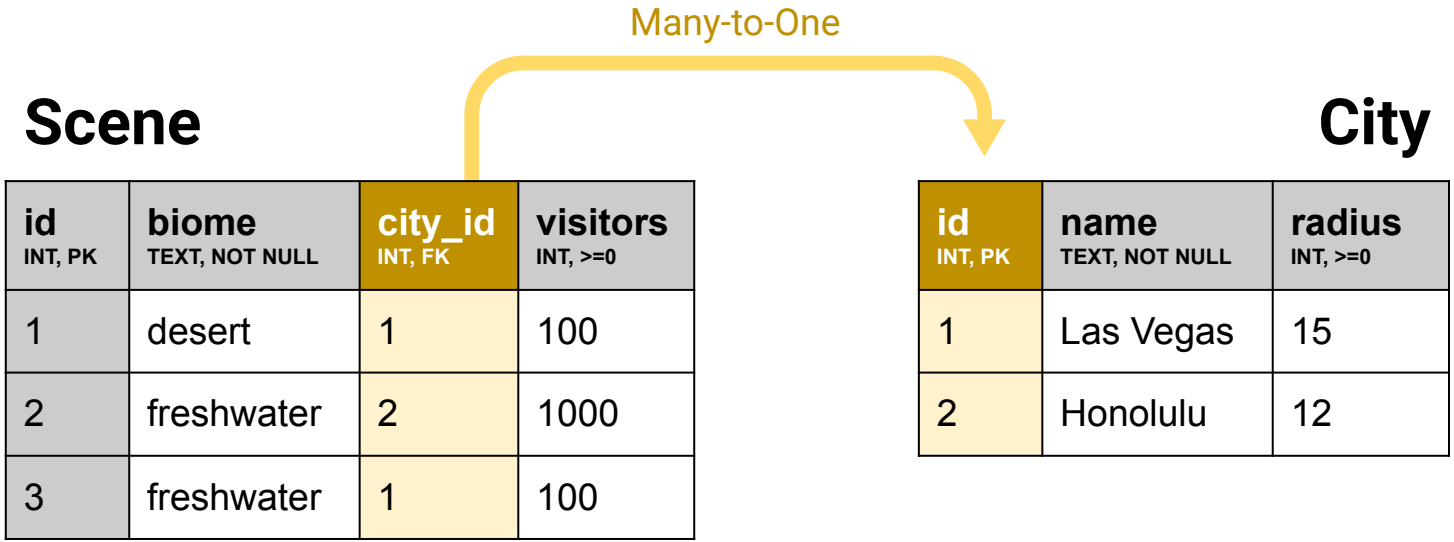
Types of Relationships

Many-to-One and Many-to-Many

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)

Many-to-One

Many-to-Many



Many Scenes to One City

Many-to-One

Many-to-Many

Scene

id INT, PK	biome TEXT, NOT NULL	city_id INT, FK	visitors INT, >=0
1	desert	1	100
2	freshwater	2	1000
3	freshwater	1	100

City

id INT, PK	name TEXT, NOT NULL	radius INT, >=0
1	Las Vegas	15
2	Honolulu	12

Scene

id INT, PK	biome TEXT, NOT NULL	city_id INT, FK	visitors INT, >=0
1	desert	1	100
2	freshwater	2	1000
3	freshwater	1	100

City

id INT, PK	name TEXT, NOT NULL	radius INT, >=0
1	Las Vegas	15
2	Honolulu	12

Biome

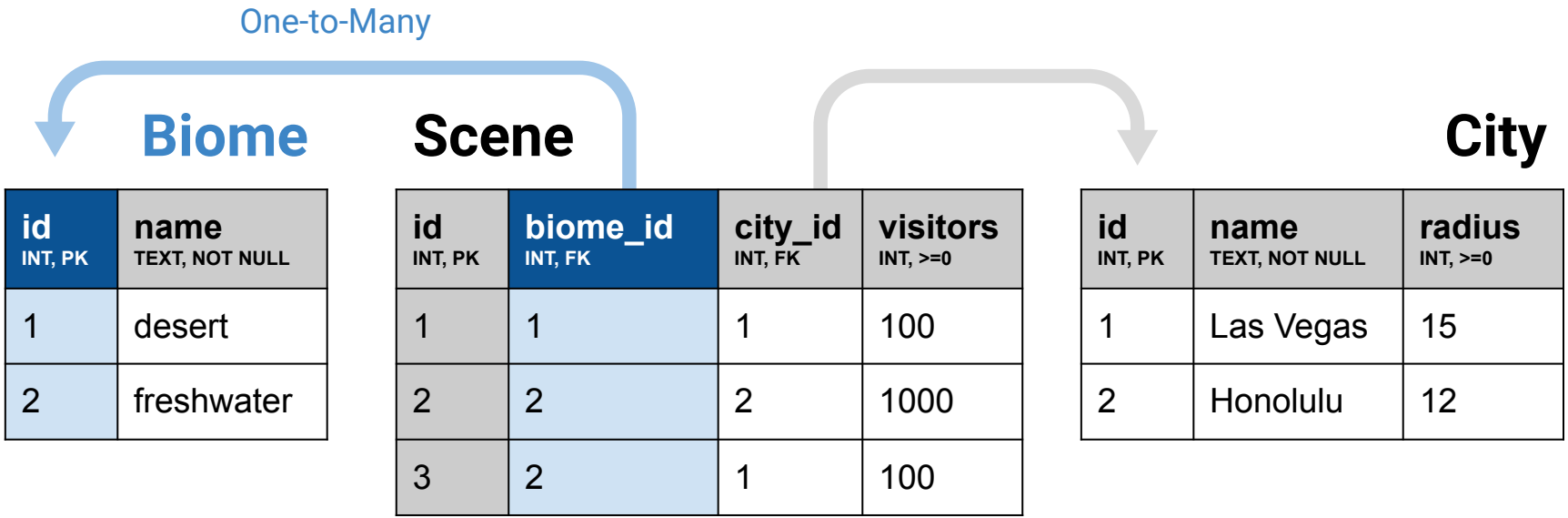
id INT, PK	name TEXT, NOT NULL
1	desert
2	freshwater

Scene

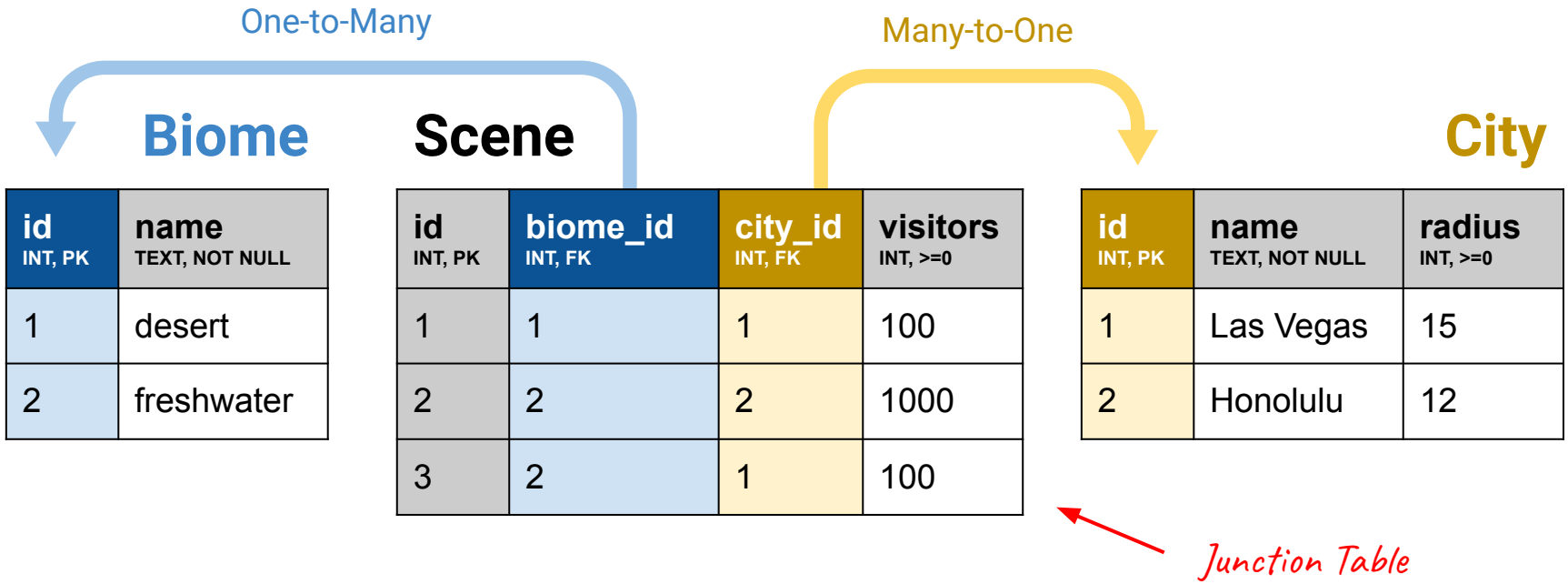
id INT, PK	biome_id INT, FK	city_id INT, FK	visitors INT, >=0
1	1	1	100
2	2	2	1000
3	2	1	100

City

id INT, PK	name TEXT, NOT NULL	radius INT, >=0
1	Las Vegas	15
2	Honolulu	12



One Biome to Many Scenes



Many Biomes to Many Cities

TAKEAWAY

Table relationships can be **one-to-many** or **many-to-many**. Use a junction table for many-to-many.

LECTURE 9

Conclusion

Takeaways

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)

Summary

- Possible to subsample directly in SQL
- Common Table Expressions allow complex queries
    - Complexity not always needed
    - CTEs preferred over subqueries, for readability
- Use IS for NULL predicates
- Simpler queries are often possible!

Scene

id INT, PK	**biome** TEXT, NOT NULL	**city** TEXT, NOT NULL	**visitors** INT, >=0
0	desert	Las Vegas	100
1	marine	Honolulu	1000
2	freshwater	Paris	50
3	marine	Taipei	100
4	desert	Austin	25
5	freshwater	Austin	240
6	freshwater	Las Vegas	100

Cross Join: All Pairs of Rows

```
SELECT * FROM Pet AS p, Breed AS b;
```

Pet

id	name
0	Apricot
1	Boots
2	Cally
4	Eugene

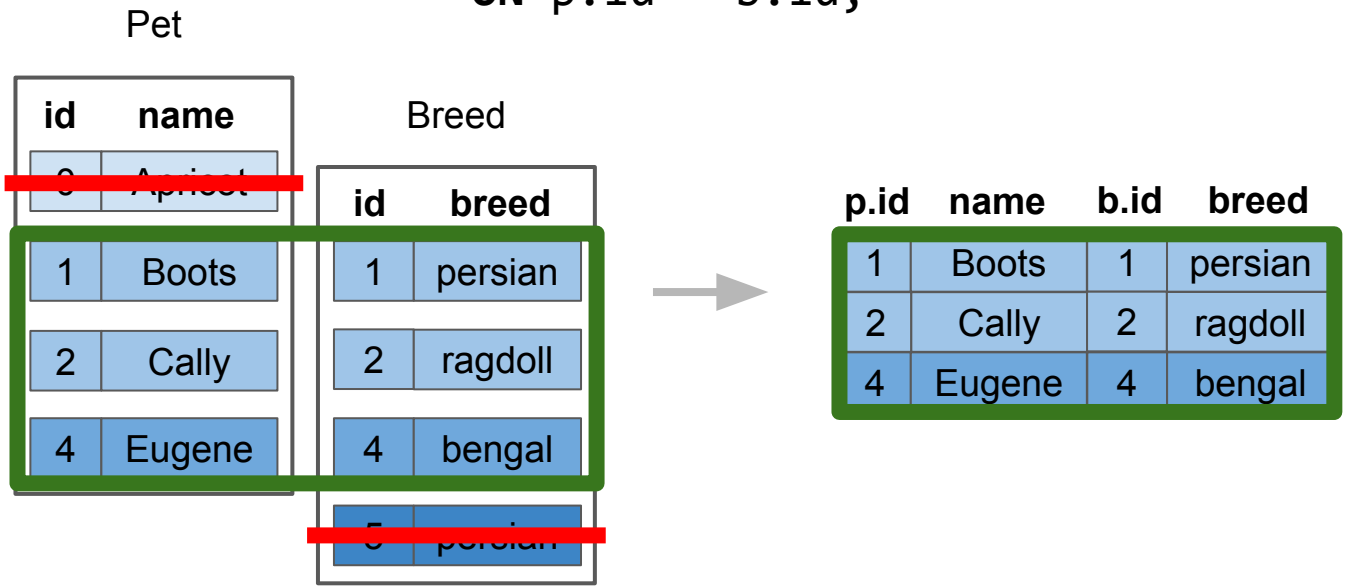
Breed

id	breed
1	persian
2	ragdoll
4	bengal
5	persian

p.id	name	b.id	breed
0	Apricot	1	persian
1	Boots	1	persian
2	Cally	1	persian
4	Eugene	1	persian
0	Apricot	2	ragdoll
1	Boots	2	ragdoll
2	Cally	2	ragdoll
4	Eugene	2	ragdoll
0	Apricot	4	bengal
1	Boots	4	bengal
2	Cally	4	bengal
4	Eugene	4	bengal
0	Apricot	5	persian
1	Boots	5	persian
2	Cally	5	persian
4	Eugene	5	persian

Inner Join: Only Matching Rows

```
SELECT * FROM Pet AS p
JOIN Breed AS b
    ON p.id = b.id;
```



Inner Join is Cross Join + Filter (conceptually)

```
SELECT *
FROM Pet AS p, Breed AS b
WHERE p.id = b.id;
```

Pet

id	name
0	Apricot
1	Boots
2	Cally
4	Eugene

Breed

id	breed
1	persian
2	ragdoll
4	bengal
5	persian

p.id	name	b.id	breed
0	Apricot	1	persian
1	Boots	1	persian
2	Cally	1	persian
4	Eugene	1	persian
0	Apricot	2	ragdoll
1	Boots	2	ragdoll
2	Cally	2	ragdoll
4	Eugene	2	ragdoll

Many-to-One

Many-to-Many

Cross Join

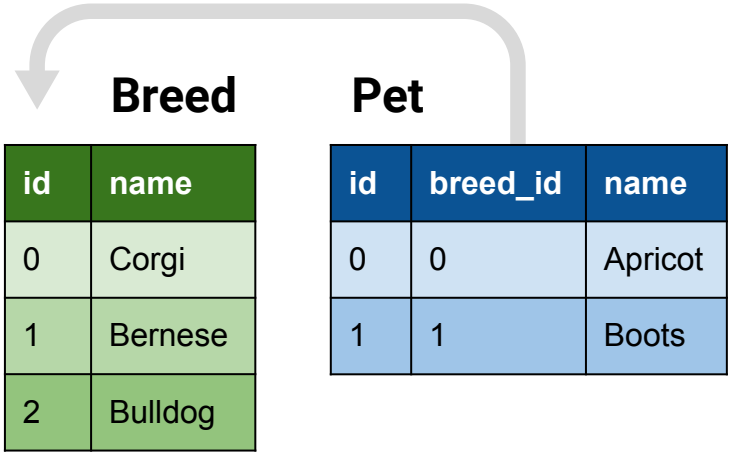
Inner Join

Outer Join

Join Conditions

Cross Join: All Pairs of Rows

```
SELECT * FROM Pet AS p, Breed AS b;
```



Breed

id	name
0	Corgi
1	Bernese
2	Bulldog

Pet

id	breed_id	name
0	0	Apricot
1	1	Boots

b.id	b.name	p.id	p.breed_id	p.name
0	Corgi	0	0	Apricot
1	Bernese	0	0	Apricot
2	Bulldog	0	0	Apricot
0	Corgi	1	1	Boots
1	Bernese	1	1	Boots
2	Bulldog	1	1	Boots

Many-to-One

Many-to-Many

Cross Join

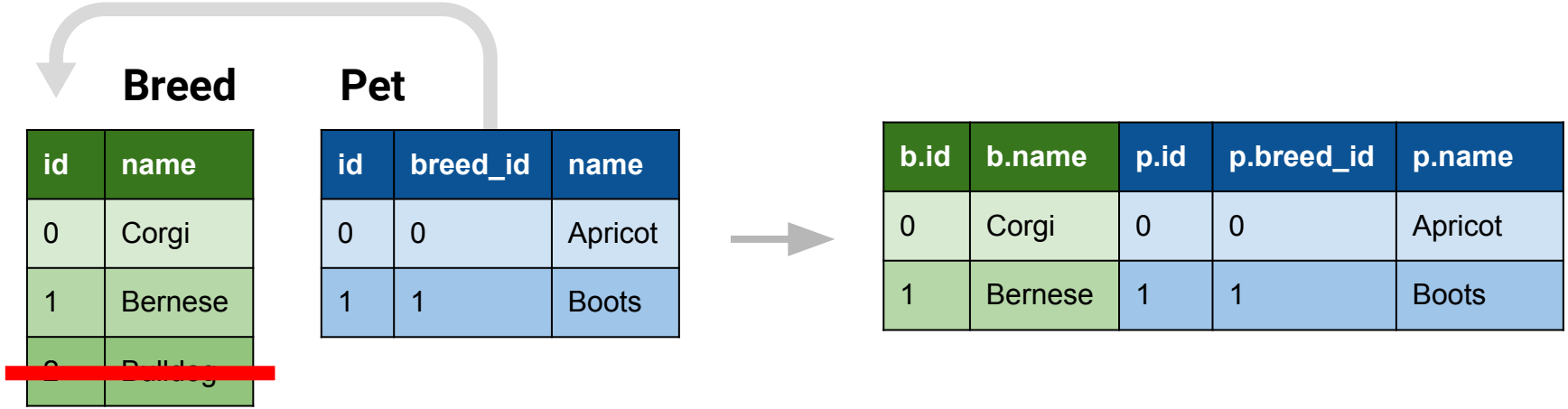
Inner Join

Outer Join

Join Conditions

Inner Join: Only Matching Rows

```
SELECT * FROM Pet AS p
JOIN Breed AS b
    ON p.breed_id = b.id;
```



Breed

id	name
0	Corgi
1	Bernese
~~2~~	~~Bulldog~~

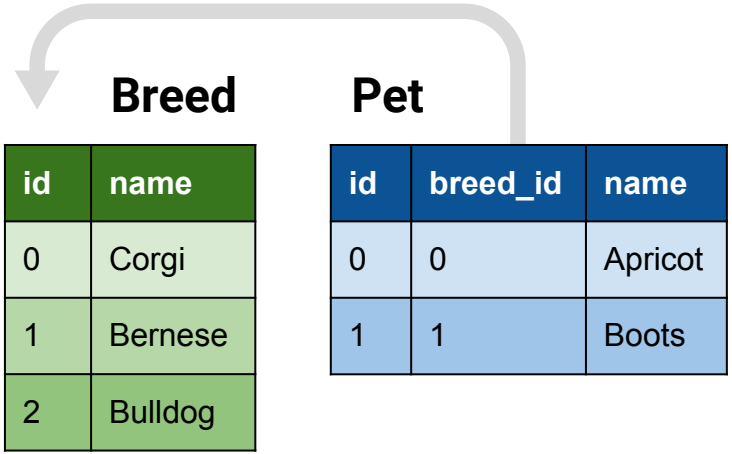
Pet

id	breed_id	name
0	0	Apricot
1	1	Boots

b.id	b.name	p.id	p.breed_id	p.name
0	Corgi	0	0	Apricot
1	Bernese	1	1	Boots

Inner Join is Cross Join + Filter (conceptually)

```
SELECT *
FROM Pet AS p, Breed AS b
WHERE p.breed_id = b.id;
```



Breed

id	name
0	Corgi
1	Bernese
2	Bulldog

Pet

id	breed_id	name
0	0	Apricot
1	1	Boots

b.id	b.name	p.id	p.breed_id	p.name
0	Corgi	0	0	Apricot
1	Bernese	0	0	Apricot
2	Bulldog	0	0	Apricot
0	Corgi	1	1	Boots
1	Bernese	1	1	Boots
2	Bulldog	1	1	Boots

Many-to-One

Many-to-Many

Cross Join

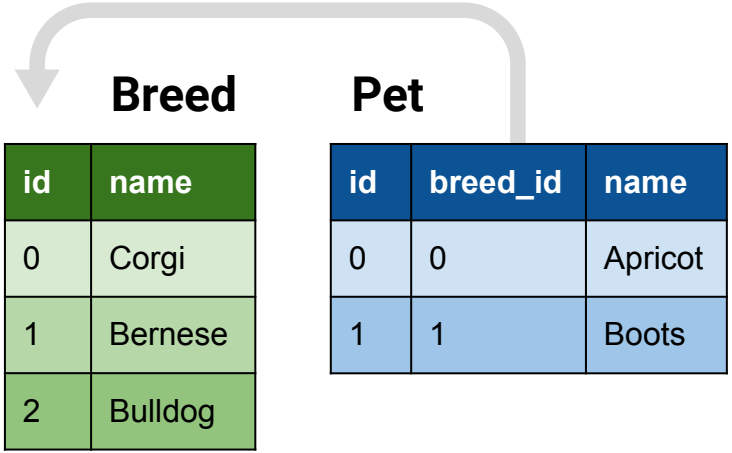
Inner Join

Outer Join

Join Conditions

Left Outer Join: All Rows in 1st Table

```
SELECT * FROM Pet AS p
LEFT JOIN Breed AS b
    ON p.id = b.id;
```



Breed

id	name
0	Corgi
1	Bernese
2	Bulldog

Pet

id	breed_id	name
0	0	Apricot
1	1	Boots

b.id	b.name	p.id	p.breed_id	p.name
0	Corgi	0	0	Apricot
1	Bernese	1	1	Boots
2	Bulldog	NULL	NULL	NULL

PRACTICAL TIP

Use `OUTER JOIN` to account for rows with no relationships.

Example: Report number of orders per user.
`INNER JOIN` would omit users with 0 orders.

Right Outer Join: All Rows in 2nd Table

NOT SUPPORTED IN SQLITE

```
SELECT * FROM Pet AS p
RIGHT JOIN Breed AS b
    ON p.id = b.id;
```

Pet

id	name
~~0~~	~~Apricot~~
1	Boots
2	Cally
4	Eugene

Breed

id	breed
1	persian
2	ragdoll
4	bengal
5	persian

p.id	name	b.id	breed
1	Boots	1	persian
2	Cally	2	ragdoll
4	Eugene	4	bengal
		5	persian

Missing values are **NULL**.

Full Outer Join: All Rows in Both Tables

NOT SUPPORTED IN SQLITE

```
SELECT * FROM Pet AS p
FULL OUTER JOIN Breed AS b
    ON p.id = b.id;
```

Pet

id	name
0	Apricot
1	Boots
2	Cally
4	Eugene

Breed

id	breed
1	persian
2	ragdoll
4	bengal
5	persian

p.id	name	b.id	breed
0	Apricot		
1	Boots	1	persian
2	Cally	2	ragdoll
4	Eugene	4	bengal
		5	persian

Many-to-One

Many-to-Many

Cross Join

Inner Join

Outer Join

Join Conditions

Join conditions don’t have to be equality

```
SELECT * FROM Student AS s
JOIN Teacher AS t
    ON s.age > t.age;
```

Student

age	name
29	Jameel
37	Jian
20	Emma

Teacher

age	name
52	Ira
27	John
36	Anuja

s.age	s.name	t.age	t.name
29	Jameel	27	John
37	Jian	27	John
37	Jian	36	Anuja

TAKEAWAY

Use a junction table for many-to-many relationships. Use **`OUTER JOIN`**s to account for rows with no relationships.