3474659

slido

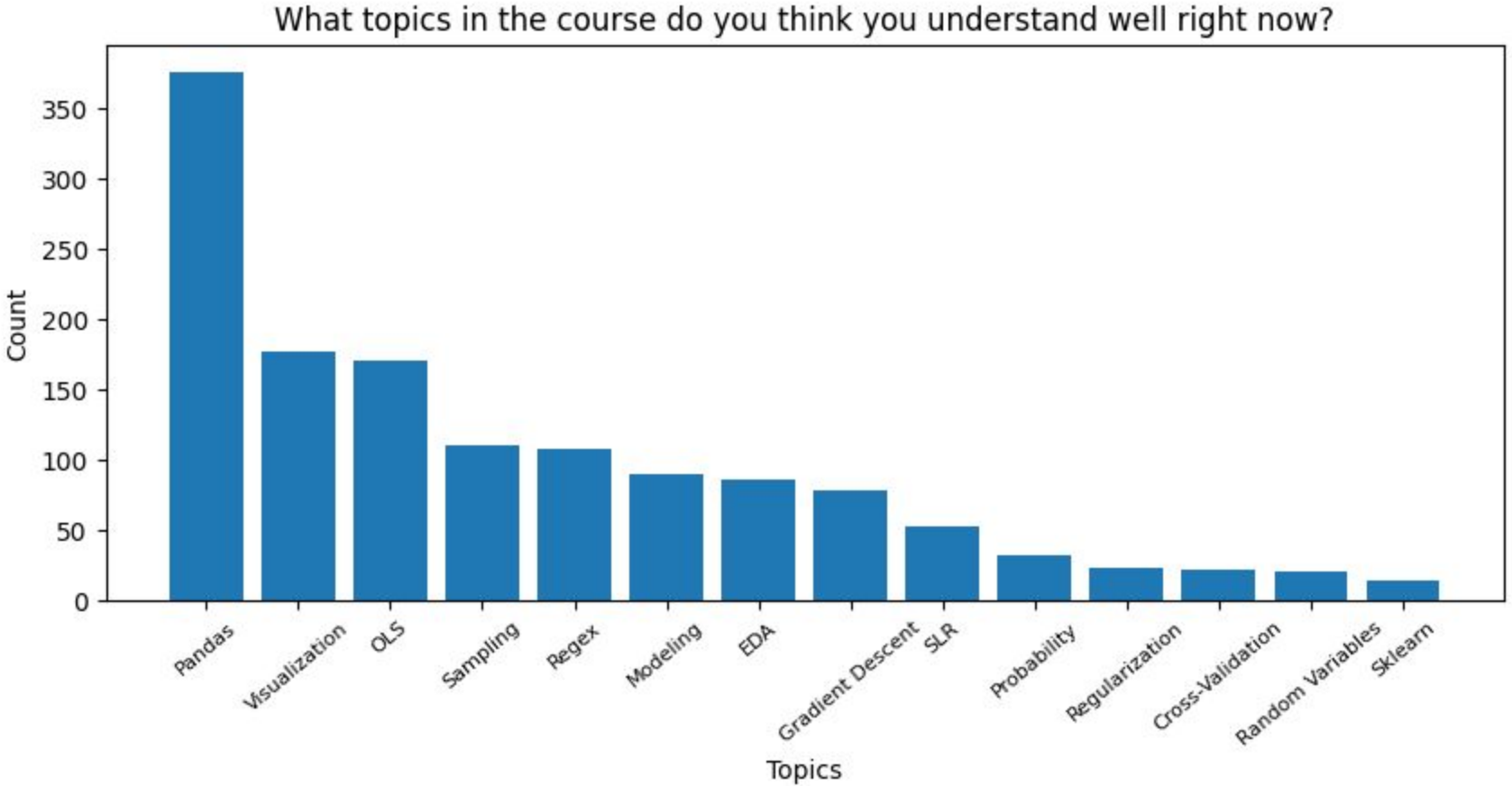
Join at slido.com
#3474659

ⓘ Click **Present with Slido** or install our Chrome extension to display joining instructions for participants while presenting.

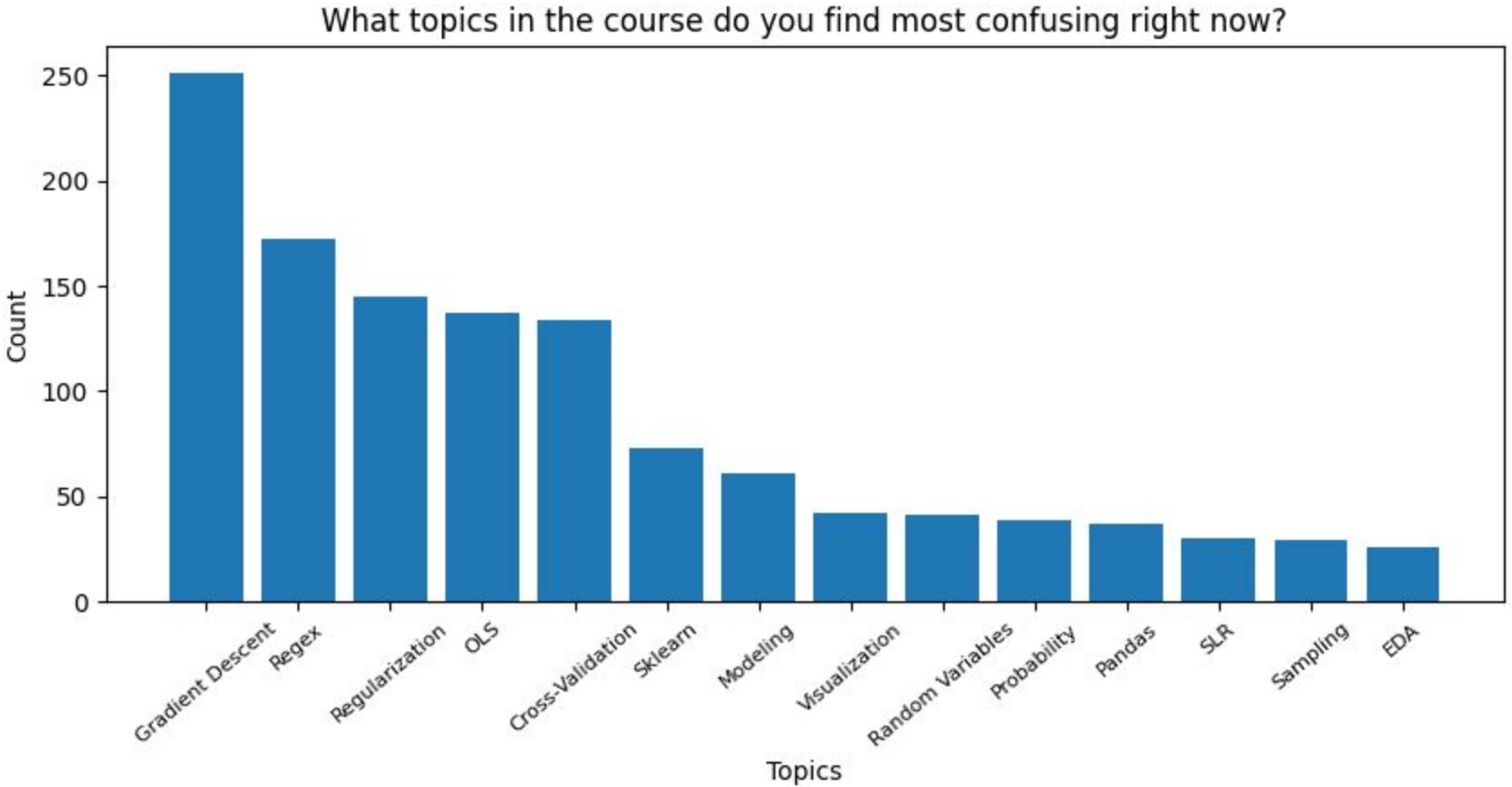
3474659

Mid-Semester Feedback

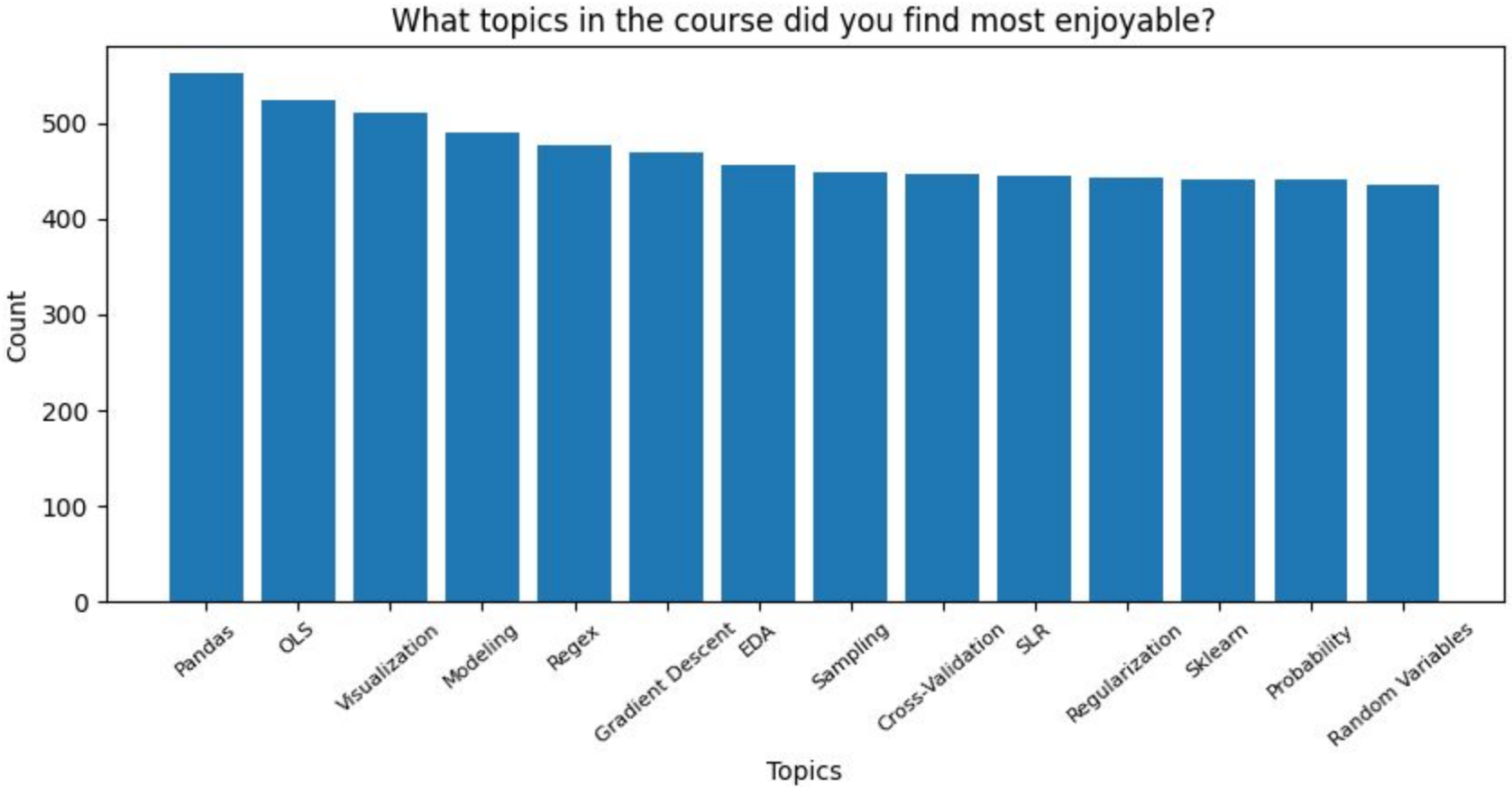
Mid-Semester Survey Results I



Mid-Semester Survey Results I

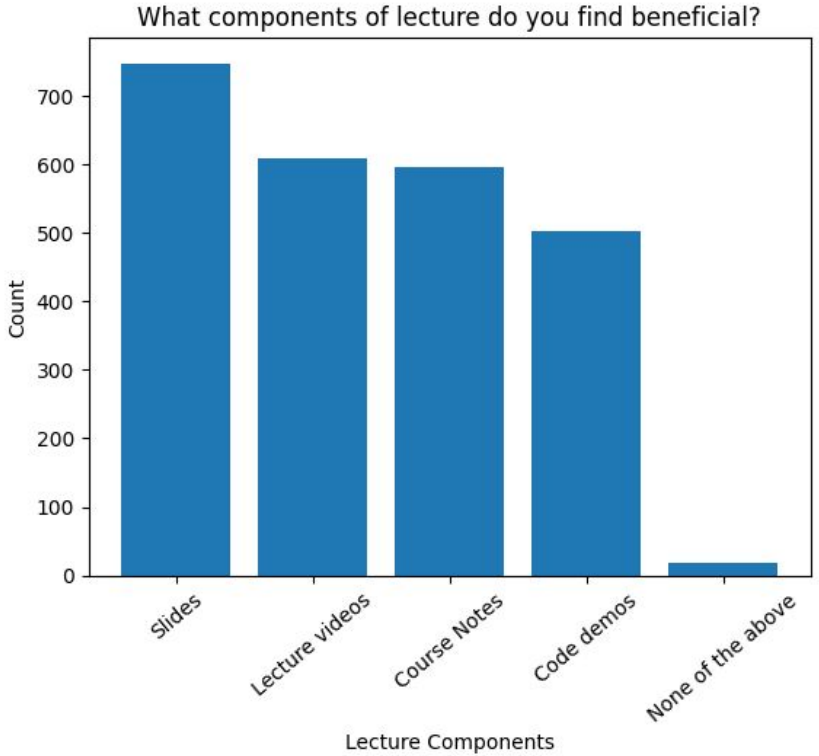


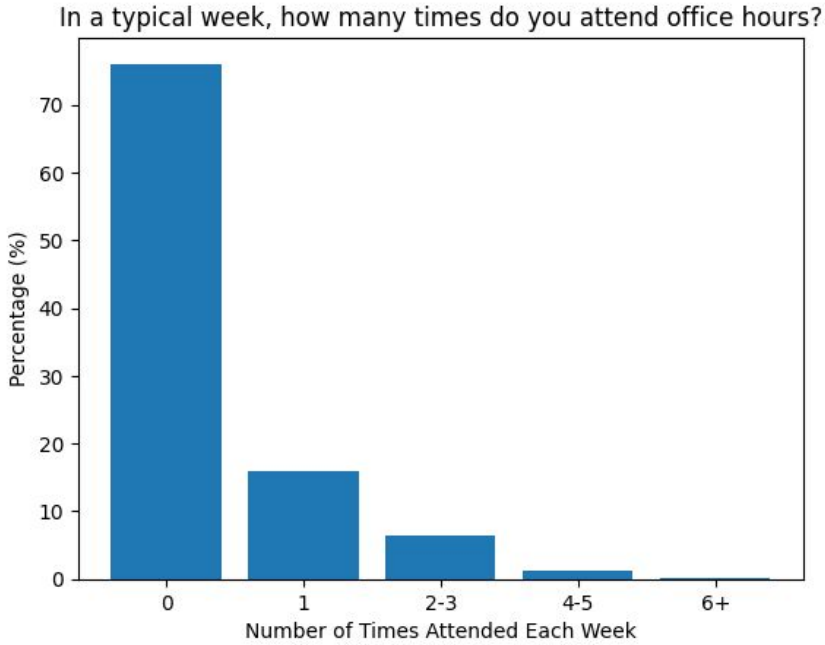
Mid-Semester Survey Results I



Mid-Semester Survey Results I - lecture and office hour

3474659

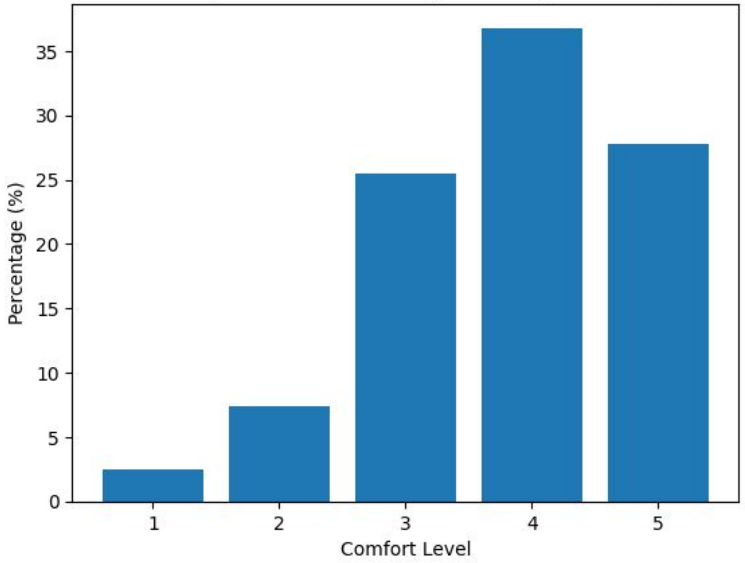




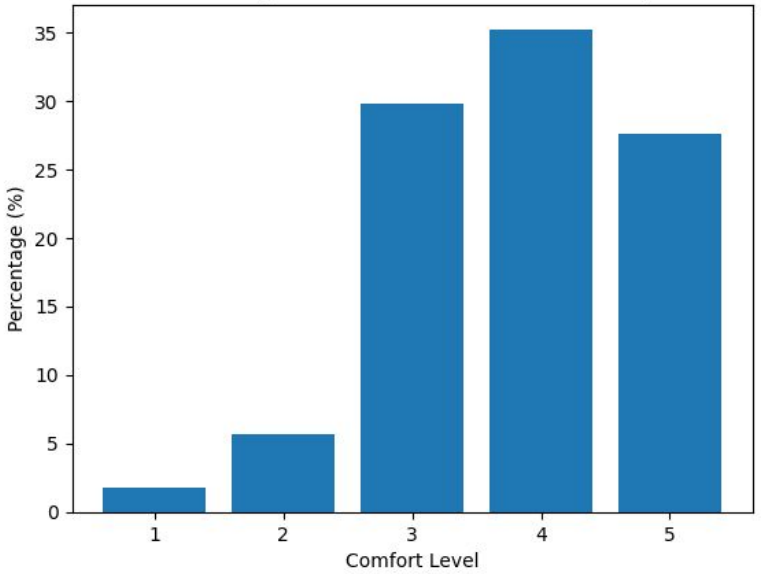
Mid-Semester Survey Results I - midterm

3474659

How comfortable were you with the setting, location, and timing of the Midterm?



How comfortable were you with the content and difficulty of the Midterm



Mid-Semester Survey Results II

3474659

Some actions we are taking to improve the student experience:

- More coding
- Faster course note uploads (upload before lecture).
- Add more Slidos to the lectures to make them more engaging
- Want office hours spread across all days of the week

What are you most looking forward to in the second half of the course?

- SQL
- More modeling
- PCA
- LLMs
- Projects
- ML
- Logistic Regression
- Sticker

We appreciate all your honest and constructive feedback!

Mid-Semester Survey Results III

3474659

Resources that students found helpful:

- Office Hours
- Discussion Sections
- Course Notes
- Mini-lectures

Reminder for other resources:

- We have a lot of office hours; please utilize them.
- Our tutors and TAs are *constantly monitoring* ***Ed*** to answer your questions.
- ***Mini-lectures*** are available for all discussions on the website.
- Use ***debugging guide*** to help with common errors for our homeworks and projects.
- Life happens and we can only help you if you communicate with us. Let us know by filling out the ***additional accommodations form***.

LECTURE 20

SQL I

SQL and databases: alternatives to `pandas` and CSV files.

Data 100/Data 200, Spring 2025 @ UC Berkeley

Narges Norouzi and Josh Grossman

Content credit: Acknowledgments

Goals for Today's Lecture

Lecture 20, Data 100 Spring 2025

Stepping away from Python and **pandas**

- Recognizing situations where we need "bigger" tools for manipulating data
- Writing our first database queries

3474659

Agenda

Lecture 20, Data 100 Spring 2025

- Why Databases?
- Intro to SQL
- Tables and Schema
- Basic Queries

3474659

Why Databases?

Lecture 20, Data 100 Spring 2025

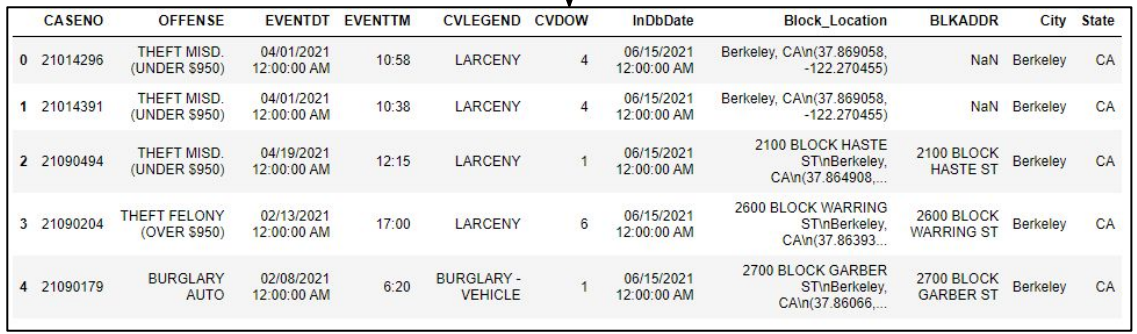
- **Why Databases?**
- Intro to SQL
- Tables and Schema
- Basic queries

So Far: CSV Files and pandas

3474659

So far in Data 100, we've worked with data stored in CSV files.

`Berkeley_PD_-_Calls_for_Service.csv` → `pd.read_csv`



	CASENO	OFFENSE	EVENTDT	EVENTTM	CVLEGEND	CVDOW	InDbDate	Block_Location	BLKADDR	City	State
0	21014296	THEFT MISD. (UNDER $950)	04/01/2021 12:00:00 AM	10:58	LARCENY	4	06/15/2021 12:00:00 AM	Berkeley, CA\n(37.869058, -122.270455)	NaN	Berkeley	CA
1	21014391	THEFT MISD. (UNDER $950)	04/01/2021 12:00:00 AM	10:38	LARCENY	4	06/15/2021 12:00:00 AM	Berkeley, CA\n(37.869058, -122.270455)	NaN	Berkeley	CA
2	21090494	THEFT MISD. (UNDER $950)	04/19/2021 12:00:00 AM	12:15	LARCENY	1	06/15/2021 12:00:00 AM	2100 BLOCK HASTE ST\nBerkeley, CA\n(37.864908,...	2100 BLOCK HASTE ST	Berkeley	CA
3	21090204	THEFT FELONY (OVER $950)	02/13/2021 12:00:00 AM	17:00	LARCENY	6	06/15/2021 12:00:00 AM	2600 BLOCK WARRING ST\nBerkeley, CA\n(37.86393...	2600 BLOCK WARRING ST	Berkeley	CA
4	21090179	BURGLARY AUTO	02/08/2021 12:00:00 AM	6:20	BURGLARY - VEHICLE	1	06/15/2021 12:00:00 AM	2700 BLOCK GARBER ST\nBerkeley, CA\n(37.86066,...	2700 BLOCK GARBER ST	Berkeley	CA

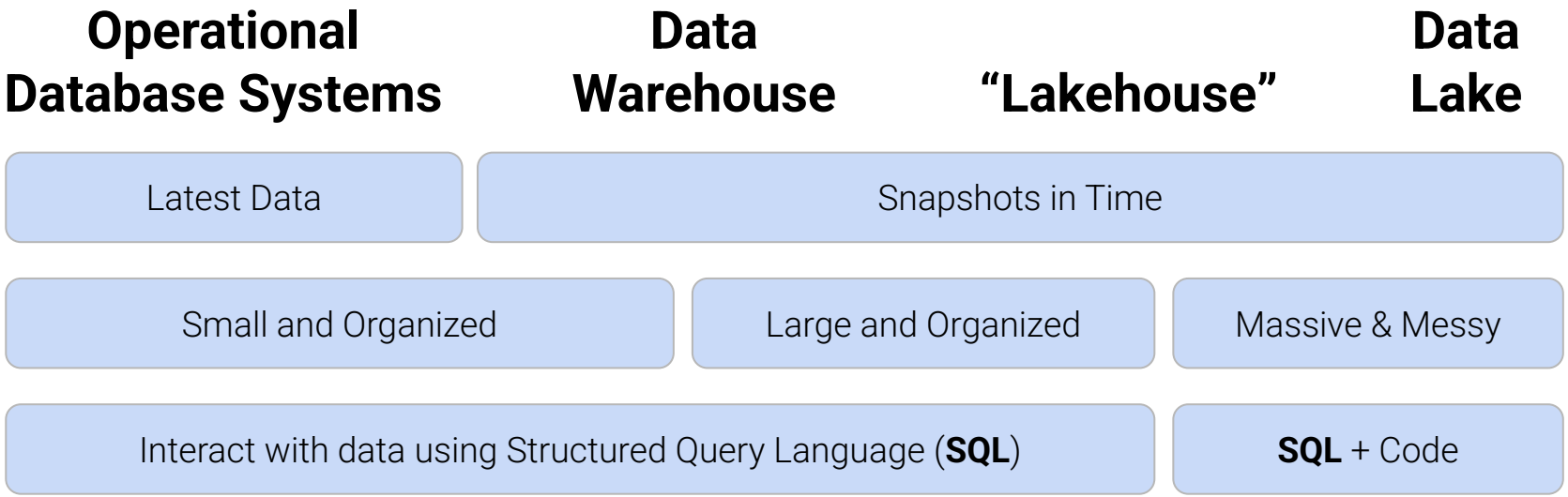
Perfectly reasonable workflow for **moderately sized data** that is **unchanging (static)**

- **Size < ~10GB (⅓ the RAM on your workstation)**
- **Weekly snapshot**

In the real world…

3474659

data is typically stored in a Database Management System:



Brief Databases Overview

3474659

A **database** is an organized collection of data.

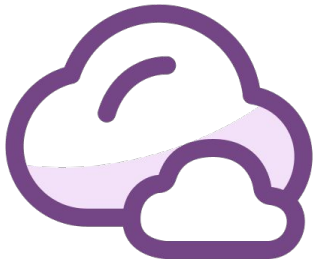
A **Database Management System (DBMS)** is a software system that **stores**, **manages**, and **facilitates access** to one or more databases.

Common **Large-Scale DBMS Systems** used in **Data Science**:

- Google BigQuery
- Amazon Redshift
- Snowflake
- Databricks
- Microsoft SQL Server
- ...

slido

3474659



Databases are often depicted using this icon. What is this a picture of?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Looks Like?

Platters on a Disk Drive

Looks Like?

1956: IBM MODEL 350 RAMAC
First Commercial Disk Drive
5MB @ 1 ton

*"...We must immediately...**attack accounting problems** under the philosophy of handling each business **transaction as it occurs**, rather than under the present condition of **batching techniques**...."*

-- F. J. Wesley IBM Senior Manager

Advantages of DBMS over Raw Files (e.g., CSV)

3474659

Data Storage:

- **Reliable storage** to survive system crashes and disk failures.
- Optimize to **compute on data that does not fit in memory**.
- Special data structures to **improve performance** (see CS (W)186).

Data Management:

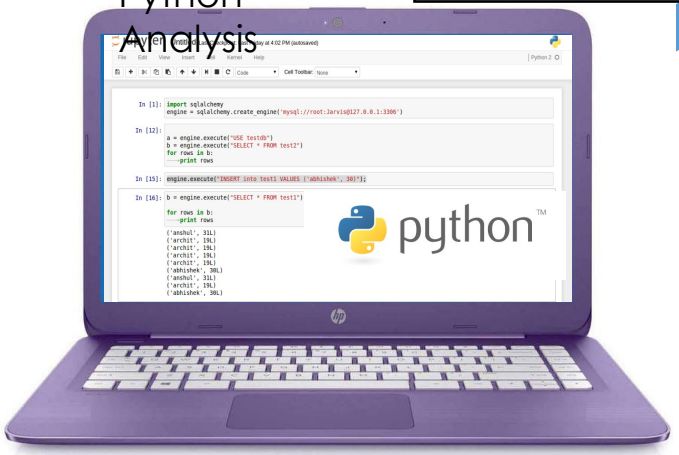
- Configure how data is **logically organized** and **who has access**.
- Can enforce guarantees on the data (e.g. non-negative person weight or age).
  - Can be used to **prevent data anomalies**.
  - Ensures **safe concurrent operations** on data (multiple users reading and writing simultaneously, e.g. ATM transactions).

Interacting with a DBMS

Query

```
SELECT * FROM
sales
WHERE price >
100.0
```

Python Analysis



Response

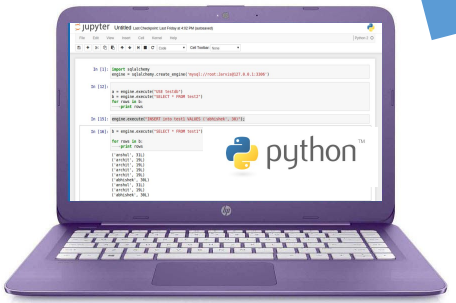
Date	Purchase ID	Name	Price
9/20/2012	1234	Sue	$200.00
8/21/2012	3453	Joe	$333.99

Interacting with a DBMS

DBMS Server

Web Applications

Python Analysis



Visualization

Often many systems will connect to a DBMS **concurrently**.

3474659

Intro to SQL

Lecture 20, Data 100 Spring 2025

- Why Databases?
- **Intro to SQL**
- Tables and Schema
- Basic queries

SQL

3474659

Today we'll be using a **programming language** called **"Structured Query Language"** or **SQL**.

- SQL is its **own programming language**, distinct from Python.
  - But often called from within other programming languages (e.g., Python)
- SQL is a **special purpose programming language** used specifically for communicating with database systems
  - **Dominant language/technology for working with data! (you must know it!)**
  - **The language of tables:** all inputs and outputs are tables
  - **Introduced in the 1970s** – Originally called "Structured English Query Language" SEQUEL – reads like English but looks funny compared to Python
  - **Most systems don't follow the standards**
    - Every system you work with will be a little different…
- We will **program in SQL** using **Jupyter notebooks** and connecting with **DuckDB** (and maybe **SQLite** systems)

Some Database Systems we will use in Data100

3474659

SQLite is an easy to use **library** that lets you directly manipulate a **database file** or **an in-memory DB** using a **simplified version of SQL**.

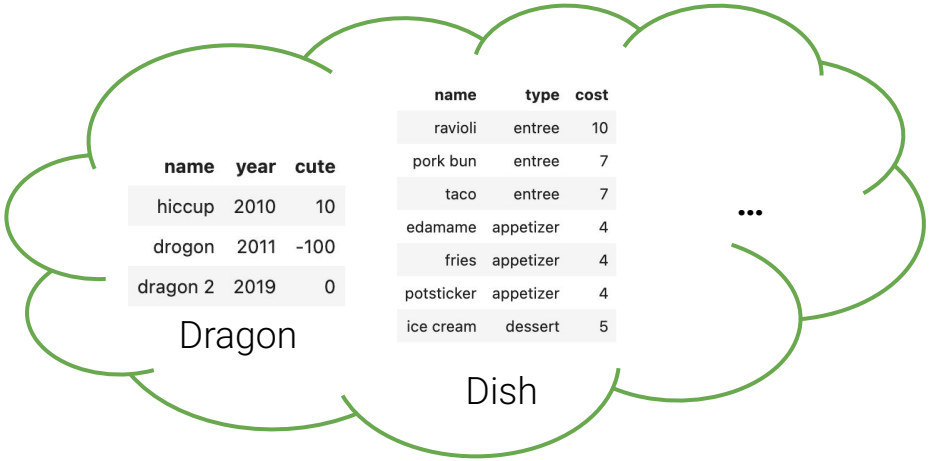
- Commonly used to store data for small apps on mobile devices (...standard on Android)
- Optimized for **simplicity** and **speed of simple data tasks** (e.g., lookup, add record etc…)

DuckDB is an easy to use **library** that lets you directly manipulate a **database file, collection of table formatted files (e.g., CSV),** or **in-memory pandas dataframes** using a **more complete version of SQL**

- Increasingly popular for **data analysis tasks** on **large datasets**
  - **You should probably learn this!**
- Optimized for **simplicity** and **speed of advanced data analysis tasks**
- We are switching to this for Data100

Quick SQL Overview

3474659



SQL query

```
SELECT *
FROM Dish;
```

results

name	type	cost
ravioli	entree	10
pork bun	entree	7
taco	entree	7
edamame	appetizer	4
fries	appetizer	4
potsticker	appetizer	4
ice cream	dessert	5

Demo

Step 1: Load the SQL Module

Our first step is to load the SQL module. We do so using the `ipython` **cell magic** command:

```
%load_ext sql
```

Step 2: Connect to a Database

3474659

Our first step is to load the SQL module. We do so using the `ipython` **cell magic** command:

```
%load_ext sql
```

The second step is to connect to a database.

We use **`%sql`** to tell Jupyter that this cell represents SQL code rather than Python code.

```
%sql duckdb:///data/example_duck.db --alias duck
```

(A note about DBMS Technologies used in Data100)

3474659

Our first step is to load the SQL module. We do so using the `ipython` **cell magic** command:

```
%load_ext sql
```

The second step is to connect to a database.

We use `%sql` to tell Jupyter that this cell represents SQL code rather than Python code.

```
%sql duckdb:///data/example_duck.db --alias duck
```

In Data 100, our database is stored in a local file. In practice, you'd probably connect to a remote server.

```
%%sql
postgresql://joshhug:mypassw@berkeley.edu/grades
```

3. Run SQL Statements

3474659

Now that we're connected, let's make some queries!

For example, we might show every row in the `Dragon` table.

```
%%sql
SELECT * FROM Dragon;
```


returns

name	year	cute
hiccup	2010	10
drogon	2011	-100
dragon 2	2019	0
puff	2010	100
smaug	2011	None

Thanks to the pandas magic, the resulting return data is displayed in a format almost identical to our Pandas tables (without an index).

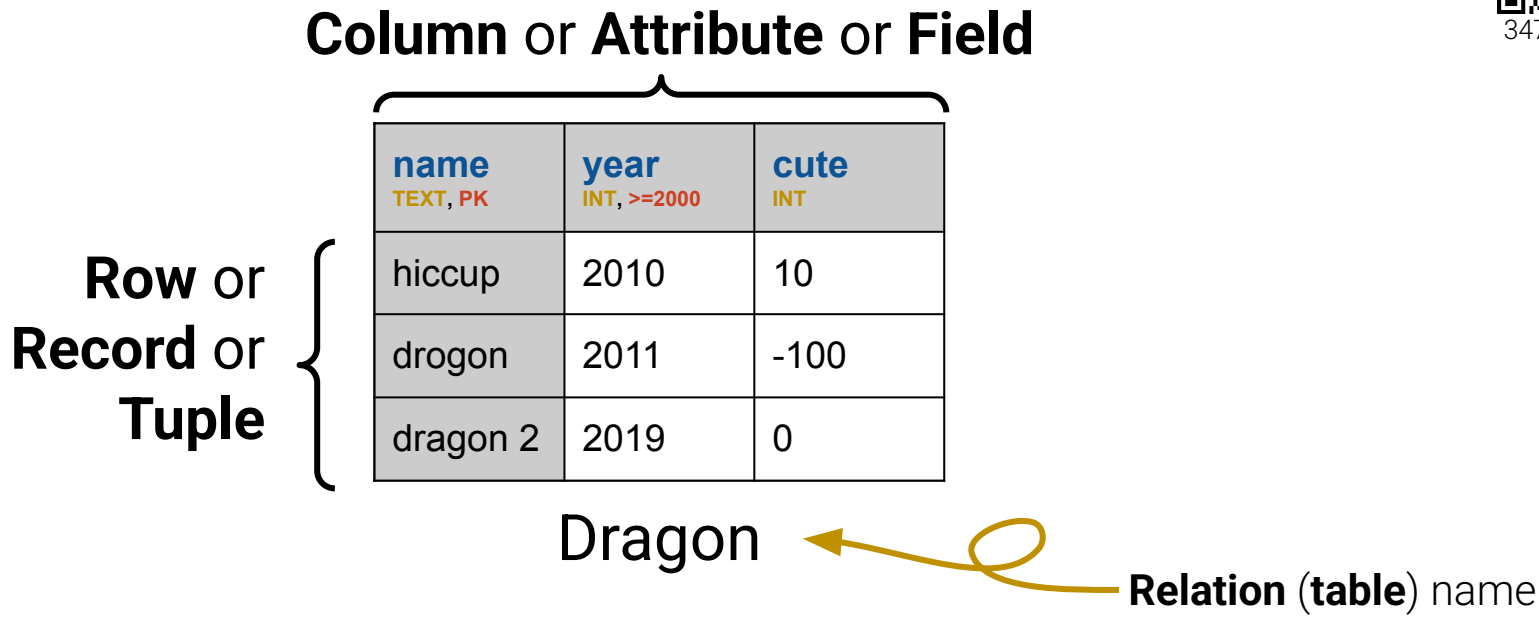
SQL statements are terminated with semicolons. A **SQL query** is a SQL statement that returns data.

Tables and Schema

Lecture 20, Data 100 Spring 2025

- Why Databases?
- Intro to SQL
- **Tables and Schema**
- Basic queries

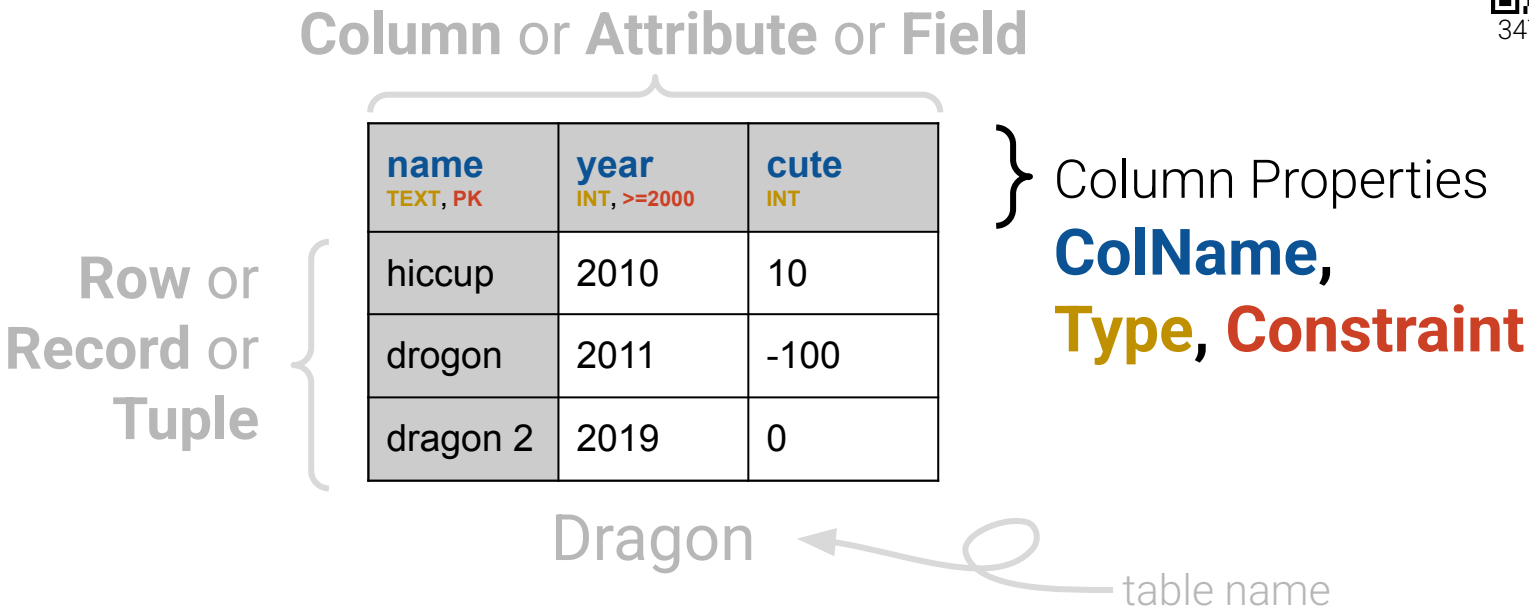
3474659



SQL tables are also **called relations**.

SQL Terminology

3474659



SQL **tables** are also called **relations**.

Every column in a SQL table has three properties: **ColName**, **Type**, and zero or more **Constraints**. (Contrast with `pandas`: `Series` have names and types, but no constraints.)

Table Schema

3474659

A **schema** describes the logical structure of a table. Whenever a new table is created, the creator must declare its schema.

For each column, specify the:

- **Column name**
- **Data type**
- **Constraint(s) on values**

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

Repeat for all tables in the database (see demo notebook):

	type	name	tbl_name	rootpage	sql
0	table	dish	dish	0	CREATE TABLE dish("name" VARCHAR PRIMARY KEY, "type" VARCHAR, "cost" INTEGER, CHECK(("cost" >= 0)));
1	table	dragon	dragon	0	CREATE TABLE dragon("name" VARCHAR PRIMARY KEY, "year" INTEGER, cute INTEGER, CHECK(("year" >= 2000)));
2	table	scene	scene	0	CREATE TABLE scene(id INTEGER PRIMARY KEY, biome VARCHAR NOT NULL, city VARCHAR NOT NULL, visitors INTEGER, created_at TIMESTAMP DEFAULT(current_date()), CHECK((visitors >= 0)));

Demo!

Example Types

Some examples of SQL **types**:

- `INT`: Integers.
- `FLOAT`: Floating point numbers.
- `VARCHAR`: Strings of text (also called `TEXT`).
- `BLOB`: Arbitrary data, e.g. songs, video files, etc.
- `DATETIME`: A date and time.

Note: Different implementations of SQL support different types.

- DuckDB: https://duckdb.org/docs/sql/data_types/overview.html
- SQLite: https://www.sqlite.org/datatype3.html
- MySQL: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

In Data 100, we will use **DuckDB**.

Example Constraints

3474659

Some examples of **constraints**:

- `CHECK`: data must obey the given check constraint.
- `PRIMARY KEY`: specifies that this key is used to uniquely identify rows in the table.
- `NOT NULL`: null data cannot be inserted for this column.
- `DEFAULT`: provides a default value to use if user does not specify on insertion.

What is this primary key constraint?

type	name	tbl_name	rootpage	sql
table	sqlite_sequence	sqlite_sequence	7	CREATE TABLE sqlite_sequence(name,seq)
table	Dragon	Dragon	2	CREATE TABLE Dragon (name TEXT PRIMARY KEY, year INTEGER CHECK (year >= 2000), cute INTEGER)
table	Dish	Dish	4	CREATE TABLE Dish (name TEXT PRIMARY KEY, type TEXT, cost INTEGER CHECK (cost >= 0))
table	Scene	Scene	6	CREATE TABLE Scene (id INTEGER PRIMARY KEY AUTOINCREMENT, biome TEXT NOT NULL, city TEXT NOT NULL, visitors INTEGER CHECK (visitors >= 0), created_at DATETIME DEFAULT (DATETIME('now')))

Primary Keys

3474659

A **primary key** is the set of column(s) used to uniquely identify each record in the table.

- In the Dragon table, the "**name**" of each Dragon is the primary key.
- In other words, no two dragons can have the same name!
- Primary key is used **to ensure data integrity** and **to optimize data access.**

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

Why specify primary keys? More next time when we discuss JOINs…

Foreign Keys

3474659

A **foreign key** is a column or set of columns that references a **primary key in another table**.

- A foreign key constraint ensures that a primary key exists in the referenced table

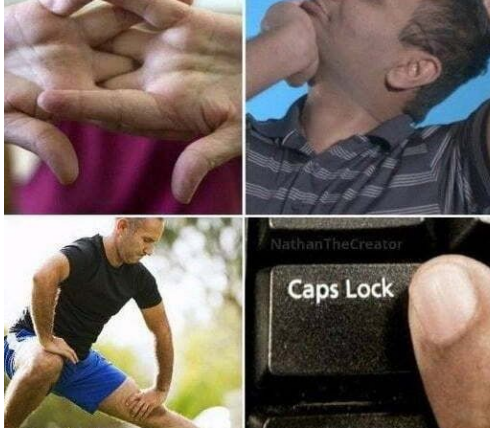
```
CREATE TABLE student (
    student_id INTEGER PRIMARY KEY,
    name VARCHAR,
    email VARCHAR
);
```

```
CREATE TABLE assignment (
    assignment_id INTEGER PRIMARY KEY,
    description VARCHAR
);
```

```
CREATE TABLE grade (
    student_id INTEGER,
    assignment_id INTEGER,
    score REAL,
    FOREIGN KEY (student_id) REFERENCES student(student_id),
    FOREIGN KEY (assignment_id) REFERENCES assignment(assignment_id)
);
```

Which of the following statements is true?

Interlude



I'm planning to make a film series on databases.
I've got the first part ready.
Now I can't think of a SQL.

Basic Queries

Lecture 20, Data 100 Spring 2025

- Why Databases?
- Intro to SQL
- Tables and Schema
- **Basic Queries**

Query Syntax So Far


```
SELECT <column list>
FROM <table>
```

;

Marks the end of a SQL statement.

Summary So Far

New keywords

```
SELECT <column list>
FROM <table>
[WHERE <predicate>]
[ORDER BY <column list>]
[LIMIT <number of rows>]
[OFFSET <number of rows>];
```

3474659

Goal of this section

By the end of this section, you will learn these new keywords!

But first, more SELECT

Recall our simplest query, which returns the full relation:

```
SELECT *
FROM Dragon;
```

table name

name	year	cute
hiccup	2010	10
drogon	2011	-100
dragon 2	2019	0
puff	2010	100
smaug	2011	None

`SELECT` specifies the column(s) that we wish to appear in the output. `FROM` specifies the database table from which to select data.

Every query must include a `SELECT` clause (how else would we know what to return?) and a `FROM` clause (how else would we know where to get the data?)

An asterisk (*) is shorthand for “all columns”. *Let’s see a bit more in our demo.*

But first, more SELECT

Recall our simplest query, which returns the full relation:

```
SELECT *
FROM Dragon;
```

table name

name	year	cute
hiccup	2010	10
drogon	2011	-100
dragon 2	2019	0
puff	2010	100
smaug	2011	None

We can also `SELECT` only a **subset of the columns**:

column expression list

```
SELECT cute, year
FROM Dragon;
```

cute	year
10	2010
-100	2011
0	2019
100	2010
None	2011

Columns selected in specified order

Aliasing with AS

To rename a `SELECT`ed column, use the `AS` keyword

```
SELECT cute AS cuteness,
       year AS birth
FROM Dragon;
```

cuteness	birth
10	2010
-100	2011
0	2019
100	2010
None	2011

An **alias** is a name given to a column or table by a programmer. Here, “cuteness” is an alias of the original “cute” column (and “birth” is an alias of “year”)

SQL Style: Newline Separators

3474659

The following two queries both retrieve the same relation:

```
SELECT cute AS cuteness,
       year AS birth
FROM Dragon;
```

(more readable)

```
SELECT cute AS
cuteness, year AS
birth FROM Dragon;
```

cuteness	birth
10	2010
-100	2011
0	2019
100	2010
None	2011

Use newlines and whitespace wisely in your SQL queries.
It will simplify your debugging process!

Uniqueness with `DISTINCT`

To return only unique values, combine `SELECT` with the `DISTINCT` keyword

```
SELECT DISTINCT year
FROM Dragon;
```

name	year	cute
hiccup	2010	10
drogon	2011	-100
dragon 2	2019	0
puff	2010	100
smaug	2011	None

year
2010
2011
2019

Notice that 2010 and 2011 only appear once each in the output.

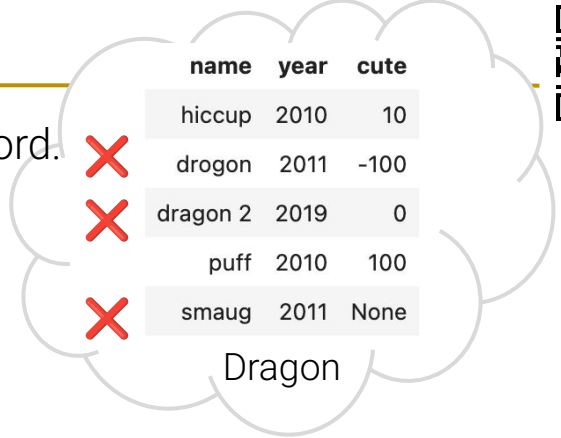
`WHERE`: Select a rows based on conditions

To select only some rows of a table, we can use the `WHERE` keyword.

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

condition

name	year
hiccup	2010
puff	2010



3474659

WHERE: Select a rows based on conditions

3474659

Comparators **OR**, **AND**, and **NOT** let us form more complex conditions.

```
SELECT name, year
FROM Dragon
WHERE cute > 0 OR year > 2013;
```

condition

name	cute	year
hiccup	10	2010
puff	100	2010
dragon 2	0	2019

Check if values are contained IN a specified list

```
SELECT name, year
FROM Dragon
WHERE name IN ('hiccup', 'puff');
```

name	year
puff	2010
hiccup	2010

Strings and SQL

3474659

Strings in SQL should use **single quote:**

- **‘Hello World’** is a ***String***
- **“Hello World”** is a column name which contains a space (you can do that…)

Double quoted strings refer to columns:

- SELECT **“birth weight”** FROM patient WHERE **“first name”** = **‘Joey’**

`WHERE` with NULL Values

NULL (the SQL equivalent of NaN) is stored in a special format – we can't use the "standard" operators =, >, and <.

Instead, check if something `IS` or `IS NOT NULL`

```
SELECT name, cute
FROM Dragon
WHERE cute IS NOT NULL;
```

name	cute
hiccup	10
drogon	-100
dragon 2	0
puff	100

Always work with NULLs using the `IS` operator. NULL does not work with standard comparisons: in fact, NULL = NULL actually returns False!

ORDER BY: Sort rows

Specify which column(s) we should order the data by

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

column

(by default, SQL orders by ascending order: **ASC**)

name	year	cute
puff	2010	100
hiccup	2010	10
dragon 2	2019	0
drogon	2011	-100
smaug	2011	None

ORDER BY: Sort rows

Specify which column(s) we should order the data by

```
SELECT *
FROM Dragon
ORDER BY year, cute DESC;
```

Can also order by multiple columns (for tiebreaks)

Sorts `year` in ascending order and `cute` in descending order. If you want `year` to be ordered in descending order as well, you need to specify `year DESC, cute DESC;`

name	year	cute
puff	2010	100
hiccup	2010	10
drogon	2011	-100
smaug	2011	None
dragon 2	2019	0

OFFSET and LIMIT?

3474659

name	year	cute
hiccup	2010	10
drogon	2011	-100
dragon 2	2019	0

Dragon

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

A.

name	year	cute
hiccup	2010	10
drogon	2011	-100

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

B.

name	year	cute
drogon	2011	-100
dragon 2	2019	0

Matching: Which query matches each relation?

What do you think the **LIMIT** and **OFFSET** keywords do?

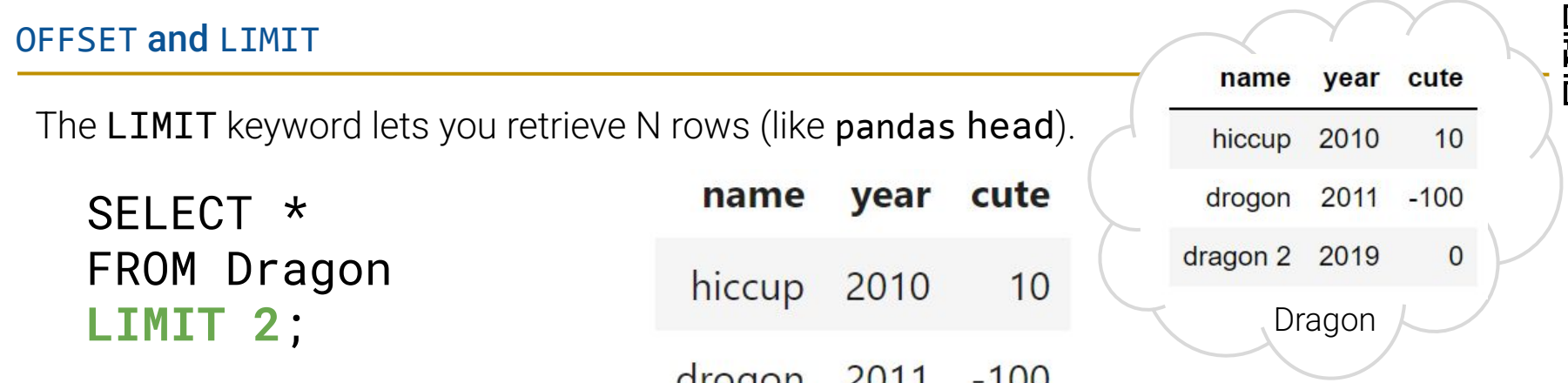
slido

3474659

Matching: Which query matches each relation?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

OFFSET and LIMIT



3474659

The **LIMIT** keyword lets you retrieve N rows (like **pandas head**).

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

name	year	cute
hiccup	2010	10
drogon	2011	-100

The **OFFSET** keyword tells SQL to skip the first N rows of the output, then apply **LIMIT**.

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

name	year	cute
drogon	2011	-100
dragon 2	2019	0

⚠️ Unless you use **ORDER BY**, there is **no guaranteed order** of rows in the relation!

New keywords

3474659

```
SELECT <column list>
FROM <table>
[WHERE <predicate>]
[ORDER BY <column list>]
[LIMIT <number of rows>]
[OFFSET <number of rows>];
```

Summary So Far

- *All* queries must include `SELECT` and `FROM`. The remaining keywords are optional.
- By convention, use **all caps** for keywords in SQL statements.
- Use **newlines** to make code more readable.

Sampling

We can use **RANDOM** or **SAMPLE** to get a sample of the dataset.

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

Randomizes the entire table (reorder rows randomly) and returns two rows as requested.

```
%%sql
SELECT *
FROM Dragon USING SAMPLE reservoir(2 ROWS) REPEATABLE (100);
```

Uses a seed to randomly draw two samples from the table.
It's more efficient than ordering the entire table using **RANDOM**.

LECTURE 20

SQL I

Data 100/Data 200, Spring 2025 @ UC Berkeley

Narges Norouzi and Josh Grossman

Content credit: Acknowledgments