



SQL with Python:

Week 2 Workshop Presentation



Workshop Agenda

Activity	Estimated Duration
Set up and check in	10 mins
Week 2 Review	60 mins
Assignment Tasks	40 mins
Break	15 mins
Remaining Assignment Tasks	100 mins
Check-Out (Feedback & Wrap-Up)	15 mins



Week 2 Review



Overview

CREATE TABLE	ORDER BY
INSERT INTO	Aggregate Functions
ALTER TABLE	GROUP BY
UPDATE	HAVING
DELETE and ON DELETE	JOIN
DROP and TRUNCATE	Set operations
SELECT	WITH
Conditional Expressions	Subqueries



Review: CREATE TABLE

Syntax:

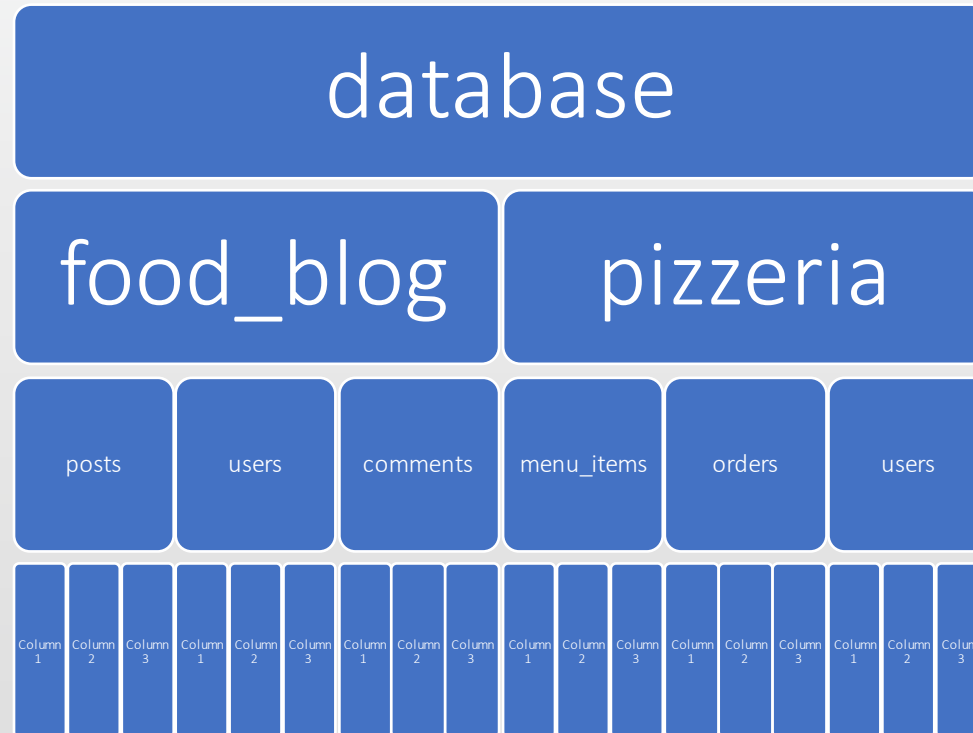
```
CREATE TABLE table_name (  
    column1 datatype column_constraints,  
    column2 datatype column_constraints,  
    column3 datatype column_constraints  
);
```

Example:

```
CREATE TABLE cars (  
    id SERIAL PRIMARY KEY,  
    year INT,  
    make TEXT NOT NULL,  
    model TEXT NOT NULL  
);
```



Review: Tables



We can think of each cell of data as living at an address specified by a unique (database, schema, table, column, row) tuple.



Review: INSERT INTO

Syntax:

```
INSERT INTO table_name (column1_name, column2_name, column3_name)  
VALUES (value1, value2, value3);
```

Example:

```
INSERT INTO cars (year, make, model)  
VALUES (2020, 'Toyota', 'Prius');
```



Review: ALTER TABLE

Examples:

```
ALTER TABLE cars  
ADD wheel_count INT NOT NULL DEFAULT 4;
```

```
ALTER TABLE accounts  
ADD CONSTRAINT fk_accounts_customers  
FOREIGN KEY (customer_id)  
REFERENCES customers;
```




Review: UPDATE

Syntax:

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE some_column = some_value;
```

Example:

```
UPDATE cars  
SET make = 'The Ford Motor Company'  
WHERE make = 'Ford';
```



Review: DELETE

Syntax:

```
DELETE FROM table_name  
WHERE some_column = some_value;
```

Example:

```
DELETE FROM cars  
WHERE year IS NULL;
```

When comparing a value against **NULL** anywhere in SQL,
use the operators **IS** and **IS NOT** instead of **=** and **!=**



Review: ON DELETE

If we delete customer #5, what happens to orders #3, #4, and #5?

orders

id	amount_spent	customer_id
1	...	7
2	...	6
3	...	5
4	...	5
5	...	5

customers

id	name	email
3
5
6
7
10

```
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  amount_spent NUMERIC NOT NULL,  
  customer_id INT NOT NULL,  
  CONSTRAINT fk_customer  
    FOREIGN KEY(customer_id)  
    REFERENCES customers(id)  
    ON DELETE CASCADE  
);
```

```
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  amount_spent NUMERIC NOT NULL,  
  customer_id INT NOT NULL,  
  CONSTRAINT fk_customer  
    FOREIGN KEY(customer_id)  
    REFERENCES customers(id)  
    ON DELETE SET NULL  
);
```

```
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  amount_spent NUMERIC NOT NULL,  
  customer_id INT NOT NULL,  
  CONSTRAINT fk_customer  
    FOREIGN KEY(customer_id)  
    REFERENCES customers(id)  
    ON DELETE SET DEFAULT  
);
```



Review: ON DELETE

If we delete customer #5, what happens to orders #3, #4, and #5?

orders

id	amount_spent	customer_id
1	...	7
2	...	6
3	...	5
4	...	5
5	...	5

customers

id	name	email
3
5
6
7
10

```
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  amount_spent NUMERIC NOT NULL,  
  customer_id INT NOT NULL,  
  CONSTRAINT fk_customer  
    FOREIGN KEY(customer_id)  
    REFERENCES customers(id)  
    ON DELETE CASCADE  
);
```

```
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  amount_spent NUMERIC NOT NULL,  
  customer_id INT NOT NULL,  
  CONSTRAINT fk_customer  
    FOREIGN KEY(customer_id)  
    REFERENCES customers(id)  
    ON DELETE SET NULL  
);
```

```
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  amount_spent NUMERIC NOT NULL,  
  customer_id INT NOT NULL,  
  CONSTRAINT fk_customer  
    FOREIGN KEY(customer_id)  
    REFERENCES customers(id)  
    ON DELETE SET DEFAULT  
);
```



Review: DROP and TRUNCATE

```
DROP TABLE cars;
```

Deletes a table and all its rows

```
DROP DATABASE week2;
```

Deletes a database and all its tables and rows

```
TRUNCATE TABLE cars;
```

Deletes all rows in a table but not the table itself



Review: SELECT

Syntax:

```
SELECT column_name1, column_name2 FROM table_name;
```

Example:

```
SELECT title, author FROM books;
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	title text	author text
1	Frankenstein	Mary Shelley
2	The Great Gatsby	F. Scott Fitzgerald
3	Big Fish	Daniel Wallace
4	Don Quixote	Miguel de Cervantes



Review: SELECT ... as ...

Change column names in result set if needed
by using the **as** keyword to create aliases

Example:

```
SELECT author as book_author, title as book_title FROM books;
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	book_author text	book_title text
1	Mary Shelley	Frankenstein
2	F. Scott Fitzgerald	The Great Gatsby
3	Daniel Wallace	Big Fish
4	Miguel de Cervantes	Don Quixote



Review: SELECT ... as ...

Using the * **wildcard character** instead of column names will result in all columns being selected, i.e. the entire table
Generally not recommended

Example:

```
SELECT * from books;
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605



Review: Conditional expressions

Comparison and logical operators used in conditional expressions

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal
AND	Logical operator AND
OR	Logical operator OR
IN	Return true if a value matches any value in a list
BETWEEN	Return true if a value is between a range of values
LIKE	Return true if a value matches a pattern
IS NULL	Return true if a value is NULL
NOT	Negate the result of other operators



Review: Conditional expressions

WHERE

Use to filter SELECT query results based on condition

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

```
SELECT title, author FROM books WHERE genre = 'Novel';
```

	title text	author text
1	Frankenstein	Mary Shelley
2	The Great Gatsby	F. Scott Fitzgerald
3	Don Quixote	Miguel de Cervantes

```
SELECT title, author, year  
FROM books  
WHERE genre = 'Novel' AND year < 1900;
```

	title text	author text	year integer
1	Frankenstein	Mary Shelley	1818
2	Don Quixote	Miguel de Cervantes	1605



Review: Conditional Expressions

DISTINCT

Eliminates duplicate results from result set

Example:

```
SELECT DISTINCT genre FROM books;
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	genre text
1	Novel
2	Magical Realism



Review: Conditional Expressions

LIMIT

Cap number of results in result set

Example:

```
SELECT DISTINCT genre
FROM books LIMIT 1;
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	genre text
1	Novel



Review: Conditional Expressions

BETWEEN

Selects rows where specified column value falls within a range, inclusive

Example:

```
SELECT b.title AS twentieth_century_books
FROM books b
WHERE b.year BETWEEN 1900 AND 1999;
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	twentieth_century_books text
1	The Great Gatsby
2	Big Fish



Review: Conditional Expressions

LIKE & ILIKE: performs pattern matching based on wildcards

% matches any sequence of 0 or more characters

_ matches any single character

LIKE is case-sensitive, ILIKE is not

Examples:

"Book titles that begin with T"

```
SELECT b.title FROM books b
WHERE b.title LIKE 'T%';
```



	title	
	text	🔒
1	The Great Gatsby	

"Book titles that contain T"

```
SELECT b.title FROM books b
WHERE b.title LIKE '%T%';
```



	title	
	text	🔒
1	The Great Gatsby	

"Book titles that contain T or t"

```
SELECT b.title FROM books b
WHERE b.title ILIKE '%T%';
```



	title	
	text	🔒
1	Frankenstein	
2	The Great Gatsby	
3	Don Quixote	



Review: Conditional Expressions

IN

Evaluates as TRUE if list of options contains specified value

Example:

```
SELECT b.title, b.year FROM books b
WHERE b.year IN (1990, 1986, 1996, 1998);
```

books table

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

result set

	title text	year integer
1	Big Fish	1998



Review: ORDER BY

Syntax:

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    sort_expression1 [ASC | DESC],
    ...
    sort_expressionN [ASC | DESC];
```

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

Example:

```
SELECT * FROM books
ORDER BY year;
```



	id [PK] integer	title text	author text	genre text	year integer
1	4	Don Quixote	Miguel de Cervantes	Novel	1605
2	1	Frankenstein	Mary Shelley	Novel	1818
3	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
4	3	Big Fish	Daniel Wallace	Magical Realism	1998



Review: Aggregate Functions

Function	Description
COUNT	Count of the column specified (includes NULL values if * is used)
MAX	Maximum value in the column (excludes NULL values)
MIN	Minimum value in the column (excludes NULL values)
AVG	Average value in the column (excludes NULL values, only works on numeric data types)
SUM	Sum of values in the column (excludes NULL values, only works on numeric data types)

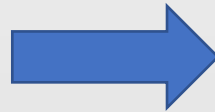


Review: Aggregate Functions

Examples:

"Count of all books"

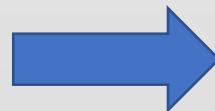
```
SELECT COUNT(*)  
AS book_count  
FROM books;
```



	book_count	
	bigint	🔒
1	4	

"Year of the oldest book"

```
SELECT MIN(b.year)  
AS year_of_oldest_book  
FROM books b;
```



	year_of_oldest_book	
	integer	🔒
1	1605	



Review: GROUP BY

Syntax:

```
SELECT
    column_1,
    column_2,
    ...,
    aggregate_function(column_3)
FROM
    table_name
GROUP BY
    column_1,
    column_2,
    ...;
```

Count number of books per genre
Return a single row for each group

Example:

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

```
SELECT genre, COUNT(*) AS book_count
FROM books GROUP BY genre;
```



	genre text	book_count bigint
1	Novel	3
2	Magical Realism	1



Review: HAVING

Syntax:

```
SELECT
    column_1,
    column_2,
    ...,
    aggregate_function(column_3)
FROM
    table_name
GROUP BY
    column_1,
    column_2,
    ...
HAVING
    conditional_expression;
```

Example:

	id [PK] integer	title text	author text	genre text	year integer
1	1	Frankenstein	Mary Shelley	Novel	1818
2	2	The Great Gatsby	F. Scott Fitzgerald	Novel	1925
3	3	Big Fish	Daniel Wallace	Magical Realism	1998
4	4	Don Quixote	Miguel de Cervantes	Novel	1605

```
SELECT genre, COUNT(*) AS book_count
FROM books
GROUP BY genre
HAVING (COUNT(*)) > 1;
```



	genre text	book_count bigint
1	Novel	3

Count books per genre such that
the count is greater than 1



Review: Set operations

Allow us to query from more than 1 table at a time

- **UNION**: Combines result set from 2 SELECT queries, removes duplicates, returns combined result set
- **UNION ALL**: Same as UNION, but does not remove duplicates
- **INTERSECT**: Combines result set from 2 SELECT queries and eliminates *unique* rows
- **EXCEPT**: Takes 2 result sets from SELECT queries, returns first result set after eliminating any rows matching those of second result set



Review: JOIN

Query from multiple tables based on values of common columns between related tables

Syntax:

```
SELECT select_list  
FROM left_table  
[INNER | LEFT | RIGHT | FULL] JOIN right_table  
ON left_column = right_column;
```

Default type is INNER JOIN:

1. Selects rows from left_table
2. Compares left_column to right_column
3. If equal, adds selected columns to result set
4. Otherwise, skips



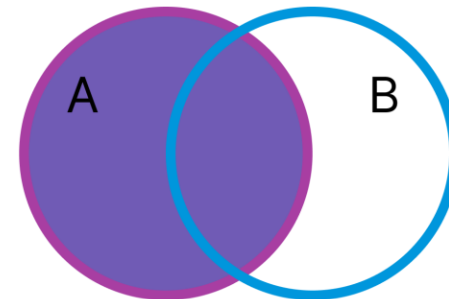
Review: LEFT JOIN

Result guaranteed to contain every row from left_table

1. Selects rows from left_table
2. Compares left_column to right_column
3. If equal, adds selected columns to result set
4. Otherwise, adds selected columns from left_table with NULL placeholders for columns in right_table

```
SELECT select_list  
FROM left_table  
[INNER | LEFT | RIGHT | FULL] JOIN right_table  
ON left_column = right_column;
```

left join



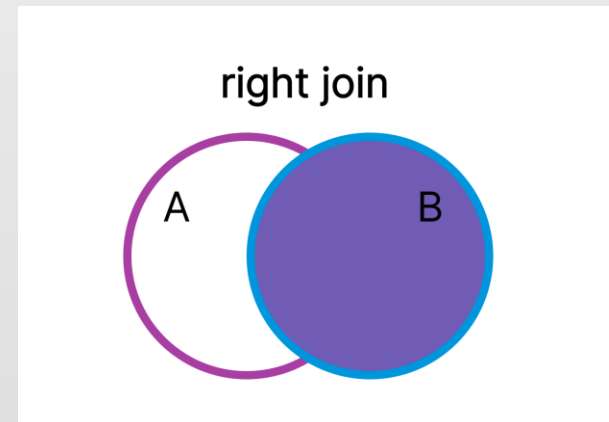


Review: RIGHT JOIN

Result guaranteed to contain every row from right_table

1. Selects rows from right_table
2. Compares left_column to right_column
3. If equal, adds selected columns to result set
4. Otherwise, adds selected columns from right_table with NULL placeholders for columns in left_table

```
SELECT select_list  
FROM left_table  
[INNER | LEFT | RIGHT | FULL] JOIN right_table  
ON left_column = right_column;
```

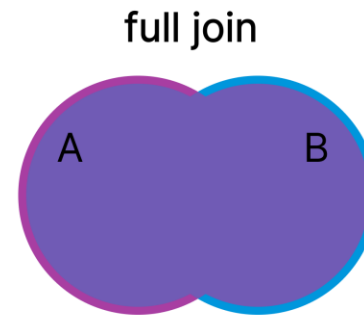




Review: FULL JOIN

Selects all rows from both
left_table and
right_table

Selects columns from
corresponding joined
table if match found,
else adds NULL
placeholder values



```
SELECT select_list  
FROM left_table  
[INNER | LEFT | RIGHT | FULL] JOIN right_table  
ON left_column = right_column;
```



Review: Subqueries

- Subqueries are nested queries
- Can use SQL logical operators: **EXISTS, ANY, ALL, IN, NOT IN**

EXISTS: Returns True if subquery returns at least 1 row

IN: Returns True if some value is in subquery's result set

NOT IN: Returns True if some value is NOT in subquery's result set

ANY: Used with a comparison operator, checks against values in subquery result set, returns True if **any** comparison evaluates as true.

ALL: Used with a comparison operator, checks against values in subquery result set, returns True if **all** comparisons evaluate as True.



Review: WITH (CTE)

- WITH keyword is used to create temporary tables called Common Table Expressions, discarded at end of query
- Break down complex queries into smaller, more manageable parts

"For the top sales regions (top 10% in total sales), find the total units sold and the total sales for each product"

```
WITH regional_sales AS (  
    SELECT region, SUM(amount) AS total_sales  
    FROM orders  
    GROUP BY region  
) , top_regions AS (  
    SELECT region  
    FROM regional_sales  
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)  
)  
SELECT region,  
    product,  
    SUM(quantity) AS product_units,  
    SUM(amount) AS product_sales  
FROM orders  
WHERE region IN (SELECT region FROM top_regions)  
GROUP BY region, product;
```



Workshop 2 Assignment

Goal: Continue to help the fictional company Northwind Traders with their database

Task 1: Write SQL queries for a multitude of situations

Task 2: Write queries to make modifications to the database structure, as well as data values.

You will be split up into groups to work on the assignment together. Talk through each step out loud with each other, code collaboratively. If your team spends more than 10 minutes trying to solve one problem, ask your instructor for help!