

# Introduction to Relational Databases and SQL

# What is a Relational Database?

- Relational databases store related data in **tabular form**
- Relationships between tables are defined with **keys** (primary and foreign)
- Most relational databases are created for use by applications and optimized for **CRUD *transactions***
  - Create
  - Read
  - Update
  - Delete
- Data Analysts/Data Scientists mostly focused on **Read** - querying the data. They want to look at the data and perform calculations without making any permanent changes.



# What is SQL

SQL stands **S**tructured **Q**uery **L**anguage. It is the language you use to interact with a database. It allows you to write out what you want to search for, goes to a database that you specify, then returns those results to you.

The Structured part of SQL means queries are written in a specific format using specific keywords.

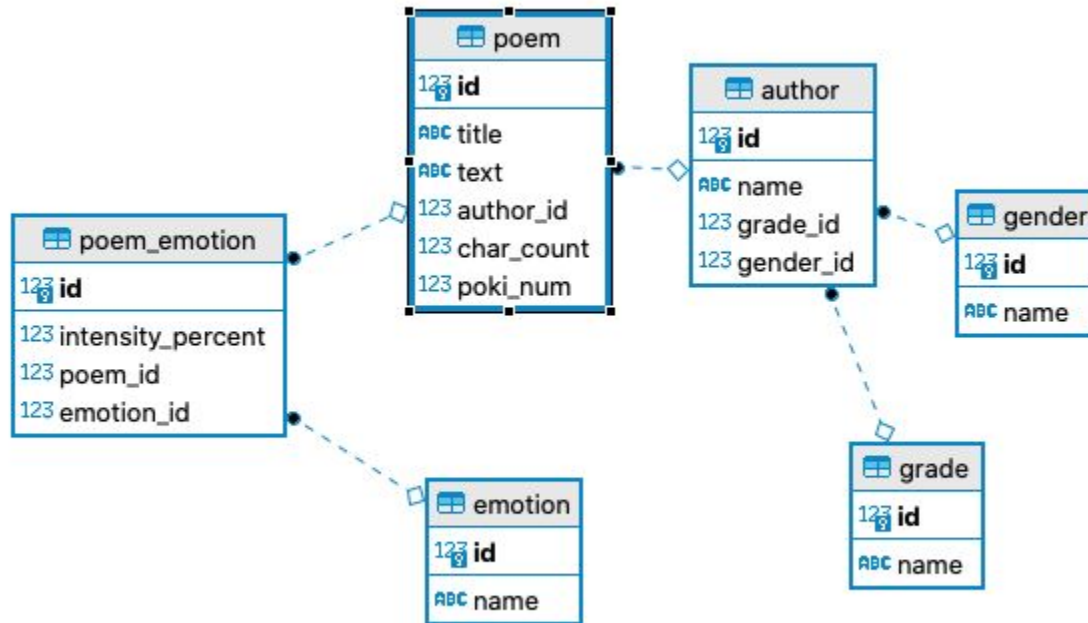


# Keywords

- **SELECT**
- **FROM**
- **AS**
- **LIMIT**
- **DISTINCT**
- **COUNT**
- **WHERE**
- **AND**
- **OR**
- **BETWEEN**
- **IN**
- **(NOT) NULL**
- **LIKE**
- **AVG(), SUM(), MAX(), MIN(), etc.**
- **ORDER BY**
- **GROUP BY**
- **HAVING**



For our learning examples, we will be using the PoetryKids database. Below is an Entity Relationship Diagram (ERD) that shows the tables in the database and the relationships between tables.

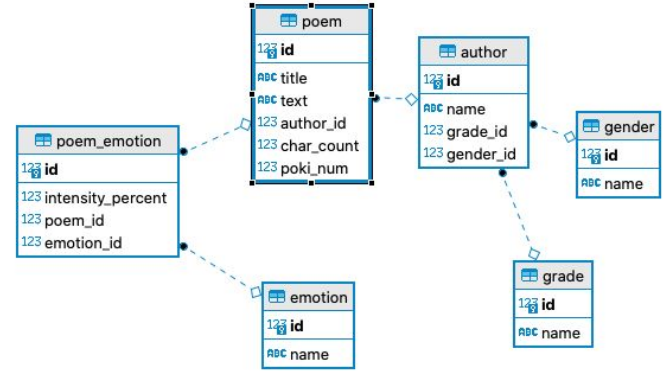


# Format of a Query

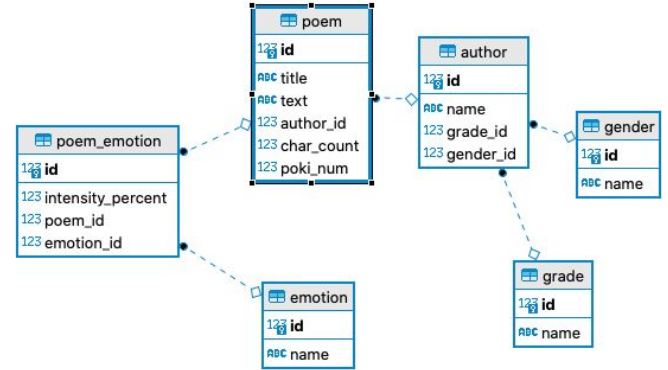
Keywords all caps

```
SELECT text  
FROM poem;
```

End with semicolon

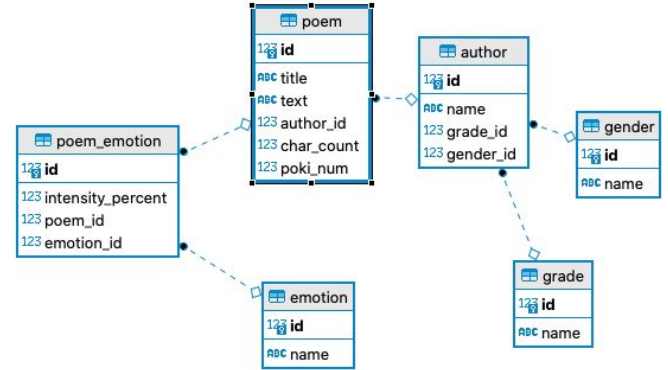


# Format of a Query



```
SELECT text
FROM poem
WHERE char_count < 10;
```

# Format of a Query



```
SELECT text
FROM poem
WHERE char_count < 10
LIMIT 5;
```



Queries are written following this basic format:

**SELECT** *columns*  
**FROM** *table*  
**WHERE** *condition*  
**GROUP BY** *columns*  
**HAVING** *condition*  
**ORDER BY** *columns*

It's helpful to know that the query **executes** in a different **order**:

**FROM**  
**WHERE**  
**GROUP BY**  
**HAVING**  
**SELECT**

<https://jvns.ca/blog/2019/10/03/sql-queries-don-t-start-with-select/>

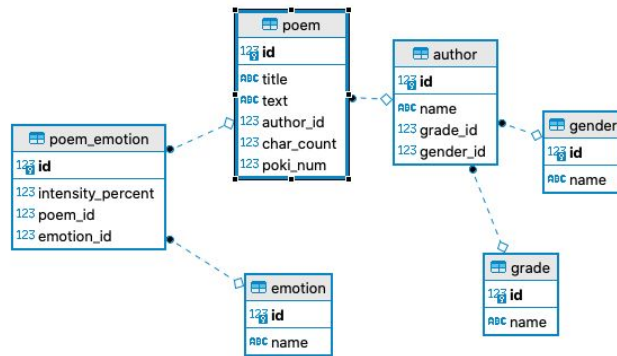


# Selecting all columns from a table

When writing a query you indicate what columns you want to see. These directions go in the **SELECT** statement. A shorthand to **SELECT ALL** is to use a **\***:

```
SELECT *  
FROM emotion;
```

	id [PK] integer	name character varying
1	1	Anger
2	2	Fear
3	3	Sadness
4	4	Joy

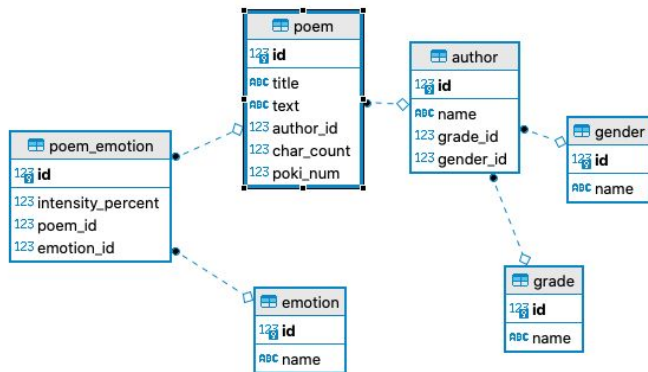


# Selecting single columns from a table

You can also specify individual columns to return, each separated by a ',':

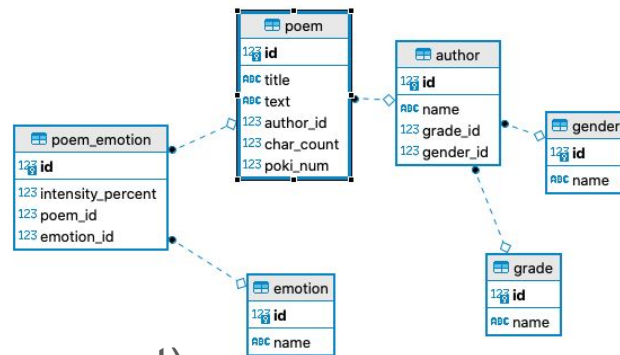
```
SELECT name, grade_id  
FROM author  
LIMIT 5;
```

	<b>name</b> character varying	<b>grade_id</b> integer
1	a	1
2	aab	1
3	aadhya	1
4	aaliyah	1
5	aanna	1



# Aggregating Data

You may want to summarize your data in different ways.  
For example you could **COUNT**, **SUM**, **AVERAGE**, or  
find the **MAX** or **MIN**:



```
SELECT COUNT(id)
FROM author
WHERE grade_id = 4;
```

	count
	bigint
1	3288

```
SELECT AVG(intensity_percent)
FROM poem_emotion
WHERE emotion_id = 2;
```

	avg
	numeric
1	45.4740880030086499

# Joining Data

Often data will be spread across **multiple tables**. In order to perform an analysis you may need to **combine the data from two or more tables**. Joins in SQL are just like merging with pandas.

Table A		
id	col_1	col_2
1	23-B	12
2	435	45
3	AB145	23
4	BB	56
5	435	123

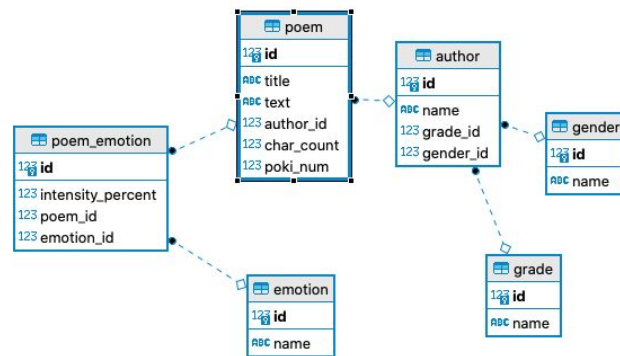


merge_table			
id	col_1	col_2	col_a
1	23-B	12	a
2	435	45	a
3	AB145	23	b
4	BB	56	b
5	435	123	a

Table B		
id	lookup_id	col_a
a1	1	a
a2	2	a
b1	3	b
b2	4	b
a3	5	a



# Joining Data - example

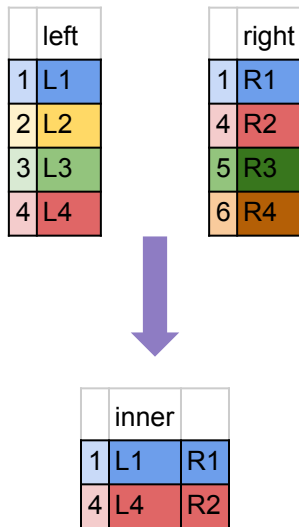


```
SELECT poem.title, author.name
FROM poem JOIN author ON poem.author_id = author.id;
```

# INNER JOIN

Combining data from a **left** and **right** table is called **JOIN**ing. There are multiple kinds of joins and each one combines the data in a different way.

An **INNER JOIN** keeps **only the rows that have matching values in both tables**. This is the default type of join if you just use the **JOIN** keyword in your query.



# LEFT JOIN and RIGHT JOIN

A **LEFT JOIN** keeps **all** rows from the left table and all **matching** rows from the right table. A **RIGHT JOIN** works similarly, except all rows from the right table are kept.

left		right	
1	L1	1	R1
2	L2	4	R2
3	L3	5	R3
4	L4	6	R4



left		
1	L1	R1
2	L2	
3	L3	
4	L4	R2

left		right	
1	L1	1	R1
2	L2	4	R2
3	L3	5	R3
4	L4	6	R4

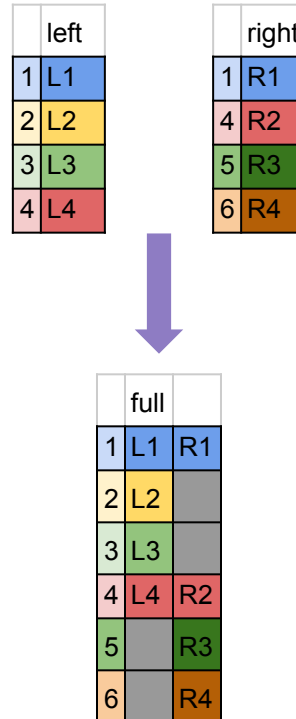


right		
1	L1	R1
4	L4	R2
5		R3
6		R4



# FULL JOIN (Sometimes called An OUTER JOIN depending on the SQL dialect)

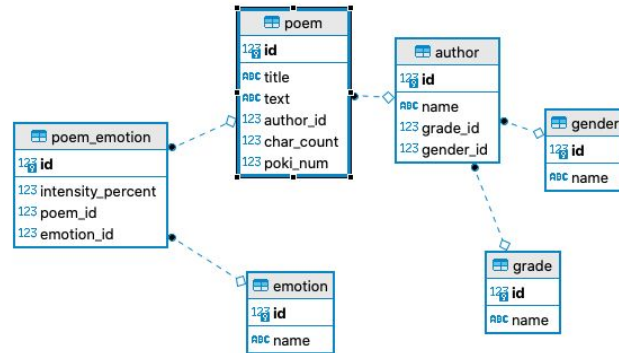
A **FULL JOIN** keeps **all rows from both tables**:



# Aggregating Data with GROUP BY

The **GROUP BY** keyword will subdivide the table based on the specified columns. You can then perform aggregations on the subgroups:

```
SELECT AVG(char_count)
FROM poem JOIN author on poem.author_id = author.id
GROUP BY grade_id;
```



	avg numeric	
1	189.8826185101580135	
2	179.7968655816757083	
3	228.0627091773824203	
4	201.2769757542106237	
5	165.7313291139240506	

# Finding unique values

Sometimes a particular column or a calculation will result in duplicate values. To get just unique values:

```
SELECT COUNT (DISTINCT name)  
FROM author;
```

	count bigint
1	7403

# Selecting/avoiding null values

Null values will likely exist in any data set you work with. It will be useful to identify or exclude records with null values:

```
SELECT title, char_count  
FROM poem  
WHERE title IS NOT NULL  
LIMIT 5;
```

	<b>title</b> character varying	<b>char_count</b> integer
1	Computer	106
2	Angel	164
3	Nature Nature and Nat...	491
4	Jack	74
5	When I awoke one mor...	325



## Additional Practice

W3Schools SQL Tutorial - <https://www.w3schools.com/sql/default.asp>

Khan Academy - <https://www.khanacademy.org/computing/computer-programming/sql>

DataCamp - <https://learn.datacamp.com/skill-tracks/sql-fundamentals>

