Writing Better SQL In 90 Minutes

Alice Zhao

Alice Zhao

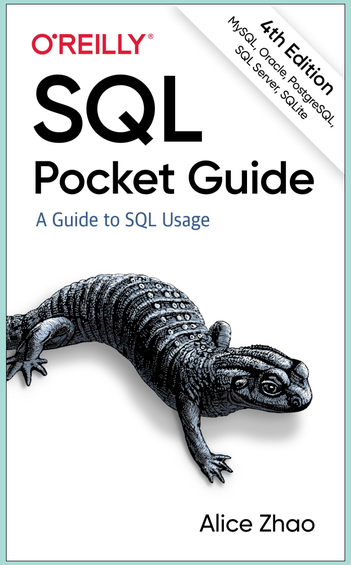
A Dash of Data

Current Role

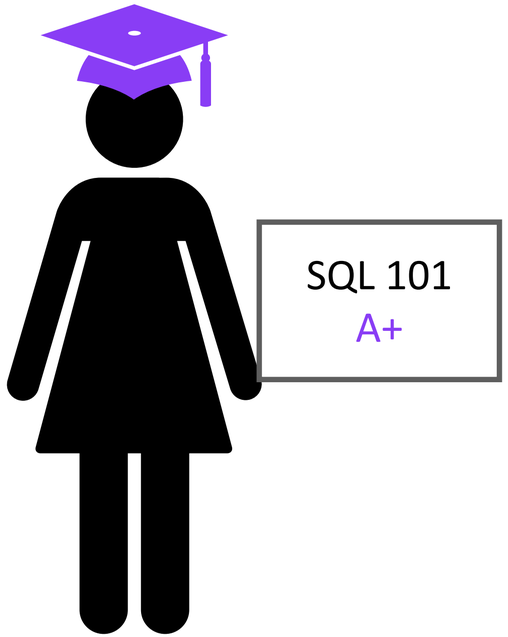
Data Science & Analytics Instructor

Past Roles

- Data Scientist
- Analyst
- Consultant
- Engineer
- Author



2013



REDFIN

“You need to learn how to write better SQL.”

Experience.

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Poll: What is your experience with SQL?

- None: I have no SQL experience
- Beginner: I am currently learning SQL / I can write simple queries
- Intermediate: I regularly write lengthy SQL queries
- Expert: I have many years of SQL experience

The process of creating anything new

1. Make something that works
2. Make it better

Cooking Example

Imagine:

You want to make pancakes.

Step 1: Make something that works

@thespruceeats

@cooksillustrated

@cooksillustrated

These pancakes taste good,

but I think I can do better.

Beginner

Step 2: Make it better

how do i make fluffier pancakes

People also ask

How do you make pancakes light and fluffy?

HERE ARE SOME TIPS FOR FLUFFY PANCAKES:

1. DON'T OVER MIX THE BATTER. Why do pancake recipes always tell you not to overmix the batter? Gluten in the flour is the answer! ...

Expert

Step 2: Make it better

I know that…

eggs / flour / milk + baking powder —> makes bubbles

I want to keep the bubbles, so I should leave my batter lumpy —> fluffier

The process of making pancakes

1. Make something that works
2. Make it better

Beginner *— Google it*

Expert *— Understand what's happening and make a change*

SQL Example


YouTube
SQL
SQL Crash Course for Beginners
Presenter: Alice Zhao
Data Science Educator & Author at O'Reilly Media
58:57

Customers

name	state
Calvin	California
Cameron	California
Camila	California
Colton	Colorado
Cora	Colorado

5 million rows

+

Employees

name	state
Tim	Texas
Vernon	Vermont
Wyatt	Wyoming

1 million rows

=

All Names

name	state
Calvin	California
Cameron	California
Camila	California
Colton	Colorado
Cora	Colorado
Tim	Texas
Vernon	Vermont
Wyatt	Wyoming

6 million rows

Goal:

Create a giant list of all customers and employees.

Step 1: Make something that works

```
-- Return all names and states
SELECT name, state FROM customers
UNION
SELECT name, state FROM employees
```

name	state
Calvin	California
Cameron	California
Camila	California
Colton	Colorado
Cora	Colorado
Tim	Texas
Vernon	Vermont
Wyatt	Wyoming

6 million rows

UNION:
combines the results of two
or more SELECT statements

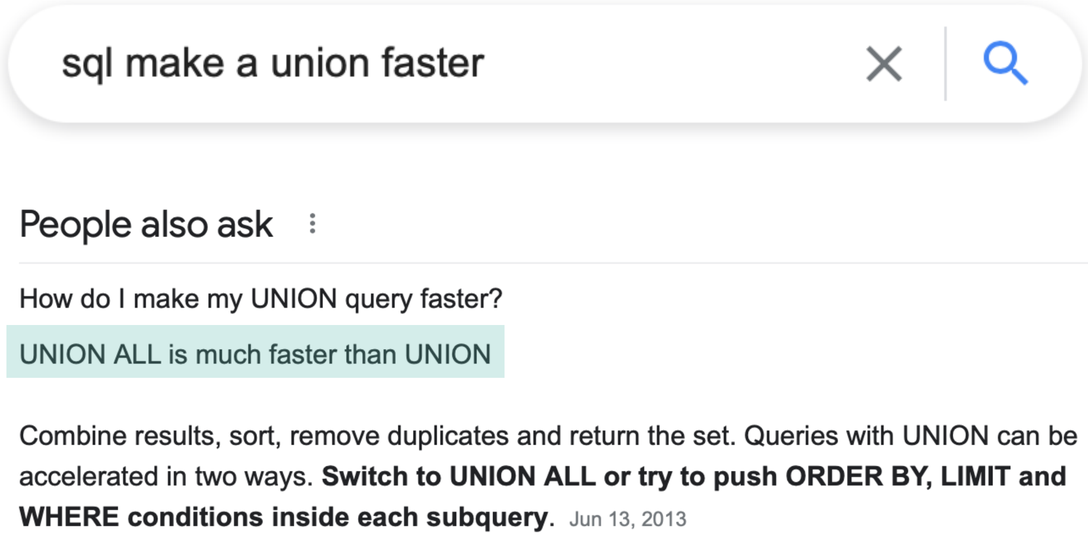
7 seconds

Pulse Check

- Who has encountered the “slow turtle” before?
- Share your experiences in the chat

Beginner

Step 2: Make it better




```
-- Return all names and states
SELECT name, state FROM customers
UNION ALL
SELECT name, state FROM employees
```

2 ms

UNION ALL:
combines the results of two or more
SELECT statements (allows duplicate values)

Customers

name	state
Calvin	California

+

Employees

name	state
Calvin	California
Wyatt	Wyoming

=

UNION

name	state
Calvin	California
Wyatt	Wyoming

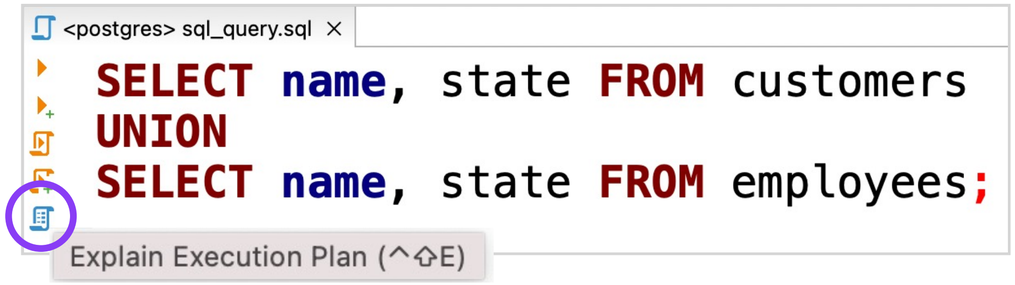
UNION ALL

name	state
Calvin	California
Calvin	California
Wyatt	Wyoming

Expert

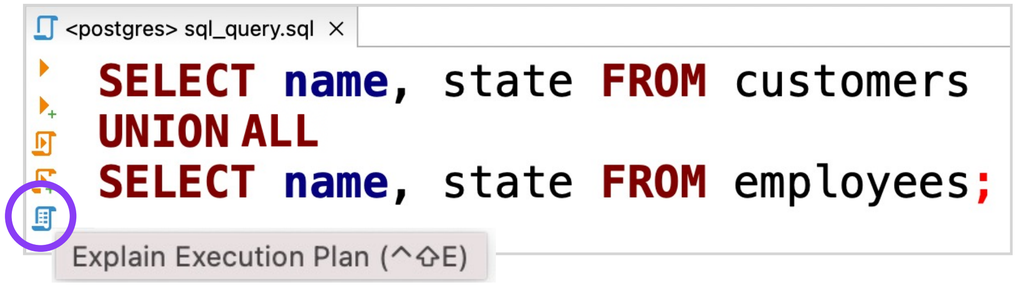
Step 2: Make it better

Let's see what's happening behind the scenes…



Node Type	Entity	Rows	Time
Unique		104110	9111.091
Sort		7472859	8214.594
Append		7472859	1761.944
Seq Scan	customers	5647426	759.655
Seq Scan	employees	1825433	239.044

7 seconds



Node Type	Entity	Rows	Time
Append		7472859	1765.632
Seq Scan	customers	5647426	763.204
Seq Scan	employees	1825433	244.862

3500x as fast!

2 ms

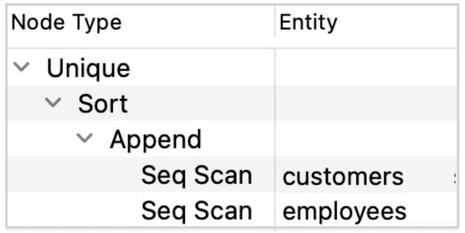
Expert

Step 2: Make it better

Execution plan comparison

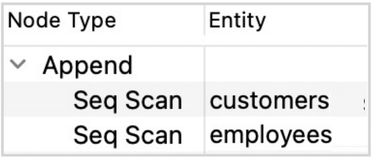
UNION: returns unique values

1. Stack tables
2. Sort data
3. Remove duplicates



UNION ALL: returns all values

1. Stack tables
2. ~~Sort data~~
3. ~~Remove duplicates~~



If you know with certainty that there are no duplicate rows, using a UNION ALL is faster.

Customers

name	state
Calvin	California
Cameron	California
Camila	California
Colton	Colorado
Cora	Colorado

5 million rows

+

Employees

name	state
Tim	Texas
Vernon	Vermont
Wyatt	Wyoming

1 million rows

=

All Names

name	state
Calvin	California
Cameron	California
Camila	California
Colton	Colorado
Cora	Colorado
Tim	Texas
Vernon	Vermont
Wyatt	Wyoming

6 million rows

Goal:

Create a giant list of all customers and employees.

```
-- Return all names and states
SELECT name, state FROM customers
UNION ALL
SELECT name, state FROM employees
```

The process of writing SQL queries

1. Make something that works
2. Make it better

Beginner *— Google it*

Expert *— Understand what's happening and make a change*

The process of making pancakes

1. Make something that works
2. Make it better

Beginner *— Google it*

Expert *— Understand what's happening and make a change*

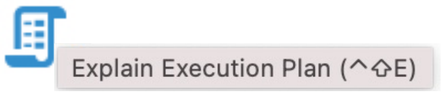
Introduction Takeaways

1. Make something that works

- Know that your code won't be perfect and expect to mess up
- Making something work *is more important than* making something optimal

2. Make it better

- Use Google to figure out a quick solution
- Look at the *execution plan* to understand what's happening and compare the execution times of code blocks



Q&A

Make It Work, Then Make it Better

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Tips for Writing Better SQL

Assumption

You've already written a working SQL query.

Better?

- Easier to read
- Runs faster

I. Write Code for Humans

II. Only Do What's Necessary

III. Break a Big Problem Into Smaller Pieces

IV. When In Doubt, Test It Out

Baby Names from Social Security Card Applications

The data (name, year of birth, sex, state, and number) are from a 100 percent sample of Social Security card applications starting with 1910.

DATA.GOV

baby_names

Name	Year	Gender	State	Num_Babies
Mary	1910	F	AK	14
Annie	1910	F	AK	12
Anna	1910	F	AK	10
Margaret	1910	F	AK	8
Helen	1910	F	AK	7

5 million rows

state_details

State	Abbreviation	Capital	Largest_City	Population
Alabama	AL	Montgomery	Huntsville	5,024,279
Alaska	AK	Juneau	Anchorage	733,391
Arizona	AZ	Phoenix	Phoenix	7,151,502
Arkansas	AR	Little Rock	Little Rock	3,011,524
California	CA	Sacramento	Los Angeles	39,538,223

51 rows

I. Write Code for Humans

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

— Martin Fowler, software developer

Tip #1: Format your code

```
select NAME, year, sd.state, Population, NUM_babies From baby_names AS bn
LEFT join state_details sd on bn.state=sd.abbreviation;
```

Group Discussion: What would you do to make this code look better?

```
SELECT bn.name, bn.year,
       sd.state, sd.population,
       bn.num_babies
FROM   baby_names bn LEFT JOIN state_details sd
       ON bn.state = sd.abbreviation;
```

- ✓ Consistent capitalization
- ✓ Indentation and spaces
- ✓ Table name prefix for each column
- ✓ Consistent aliasing

Tip #2: Add comments

```sql
SELECT   name,
         SUM(num_babies) AS total
FROM     baby_names
WHERE    year BETWEEN 1990 AND 1999
GROUP BY name
ORDER BY total DESC
LIMIT 5;
```

```sql
-- Top 5 most popular names of the 90s
SELECT   name,
         SUM(num_babies) AS total
FROM     baby_names
WHERE    year BETWEEN 1990 AND 1999
GROUP BY name
ORDER BY total DESC
LIMIT 5;
```

```sql
/*
Author: Alice Zhao
Description: Pull all Illinois data
Modified Date: 2/2/2022
*/
SELECT *
FROM state_details
WHERE abbreviation = 'IL';
```

-- for single line comments

/* */ for multi-line comments

I. Write Code for Humans

- Tip #1: Format your code
  - ✓ Consistent capitalization
  - ✓ Indentation and spaces
  - ✓ Table name prefix for each column
  - ✓ Consistent aliasing
- Tip #2: Add comments
  - ✓ -- for single line comments
  - ✓ /* */ for multi-line comments

II. Only Do What’s Necessary

Tip #3: Take only what you need

You are looking at a table for the first time.

```
SELECT * FROM baby_names;
```

3 seconds

Limit the number of ***rows***

Group Discussion: How could you do this?

```
SELECT * FROM baby_names
LIMIT 100;
```

3 ms

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

100 ms

- ✓ LIMIT
- ✓ COUNT
- ✓ WHERE
- ✓ RANDOM

```
SELECT * FROM baby_names
WHERE year = 2000;
```

300 ms

```
SELECT * FROM baby_names
ORDER BY RANDOM() LIMIT 100;
```

1.5 seconds

Tip #3: Take only what you need

Limit the number of ***columns***

```sql
-- All columns from the joined tables
SELECT *
FROM   baby_names bn
       LEFT JOIN state_details sd
       ON bn.state = sd.abbreviation;
```

8 seconds

```sql
-- Subset of columns from the joined tables
SELECT bn.name, bn.year,
       sd.state, sd.population,
       bn.num_babies
FROM   baby_names bn
       LEFT JOIN state_details sd
       ON bn.state = sd.abbreviation;
```

4 seconds

✓ Specify the columns of interest

Tip #4: Avoid unnecessary calculations

When using the DISTINCT keyword, think about whether you really need it

```sql
-- List the top popular names of the year
SELECT DISTINCT name, gender, state, year, num_babies
FROM baby_names
ORDER BY num_babies DESC
LIMIT 5;
```

4 seconds

name	gender	state	year	num_babies
Robert	M	NY	1947	10023
John	M	NY	1947	9634
Robert	M	NY	1946	9296
Michael	M	NY	1963	9250
Robert	M	NY	1952	9208

```sql
-- List the top popular names of the year
SELECT name, gender, state, year, num_babies
FROM baby_names
ORDER BY num_babies DESC
LIMIT 5;
```

0.5 seconds

name	gender	state	year	num_babies
Robert	M	NY	1947	10023
John	M	NY	1947	9634
Robert	M	NY	1946	9296
Michael	M	NY	1963	9250
Robert	M	NY	1952	9208

Tip #4: Avoid unnecessary calculations

When using the wildcard ‘%’ to search, think about whether you really need it

```sql
-- Find baby names that start with Lil
SELECT   name,
         COUNT(num_babies) AS babies
FROM     baby_names
WHERE    name LIKE '%Lil%'
GROUP BY name
ORDER BY babies DESC;
```

0.5 seconds

name	babies
Lillian	4411
Lila	2685
Lillie	2472
Lily	2236
Lilly	1807
Delilah	1017

```sql
-- Find baby names that start with Lil
SELECT   name,
         COUNT(num_babies) AS babies
FROM     baby_names
WHERE    name LIKE 'Lil%'
GROUP BY name
ORDER BY babies DESC;
```

0.35 seconds

name	babies
Lillian	4411
Lila	2685
Lillie	2472
Lily	2236
Lilly	1807
Liliana	945

II. Only Do What's Necessary

- Tip #3: Take only what you need
  - ✓ Limit the number of rows — LIMIT, COUNT, WHERE, RANDOM
  - ✓ Limit the number of columns — specify the columns of interest
- Tip #4: Avoid unnecessary calculations
  - ✓ DISTINCT
  - ✓ Wildcard '%'
  - ✓ and more…

III. Break a Big Problem Into Smaller Pieces

Clause Order vs Execution Order

Clause Order

SELECT	columns to display
FROM	table(s) to pull from
WHERE	filter rows
GROUP BY	split rows into groups
HAVING	filter grouped rows
ORDER BY	columns to sort

Clause Order

SELECT	columns to display
FROM	table(s) to pull from
WHERE	filter rows
GROUP BY	split rows into groups
HAVING	filter grouped rows
ORDER BY	columns to sort

Execution Order

FROM	table(s) to pull from
WHERE	filter rows
GROUP BY	split rows into groups
HAVING	filter grouped rows
	columns to display
ORDER BY	columns to sort

Tip #5: First Reduce, Then Combine

baby_names

State	Year	Gender	Name	Num_Babies
CA	1950	F	Mary	3,134
IL	2012	F	Sophia	959
NY	1975	M	Richard	1,569

state_details

Abbreviation	State
CA	California
CO	Colorado

Goal

State	Year	Name	Total
California	1990	Michael	8306
California	1991	Michael	7627
California	1991	Jessica	6994

```sql
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         SUM(bn.num_babies) AS total          2b
FROM     baby_names bn
         LEFT JOIN state_details sd           1
         ON bn.state = sd.abbreviation
WHERE    year BETWEEN 1990 AND 1999           2a
GROUP BY sd.state, bn.year, bn.name
ORDER BY total DESC;                          3
```

2.5 sec

✓ Combine 2 tables
✓ Only look at 90s names
✓ Sum girl and boy names
✓ Order by popularity

1. Combine (2 large tables)
2. Filter + calculate
3. Sort

Tip #5: First Reduce, Then Combine

```sql
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         SUM(bn.num_babies) AS total
FROM     baby_names bn
         LEFT JOIN state_details sd
         ON bn.state = sd.abbreviation
WHERE    year BETWEEN 1990 AND 1999
GROUP BY sd.state, bn.year, bn.name
ORDER BY total DESC;
```

2.5 sec

1. Combine (2 large tables)
2. Filter + calculate
3. Sort

Subquery

```sql
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         total
FROM     (SELECT state, year, name,
                 SUM(num_babies) as total
          FROM baby_names                                  1
          WHERE year BETWEEN 1990 and 1999
          GROUP BY state, year, name) AS bn
         LEFT JOIN state_details sd
         ON bn.state = sd.abbreviation                     2
ORDER BY total DESC;                                       3
```

1.5 sec

1. Filter + calculate
2. Combine (2 smaller tables)
3. Sort

Tip #5: First Reduce, Then Combine

```sql
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         SUM(bn.num_babies) AS total
FROM     baby_names bn
         LEFT JOIN state_details sd
         ON bn.state = sd.abbreviation
WHERE    year BETWEEN 1990 AND 1999
GROUP BY sd.state, bn.year, bn.name
ORDER BY total DESC;
```

2.5 sec

1. Combine (2 large tables)
2. Filter + calculate
3. Sort

Subquery

```sql
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         total
FROM     (SELECT state, year, name,
                 SUM(num_babies) as total
          FROM baby_names
          WHERE year BETWEEN 1990 and 1999
          GROUP BY state, year, name) AS bn
          LEFT JOIN state_details sd
          ON bn.state = sd.abbreviation
ORDER BY total DESC;
```

1.5 sec

1. Filter + calculate
2. Combine (2 smaller tables)
3. Sort

Tip #6: Use a CTE as a Subquery Alternative

Subquery

```
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         total
FROM     (SELECT state, year, name,
                 SUM(num_babies) as total
          FROM baby_names
          WHERE year BETWEEN 1990 and 1999
          GROUP BY state, year, name) AS bn
          LEFT JOIN state_details sd
          ON bn.state = sd.abbreviation
ORDER BY total DESC;
```

Common Table Expression (CTE)

```
-- Popular names in each state in the 90s
WITH bn AS (SELECT state, year, name,
                   SUM(num_babies) as total
            FROM baby_names
            WHERE year BETWEEN 1990 and 1999
            GROUP BY state, year, name)

SELECT   sd.state, bn.year, bn.name,
         total
FROM     bn
         LEFT JOIN state_details sd
         ON bn.state = sd.abbreviation
ORDER BY total DESC;
```

Tip #6: Use a CTE as a Subquery Alternative

Subquery

```
-- Popular names in each state in the 90s
SELECT   sd.state, bn.year, bn.name,
         total
FROM     (SELECT state, year, name,
                 SUM(num_babies) as total
          FROM baby_names
          WHERE year BETWEEN 1990 and 1999
          GROUP BY state, year, name) AS bn
          LEFT JOIN state_details sd
          ON bn.state = sd.abbreviation
ORDER BY total DESC;
```

Common Table Expression (CTE)

```
-- Popular names in each state in the 90s
WITH bn AS (SELECT state, year, name,
                   SUM(num_babies) as total
            FROM baby_names
            WHERE year BETWEEN 1990 and 1999
            GROUP BY state, year, name)

SELECT   sd.state, bn.year, bn.name,
         total
FROM     bn
         LEFT JOIN state_details sd
         ON bn.state = sd.abbreviation
ORDER BY total DESC;
```

- ✓ Cleaner look
- ✓ Can be referenced multiple times

III. Break a Big Problem Into Smaller Pieces

- Tip #5: First Reduce, Then Combine
  - ✓ Execution order of a query
  - ✓ Use subqueries
- Tip #6: Use a CTE as a Subquery Alternative
  - ✓ Cleaner code
  - ✓ Can be referenced multiple times
- Side Note: temp tables and views

IV. When In Doubt, Test It Out

Tip #7: Each query engine works differently

RDBMS: Relational database management system


Microsoft
SQL Server


MySQL

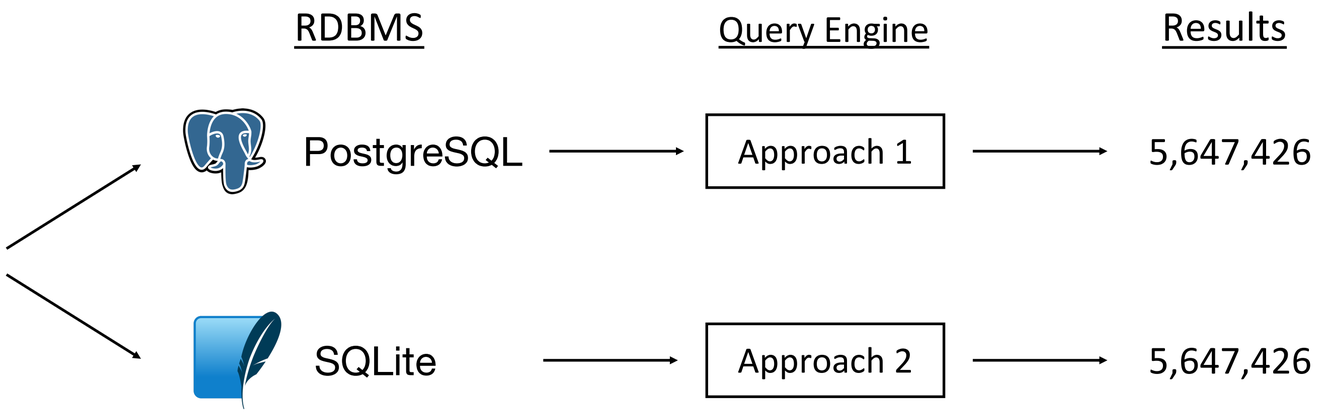

Oracle


PostgreSQL


SQLite

Query engine: interprets SQL code and actually makes it happen

```sql
SELECT COUNT(*)
FROM baby_names bn
     INNER JOIN state_details sd
     ON sd.abbreviation = bn.state;
```



Tip #7: Each query engine works differently

```
/* Number of baby_names rows with a state
   that exists in the state table */
SELECT COUNT(*)
FROM baby_names bn
WHERE EXISTS (SELECT *
              FROM state_details sd
              WHERE sd.abbreviation = bn.state);
```

Makes sense, but EXISTS is known to be slow.

```
/* Number of baby_names rows with a state
   that exists in the state table */
SELECT COUNT(*)
FROM baby_names bn
     INNER JOIN state_details sd
     ON sd.abbreviation = bn.state;
```

Less clear, but runs faster (sometimes).

Demo

Timing Queries + Viewing Execution Plans

SQLite

PostgreSQL

Tip #7: Each query engine works differently

```sql
/* Number of baby_names rows with a state
   that exists in the state table */
SELECT COUNT(*)
FROM baby_names bn
WHERE EXISTS (SELECT *
              FROM state_details sd
              WHERE sd.abbreviation = bn.state);
```

Makes sense, but EXISTS is known to be slow.

PostgreSQL

```sql
/* Number of baby_names rows with a state
   that exists in the state table */
SELECT COUNT(*)
FROM baby_names bn
     INNER JOIN state_details sd
     ON sd.abbreviation = bn.state;
```

Less clear, but runs faster (sometimes).

SQLite

Tip #8: Each RDBMS has different features

ANSI SQL: Standard SQL that will run in any RDBMS

```
SELECT * FROM baby_names;
```

Each RDBMS has additional features and keywords beyond the standard

```
-- Number of new names created in the 2000s
SELECT COUNT(DISTINCT name)
FROM baby_names
WHERE year >= 2000 AND
      name NOT IN (SELECT name
                   FROM baby_names
                   WHERE year BETWEEN 1990 AND 1999);
```

Will work in any RDBMS.

```
-- Number of new names created in the 2000s
SELECT COUNT(DISTINCT name) FROM
(SELECT name FROM baby_names WHERE year >= 2000
EXCEPT
SELECT name FROM baby_names WHERE year BETWEEN 1990 AND 1999);
```

Works in PostgreSQL, SQL Server and SQLite.

Pro Tip

When googling SQL syntax,
include the RDBMS in your search

sql return 10 rows mysql

```
SELECT *
FROM baby_names
LIMIT 10;
```

sql return 10 rows sql server

```
SELECT TOP 10 *
FROM baby_names;
```

Tip #9: Troubleshoot instead of memorize

We've covered a lot.

- UNION ALL is faster than UNION
- Avoid unnecessary DISTINCTs, wildcards, etc.
- Use subqueries, except in certain situations…

It's impossible to
remember every tip and trick!
(just like cooking)

Tip #9: Troubleshoot instead of memorize

We've covered a lot.

- UNION ALL is faster than UNION
- Avoid unnecessary DISTINCTs, wildcards, etc.
- Use subqueries, except in certain situations…

It's impossible to
remember every tip and trick!
(just like cooking)

Instead,

- Troubleshoot — (1) Reproduce the problem. (2) Check each component.
- Over time, you will build experience.

IV. When In Doubt, Test It Out

- Tip #7: Each query engine works differently

  ✓ Best to time queries and view the execution plans

- Tip #8: Each RDBMS has different features

  ✓ Best to include RDBMS name in Google searches

- Tip #9: Troubleshoot instead of memorize

  1. Reproduce the problem
  2. Check each component’s run time and execution plans

I. Write Code for Humans

II. Only Do What's Necessary

III. Break a Big Problem Into Smaller Pieces

IV. When In Doubt, Test It Out

I. Write Code for Humans	II. Only Do What's Necessary	III. Break a Big Problem Into Smaller Pieces	IV. When In Doubt, Test It Out
1. Format your code	3. Take only what you need	5. First reduce, then combine	7. Each query engine works differently
2. Add comments	4. Avoid unnecessary calculations	6. Use a CTE as a subquery alternative	8. Each RDMBS has different features
			9. Troubleshoot instead of memorize

10. Use indexes

Q&A

Tips for Writing Better SQL

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

5 Minute Break

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Index Example

Column is indexed

baby_names

Name	Year	Gender	State	Num_Babies
Mary	1910	F	AK	14
Annie	1910	F	AK	12
Anna	1975	F	AK	10
Margaret	2000	F	AK	8
Helen	2000	F	AK	7

5 million rows

50 ms

```
SELECT *
FROM baby_names
WHERE year = 2000;
```

Every row is scanned and checked.

20 ms

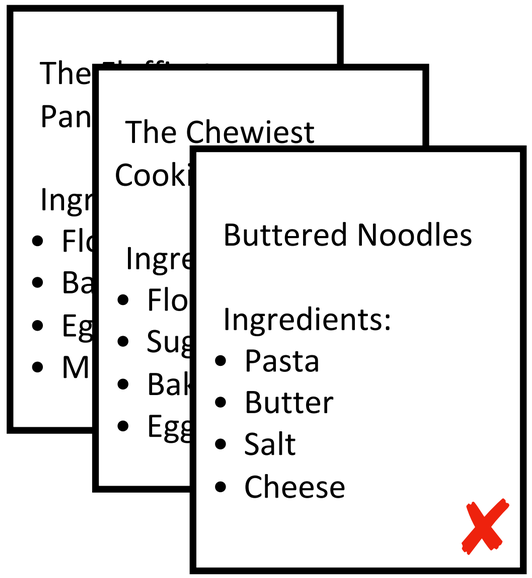
```
SELECT *
FROM baby_names
WHERE year = 2000;
```

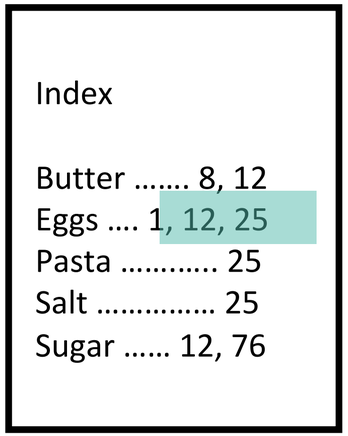
An index is used to quickly look up values.

How does an index work?

Imagine you want to find all pages that contain eggs.

Imagine you want to find all **rows** that contain 2000.





A book has many pages.

Each page has different ingredients.

1. Scan through each page.

2. Check the index to quickly find the pages.

A table has many rows.

Each row has different years.

1. Scan through each row.

2. Check the index to quickly find the rows.

How does an index work?

Imagine you want to find all **rows** that contain 2000.

A table has many rows.

Each row has different years.

1. Scan through each row.

2. Check the index to quickly find the rows.

How does an index work?

Imagine you want to find all **rows** that contain 2000.

A table has many rows.

Each row has different years.

1. Scan through each row.

2. Check the index to quickly find the rows.

baby_names

Name	Year	Gender	State	Num_Babies
Mary	1910	F	AK	14
Annie	1910	F	AK	12
Anna	1975	F	AK	10
Margaret	2000	F	AK	8
Helen	2000	F	AK	7

← 2000?

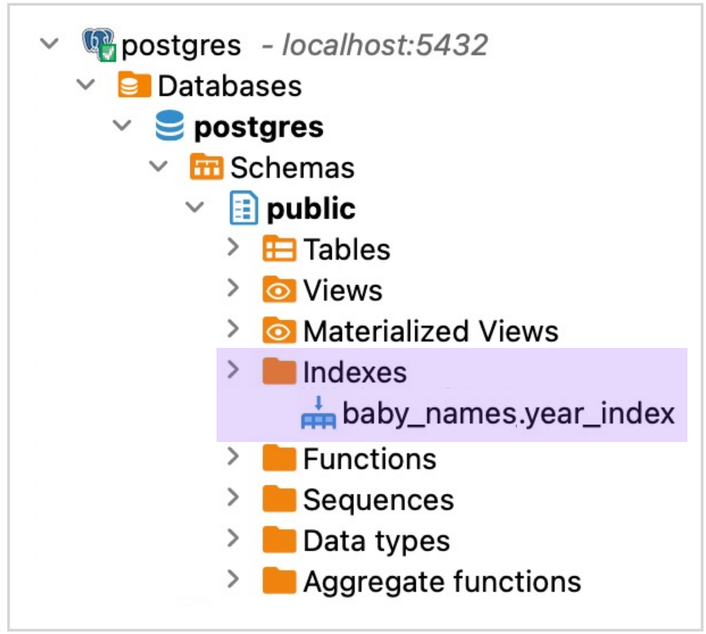
```
SELECT *
FROM baby_names
WHERE year = 2000;
```

Year	Rows
1910	1, 2
1975	3
2000	4, 5

How does an index work in practice?

An index is created one time (it may take a while):

```
CREATE INDEX year_index ON baby_names (year);
```



You can check to see which column(s) are indexed:

```
-- View indexes in SQLite
SELECT sql
FROM   SQLite_master
WHERE  type = 'index'
       AND tbl_name = 'baby_names';
```

```
"CREATE INDEX year_index ON baby_names(year)"
```

```
-- View indexes in PostgreSQL
SELECT indexname, indexdef
FROM   pg_indexes
WHERE  tablename = 'baby_names';
```

```
 indexname  |                                      indexdef
------------+-----------------------------------------------------------------------
 year_index | CREATE INDEX year_index ON public.baby_names USING btree (year)
```

Tip #10: Use indexes

1. You decide to work with a particular table
2. You start writing queries and…
    - Your queries are slow
    - You are doing a lot of filtering, joining, etc.
3. Check to see the available indexes
4. Incorporate the indexes into your queries

Tip #10: Use indexes

Filter on indexed columns

50 ms

```
SELECT *
FROM baby_names
WHERE year = 2000;
```

20 ms

```
SELECT *
FROM baby_names
WHERE year = 2000;
```

Join on indexed columns

4 sec

```
SELECT *
FROM   baby_names b1
       INNER JOIN baby_names b2
       ON b1.year = b2.year;
```

2 ms

```
SELECT *
FROM   baby_names b1
       INNER JOIN baby_names b2
       ON b1.year = b2.year;
```

Tip #10: Use indexes

When filtering on an indexed column, use IN instead of AND / OR

```
SELECT *
FROM   baby_names
WHERE  year = 2000
       OR year = 2001
       OR year = 2002;
```

290 ms

- Bitmap Heap Scan
  - BitmapOr
    - Bitmap Index Scan
    - Bitmap Index Scan
    - Bitmap Index Scan

```
SELECT *
FROM   baby_names
WHERE  year IN (2000, 2001, 2002);
```

120 ms

- Bitmap Heap Scan
  - Bitmap Index Scan

✓ The fewer scans, the better

Tip #10: Use indexes

Creating indexes

- ✓ You must have the appropriate permissions to create indexes
- ✓ You may need to talk to your data warehousing team

Querying with indexes

- ✓ Can speed up filters and joins
- ✓ Check to see which columns are indexed
- ✓ Check the execution plan to determine if the index is being used correctly / optimally

Writing Better SQL Tips

I. Write Code for Humans

II. Only Do What's Necessary

III. Break a Big Problem Into Smaller Pieces

IV. When In Doubt, Test It Out

Use Indexes

Q&A

Indexes

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Q&A (5 minutes)

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Q&A (5 minutes)

Katacoda Scenario

Improve a SQL Query

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Agenda

- Introduction (15 minutes)
- Tips for Writing Better SQL (40 minutes)
- Break (5 minutes)
- Indexes (10 minutes)
- Interactive Exercise (15 minutes)
- Wrap Up (5 minutes)

Writing Better SQL Summary

Steps

1. Write code that works
2. Make it better

In Practice

1. Time your queries
2. View the execution plans (EXPLAIN, EXPLAIN QUERY PLAN, etc.)

Writing Better SQL Tips

I. Write Code for Humans	II. Only Do What's Necessary	III. Break a Big Problem Into Smaller Pieces	IV. When In Doubt, Test It Out
1. Format your code	3. Take only what you need	5. First reduce, then combine	7. Each query engine works differently
2. Add comments	4. Avoid unnecessary calculations	6. Use a CTE as a subquery alternative	8. Each RDMBS has different features
			9. Troubleshoot instead of memorize

10. Use indexes

Group Discussion

- What tips from today will you keep in mind going forward?
- What are some other tips you have that weren’t covered today?

Writing Better SQL In 90 Minutes

Alice Zhao

A Dash of Data