What you write is not what SQL sees

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

SQL

Amy McCarty
Instructor

datacamp

Algebraic order of operations

- Lexical (as written)

- Logical (as executed)

PEMDAS	BODMAS
Parenthesis	**B**rackets
Exponents	**O**rder
Multiplication /**D**ivision	**D**ivision /**M**ultiplication
Addition /**S**ubtraction	**A**ddition /**S**ubtraction

Applying the order of operations

Lexical:

$$x = 2 + (8 + 4) \div 2$$

$$x = 10 + 4) \div 2$$

$$x = 14 \div 2$$

$$x = 7$$

Logical:

$$x = 2 + (8 + 4) \div 2$$

$$x = 2 + \frac{12}{2}$$

$$x = 2 + 6$$

$$x = 8$$

SQL logical order of operations

Order	Clause	Purpose
1	**FROM**	Provides directions to the table or tables if the query includes joins
2	**WHERE**	Filters or limits the records
3	**GROUP BY**	Places records into categories
4	**SUM(), COUNT(), etc**	Aggregates
5	**SELECT**	identifies columns to return

```
SELECT COUNT(*) FROM tableA WHERE col1 = 77
```

Group by and aggregations

event_location	storm	elements	days
Russia	blizzard	water	1
Argentina	tornado	water	1
Argentina	tornado	wind	1
Australia	tornado	wind	1
Kuwait	haboob	wind	2
USA	haboob	wind	2

```
SELECT elements, storm, COUNT(*)
FROM weather_events
GROUP BY elements
```

```
No output -
your code generated an error

column "storm" must appear in the
GROUP BY clause or be used in an
aggregate function
```

Group by and aggregations order of operations

```sql
SELECT elements, storm, COUNT(*)
FROM weather_events
GROUP BY elements
```

order	SQL clause	available columns
1	FROM weather_events	all
3	GROUP BY elements	elements
4	COUNT	elements

Group by matches the aggregations

```sql
SELECT elements, storm, COUNT(*)
FROM weather_events
GROUP BY elements, storm
```

Group by matches the aggregations

```sql
SELECT elements, storm, COUNT(*)
FROM weather_events
GROUP BY elements, storm
```

Results

elements	storm	count
water	blizzard	1
water	tornado	1
wind	tornado	2
wind	haboob	2

SQL logical order of operations continued

Order	Clause	Purpose
...		
5	**SELECT**	identifies columns to return
6	**DISTINCT**	removes duplicates
7	**ORDER BY**	arranges results
8	**LIMIT**	removes rows

Distinct and limit

row_no	location	storm	elements	day
1	Russia	blizzard	water	1
2	Argentina	tornado	water	1
3	Argentina	tornado	wind	1
4	Australia	tornado	wind	1
5	Kuwait	haboob	wind	2
6	USA	haboob	wind	2

```
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

row_no	location	storm	elements	day
1	Russia	blizzard	water	1
2	Argentina	tornado	water	1
3	Argentina	tornado	wind	1
4	Australia	tornado	wind	1
5	Kuwait	haboob	wind	2
6	USA	haboob	wind	2

```
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

row_no	location	storm	elements	day
1	Russia	blizzard	water	1
2	Argentina	tornado	water	1
3	Argentina	tornado	wind	1
4	Australia	tornado	wind	1
5	Kuwait	haboob	wind	2
6	USA	haboob	wind	2

order	SQL clause	available rows
1	FROM weather_events	all

```
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

row_no	storm	elements
1	blizzard	water
2	tornado	water
3	tornado	wind
5	haboob	wind

order	SQL clause	available rows
1	FROM weather_events	all
5	SELECT storm, elements	all

```
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

row_no	storm	elements
1	blizzard	water
2	tornado	water
3	tornado	wind
5	haboob	wind

order	SQL clause	available rows
1	FROM weather_events	all
5	SELECT storm, elements	all
6	DISTINCT	1, 2, 3, 5

```sql
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

row_no	storm	elements
1	blizzard	water
5	haboob	wind
2	tornado	water
3	tornado	wind

order	SQL clause	available rows
1	FROM weather_events	all
5	SELECT storm, elements	all
6	DISTINCT	1, 2, 3, 5
7	ORDER BY storm	1, 5, 2, 3

```
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

row_no	storm	elements
1	blizzard	water
5	haboob	wind
2	tornado	water

order	SQL clause	available rows
1	FROM weather_events	all
5	SELECT storm, elements	all
6	DISTINCT	1, 2, 3, 5
7	ORDER BY storm	1, 5, 2, 3
8	LIMIT 3	1, 5, 2

```
SELECT DISTINCT storm, elements
FROM weather_events
ORDER BY storm LIMIT 3
```

Order	Clause	Purpose	Limits
1	**FROM**	provides directions to the table(s)	
2	**WHERE**	filters or limits the records	# rows
3	**GROUP BY**	places records into categories	# columns
4	**SUM**, **COUNT**, etc	aggregates	# rows
5	**SELECT**	identifies columns to return	# columns
6	**DISTINCT**	removes duplicates	# rows
7	**ORDER BY**	arranges results	
8	**LIMIT**	filters records	# rows

Let's practice!

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

Filtering in the WHERE clause

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

SQL

Amy McCarty
Instructor

datacamp

Limit the data

Order	Clause	Purpose
1	FROM	provides directions to the table(s)
2	**WHERE**	**filters or limits the records**

- Occurs early

- Fewer records

EXPLAIN

```
EXPLAIN

SELECT * FROM phones
```

Query planner


```
Seq Scan on phones (cost = 0.00..22.7
                    ,rows=1270
                    ,width=36)
```

- Number of execution steps

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

EXPLAIN with WHERE

```
EXPLAIN
SELECT * FROM phones
WHERE phone_code = 235
```

Query planner


```
Seq Scan on phones (cost = 0.00..25.8
                    ,rows=6,width=636)
   Filter: (phone_code=235)
```

Good - Filtering for similar values with LIKE OR

country	phone_code	reliability
Chad	235	medium
China	86	high
Costa Rica	506	high
India	91	medium
Indonesia	62	medium
Iraq	964	low

```
EXPLAIN
SELECT * FROM phones
WHERE country LIKE 'Ch%'
    OR country LIKE 'In%'
```

```
Seq Scan on phones (cost = 0.00..29.05
                    ,rows=13,width=36)
    Filter: ((country~~'Ch%'::text)
            OR(country~~'In%'::text))
```

Better - Filtering for similar values with LIKE ANY

country	phone_code	reliability
Chad	235	medium
China	86	high
Costa Rica	506	high
India	91	medium
Indonesia	62	medium
Iraq	964	low

```
EXPLAIN
SELECT * FROM phones
WHERE country
    LIKE ANY(ARRAY['Ch%','In%'])
```

```
Seq Scan on phones (cost = 0.00..25.88
                    ,rows=13,width=36)
    Filter: ((country~~ANY('{Ch%,In%}'
                          ::text[]))
```

Good - Filtering for exact values with OR

country	phone_code	reliability
Chad	235	medium
China	86	high
Costa Rica	506	high
India	91	medium
Indonesia	62	medium
Iraq	964	low

```
EXPLAIN
SELECT * FROM phones
WHERE country = 'Chad'
    OR country = 'China'
```

```
Seq Scan on phones (cost = 0.00..29.05
                    ,rows=13,width=36)
    Filter: ((country='Chad'::text)
            OR(country='China'::text))
```

Better - Filtering for exact values with IN

country	phone_code	reliability
Chad	235	medium
China	86	high
Costa Rica	506	high
India	91	medium
Indonesia	62	medium
Iraq	964	low

```
EXPLAIN
SELECT * FROM phones
WHERE country IN ('Chad','China')
```

```
Seq Scan on phones (cost = 0.00..25.88
                  ,rows=13,width=36)
  Filter: ((country=ANY('{Chad,China}'
                          ::text[]))
```

Best - Filtering for numbers

country	phone_code	reliability
Chad	235	medium
China	86	high
Costa Rica	506	high
India	91	medium
Indonesia	62	medium
Iraq	964	low

```
EXPLAIN
SELECT *
FROM phones
WHERE phone_code IN (235,86)
```

```
Seq Scan on phones (cost = 0.00..25.88
                   ,rows=13,width=36)
    Filter: (phone_code=ANY('{235,86}'
                          ::integer[]))
```

Summarizing the best WHERE filters

Numeric advantages

- Shorter length

- Smaller storage

- Speeds performance

Summarizing the best WHERE filters

Numeric advantages

- Shorter length

- Smaller storage

- Speeds performance

Good	Better
Text	Numeric
OR	IN, ARRAY

Let's practice!

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

Filtering while joining

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

SQL

Amy McCarty
Instructor

Joins revisited

Joins

- Link tables using at least 1 common field

```
SELECT *
FROM TABLE_A AS A
INNER JOIN TABLE_B AS B
  ON A.NAME = B.NAME
```

Joins to combine data

- From multiple tables

- Inner and outer

Joins to filter data

- Inner join

- Outer join with non-linking join condition

Patient and appointments data

Appointments

visit_id	reason	patient_id
01	checkup	999
02	infection	888

Patients

patient_id	name	sex
999	Lotte Smith	F
888	Zhang Wei	M
777	Amelia Hernandez	F

Inner joins to filter

```
SELECT *
FROM appointments a
INNER JOIN patients p
ON a.patient_id = p.patient_id
```

Appointments

visit_id	reason	patient_id
01	checkup	999
02	infection	888

Patients

patient_id	name	sex
999	Lotte Smith	F
888	Zhang Wei	M
777	Amelia Hernandez	F

Inner joins to filter

```
SELECT *
FROM appointments a
INNER JOIN patients p
ON a.patient_id = p.patient_id
```

Output

visit_id	reason	patient_id	name	sex
01	checkup	999	Lotte Smith	F
02	infection	888	Zhang Wei	M

Outer joins to filter

```sql
SELECT *
FROM appointments a
LEFT JOIN patients p
  ON a.patient_id = p.patient_id
  AND p.sex = 'M'
```

Appointments

visit_id	reason	patient_id
01	checkup	999
02	infection	888

Patients

patient_id	name	sex
999	Lotte Smith	F
888	Zhang Wei	M
777	Amelia Hernandez	F

Outer joins to filter

```sql
SELECT *
FROM appointments a
LEFT JOIN patients p
  ON a.patient_id = p.patient_id
  AND p.sex = 'M'
```

Output

visit_id	reason	patient_id	name	sex
01	checkup	999		
02	infection	888	Zhang Wei	M

Filter pitfalls

```sql
SELECT *
FROM appointments a
LEFT JOIN patients p
  ON a.patient_id = p.patient_id
WHERE p.sex = 'M'
```
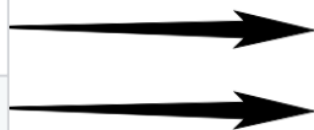
1) FROM

Appointments

visit_id	reason	patient_id
01	checkup	999
02	infection	888

Patients

patient_id	name	sex
999	Lotte Smith	F
888	Zhang Wei	M
~~777~~	~~Amelia Hernandez~~	~~F~~

2) WHERE

Output

visit_id	reason	patient_id	name	sex
~~01~~	~~checkup~~	~~999~~	~~Lotte Smith~~	~~F~~
02	infection	888	Zhang Wei	M

Filter pitfalls

```sql
SELECT *
FROM appointments a
LEFT JOIN patients p
  ON a.patient_id = p.patient_id
WHERE p.sex = 'M'
```

Output

visit_id	reason	patient_id	name	sex
02	infection	888	Zhang Wei	M

Filter pitfalls

```sql
SELECT *
FROM appointments a
LEFT JOIN patients p
    ON a.patient_id = p.patient_id
WHERE p.sex = 'M'
```

```sql
SELECT *
FROM appointments a
LEFT JOIN patients p
    ON a.patient_id = p.patient_id
    AND p.sex = 'M'
```

Output

visit_id	reason	patient_id	name	sex
02	infection	888	Zhang Wei	M

visit_id	reason	patient_id	name	sex
01	checkup	999		
02	infection	888	Zhang Wei	M

Filter pitfalls improved

```
SELECT *
FROM appointments a
INNER JOIN patients p
  ON a.patient_id = p.patient_id
WHERE p.sex = 'M'
```

```
SELECT *
FROM appointments a
INNER JOIN patients p
  ON a.patient_id = p.patient_id
  AND p.sex = 'M'
```

visit_id	reason	patient_id	name	sex
02	infection	888	Zhang Wei	M

Let's practice!

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

Aggregating with different data granularities

IMPROVING QUERY PERFORMANCE IN POSTGRESQL

SQL

Amy McCarty
Instructor

datacamp

Data granularity - level of detail

- Makes a row unique

- 1+ columns

Video_Games

id	game	first_yr
012	Grand Theft Auto	1997
234	Legend of_Zelda	1986

Game_Platforms

game_id	platform	year
234	FCDS	1986
234	GameCube	2003
234	Wii	2006

Joining different granularities

Video_Games

id	game	first_yr
012	Grand Theft Auto	1997
234	Legend of_Zelda	1986

Game_Platforms

game_id	platform	year
234	FCDS	1986
234	GameCube	2003
234	Wii	2006

id	game	first_yr	no_platforr
234	Legend_of_Zelda	1986	3
234	Legend_of_Zelda	1986	3
234	Legend_of_Zelda	1986	3
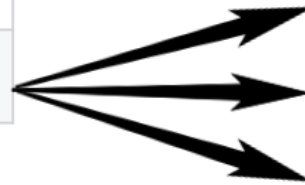
```
SELECT g.id, g.game, g.first_yr
    , COUNT(platform) AS no_platforms
FROM video_games g
INNER JOIN game_platforms p
    ON g.id = p.game_id
GROUP BY g.id, g.game, g.first_yr
```

Changing the table granularity

Game_Platforms

game_id	platform	year
234	FCDS	1986
234	GameCube	2003
234	Wii	2006

```sql
SELECT game_id
  , COUNT(platform) as no_platforms
FROM game_platforms
GROUP BY game_id
```

Changing the table granularity

Game_Platforms

game_id	platform	year
234	FCDS	1986
234	GameCube	2003
234	Wii	2006

```
SELECT game_id
  , COUNT(platform) as no_platforms
FROM game_platforms
GROUP BY game_id
```

Output

game_id	no_platforms
234	3

CTEs revisited

- Standalone query with temporary results set

- With statement structure

```
WITH cte AS
  ( SELECT * FROM tableA )


SELECT * FROM cte
```

- Allows aggregation before joining

CTEs to the granularity rescue

```
WITH platforms_cte AS
(   SELECT game_id
        , COUNT(platforms) as no_platforms
    FROM game_platforms
    GROUP BY game_id
)
```

```
SELECT g.id, g.game , cte.no_platforms
FROM video_games g
INNER JOIN platforms_cte cte
  ON g.id = cte.game_id
```

CTEs to the granularity rescue

```
WITH platforms_cte AS
(   SELECT game_id
        , COUNT(platforms) as no_platforms
    FROM game_platforms
    GROUP BY game_id
)
```

```
SELECT g.id, g.game , cte.no_platforms
FROM video_games g
INNER JOIN platforms_cte cte
   ON g.id = cte.game_id
```

Output

id	game	no_platforms
234	Legend of Zelda	3

Matching data granularity when joining

- No repeats or duplicates

- Minimum needed results

- No double counting

Let's practice!

IMPROVING QUERY PERFORMANCE IN POSTGRESQL