PostgreSQL Development: The Path of a Query

05.05.2021
Peter Moser

TECHPARK SÜDTIROL / ALTO ADIGE

Nature of Innovation.

Peter Moser

~ Husband, and father of 3 kids :-)

Education:

- Master Degree in Computer Science

Work:

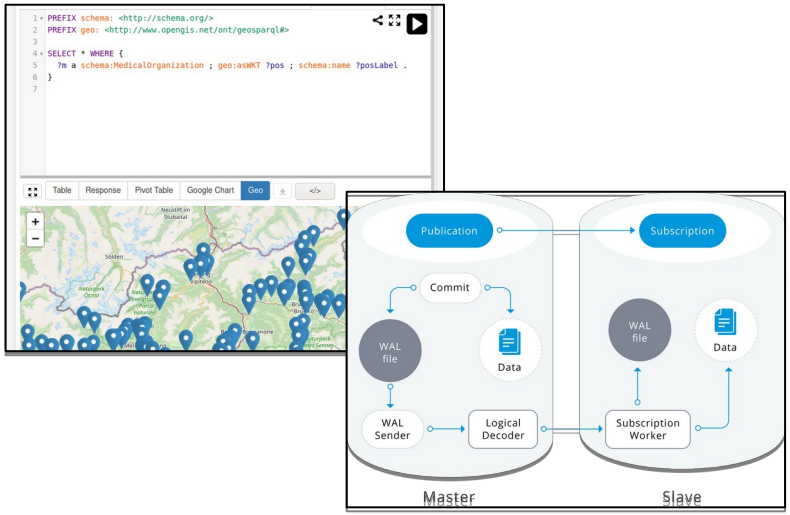
- 2019 → NOI (Databases, DevOp & Software Architect)
- 2018 → IDM (Databases & Software Architect)
- Until 2017 → Free University of Bolzano (Research Assistant)

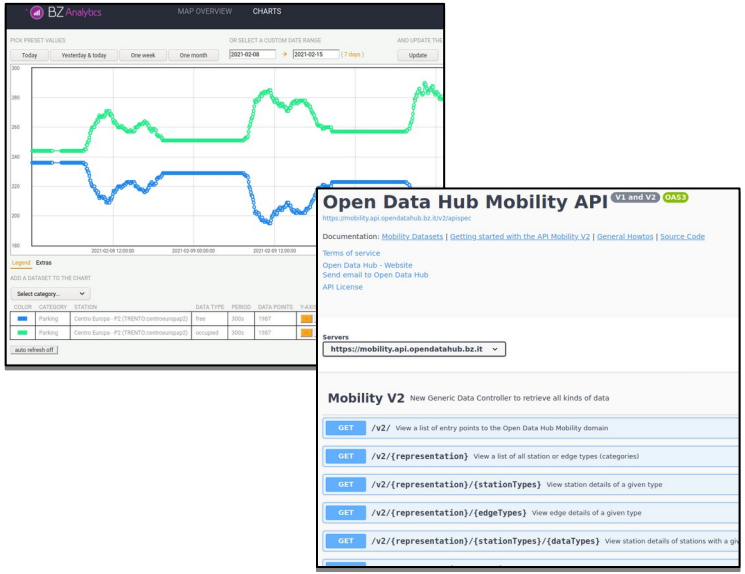
Work-related Interests:

- Community Building and Team-Work
- Databases
- Database Algorithms and Core Development
- Backend
- Development methodologies
- DevOp and Automation
- Licensing
- Contributor to Open Source Projects (Linux Mint)

Tech Transfer Digital Team @ NOI Techpark

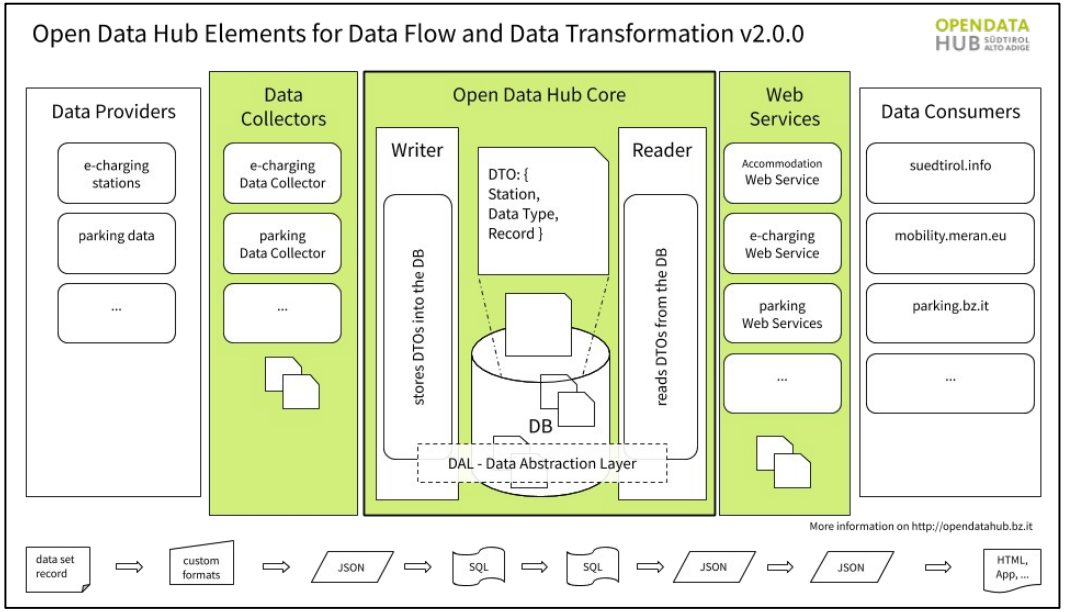
- Helping companies, freelancer and enthusiasts in their projects
- Research & Development projects
- Creating and maintaining the Open Data Hub

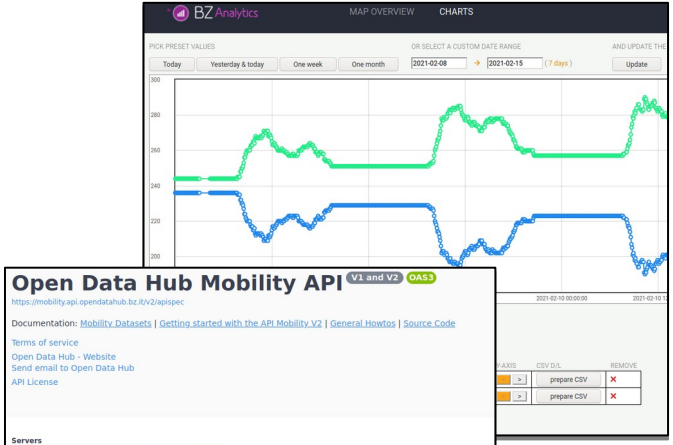




Projects: Open Data Hub

- Access to traffic, mobility, environment and tourism data
- Data integration and data access
- Visualization, demo apps and community building





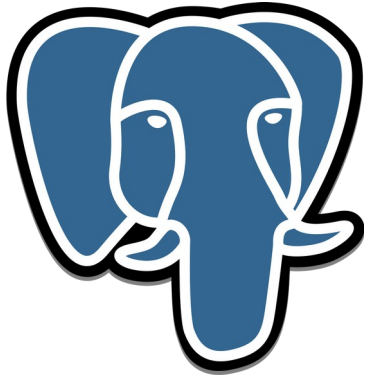
A glance into the PostgreSQL kernel

1) The path of a query

2) Patch Development

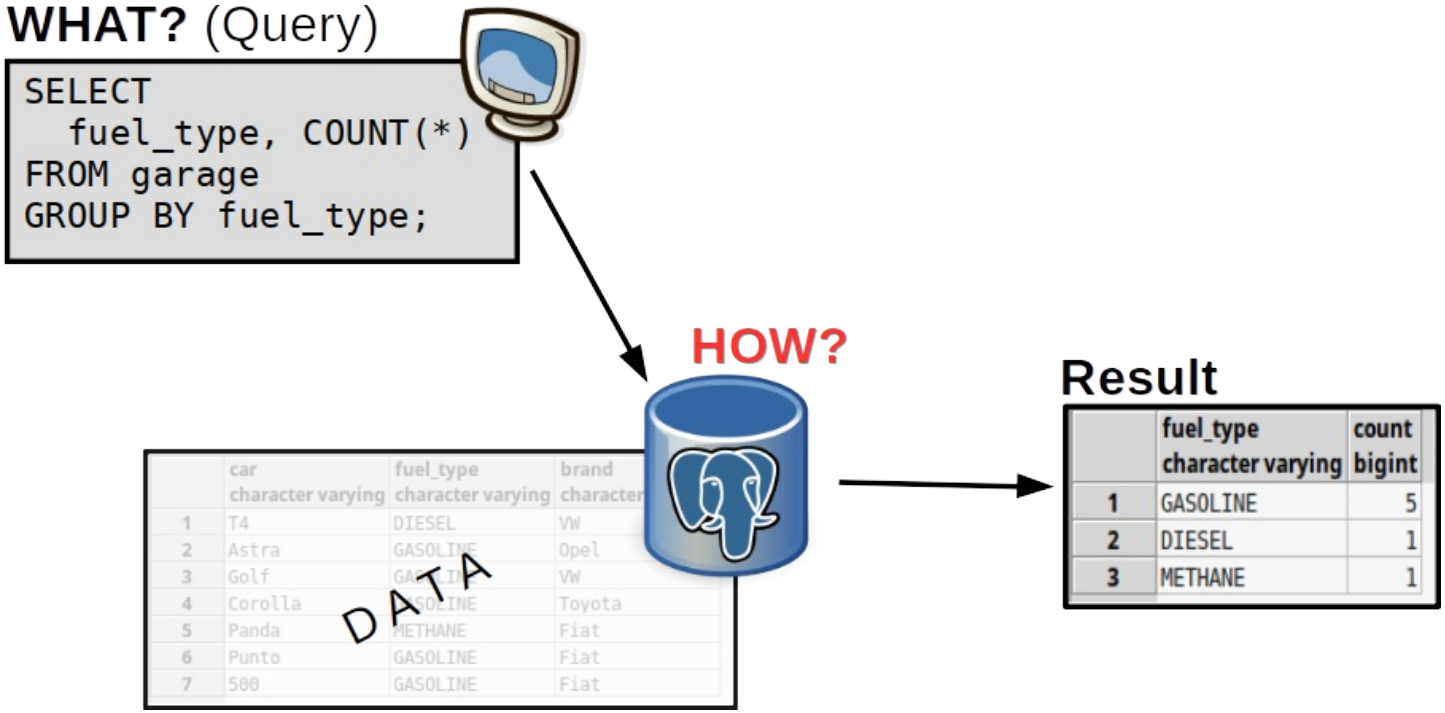
```
9943 select_no_parens:
9944             simple_select                          { $$ = $1; }
9945           | select_clause sort_clause
9946               {
9947                   insertSelectOptions((SelectStmt *) $1, $2
9948                                       NULL, NULL, NULL,
9949                                       yyscanner);
9950                   $$ = $1;
9951               }
9952           | select_clause opt_sort_clause for_locking_claus
9953               {
9954                   insertSelectOptions((SelectStmt *) $1, $2
9955                                       list_nth($4, 0), list
9956                                       NULL,
9957                                       yyscanner);
9958                   $$ = $1;
9959               }
9960           | select_clause opt_sort_clause select_limit opt_
9961               {
9962                   insertSelectOptions((SelectStmt *) $1, $2
9963                                       list_nth($3, 0), list
```

3) Community and beyond

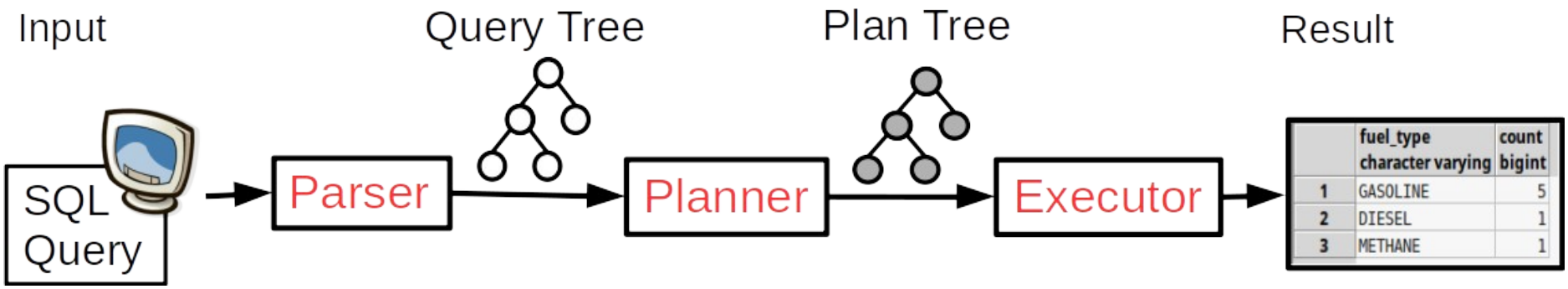


How does Postgres process a query?

How many cars with a specific fuel type are in our garage?



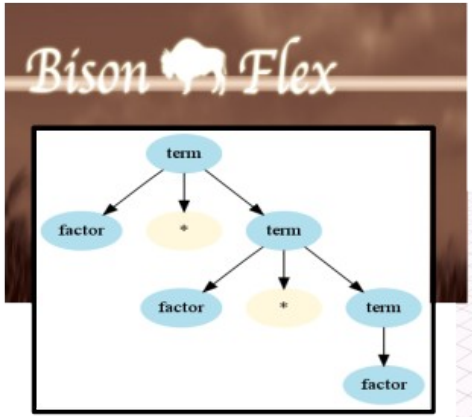
The path of a query from bird's-eye view



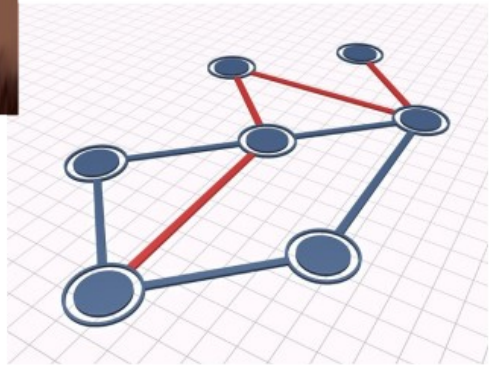
	fuel_type character varying	count bigint
1	GASOLINE	5
2	DIESEL	1
3	METHANE	1

Three main stages of query processing

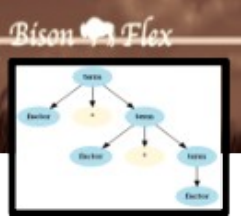
Parser stage



Planner stage



Executor stage



Parser stage

- Checks the SQL syntax
  - Syntax error:

```
SELECT * FRAM garage;
```

"FRAM" is not a SQL key word

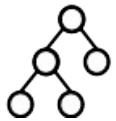
- Makes lookups in a catalog
  - Error in semantics:

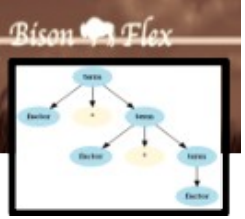
```
SELECT * FROM garauge;
```

Table "garauge" does not exist

  - Adds information (eg. column names, aliases)

- Builds the Query Tree
  - Internal representation of a Query





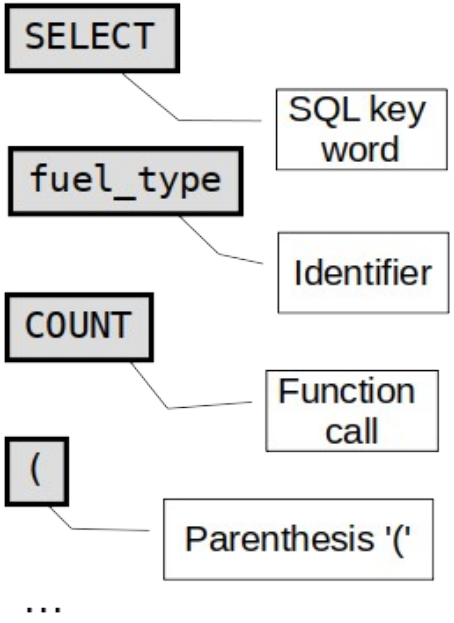
Parser stage: Identify tokens

Query String
arrives as plain text

```
SELECT
  fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type;
```

(C) wiseGeek

The parser
generates tokens



Parser stage: Build query tree

Grammar rules

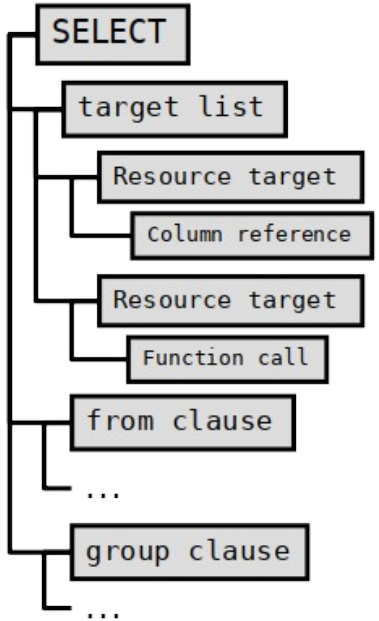
```
simple_select:
      SELECT target_list
      FROM from_clause
      GROUP BY group_clause
```

```
target_list:
      target_el
      | target_list ',' target_el
```

```
target_el:
      a_expr AS ColLabel
      | a_expr
      | '*'
```

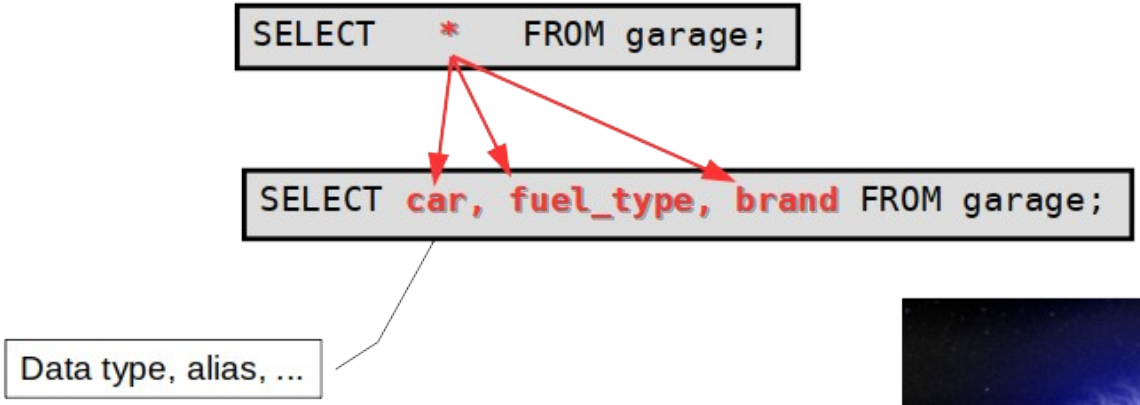
```
a_expr:
      ...
```

Query tree



Query Tree Annotation

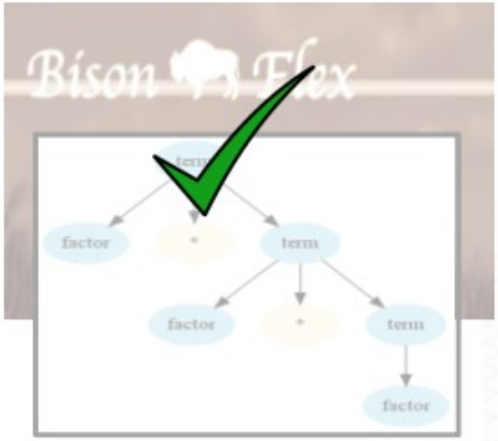
- Example: A_STAR expansion



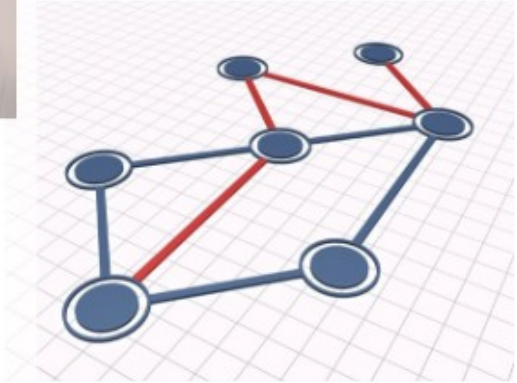
(C) wallpaperhi

Three main stages of query processing

Parser stage



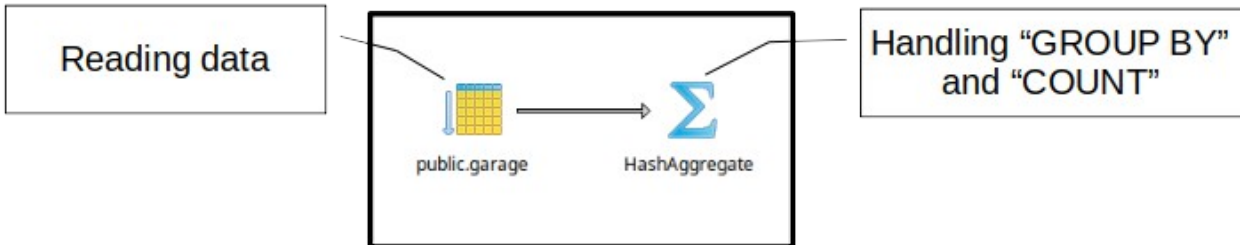
Planner stage



Executor stage

Planner stage

- There is a standard way to execute SQL
- The planner tries to find the best plan tree
- Creates the Plan Tree



Planner decisions

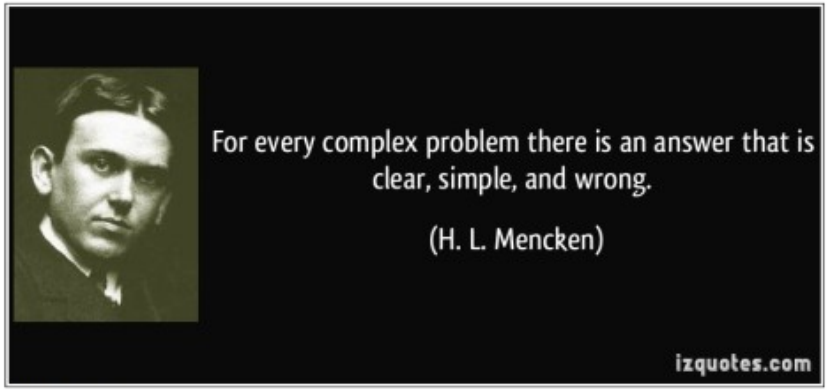
- Access Scan Types
  - Sequential Scan, (Bitmap) Index Scan, Index-Only Scan

- Join
  - Join order
  - Join strategy: nested loop, merge join, hash join
  - Inner vs. Outer join
- Aggregation
  - Plain, sorted, hashed

MIN, MAX, COUNT, AVG, SUM, ...

Planner stage

- Query planning is a complex problem
- Finding all plans could be too expensive
- If it is not feasible, it uses heuristics



Wikimedia Commons

Planner example

```
SELECT fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type;
```

Plan #1

50

Cost estimates

- Scan: 16
- Sort: 48
- Group: **50**

Plan #2

20

Cost estimates

- Scan: 16
- HashAgg: **20**

Smaller cost means faster!

is Plan #2

```
SELECT fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type;
```

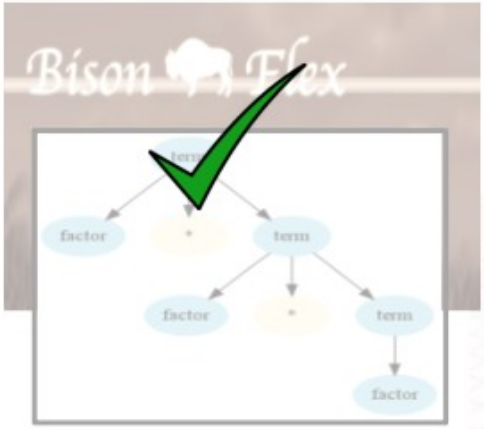
FINAL CHOICE:
Sequential Scan on “public.garage” and aggregation with a hash

Explaining the Query Plan Tree in pgAdmin

	QUERY PLAN text
1	HashAggregate (cost=19.75..21.75 rows=200 width=32) (actual time=0.024..0.025 rows=3 loops=1)
2	Group Key: fuel_type
3	-> Seq Scan on garage (cost=0.00..16.50 rows=650 width=32) (actual time=0.010..0.012 rows=7 loops=1)
4	Planning time: 0.071 ms
5	Execution time: 0.075 ms

Three main stages of query processing

Parser stage

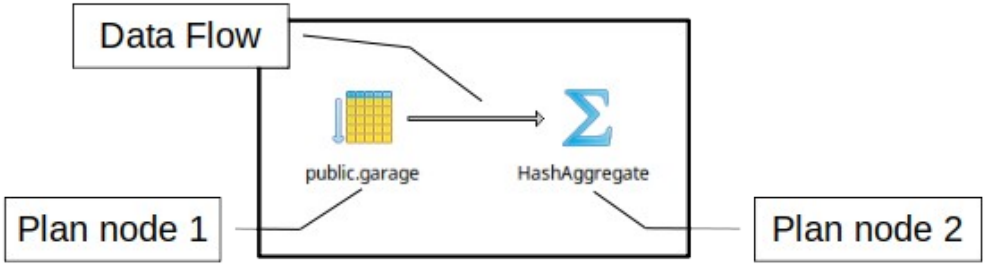


Planner stage

Executor stage

Executor stage

- Processes the (fastest) Plan Tree



- Each plan node returns a single row on demand

- Creates the Result

	fuel_type character varying	count bigint
1	GASOLINE	5
2	DIESEL	1
3	METHANE	1

Executor implementation

- Each executor node has a strict interface

`ExecInitNode()` `ExecProcNode()` ... `ExecEndNode()`

- Interface routines:

```
ExecInitNode():
    initialize a plan node and its
        subplans

ExecProcNode():
    get a tuple by executing
        the plan node
    return the tuple, or NULL

ExecEndNode():
    shut down a plan node and its
        subplans
```

Calling "ExecProcNode" twice

	car character varying	fuel_type character varying	brand character varying
1	T4	DIESEL	VW
2	Astra	GASOLINE	Opel
3	Golf	GASOLINE	VW
4	Corolla	GASOLINE	Toyota
5	Panda	METHANE	Fiat
6	Punto	GASOLINE	Fiat
7	500	GASOLINE	Fiat

Executor stage by example

- **Our query** (+ ordering):

```
SELECT fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type
ORDER BY count DESC;
```

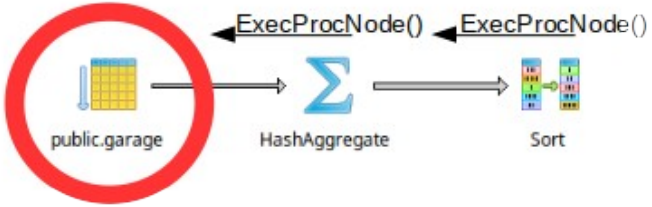
- Execution Plan:

Data:

	car character varying	fuel_type character varying	brand character varying
1	T4	DIESEL	VW
2	Astra	GASOLINE	Opel
3	Golf	GASOLINE	VW
4	Corolla	GASOLINE	Toyota
5	Panda	METHANE	Fiat
6	Punto	GASOLINE	Fiat
7	500	GASOLINE	Fiat

- The executor recursively calls itself to process subplans (starting from the top node Sort)

Executor stage




```
SELECT fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type
ORDER BY count DESC;
```

- Run the Sequential Scan node
- Return the result to the node HashAggregate
- Input:

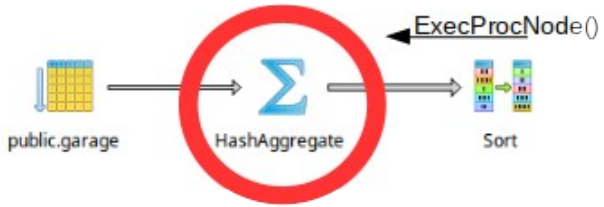
(c) Simutek

→

Result:

	car character varying	fuel_type character varying	brand character varying
1	T4	DIESEL	VW
2	Astra	GASOLINE	Opel
3	Golf	GASOLINE	VW
4	Corolla	GASOLINE	Toyota
5	Panda	METHANE	Fiat
6	Punto	GASOLINE	Fiat
7	500	GASOLINE	Fiat

Executor stage




```
SELECT fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type
ORDER BY count DESC;
```

- Build a hash upon “fuel_type”
- Use the hash to find candidates for fuel type groups
- Grouping by hash

Group #1
Group #2
Group #3

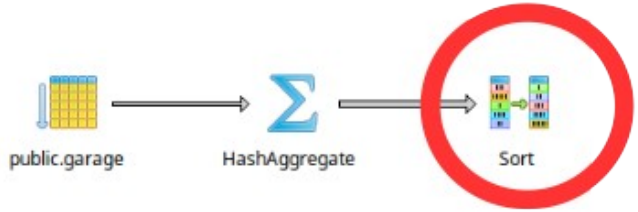
	car character varying	fuel_type character varying	brand character varying	hash text
1	T4	DIESEL	VW	167
2	Panda	METHANE	Fiat	b01
3	Golf	GASOLINE	VW	e6c
4	Punto	GASOLINE	Fiat	e6c
5	500	GASOLINE	Fiat	e6c
6	Corolla	GASOLINE	Toyota	e6c
7	Astra	GASOLINE	Opel	e6c

→

Result

	fuel_type character varying	count bigint
1	DIESEL	1
2	GASOLINE	5
3	METHANE	1

Executor stage




```
SELECT fuel_type, COUNT(*)
FROM garage
GROUP BY fuel_type
ORDER BY count DESC;
```

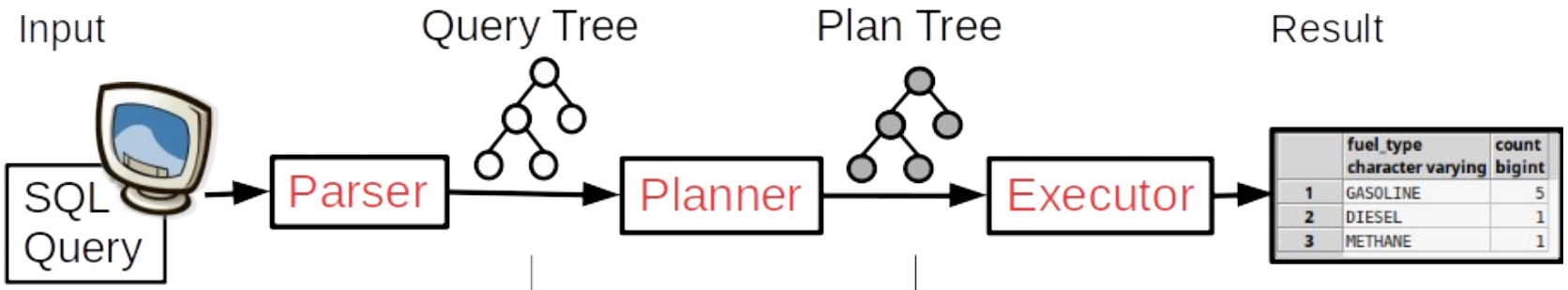
- Sort the rows from the subplan HashAggregate with sort key “count”
- Return the sorted groups as result

	fuel_type character varying	count bigint
1	DIESEL	1
2	GASOLINE	5
3	METHANE	1

→

	fuel_type character varying	count bigint
1	GASOLINE	5
2	DIESEL	1
3	METHANE	1

In summary, the path of a query...



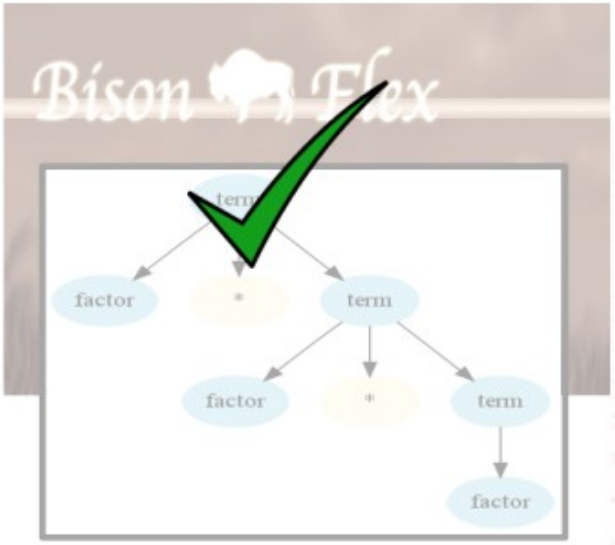
	fuel_type character varying	count bigint
1	GASOLINE	5
2	DIESEL	1
3	METHANE	1

- checks SQL syntax
- Builds the Query Tree

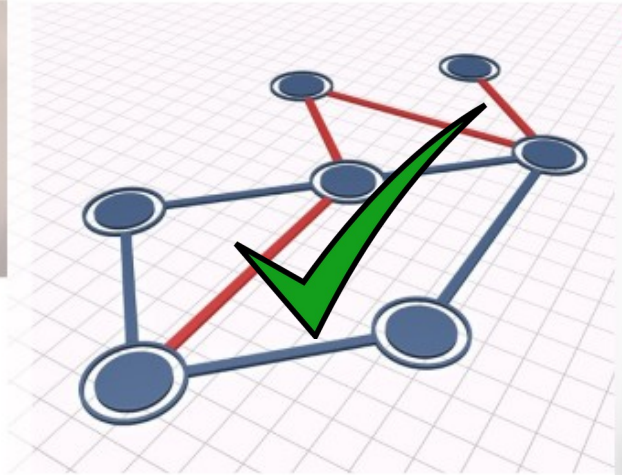
- Finds a good Execution Plan
- Builds the Plan Tree

- Executes the Plan
- Builds the Result

Parser stage



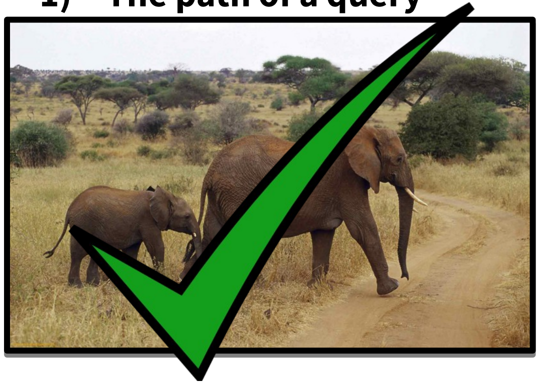
Planner stage



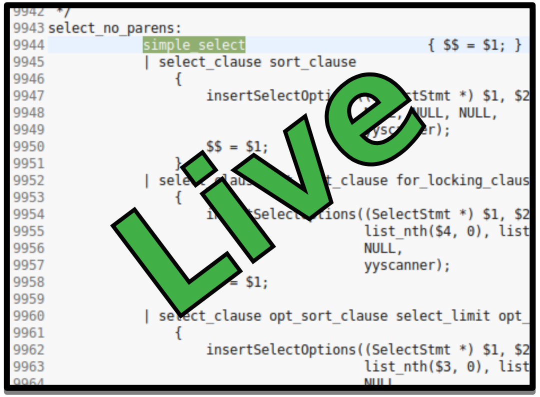
Executor stage

A glance into the PostgreSQL kernel

1) The path of a query

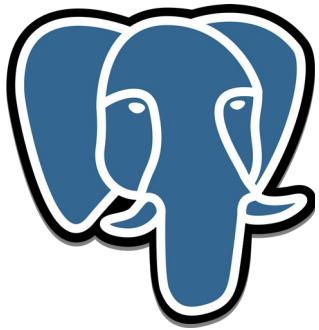


2) Patch Development



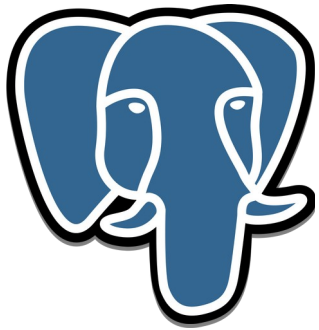
3) Community and beyond

https://github.com/Piiit/postgres-twice-patch/tree/new



- Hacker’s Mailinglist Culture https://lists.postgresql.org
  - Various mailing lists about PG exist, choose the right one
  - learning community communicating with the intention of learning, sharing and refining ideas
  - Wide range of expertise: databases, SW development and sysadmins, ...
  - Start reading it, to get a glance into ongoing discussions

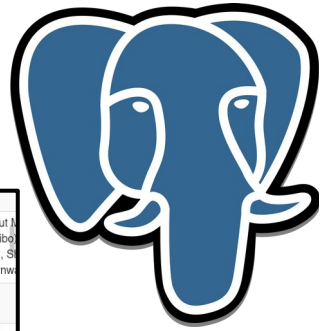
https://www.postgresql.org/list/



- Hacker Documentation
  - Getting started? See So, you want to be a developer?
  - General information:
    - https://wiki.postgresql.org/wiki/Development_information
    - https://wiki.postgresql.org/wiki/Developer_FAQ
    - https://wiki.postgresql.org/wiki/Todo

https://commitfest.postgresql.org/24/

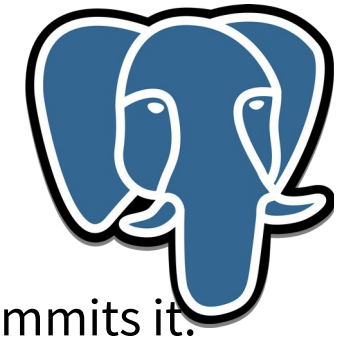
Community and beyond



- Commitfests
  - Patch review & committing
  - Other development stops
  - Everyone should become a reviewer for some time
  - ~ every two months

SimpleLruTruncate() rounding (data loss)				
Problem with default partition pruning	Committed	stable	Yuzuko Hosoya (yuzu)	Thibaut M (madtibo), (ibrar), S (shawnw
propagating replica identity to partitions	Returned with feedback	13	Álvaro Herrera (alvherre)	
Fix unique join costings	Moved to next CF	stable	David Rowley (davidrowley)	Tom Lan
fix for BUG #3720: wrong results at using ltree	Moved to next CF		Filip Rembiałkowski (filiprem)	Tom Lan (ibrar), B
Unaccent extension python script Issue in Windows	Committed		Hugh Ranalli (hughr), Ramanarayana M (ramnar)	
Problem during Windows service start	Withdrawn		Ramanarayana M (ramnar)	
Fix issues with "x SIMILAR TO y ESCAPE NULL"	Committed	13	Tom Lane (tgl)	
ALTER TABLE restructuring	Moved to next CF		Tom Lane (tgl)	Álvaro H
standby recovery fails when re-replaying due to missing directory which was removed in previous replay.	Moved to next CF	stable	Kyotaro Horiguchi (horiguti), Paul Guo (paulguo)	
Fix support for hypothetical indexes using BRIN access method	Moved to next CF		Julien Rouhaud (rjuju)	Heikki Li Michael P kun)
Duplicated LSN in ReorderBuffer	Committed		Ildar Musin (i.musin)	Masahiko (masahik
Windows could not stat file - over 4GB	Moved to next CF		Juanjo Santamaria Flecha (juanjo.santamaria.gmail.com)	Emil Iggl
CVE-2017-7484-induced bugs, or, btree cmp functions are not leakproof?	Moved to next CF		Dilip Kumar (dilip.kumar), Amit Langote (amitlan)	Dilip Kum Amit Lan

https://commitfest.postgresql.org/24/



- Development Cycles
  - Main thing: Discuss and post patch to pgsql-hackers
  - Workflow A: A committer picks it up immediately and commits it.
  - Workflow B: You add the patch to the open commitfest queue
    - A committer picks up the patch from the queue, and commits it
  - Rejection: for technical, style, or other reasons
  - Moved to next commitfest (=not ready yet, returned with feedback)
  - Withdrawn

https://wiki.postgresql.org/wiki/Submitting_a_Patch
https://bucardo.org/postgres_all_versions.html

TECHPARK SÜDTIROL/ALTO ADIGE

THANK YOU.

NOI TECHPARK
SÜDTIROL/ALTO ADIGE

VIA A.-VOLTA-STRASSE 13/A
I – 39100 BOZEN/BOLZANO

+39 0471/066 600
INFO@NOI.BZ.IT
WWW.NOI.BZ.IT