HypoPG

HypoPG is a PostgreSQL extension adding support for hypothetical indexes.

A hypothetical – or virtual – index is an index that doesn't really exist, and thus doesn't cost CPU, disk or any resource to create. They're useful to know if specific indexes can increase performance for problematic queries, since you can know if PostgreSQL will use these indexes or not without having to spend resources to create them.

For more thorough information, please consult the official documentation.

For other general information, you can also consult this blog post.

Installation

- Compatible with PostgreSQL 9.2 and above
- Needs PostgreSQL header files
- Decompress the tarball
- `sudo make install`
- In every needed database: `CREATE EXTENSION hypopg;`

Updating the extension

Note that hypopg doesn't provide extension upgrade scripts, as there's no data saved in any of the objects created. Therefore, you need to first drop the extension then create it again to get the new version.

Usage

NOTE: The hypothetical indexes are contained in a single backend. Therefore, if you add multiple hypothetical indexes, concurrent connections doing `EXPLAIN` won't be bothered by your hypothetical indexes.

Assuming a simple test case:

```
rjuju=# CREATE TABLE hypo AS SELECT id, 'line ' || id AS val FROM generate_series(1,10000) i
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                      QUERY PLAN
-------------------------------------------------------
 Seq Scan on hypo  (cost=0.00..180.00 rows=1 width=13)
   Filter: (id = 1)
(2 rows)
```

The easiest way to create an hypothetical index is to use the `hypopg_create_index` functions with a regular `CREATE INDEX` statement as arg.

For instance:

```
rjuju=# SELECT * FROM hypopg_create_index('CREATE INDEX ON hypo (id)');
```

NOTE: Some information from the `CREATE INDEX` statement will be ignored, such as the index name if provided. Some of the ignored information will be handled in a future release.

You can check the available hypothetical indexes in your own backend:

```
rjuju=# SELECT * FROM hypopg_list_indexes ;
 indexrelid |          index_name           | schema_name | table_name | am_name
------------+-------------------------------+-------------+------------+---------
      50573 | <50573>btree_hypo_id          | public      | hypo       | btree
```

If the `CREATE INDEX` command you want to use also needs quoting, using the dollar quoting syntax is recommended. For instance:

```
rjuju=# SELECT * FROM hypopg_create_index($$CREATE INDEX ON hypo (id) WHERE val = 'line 1'$$
```

If you need more technical information on the hypothetical indexes, the `hypopg()` function will return the hypothetical indexes in a similar way as `pg_index` system catalog.

And now, let's see if your previous `EXPLAIN` statement would use such an index:

```
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                                     QUERY PLAN
-------------------------------------------------------------------------------------
 Index Scan using <41072>hypo_btree_hypo_id on hypo  (cost=0.29..8.30 rows=1 width=13)
   Index Cond: (id = 1)
(2 rows)
```

Of course, only `EXPLAIN` without `ANALYZE` will use hypothetical indexes:

```
rjuju=# EXPLAIN ANALYZE SELECT * FROM hypo WHERE id = 1;
                                           QUERY PLAN
-----------------------------------------------------------------------------------------------
 Seq Scan on hypo  (cost=0.00..180.00 rows=1 width=13) (actual time=0.036..6.072 rows=1 loop
   Filter: (id = 1)
   Rows Removed by Filter: 9999
 Planning time: 0.109 ms
 Execution time: 6.113 ms
(5 rows)
```

To remove your backend's hypothetical indexes, you can use the function `hypopg_drop_index(indexrelid)` with the OID that the `hypopg_list_indexes` view returns and call `hypopg_reset()` to remove all at once, or just close your current connection.

Continuing with the above case, you can `hide existing indexes`, but should be use `hypopg_reset()` to clear the previous effects of other indexes at first.

Create two real indexes and run `EXPLAIN`:

```
rjuju=# SELECT hypopg_reset();
```

```
rjuju=# CREATE INDEX ON hypo(id);
rjuju=# CREATE INDEX ON hypo(id, val);
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                                    QUERY PLAN
----------------------------------------------------------------------------------
Index Only Scan using hypo_id_val_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

The query plan is using the `hypo_id_val_idx` index. Use `hypopg_hide_index(oid)` to hide one of the indexes:

```
rjuju=# SELECT hypopg_hide_index('hypo_id_val_idx'::REGCLASS);
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                            QUERY PLAN
-------------------------------------------------------------------------
Index Scan using hypo_id_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

The query plan is using the other index `hypo_id_idx` now. Use `hypopg_hide_index(oid)` to hide it:

```
rjuju=# SELECT hypopg_hide_index('hypo_id_idx'::REGCLASS);
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                    QUERY PLAN
-------------------------------------------------------
Seq Scan on hypo  (cost=0.00..180.00 rows=1 width=13)
Filter: (id = 1)
(2 rows)
```

And now the query plan changes back to `Seq Scan`. Use `hypopg_unhide_index(oid)` to restore index:

```
rjuju=# SELECT hypopg_unhide_index('hypo_id_idx'::regclass);
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                            QUERY PLAN
-------------------------------------------------------------------------
Index Scan using hypo_id_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

Of course, you can also hide hypothetical indexes:

```
rjuju=# SELECT hypopg_create_index('CREATE INDEX ON hypo(id)');
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                                    QUERY PLAN
------------------------------------------------------------------------------------
Index Scan using "<12659>btree_hypo_id" on hypo  (cost=0.04..8.05 rows=1 width=13)
Index Cond: (id = 1)
```

```
(2 rows)

rjuju=# SELECT hypopg_hide_index(12659);
rjuju=# EXPLAIN SELECT * FROM hypo WHERE id = 1;
                    QUERY PLAN
-------------------------------------------------------
Seq Scan on hypo  (cost=0.00..180.00 rows=1 width=13)
Filter: (id = 1)
(2 rows)
```

You can check which indexes are hidden using `hypopg_hidden_indexes()` or the `hypopg_hidden_indexes` view:

```
rjuju=# SELECT * FROM hypopg_hidden_indexes();
indexid
---------
526604
526603
12659
(3 rows)

rjuju=# SELECT * FROM hypopg_hidden_indexes;
 indexrelid |      index_name      | schema_name | table_name | am_name | is_hypo
------------+----------------------+-------------+------------+---------+---------
      12659 | <12659>btree_hypo_id | public      | hypo       | btree   | t
     526603 | hypo_id_idx          | public      | hypo       | btree   | f
     526604 | hypo_id_val_idx      | public      | hypo       | btree   | f
(3 rows)
```

To restore all existing indexes, you can use the function `hypopg_unhide_all_indexes()`. Note that the functionality to hide existing indexes only applies to the EXPLAIN command in the current session and will not affect other sessions.