Hypothetical Indexes on PostgreSQL

This tutorial shows the usage of hypothetical indexes on PostgreSQL. Hypothetical indexes were first discussed in [Frank, Omiecinski, Navathe, 92]

Hypothetical indexes are simulated index structures created solely in the database catalog. This type of index has no physical extension and, therefore, cannot be used to answer queries. The main benefit is to provide a means for simulating how query execution plans would change if the hypothetical indexes were actually created in the database. Thus this feature is useful for database tuners and DBAs.

Index selection tools, such as Microsoft's SQL Server Index Tuning Wizard, make use of hypothetical indexes in the database server to evaluate candidate index configurations.

We have made some server extensions to PostgreSQL 9.0.1 to include the notion of hypothetical indexes in the system. We have introduced three new commands: `create hypothetical index`, `drop hypothetical index` and `explain hypothetical`. To download the Hypothetical Plugin code, click here.

Begin Tutorial

Hypothetical Indexes on PostgreSQL

This tutorial uses an example database that stores product (produto) and sale (venda) information for an enterprise. The database consists of two relations:

Product (Produto)

num – int
descricao – varchar(50)

Number of tuples: 4

Sale (Venda)

num – int
prodNum – int
data – date
qtd – int
valor – numeric(10,2)

Number of tuples: 400000

Actual indexes are created for all of the tables' primary keys. Scripts to create the enterprise database can be found here.

Previous Back to Beginning Next

Hypothetical Indexes on PostgreSQL

The following query is very frequently issued by the enterprise application:

```
Select prodNum, data, sum(valor) as total
from   venda
where valor > 1500000 and
      data between '2004-01-01' and '2004-01-31'
      group by prodNum, data;
```

Lets take a look at its query execution plan using the *explain* statement:

```
explain
select prodNum, data, sum(valor) as total
from   venda
where valor > 1500000 and
      data between '2004-01-01' and '2004-01-31'
      group by prodNum, data;
```

Previous Back to Beginning Next

Hypothetical Indexes on PostgreSQL

Query Execution Plan

```
HashAggregate  (cost=9778.97..9779.11 rows=11 width=18)

 ->  Seq Scan on venda  (cost=0.00..9759.00 rows=2663 width=18)

       Filter: ((valor > 1500000::numeric) AND (data >= '2004-01-01'::date) AND
                 (data <= '2004-01-31'::date))
```

A sequential scan was chosen by the planner to access the venda table. Perhaps we could improve this by creating an index on the *valor* and *data* columns.

Previous Back to Beginning Next

Hypothetical Indexes on PostgreSQL

Although we could benefit from the existence of an index on the *valor* and *data* columns , we should be careful to create it. Firstly, we do not know if the DBMS will actually choose to use an index in the *valor* and *data* columns if it exists. Secondly, if we try to create an actual index in these columns, the DBMS will prevent writers from accessing the table. So it is hard to experiment with new indexes and evaluate how good they are.

Instead of incurring the burden of creating an actual index on the columns, we could simulate if this index would be useful to the database. To do that, we create it as a hypothetical index:

```
create hypothetical index hi_venda_valor_data
on venda (valor, data);
```

The create hypothetical index command also exists in other DBMSs, but with a different syntax. For example, see the syntax proposed for SQL Server in [Chaudhuri, Narasayya, 98]

Previous Back to Beginning Next

Hypothetical Indexes on PostgreSQL

The hypothetical index is not actually materialized in the database. Therefore, we will not incur in heavy creation costs or obtain locks on the underlying table to create it. The DBMS, however, cannot use the hypothetical index to answer a user query. If we query the database again or use the *explain* statement, the system will still use a sequential scan to access the *employee* table.

We can see how the DBMS would behave if the hypothetical index were materialized using the *explain hypothetical* statement:

```
explain hypothetical
select prodNum, data, sum(valor) as total
from   venda
where valor > 1500000 and
      data between '2004-01-01' and '2004-01-31'
      group by prodNum, data;
```

Hypothetical Indexes on PostgreSQL

Query Execution Plan

```
HashAggregate  (cost=3976.05..3976.19 rows=11 width=18)

 ->  Bitmap Heap Scan on venda  (cost=1072.26..3956.08 rows=2663 width=18)
       Recheck Cond: ((valor > 1500000::numeric) AND (data >= '2004-01-01'::date)
                AND (data <= '2004-01-31'::date))

       ->  Bitmap Index Scan on hi_venda_valor_data  (cost=0.00..1071.60
                rows=2663 width=0)
             Index Cond: ((valor > 1500000::numeric) AND (data >= '2004-01-01'::date)
                AND (data <= '2004-01-31'::date))
```

If the index *hi_venda_valor_data* was materialized, the DBMS would use it to process the query. The estimated cost to process the query would drop from 9778.97..9779.11 using the sequential scan to 3976.05..3976.19 using the index scan.

Previous

Back to Beginning

Next

Hypothetical Indexes on PostgreSQL

Now that we know that the index is beneficial to performance, we can drop the hypothetical index and create a corresponding actual one:

```
drop hypothetical index hi_venda_valor_data;

create index i_venda_valor_data on venda ( valor, data );
```

Hypothetical Indexes on PostgreSQL

Lets check the query execution plan for the query with the actual index created:

```
explain
select prodNum, data, sum(valor) as total
from   venda
where valor > 1500000 and
      data between '2004-01-01' and '2004-01-31'
      group by prodNum, data;
```

Hypothetical Indexes on PostgreSQL

Query Execution Plan

```
HashAggregate  (cost=3880.04..3880.18 rows=11 width=18)

 ->  Bitmap Heap Scan on venda  (cost=976.25..3860.07 rows=2663 width=18)
       Recheck Cond: ((valor > 1500000::numeric) AND (data >= '2004-01-01'::date)
                AND (data <= '2004-01-31'::date))

       ->  Bitmap Index Scan on i_venda_valor_data  (cost=0.00..975.59
                rows=2663 width=0)
             Index Cond: ((valor > 1500000::numeric) AND (data >= '2004-01-
                01'::date) AND (data <= '2004-01-31'::date))
```

The cost estimated by the planner for the query using the hypothetical index was 3976.05..3976.19. With the actual index, the planner gave us an estimate of 3880.04..3880.18. Cost estimates for hypothetical indexes tend to be conservative, but always close to the cost of using the actual index.

Previous

Back to Beginning

Next

Hypothetical Indexes on PostgreSQL

Estimations made for hypothetical indexes tend to produce conservative cost values. The cost values are bigger than those verified for the corresponding actual indexes. This happens because we have made some approximations to estimate the index size.

We approximate the total number of tuples present in the index by the number of tuples present in the table being indexed. We also approximate the number of pages in the index by the number of pages in the table.

Using these estimates, the query optimizer tends to consider the index bigger than it would actually be if materialized. This means that the cost calculations made for this index will produce higher I/O figures than the calculations for the corresponding actual index.

A positive consequence of this policy is that we will never recommend an index that will not be chosen by the optimizer when materialized.

Previous Back to Beginning Next

Hypothetical Indexes on PostgreSQL

That ends our tutorial. We hope the tutorial has been useful for you to understand how hypothetical indexes can be used to simulate index configurations for the database. One important fact to notice is that the addition of hypothetical indexes does not impact previously existing applications. The new feature is aimed primarily at database tuners and DBAs.

After implementing the server extensions for hypothetical indexes, the next logical step is to implement automatic index selection tools and algorithms for the PostgreSQL database. We are currently doing that at PUC-Rio. One interesting research prototype we obtained is a software agent, written in C++, that can be integrated to the DBMS in order to make index selection and creation totally autonomic.

Previous Back to Beginning Next

Hypothetical Indexes on PostgreSQL

If you would like more details on how all of this stuff was implemented, email abrito@inf.puc-rio.br. You can also contact our research group head sergio@inf.puc-rio.br.

Thank you for your interest! ☺

Previous

Back to Beginning