

JULY 24

Found a full usage for PERM SQL-PLE. All info is in the paper <http://cs.iit.edu/~dbgroup/assets/pdfpubs/G10a.pdf>

If need to find specific usage, search in the paper.

This appendix presents a grammar for *SQL-PLE*. We use the grammatical representation that is used in the *PostgreSQL* manual. *SQL-PLE* extensions are highlighted in red. The new language constructs added by *SQL-PLE* are presented below.

| Construct | Description |
|-----------------------------|---|
| PROVENANCE | Marks a query for provenance computation |
| ON CONTRIBUTION (cs_type) | Instructs <i>Perm</i> to use a certain <i>CS</i> type |
| BASERELATION | Handle a from-clause item as if it were a base relation |
| PROVENANCE (attr_list) | Handle attributes from <i>attr_list</i> as provenance attributes |
| TRANSPROV/TRANSXML/TRANSXML | Mark a query for transformation provenance computation |
| EXPLAIN SQLTEXT | Return the (rewritten) SQL text of an query |
| EXPLAIN GRAPH | Return an algebra tree for an query (as a dot-language script) |
| MAPPROV | Compute the <i>mapping</i> provenance and represent it as sets of mappings. |
| THIS.childpath | A shortcut for generating the XML representation of the sub-query accessed by <i>path</i> . |
| XSLT.f (xml_param) | Apply XSLT function <i>f</i> to XML document <i>xml_param</i> . |
| cxpath (path_expr, input) | Evaluate X-Path expression <i>path_expr</i> over <i>input</i> . Returns true if the evaluation of the XPath expression returns at least one result. |
| ANNOT(annotation) | Annotates the <i>FROM</i> clause item it is appended to with <i>annotation</i> . |

All CS_TYPE:

I-CS (Influence CS):

Contribution semantics belonging to this class include all data items in the provenance of an output data item *d*, that had some influence on the creation of *d*.

(influence)

C-CS (Copy CS):

Under copy-CS(C-CS) a data item is considered to belong to the provenance of an output data item, if it has been copied literally from the input to the output (we do not specify if complete or just partial copying is required).

(copy + partial/complete/none + transitive/none(non-

transitive))

| CS type | Description |
|---|---|
| Complete-Direct-Copy-CS (CDC-CS) | Only tuples that have been copied directly as a whole from the input to the output of a query are considered to belong to the provenance. |
| Partial-Direct-Copy-CS (PDC-CS) | Only tuples from which at least one attribute value has been copied directly from the input to the output belong to the provenance. |
| Complete-Transitive-Copy-CS (CTC-CS) | <i>CTC-CS</i> contains all directly copied tuples. In addition, implied equalities enforced by selection conditions are handled as copy operations. |
| Partial-Transitive-Copy-CS (PTC-CS) | Like <i>PDC-CS</i> , but implied equalities are considered as copy operations. |

Figure 3.12: C-CS types

Where-CS:
 (where)
How-CS:
 (how)
Input-CS:
 Not Usable in PERM

Following are the outputs from How-Provenance.

DATABASE:

```
links (  
    moviold integer,  
    imdbld integer,  
    tmdbld integer  
);  
  
ratings (  
    userId integer,  
    moviold integer,  
    rating real,  
    timestamp integer  
);  
Imported from two csv files. Tried to restore postgresql sample
```

database, failed. Maybe due to PERM's postgresql version is too old (8.3).

links.csv

comma-separated values
183 KB



ratings_small.csv

comma-separated values
2.4 MB



1.
select provenance l.movielid
from links as l join ratings as r
on l.movielid = r.movielid and r.rating > 4 and r.userId = 10;

| movielid | rcv_public_links_movielid | rcv_public_links_tmdbid | rcv_public_links_tmdbid | rcv_public_ratings_userid | rcv_public_ratings_movielid | rcv_public_ratings_rating | rcv_public_ratings_timestamp |
|----------|---------------------------|-------------------------|-------------------------|---------------------------|-----------------------------|---------------------------|------------------------------|
| 50 | 50 | 114814 | 629 | 10 | 50 | 5 | 942766420 |
| 1358 | 1358 | 117666 | 12498 | 10 | 1358 | 5 | 942766420 |
| 1611 | 1611 | 102494 | 468 | 10 | 1611 | 5 | 942767029 |
| 1719 | 1719 | 120235 | 10217 | 10 | 1719 | 5 | 942766472 |
| 1923 | 1923 | 129387 | 544 | 10 | 1923 | 5 | 942766515 |
| 2344 | 2344 | 89941 | 11893 | 10 | 2344 | 5 | 942766991 |
| 2571 | 2571 | 133893 | 683 | 10 | 2571 | 5 | 942766515 |
| 2826 | 2826 | 120657 | 1911 | 10 | 2826 | 5 | 942766109 |
| 2926 | 2926 | 95270 | 11054 | 10 | 2926 | 5 | 942767121 |

movie_why

data
2 KB



Why provenance for (finding movie with rating > 4 and rated by user 10)

2.
select provenance on contribution (how) l.movielid
from links as l join ratings as r
on l.movielid = r.movielid and r.rating > 4 and r.userId = 10;

| movieid | howprov |
|----------|---------|
| 1611 | * 0 0 |
| 2926 | * 0 0 |
| 2344 | * 0 0 |
| 2826 | * 0 0 |
| 1358 | * 0 0 |
| 1923 | * 0 0 |
| 50 | * 0 0 |
| 2571 | * 0 0 |
| 1719 | * 0 0 |
| (9 rows) | |

movie_how

data

221 bytes



How provenance. The output is really confusing, so I tried on a smaller database.

Small Database:

```
student (
  name  varchar,
  age   integer,
  major  car char
)
```

```
siebel (
  name  archer,
  time   integer
)
```

This is the database from last week.

```
1.
select provenance s.name
from student as s join siebel as b
on s.name = b.name;
```

| name | prov_public_student_name | prov_public_student_age | prov_public_student_major | prov_public_siebel_time | prov_public_siebel_name |
|------|--------------------------|-------------------------|---------------------------|-------------------------|-------------------------|
| a | a | 1 | cs | 10 | a |
| a | a | 1 | cs | 11 | a |
| a | a | 1 | cs | 12 | a |
| a | a | 1 | cs | 13 | a |
| a | a | 1 | cs | 14 | a |
| a | a | 1 | cs | 15 | a |
| a | a | 1 | cs | 16 | a |
| a | a | 1 | cs | 17 | a |
| a | a | 1 | cs | 18 | a |
| a | a | 1 | cs | 19 | a |
| a | a | 1 | cs | 20 | a |
| b | b | 2 | ces | 8 | b |

(12 rows)

Why provenance same as last week

2.

select provenance on contribution (how) s.name
from student as s join siebel as b
on s.name = b.name;

| name | howprov |
|------|---|
| b | * 0 0 |
| a | + + + + + + + + * 0 0 * 0 0 * 0 0 * 0 0 * 0 0 * 0 0 * 0 0 * 0 0 * 0 0 * 0 0 |

(2 rows)

sqltest_how

data

436 bytes



How provenance. Here we could see that this is represented as a tree structure. The first are the roots, and the following are the nodes.

The output is actually $*00 + *00 + *00 + \dots + *00$. And $*00$ is actually $0 * 0$.

$0 * 0$ means join of two tables.

$0 * 0 + 0 * 0$ means that there are two possible tuple joins which could form the result.

And the above how-provenance result means that there are 11 possible tuple joins which may form the final result. This corresponds to the output from why-provenance.

The only problem is that I can't know which two tables are joined since they are represented by two identical 0.

Then, I tried a join of three tables

third (

```
3.
select provenance ttt.n
from third join
(select s.name as n, s.age as a
from student as s join siebel as b
on s.name = b.name) as ttt
on third.id = ttt.a;
```

why-provenance

```
4.
select provenance on contribution (how) ttt.n
from third join
(select s.name as n, s.age as a
from student as s join siebel as b
on s.name = b.name) as ttt
on third.id = ttt.a;
```

```
sqltest3_how
data
628 bytes
```



third join (student join Siebel).

Now, it is clear about how to understand the output of how-provenance. The only drawback is that it can not show exactly which tuple or table contributes to the join.