Department of Informatics, University of Zürich

Report

Mapping of Queries with Statement Modifiers to Queries with Temporal Primitives

Oliver Leumann

Email: oliver.leumann@uzh.ch

April 21, 2013

supervised by Prof. Dr. M. Böhlen and A. Dignös

University of Zurich UZH

Department of Informatics

D B T G

Contents

1 Introduction

In this self-study project, the main goal is to define a SQL mapping that translates queries with statement modifiers to queries with temporal primitives over algebra expressions. The comprehension of the temporal primitives and the reduction of temporal operators [1] are important prerequisites before we can find a solution for the SQL mapping. Also the use of statement modifiers [2], as well as having a look on the PostgreSQL parser [3], are necessary to get an idea how the mapping could work.

First of all, we will start with a relatively simple example of a nontemporal query on a sample database. Then, a second example will be proposed which expands the database, so we can run a temporal counterpart of the first query. After that, the second chapter shows the knowledge we need, including examples that build on the temporal query. Having the needed background, the third chapter will then examine the SQL mapping.

Example 1. Consider employees with each of them working at a certain department. For each department, there is one manager responsible for the employees. The employees are recorded in relation $\mathbf{E}$, where $Name$ is the name of an employee and $Dept$ the department the employee is working at. Relation $\mathbf{M}$ records the managers, where Mgr is a manager's name and $Dept$ the department he's responsible for. The relations of our first example are shown in Figure 1.1(a).

If we want to determine the number of employees per manager, we would formulate something like the following SQL-query Q:

```
SELECT Mgr, count(*)
FROM M JOIN E ON M.Dept = E.Dept
GROUP BY Mgr
```

The result of the query Q is shown in Figure 1.1(b). Since the employees Amber and Chelsea (e_1 and e_3) work at department one and Billy (e_2) at department two, we get in our result-relation one tuple that records that Xavier supervises two employees (z_1) whereas Yvonne has only one employee (z_2).

$\mathbf{E}$

	Name	*Dept*
e_1	Amber	1
e_2	Billy	2
e_3	Chelsea	1

$\mathbf{M}$

	Mgr	*Dept*
m_1	Xavier	1
m_2	Yvonne	2

(a) Nontemporal Relations

$\mathbf{Z}$

	Mgr	*count*
z_1	Xavier	2
z_2	Yvonne	1

(b) Result

Figure 1.1: Nontemporal example.

Example 2. In Figure 1.2(a) we have almost the same example as before. The only difference is that we extend the previous relations from the first example, so that we are able to run temporal queries on them.

Again we have the two relations consisting of employees $\mathbf{E}$ and managers $\mathbf{M}$. But this time, the data is timestamped with intervals, represented by the attribute T. Keep in mind that the intervals are represented as a pair $[T_s, T_e)$, where the start-point T_s is inclusive and end-point T_e is exclusive. For instance, e_3 records that Chelsea works from the beginning of July until the end of November. Furthermore the new tuple e_4 is added to the employee-relation $\mathbf{E}$, which records that Amber changes her department during the year. She changes the department at the end of August and works at department two from September until December.

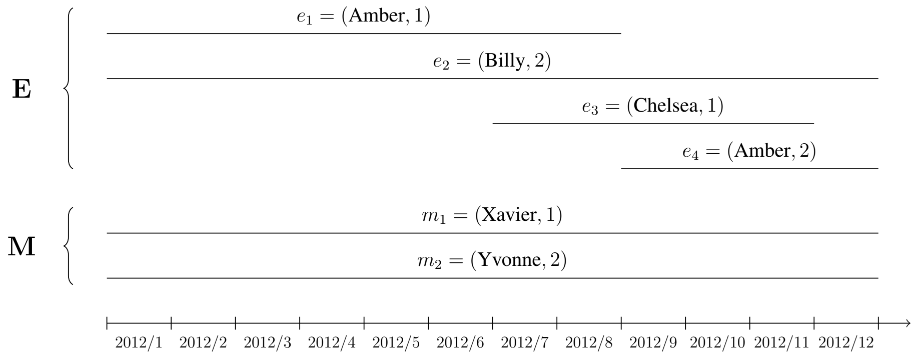
E

	Name	*Dept*	*T*
e_1	Amber	1	[2012/01, 2012/09)
e_2	Billy	2	[2012/01, 2013/01)
e_3	Chelsea	1	[2012/07, 2012/12)
e_4	Amber	2	[2012/09, 2013/01)

M

	Mgr	*Dept*	*T*
m_1	Xavier	1	[2012/01, 2013/01)
m_2	Yvonne	2	[2012/01, 2013/01)

(a) Temporal Relations



E
$e_1 = (\text{Amber}, 1)$
$e_2 = (\text{Billy}, 2)$
$e_3 = (\text{Chelsea}, 1)$
$e_4 = (\text{Amber}, 2)$
M
$m_1 = (\text{Xavier}, 1)$
$m_2 = (\text{Yvonne}, 2)$
2012/1
2012/2
2012/3
2012/4
2012/5
2012/6
2012/7
2012/8
2012/9
2012/10
2012/11
2012/12

(b) Graphical Representation

Figure 1.2: Temporal Example.

In contrast to query Q we are now interested in a temporal query and its result with respect to the *sequenced semantics*. That means we want to determine the number of employees per manager at each point in time. In order to do this, we prepend the statement modifier for sequenced queries `SEQ VT` to the previous query Q and get the new temporal sql-query Q^T:

```
SEQ VT
SELECT Mgr, count(*)
FROM M JOIN E ON M.Dept = E.Dept
GROUP BY Mgr
```

Figure 1.2(b) shows a graphical representation of the temporal relations, where the tuples are drawn as horizontal lines according to their timestamps. With the help of this graphical representation it is quite easy to see what the result of the query Q^T should be. For instance, Xavier (m_1) has one employee (Amber: e_1) from January until June, two employees (Amber: e_1, Chelsea: e_3) from July until August and again only one employee (Chelsea: e_3) from September until November. The result of the query Q^T is shown in Figure 1.3.

Z

	Mgr	*count*	*T*
z_1	Xavier	1	[2012/01, 2012/07)
z_2	Xavier	2	[2012/07, 2012/09)
z_3	Xavier	1	[2012/09, 2012/12)
z_4	Yvonne	1	[2012/01, 2012/09)
z_5	Yvonne	2	[2012/09, 2013/01)

Figure 1.3: Result of the Temporal Query Q^T.

2 Background

This section explains in what way the reduction rules are using temporal primitives to reduce the operators of a temporal algebra to their nontemporal counterparts. Some detailed examples will thereby help to understand the functional principle of the temporal primitives.

2.1 Reduction Rules

The relational algebra representation of the query Q from the first example in the chapter before would look as follows:

$$Q_{ra} \quad = \quad {}_{Mgr}\vartheta_{count(*)}(\mathbf{M} \bowtie_{\mathbf{M}.Dept=\mathbf{E}.Dept} \mathbf{E})$$

Similarly, the query Q^T from the second example, can be represented as a relational algebra expression. We just have to mark the join- and the aggregation-operator as temporal operators ϑ^T and $\bowtie^T$:

$$Q^T_{ra} \quad = \quad {}_{Mgr}\vartheta^T_{count(*)}(\mathbf{M} \bowtie^T_{\mathbf{M}.Dept=\mathbf{E}.Dept} \mathbf{E})$$

Note that in order to get the result of the query, we first have to compute a temporal inner join and then perform a temporal aggregation. The reduction rules introduced by Dignös et al. [1] will help us to transform the temporal algebra expression Q^T_{ra} to its nontemporal representation. The rules are shown here in table 2.1.

We split the query Q^T_{ra} up to the temporal inner join and the temporal aggregation and apply the corresponding reduction rules at the temporal operators:

$$\mathbf{X} \leftarrow \alpha((\mathbf{M}\Phi_{\mathbf{M}.Dept=\mathbf{E}.Dept}\mathbf{E}) \bowtie_{\mathbf{M}.Dept=\mathbf{E}.Dept \wedge \mathbf{M}.T=\mathbf{E}.T} (\mathbf{E}\Phi_{\mathbf{M}.Dept=\mathbf{E}.Dept}\mathbf{M}))$$

$${}_{Mgr,T}\vartheta_{count(*)}(\mathcal{N}_{Mgr}(\mathbf{X};\mathbf{X}))$$

As we have a look at the rules for the inner join and the aggregation, we see that there are several operators which we don't know from the traditional relational algebra. $\alpha(\mathbf{r})$ is called the absorb operator and is an operator that eliminates temporal duplicates. The alignment operator $\mathbf{r}\Phi_\theta\mathbf{s}$ as well as the normalization operator $\mathcal{N}_A(\mathbf{r};\mathbf{r})$ are the previously mentioned temporal primitives, that adjust the timestamps of the tuples of our argument relations.

Operator			**Reduction**
Selection	$\sigma^T_\theta(r)$	$=$	$\sigma_\theta(r)$
Projection	$\pi^T_B(r)$	$=$	$\pi_{B,T}(\mathcal{N}_B(r;r))$
Aggregation	${}_B\vartheta^T_F(r)$	$=$	${}_{B,T}\vartheta_F(\mathcal{N}_A(r;r))$
Difference	$r -^T s$	$=$	$\mathcal{N}_A(r;s) - \mathcal{N}_A(s;r)$
Union	$r \cup^T s$	$=$	$\mathcal{N}_A(r;s) \cup \mathcal{N}_A(s;r)$
Intersection	$r \cap^T s$	$=$	$\mathcal{N}_A(r;s) \cap \mathcal{N}_A(s;r)$
Cart. Prod.	$r \times^T_\theta s$	$=$	$\alpha((r\Phi_{true}s) \times_{r.T=s.T} (s\Phi_{true}r))$
Inner Join	$r \bowtie^T_\theta s$	$=$	$\alpha((r\Phi_\theta s) \bowtie_{\theta\wedge r.T=s.T} (s\Phi_\theta r))$
Left O. Join	$r ⟕^T_\theta s$	$=$	$\alpha((r\Phi_\theta s) ⟕_{\theta\wedge r.T=s.T} (s\Phi_\theta r))$
Right O. Join	$r ⟖^T_\theta s$	$=$	$\alpha((r\Phi_\theta s) ⟖_{\theta\wedge r.T=s.T} (s\Phi_\theta r))$
Full O. Join	$r ⟗^T_\theta s$	$=$	$\alpha((r\Phi_\theta s) ⟗_{\theta\wedge r.T=s.T} (s\Phi_\theta r))$
Anti Join	$r \triangleright^T_\theta s$	$=$	$(r\Phi_\theta s) \triangleright_{\theta\wedge r.T=s.T} (s\Phi_\theta r)$

Table 2.1: Reduction Rules

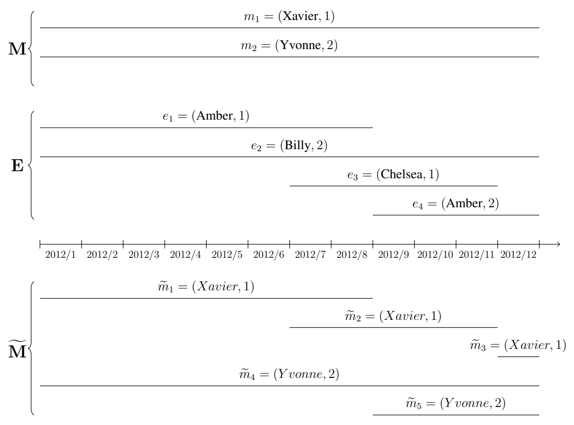
2.2 The Alignment Operator

The temporal alignment operator $\mathbf{r}\Phi_\theta\mathbf{s}$ occurs two times in our example and at its first occurrence it has the form $\mathbf{M}\Phi_{\mathbf{M}.Dept=\mathbf{E}.Dept}\mathbf{E}$. This means that the manager-relation $\mathbf{M}$ will be aligned with respect to the employee-relation $\mathbf{E}$ whereby the condition $\mathbf{M}.Dept = \mathbf{E}.Dept$ has to be satisfied. In detail, it says that we get the aligned result tuples as follows: We search for each tuple in $\mathbf{M}$ the tuple(s) in $\mathbf{E}$ with the same department and for each tuple in $\mathbf{E}$ that fulfills the condition, we get a result-tuple with the nontemporal attributes of the tuple of $\mathbf{M}$ and the time interval that results from the intersection of the $\mathbf{M}$-tuple with the particular $\mathbf{E}$-tuple. Also for each gap in the interval of a $\mathbf{M}$-tuple that is not covered by any $\mathbf{E}$-tuple, we get a result-tuple.

Example 3. Figure 2.1(a) shows the alignment of $\mathbf{M}$ with respect to $\mathbf{E}$ using the theta-condition $\theta \equiv (\mathbf{M}.Dept = \mathbf{E}.Dept)$. For instance, the first result tuple, (Xavier, 1, [2012/1, 2012/9)), is derived from m_1 and e_1 over their common interval, and the second result tuple, (Xavier, 1, [2012/9,2012/12)), from m_1 and e_2 over their common interval. The third result tuple matches an interval that is a sub-interval of m_1 which is not covered by any tuple in $\mathbf{E}$.

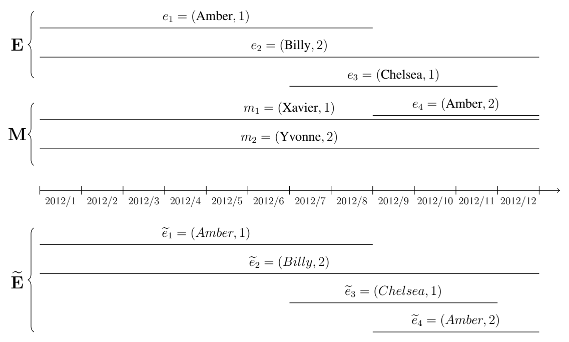
Simultaneously, the alignment operator will also work the same way for its second occurrence $(\mathbf{E}\Phi_{\mathbf{M}.Dept=\mathbf{E}.Dept}\mathbf{M})$. Only this time, $\mathbf{E}$ will be aligned with respect to $\mathbf{M}$, but notice that the condition $\mathbf{M}.Dept = \mathbf{E}.Dept$ is still the same. The second alignment operation with its result relation $\widetilde{\mathbf{E}}$ is shown in figure 2.1(b).

Example 4. After the alignment of the relations $\mathbf{M}$ and $\mathbf{E}$, let us perform the inner join on the aligned argument relations $\widetilde{\mathbf{M}}$ and $\widetilde{\mathbf{E}}$. As we can see from the reduction rule for inner joins in table 2.1, $\alpha((\mathbf{r}\Phi_\theta\mathbf{s}) \bowtie_{\theta\wedge\mathbf{r}.T=\mathbf{s}.T} (\mathbf{s}\Phi_\theta\mathbf{r}))$, we have have to perform the join not only with respect to the theta-condition $\theta \equiv (\mathbf{M}.Dept = \mathbf{E}.Dept)$, but also with respect to the fact that the interval timestamps from $\widetilde{\mathbf{M}}$ and $\widetilde{\mathbf{E}}$ should be equal, i.e. $\widetilde{\mathbf{M}}.T = \widetilde{\mathbf{M}}.T$. The result relation



M
$m_1 = (\text{Xavier}, 1)$
$m_2 = (\text{Yvonne}, 2)$
E
$e_1 = (\text{Amber}, 1)$
$e_2 = (\text{Billy}, 2)$
$e_3 = (\text{Chelsea}, 1)$
$e_4 = (\text{Amber}, 2)$
2012/1 2012/2 2012/3 2012/4 2012/5 2012/6 2012/7 2012/8 2012/9 2012/10 2012/11 2012/12
$\widetilde{\mathbf{M}}$
$\widetilde{m}_1 = (Xavier, 1)$
$\widetilde{m}_2 = (Xavier, 1)$
$\widetilde{m}_3 = (Xavier, 1)$
$\widetilde{m}_4 = (Yvonne, 2)$
$\widetilde{m}_5 = (Yvonne, 2)$

(a) Alignment of $\mathbf{M}$



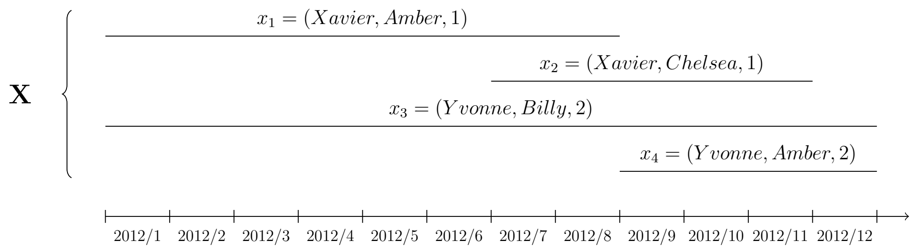
E
$e_1 = (\text{Amber}, 1)$
$e_2 = (\text{Billy}, 2)$
$e_3 = (\text{Chelsea}, 1)$
M
$m_1 = (\text{Xavier}, 1)$
$e_4 = (\text{Amber}, 2)$
$m_2 = (\text{Yvonne}, 2)$
2012/1 2012/2 2012/3 2012/4 2012/5 2012/6 2012/7 2012/8 2012/9 2012/10 2012/11 2012/12
$\widetilde{\mathbf{E}}$
$\widetilde{e}_1 = (Amber, 1)$
$\widetilde{e}_2 = (Billy, 2)$
$\widetilde{e}_3 = (Chelsea, 1)$
$\widetilde{e}_4 = (Amber, 2)$

(b) Alignment of $\mathbf{E}$

Figure 2.1: Temporal Alignments

$\mathbf{X}$ after this inner join is shown graphically in Figure 2.2.

The absorb operator $\alpha(\mathbf{r})$, which has to be performed as last of the reduction of temporal inner joins, helps us to remove temporal duplicates. If in the argument relation exist value-equivalent tuples, such that one of those timestamp intervals is a subset of the other tuple's interval, this temporal duplicate would be removed. In our previously computed relation $\mathbf{X}$ we have no such temporal duplicates. But hypothetically, if there would exist a fifth tuple x_5 with the values $(Yvonne, Amber, 2)$ over the interval $[2012/10, 2012/12)$, this tuple x_5 would be removed by the absorb operator since it is contained in the value-equivalent tuple x_4.



X
$x_1 = (Xavier, Amber, 1)$
$x_2 = (Xavier, Chelsea, 1)$
$x_3 = (Yvonne, Billy, 2)$
$x_4 = (Yvonne, Amber, 2)$
2012/1 2012/2 2012/3 2012/4 2012/5 2012/6 2012/7 2012/8 2012/9 2012/10 2012/11 2012/12

Figure 2.2: Result after Inner Join

2.3 The Normalization Operator

The temporal normalization operator $\mathcal{N}_A(\mathbf{r}; \mathbf{s})$ is the next operator that has to be applied in our example and then would look like $\mathcal{N}_{Mgr}(\mathbf{X}; \mathbf{X})$. So the relation $\mathbf{X}$, that is created after the inner join, will be normalized with respect to itself, $\mathbf{X}$, and the Attribute Mgr. That means that we get the normalized result tuples if we adjust the intervals of each tuple in $\mathbf{X}$ by the start and end point of all tuples of $\mathbf{X}$ in the same group. In our example, $\mathbf{X}$ is grouped by the attribute Mgr into two groups, one with $Mgr = Xavier$ and the other with $Mgr = Yvonne$.

And again, similarly to the alignment operator, each gap in the interval of a $\mathbf{X}$-tuple that is not covered by any $\mathbf{X}$-tuple in the same group, we get a result-tuple as well.

Example 5. Figure 2.3 shows the normalization of $\mathbf{X}$ and the attribute Mgr is used for grouping. For instance, the first result tuple, (Xavier, Amber, [2012/1,2012/7)), and the the second result tuple, (Xavier, Amber, [2012/7,2012/9)) are derived from x_1 which is splitted by the start point of x_3. The third result tuple, (Xavier, Chelsea, [2012/7,2012/9)), and the fourth result tuple, (Xavier, Chelsea, [2012/9,2012/12)) are derived from x_3 which is splitted by the end point of x_1.

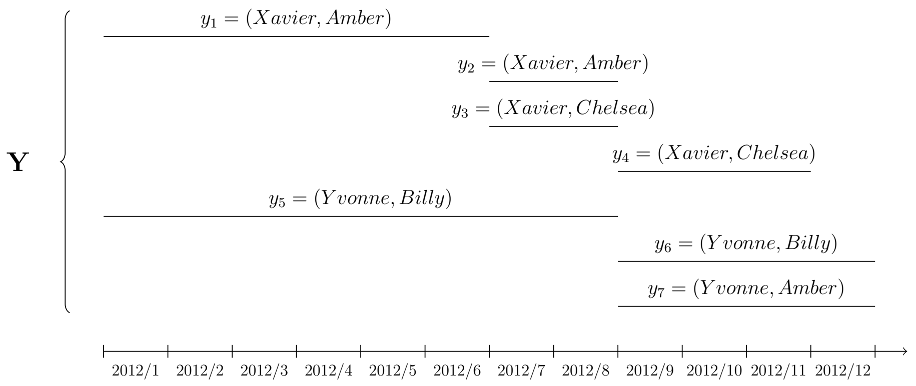


Figure 2.3: Temporal Normalization

After that we have accomplished the normalization, we can perform the aggregation over the previously gained relation $\mathbf{Y}$. Figure 2.4 shows the result of ${}_{Mgr,T}\vartheta_{count(*)}(\mathbf{Y})$ which is equal to the assumed result of the query $Q^T = {}_{Mgr}\vartheta^T_{count(*)}(\mathbf{M} \bowtie^T_{\mathbf{M}.Dept=\mathbf{E}.Dept} \mathbf{E})$ shown in figure 1.3:

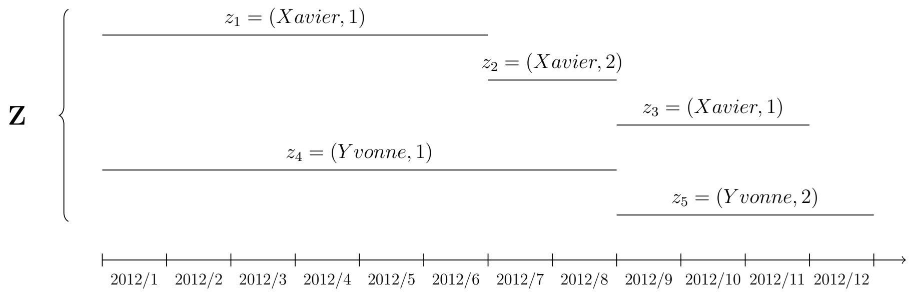


Figure 2.4: Aggregation

3 The SQL mapping

3.1 Implementation of the Temporal Primitives

Dignös et al. [1] have described the implementation of the temporal primitives in the kernel of the PostgreSQL database system. A very straightforward approach has been used so that, if we know how to write the relational algebra expression of a desired query, we won't have any problems to formulate the query in SQL. Table 3.1 shows the SQL syntax for the temporal primitives. Instead of the symbol for the temporal alignment operator Φ, we use the keyword `ALIGN` and the theta-condition θ has to be written by starting with the keyword `ON` followed by the actual condition. In order to use the normalization operator $\mathcal{N}$, we have to write `NORMALIZE` and we can define the group-attribute by writing it inside of the brackets of `USING()`.

$\mathbf{M}\Phi_{\mathbf{M}.Dept=\mathbf{E}.Dept}\mathbf{E}$	$\mathcal{N}_{Mgr}(\mathbf{X};\mathbf{X})$
`SELECT *`	`SELECT *`
`FROM` **`(M ALIGN E ON M.Dept = E.Dept)`** `m`	`FROM` **`(X NORMALIZE X USING(Mgr))`** `Y`

Table 3.1: Implementation of Temporal Primitives

The complete formulation of our query Q^T_{ra} as SQL-query with temporal primitives would approximately look like:

```
WITH X AS (
    SELECT ABSORB Mgr, Name, M.Ts, M.Te
    FROM (M ALIGN E ON M.Dept = E.Dept) M
        JOIN
        (E ALIGN M ON M.Dept = E.Dept) E
        ON M.Dept = E.Dept AND M.Ts=E.Ts AND M.Te=E.Te
)
SELECT Mgr, count(*), Ts, Te
FROM (X NORMALIZE X USING(Mgr)) Y
GROUP BY Mgr, Ts, Te;
```

3.2 The Mapping of Queries with Statement Modifiers to Queries with Temporal Primitives

So up to now, we know the reduction-rules, the temporal primitives, the relational algebra representations as well as the PostgreSQL representations of the primitives and how they work. What we don't know and what the actual problem is, is how we can achieve to map a query with statement modifiers to a query with temporal primitives. Without statement modifiers and the mapping, every time we desire to execute some temporal queries, we would be forced to manually formulate complex queries with temporal primitives:

```
WITH X AS (
    SELECT ABSORB Mgr, Name, M.Ts, M.Te
    FROM (M ALIGN E ON M.Dept = E.Dept) M
        JOIN
        (E ALIGN M ON M.Dept = E.Dept) E
        ON M.Dept = E.Dept
            AND M.Ts=E.Ts AND M.Te=E.Te
)
SELECT Mgr, count(*), Ts, Te
FROM (X NORMALIZE X USING(Mgr)) Y
GROUP BY Mgr, Ts, Te;
```

```
SEQ VT
SELECT Mgr, count(*)
FROM M JOIN E ON M.Dept = E.Dept
GROUP BY Mgr
```

Figure 3.1: Actual Reality (l.) and Wishful Thinking (r.).

This section investigates the grammar of the PostgreSQL parser (src/backend/parser/gram.y[1]) and how our desired mapping could be achieved. Since one of our goals is to implement statement modifiers, we have to modify the parser grammar to make it work. For now, we are interested at statement modifiers for select statements. The probably most important grammar rule for select statements can be found at the definition of the `simple_select` category. This is also the place where we have to make our first modifications to the grammar. Figure 3.2 shows the original-code on the left side and on the right side the grammar after we did some modifications. First, we have to add the non-terminal `modifiers` at the beginning of the rule. Whatever this non-terminal returns, will be passed to the argument \$1 which will be stored in the variable `n->sequencedValidTimeFlag` of the `SelectStmt`-node. Because of the non-terminal `modifiers` we've just added, the rank of the other non-terminals has changed and so we have to adjust the argument-variables accordingly. As next we have to define the category for the `modifiers` non-terminal and also its rules. One rule is that if the keyword `SEQ VT` has been used, it will return the value `TRUE` which will end up being saved into the variable `sequencedValidTimeFlag`. The other rule is for the case if we don't prepend the statement modifier to our SQL query, so that `FALE` will be returned.

Unfortunately we can't realize more of the mapping inside the PostgreSQL grammar. The rest will happen after the parser has created the parse tree. Figure 3.3 shows the parse tree, whereas $\theta \equiv (\mathbf{M}.\mathit{Dept} = \mathbf{E}.\mathit{Dept})$. Remember that the parse tree is a `SelectStmt` which stores in its modified version whether the flag `sequencedValidTimeFlag` is set and if so, the parse tree should be processed regarding to the sequenced semantics. If the flag is set,

[1]http://www.ifi.uzh.ch/dbtg/research/align.html

```
simple_select:

    SELECT opt_distinct opt_absorb
    target_list into_clause from_clause
    where_clause group_clause having_clause
    window_clause
    {
        SelectStmt *n = makeNode(SelectStmt);

        n->distinctClause = $2;
        n->absorbClause = $3;
        n->targetList = $4;
        n->intoClause = $5;
        n->fromClause = $6;
        n->whereClause = $7;
        n->groupClause = $8;
        n->havingClause = $9;
        n->windowClause = $10;
        $$ = (Node *)n;
    }

    /* some other rules for simple_select */
;
```

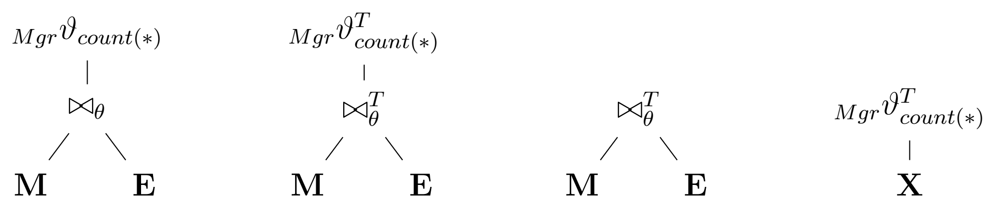
```
simple_select:
    modifiers
    SELECT opt_distinct opt_absorb
    target_list into_clause from_clause
    where_clause group_clause having_clause
    window_clause
    {
        SelectStmt *n = makeNode(SelectStmt);
        n->sequencedValidTimeFlag = $1;
        n->distinctClause = $3;
        n->absorbClause = $4;
        n->targetList = $5;
        n->intoClause = $6;
        n->fromClause = $7;
        n->whereClause = $8;
        n->groupClause = $9;
        n->havingClause = $10;
        n->windowClause = $11;
        $$ = (Node *)n;
    }

    /* some other rules for simple_select */
;

modifiers:
    SEQ VT      { $$ = TRUE; }
    | /*EMPTY*/ { $$ = FALE; }
;
```

Figure 3.2: Modifications of the PostgreSQL Grammar.

we can mark the parse tree accordingly by replacing the relational algebra operators $\bowtie$ and ϑ in the parse tree by their temporal representations $\bowtie^T$ and ϑ^T, which is shown in figure 3.3(b).

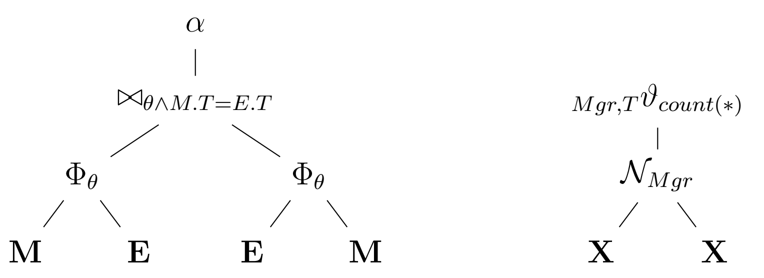

$_{Mgr}\vartheta_{count(*)}$
$\bowtie_\theta$
M
E
$_{Mgr}\vartheta^T_{count(*)}$
$\bowtie^T_\theta$
M
E
$\bowtie^T_\theta$
M
E
$_{Mgr}\vartheta^T_{count(*)}$
X

(a) Parse Tree (b) Temp. P.T. (c) Temp. Join (d) Temp. Agg.

Figure 3.3: Mapping Parse Trees

For better visualization purposes, we split the temporal parse tree into two pieces, at which the temporal join in figure 3.3(c) is shown as node $\mathbf{X}$ in the temporal aggregation in figure 3.3(d).

Now we proceed with the mapping by reducing the operators with sequenced semantics to their nontemporal counterparts. Hence the temporal join is processed before the temporal aggregation, we reduce the temporal join as first, which is shown in figure 3.4(a). As we know it from the reduction rules from table 2.1, the argument relations $\mathbf{M}$ and $\mathbf{E}$ must be both aligned. Keep in mind that θ still equates to $(\mathbf{M}.\mathit{Dept} = \mathbf{E}.\mathit{Dept})$. The condition for the nontemporal inner join equates to $(\mathbf{M}.\mathit{Dept} = \mathbf{E}.\mathit{Dept} \wedge \mathbf{M}.T = \mathbf{E}.T)$. The next step is that we reduce the temporal aggregation as shown in figure 3.4(b). The reduction demands to perform the normalization by taking the previously reduced temporal join $\mathbf{X}$ as both input-arguments of the normalization operator. Also mind that the normalization has to processed with respect to the group-attribute Mgr and that the grouping for the aggregation has to happen with respect to Mgr as well as to T. After that, we have completed the mapping of queries with statement modifiers to queries with temporal primitives over algebra expressions.


α
$\bowtie_{\theta \wedge M.T=E.T}$
Φ_θ
Φ_θ
$\mathbf{M}$ $\mathbf{E}$ $\mathbf{E}$ $\mathbf{M}$
${}_{Mgr,T}\vartheta_{count(*)}$
$\mathcal{N}_{Mgr}$
$\mathbf{X}$ $\mathbf{X}$

(a) Reduced Temporal Join

(b) Reduced Temporal Aggregation

Figure 3.4: Reduction on Temporal Operators

We know that, an ordinary select-statement is processed in the order shown in figure 3.5(a). This makes it possible to determine a general case for the order of reduction rules that have to be performed, which is shown in figure 3.5(b). So first of all, we have to do the reduction on all the joins and/or cartesian products. As second step, the `WHERE`-clause takes place, but nothing special happens here since a temporal selection is processed the same as a usual selection. As next we have to apply the reduction rules on aggregates. After that the `HAVING`-clause has to be handled, but since its behaviour does no differ from a general selection, there is also no special reduction necessary. And in the end, the `SELECT`-clause has to be handled by applying the reduction rules of a temporal-projection.

$$\begin{tikzcd}
\text{from items} \arrow[r] & \text{from} \arrow[r, "\text{table}"'] & \text{where} \arrow[r, "\text{subset of table}"'] & \text{group} \arrow[r, "\text{grouped table}"'] & \text{having} \arrow[r, "\text{subset of groups}"'] & \text{select} \arrow[r] & \text{subset of groups}
\end{tikzcd}$$

(a) General Processing Order

$$\texttt{FROM}\left\{\begin{array}{ccc}
r \times_{\theta}^{T} s & = & \alpha((r\Phi_{true}s) \times_{r.T=s.T} (s\Phi_{true}r)) \\
r \bowtie_{\theta}^{T} s & = & \alpha((r\Phi_{\theta}s) \bowtie_{\theta\wedge r.T=s.T} (s\Phi_{\theta}r)) \\
r \mathbin{⟕}_{\theta}^{T} s & = & \alpha((r\Phi_{\theta}s) \mathbin{⟕}_{\theta\wedge r.T=s.T} (s\Phi_{\theta}r)) \\
r \mathbin{⟖}_{\theta}^{T} s & = & \alpha((r\Phi_{\theta}s) \mathbin{⟖}_{\theta\wedge r.T=s.T} (s\Phi_{\theta}r)) \\
r \mathbin{⟗}_{\theta}^{T} s & = & \alpha((r\Phi_{\theta}s) \mathbin{⟗}_{\theta\wedge r.T=s.T} (s\Phi_{\theta}r)) \\
r \triangleright_{\theta}^{T} s & = & (r\Phi_{\theta}s) \triangleright_{\theta\wedge r.T=s.T} (s\Phi_{\theta}r)
\end{array}\right.$$

$$\begin{array}{ccc} \downarrow & & \downarrow \end{array}$$

$$\texttt{WHERE}\left\{\begin{array}{ccc} \sigma_{\theta}^{T}(r) & = & \sigma_{\theta}(r) \end{array}\right.$$

$$\begin{array}{ccc} \downarrow & & \downarrow \end{array}$$

$$\texttt{GROUP}\left\{\begin{array}{ccc} {}_{B}\vartheta_{F}^{T}(r) & = & {}_{B,T}\vartheta_{F}(\mathcal{N}_{A}(r;r)) \end{array}\right.$$

$$\begin{array}{ccc} \downarrow & & \downarrow \end{array}$$

$$\texttt{HAVING}\left\{\begin{array}{ccc} \sigma_{\theta}^{T}(r) & = & \sigma_{\theta}(r) \end{array}\right.$$

$$\begin{array}{ccc} \downarrow & & \downarrow \end{array}$$

$$\texttt{SELECT}\left\{\begin{array}{ccc} \pi_{B}^{T}(r) & = & \pi_{B,T}(\mathcal{N}_{B}(r;r)) \end{array}\right.$$

(b) General Order of the Temporal Operators / Reduction Rules

Figure 3.5: General Case

4 Conclusion

In this report we have looked at reduction rules that are using temporal primitives and thereby define a temporal algebra with sequenced semantics. We have used this reduction rules to define a SQL mapping that translates queries with statement modifiers to queries with temporal primitives. As part of the mapping we have shown in what way the PostgreSQL parser can be used to implement statement modifiers. The statement modifiers in turn can be used to flag select-statements/parse-trees, so that our desired mapping is possible after the creation of the flagged parse tree. In the end, we have illustrated the general case of our SQL mapping by opposing the algebra expressions with the processing order of typical SQL queries.

Bibliography

[1] A. Dignös, M. H. Böhlen, and J. Gamper. Temporal alignment. In Proceedings of the 2012 international conference on Management of Data, SIGMOD '12, pages 433–444. ACM, 2012.

[2] M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, 2000.

[3] H. Garcia-Molina, J. D. Ullman, and J. Widom. Parsing. In Database systems - the complete book (international edition), chapter 16.1, pages 788-795. Pearson Education, 2002.