Zürich, 20 March 2017

MSc Project: Integrating Generally Valid Relational Algebra Operators for Ongoing Relations into PostgreSQL

Ongoing time points, such as now are common in relations that include a valid-time attribute to denote that a tuple is valid until the present time.

As ongoing time points change as time passes by, the results of queries do so as well. To evaluate queries to results that remain valid, we proposed general validity to calculate their result and ongoing relations to represent the result. A generally valid query is evaluated for all reference times and thus, independent of a specific time. An ongoing relation contains a reference-times attribute that states at which reference times a tuple is observable in the relation.

![Diagram showing ongoing input relations and query result remaining valid as time passes by.](image)

Figure 1: Bug tracking as application scenario for ongoing time points.
An application scenario for ongoing time points is bug tracking. For instance, a company keeps track of bugs associated with their components like spam filter and mail storage. A prioritized bug has a fixed start date indicating its discovery and a fixed end date indicating the deadline for resolving it. A deprioritized bug has also a fixed start date, but an end date that keeps increasing until the bug is resolved. The end date is ongoing. In Figure 1, relation B contains an excerpt of the bugs associated with the spam filter component. For instance, tuple $b_1$ refers to a prioritized bug that was discovered on 17/02/01 and must be resolved until 17/07/01. Tuple $b_2$ refers to a deprioritized bug that persists from 17/02/01 onwards. Relation RS contains an excerpt of the releases scheduled for the component. To determine the bugs that are affecting past, present, and future releases since the bug's discovery together with the id of the affected releases, query $Q : \text{Res} \leftarrow \pi_{C,RID,BID,BVT}(B \bowtie \leftarrow \bowtie C = CID \land BVT \text{ overlaps } RVT \text{ RS})$ is stated. In the result, tuple $res_1$ records that release 300 of the spam filter component is affected by bug 500 and this is observable for all reference times. Tuple $res_2$ records that release 300 of the spam filter component is affected by bug 501 and this is observable from reference time 17/02/06 onwards. The restriction of the reference times results from the join predicate being $true$ at some reference times (thus, the tuple is observable at these times) and $false$ at others (thus, the tuple is not observable at these times).

When applying operators like projection, selection, and join to ongoing relations, the input tuples’ value of the reference-times attribute and the operators’ predicates determine the output tuple’s value of the reference-times attribute. The goal of this project is the integration of the basic operators of the relational algebra into the kernel of the widely-used, open-source database system PostgreSQL, so that the operators evaluate on ongoing relations to results that remain valid as time passes by.

Tasks

1. Literature study on
   - ongoing time points, especially the basic ongoing time point now [1]: the definition of now, application scenarios for ongoing time points and the advantages of ongoing time points compared to fixed time points, and how ongoing time points affect query evaluation.
   - ongoing relations [3]: purpose of the reference-times attribute $RT$
   - generally valid relational algebra operators for ongoing relations [3]: what a generally valid operator is and how the definitions for generally valid operators differ from the well-known operators for relations without ongoing time points.

2. Study [4, 2] and the source code of PostgreSQL to understand the workflow of parsing, analyzing, optimizing, and executing a query.

3. Integrate the generally valid operators, projection $\pi_B(R)$, selection $\sigma_B(R)$, and inner join $R \bowtie S$ defined in [3] into the kernel of PostgreSQL:
   - Extend the parser to accept the operators mentioned above. Use the keyword ongoing to distinguish these operators from the existing ones, e.g., ONGOING JOIN to distinguish from JOIN.
Extend the analyzer and optimizer to integrate the operators into a query plan tree. Use the existing operators to express the operator for ongoing relations in order to be able to exploit the existing optimization strategies (e.g., join strategies).

Extend the executor if necessary.

**Optional:** Provide the same support for the set operators, union $R \cup S$, intersection $R \cap S$, and difference $R - S$.

4. Generate synthetic datasets that consist of ongoing relations with at least 100,000 tuples.
5. Evaluate your implementation with a set of reference queries on the synthetic datasets.

References


**Supervisor:** Yvonne Mülle (muelle@ifi.uzh.ch)

**Start date:**

**End date:**

University of Zurich
Department of Informatics

Prof. Dr. Michael Böhlen
Professor