Applying Patterns and SQL Idioms to Optimise Relational Database Queries

Dissertation submitted in partial fulfilment
of the requirements for the Open University's
Master of Science Degree
in Software Development.

Walter Weinmann
(U7637432)

25 February 2007

Word Count: **14,973**

Preface

I would like to thank all those who supported and encouraged me during this dissertation, especially Mutti for her invaluable help in proofreading the text, much of which she claimed to have no understanding of whatsoever.

Special thanks go to my beloved Jane who prompted me to start my studies with the Open University and who has accompanied me on my journey all these years providing endless encouragement, comprehension and support (including a constant supply of my favourite cake).

This dissertation is dedicated to the memory of my parents who would have been very proud of me.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
AST	Abstract Syntax Tree
CBO	Cost-Based Optimiser
QGM	Query Graph Model
RDBMS	Relational Database Management System
REPSI	Recording the Efficiency of Patterns / SQL Idioms
TQP_88899	Test query pair number 99 of pattern 888 (usually related to Appendix B and Appendix C)

List of Conventions

`Courier`	Courier font is used to represent commands, source code, SQL key words or table names
`{…}`	Mandatory elements are enclosed by curly brackets
`[…]`	Optional elements are enclosed by squared brackets
____	Default values are underlined.
`|`	Choices are separated by vertical lines
…	Repeating elements are expressed by three consecutive dots

Abstract

Retrieving data from a relational database by formulating and executing queries is one of the main features of the relational database language SQL. Because SQL is a declarative language, an SQL query only describes the properties of the data to be retrieved, relinquishing the choice of processing method to the query optimiser of the RDBMS (Relational Database Management System). This dissertation, therefore, aimed to establish if it is possible to increase the efficiency of existing relational database query optimisers by applying suitable patterns and SQL idioms when formulating the relational database query.

The patterns and SQL idioms under examination were developed based on literature research related to topics such as optimisation, performance, and tuning in connection with databases and SQL. SQL queries particularly designed to demonstrate the effectiveness of the revealed patterns and SQL idioms were then tested both with, and without, the application of the pattern or SQL idiom using the most current Oracle RDBMS (Oracle® Database 10g Release 2). These tests were supported by a bespoke tool named REPSI (Recording the Efficiency of Patterns / SQL Idioms) (Weinmann, 2006) which measures and records the response times of pairs of queries. The evaluation of the recorded measurements was based on two methods. The first involved the usage of statistical tools; box plots and dot charts (both derived from the measured response times). The second employed Oracle specific tools (EXPLAIN PLAN, SQL TRACE, and TKPROF) which enabled the examination of the applied database access plans and the accumulated SQL trace data.

The results demonstrate that the query optimiser is not always able to determine the processing method which retrieves the requested data in the shortest response time. In fact, the response time of certain SQL queries can sometimes be significantly improved simply by reformulating the SQL query based on the application of suitable patterns and SQL idioms.

Chapter 1 Introduction

This chapter provides a brief overview of the basic concepts of the relational database language SQL and the interconnection between SQL, query optimisation and query processing. This is followed by a discussion of conceptual deficiencies. Finally the research question is presented including the complete aims and objectives of this dissertation. The tool used to support the empirical analysis as part of the research is also outlined.

The rest of this document proceeds as follows:

- Chapter 2: Introduces patterns and their application in software development.
- Chapter 3: Presents an overview of existing work related to SQL in connection with optimisation.
- Chapter 4: Describes the applied experimental environment and the chosen research methods.
- Chapter 5 and 6: Present the results of this dissertation. First the defined patterns and SQL idioms are presented broken down by the basic concepts according to the various SQL query clauses and the more advanced query concepts subqueries and joins. Then the results of applying these patterns are discussed.

- Chapter 7: Analyses the revealed patterns and the results of their application. It also critically discusses the chosen research methods and tools. A set of guidelines related to the application of the patterns within the Oracle® Database are then provided.

- Chapter 8: Provides the overall conclusions, explores the implications of these and suggests further areas of work to be explored.

The applied database schema, detailed measurement results, references, and index can be found in the appendices.

1.1 Relational Database Language SQL

Despite the relatively short history of computing, relational databases have featured for a significant part of this time (Ullman, 1987) and, 35 years after the seminal paper of Codd (Codd, 1970), they are still the widely-used technology to store structured data (Gartner, 2002). One of the reasons for this popularity is the relatively deceptive appearance of a relational database system. At first glance it appears to consist of easily understandable views of tables, rows and columns. Beneath this, however, lies a strict mathematical foundation based on relational calculus. Similarly, the most popular relational database language, SQL, also appears relatively simple and can easily be used by a layperson, at least for simple queries. In reality, SQL is a syntactic version of the tuple relation calculus (Dietrich, 2001) and comprises, in its latest standardised version (ISO/IEC 9075, 2003), a document of 1245 pages (ISO/IEC JTC 1/SC 32/WG 3, 2003) describing only its foundations (volume number 2 of total 9 volumes).

A database query expressed in SQL is a high-level interface to the data to be retrieved (Chaudhuri, 1998) and only describes the conditions that the final result must satisfy not how to achieve it (Freytag, 1987). Ideally this enables the database user to focus on clarity, understandability and maintainability when formulating queries without being forced to consider potential efficiency issues that could arise during the execution of the query.

1.2 Query Optimisation and Query Processing

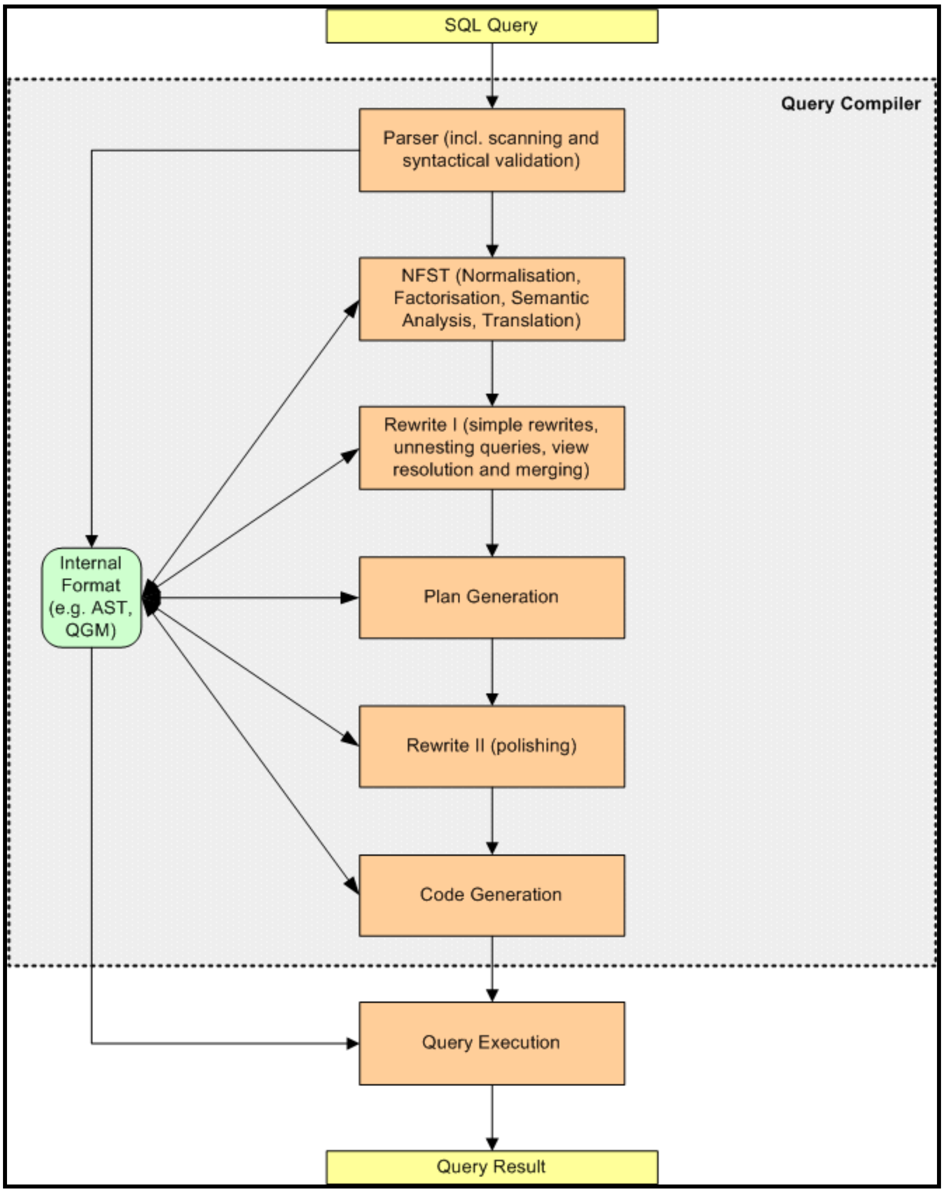


Figure 1: Typical steps in query compiling and processing (adapted from (Elmasri and Navathe, 2007, IBM, 2004, Moerkotte, 2006))

A group of components, collectively called a query processor and usually including a query compiler (see Figure 1), fills the gap between database query languages and data stored in file systems (Graefe, 1993). The query processor achieves this by translating these declarative database queries into a so called query evaluation plan. This plan contains a sequence of operations, which retrieve the required data from the underlying operating system files.

In the area of query processing, however, no common denominator exists - not to mention any standardisation of any of its particulars, for example:

- From an architectural point of view the query can be processed in an interpretation mode (Widom, 1996) or in a compiler mode (Chamberlin et al., 1981). The optimiser can even be generated specific to a schema as in the Exodus (Graefe and DeWitt, 1987) and Volcano (Graefe and McKenna, 1993) optimisers.

- The modularisation structure of a query compiler may vary from a monolithic, as in System R, to a dedicated object oriented model as with Apache Derby (Saunders and Anderson, 2006).

- For communication between the different modules a multiplicity of internal representations are used. A few examples are abstract syntax trees (AST), expression trees as in Cascades (Graefe, 1995), operator trees (Graefe and DeWitt, 1987, Smith and Chang, 1975, Warshaw and Miranker, 1999) or the query graph model (QGM) in Starburst (Haas et al., 1989).

Another difficulty in query optimisation is the growing variety in optimisation strategies. These are mostly based on sophisticated theoretical concepts like relational algebra, domain relational or tuple relational calculus (Dietrich, 2001), extended three-valued predicate calculus (Negri et al., 1991), combinatory-based algebras (Cherniack and Zdonik, 1996), dynamic programming (Kossmann and Stocker, 2000), and others. In addition, the optimisation strategies often employ a large number of different optimisation techniques each forced to consider all the various subtleties of the SQL query language. For example:

- Adaptive query optimisation, a mix of optimisation and execution (Avnur and Hellerstein, 2000).

- Heuristic techniques which are intuitively right, but do not guarantee optimal execution (e.g. cross products executed as late as possible, pushdown of projection and selection, or reduction of intermediate results) (Elmasri and Navathe, 2007).

- Join ordering based on deterministic or randomised algorithms or a combination of both, where joins include the various types like equi-join, full join, inner join, natural join, outer join, and so on (Kossmann and Stocker, 2000).

- Multiple query optimisation which may be suboptimal for an individual query (Roy et al., 2000).

- Rewriting the query by eliminating redundant joins, unnesting queries, merging views (Goldstein and Larson, 2001, Celis and Zeller, 1997, Bhargava et al., 1995).

- Rule based techniques which separate search algorithm and search space (Freytag, 1987, Graefe and DeWitt, 1987).

- Semantic query optimisation based on constraints defined in the database schema (Chakravarthy et al., 1990).

The preceding details (although far from complete) demonstrate the complexity of query optimisation. Yet despite this, none of the strategies and methods developed so far is able to guarantee an optimal processing of every type of SQL query. Even database researchers are unhappy with the state of query optimisation:

- (Jarke and Koch, 1984): "Exact optimization of query evaluation procedures is in general computationally intractable and is hampered further by the lack of precise statistical information about the database."

- (Ioannidis, 1996): "Despite all the work that has been done on query optimization, in every single module of the architecture […], there are many questions for which we do not have complete answers, even for the simplest, single-query, relational optimizations."

- (Chu et al., 1999): "In practice, things are not so simple. There are two major problems: (1) there are far too many possible plans for an optimizer to consider them all, and (2) accurate cost estimation is virtually impossible, since it requires detailed a priori knowledge of the nature of the data and the run-time environment."

- (Winslett, 2002): "I think query optimization is a huge hole;" and "Query optimization is 22 years old at this point. Everybody does exactly the same thing, all based on work that Pat Selinger and the System R team did, and it doesn't work [any more]."

If one considers computer science to be a part of information science (which according to Moody is “an applied rather than a pure discipline” (Moody, 2000)) then there appears to be a significant gap in the transfer of knowledge from database researchers to SQL users in supporting the formulation of optimal SQL queries. The majority of strategies and methods for query optimisation developed by database researchers are hard to understand and difficult to apply without a deep knowledge of certain theoretical concepts like grammars, languages, predicate logic, or relational algebra. There also seems to be insufficient knowledge to support the development of query optimizers which always achieve a high-performance level.

This dilemma leaves the SQL user with a good proportion of the responsibility for optimisation considerations when formulating a database query despite the claim of the SQL language that the declarative approach should suffice. The user has no practical guide or tool to help him to formulate a query that can be processed in the most efficient way.

1.3 Emergence of Patterns

Based on the work of Christopher Alexander (Lea, 1994) about patterns in architecture, many disciplines of software engineering, (e.g. software architecture, analysis, design, implementation, human computer interaction and so on) adapted the methodology of describing their “best practices” in the standardised format of patterns. As a minimum this pattern format consisted of sections such as context, recurring problem and solution (Derntl, 2003). Patterns are an important contribution to the work of practitioners in the sense of Moody who argues that “to be effective, research must be both (1) relevant to the needs of practice and (2) disseminated and used by practitioners”.

1.4 Aims and Objectives

Surprisingly no explicit idioms or patterns for writing relational database queries in SQL to achieve an efficient execution can be found in the database literature. Indeed, despite the complexity of contemporary relational database query optimisers none of the strategies and methods applied so far are able to guarantee an optimal processing of every type of SQL query. Applying an appropriate set of patterns during the development of an SQL database query might, therefore, be a way to improve the efficient execution of SQL queries.

This dissertation seeks to explore this idea by uncovering such idioms and patterns mainly based on appropriate literature like conference and journal articles, and textbooks. These idioms and patterns will make a contribution to transfer the scientific knowledge of query optimisation - often understandable only for highly specialised academics - to practitioners who should be able to apply such knowledge in their practical work. Patterns not only capture experience and knowledge ("communicate wisdom and insight" (Meszaros and Doble, 2005)), but also contribute to understanding, especially when reducing complexity by decomposing complex structures (Derntl, 2003).

Therefore, the intended research question reads as follows:

> Is it possible to increase the efficiency of existing relational database query optimisers by applying suitable patterns and SQL idioms when formulating the relational database query?

To address this question this dissertation aimed to achieve the following objectives:

- To investigate current recommendations for improving the efficiency in executing SQL queries in research and industry.
- To identify those recommendations which could be used as a basis for performance improving patterns and SQL idioms.
- To write and evaluate a selection of patterns and SQL idioms.
- To provide a set of guidelines that the SQL user can follow with the intention of improving the efficiency in the execution of SQL queries in addition to the efforts of the database query optimiser.

In addition this dissertation included the development of a tool named REPSI (Recording the Efficiency of Patterns / SQL Idioms) (Weinmann, 2006) which is able to generate an appropriate schema and instance as a basis to run SQL queries both with, and without, the application of the patterns and idioms. This tool was developed with the following objectives in mind:

- To measure and record the response times of the queries to allow an estimation of whether the pattern or SQL idiom is used by the database optimiser.
- Based on the results of running the tool with a specific database, to provide the SQL user with a guide to improve the efficiency of the specific database optimiser by the application of patterns and SQL idioms.

The tool was applied to the RDBMS Oracle® Database 10g Release 2 to demonstrate one selected relational database system and analyse the results.

1.5 Limitations

The optimisation of SQL queries in this dissertation is limited to improving the response time of SQL queries by simply reformulating the query. Other approaches for optimisation e.g. improving the hardware, adjusting configuration parameters, or database vendor specific tools are not considered. Further, the experiments performed are restricted to centralised databases only with database client and database server residing on the same machine. The SQL features considered are chosen from an intersection of the SQL:2003 standard (ISO/IEC JTC 1/SC 32/WG 3, 2003) and the Oracle® Database 10g SQL language (Lorentz, 2005).

1.6 Summary

This chapter has covered the presentation of the research question including the complete aims and objectives of this dissertation. It also outlined a tool designed to support the empirical analysis as part of the research. The next chapter will discuss the role of patterns in software development in general and the application of the pattern format in this dissertation in particular.

Chapter 2 The Application of Patterns in Software Development

After a short treatment of the origins of patterns this chapter discusses the pros and cons of pattern application in software development. Subsequently the motivation for considering patterns as a framework for reformulating queries to improve their performance is presented. Finally the chosen pattern format used in this dissertation is described.

2.1 From Architecture to Architectural Patterns

In the late 1970's, Christopher Alexander, Professor of Architecture at the University of California, Berkeley, introduced the concept of patterns in architecture. The main aspects of his research are described in the following two books:

- "A pattern Language – Towns, Buildings, Construction" (Alexander et al., 1977). This is essentially a collection of patterns to build any aspects of architectural objects, and

- "A Timeless Way of Building" (Alexander, 1979) which tries to develop a language based on these patterns to enable people to build quality architectural objects of any complexity through the application of the generative power of this pattern language.

Alexander (Alexander et al., 1977) writes his patterns in a uniform format comprising the following parts:

- **Pattern name**: characterisation of the pattern by a catch phrase and an appropriate picture,
- **Context**: describing a recurring set of situations in which the pattern can be applied and the relationship to other patterns,
- **Problem**: consisting of three diamonds to mark the beginning of the pattern's core, a short description of the problem in bold, and a more detailed problem discussion ending with 'therefore' to introduce the solution.
- **Solution**: outlining the core of the solution in a short bold paragraph, if necessary followed by a more detailed description of the solution along with an illustrative diagram, and again completed with three diamonds,
- **Examples**: examples and secondary patterns may conclude the pattern.

2.2 From Architectural Patterns to Software Patterns

In 1987, Cunningham and Beck (Cunningham and Beck, 1987) experimented with Alexander's pattern approach in the user interface design of a project (Appleton, 2000). They presented their experiences in the same year during a workshop at the OOPSLA conference in Orlando. The next important step was Coplien's book "Advanced C++ programming styles and patterns" (Coplien, 1992) where he described C++ programming patterns which he called idioms. In contrast to the more general approach of patterns, idioms (or programming patterns) cover problems and solutions strongly related to specific programming language.

The first peak in the pattern movement in computing was reached with the publication of the seminal book "Design patterns : elements of reusable object-oriented software" (Gamma et al., 1995) by Gamma, Helm, Johnson, and Vlissides (often called the Gang Of Four). This book contained an extensive collection of patterns for object-oriented design illustrated by OMT diagrams and C++ / Smalltalk code. The book initiated broad support for the idea of using patterns to record practical experience and to communicate this knowledge to others in many aspects of the software engineering process. For example:

- Requirements specification (Ferdinandi, 2002),
- Software analysis (Fowler, 1997),
- Software architecture (Buschmann et al., 1996),
- Software design (Larman, 2002, Grand, 2001),
- User interface design (Tidwell, 2006),
- Programming (Beck, 1997, Langr, 2000),
- Database access (Nock, 2004), and many more.

The main motivations for applying patterns are the flexible utilisation of experience and the benefits of the unique form of communicating this knowledge. Patterns don't deliver new solutions but rather offer distilled experience (Appleton, 2000). They exhibit "best practices" or "lessons learned" intended to solve known recurring problems in many aspects of software engineering from analysis and design to coding and testing. Patterns don't dictate a black or white solution, but instead provide suggestions including trade-offs and compromises.

The underlying core schema of a pattern consists of the pattern name, the context describing the problem environment, the problem itself, and a proven solution to the problem. Though the schema may be extended by other elements like forces, resulting context, rationale, or known uses (Gamma et al., 1995), the core schema is the base enabling the communication of the solution from experts to the more inexperienced via a standard vocabulary (Cline, 1996, Coplien, 1997).

There are also criticisms of pattern application. Wiki Wiki Web (Wiki Wiki Web, 2006) claims, for example, that the need for design pattern results merely from a lack of appropriate functionality in the programming languages. The same source points out that the application of patterns leads inevitably to duplication of code, which violates one of the base principles of modern software development techniques (Fowler and Beck, 1999).

Component reuse, another important software development principle, is also incompatible to the approach of patterns. Bertrand Meyer suggests creating components covering the functionality of patterns to overcome the latter issue and describes, as examples, component solutions for the factory pattern (Arnout and Meyer, 2004) and for the visitor pattern (Meyer and Arnout, 2006).

2.3 From Software Patterns to the Application of the Patterns in this Dissertation

The main reasons for considering patterns as a framework for reformulating queries to improve their performance include:

- The inherent inclusion of discussing alternatives in contrast to pure black and white approaches.

- The provision for identifying and specifying a higher level of abstraction than the discussion on a pure syntax level.

- The promotion of encapsulating well-defined performance optimisation problems and their potential solutions as autonomous entities (Lea, 1994).

- The higher degree in recognising and memorising patterns which is triggered by the pattern name and the uniform structure as appose to style guide like descriptions (Langr, 2000).

- The creation of a common vocabulary and understanding of performance optimisation problems (Buschmann et al., 1996).

Most of the patterns used in this dissertation are based on a very specific SQL coding level and, therefore, should be called SQL idioms. For purposes of simplification the rest of the paper will, however, not differentiate between patterns and idioms and rather call them all patterns.

Name	Defensive Projection
Problem	How can choices related to the projection (choice of columns) contribute to improved performance?
Potential Solutions	(a) Be specific and avoid the asterisk in the `SELECT` clause. (b) If possible restrict the columns in the `SELECT` clause to the indexed ones.
Related	Index Exploitation, Minimise Sorting Columns
References	(Celko, 2005a, Powell, 2007, Shasha and Bonnet, 2003)

Figure 2: Example of the chosen pattern structure

The chosen pattern format (see Figure 2) for this research is relatively simple as only a few of the theoretically possible pattern elements are considered applicable to patterns very close to a programming language like SQL (Beck, 1997, Langr, 2000):

- **Name**: A short but precise and easy to memorise characterisation of the pattern.

- **Problem**: The problem stated as a question should be a clear indicator for the applicability of the pattern to a given context.
- **Potential Solution(s)**: Describes at least one, but usually several, potential solutions – typically development actions related to the implementation of SQL queries.
- **Related**: Cross reference to patterns with a potential relationship to the problem under investigation.
- **References**: Indication of sources which contributed to the content of the pattern or which facilitate gaining more knowledge to the problem.

A discussion of measurement results extracted from running a set of SQL queries related to the pattern in the Oracle® Database follows each pattern description in this paper.

2.4 Summary

This chapter has discussed the origins of patterns and their application in software development at large and in this dissertation in particular. The next chapter will present a survey of the existing literature especially related to the optimisation of SQL queries but also considering database performance optimisation and tuning.

Chapter 3 Related Work

This chapter presents literature related to topics such as optimisation, performance, and tuning in connection with database and SQL. The literature research is divided into the examination of academic textbooks related to database systems, and mainly non-academic books related to SQL optimisation especially in the context of both the SQL language and tuning. Finally an upcoming book covering SQL and patterns is mentioned. Despite an extensive search of the internet, information about SQL and patterns, beyond references to such books, was mainly related to pattern recognition and not to the optimisation or the tuning of SQL queries.

3.1 SQL Optimisation in Database Textbooks

SQL plays only a minor role in academic textbooks about database systems (e.g. "Foundations of Databases" (Abiteboul et al., 1995), "Fundamentals of Database Systems" (Elmasri and Navathe, 2007), "Database Systems : The Complete Book" (Garcia-Molina et al., 2002), "Database Management Systems" (Ramakrishnan and Gehrke, 2003), or "Database System Concepts" (Silberschatz, 2006)). The treatment of SQL language is usually restricted to an overview of important syntactical and semantic aspects. One of the most comprehensive treatises in the mentioned textbooks is given by Garcia-Moliana, who delineates SQL on 46 pages of 1119 total pages. This is also the only book discussing an optimisation aspect of SQL by recommending the deliberate use of the `DISTINCT` operator. In these textbooks query optimisation describes, in detail, the process of selecting a suitable execution strategy from the many possible strategies to process a given SQL query.

3.2 SQL Optimisation in SQL Related Textbooks

There are a lot of non-academic textbooks covering different aspects of the SQL language. Almost all those studied do not, however, consider performance issues. The types of books available (with examples) are:

- Introductory books to the SQL language, e.g. "SQL : Practical Guide for Developers" (Donahoo and Speegle, 2005),
- Practical solutions for applying the SQL language for specific problems, e.g. "SQL Cookbook" (Molinaro, 2005),
- Syntactical and semantic details of different SQL language variations, e.g. "SQL in a Nutshell" (Kline et al., 2004),
- Detailed explanations of the SQL standard language, e.g. "A Guide to the SQL Standard" (Date and Darwen, 1997) or " SQL:1999 : Understanding Relational Language Components" (Melton and Simon, 2002), or
- Recommended style guides for writing SQL statements, e.g. "SQL Programming Style" (Celko, 2005b).

"SQL for Smarties : Advanced SQL Programming" (Celko, 2005a) presents a more advanced coverage of the SQL language. In addition to some not so prominent presented optimisation recommendations (for example, with the predicates `EXISTS`, IN, or LIKE), Celko dedicates the whole last chapter of the book to the optimisation of SQL statements.

Following Sun Tzu's "Art of War" (Tao et al., 2000) based on an old Chinese book (6^{th} century BC) describing military strategies, the book "The Art of SQL" (Faroult and Robson, 2006) tries to adapt the ideas of war strategies to SQL performance by equating database design with military planning, indexing with tactical dispositions, monitoring performance with employment of spies and so on. Despite the originality of the approach, the book contributes less interesting optimising topics for SQL queries.

In O'Reilly's Hacks Series the book "SQL Hacks : Tips & Tools for Digging into Your Data" (Cumming and Russell, 2007) addresses the SQL language. Hacking here, is meant in the positive sense of a clever or quick solution to a data access problem to be solved applying SQL. Only 5 out of 100 of the presented hacks (hack numbers 10, 11, 51, 75, and 89) can be related to SQL performance in the sense of this dissertation despite a chapter called "Locking and Performance" including 13 hacks. The main intention of the book is to uncover unorthodox solutions or to solve problems with SQL which are usually solved by other means. A good example is hack number 24, which shows an elegant solution for the missing PRODUCT() function in SQL.

"Understanding Relational Database Query Languages" (Dietrich, 2001) provides a more prominent contribution to the optimisation of SQL queries. Based on an introduction to the relational data model and relational algebra, the core of the book presents three relational database query languages: domain relational calculus, tuple relational calculus and SQL. Both in the relational algebra chapter and in the SQL chapter there are sections called query optimisation which cover suggestions to formulate more efficient queries.

Nearly the whole first half of "SQL Performance Tuning" (Gulutzan and Pelzer, 2003) discusses tuning possibilities directly related to various SQL language features, like selection, sorting, grouping, joins, subqueries, columns, and tables. An interesting aspect is the review of database systems which support a certain tuning method and, of those, which ones actually improve their performance by applying this tuning method. The database systems evaluated were Borland Interbase® 6.0, IBM® DB2 Universal Database 7.2, IBM Informix® Dynamic Server 9.3, Ingres® II 2.5, Microsoft® SQL Server 2000, MySQL® 3.23, Oracle® Database 9i, and Sybase Adaptive Server® Enterprise 12.5.

3.3 SQL Optimisation in Tuning Related Books

The tuning of relational databases and SQL statements is covered by a broad range of books using various approaches. "Oracle High-Performance SQL Tuning " (Burleson, 2001), "Oracle SQL Tuning & CBO Internals" (Floss, 2004), "Oracle Tuning : The Definitive Guide" (Burleson and Gogala, 2005), or "Oracle Tuning in der Praxis : Rezepte und Anleitungen für Datenbankadministratoren und -entwickler" (Haas, 2005) mainly focus on the Oracle® Database. To a greater or lesser extent they address the following topics:

- Oracle's rule-based query optimiser (deprecated since version 10g (Oracle Corporation, 2005)) and cost-based query optimiser (CBO),
- Oracle specific performance tracing utilities like EXPLAIN PLAN (showing the execution plan of an SQL statement without executing it), SQL*Net diagnostics

(managing network problems), SQL_TRACE (creating files containing trace information of the SQL statement execution) and TKPROF (evaluating SQL_TRACE files), or STATSPACK (monitoring the behaviour of SQL in the library cache and creating automated alert reports),

- The concepts of hints (Oracle specific extension of SQL to direct the optimiser), outlines and SQL profiles (both execution plans stored in the data dictionary as a reinforcement of hints),
- The various possibilities for the optimisation of physical structures like memory and disk storage,
- The concept of parallel execution of queries (normally in connection with partitioning),
- The database parameters relevant for tuning purposes, and
- The implementation of high available databases considering hardware, backup and recovery, and applications.

This type of book mainly addresses the database administrator and covers in depth practical methods and different options to improve the performance of an Oracle® Database. Recommendations for the optimisation of SQL queries, simply by reformulating them, are only found on a small scale.

Based on an introduction into basic aspects of relational database like caching, indexes, single-table access paths, and joins, the book, "SQL Tuning" (Tow, 2003), provides an explanation on how to view, interpret, and control the execution plans of SQL statements in the relational database systems IBM DB2®, Microsoft® SQL

Server and Oracle® Database. The main part of this book covers the presentation of a graph based tool called query diagram and a corresponding process to tune SQL statements by manually determining a (near) optimal execution plan. The elements of the query diagrams (see Figure 3) are:

- Nodes representing the tables in the `FROM` clause,
- Directed edges standing for joins between the tables,
- Underlined numbers next to the nodes describing the fraction of rows of this table which satisfy the non-join predicates, and
- Non-underlined numbers next to the end of the edges representing the average number of rows matching the join predicates on the other end of the edge.

Query:

```
SELECT …
  FROM Employees E, Departments D
 WHERE E.Department_Id = D.Department_Id
   AND E.Exempt = 'Y'
   AND D.US_Based_Flag = 'Y';
```

Assumptions: 10% of the rows in table Employees have Exempt_Flag = 'Y',
50% of the rows in table Departments have US_Based_Flag = 'Y',
the average row of table Employees joins to a row of table Departments in 98% of time,
the average row of table Departments joins to 20 rows of table Employees.

Diagram:

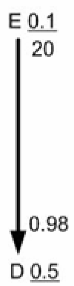


Figure 3: Example query diagram (adapted from (Tow, 2003))

This book is more concerned with explaining the construction and interpretation of the query diagrams with the different types of SQL queries than with discussing interesting optimising topics for SQL queries which could serve as pattern candidates.

"Database Tuning: Principles, Experiments, and Troubleshooting Techniques" (Shasha and Bonnet, 2003) treats a broad range of database tuning aspects, from hardware, operating system, and communication, further locking, logging, index tuning, normalisation, and triggers, to e-commerce applications and data warehouses. For each topic, a few selected aspects are considered in depth rather than attempting an exhaustive but inevitably shallow treatment. For example, the chapter about query tuning presents sophisticated discussions on how to minimise the `DISTINCT` operator and how to rewrite nested queries. One of the chapters explains some popular tuning tools like access plans, performance and event monitoring with reference to IBM DB2®, Microsoft® SQL Server, and Oracle® Database. Much of the presented material is supported by a well commented bibliography.

"Oracle Performance Tuning for 10gR2" (Powell, 2007) deals with performance tuning for the latest version of the Oracle® Database. More than a third of the 904 page book covers the topic SQL code tuning. Performance optimisation by reformulating SQL queries is presented, in detail, in chapter 6 (The Basics of Efficient SQL) and chapter 7 (Advanced Concepts of Efficient SQL). These chapters also include corresponding material regarding reformulating SQL from the same author's book "Oracle Data Warehouse Tuning for 10g" (Powell, 2005). The rest of the book discusses the relational database model (in part 1), tuning based on data modelling (in part 2), physical and configuration tuning (in part 4), and finally Oracle specific aspects like wait events including the wait even interface, latches, and tools and utilities (in part 5).

3.4 SQL and Patterns

SQL and patterns, as one topic, seem not yet prominently addressed in any book. Only the publisher Rampant TechPress (Rampant TechPress, 2006) has announced the October 2007 release of a new book authored by Vadim Tropashko called "SQL Design Patterns : The Expert Guide to SQL Programming". According to the details on their web site the book does not target the optimisation of SQL but rather the solving of complex problems by means of the SQL language analogue to the design pattern movement. According to the published table of contents the book will mostly include: counting, integer generators, exotic operators (list aggregate, product, factorial), constraints, trees, and graphs.

3.5 Summary

This chapter has presented the literature related to aspects of SQL and database optimisation and tuning which are relevant for this dissertation. The next chapter describes the research methods and tools used to perform the empirical analysis and to evaluate the effect of the patterns derived mainly from this literature research.

Chapter 4 Research Methods and Tools

This chapter describes the main aspects of the research methods including the functionality of the REPSI tool. The detailed procedure to measure and evaluate response times related to patterns then follows. Finally two important considerations are presented: the difficulties of getting deterministic response times and the application of box plots as a statistical tool.

The evaluation of patterns in this paper is based mainly on a black box measurement of the response times of two different SQL queries – one of them formulated without, and the other with, the application of a given pattern. The REPSI tool executes both queries in the same database instance, checking the equality of the retrieved data and recording the response time results in a database. By means of additional applied tools like R Project for Statistical Computing (The R Foundation for Statistical Computing, 2006), Excel compatible spread sheet software, and Oracle utilities (EXPLAIN PLAN, SQL TRACE, and TKPROF) the results are then analysed to assess the effect of the chosen pattern.

4.1 Functionality of the REPSI Tool

The REPSI tool (see also Appendix E) is written in the Java programming language and applies the JDBC (Java Database Connectivity) API to access the relational database. The REPSI tool enables a response time comparison of two SQL queries, for example, one without the application of a pattern (or SQL idiom) and one with. The response time is measured by applying the Java method `System.nanoTime()` immediately before and immediately after the corresponding JDBC method, which executes the SQL query. This allows the user to understand the extent to which the applied pattern might improve efficiency in the execution of the SQL query. To do this the tool was designed with the following functionality:

- Creation and maintenance of a master database containing:
    - Documentation of patterns and SQL idioms,
    - Pairs of SQL queries related to a certain pattern,
    - Test suites consisting of predefined sequences of actions such as:
        - Drop and / or create database tables in the test database,
        - Generate rows in a test database table, or
        - Execute pairs of SQL queries,
    - Description of runtime environments including information about processors, operating systems, and test database instances.
- Running a pair of SQL queries in a given frequency of occurrence (calibration mode) thereby recording, in the master database, information about the runtime environment, the response times, and a variety of statistics, e.g. kurtosis, skewness, standard deviation, and variance.

- Running a test suite (trial mode) and recording the complete runtime information in the master database.
- Exporting the recorded information from the calibration runs or trial runs in Excel style files.
- Exporting the response time data from the calibration runs into flat files containing R code which can then be imported into and subsequently processed by the R Project for Statistical Computing tool. R Project for Statistical Computing is an open source tool for statistical computations running on different platforms (among others UNIX, Windows, and MacOS).

Based on the results of running the tool with a specific database, an SQL user should then have a guide to improve the efficiency of the specific database optimiser by the application of suitable patterns and SQL idioms when formulating SQL queries.

4.2 Detailed Measurement Procedure

The relational database system used for both executing the pattern test queries and for storing the results is a standard installation of the Oracle® Database 10g Enterprise Edition on Windows XP. The operating system was a full version of Microsoft Windows® XP Professional including service pack 2. The main components of the hardware equipment are an Intel Pentium® D CPU 930 (dual core) with a clock speed of 3,000 MHz and 2,048MB RAM.

The initialisation parameters (Rich, 2005) of the installed RDBMS Oracle® Database 10g Release 2 were modified as shown in Table 1 with the intention of reducing the influence of caching effects to the measured response times.

Initialisation Parameter	Modified Value	Original Value
DB_CHACHE_SIZE	48 MB	584 MB
HASH_AREA_SIZE	0	1,048,576
LARGE_POOL_SIZE	0	4 MB
PGA_AGGREGATE_TARGET	10 MB	194 MB
SESSION_CACHED_CURSORS	0	20
SGA_MAX_SIZE	0	144 MB
SGA_TARGET	0	584 MB
SORT_AREA_SIZE	49,152	262,144

Table 1: Modification of the database initialisation parameters

The measurement procedure utilised the following tools:

- The REPSI tool for running the SQL queries, initialising Oracle's SQL TRACE tool, recording the response times, creating Excel compatible files containing the measurement results, and creating flat files containing R code processable by the R Project for Statistical Computing.
- The R Project for Statistical Computing for calculating the median and creating the box plots based on the output of the REPSI tool.
- An Excel compatible spreadsheet tool (e.g. Microsoft Office Excel® or OpenOffice Calc) for creating the plot charts based on the output of the REPSI tool.
- The Oracle tools SQL TRACE and TKPROF (including EXPLAIN PLAN) (Chan, 2005) for revealing the database access plans, the number of processed rows and the consumed time.

The following five steps formed the basis of the evaluation process. The mentioned files and the REPSI tool user manual are elements included in the REPSI tool software package.

Step 1: Optionally clear the master database with the REPSI tool:

- Drop the existing standard master database tables by applying the file `std_master_db_schema_drop.xml`.

- Create the database schema related to the standard master database tables by applying the file `std_master_db_schema_create.sql`.

- Create the database instance data related to the standard master database tables by applying the file `std_master_db_instance_mand.xls`.

- Create the database instance data related to the used vendors, database systems, operating systems, processors, and database instances (see file `std_master_db_instance_mand.xls` as an example).

Step 2: Apply the calibration functionality of the REPSI tool:

- Create an appropriate Excel compatible file containing the data related to the database tables `TMD_PATTERN_SQL_IDIOM` and `TMD_TEST_QUERY_PAIR` (see file `basic_master_db_instance.xls as an example`).

- Load the data contained in the Excel compatible file into the master database (see more details in the description of task "Set up the Master Database Schema and Default Instance" covered in the chapter 4 of the REPSI tool user manual).

- Run the REPSI tool in the calibration mode (see more details in the description of task "Perform a Calibration Run" covered in the chapter 4 of the REPSI tool user manual). According to the settings the REPSI tool executes the two queries under observation alternating 50 times.

Step 3: Create the box plot and determine the median with the R Project for Statistical Computing:

- Export the necessary R code to execute the R Project for Statistical Computing (see more details in the description of task "Export Calibration Data into R Compatible Files" covered in the chapter 4 of the REPSI tool user manual).

- Load the console of the R Project for Statistical Computing (upper left window in Figure 4) and process the R code. The lower window shows the R code, and the upper right window shows the resulting box plots.

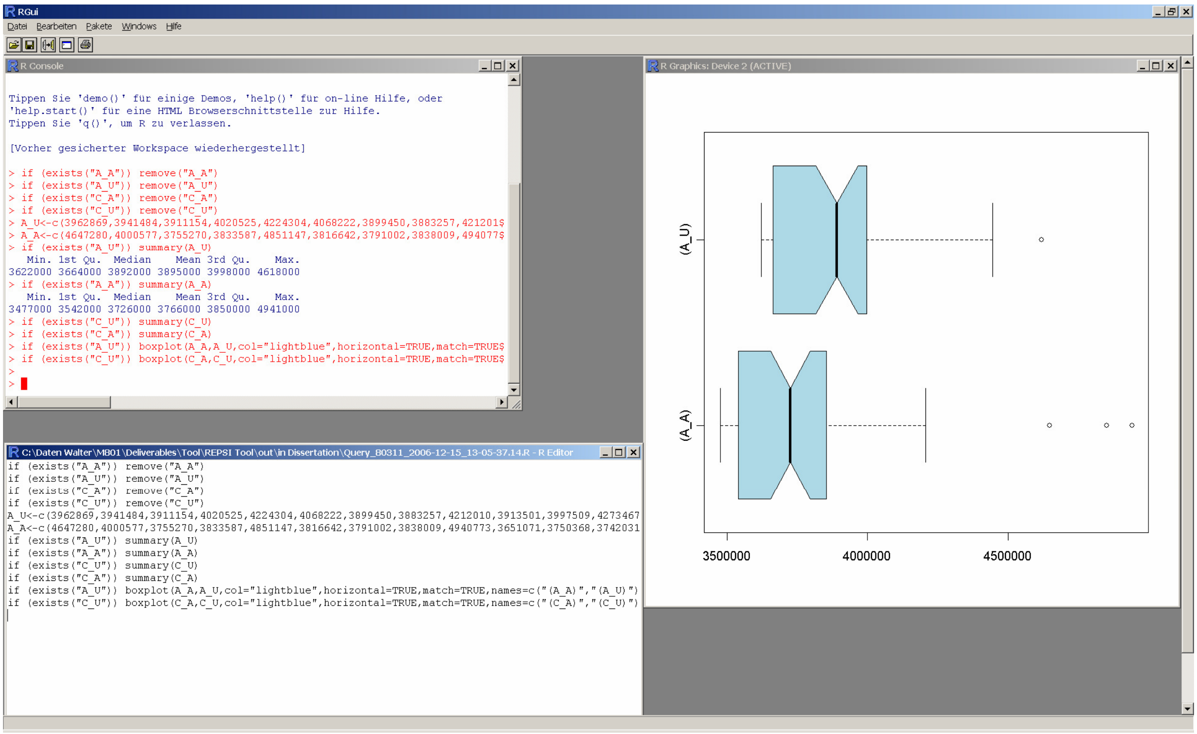


Figure 4: Processing a file exported by the REPSI tool with the R Project for Statistical Computing

Step 4: Create the dot chart with an Excel compatible tool:

- Export the results of the calibration run into an Excel compatible file (see more details in the description of task "Export Calibration Results into Excel Compatible Files" covered in the chapter 4 of the REPSI tool user manual).

▪Use an Excel compatible tool to create the dot charts from this file, e.g. with Microsoft Office Excel® use the chart wizard as shown in Figure 5 from the insert menu.

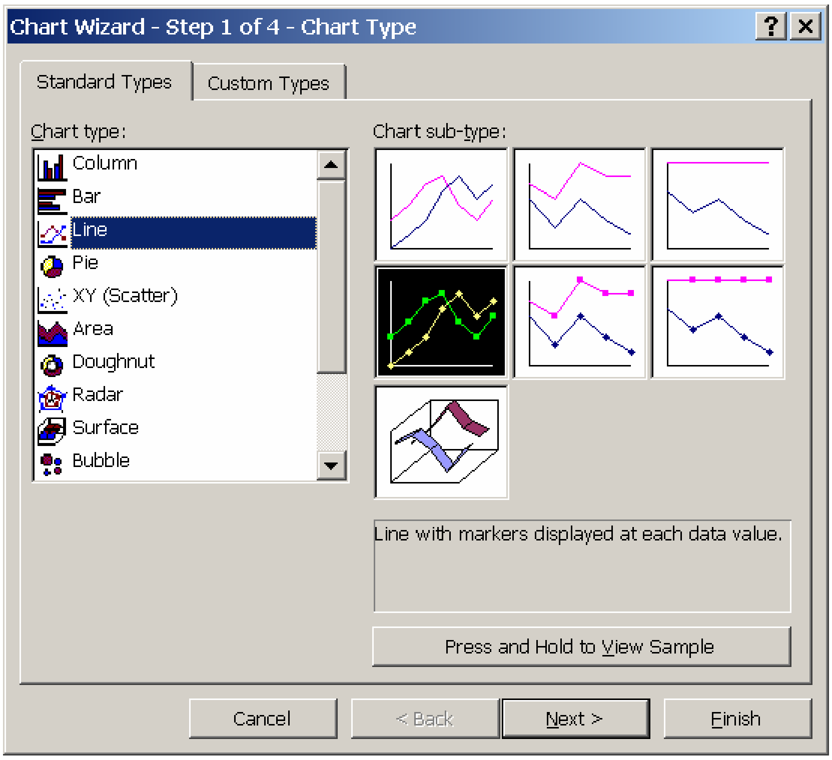


Figure 5: Chart wizard of Microsoft Office Excel® 2003

Step 5: Create an overview containing the database access plans, the number of processed rows, and the consumed time:

▪Before the REPSI tool runs the SQL queries it activates the Oracle SQL TRACE functionality. This induces Oracle to create files in the UDUMP directory containing very detailed trace data about the execution of the SQL queries. These files can then be processed by Oracle's TKPROF tool to create a more readable format of these data as shown in Figure 6: the first part shows the original SQL statement, next are the statistics related to the number of processed rows and consumed time in seconds and finally the access plan is shown.

```
SELECT DISTINCT AMOUNT_SERVICE
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      178      0.01       0.02          0        693          0        1774
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      180      0.01       0.02          0        693          0        1774

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1774  HASH UNIQUE (cr=693 pr=0 pw=0 time=20573 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33244 us)
```

Figure 6: Example of a TKPROF output

4.3 Deterministic Response Times

The Java method `System.nanoTime` delivers nanosecond precision but the underlying platform can limit the accuracy: for example, with Novell openSUSE 10.2 and AMD Duron™ processors (see Table 2) the maximum precision available is microseconds (µs), whereas with Microsoft Windows® XP and Intel Pentium® M processors the precision is available to nanoseconds (ns).

Number of Execution	Microsoft Windows® XP and Intel Pentium®	Novell openSUSE 10.2 and AMD Duron™
1	1,067	2,000
2	557	0
3	506	1,000
4	526	0
5	551	1,000
6	532	0
7	531	0
8	541	1,000
9	531	0
10	532	0

Table 2: Results of measuring the response time of the `nanoTime()` method on different platforms

In any case the measured time spans are only comparable amongst each other because they represent the elapsed time based on processor cycles, as appose to the method `Calendar.getTimeInMillis()` which delivers the time in UTC milliseconds (Sun Microsystems Inc., 2004). According to Cyran (Cyran, 2005) the first execution of a query can be expected to be slower because of additional parsing and optimisation efforts which can be saved due to caching mechanisms with the second and following execution of the same query. This expected behaviour could not, however, be reproduced - see Figure 7. The response times of the queries are very much scattered by outliers and in the extreme, depending on the granularity of the time resolution, it is hard to find two identical response times of the same query.

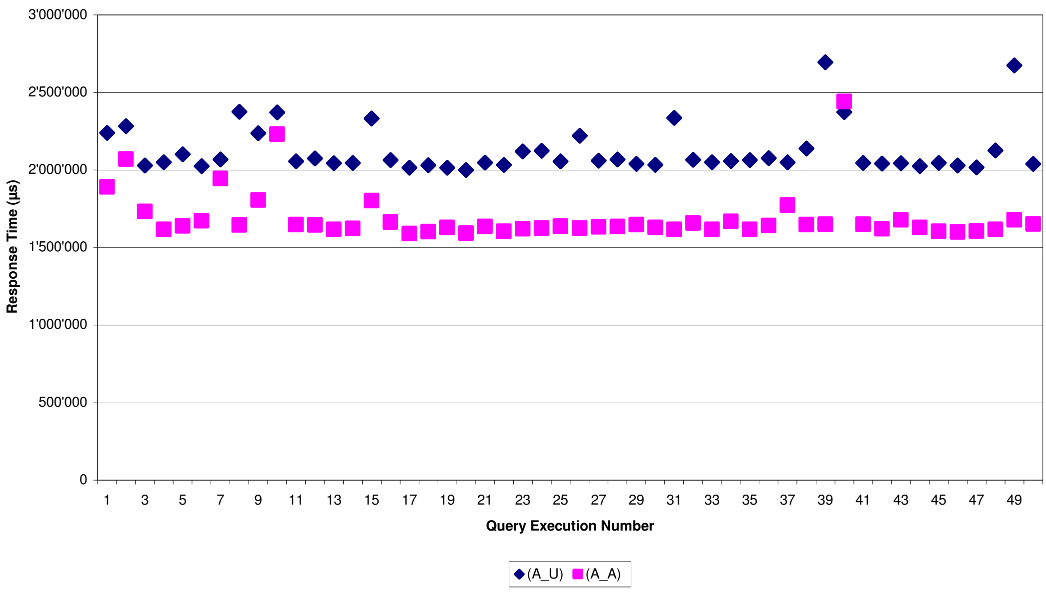


Figure 7: Example of 50 executions of two queries

There are a number of factors, inherent in modern operating systems like Linux or Windows XP, responsible for these effects (Barabanov, 1997, Brosky and Rotolo, 2003). The following are worth mentioning:

- The multi-tasking functionality forces the process scheduler to interrupt a running process with lower priority in order to (re)start a process with higher priority based on a set of predefined rules.

- Depending on the re-entrant capabilities of the kernel, a process running in the kernel mode cannot be interrupted until the system call completes or until it blocks. A similar problem arises if interrupts are disabled to simplify the handling of synchronisation processes associated with shared kernel data structures.

- Ignoring the available RAM size and utilising disk space for the persistence of the virtual memory raises the level of unpredictability in modern operating systems.

Graphics and network activities, as well as garbage collection tasks are further candidates preventing deterministic response times when executing the same tasks.

4.4 Box Plots as a Statistical Tool

Because of the non-deterministic behaviour related to response times, statistical methods have to be employed to assess the benefits of a pattern. As opposed to pure mathematical models, which often require that the sample data conform to a special distribution (e.g. Gaussian) or to a family of distributions (e.g. normal). Box plots can be applied without any specific underlying statistical assumptions as in contrast to other statistical tools such as F-test for comparing variances, or t-test for comparing means which are based on assumptions such as normal distribution and equal variance, or are susceptible to the occurrence of outliers. Additionally a graphical display usually conveys the statistical context much more visually than a set of mathematical figures or tables (Grier, 1992).

A box plot is a simple graphical statistic tool which is, therefore, used to compare the data of different samples. The box plot (see Figure 8) depicts the minimum and maximum value, the first and third quartile, and the median. A rectangle containing 50% of the data is drawn based on both quartiles with the median as a middle line. Whiskers bordered with fences on both ends mark the area of reasonable values, outliers can be found beyond the fences.

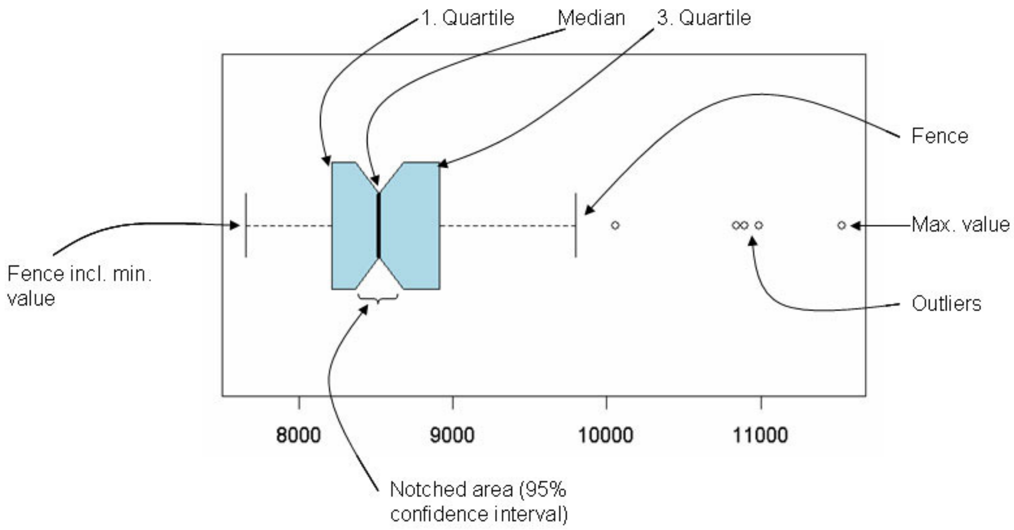


Figure 8: Elements of a box plot

McGill (McGill et al., 1978) introduced the notched area to denote a 95% confidence interval for the median allowing the conclusion that non-overlapping notches of two box plots guarantee a significant difference of the two medians at about a 95% confidence level.

4.5 Summary

This chapter described the functionality of the REPSI tool and then the detailed procedure to measure and evaluate response times related to patterns. Two important aspects of the empirical analysis were then considered in detail: the difficulties of getting deterministic response times and the choice of box plots as the tool to be used for recording and presenting the results of the analysis. This research approach was designed to evaluate the performance of two queries, one with, and one without, the application of a pattern. Potential pattern candidates, related to the basic clauses of an SQL query: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` and `ORDER BY`, are presented and discussed in the next chapter together with the results of their application.

Chapter 5 Patterns Related to Basic Query Concepts

This chapter briefly outlines the basic structure of the `SELECT` statement according to the SQL:2003 standard. Following the sequence of the `SELECT` statement clauses, it then goes on to describe specific patterns. These patterns are designed, based on concepts from the literature research, with the intention of improving the performance of SQL queries simply by reformulating them. The evaluation of the performance (related to the Oracle® Database) of the query both with, and without, the application of the pattern is then presented and discussed in detail (for full details see also Appendix B).

5.1 Basic Query Concepts

Based on the principles of relational algebra and tuple relational calculus (Silberschatz, 2006) the `SELECT` statement represents one of the pivotal aspects of SQL's query capabilities. The `SELECT` statement (see Figure 9) allows the user to retrieve a resulting table of rows – single or grouped – and columns – simple or derived – from one or more tables of a relational database. Omitting the `INTO` clause, relevant only for single row queries, and the, not yet in Oracle supported, `WINDOWS` clause, the `SELECT` statement is defined according to the SQL:2003 (ISO/IEC JTC 1/SC 32/WG 3, 2003) standard as follows:

```
  SELECT [ ALL | DISTINCT ] <select list>
    FROM <from clause>
[  WHERE <where clause> ]
[  GROUP BY <group by clause> ]
[ HAVING <having clause> ]
[  ORDER BY <order clause> ]
```

Figure 9: `SELECT` statement syntax (SQL:2003)

The `SELECT` clause defines the desired set of columns, constants or derived values to appear in the output table which constitutes the projection in terms of the relational algebra. The default set quantifier `ALL` allows for duplicate rows being contained in the resulting table as in a multiset (bag). The `DISTINCT` set quantifier, on the other hand, forces the set property of the resulting table in the same manner as the relational algebra. The `FROM` clause specifies a Cartesian product composed of table expressions, which themselves are made up of base tables or derived tables (e.g. viewed tables or subqueries). The selection operation of the relational algebra is represented by the `WHERE` clause which is applied as a filter to the preceding `FROM` clause. The `GROUP BY` clause transforms the intermediate table resulting from the previous `FROM` and `WHERE` clauses into a grouped table. `GROUP BY` is a mandatory clause if the projection contains, amongst others, columns resulting from aggregate functions. In the same way that the `WHERE` clause is applied to the intermediate table out of the `FROM` clause, the `HAVING` clause is applied as a filter to the grouped table resulting from the explicit or implicit `GROUP BY` clause. Finally the `ORDER BY` clause determines the sequence of the rows in the resulting table.

5.2 Selection Patterns (`WHERE`)

5.2.1 Index Exploitation

Name	Index Exploitation
Problem	How to avoid having unemployed indexes during the selection phase?
Potential Solutions	(a) Don't apply functions to indexed columns. (b) Move function calls to the non-indexed components. (c) Avoid implicit applied functions like type conversions. (d) Avoid leading wildcards.
Related	Defensive Projection, `WHERE` Outperforms `HAVING`
References	(Celko, 2005a, Haas, 2005, Powell, 2007)

The major task of an index is to improve the performance in connection with both selecting rows and joining tables (Garcia-Molina et al., 2002). The Oracle optimiser shows significant performance problems if the query applies a function to the indexed column in the selection condition. The performance worsening is less if the cardinality of the result set is higher (TQP_80402 4% with a result set cardinality of 20%), but is more significant with a low cardinality resulting from the applied full table scan (TQP_80401 from 3 ms to 1,116 ms with a result cardinality 1 of 1,945,431 rows). The database access plan and consequently the performance behaviour are very similar for functions applied implicitly by the query processor, e.g. type conversions (TQP_80421). The optimiser fails, in this case, to apply the implicit function to the non-indexed components of the selection condition, which would then prevent the performance worsening (TQP_80411).

LIKE is a predicate for pattern matching (format: <source value> [NOT] LIKE <pattern value> …) allowing for the optional inclusion of two wildcard characters into the pattern value: _ which represents exactly one arbitrary character and % which represents an arbitrary substring of any length. Starting the pattern value with a wildcard character also prevents the use of an existing index.

From a performance point of view, the Oracle query processor shows no significant difference between the use of both wildcard characters independent of the given result cardinality(TQP_80431 and 80433). The query processor also shows no significant difference in performance when the LIKE predicate is substituted by an equivalent range selection (TQP_80432).

5.3 Grouping Patterns (GROUP BY)

The GROUP BY clause defines the partitioning of the rows derived from the FROM and WHERE clauses based on a set of grouping columns. The GROUP BY clause is mandatory if the SELECT clause consists of a mixture of columns and aggregate functions. The output of the GROUP BY clause (the so called grouped table) is unique with respect to the combination of values in the grouping columns – all columns containing NULL are considered to have the same value.

5.3.1 `GROUPING SETS` Beats `UNION`

Name	`GROUPING SETS` Beats `UNION`
Problem	What is the best way to include subtotals into the result set?
Potential Solutions	(a) The use of a `GROUPING SETS` clause instead of the application of a `UNION` should result in a significant performance improvement. (b) `CUBE` and `ROLLUP` are shortcuts that can be used to simplify the application of certain variants of the `GROUPING SETS` clause.
Related	Minimise Grouping Columns
References	(Gulutzan and Pelzer, 2003, Powell, 2007)

`GROUPING SETS`, introduced with the SQL:1999 standard, allow for the introduction of subtotals and of grand totals provided that `()` is contained in the grouping column list. `CUBE` and `ROLLUP` are shortcut versions of particular combinations of grouping column lists. For Example:

- `GROUP BY GROUPING SETS ((period, sign))` is semantically equivalent to `GROUP BY period, sign`.
- `GROUP BY GROUPING SETS (period, sign, ())` contains only the subtotals of the column period, the subtotals of the column sign, and the grand total.
- `GROUP BY ROLLUP (period, sign)` is semantically equivalent to `GROUP BY GROUPING SETS ((period, sign), period, ())`.
- `GROUP BY CUBE (period, sign)` is semantically equivalent to `GROUP BY GROUPING SETS ((period, sign), period, sign, ())`.

Only the semantic equivalence of `GROUP BY CUBE` with the corresponding `GROUP BY GROUPING SETS` is not recognised by the Oracle® Database: the `GROUP BY CUBE` is 31% more efficient then its semantically equivalent counterpart with `GROUPING SETS` (TQP_80511).

`GROUPING SETS` can be simulated by the application of `UNION`, but this causes an immense performance degradation, because each additional `UNION` usually requires the reprocessing of all the related database tables (Celko, 2005a) as demonstrated by TQP_80501 239% with `CUBE`, TQP_80502 136% with `GROUPING SETS`, and TQP_80503 160% with `ROLLUP`.

5.3.2 Minimise Grouping Columns

Name	Minimise Grouping Columns
Problem	Does the number of grouping columns affect the query performance?
Potential Solution	Keep the number of grouping columns small.
Related	`GROUPING SETS` Beats `UNION`, Minimise Sorting Columns
References	(Gulutzan and Pelzer, 2003)

Reducing the number of grouping columns should theoretically improve the performance as a result of reduced processing cost. The main difficulty is to find ways of reducing the grouping columns. One possibility is to combine character columns by concatenation or numeric columns by appropriate multiplications. Even assuming that the changed projection is sufficient for the user's requirements, the performance of the Oracle query processor does not actually improve with this pattern. On the contrary the performance even significantly deteriorates under certain circumstances, e.g.

TQP_80701 with implicit type conversions and concatenations is 20% worse. Another possibility is to remove superfluous grouping columns, for example, if due to a selection, a grouping column contains only one value. But in this case the Oracle query processor shows no significant performance improvement (TQP_80702).

5.4 Group Selection Patterns (`HAVING`)

5.4.1 `WHERE` Outperforms `HAVING`

Name	`WHERE` Outperforms `HAVING`
Problem	Can the `HAVING` clause be used instead of the `WHERE` clause?
Potential Solution	The selection based on the `WHERE` clause happens before the selection based on the `HAVING` clause. Therefore the `WHERE` clause should always be chosen if the aggregated values are not affected by the selection.
Related	Index Exploitation
References	(Gulutzan and Pelzer, 2003, Powell, 2007, Shasha and Bonnet, 2003)

The purpose of the `HAVING` clause is to filter groups created by the `GROUP BY` clause rather than to filter single rows. Simply filtering rows with the `HAVING` clause is normally a misuse of the `HAVING` clause because this could be done much earlier with the `WHERE` clause. This is demonstrated by the much earlier appearance of the WHERE clause in the common evaluation sequence of the query (`FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, and finally `SELECT` (Melton and Simon, 2002)), making it clear that the later use of the `HAVING` clause forces the unnecessary processing of intermediate data thereby worsening the performance.

The issue for the Oracle query processor is not that the selection via `WHERE` results in improved performance compared to the `HAVING` clause, but whether the Oracle query processor is able to optimise the query by propagating appropriate conditions from the HAVING clause to the WHERE clause. Here the Oracle optimiser fails (TQP_80801 and TQP_80802).

5.5 Row Order Patterns (ORDER BY)

5.5.1 Avoid Explicit Sorting

Name	Avoid Explicit Sorting
Problem	Is it possible to exploit implicit sorting activities?
Potential Solutions	(a) A DISTINCT clause matching an index may cover sorting requirements. (b) A GROUP BY clause matching an index may cover sorting requirements.
Related	Distinct Rows
References	(Celko, 2005a, Powell, 2007)

Sorting algorithms can be applied explicitly based on an ORDER BY clause or implicitly to determine distinct rows or to group data. If the query statement already employs a suitable implicit sort, there should be a significant performance improvement if the then superfluous explicit sort is omitted. The Oracle query processor chooses a sorting or a hashing operation depending on whether the ORDER BY clause is part of the query or not (see Table 3).

ORDER BY	DISTINCT	GROUP BY
No	HASH UNIQUE	HASH GROUP
Yes	SORT UNIQUE	SORT GROUP

Table 3: Operation type chosen by the Oracle query processor depending on an explicit sort requirement

The hashing operation usually runs faster than the sorting operation (`DISTINCT`: TQP_80901 and TQP_80902, `GROUP BY`: TQP_80911 and TQP_80912). A noteworthy exception exists if the explicit sort requirement can be satisfied via an appropriate index, i.e. the data are in this case already pre-sorted and the query with the `ORDER BY` clause runs 14% more efficiently than the one without (`DISTINCT`: TQP_80903, `GROUP BY`: TQP_80913).

5.5.2 Minimise Sorting Columns

Name	Minimise Sorting Columns
Problem	Does the number of columns affect the query performance?
Potential Solutions	(a) Minimise the number of columns (expressions) in the `ORDER BY` clause. (b) Minimise the number of columns to be projected.
Related	Defensive Projection, Minimise Grouping Columns
References	(Gulutzan and Pelzer, 2003)

In the same way as for the grouping columns, reducing the number of sorting columns should also theoretically improve the performance as a result of reduced processing cost. The main difficulty is again to find ways of reducing the sorting columns. One possibility is to combine character columns by concatenation or numeric columns by appropriate multiplications. The Oracle query processor shows no performance improvement with the application of this pattern. Combining numeric columns by multiplication also has no significant performance effect (TQP_81001). String concatenations with previous implicit type conversion significantly worsen performance by 20% (TQP_81002). Removing superfluous sorting columns, for example, where as a result of a selection the content of a sorting column is reduced to only one value, shows no significant performance improvement with the Oracle query processor (TQP_81003).

As expected, performance improves based on the reduction of columns to be projected. Without a sort, the performance improvement of the same query was 30% (TQP_80301) and with an `ORDER BY` clause the performance improvement rose to 40% (TQP_81011).

5.6 Projection Patterns (`SELECT`)

5.6.1 All Rows / Distinct Rows

Name	All Rows
Problem	What is the best way to include duplicates into the result set?
Potential Solutions	(a) Within a query use `SELECT [ ALL ]`. (b) Within certain aggregate functions use `<aggregate function> ( [ ALL ] <value expression> )`. (c) Use the set operator `UNION ALL` instead of `UNION`.
Related	Distinct Rows
References	(Celko, 2005a, Garcia-Molina et al., 2002, Powell, 2007, Shasha and Bonnet, 2003)

Name	Distinct Rows
Problem	What is the best way to exclude duplicates from the result set?
Potential Solutions	(a) Within a query either include a unique key of each table in the `FROM` clause into the projection or use `SELECT DISTINCT`. Using `GROUP BY` has the same effect as `SELECT DISTINCT`. (b) Within certain aggregate functions use `<aggregate function> ( DISTINCT <value expression> )`. (c) Use `SELECT DISTINCT` instead of `UNION`.
Related	All Rows, Avoid Explicit Sorting, `JOIN` and `DISTINCT`
References	(Celko, 2005a, Garcia-Molina et al., 2002, Powell, 2007, Shasha and Bonnet, 2003)

Unlike sets provided by relational algebra queries, an SQL query may provide a multiset of rows including duplicates in the result (Chaudhuri and Vardi, 1993). Applying the `DISTINCT` operator in the projection eliminates the duplicate rows but this in turn means an extra effort for the query processor (Powell, 2007).

The Oracle query processor executes a hashing operation called `HASH UNIQUE` which requires a substantial additional expenditure (TQP_80101 (33%), TQP_80103 (26%), and TQP_80104 (17%)). Restricting the projection to indexed columns does not reduce the effort significantly (TQP_80102).

The inadvertently redundant use of the `DISTINCT` operator should be recognised and consequently ignored by the query processor. Due to semantic weaknesses, the Oracle query processor performs an expensive superfluous hash operation even if the projection list includes a unique key (TQP_80103). Applying `GROUP BY` clause instead of the `DISTINCT` operator results in a similar increased effort although the

Oracle query optimiser employs a slightly different operation with `GROUP BY` clause called `HASH GROUP BY` (TQP_80105).

In addition to causing performance issues, the use of the `DISTINCT` operator also limits the use of some other SQL language features, e.g. a cursor or a view based on any query expression containing a `DISTINCT` operator is not updatable, or in an SQL query statement containing `SELECT DISTINCT` all columns referred to in the `ORDER BY` clause have to be contained in the projection.

In connection with the aggregate functions `AVG` (TWP 80111), `COUNT` (TQP_80112) and `SUM` (TQP_80115) the Oracle query optimiser operates with the different sort operations, `SORT AGGREGATE` without `DISTINCT` operator, and `SORT GROUP BY` with `DISTINCT` operator. The latter operation requires a significant extra effort, e.g. 168% with `AVG`, 167% with `COUNT`, or 170% with `SUM`. Based on a correct semantic analysis, the Oracle query optimiser treats the aggregate functions `MAX` and `MIN` equally, with or without, the application of `DISTINCT` (TQP_80113 and 80114).

Applying the set operators `EXCEPT`, `INTERSECT`, and `UNION` in an SQL query produces, by default, a set of rows excluding the duplicates in the result. In this case the Oracle query processor adds an extra sort operation called `SORT UNIQUE` on top of the `UNION ALL` effort. With a table full scan the extra effort of `UNION` is 51% per 1,000 resulting rows (TQP_80121). The `UNION ALL` operator should, therefore, be explicitly chosen in every case where duplicate rows do not pose a problem. If the projection includes only unique keys the applied (full) index scan shows, as is to be expected, no significant difference (TQP_80122).

Applying the `HASH UNIQUE` operation instead of the `SORT UNIQUE` operation by formulating the query `SELECT DISTINCT * FROM (SELECT … UNION ALL SELECT …)` instead of `SELECT … UNION SELECT …` shows a significant improvement with the performance (TQP_80123 14%).

5.6.2 Defensive Projection

Name	Defensive Projection
Problem	How can choices related to the projection (choice of columns) contribute to improved performance?
Potential Solutions	(a) Be specific and avoid the asterisk in the `SELECT` clause. (b) If possible restrict the columns in the `SELECT` clause to the indexed ones.
Related	Index Exploitation, Minimise Sorting Columns
References	(Celko, 2005a, Powell, 2007, Shasha and Bonnet, 2003)

Database tables, big in number of columns and number of rows, should be projected with as few as possible columns. Projecting only one instead of all 32 columns of a table with 102'583 rows showed a significant performance improvement (TQP_80301 30%). Including large binary or text columns into a needless projection is expected to worsen the performance significantly because of the much more expensive retrieval effort (Cumming and Russell, 2007).

Without an appropriate selection in the `WHERE` clause the projection of indexed columns shows only minor performance improvements against the projection of non indexed columns (TQP_80311 4%). The Oracle query processing time shows no significant change whether or not the sequence of the columns in the projection matches with the sequence of the columns in the index (TQP_80321).

5.7 Summary

This chapter has presented patterns, each related to a clause of a simple SQL query. The patterns were intended to improve the performance of SQL queries by simply reformulating the queries. The effectiveness of patterns was evaluated based on selected SQL queries executed with an Oracle® Database. The next chapter will present and discuss, in detail, patterns related to subqueries and joins the more advanced concepts of SQL queries.

Chapter 6 Patterns Related to Subqueries and Joined Tables

After a short description of subqueries, this chapter examines patterns related to semantic equivalences of predicates (`EXISTS`, `IN`) and quantified comparison operators (`ALL`, `ANY`). Similarly, after a brief introduction of the joined table concept, patterns intended to improve performance of joined tables are discussed (for full details see also Appendix B and Appendix C).

6.1 Subqueries

A subquery is a complete query (only excluding the `ORDER BY` clause) enclosed by parenthesis which is arbitrarily deep nested into another query. A subquery may not, however, be used in the argument of an aggregate function. The SQL:2003 standard distinguishes the following subquery types:

- scalar subqueries returning an atomic or scalar value as a result (degree equals to 1 and cardinality less or equals to 1),
- row subqueries (including scalar subqueries) returning one row as a result (degree greater than zero and cardinality less or equals to 1), and
- table subqueries (including row subqueries) returning a whole table (degree greater than zero and any cardinality).

According to the SQL:2003 standard the subquery type determines the applicability of the subquery in the different clauses of the `SELECT` statement as shown in Table 4. Different RDBMS products may impose restrictions on the format of the subquery or on the application of subqueries in the different clauses of the `SELECT` statement. For example, in the former case, Oracle® Database limits the use of the `GROUP BY` clause in the subquery, and in the latter, doesn't allow the use of scalar subqueries in the `GROUP BY` clause (Lorentz, 2005).

Clause	Subquery type
SELECT	Scalar
FROM	Table
WHERE	Row
GROUP BY	Scalar
HAVING	Row
ORDER BY	Scalar

Table 4: Applicability of subquery types in the different clauses of the `SELECT` statement

A subquery may be correlated to any surrounding query by appropriate references in the `WHERE` clause or the `HAVING` clause of the subquery. Uncorrelated subqueries must be executed only once for the outer query. Correlated subqueries are, on the other hand, dependent on values retrieved by the outer query and must, therefore, be executed once for every row retrieved by the outer query (Dietrich, 2001).

6.2 Subquery Patterns

The SQL:2003 standard defines explicitly the equivalence of `ANY` and `SOME` and of `= ANY` and `IN`. The implementation of these equivalences is straightforward and is not therefore further investigated. Subqueries applied in the `FROM` clause show no significant performance degradation with the Oracle® Database (TQP_99901).

6.2.1 `ALL` / `ANY` or `EXISTS`

Name	`ALL` / `ANY` or `EXISTS`
Problem	In the case of semantic equality, how can a quantified comparison operator `ALL` or `ANY` be replaced by a predicate `EXISTS`?
Potential Solutions	(a) A query of the form `col1 comp ALL (SELECT col2 FROM … WHERE cond)` can be rewritten as `NOT EXISTS (SELECT … FROM … WHERE cond AND NOT col1 negatived_comp col2).` (b) A query of the form `col1 comp ANY (SELECT col2 FROM … WHERE cond)` can be rewritten as `EXISTS (SELECT … FROM … WHERE cond AND col1 negatived_comp col2).`
Related	`ALL / ANY` or `MAX / MIN`, `EXISTS` or `COUNT`, `EXISTS` or `IN`
References	(Celko, 2005a, Moerkotte, 2006)

"`ANY`, `SOME`, and `ALL` comparisons are generally not conducive to SQL tuning. In some respects, they are best not used." (Powell, 2007). Replacing `ALL` or `ANY` by an equivalent rewritten query applying `EXISTS` as suggested by Celko (Celko, 2005a) and Moerkotte (Moerkotte, 2006) may show an improved performance with certain databases.

It must be considered that an `ALL` operator evaluates to unknown if any row of the subquery returns a `NULL` value (ISO/IEC JTC 1/SC 32/WG 3, 2003). The `MAX` and `MIN` functions eliminate all `NULL` values and return only `NULL` if no other value exists. The rewritten query is therefore only equivalent to the original one, if the argument of the corresponding extreme function doesn't contain a `NULL` value in any row.

The Oracle® Database indeed shows a significant performance improvement mainly based on a substantially reduced effort in processing the subquery (TQP_86201 `ALL` / `EXISTS` and TQP_86202 `ANY` / `EXISTS`).

6.2.2 ALL / ANY or MAX / MIN

Name	ALL / ANY or MAX / MIN
Problem	Which of the quantified comparison operators ALL or ANY should be replaced by a semantically equivalent query applying an extreme function?
Potential Solutions	(a) The operators < ALL or <= ALL may be replaced by < or <= along with a MIN function. (b) The operators < ANY or <= ANY may be replaced by < or <= along with a MAX function. (c) The operators > ALL or >= ALL may be replaced by > or >= along with a MAX function. (d) The operators > ANY or >= ANY may be replaced by > or >= along with a MIN function.
Related	ALL / ANY or EXISTS, EXISTS or COUNT, EXISTS or IN
References	(Celko, 2005a, Ganski and Wong, 1987, Gulutzan and Pelzer, 2003)

The intention here is that the query processor may be able to exploit a potential existing index on the argument of the MAX or MIN function with the rewritten format. As with the previous pattern, it is important to consider the different semantics between ALL or ANY and EXISTS in the case of NULL values in the subquery.

The Oracle® Database in fact applies different access plans, but the result as measured by performance shows no significant difference (TQP_86101 <= ANY and MAX, TQP_86111 <= ALL and MIN, TQP_86121 >= ANY and MAX, and 86131 >= ALL and MIN function).

6.2.3 EXISTS or COUNT

Name	`EXISTS` or `COUNT`
Problem	In the case of semantic equality, how can the predicate `EXISTS` be replaced by applying the `COUNT` function?
Potential Solutions	(a) A query of the form `WHERE EXISTS (SELECT col1 FROM …)` can be rewritten as `0 < (SELECT COUNT(*) FROM …).` (b) A query of the form `WHERE NOT EXISTS (SELECT col1 FROM …)` can be rewritten as `0 = (SELECT COUNT(*) FROM …).`
Related	`ALL` / `ANY` or `EXISTS`, `ALL / ANY` or `MAX / MIN`, `EXISTS` or `IN`
References	(Ganski and Wong, 1987)

Ganski (Ganski and Wong, 1987) showed these transformation possibilities amongst others in his reply to a few problems contained in a paper from Kim (Kim, 1982) which first presented algorithms for unnesting subqueries based on a new subquery classification schema.

The Oracle® Database appears to recognise the semantic equivalence of the queries and therefore applies identical access plans. In return no significant difference in the performance of the two query formats is shown (TQP_86401 with `EXISTS` and TQP_86411 with `NOT EXISTS`).

6.2.4 EXISTS or IN

Name	`EXISTS` or `IN`
Problem	Which of the predicates `EXISTS` or `IN` should be preferred in the case of semantic equality?
Potential Solutions	(a) If the outer table has many rows and the inner table has few rows, then use `IN`. (b) If most of the rows are filtered by the outer query, then use `EXISTS`. (c) If most of the rows are filtered by the inner query, then use `IN`. (d) If the outer query is of format "`WHERE NOT …`", then use `NOT EXISTS`. (e) Order the list of values of an `IN` predicate by frequency of probable usage. (f) Try to avoid duplicates in the list of values of an `IN` predicate.
Related	`ALL` / `ANY` or `EXISTS`, `ALL / ANY` or `MAX / MIN`, `EXISTS` or `COUNT`
References	(Celko, 2005a, Gulutzan and Pelzer, 2003, Haas, 2005, Powell, 2007)

The result of the `EXISTS` predicate is true, if the cardinality of the table produced by the applied table subquery is greater than zero. The `IN` predicate, in connection with a table subquery, is true, if a given row of comparative values is contained in the resulting table produced by the table subquery. For both predicates the table subquery may be correlated with an outer query. As shown in Figure 10, many queries formulated using `EXISTS` can be reformulated to be semantically equivalent using `IN` and vice versa.

```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE EXISTS (SELECT *
                FROM TSMT_SERVICE_PROVIDER SEP
               WHERE SEP.HIERARCHY_CD LIKE 'PGIN%'
                 AND SEP.IDENT = DDC.SERVICE_PROVIDER_ID)

SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE SERVICE_PROVIDER_ID IN (SELECT IDENT
                                 FROM TSMT_SERVICE_PROVIDER
                                WHERE HIERARCHY_CD LIKE 'PGIN%')
```

Figure 10: Semantically equivalent queries, the first formulated using `EXISTS` and the second using `IN`

None of the recommendations for the choice between `EXISTS` and `IN` show any significant performance improvement with the Oracle® Database:

(a) `IN` instead of `EXISTS`: different access plans but similar performance (TQP_86301 without an indexed column and TQP_86302 with an indexed column),

(b) `EXISTS` instead of `IN`: different access plans but similar performance (TQP_86303 without an indexed column and TQP_86304 with an indexed column),

(c) `IN` instead of `EXISTS`: identical access plans and similar performance (TQP_86305 without an indexed column and TQP_86306 with an indexed column),

(d) `NOT EXISTS` instead of `WHERE NOT`: identical access plans and similar performance (TQP_86307 without an indexed column, TQP_86308 with an indexed column, TQP_86312),

(e) Order by frequency: different access plans and similar performance (TQP_86309), and

(f) Avoidance of duplicates: same access plan (TQP_86310 with `DISTINCT`), different access plans (TQP_86311 without duplicates in table `TSMT_DETAILED_CHARGING_86311`, TQP_86312 with ordering by frequency in table `TSMT_DETAILED_CHARGING_86309`). Performance was similar in all cases.

6.3 Joined Tables

A joined table operation consists of at least two, not necessarily, distinct tables (i.e. base table, derived table, or viewed table in the sense of SQL:2003) in the `FROM` clause. The main purpose of a joined table is to construct a projection based on more than one table. Each pair of joined tables is optionally connected with a join condition and / or a join type (Silberschatz, 2006).

The join condition specifies the matching type of the rows contained in the joined table:

- `NATURAL`: the matching is based on all the column references which have the same name in both tables.
- `ON <condition>`: any condition describing the matching criteria.
- `USING (col1, col2, …)`: similar to `NATURAL`, but restricting the matching column references with the same name.

Join conditions are exclusive and if the join condition is missing, only a simple Cartesian product will be created. The `NATURAL` and the `USING` clause are solely equi-joins, meaning that the comparison of the joining column references is based only on the equality operator. The `ON` clause allows the additional use of other comparison operators which is then called a theta-join.

The join type defines the treatment of the unmatched rows based on the join condition:

- `INNER JOIN` (default value): the unmatched rows are ignored,
- `LEFT OUTER JOIN`: the unmatched rows of the left table are preserved and the missing columns from the right table are replaced by `NULL`,
- `RIGHT OUTER JOIN`: the unmatched rows of the right table are preserved and the missing columns from the left table are replaced by `NULL`,
- `FULL OUTER JOIN`: combines the functionality of `LEFT OUTER JOIN` and `RIGHT OUTER JOIN`.

The `CROSS JOIN` is an `INNER JOIN` without a join condition and produces therefore only a cartesian product. The complete syntax of joined table operations as adapted from the SQL:2003 standard can be found in Figure 11. This kind of join syntax was first introduced with SQL-92 and then extended with SQL:1999 (Melton and Simon, 2002). Prior to these standards joined tables were only supported by defining a Cartesian product in the `FROM` clause and reducing the same by defining an appropriate `WHERE` clause.

```
FROM <table1> [AS <alias1>]
              CROSS JOIN <table2> [AS <alias2>]
     [...]
```

```
FROM <table1> [AS <alias1>]
              NATURAL [FULL [OUTER] |
                       INNER        |
                       LEFT [OUTER] |
                       RIGHT [OUTER]]
              JOIN <table2> [AS <alias2>]
     [...]
```

```
FROM <table1> [AS <alias1>]
                 [FULL [OUTER] |
                  INNER        |
                  LEFT [OUTER] |
                  RIGHT [OUTER]]
              JOIN <table2> [AS <alias2>]
                 [ON <condition1> [{AND | OR} <condition2> [...]] |
                  USING (<column1> [, ...])]
     [...]
```

Figure 11: Variants of joined table syntax (SQL:2003)

6.4 Joined Table Patterns

6.4.1 Complete JOIN Selection

Name	Complete `JOIN` Selection
Problem	Does redundant selection information affect performance?
Potential Solution	Maximising the selection information in the `JOIN` and `WHERE` clauses (even if this appears redundant) may support the query optimiser in improving the access plans and in reducing the number of iterations of nested loops.
Related	Ordering `INNER JOIN`
References	(Celko, 2005a, Gulutzan and Pelzer, 2003)

The Oracle® Database shows no significant performance improvement neither with the placement of the selection predicates in the `WHERE` clause or in the `JOIN` clause (TQP_87401), nor with a maximum redundancy in formulating the selection predicates (TQP_87402).

6.4.2 JOIN and DISTINCT

Name	`JOIN` and `DISTINCT`
Problem	When is the `DISTINCT` clause superfluous in a query containing joined tables?
Potential Solution	If the projection of a `JOIN` contains a unique key of one table and the key columns of all other tables are joined by an equi-join, then the `DISTINCT` clause in the projection is unnecessary.
Related	Distinct Rows
References	(Shasha and Bonnet, 2003)

The redundant `DISTINCT` clause shows no significant performance degradation with the Oracle® Database even though the database access plan is slightly different (TQP_87301).

6.4.3 Ordering INNER JOIN

Name	Ordering `INNER JOIN`
Problem	How should the joined tables in an `INNER JOIN` be ordered to improve the performance?
Potential Solutions	(a) The smallest table should be the inner table. (b) The table with the best index should be the inner table. (c) The table with the most restrictive clause should be the outer table. (d) A `JOIN` in the `FROM` clause may beat `JOIN` in the `WHERE` clause.
Related	Complete `JOIN` Selection
References	(Celko, 2005a, Dietrich, 2001, Gulutzan and Pelzer, 2003, Shasha and Bonnet, 2003)

None of the recommendations (a) to (d) show a significant performance improvement with the Oracle® Database as shown by TQP_87201, TQP_87211 to TQP_87216, TQP_87221, and TQP_87231 respectively. The Oracle® Database appears to recognise the semantic equivalence of the queries and therefore applies identical access plans.

6.5 Summary

This chapter has examined patterns related to the more advanced SQL concepts of subqueries and joined tables. Only one of the subquery patterns showed a performance improvement with the Oracle® Database. The next chapter will present a final analysis of the found patterns and the results measured with the Oracle® Database.

Chapter 7 Analysis of the Results

This chapter analyses the results (see also Appendix D) presented in the preceding two chapters. Firstly the characteristics of the revealed patterns are discussed mainly backed by some statistical findings. A review of the research methods and tools then follows. Finally the performance improvements achieved in the Oracle® Database environment, as a result of the application of the patterns, is discussed. Guidelines to improve the SQL query execution specific to the utilised Oracle® Database are also provided.

7.1 Characteristics of the Patterns

The patterns in Chapter 5 and Chapter 6 conform to the chosen format (see Figure 2) consisting of the clauses; name, problem, potential solution(s), related, and references. Additionally, every pattern definition is followed by a detailed discussion of its effectiveness with the Oracle® Database. The patterns relate strongly to specific language features of SQL and as such can be classified as idioms or programming patterns, according to the prevalent pattern classification (amongst others: Appleton, 2000, Coplien, 1997, Derntl, 2003).

All the revealed patterns, derived mainly from the literature research, fulfil the needs defined in section 2.3 as demonstrated by the following. Alternatives as apposed to single solutions are discussed. Only four patterns are restricted to a sole solution (see column 'Solutions' in Table 5). On average (median) two solutions are provided - maximum six solutions. The pattern names are short (see column 'Words' in Table 5) with an average of three words (median) - five words maximum. These relatively short names and the uniform structure should support the easier recognition and memorisation of the patterns eventually supporting the creation of a common vocabulary. Another important aspect of the patterns is their relationship to each other as documented in the pattern section 'Related'. Each pattern may be applied as an autonomous entity but by additionally considering the related patterns, may provide the SQL query developer with further insight into the problem to be solved. Every pattern is linked to two other patterns on average (median) - maximum three links (see column 'Relations' in Table 5).

Nr	Pattern	Relations	Solutions	Words
1	ALL / ANY or EXISTS	3	2	4
2	ALL / ANY or MAX / MIN	3	4	5
3	All Rows	1	3	2
4	Avoid Explicit Sorting	1	2	3
5	Complete JOIN Selection	1	1	3
6	Defensive Projection	2	2	2
7	Distinct Rows	3	3	2
8	EXISTS or COUNT	3	2	3
9	EXISTS or IN	3	6	3
10	GROUPING SETS Beats UNION	1	2	4
11	Index Exploitation	2	4	2
12	JOIN and DISTINCT	1	1	2
13	Minimise Grouping Columns	2	1	3
14	Minimise Sorting Columns	2	2	3
15	Ordering INNER JOIN	1	4	3
16	WHERE Outperforms HAVING	1	1	3
	Total	30	40	47
	Minimum	1	1	2
	Maximum	3	6	5
	Mean	1.88	2.50	2.94
	Median	2.00	2.00	3.00

Table 5: Pattern statistics

In addition to classifying the patterns according to the related language features as in Chapter 5 and Chapter 6, the discussed solutions can be categorised according to the responsible cause for the performance problems (see Table 6). The first category contains solutions mainly dealing with the misuse of SQL language features. For example, if the SQL query developer employs unnecessarily expensive language features like `DISTINCT` or `UNION`, or doesn't restrict the number of columns to be grouped, projected, or sorted to the absolute minimum. The second category contains solutions mainly based on the query optimiser's inability to recognise semantically equivalent SQL queries and its tendency to apply different execution strategies leading to different performance behaviours for retrieving identical query results. 70% of the discussed solutions can be classified into this second category. Typical examples of this category are subqueries with a quantified comparison operator `ALL` which can be reformulated in a semantically equivalent way by applying an appropriate `NON EXISTS` predicate, or using different processing strategies of a `CUBE` clause and of the equivalent `GROUPING SETS` clause.

Nr	Pattern	Misuse of Language Features (MLF)	Non-Recognition of Semantic Equivalence (NSE)
1	ALL / ANY or EXISTS		2
2	ALL / ANY or MAX / MIN		4
3	All Rows	3	
4	Avoid Explicit Sorting	2	
5	Complete JOIN Selection		1
6	Defensive Projection	2	
7	Distinct Rows	2	1
8	EXISTS or COUNT		2
9	EXISTS or IN		6
10	GROUPING SETS Beats UNION		2
11	Index Exploitation	1	3
12	JOIN and DISTINCT		1
13	Minimise Grouping Columns		1
14	Minimise Sorting Columns	2	
15	Ordering INNER JOIN		4
16	WHERE Outperforms HAVING		1
	Total	12	28

Table 6: Categorisation of pattern solutions

7.2 Review of the Research Methods and Tools

7.2.1 Test Database and Test SQL Query Pairs

The test database instance (see also Appendix A) was mainly drawn from a productive IT service management application thereby providing an already in use approved data model and a reasonable sized amount of data both helping to avoid a biased data environment. The database parameters were modified deliberately to minimise the falsification of the measurements by caching effects. Therefore it is possible that the performance improvements in an unmodified environment could be less obvious.

The SQL query pairs were created with the intention to enable the evaluation of the pattern solutions in a context as isolated as possible. Usually several different variants related to a solution were investigated, for example, with and without the use of appropriate indexed columns. This approach, however, can only guarantee the same performance behaviour in a similar environment consisting of the same RDBMS version, a similar database instance and a comparable SQL query structure.

It should be remembered that reformulating SQL queries to improve performance is always a trade off between the software quality (Pressman, 2005) factors of correctness, maintainability and usability. Changing the SQL code bears the risk of getting a different and therefore incorrect result and this may not be obvious at a first glance. This aspect was addressed by appropriate control mechanisms built into the REPSI tool and information delivered by the Oracle tool TKPROF. Managing qualities such as maintainability and usability is more difficult because they are more subjective, dependent, for example, on the user's knowledge and experience.

Choosing a different SQL language feature, for example, a joined table instead of a subquery may affect maintainability by complicating subsequent adaptations or enhancements. It could also affect usability because the reformulated query may be better performing but be much harder to comprehend. In this dissertation maintainability and usability were addressed by keeping the pattern solutions simple, but a trade off was that more complicated algorithms such as those offered by Kim and others to flatten subqueries (Dayal, 1987, Ganski and Wong, 1987, Kim, 1982, Muralikrishna, 1992) were not considered. These are hard to employ manually and should rather be considered in the query optimiser.

7.2.2 Statistical Tools

As expected, the box plot provided a simple visual graphical representation which could easily be used in most cases to compare the performance data of two SQL queries. To do this, two requirements needed to be fulfilled. First, the notches of both box plots were required not to be overlapping (see Figure 12) thereby guaranteeing a significant difference of their two medians at about a 95% level (McGill et al., 1978). Second, the related dot chart needed to support the findings revealed by the box plots by clearly showing the same performance behaviour with the predominant majority of measurements (see Figure 13).

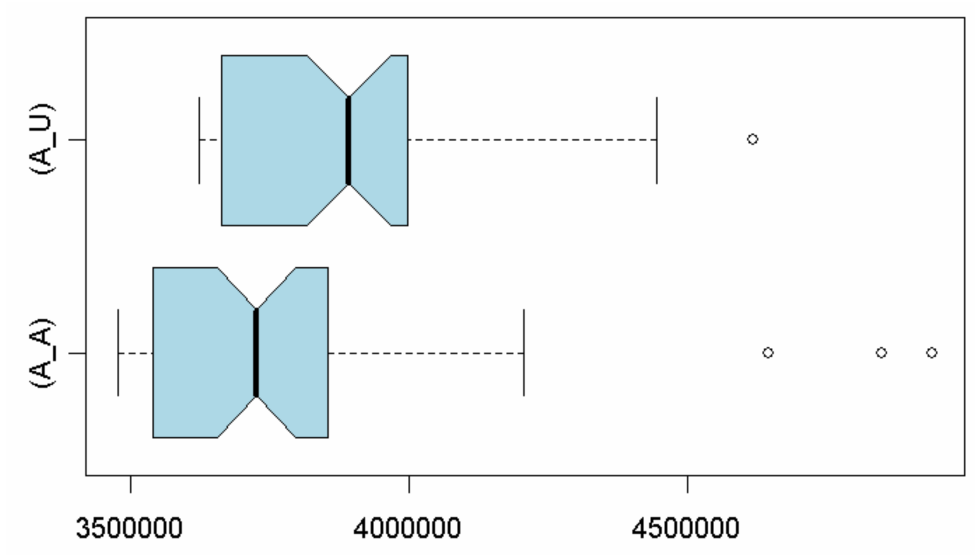


Figure 12: Box plot of TQP_80311

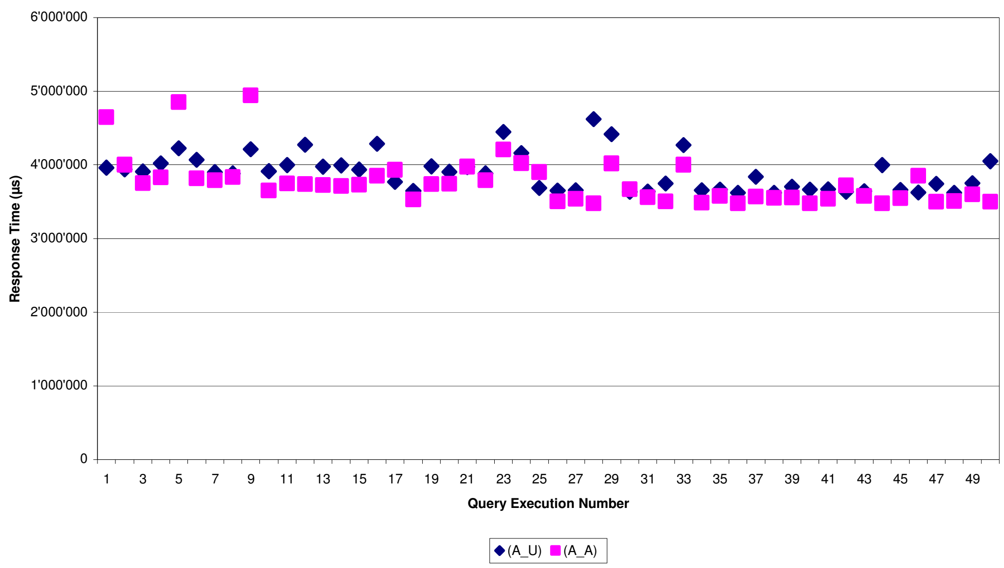


Figure 13: Dot chart of TQP_80311

This systematic worked in all cases with the exception of TQP_80431. In this case the notches of the box plot were visually so close that it was difficult to judge that the notches did not overlap (see Figure 14). However the actual results showed that the notches did not overlap which would have suggested a performance improvement: the query with the pattern applied covered the interval (760,468.4 - 766,579.6) and the query without the pattern applied covered the interval (767,237.6 - 774,475.4). But from the dot chart it was practically impossible to determine any visible difference in performance (see Figure 15). So my decision here was that the performance improvement wax not significant, despite the appearance of the box plot.

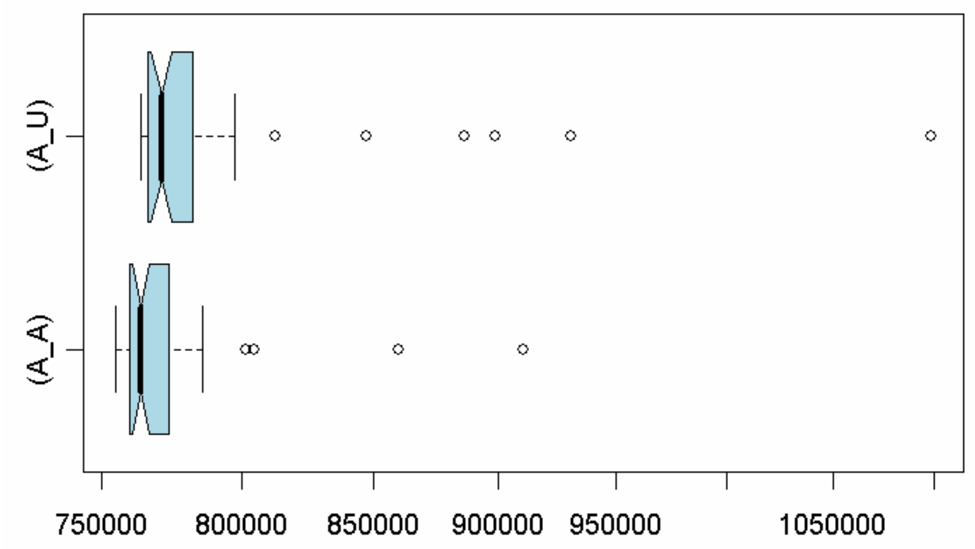


Figure 14: Box plot of TQP_80431

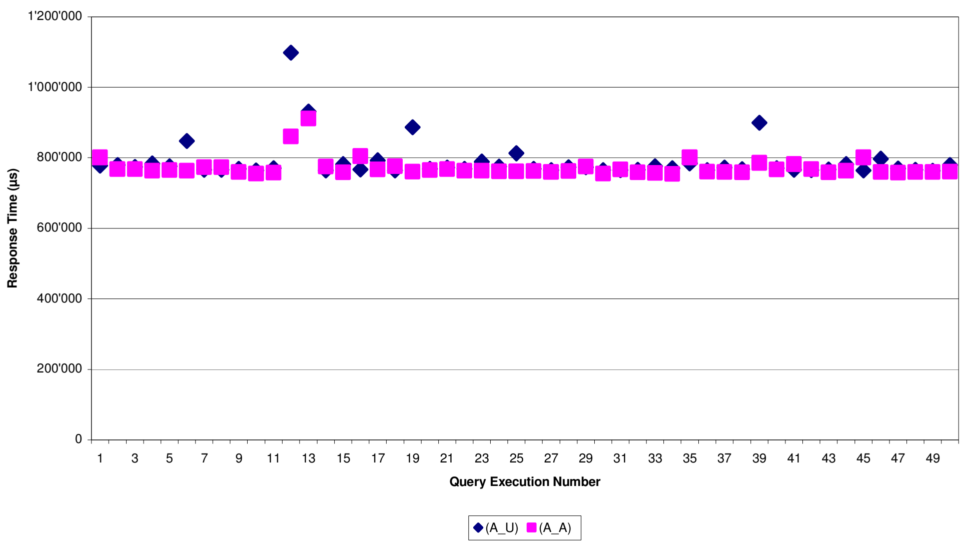


Figure 15: Dot chart of TQP_80431

7.2.3 REPSI Tool

The strengths of the REPSI tool rested on its capability to store all the relevant information required for testing the performance behaviour of the test SQL query pairs. Information stored included test database environments, pattern descriptions and their related test SQL query pairs. The REPSI tool also executed the SQL queries in a specific RDBMS and recorded the measured response times subsequently used for assessing the effectiveness of the patterns. If required, the REPSI tool could also check the equality of the result sets to prevent wrong decisions about the semantic equivalence of the SQL queries. Other important features of the REPSI tool were the different useful interfaces supporting the creation of box plots (flat files containing R code processable by the R Project for Statistical Computing tool) and dot charts (Excel style flat files processable by appropriate spreadsheet software). The downside was the somewhat cumbersome efforts to run the extra tools outside REPSI.

7.3 Performance Improvements and Derived Guidelines Specific to the Oracle® Database

As documented, in detail (see also Appendix D), a performance improvement resulted from the application of a solution from half of the patterns when applied with the chosen Oracle® Database. Table 7 shows a summary with separate columns numerating the number of solutions per pattern resulting in no performance improvement (columns titled 'No Effect') and with a performance improvement (columns titled 'Improved'). These columns are further detailed according to the solutions categorised as either a misuse of a language feature by a user (columns titled 'MLF') or a non-recognition of a semantic equivalence by the query processor (columns titled 'NSE'). For example, pattern 11 (Index Exploitation) offers four solutions, the first three of them categorised as non-recognition of semantic equivalence and one categorised as a misuse of an SQL language feature. The application of the first three solutions shows a performance improvement, and the last one has no effect. In total Table 7 reveals that 35% of the pattern solutions show a significant performance improvement when applied to the Oracle® Database and 61% of these improvements can be categorised as non-recognition of semantic equivalences. It seems therefore that the greatest opportunity for performance improvement lies in improving the ability of the database optimiser to fully recognise semantic equivalences of SQL queries.

		No Effect		Improved	
Nr	Pattern	MLF	NSE	MLF	NSE
1	`ALL` / `ANY` or `EXISTS`				2.0
2	`ALL` / `ANY` or `MAX` / `MIN`		4.0		
3	All Rows	1.3		1.7	
4	Avoid Explicit Sorting	2.0			
5	Complete `JOIN` Selection		1.0		
6	Defensive Projection	0.5		1.5	
7	Distinct Rows	0.8		1.2	1.0
8	`EXISTS` or `COUNT`		2.0		
9	`EXISTS` or `IN`		6.0		
10	`GROUPING SETS` Beats `UNION`		0.5		1.5
11	Index Exploitation	1.0			3.0
12	`JOIN` and `DISTINCT`		1.0		
13	Minimise Grouping Columns		1.0		
14	Minimise Sorting Columns	1.0		1.0	
15	Ordering `INNER JOIN`		4.0		
16	`WHERE` Outperforms `HAVING`				1.0
	Total	6.6	19.5	5.4	8.5

Table 7: Improvements with the Oracle® Database per pattern and category

The following checkpoints form a guideline with the intention to improve the performance of an SQL query executed with an Oracle® Database 10g Release 2. The sequence of the single checkpoints is according to their expected contribution to the performance improvement, starting with the biggest and ending with the smallest.

1. Check the indexed columns in the `WHERE` clause for removing applied functions from them (see pattern: Index Exploitation).

2. Check the indexed columns in the `WHERE` clause for avoiding implicit type conversions (see pattern: Index Exploitation).

3. Replace the quantified comparison operators `ALL` or `ANY` by an `EXISTS` predicate (see pattern: `ALL` / `ANY` or `EXISTS`).

4. Try to use `CUBE`, `GROUPING SETS`, or `ROLLUP` clauses instead of the `UNION` clause (see pattern: `GROUPING SETS` Beats `UNION`).

5. Remove any superfluous `DISTINCT` operators from the projections and from the aggregate functions (see patterns: All Rows, Distinct Rows).

6. Avoid the asterisk and project in general, only those columns essential for further processing (see patterns: Defensive Projection, Minimise Sorting Columns).

7. Replace `UNION` by `UNION ALL` if duplicates in the result set are not relevant (see patterns: All Rows, Distinct Rows).

8. Never select in the `HAVING` clause what can be selected in the `WHERE` clause (see pattern: `WHERE` Outperforms `HAVING`).

9. Remove any superfluous `DISTINCT` operator from the projections (see patterns: All Rows, Distinct Rows)

10. Remove '`DISTINCT *`' from the projection if all tables in the `FROM` clause contain a unique key (see patterns: All Rows, Distinct Rows).

11. Replace a '`SELECT … UNION SELECT …`' by '`SELECT DISTINCT * FROM (SELECT … UNION ALL SELECT …)`' (see pattern: Distinct Rows).

12. If possible, restrict the projected columns to the indexed ones (see pattern: Defensive Projection).

7.4 Summary

This chapter has analysed the revealed patterns and research methods and tools applied to assess the patterns' effectiveness in improving the performance of SQL queries by purely reformulating. The chapter has exposed some underlying assumptions and limitations with respect to the chosen patterns, employed methods and techniques. The next chapter will present the final conclusions and some recommendations for future work.

Chapter 8 Conclusions and Recommendations

The results demonstrated in the previous chapters show that it is indeed possible to increase the efficiency of existing relational database query optimisers by applying suitable patterns when formulating the relational database query.

8.1 Achievement of Objectives

Mainly based on literature research, 16 patterns were written and evaluated. Each pattern describes a specific performance problem related to an SQL query and then presents one or more potential solutions to improve performance by reformulating the SQL query. For each of these patterns the dissertation also delivered an appropriate set of SQL query pairs, each of them related to one of the potential solutions - one of the queries formulated without, and the other with, the application of the solution. Each of the SQL query pairs was executed by means of the REPSI tool which also recorded the response times. The chosen test database was Oracle® Database 10g Release 2, the latest version of the market leader in relational databases. In 2005, Oracle alone covered nearly 50% of the relational database market based on software revenue (Gartner, 2007).

The subsequent evaluation, mainly based on the statistical tools box plot and dot chart, showed a significant performance improvement with about a third of the potential solutions spread over half of the patterns. Based on these findings, a set of guidelines specific to Oracle® Database 10g Release 2 were developed which, when applicable for a given SQL query, should improve the performance of the query execution. The application of two thirds of the guidelines is necessary simply because the query processor is unable to recognise semantic equivalences. The remaining guidelines stem from the fact that users employ suboptimal time consuming SQL language features when formulating the SQL query.

8.2 Relevance and Implications

Minimising response time is an important aspect to be considered in developing SQL queries supporting modern business applications. Typical examples of database driven applications are order execution systems at stock exchanges (e.g. The Chicago Stock Exchange) or transaction management of credit card companies (e.g. Visa International Service Association). Other areas relevant for many internet users are web applications like online auctions (e.g. eBay Inc.), online shopping (e.g., Amazon.com, Inc.), or flight bookings (e.g. KLM Royal Dutch Airlines). These web applications typically offer personalised web sites increasing the necessity for optimal database performance.

Many of the mentioned applications apply the same queries in a very high frequency only varying the binding variables, for example, Visa claims to be "capable of processing over 6,800 transactions a second" (Visa International Service Association, 2007). This makes optimising database queries a key focus for the developers of the SQL queries.

The developer of the SQL queries must always assume responsibility for his misuse of SQL language features and the performance problems this causes. According to the declarative character of the SQL language, however, the SQL developer should have free choice in the semantically equivalent language variants when formulating the SQL query.

RDBMS specific guidelines, such as those derived for the Oracle® Database in the previous chapter, could be an important means for the SQL query developer and the RDBMS developer to increase efficiency, either by reformulating the queries, or by addressing the issues relating to the semantic equivalences in the RDBMS.

8.3 Further Work

Although the original objectives of the dissertation were achieved, more needs to be done to deliver a complete picture of the possibilities for increasing the efficiency of query optimisers by simply reformulating database queries. Three areas of further work could be undertaken.

8.3.1 Improving the Usability of the REPSI Tool

The command line interface of the REPSI tool was sufficient for the research phase of this dissertation. To extend the tool's reach to a wider user group, a graphical user interface to improve the usability is essential. Additionally, the fairly cumbersome evaluation process could be significantly improved for a broader audience, if the statistical analysis of the measurement results could be fully integrated into the REPSI tool. This would considerably ease the evaluation of the results achieved by applying a specific pattern.

8.3.2 Additional Pattern Work

The creation of the patterns in this dissertation relied mainly on an analysis of the basic elements of the `SELECT` statement along with subqueries and joined tables. The addition of new patterns and SQL query pairs reflecting not yet investigated SQL language areas, such as viewed tables, could be the next steps here. Furthermore existing patterns and their related SQL query pairs could be refined and extended as new knowledge and experience are gained.

8.3.3 Evaluation of Other RDBMSs

The remaining 50% of the commercial RDBMS market not covered by Oracle Corporation is mainly divided up by IBM Corp., Microsoft Corporation, NCR Corporation (Teradata), and Sybase Inc. (Gartner, 2007). There is also a, not to be underestimated, market of open source RDBMSs such as MySQL®, Firebird, PostgreSQL, and Sleepycat (Berkley DB) the market leaders in 2005 (Evans Data Corporation, 2007). It is clearly therefore worth extending the research to these database products. A standardised database schema along with an appropriate database instance, fully compatible with the SQL query pairs related to the patterns and easily portable to those different RDBMSs, could serve as a reference for assessing their strengths and weaknesses related to the existing performance patterns.

Appendix A – Applied Database Schema

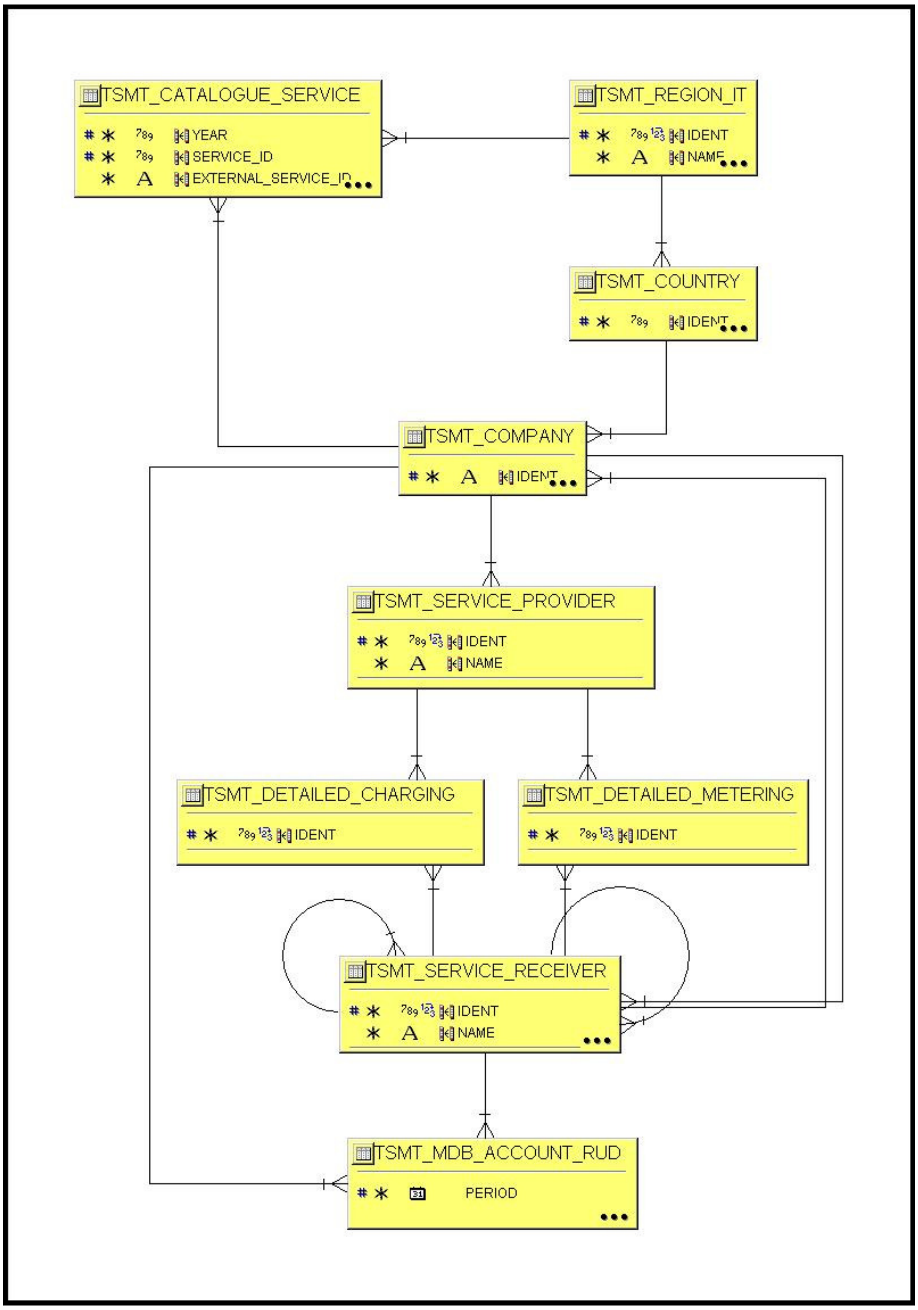


Figure 16: ERD diagram

Table TSMT_CATALOGUE_SERVICE (647 rows)

```
CREATE TABLE TSMT_CATALOGUE_SERVICE
(
  YEAR                        NUMBER(4)                                NOT NULL,
  SERVICE_ID                  NUMBER(8)                                NOT NULL,
  CREATED_BY                  VARCHAR2(30 BYTE),
  DATE_CREATED                DATE,
  DATE_MODIFIED               DATE,
  MODIFIED_BY                 VARCHAR2(30 BYTE),
  AGREED_FIXED_PRICE          CHAR(1 BYTE),
  APPROVAL_BY                 VARCHAR2(30 BYTE),
  APPROVAL_DATE               DATE,
  BUSINESS_IMPORTANCE         NUMBER,
  COMPANY_CD                  VARCHAR2(4 BYTE),
  CURRENCY_CD                 VARCHAR2(3 BYTE),
  CUSTOMER_RESTRICTION        NUMBER(8),
  DELIVERY_FROM               NUMBER(2)          DEFAULT 01,
  DELIVERY_TO                 NUMBER(2)          DEFAULT 12,
  DESCRIPTION                 VARCHAR2(4000 BYTE),
  EXTERNAL_SERVICE_ID         VARCHAR2(10 BYTE)                        NOT NULL,
  GEOGRAPHICAL_SCOPE          CHAR(1 BYTE),
  METERING_DETAIL             VARCHAR2(255 BYTE),
  NAME                        VARCHAR2(50 BYTE)                        NOT NULL,
  PRICE                       NUMBER(8,2),
  PRICE_PERIOD                CHAR(1 BYTE),
  REGION_IT_ID                NUMBER,
  SAP_CURRENT_MONTH           NUMBER(2),
  SAP_FIRST_MONTH             NUMBER(2),
  SAP_SERVICE_ID              VARCHAR2(6 BYTE),
  SERVICE_PROVIDER_HIERARCHY  VARCHAR2(10 BYTE),
  SERVICE_SHEET               VARCHAR2(255 BYTE),
  SERVICE_TYPE                CHAR(2 BYTE)                             NOT NULL,
  SOURCE_CREATED_BY           VARCHAR2(30 BYTE),
  SOURCE_DATE_CREATED         DATE,
  SOURCE_DATE_MODIFIED        DATE,
  SOURCE_MODIFIED_BY          VARCHAR2(30 BYTE),
  UNIT                        NUMBER                                   NOT NULL,
  METERING_METHOD             VARCHAR2(30 BYTE),
  METERING_RESPONSIBLE        VARCHAR2(8 BYTE),
  SERVICE_CATEGORY            VARCHAR2(30 BYTE),
  PRODUCT_DESCRIPTION_ID      NUMBER,
  SERVICE_GROUP               VARCHAR2(30 BYTE),
  PRICE_TYPE                  CHAR(1 BYTE)       DEFAULT 'U'           NOT NULL
)

CREATE UNIQUE INDEX UNQ_T_SMT_CSE ON TSMT_CATALOGUE_SERVICE (YEAR,
EXTERNAL_SERVICE_ID);

CREATE UNIQUE INDEX PK_T_SMT_CSE ON TSMT_CATALOGUE_SERVICE (YEAR, SERVICE_ID);

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1150889674_PRICE_000
 CHECK (PRICE_TYPE IN ('A', 'U')));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1140432530_GEOGR_000
 CHECK (GEOGRAPHICAL_SCOPE IN ('G', 'L', 'R')));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1076015607_SAP_F_001
 CHECK (SAP_FIRST_MONTH BETWEEN 1 AND 12));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1076015607_SAP_C_001
 CHECK (SAP_CURRENT_MONTH BETWEEN 1 AND 12));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1076015607_DELIV_003
 CHECK (DELIVERY_TO BETWEEN 1 AND 12));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1076015607_DELIV_002
 CHECK (DELIVERY_FROM BETWEEN 1 AND 12));
```

```
ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT AVCON_1076015607_AGREE_000
 CHECK (AGREED_FIXED_PRICE IN ('N', 'Y')));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT PK_T_SMT_CSE
 PRIMARY KEY (YEAR, SERVICE_ID));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT UNQ_T_SMT_CSE
 UNIQUE (YEAR, EXTERNAL_SERVICE_ID));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_SC
 FOREIGN KEY (SERVICE_CATEGORY)
 REFERENCES TSMT_SERVICE_CATEGORY (SERVICE_CATEGORY));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_CPTD
 FOREIGN KEY (YEAR, PRODUCT_DESCRIPTION_ID)
 REFERENCES TSMT_CATALOGUE_PRODUCT_DESCRIP (YEAR,PRODUCT_DESCRIPTION_ID));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_SEG
 FOREIGN KEY (SERVICE_GROUP)
 REFERENCES TSMT_SERVICE_GROUP (SERVICE_GROUP));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_MR
 FOREIGN KEY (METERING_RESPONSIBLE)
 REFERENCES TSMT_MANAGER (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_SPH
 FOREIGN KEY (SERVICE_PROVIDER_HIERARCHY)
 REFERENCES TSMT_SERVICE_PROVIDER_HIERARCH (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_COMY
 FOREIGN KEY (COMPANY_CD)
 REFERENCES TSMT_COMPANY (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_RI
 FOREIGN KEY (REGION_IT_ID)
 REFERENCES TSMT_REGION_IT (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_STY
 FOREIGN KEY (SERVICE_TYPE)
 REFERENCES TSMT_SERVICE_TYPE (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_QU
 FOREIGN KEY (UNIT)
 REFERENCES TSMT_QUANTITY_UNIT (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_BI
 FOREIGN KEY (BUSINESS_IMPORTANCE)
 REFERENCES TSMT_BUSINESS_IMPORTANCE (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_CUY
 FOREIGN KEY (CURRENCY_CD)
 REFERENCES TSMT_CURRENCY (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_GS
 FOREIGN KEY (GEOGRAPHICAL_SCOPE)
 REFERENCES TSMT_GEOGRAPHICAL_SCOPE (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_PP
 FOREIGN KEY (PRICE_PERIOD)
```

```
 REFERENCES TSMT_PRICE_PERIOD (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_YR
 FOREIGN KEY (YEAR)
 REFERENCES TSMT_YEAR (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_CR
 FOREIGN KEY (CUSTOMER_RESTRICTION)
 REFERENCES TSMT_CUSTOMER_RESTRICTION (IDENT));

ALTER TABLE TSMT_CATALOGUE_SERVICE ADD (
  CONSTRAINT FK_T_SMT_CSE_T_SMT_SSI
 FOREIGN KEY (SAP_SERVICE_ID)
 REFERENCES TSMT_SAP_SERVICE_ID (SAP_SERVICE_ID));
```

Table TSMT_COMPANY (308 rows)

```
CREATE TABLE TSMT_COMPANY
(
  IDENT                       VARCHAR2(4 BYTE)                                  NOT NULL,
  CREATED_BY                  VARCHAR2(30 BYTE),
  DATE_CREATED                DATE,
  MODIFIED_BY                 VARCHAR2(30 BYTE),
  DATE_MODIFIED               DATE,
  LAST_TRANSFER               DATE,
  CITY                        VARCHAR2(25 BYTE),
  COUNTRY_ID                  NUMBER(3),
  LEGAL_NAME                  VARCHAR2(55 BYTE),
  OBSOLETE                    CHAR(1 BYTE)        DEFAULT 'N',
  SAPINST_NO                  NUMBER(4),
  SHORT_NAME                  VARCHAR2(35 BYTE),
  VALID_IT_SMART              CHAR(1 BYTE)        DEFAULT 'Y',
  CURRENCY_CD                 VARCHAR2(3 BYTE)                                  NOT NULL,
  CURRENCY_CD_IT              VARCHAR2(3 BYTE)                                  NOT NULL,
  LANDLORD_ODG_NO             VARCHAR2(2 BYTE),
  CONTRACTING_MODE            CHAR(3 BYTE)        DEFAULT 'n/a',
  CONTRACTING_SERVICE         NUMBER(8),
  HIERARCHY_FLAG_RUD          CHAR(1 BYTE)        DEFAULT 'N',
  DEFAULT_SERVICE_RECEIVER    NUMBER,
  ADDRESS_LINE1               VARCHAR2(50 BYTE),
  ADDRESS_LINE2               VARCHAR2(50 BYTE),
  ADDRESS_LINE3               VARCHAR2(50 BYTE),
  ADDRESS_LINE4               VARCHAR2(50 BYTE),
  ADDRESS_LINE5               VARCHAR2(50 BYTE),
  ADDRESS_LINE6               VARCHAR2(50 BYTE),
  COST_CENTER_GLOBAL_VOICE    VARCHAR2(10 BYTE),
  LANDLORD_RULE_FLAG          CHAR(1 BYTE)        DEFAULT 'Y',
  GEOGRAPHICAL_SCOPE          CHAR(1 BYTE);

CREATE UNIQUE INDEX PK_T_SMT_COMY ON TSMT_COMPANY (IDENT);

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT AVCON_1143737005_GEOGR_015
 CHECK (GEOGRAPHICAL_SCOPE IN ('G', 'L', 'R')));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT AVCON_1068106159_VALID_005
 CHECK (VALID_IT_SMART IN ('N', 'Y')));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT AVCON_1068106159_OBSOL_006
 CHECK (OBSOLETE IN ('N', 'Y')));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT AVCON_1114614594_LANDL_000
 CHECK (LANDLORD_RULE_FLAG IN ('N', 'Y')));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT AVCON_1088406510_HIERA_000
 CHECK (HIERARCHY_FLAG_RUD IN ('N', 'R')));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT AVCON_1078504753_CONTR_001
 CHECK (CONTRACTING_MODE IN ('GMA', 'GMI', 'MAN', 'n/a')));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT PK_T_SMT_COMY
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_CC
 FOREIGN KEY (SAPINST_NO, COST_CENTER_GLOBAL_VOICE)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_GS
 FOREIGN KEY (GEOGRAPHICAL_SCOPE)
 REFERENCES TSMT_GEOGRAPHICAL_SCOPE (IDENT));
```

```
ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_CUY_IT
 FOREIGN KEY (CURRENCY_CD_IT)
 REFERENCES TSMT_CURRENCY (IDENT));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_COUY
 FOREIGN KEY (COUNTRY_ID)
 REFERENCES TSMT_COUNTRY (IDENT));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_SE
 FOREIGN KEY (CONTRACTING_SERVICE)
 REFERENCES TSMT_SERVICE (SERVICE_ID));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_OG
 FOREIGN KEY (LANDLORD_ODG_NO)
 REFERENCES TSMT_ODG (IDENT));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_CUY
 FOREIGN KEY (CURRENCY_CD)
 REFERENCES TSMT_CURRENCY (IDENT));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_SI
 FOREIGN KEY (SAPINST_NO)
 REFERENCES TSMT_SAP_INSTANCE (IDENT));

ALTER TABLE TSMT_COMPANY ADD (
  CONSTRAINT FK_T_SMT_COMY_T_SMT_SR
 FOREIGN KEY (DEFAULT_SERVICE_RECEIVER)
 REFERENCES TSMT_SERVICE_RECEIVER (IDENT));
```

Table TSMT_COUNTRY (250 rows)

```
CREATE TABLE TSMT_COUNTRY
(
  IDENT                 NUMBER(3)                                            NOT NULL,
  CREATED_BY            VARCHAR2(30 BYTE),
  DATE_CREATED          DATE,
  MODIFIED_BY           VARCHAR2(30 BYTE),
  DATE_MODIFIED         DATE,
  LAST_TRANSFER         DATE,
  LOCAL_CURRENCY_CD     VARCHAR2(3 BYTE)                                     NOT NULL,
  LOCAL_CURRENCY_CD_IT  VARCHAR2(3 BYTE)                                     NOT NULL,
  NAME                  VARCHAR2(40 BYTE),
  OBSOLETE              CHAR(1 BYTE)            DEFAULT 'N',
  REGION_IT_ID          NUMBER                                               NOT NULL,
  VALID_IT_SMART        CHAR(1 BYTE)            DEFAULT 'N'
);

CREATE UNIQUE INDEX PK_T_SMT_COUY ON TSMT_COUNTRY (IDENT);

ALTER TABLE TSMT_COUNTRY ADD (
  CONSTRAINT AVCON_1068106159_OBSOL_003
 CHECK (OBSOLETE IN ('N', 'Y')));

ALTER TABLE TSMT_COUNTRY ADD (
  CONSTRAINT AVCON_1068106159_VALID_002
 CHECK (VALID_IT_SMART IN ('N', 'Y')));

ALTER TABLE TSMT_COUNTRY ADD (
  CONSTRAINT PK_T_SMT_COUY
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_COUNTRY ADD (
  CONSTRAINT FK_T_SMT_COUY_T_SMT_CUY
 FOREIGN KEY (LOCAL_CURRENCY_CD)
 REFERENCES TSMT_CURRENCY (IDENT));

ALTER TABLE TSMT_COUNTRY ADD (
  CONSTRAINT FK_T_SMT_COUY_T_SMT_RI
 FOREIGN KEY (REGION_IT_ID)
 REFERENCES TSMT_REGION_IT (IDENT));

ALTER TABLE TSMT_COUNTRY ADD (
  CONSTRAINT FK_T_SMT_COUY_T_SMT_CUY_IT
 FOREIGN KEY (LOCAL_CURRENCY_CD_IT)
 REFERENCES TSMT_CURRENCY (IDENT));
```

Table TSMT_DETAILED_CHARGING (33,183 rows)

```
CREATE TABLE TSMT_DETAILED_CHARGING
(
  IDENT                           NUMBER(8)                                  NOT NULL,
  CREATED_BY                      VARCHAR2(30 BYTE),
  DATE_CREATED                    DATE,
  AMOUNT                          NUMBER(12,2)                               NOT NULL,
  AMOUNT_ROCHE_CURRENCY           NUMBER(12,2)                               NOT NULL,
  CHARGING_METHOD                 VARCHAR2(30 BYTE)                          NOT NULL,
  COMPONENT_ID                    NUMBER                                     NOT NULL,
  CONTRACT_ID                     NUMBER(8),
  COST_CENTER_CD_CROSS            VARCHAR2(10 BYTE),
  COST_CENTER_CD_PROVIDER         VARCHAR2(10 BYTE),
  COST_CENTER_CD_RECEIVER         VARCHAR2(10 BYTE),
  CURRENCY_CD                     VARCHAR2(3 BYTE)                           NOT NULL,
  PERIOD                          DATE                                       NOT NULL,
  QUANTITY                        NUMBER(16,6)   DEFAULT 1                   NOT NULL,
  REQUEST_ID                      NUMBER(8),
  SAP_SERVICE_ID                  VARCHAR2(6 BYTE)                           NOT NULL,
  SAPINST_NO                      NUMBER(4)                                  NOT NULL,
  SERVICE_ID                      NUMBER(8)                                  NOT NULL,
  SERVICE_PROVIDER_ID             NUMBER                                     NOT NULL,
  SERVICE_RECEIVER_ID             NUMBER                                     NOT NULL,
  SIGN                            CHAR(1 BYTE)   DEFAULT '+'                 NOT NULL,
  SLA_ID                          NUMBER(8),
  CHARGING_TYPE                   CHAR(1 BYTE)   DEFAULT 'C',
  SERVICE_PROVIDER_HIERARCHY      VARCHAR2(10 BYTE),
  AMOUNT_SERVICE                  NUMBER(12,2),
  AMOUNT_SERVICE_ROCHE_CURRENCY   NUMBER(12,2),
  COST_CENTER_CD_DISCHARGING      VARCHAR2(10 BYTE),
  ELEMENT_ID                      NUMBER(8)                                  NOT NULL,
  QUANTITY_SERVICE                NUMBER(16,6)   DEFAULT 1                   NOT NULL,
  MODIFIED_BY                     VARCHAR2(30 BYTE),
  DATE_MODIFIED                   DATE,
  MANUAL_FLAG                     CHAR(1 BYTE)   DEFAULT 'N',
  SAP_ORDER_NUMBER                VARCHAR2(12 BYTE)
)

CREATE INDEX IDX_T_SMT_DC_SERVICE_RECEIVER ON TSMT_DETAILED_CHARGING
(SERVICE_RECEIVER_ID);

CREATE UNIQUE INDEX PK_T_SMT_DC ON TSMT_DETAILED_CHARGING (IDENT);

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT AVCON_1087969481_CHARG_002
 CHECK (CHARGING_TYPE IN ('C', 'I', 'P', 'S')));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT AVCON_1106030929_MANUA_000
 CHECK (MANUAL_FLAG IN ('N', 'Y')));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT AVCON_1082665162_SIGN_001
 CHECK (SIGN IN ('+', '-')));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT PK_T_SMT_DC
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CC_DISCHARG
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_DISCHARGING)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_SPH
 FOREIGN KEY (SERVICE_PROVIDER_HIERARCHY)
 REFERENCES TSMT_SERVICE_PROVIDER_HIERARCH (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CC_PROVIDER
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_PROVIDER)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));
```

```
ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_SI
 FOREIGN KEY (SAPINST_NO)
 REFERENCES TSMT_SAP_INSTANCE (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_PD
 FOREIGN KEY (PERIOD)
 REFERENCES TSMT_PERIOD (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CUY
 FOREIGN KEY (CURRENCY_CD)
 REFERENCES TSMT_CURRENCY (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CC_CROSS
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_CROSS)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CC_RECEIVER
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_RECEIVER)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_GCE_RT
 FOREIGN KEY (REQUEST_ID)
 REFERENCES TGCE_REQUEST (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_SP
 FOREIGN KEY (SERVICE_PROVIDER_ID)
 REFERENCES TSMT_SERVICE_PROVIDER (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_SR
 FOREIGN KEY (SERVICE_RECEIVER_ID)
 REFERENCES TSMT_SERVICE_RECEIVER (IDENT));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CN
 FOREIGN KEY (CONTRACT_ID)
 REFERENCES TSMT_CONTRACT (CONTRACT_ID));

ALTER TABLE TSMT_DETAILED_CHARGING ADD (
  CONSTRAINT FK_T_SMT_DC_T_SMT_CTY
 FOREIGN KEY (CHARGING_TYPE)
 REFERENCES TSMT_CHARGING_TYPE (IDENT));
```

Table TSMT_DETAILED_METERING (102,583 rows)

```
CREATE TABLE TSMT_DETAILED_METERING
(
  IDENT                          NUMBER(8)                                  NOT NULL,
  CREATED_BY                     VARCHAR2(30 BYTE),
  DATE_CREATED                   DATE,
  AMOUNT                         NUMBER(12,2)                               NOT NULL,
  AMOUNT_ROCHE_CURRENCY          NUMBER(12,2)                               NOT NULL,
  COMPONENT_ID                   NUMBER,
  CONTRACT_ID                    NUMBER(8),
  COST_CENTER_CD_CROSS           VARCHAR2(10 BYTE),
  COST_CENTER_CD_PROVIDER        VARCHAR2(10 BYTE),
  COST_CENTER_CD_RECEIVER        VARCHAR2(10 BYTE),
  CURRENCY_CD                    VARCHAR2(3 BYTE)                           NOT NULL,
  DETAILED_INFORMATION           VARCHAR2(4000 BYTE),
  METERING_METHOD                VARCHAR2(30 BYTE)                          NOT NULL,
  PERIOD                         DATE                                       NOT NULL,
  QUANTITY                       NUMBER(16,6)   DEFAULT 1                   NOT NULL,
  REQUEST_ID                     NUMBER(8),
  SAP_SERVICE_ID                 VARCHAR2(6 BYTE),
  SAPINST_NO                     NUMBER(4)                                  NOT NULL,
  SERVICE_ID                     NUMBER(8)                                  NOT NULL,
  SERVICE_PROVIDER_ID            NUMBER,
  SERVICE_RECEIVER_ID            NUMBER                                     NOT NULL,
  SIGN                           CHAR(1 BYTE)   DEFAULT '+'                 NOT NULL,
  SLA_ID                         NUMBER(8),
  CHARGING_TYPE                  CHAR(1 BYTE)   DEFAULT 'C'                 NOT NULL,
  SERVICE_PROVIDER_HIERARCHY     VARCHAR2(10 BYTE),
  SAP_ORDER_NUMBER               VARCHAR2(12 BYTE),
  AMOUNT_SERVICE                 NUMBER(12,2),
  AMOUNT_SERVICE_ROCHE_CURRENCY  NUMBER(12,2),
  COST_CENTER_CD_DISCHARGING     VARCHAR2(10 BYTE),
  ELEMENT_ID                     NUMBER(8),
  QUANTITY_SERVICE               NUMBER(16,6)   DEFAULT 1,
  PSP_ELEMENT                    VARCHAR2(24 BYTE)
);

CREATE INDEX IDX_T_SMT_DM_SAP_ORDER_NUMBER ON TSMT_DETAILED_METERING (SAPINST_NO,
SAP_ORDER_NUMBER);

CREATE UNIQUE INDEX PK_T_SMT_DM ON TSMT_DETAILED_METERING (IDENT);

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT AVCON_1087969481_CHARG_000
 CHECK (CHARGING_TYPE IN ('C', 'I', 'P', 'S')));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT AVCON_1082665753_SIGN_000
 CHECK (SIGN IN ('+', '-')));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT PK_T_SMT_DM
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_SON
 FOREIGN KEY (SAPINST_NO, SAP_ORDER_NUMBER)
 REFERENCES TSMT_SAP_ORDER_NUMBER (SAPINST_NO,SAP_ORDER_NUMBER));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CC_DISCHARG
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_DISCHARGING)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_SPH
 FOREIGN KEY (SERVICE_PROVIDER_HIERARCHY)
 REFERENCES TSMT_SERVICE_PROVIDER_HIERARCH (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CC_CROSS
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_CROSS)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));
```

```
ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CTY
 FOREIGN KEY (CHARGING_TYPE)
 REFERENCES TSMT_CHARGING_TYPE (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_SSI
 FOREIGN KEY (SAP_SERVICE_ID)
 REFERENCES TSMT_SAP_SERVICE_ID (SAP_SERVICE_ID));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_PD
 FOREIGN KEY (PERIOD)
 REFERENCES TSMT_PERIOD (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CUY
 FOREIGN KEY (CURRENCY_CD)
 REFERENCES TSMT_CURRENCY (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CN
 FOREIGN KEY (CONTRACT_ID)
 REFERENCES TSMT_CONTRACT (CONTRACT_ID));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_GCE_RT
 FOREIGN KEY (REQUEST_ID)
 REFERENCES TGCE_REQUEST (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CC_PROVIDER
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_PROVIDER)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_SI
 FOREIGN KEY (SAPINST_NO)
 REFERENCES TSMT_SAP_INSTANCE (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_SP
 FOREIGN KEY (SERVICE_PROVIDER_ID)
 REFERENCES TSMT_SERVICE_PROVIDER (IDENT));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_CC_RECEIVER
 FOREIGN KEY (SAPINST_NO, COST_CENTER_CD_RECEIVER)
 REFERENCES TSMT_COST_CENTER (SAPINST_NO,COST_CENTER_CD));

ALTER TABLE TSMT_DETAILED_METERING ADD (
  CONSTRAINT FK_T_SMT_DM_T_SMT_SR
 FOREIGN KEY (SERVICE_RECEIVER_ID)
 REFERENCES TSMT_SERVICE_RECEIVER (IDENT));
```

Table TSMT_MDB_ACCOUNT_RUD (1,945,431 rows)

```
CREATE TABLE TSMT_MDB_ACCOUNT_RUD
(
  PERIOD               DATE                                                   NOT NULL,
  USER_ID              VARCHAR2(8 BYTE)                                       NOT NULL,
  LAST_TRANSFER        DATE,
  CATEGORY             NUMBER(1),
  COMPANY_CD           VARCHAR2(4 BYTE)                                       NOT NULL,
  DEPARTMENT_TXT       VARCHAR2(40 BYTE),
  DIVISION_NO          VARCHAR2(2 BYTE)                                       NOT NULL,
  SERVICE_RECEIVER_ID  NUMBER                                                 NOT NULL,
  COST_CENTER_CD       VARCHAR2(10 BYTE),
  SAPINST_NO           NUMBER(4),
  CREATED_BY           VARCHAR2(30 BYTE),
  DATE_CREATED         DATE,
  MODIFIED_BY          VARCHAR2(30 BYTE),
  DATE_MODIFIED        DATE,
  HIERARCHY_CD         VARCHAR2(10 BYTE),
  LOCATION_CD          VARCHAR2(3 BYTE)
);

CREATE INDEX T_SMT_MAR_COMPANY_CD ON TSMT_MDB_ACCOUNT_RUD (COMPANY_CD);

CREATE UNIQUE INDEX PK_T_SMT_MAR ON TSMT_MDB_ACCOUNT_RUD (PERIOD, USER_ID);

CREATE INDEX T_SMT_MAR_PERIOD ON TSMT_MDB_ACCOUNT_RUD (PERIOD);

CREATE INDEX T_SMT_MAR_PD_SERVICE_RECEIVER ON TSMT_MDB_ACCOUNT_RUD (PERIOD,
SERVICE_RECEIVER_ID);

CREATE INDEX T_SMT_MAR_PD_COMY_COST_CENTER ON TSMT_MDB_ACCOUNT_RUD (PERIOD,
COMPANY_CD, COST_CENTER_CD);

ALTER TABLE TSMT_MDB_ACCOUNT_RUD ADD (
  CONSTRAINT PK_T_SMT_MAR
 PRIMARY KEY (PERIOD, USER_ID));

ALTER TABLE TSMT_MDB_ACCOUNT_RUD ADD (
  CONSTRAINT FK_T_SMT_MAR_T_SMT_DN
 FOREIGN KEY (DIVISION_NO)
 REFERENCES TSMT_DIVISION (IDENT));

ALTER TABLE TSMT_MDB_ACCOUNT_RUD ADD (
  CONSTRAINT FK_T_SMT_MAR_T_SMT_PD
 FOREIGN KEY (PERIOD)
 REFERENCES TSMT_PERIOD (IDENT));

ALTER TABLE TSMT_MDB_ACCOUNT_RUD ADD (
  CONSTRAINT FK_T_SMT_MAR_T_SMT_COMY
 FOREIGN KEY (COMPANY_CD)
 REFERENCES TSMT_COMPANY (IDENT));

ALTER TABLE TSMT_MDB_ACCOUNT_RUD ADD (
  CONSTRAINT FK_T_SMT_MAR_T_SMT_SI
 FOREIGN KEY (SAPINST_NO)
 REFERENCES TSMT_SAP_INSTANCE (IDENT));
```

Table TSMT_REGION_IT (4 rows)

```
CREATE TABLE TSMT_REGION_IT
(
  IDENT          NUMBER                                                 NOT NULL,
  CREATED_BY     VARCHAR2(30 BYTE),
  DATE_CREATED   DATE,
  MODIFIED_BY    VARCHAR2(30 BYTE),
  DATE_MODIFIED  DATE,
  NAME           VARCHAR2(30 BYTE)                                      NOT NULL
);

CREATE UNIQUE INDEX PK_T_SMT_RI ON TSMT_REGION_IT (IDENT);

CREATE UNIQUE INDEX UNQ_T_SMT_RI ON TSMT_REGION_IT (NAME);

ALTER TABLE TSMT_REGION_IT ADD (
  CONSTRAINT PK_T_SMT_RI
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_REGION_IT ADD (
  CONSTRAINT UNQ_T_SMT_RI
 UNIQUE (NAME));
```

Table TSMT_SERVICE_PROVIDER (132 rows)

```
CREATE TABLE TSMT_SERVICE_PROVIDER
(
  IDENT                     NUMBER                                              NOT NULL,
  CREATED_BY                VARCHAR2(30 BYTE),
  DATE_CREATED              DATE,
  MODIFIED_BY               VARCHAR2(30 BYTE),
  DATE_MODIFIED             DATE,
  COMPANY_CD                VARCHAR2(4 BYTE)                                    NOT NULL,
  COST_CENTER_CD            VARCHAR2(10 BYTE),
  COST_ELEMENT_CD           VARCHAR2(10 BYTE),
  GEOGRAPHICAL_SCOPE        CHAR(1 BYTE)         DEFAULT 'L'                    NOT NULL,
  HIERARCHY_CD              VARCHAR2(10 BYTE)                                   NOT NULL,
  NAME                      VARCHAR2(50 BYTE)                                   NOT NULL,
  RESPONSIBLE_APPROVAL      VARCHAR2(8 BYTE)                                    NOT NULL,
  RESPONSIBLE_METERING      VARCHAR2(8 BYTE)                                    NOT NULL,
  RESPONSIBLE_OPERATION     VARCHAR2(8 BYTE),
  SAPINST_NO                NUMBER(4),
  COST_CENTER_CD_PROPOSAL   VARCHAR2(10 BYTE)
)

CREATE UNIQUE INDEX PK_T_SMT_SP ON TSMT_SERVICE_PROVIDER (IDENT);

CREATE UNIQUE INDEX UNQ_T_SMT_SP ON TSMT_SERVICE_PROVIDER (NAME);

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT AVCON_1140432530_GEOGR_015
 CHECK (GEOGRAPHICAL_SCOPE IN ('G', 'L', 'R')));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT PK_T_SMT_SP
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT UNQ_T_SMT_SP
 UNIQUE (NAME));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_GS
 FOREIGN KEY (GEOGRAPHICAL_SCOPE)
 REFERENCES TSMT_GEOGRAPHICAL_SCOPE (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_MR_METERING
 FOREIGN KEY (RESPONSIBLE_METERING)
 REFERENCES TSMT_MANAGER (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_MR_APPROVAL
 FOREIGN KEY (RESPONSIBLE_APPROVAL)
 REFERENCES TSMT_MANAGER (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_MR_OPERATION
 FOREIGN KEY (RESPONSIBLE_OPERATION)
 REFERENCES TSMT_MANAGER (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_COMY
 FOREIGN KEY (COMPANY_CD)
 REFERENCES TSMT_COMPANY (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_SI
 FOREIGN KEY (SAPINST_NO)
 REFERENCES TSMT_SAP_INSTANCE (IDENT));

ALTER TABLE TSMT_SERVICE_PROVIDER ADD (
  CONSTRAINT FK_T_SMT_SP_T_SMT_SPH
 FOREIGN KEY (HIERARCHY_CD)
 REFERENCES TSMT_SERVICE_PROVIDER_HIERARCH (IDENT));
```

Table TSMT_SERVICE_RECEIVER (831 rows)

```
CREATE TABLE TSMT_SERVICE_RECEIVER
(
  IDENT                        NUMBER                                          NOT NULL,
  CREATED_BY                   VARCHAR2(30 BYTE),
  DATE_CREATED                 DATE,
  MODIFIED_BY                  VARCHAR2(30 BYTE),
  DATE_MODIFIED                DATE,
  COMPANY_CD                   VARCHAR2(4 BYTE)                                NOT NULL,
  COST_CENTER_CD               VARCHAR2(10 BYTE),
  DIVISION_NO                  VARCHAR2(2 BYTE)                                NOT NULL,
  HIERARCHY_CD                 VARCHAR2(10 BYTE),
  NAME                         VARCHAR2(50 BYTE)                               NOT NULL,
  RESPONSIBLE_CONTRACTING      VARCHAR2(8 BYTE)                                NOT NULL,
  SAPINST_NO                   NUMBER(4),
  CENTER_TYPE_FLAG             CHAR(1 BYTE)      DEFAULT 'N',
  STATISTICAL_FLAG             CHAR(1 BYTE)      DEFAULT 'N',
  CONTRACT_PROPOSAL_RECEIVER   NUMBER,
  REPORTING_RECEIVER           NUMBER,
  REGION_IT_DIA_CD             VARCHAR2(6 BYTE)
);

CREATE UNIQUE INDEX PK_T_SMT_SR ON TSMT_SERVICE_RECEIVER (IDENT);

CREATE UNIQUE INDEX UNQ_T_SMT_SR_NAME ON TSMT_SERVICE_RECEIVER (NAME);

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT AVCON_1082127640_CENTE_000
 CHECK (CENTER_TYPE_FLAG IN ('C', 'N', 'R')));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT AVCON_1082127640_STATI_000
 CHECK (STATISTICAL_FLAG IN ('N', 'Y')));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT PK_T_SMT_SR
 PRIMARY KEY (IDENT));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT UNQ_T_SMT_SR_NAME
 UNIQUE (NAME));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_SR_RG
 FOREIGN KEY (REPORTING_RECEIVER)
 REFERENCES TSMT_SERVICE_RECEIVER (IDENT));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_SR_CP
 FOREIGN KEY (CONTRACT_PROPOSAL_RECEIVER)
 REFERENCES TSMT_SERVICE_RECEIVER (IDENT));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_RID
 FOREIGN KEY (REGION_IT_DIA_CD)
 REFERENCES TSMT_REGION_IT_DIA (REGION_IT_DIA_CD));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_SRH
 FOREIGN KEY (DIVISION_NO, HIERARCHY_CD)
 REFERENCES TSMT_SERVICE_RECEIVER_HIERARCH (DIVISION_NO,HIERARCHY_CD));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_DN
 FOREIGN KEY (DIVISION_NO)
 REFERENCES TSMT_DIVISION (IDENT));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_MR
 FOREIGN KEY (RESPONSIBLE_CONTRACTING)
 REFERENCES TSMT_MANAGER (IDENT));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
```

```
  CONSTRAINT FK_T_SMT_SR_T_SMT_COMY
 FOREIGN KEY (COMPANY_CD)
 REFERENCES TSMT_COMPANY (IDENT));

ALTER TABLE TSMT_SERVICE_RECEIVER ADD (
  CONSTRAINT FK_T_SMT_SR_T_SMT_SI
 FOREIGN KEY (SAPINST_NO)
 REFERENCES TSMT_SAP_INSTANCE (IDENT));
```

Appendix B – Detailed Measurement Results Based on Patterns

ALL / ANY or EXISTS

TQP_86201: A query of the form `col1 comp ALL (SELECT col2 FROM … WHERE cond)` can be rewritten as `NOT EXISTS (SELECT … FROM … WHERE cond AND col1 negatived_comp col2)`

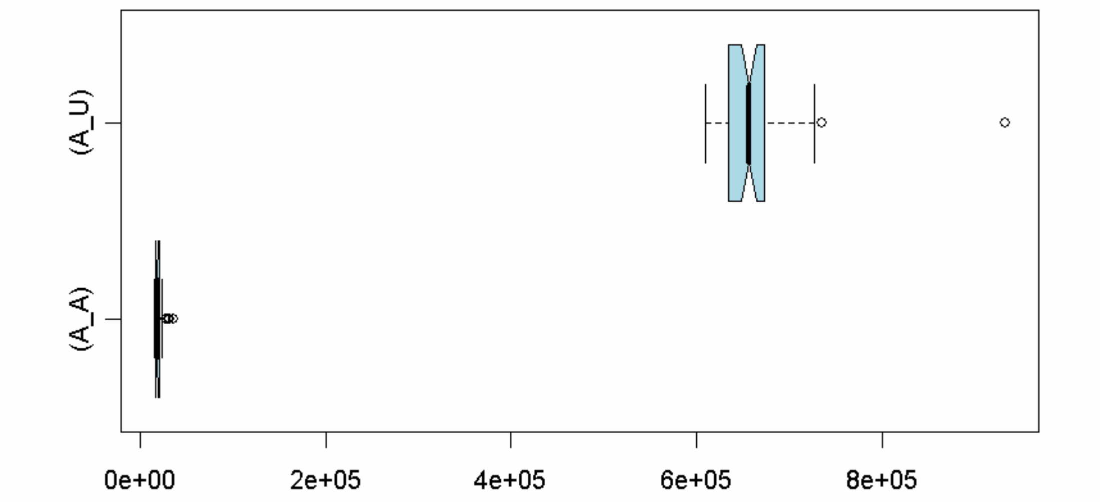
Median 656,700 µs (A_U):

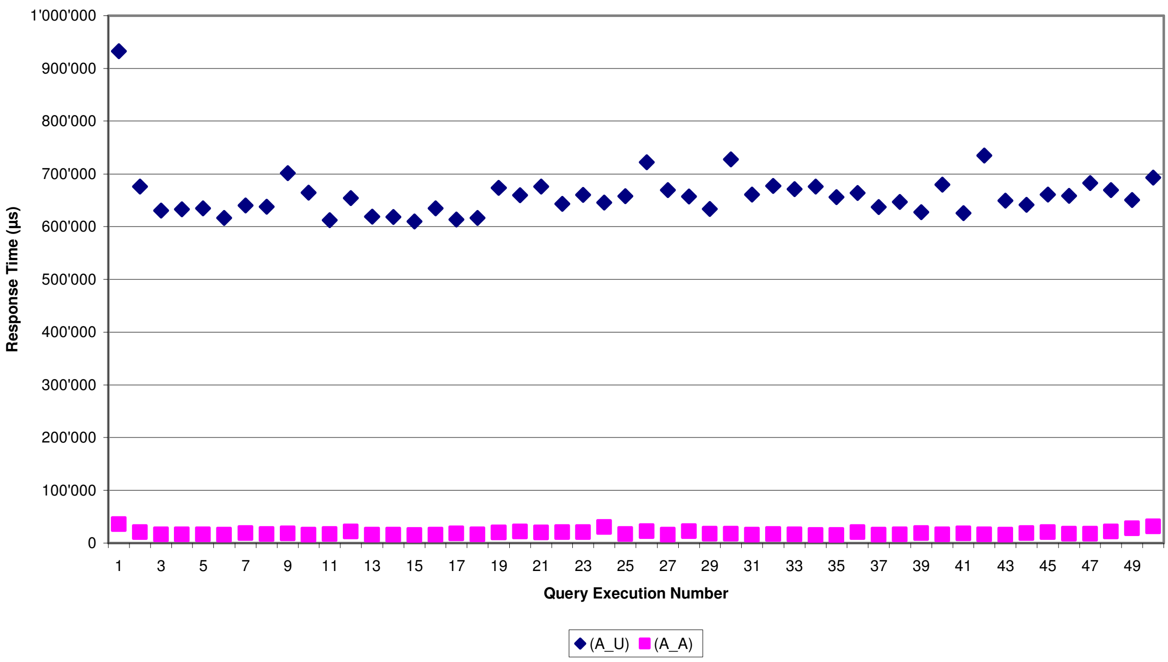
```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND 10000 <= ALL (SELECT DC.QUANTITY
                       FROM TSMT_DETAILED_CHARGING DC
                      WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                        AND DC.SERVICE_ID = CSE.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        4      0.57       0.66          0      62731          0          36
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6      0.57       0.66          0      62731          0          36

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     36  TABLE ACCESS BY INDEX ROWID TSMT_CATALOGUE_SERVICE (cr=62731 pr=0 pw=0
time=295745 us)
     36   INDEX RANGE SCAN PK_T_SMT_CSE (cr=62720 pr=0 pw=0 time=295523 us)(object id
83844)
     59    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=62714 pr=0 pw=0 time=667977
us)
```

Median 17,490 µs (A_A):

```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND NOT EXISTS (SELECT *
                     FROM TSMT_DETAILED_CHARGING DC
                    WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                      AND DC.SERVICE_ID = CSE.SERVICE_ID
                      AND NOT DC.QUANTITY >= 10000)
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        4      0.00       0.01          0        724          0          36
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6      0.00       0.01          0        724          0          36

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     36  HASH JOIN ANTI (cr=724 pr=0 pw=0 time=10939 us)
     95   TABLE ACCESS FULL TSMT_CATALOGUE_SERVICE (cr=31 pr=0 pw=0 time=252 us)
   1572   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9941 us)
```





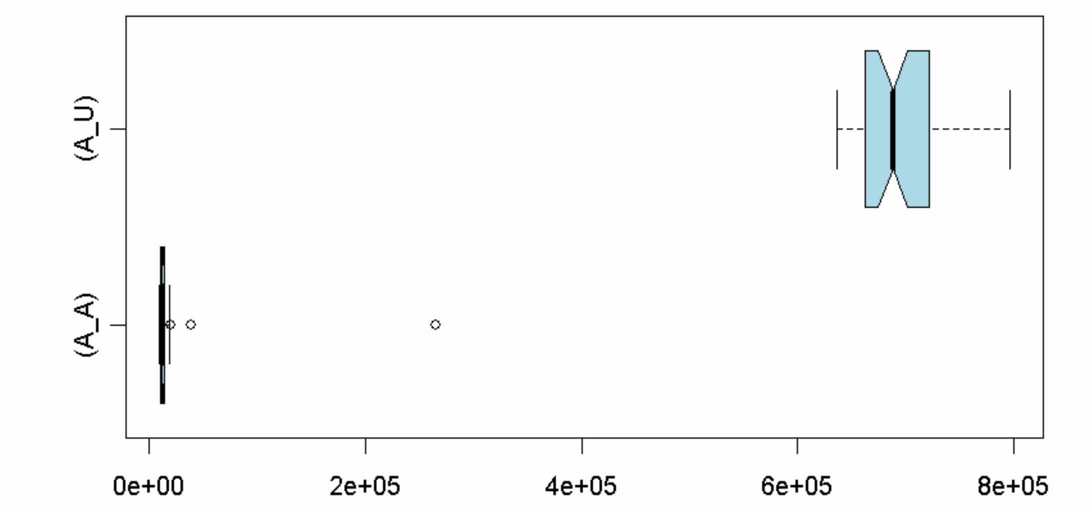
TQP_86211: A selection of the form col1 comp `ANY (SELECT col2 FROM … WHERE cond)` can be rewritten as `NOT EXISTS (SELECT … FROM … WHERE cond AND NOT col1 negatived_comp col2)`

Median 692,200 µs (A_U):

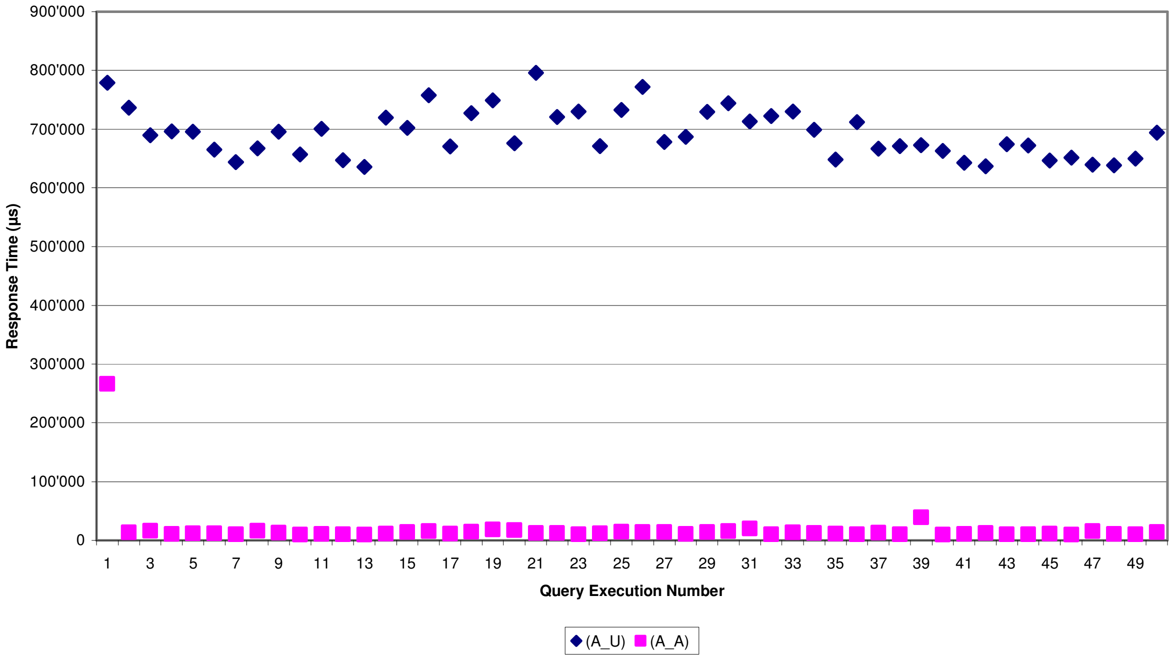
```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND 10000 <= ANY (SELECT DC.QUANTITY
                       FROM TSMT_DETAILED_CHARGING DC
                      WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                        AND DC.SERVICE_ID = CSE.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.67       0.68          0      65257          0          16
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.67       0.68          0      65257          0          16

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     16  TABLE ACCESS BY INDEX ROWID TSMT_CATALOGUE_SERVICE (cr=65257 pr=0 pw=0
time=834846 us)
     16   INDEX RANGE SCAN PK_T_SMT_CSE (cr=65248 pr=0 pw=0 time=834712 us)(object id
83844)
     16    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=65244 pr=0 pw=0 time=680763
us)
```

Median 18,350 µs (A_A):

```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND EXISTS (SELECT *
                 FROM TSMT_DETAILED_CHARGING DC
                WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                  AND DC.SERVICE_ID = CSE.SERVICE_ID
                  AND DC.QUANTITY >= 10000)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0        725          0          16
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.00       0.00          0        725          0          16

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     16  HASH JOIN SEMI (cr=725 pr=0 pw=0 time=5790 us)
     95   TABLE ACCESS FULL TSMT_CATALOGUE_SERVICE (cr=31 pr=0 pw=0 time=241 us)
     39   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=694 pr=0 pw=0 time=4466 us)
```



(A_U)
(A_A)
0e+00
2e+05
4e+05
6e+05
8e+05



Response Time (µs)
900'000
800'000
700'000
600'000
500'000
400'000
300'000
200'000
100'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

ALL / ANY or MAX / MIN

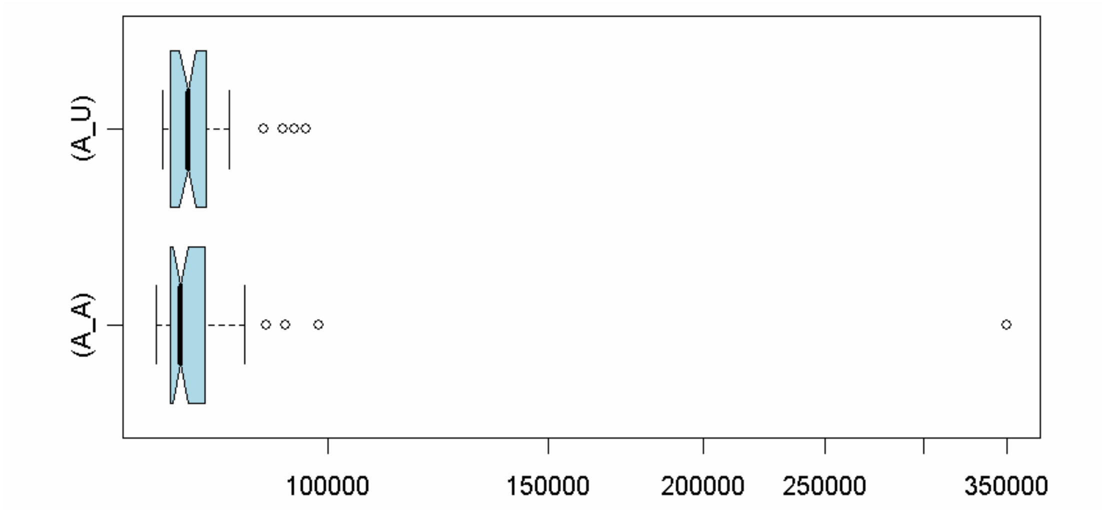
TQP_86101: The operators < ANY or <= ANY may be replaced by < or <= along with a MAX function

Median 77,150 µs (A_U):

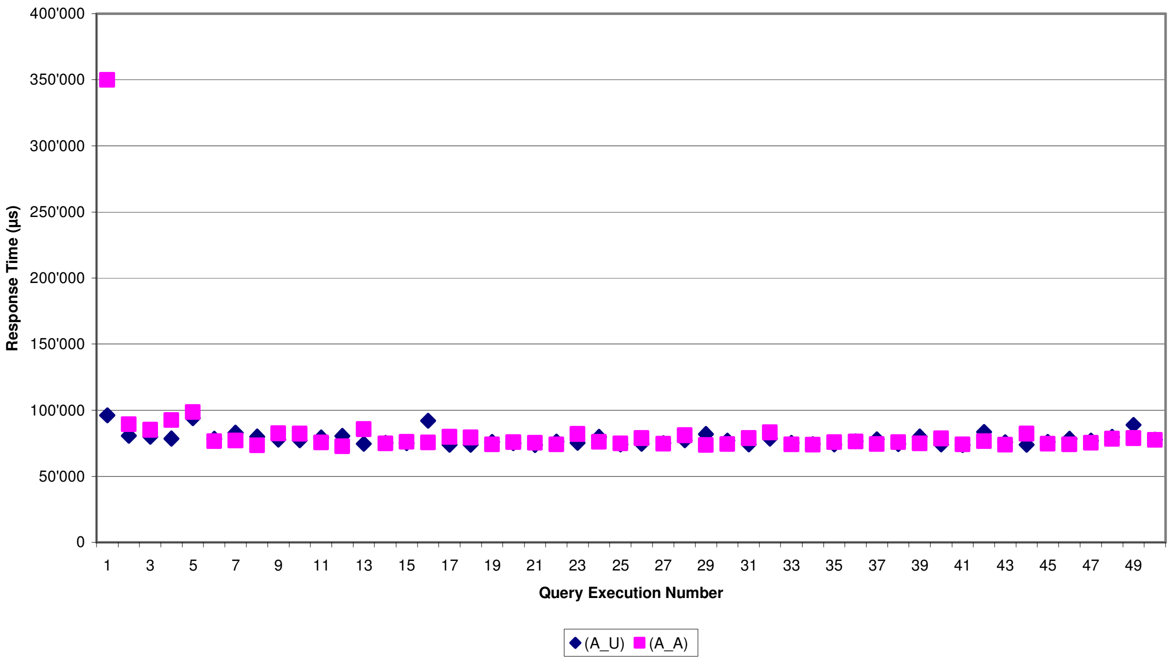
```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID <=
                      ANY (SELECT DC2.SERVICE_RECEIVER_ID
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.02          0       1545          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.02          0       1545          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH JOIN RIGHT SEMI (cr=1545 pr=0 pw=0 time=26726 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=10408 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=9306 us)
```

Median 76,100 µs (A_A):

```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID <=
                          (SELECT MAX (DC2.SERVICE_RECEIVER_ID)
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.01       0.00          0          0          0           0
Fetch      162      0.03       0.02          0       1545          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.04       0.02          0       1545          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH JOIN  (cr=1545 pr=0 pw=0 time=27463 us)
     59   VIEW  VW_SQ_1 (cr=693 pr=0 pw=0 time=10897 us)
     59    HASH GROUP BY (cr=693 pr=0 pw=0 time=10834 us)
   1611     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9490 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=11099 us)
```



(Logarithmic scale)



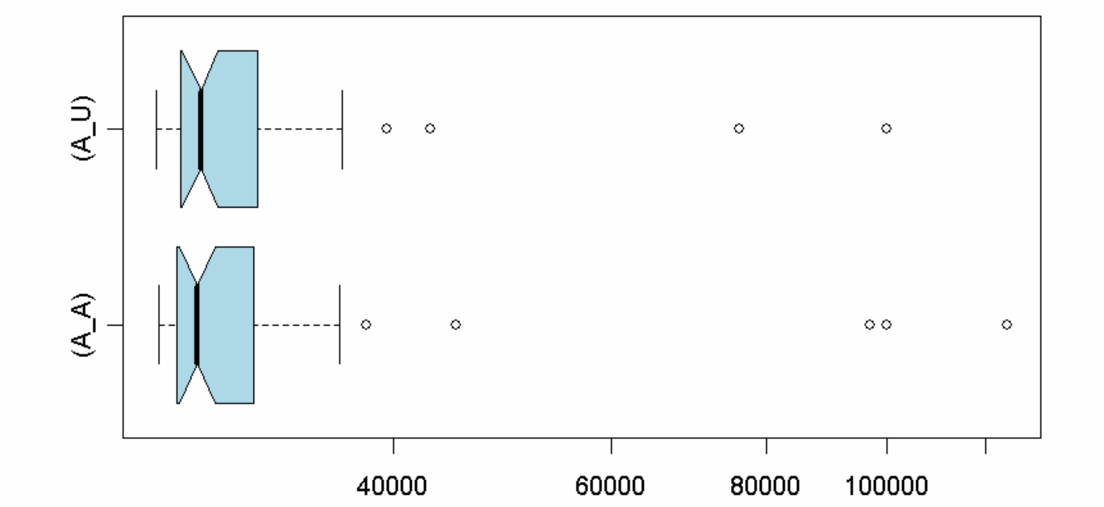
TQP_86111: The operators < `ALL` or <= `ALL` may be replaced by < or <= along with a `MIN` function

Median 28,000 µs (A_U):

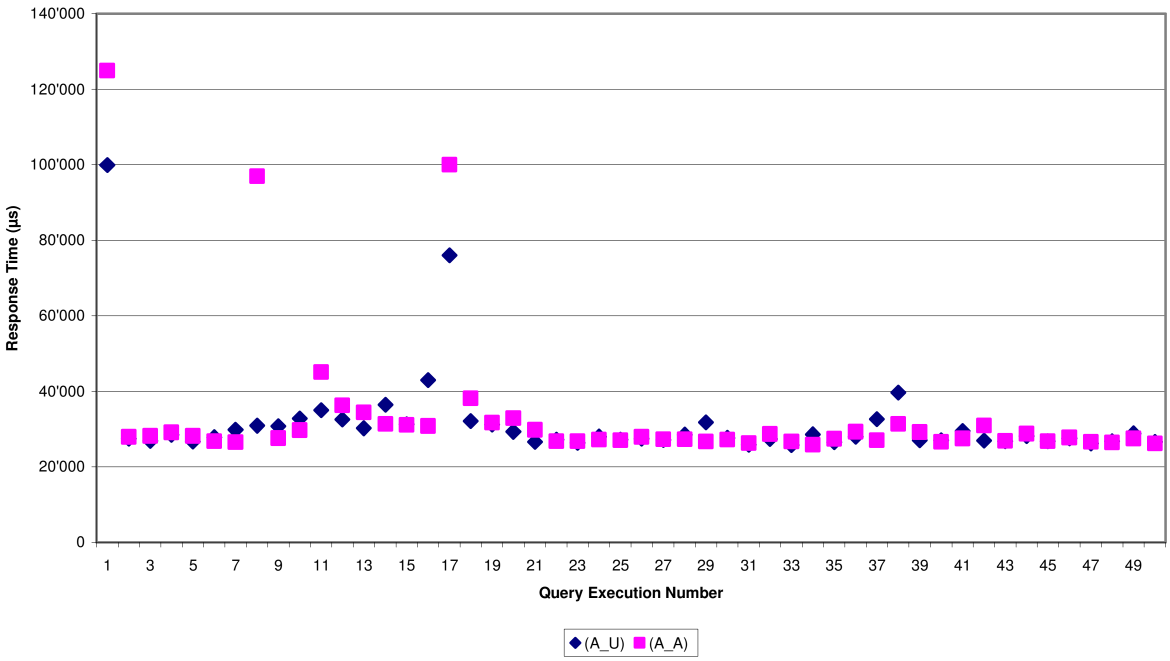
```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID <=
                      ALL (SELECT DC2.SERVICE_RECEIVER_ID
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        7      0.03       0.02          0       1391          0          61
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        9      0.03       0.02          0       1391          0          61

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     61  HASH JOIN RIGHT ANTI (cr=1391 pr=0 pw=0 time=18429 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9541 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=698 pr=0 pw=0 time=9365 us)
```

Median 27,830 µs (A_A):

```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID <=
                           (SELECT MIN (DC2.SERVICE_RECEIVER_ID)
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        7      0.01       0.02          0       1391          0          61
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        9      0.01       0.02          0       1391          0          61

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     61  HASH JOIN  (cr=1391 pr=0 pw=0 time=20092 us)
     59   VIEW  VW_SQ_1 (cr=693 pr=0 pw=0 time=10836 us)
     59    HASH GROUP BY (cr=693 pr=0 pw=0 time=10773 us)
   1611     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9712 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=698 pr=0 pw=0 time=9817 us)
```



(Logarithmic scale)



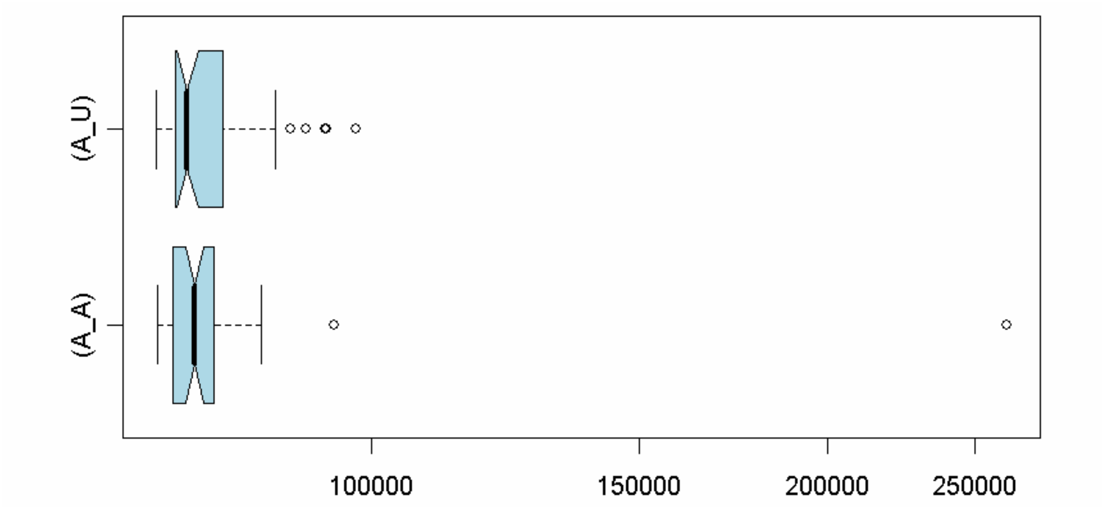
TQP_86121: The operators > `ANY` or >= `ANY` may be replaced by > or >= along with a `MIN` function

Median 75,540 µs (A_U):

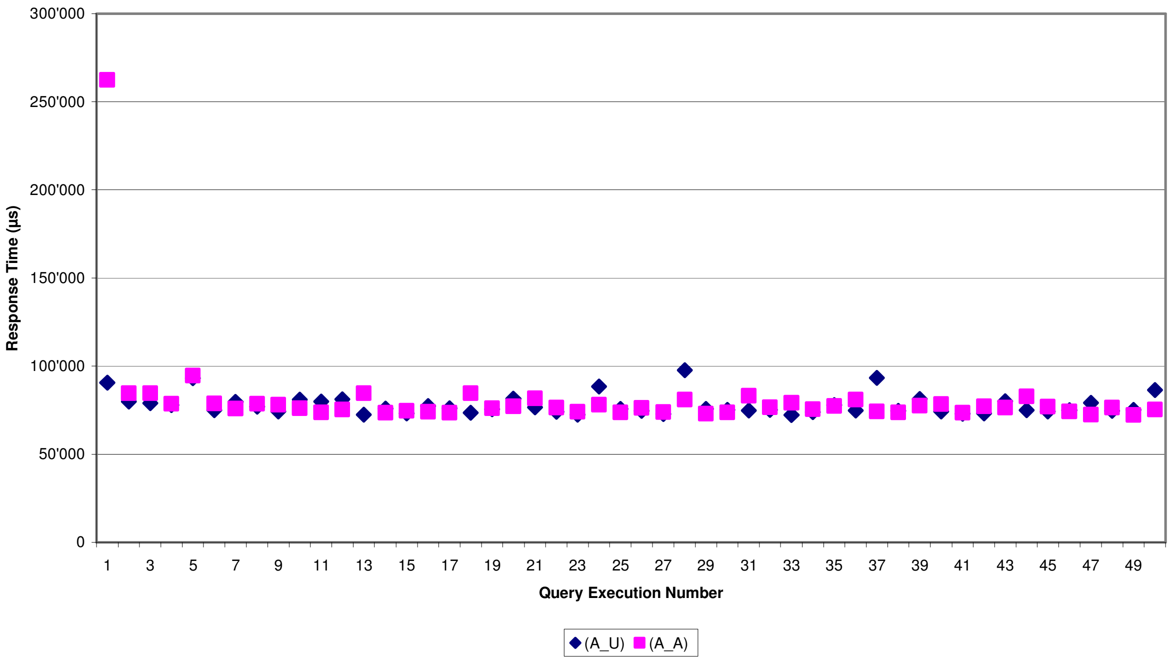
```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID >=
                      ANY (SELECT DC2.SERVICE_RECEIVER_ID
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.02          0       1545          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.02          0       1545          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH JOIN RIGHT SEMI (cr=1545 pr=0 pw=0 time=27854 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9923 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=10824 us)
```

Median 76,360 µs (A_A):

```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID >=
                          (SELECT MIN (DC2.SERVICE_RECEIVER_ID)
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.02          0       1545          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.02          0       1545          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH JOIN  (cr=1545 pr=0 pw=0 time=27362 us)
     59   VIEW  VW_SQ_1 (cr=693 pr=0 pw=0 time=10686 us)
     59    HASH GROUP BY (cr=693 pr=0 pw=0 time=10624 us)
   1611     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9551 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=9517 us)
```



(Logarithmic scale)



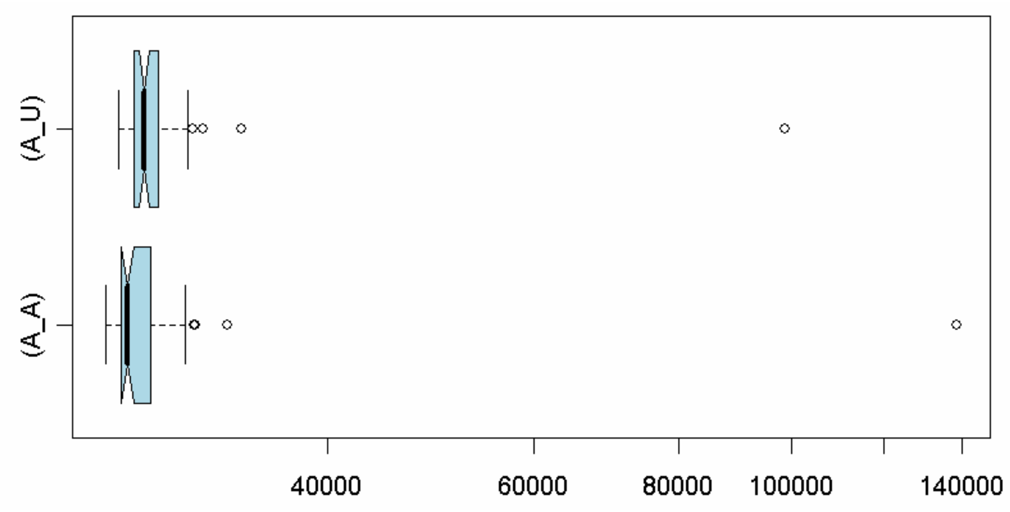
TQP_86131: The operators > `ALL` or >= `ALL` may be replaced by > or >= along with a `MAX` function

Median 27,800 µs (A_U):

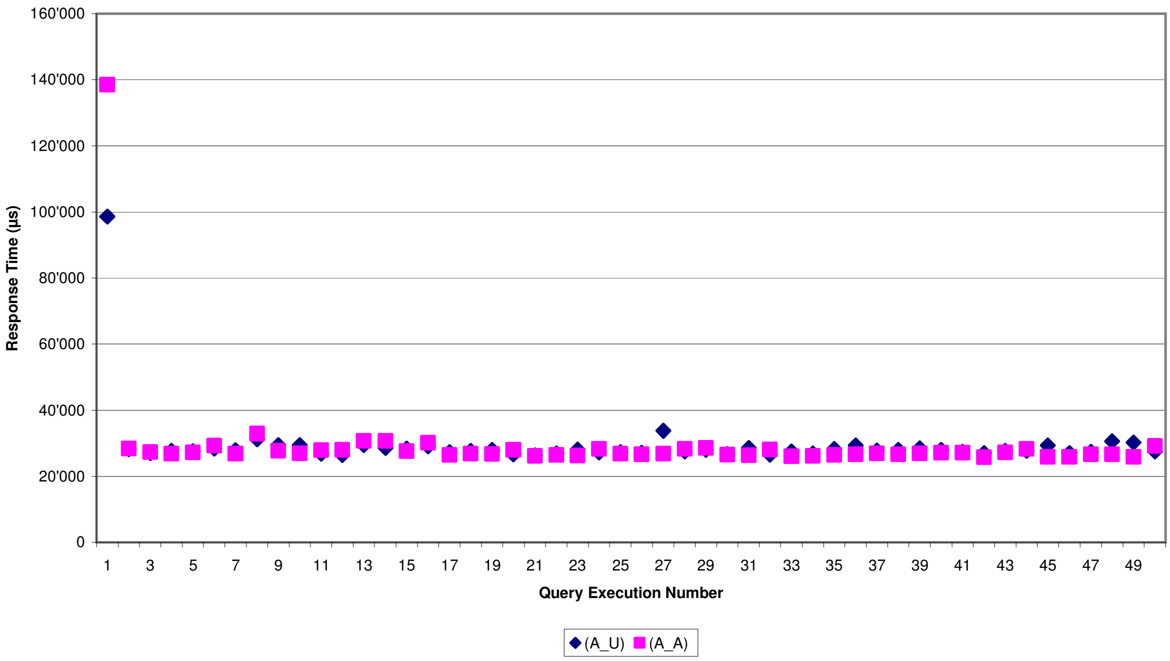
```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID >=
                      ALL (SELECT DC2.SERVICE_RECEIVER_ID
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        7      0.01       0.02          0       1392          0          61
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        9      0.01       0.02          0       1392          0          61

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     61  HASH JOIN RIGHT ANTI (cr=1392 pr=0 pw=0 time=18011 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9411 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=699 pr=0 pw=0 time=9342 us)
```

Median 26,920 µs (A_A):

```
SELECT DC1.IDENT
  FROM TSMT_DETAILED_CHARGING DC1
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC1.SERVICE_RECEIVER_ID >=
                           (SELECT MAX (DC2.SERVICE_RECEIVER_ID)
                             FROM TSMT_DETAILED_CHARGING DC2
                            WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                              AND DC2.SERVICE_ID = DC1.SERVICE_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        7      0.01       0.02          0       1392          0          61
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        9      0.01       0.02          0       1392          0          61

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     61  HASH JOIN  (cr=1392 pr=0 pw=0 time=20899 us)
     59   VIEW  VW_SQ_1 (cr=693 pr=0 pw=0 time=10989 us)
     59    HASH GROUP BY (cr=693 pr=0 pw=0 time=10871 us)
   1611     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9779 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=699 pr=0 pw=0 time=10579 us)
```



(Logarithmic scale)



All Rows / Distinct Rows

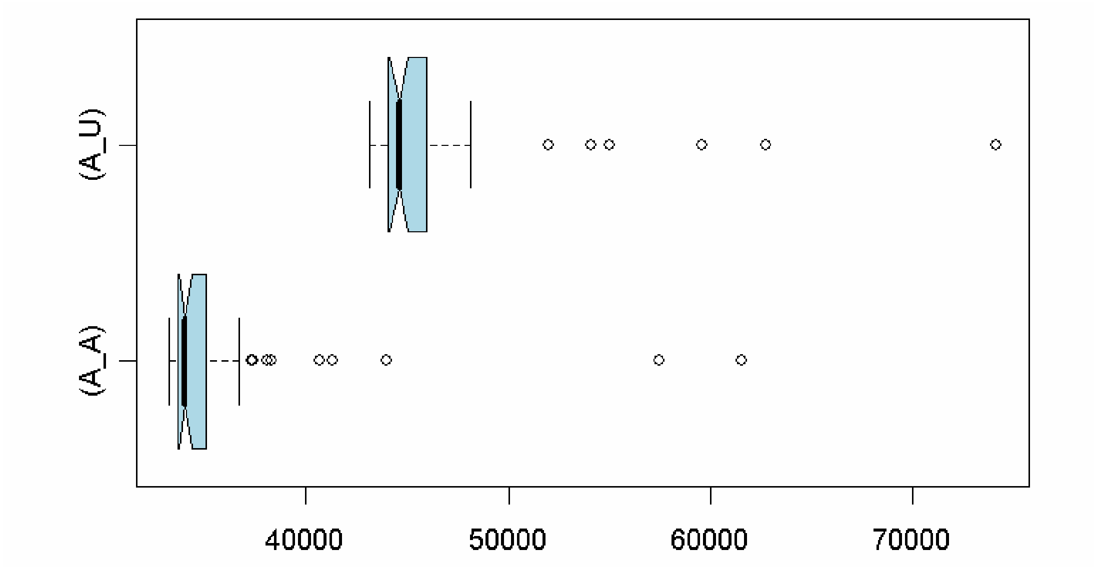
TQP_80101: Within a query use `SELECT [ ALL | DISTINCT ]`: Projection of a non-indexed column

Median 44,590 µs (A_U, per 1,000 resulting rows):

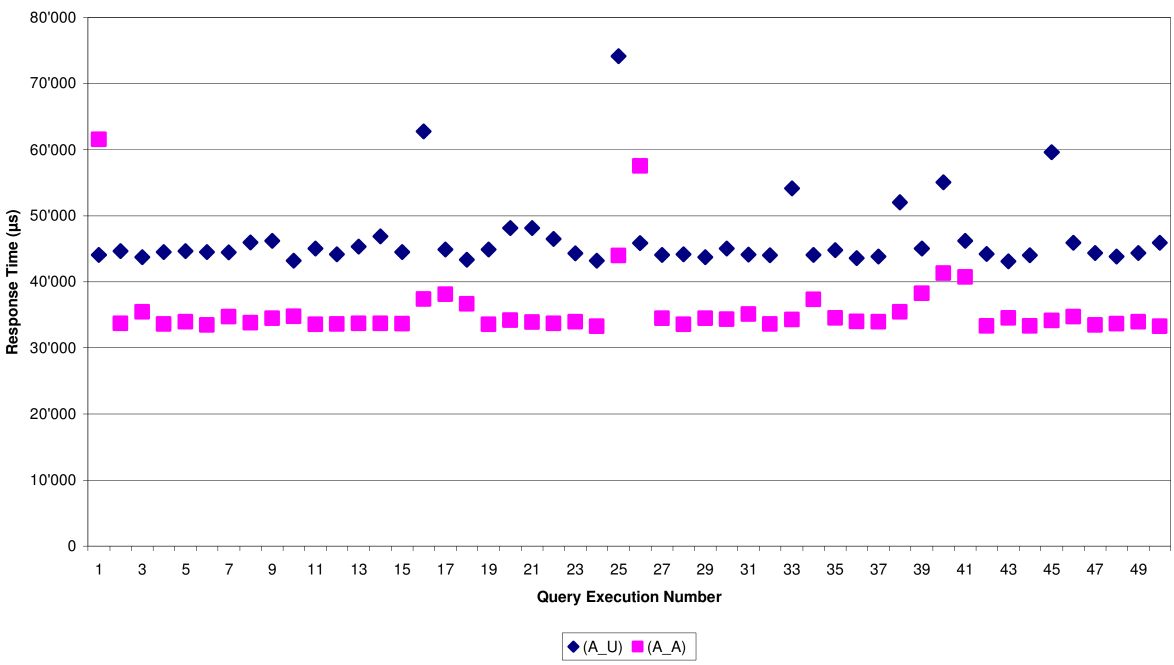
```
SELECT DISTINCT AMOUNT_SERVICE
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      178      0.01       0.02          0        693          0        1774
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      180      0.01       0.02          0        693          0        1774

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1774  HASH UNIQUE (cr=693 pr=0 pw=0 time=20573 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33244 us)
```

Median 33,640 µs (A_A, per 1,000 resulting rows):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.04       0.05          0       3952          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.04       0.05          0       3952          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33279 us)
```



Response times per 1,000 resulting rows



Response times per 1,000 resulting rows

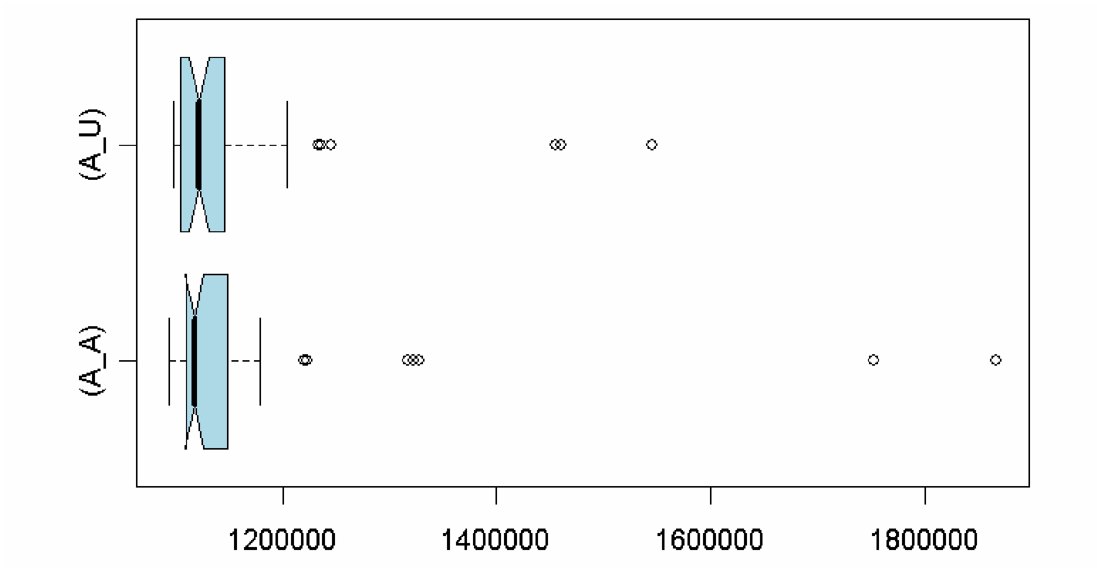
TQP_80102: Within a query use `SELECT [ ALL | DISTINCT ]`: Projection of an indexed column

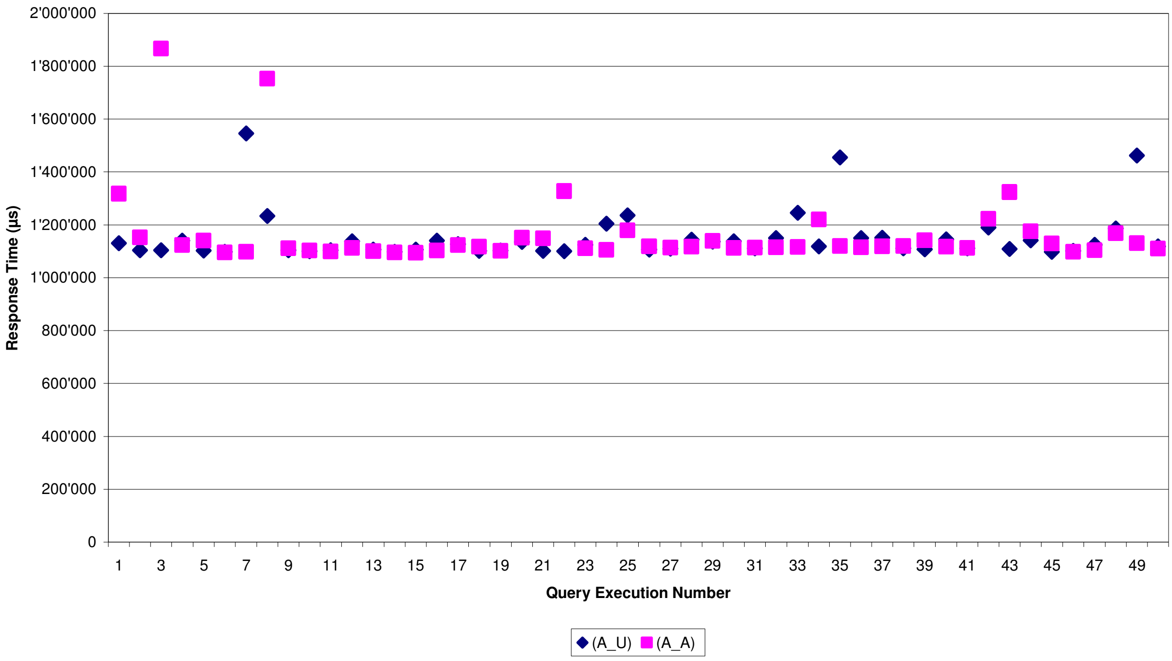
Median 1,122,000 µs (A_U):

```
SELECT DISTINCT IDENT
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.06       0.05          0         75          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.06       0.05          0         75          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH UNIQUE (cr=75 pr=0 pw=0 time=48936 us)
  33183   INDEX FAST FULL SCAN PK_T_SMT_DC (cr=75 pr=0 pw=0 time=32 us)(object id
83890)
```

Median 1,118,000 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.01       0.04          0       3387          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.01       0.04          0       3387          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  INDEX FAST FULL SCAN PK_T_SMT_DC (cr=3387 pr=0 pw=0 time=33275 us)(object id
83890)
```


(A_U)
(A_A)
1200000
1400000
1600000
1800000


2'000'000
1'800'000
1'600'000
1'400'000
1'200'000
1'000'000
800'000
600'000
400'000
200'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

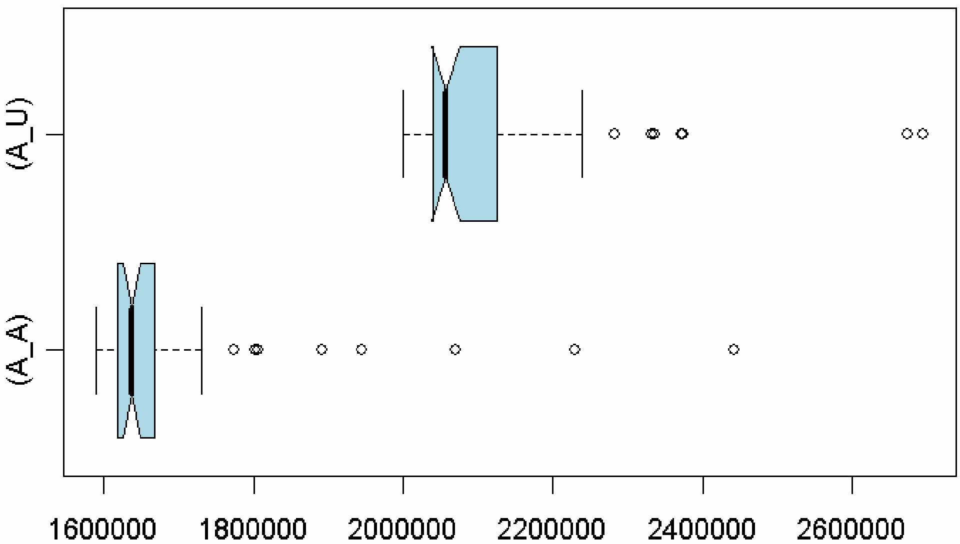
TQP_80103: Within a query use `SELECT [ ALL | DISTINCT ]`: Projection of all columns

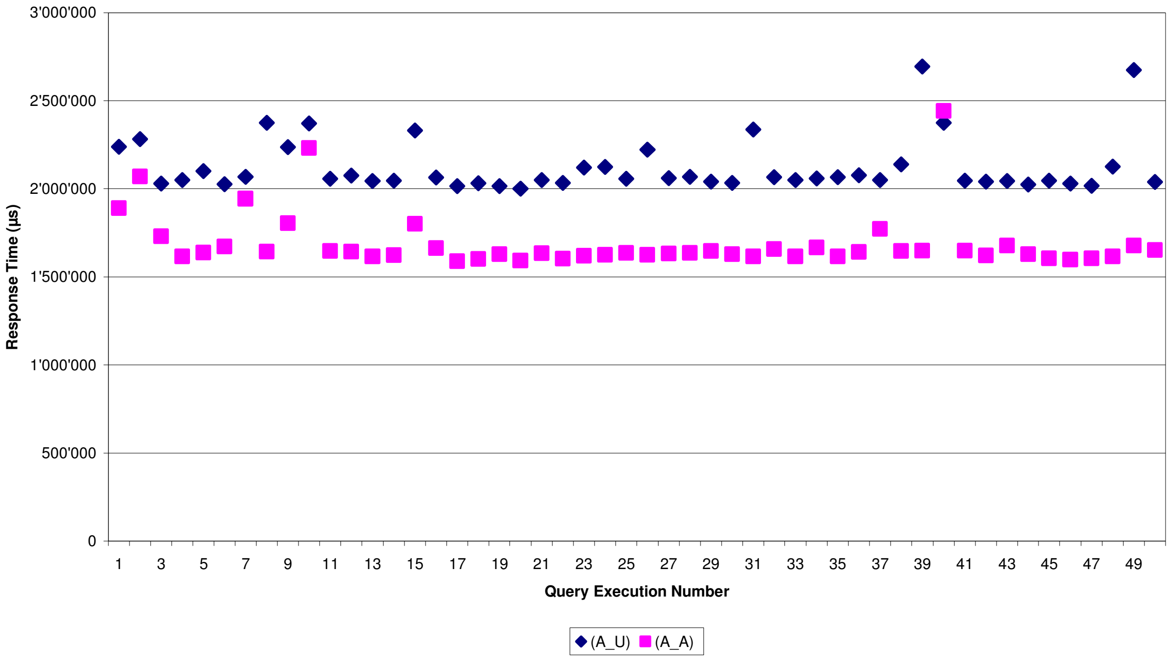
Median 2,057,000 µs (A_U):

```
SELECT DISTINCT *
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.45       0.61        735        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.45       0.61        735        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH UNIQUE (cr=693 pr=735 pw=735 time=489516 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33229 us)
```

Median 1,638,000 µs (A_A):

```
SELECT *
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.20       0.17          0       3952          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.20       0.17          0       3952          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=99656 us)
```




3'000'000
2'500'000
2'000'000
1'500'000
1'000'000
500'000
0
Response Time (µs)
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
Query Execution Number
(A_U)
(A_A)

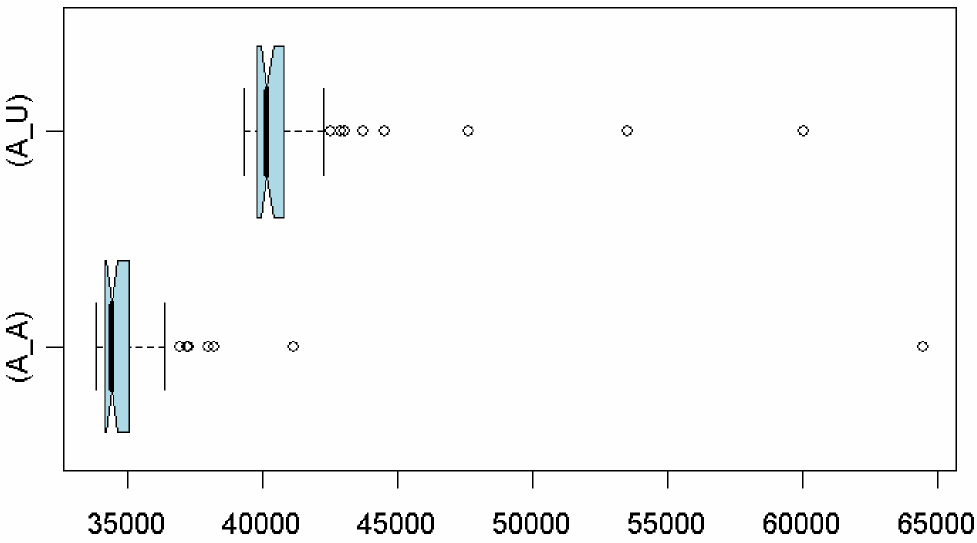
TQP_80104: Within a query use `SELECT [ ALL | DISTINCT ]`: Projection of a non-indexed column – bigger table

Median 40,170 µs (A_U, per 1,000 resulting rows):

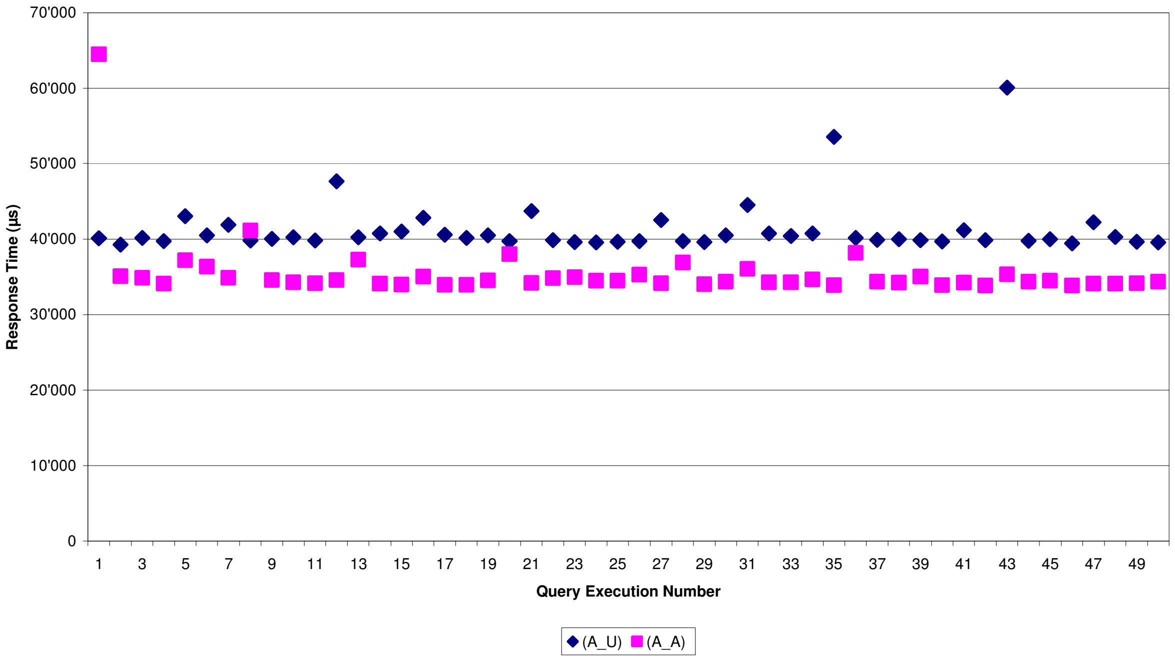
```
SELECT DISTINCT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1161      0.10       0.09       1742       1954          0       11605
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1163      0.10       0.09       1742       1954          0       11605

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  11605  HASH UNIQUE (cr=1954 pr=1742 pw=0 time=89901 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1742 pw=0 time=205414
us)
```

Median 34,420 µs (A_A, per 1,000 resulting rows):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.17       0.18       1778      12031          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.17       0.18       1778      12031          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=12031 pr=1778 pw=0 time=205620
us)
```



Response times per 1,000 resulting rows



Response times per 1,000 resulting rows

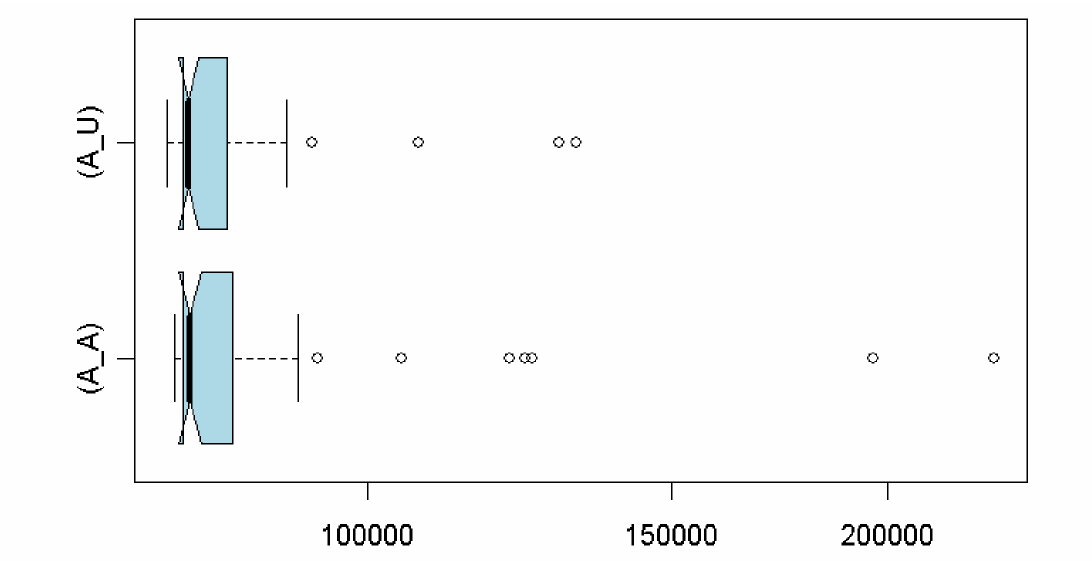
TQP_80105: Within a query use `SELECT [ ALL | DISTINCT ]`: Projection of a non-indexed column – `GROUP BY` instead of `DISTINCT`

Median 78,870 µs (A_U):

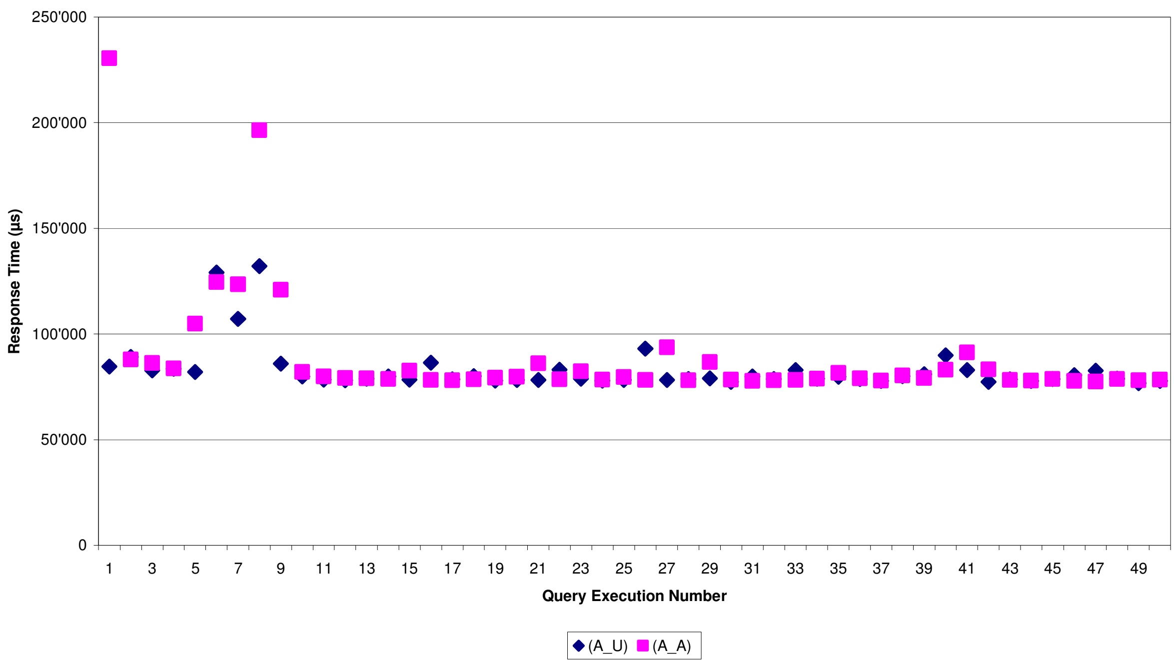
```
SELECT DISTINCT AMOUNT_SERVICE
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      178      0.01       0.02          0        693          0        1774
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      180      0.01       0.02          0        693          0        1774

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1774  HASH UNIQUE (cr=693 pr=0 pw=0 time=22060 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33242 us)
```

Median 79,070 µs (A_A):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_CHARGING
 GROUP BY AMOUNT_SERVICE

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      178      0.01       0.02          0        693          0        1774
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      180      0.01       0.02          0        693          0        1774

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1774  HASH GROUP BY (cr=693 pr=0 pw=0 time=19920 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33259 us)
```



(Logarithmic scale)



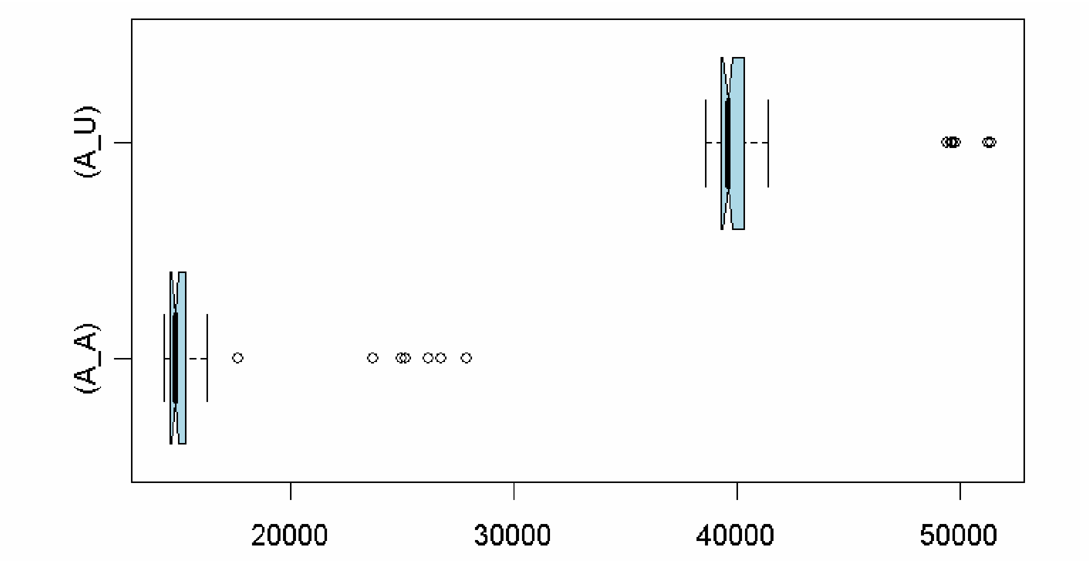
TQP_80111: Within certain aggregate functions use `<aggregate function> ( [ ALL | DISTINCT ] <value expression> )`: Function `AVG`

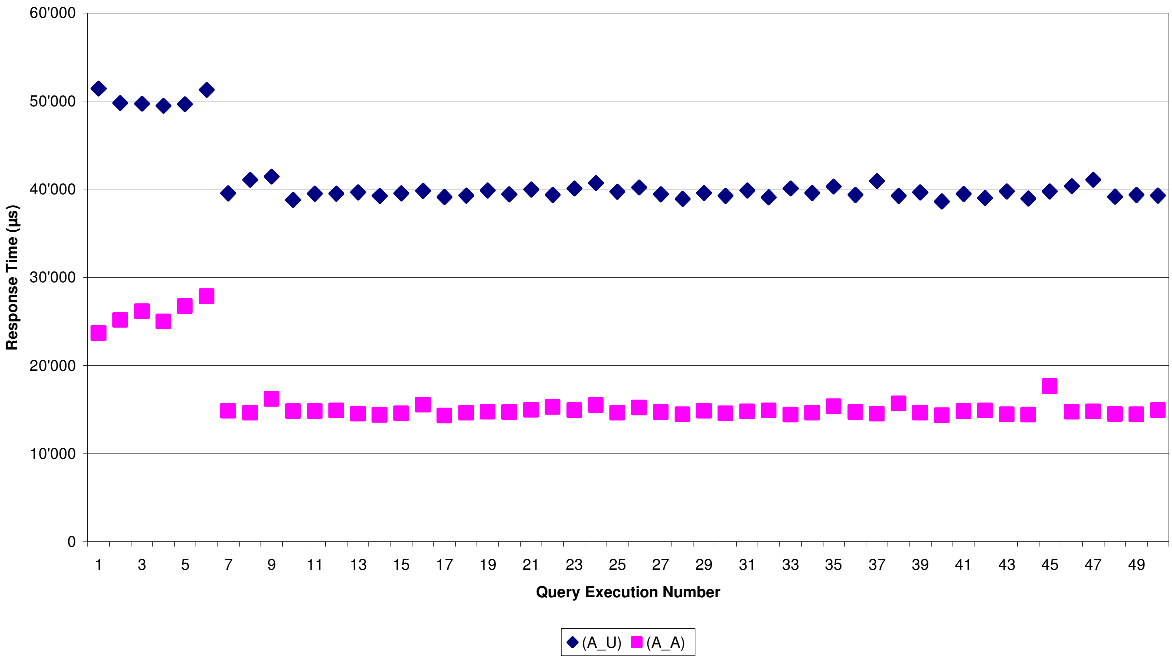
Median 39,590 µs (A_U):

```
SELECT AVG (DISTINCT AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.03       0.03          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.03       0.03          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT GROUP BY (cr=693 pr=0 pw=0 time=36713 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33240 us)
```

Median 14,790 µs (A_A):

```
SELECT AVG (AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=693 pr=0 pw=0 time=12289 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33223 us)
```


(A_U)
(A_A)
20000
30000
40000
50000


60'000
50'000
40'000
30'000
20'000
10'000
0
Response Time (µs)
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
Query Execution Number
(A_U)
(A_A)

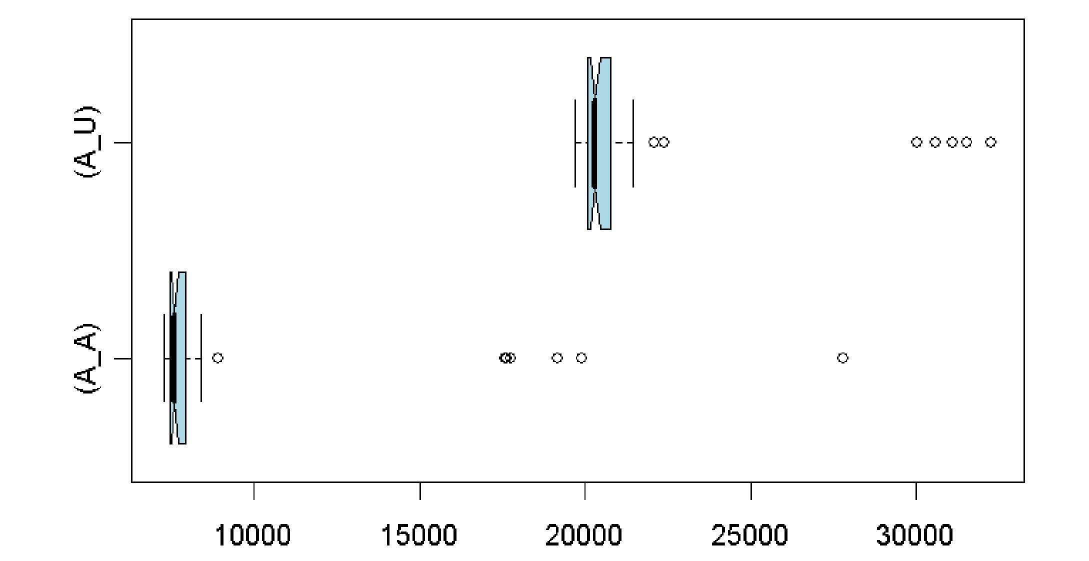
TQP_80112: Within certain aggregate functions use <aggregate function> ([ALL | DISTINCT] <value expression>): Function COUNT

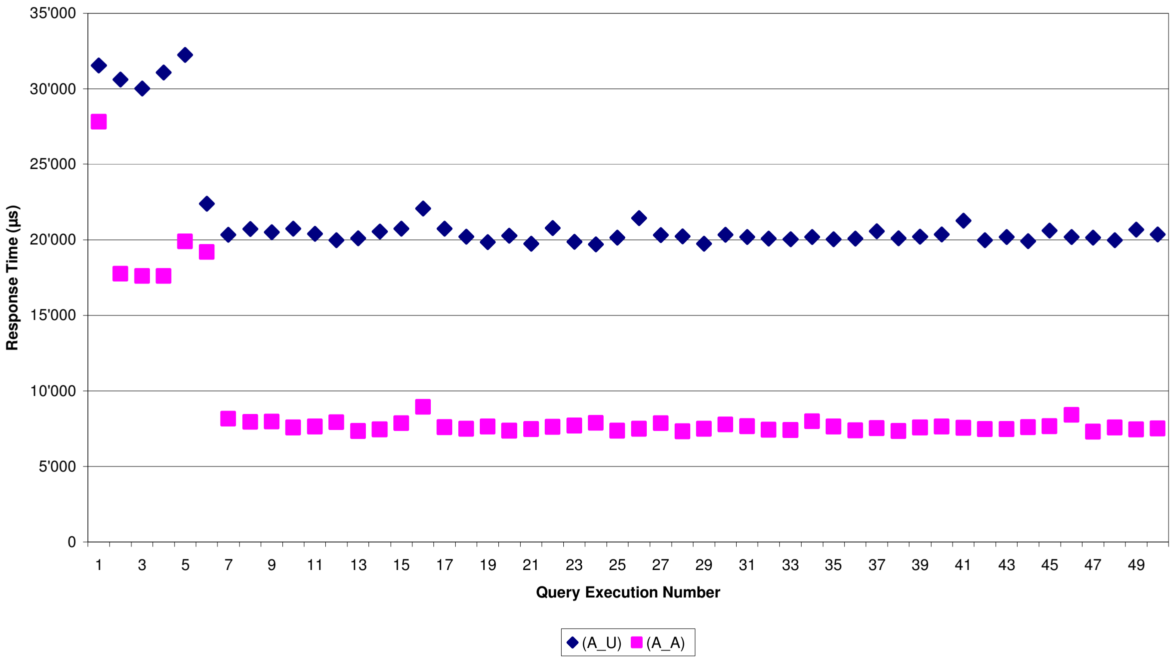
Median 20,300 µs (A_U):

```
SELECT COUNT (DISTINCT PERIOD)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT GROUP BY (cr=693 pr=0 pw=0 time=17540 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33224 us)
```

Median 7,610 µs (A_A):

```
SELECT COUNT (PERIOD)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0         75          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0         75          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=75 pr=0 pw=0 time=5280 us)
  33183   INDEX FAST FULL SCAN PK_T_SMT_DC (cr=75 pr=0 pw=0 time=38 us)(object id 
83890)
```


(A_U)
(A_A)
10000
15000
20000
25000
30000


35'000
30'000
25'000
20'000
15'000
10'000
5'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U)
(A_A)

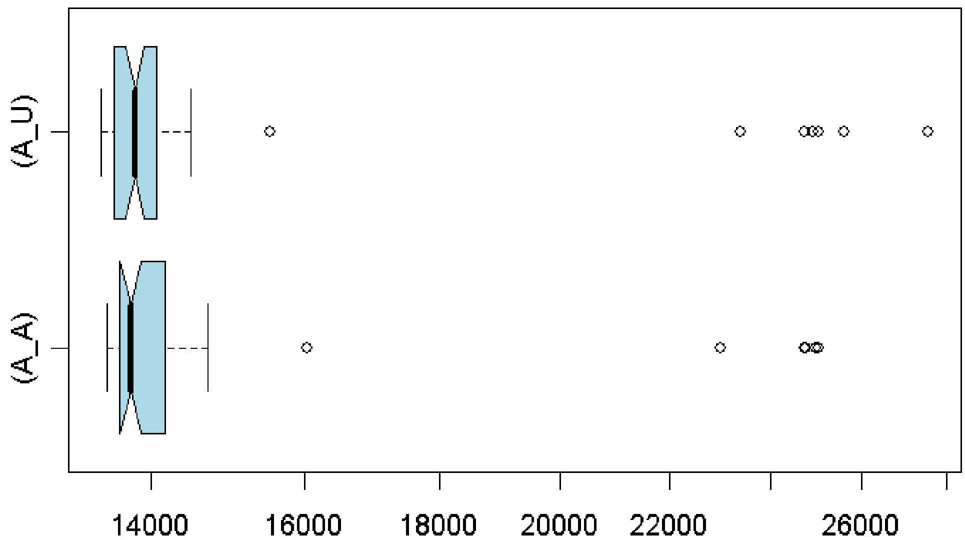
TQP_80113: Within certain aggregate functions use `<aggregate function> ( [ ALL | DISTINCT ] <value expression> )`: Function `MAX`

Median 13,800 µs (A_U):

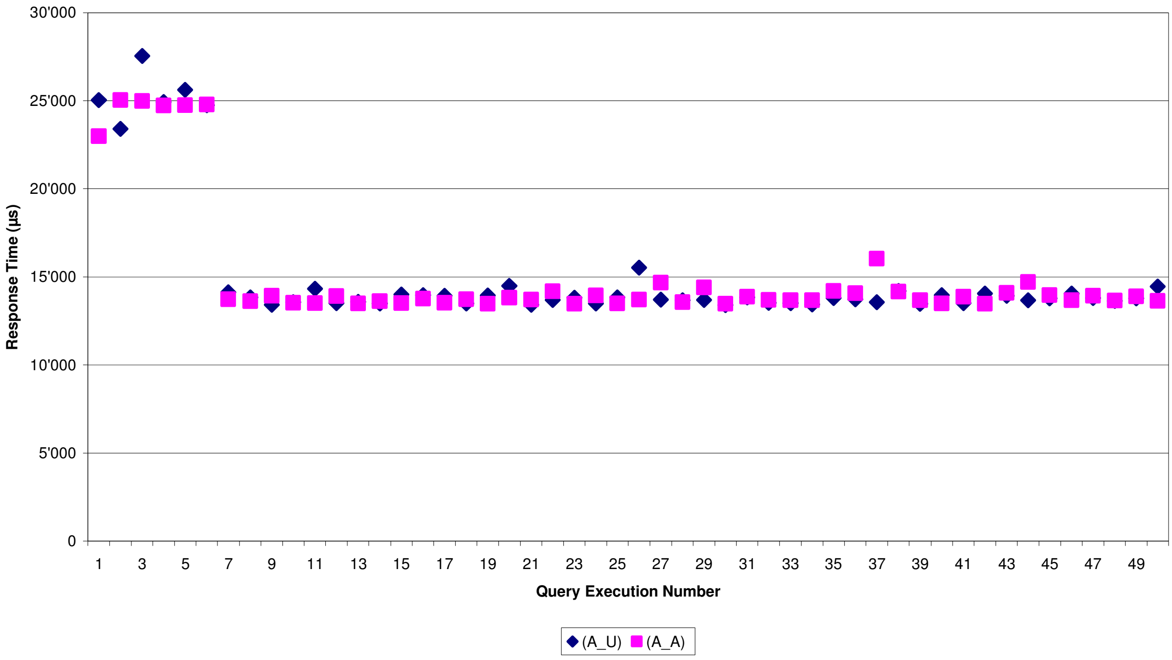
```
SELECT MAX (DISTINCT AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=693 pr=0 pw=0 time=11251 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33221 us)
```

Median 13,740 µs (A_A):

```
SELECT MAX (AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.03       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=693 pr=0 pw=0 time=11334 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33224 us)
```



(Logarithmic scale)



30'000
25'000
20'000
15'000
10'000
5'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

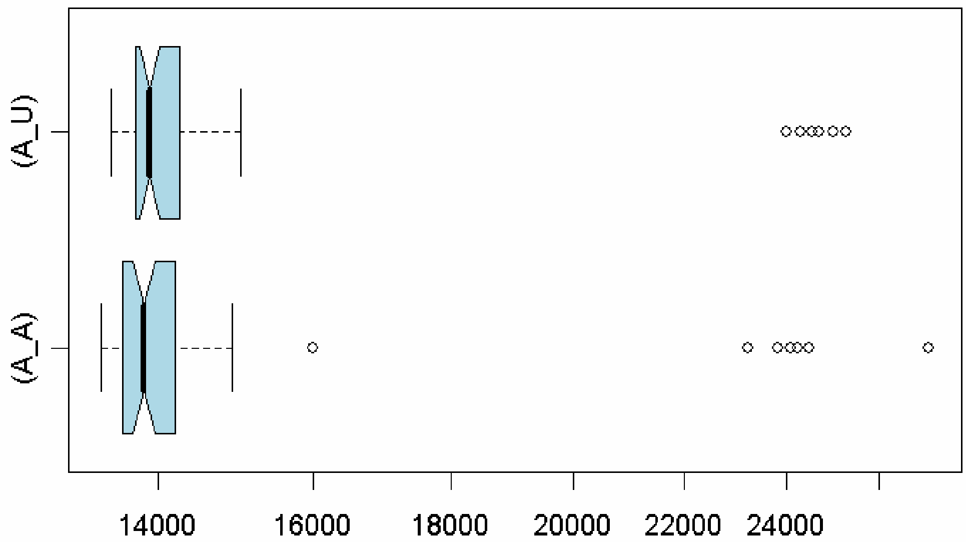
TQP_80114: Within certain aggregate functions use `<aggregate function> ( [ ALL | DISTINCT ] <value expression> )`: Function `MIN`

Median 13,900 µs (A_U):

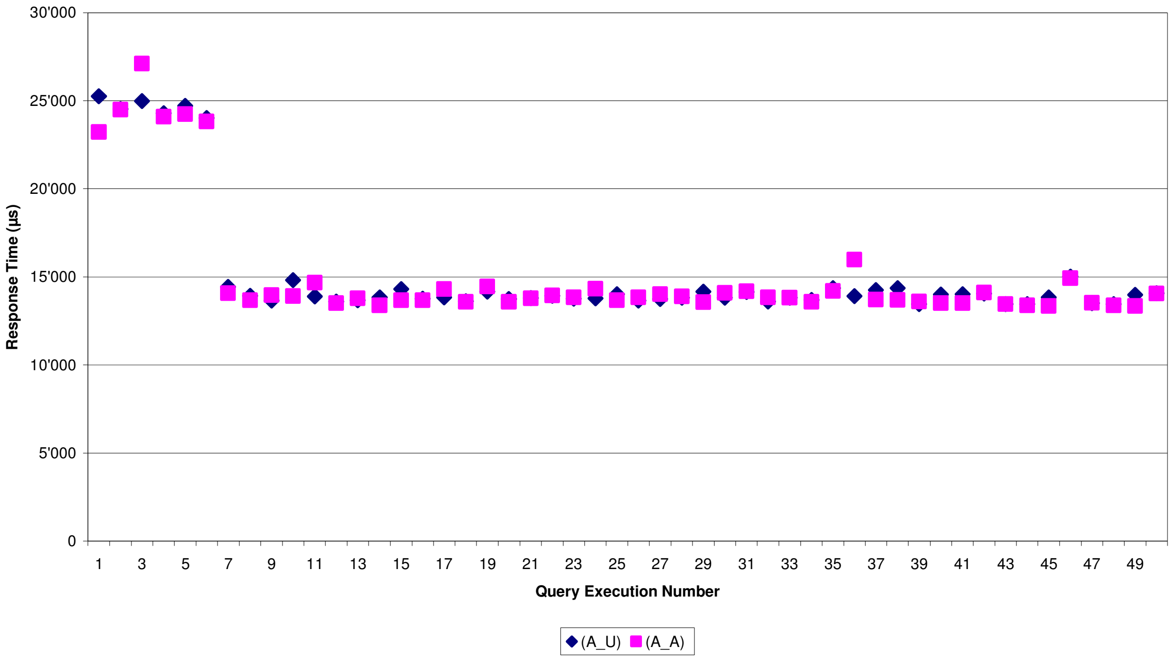
```
SELECT MIN (DISTINCT AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=693 pr=0 pw=0 time=11378 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33221 us)
```

Median 13,840 µs (A_A):

```
SELECT MIN (AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=693 pr=0 pw=0 time=11451 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33221 us)
```



(Logarithmic scale)


30'000
25'000
20'000
15'000
10'000
5'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U)
(A_A)

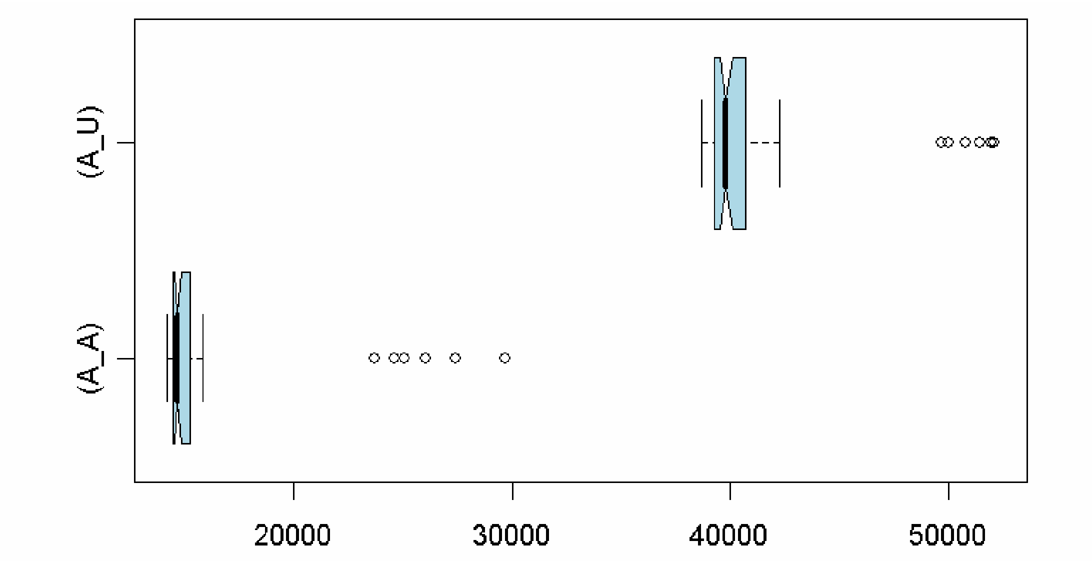
TQP_80115: Within certain aggregate functions use `<aggregate function> ( [ ALL | DISTINCT ] <value expression> )`: Function `SUM`

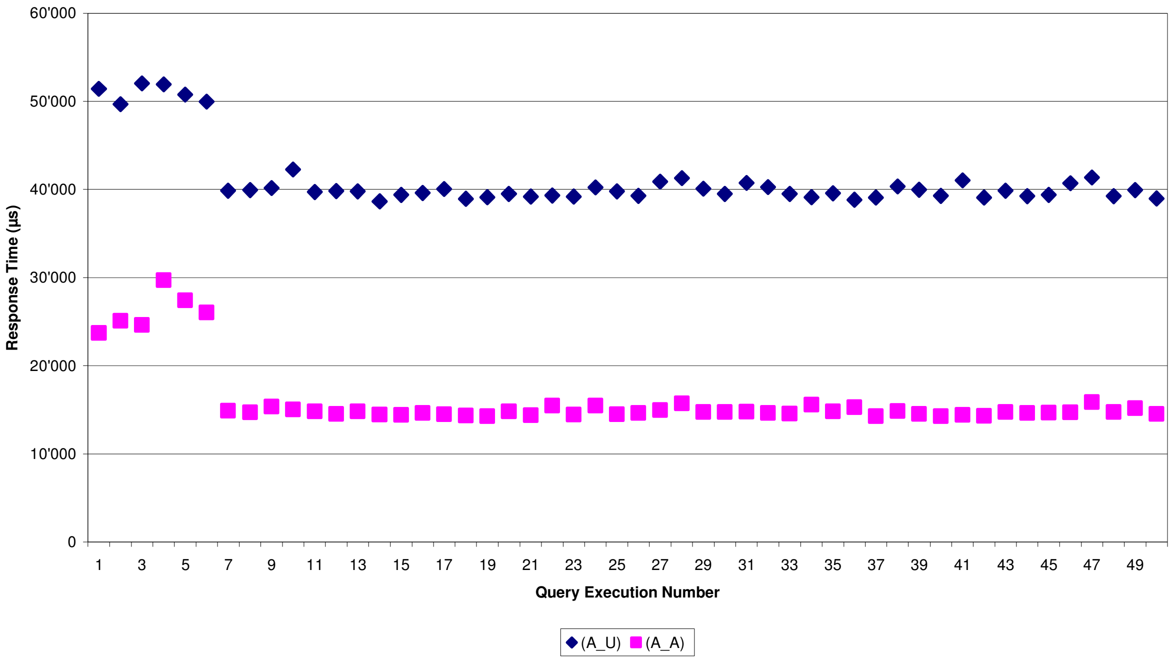
Median 39,810 µs (A_U):

```
SELECT SUM (DISTINCT AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.04       0.03          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.04       0.03          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT GROUP BY (cr=693 pr=0 pw=0 time=37352 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33220 us)
```

Median 14,720 µs (A_A):

```
SELECT SUM (AMOUNT_SERVICE)
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=693 pr=0 pw=0 time=12071 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33224 us)
```


(A_U)
(A_A)
20000
30000
40000
50000


60'000
50'000
40'000
30'000
20'000
10'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

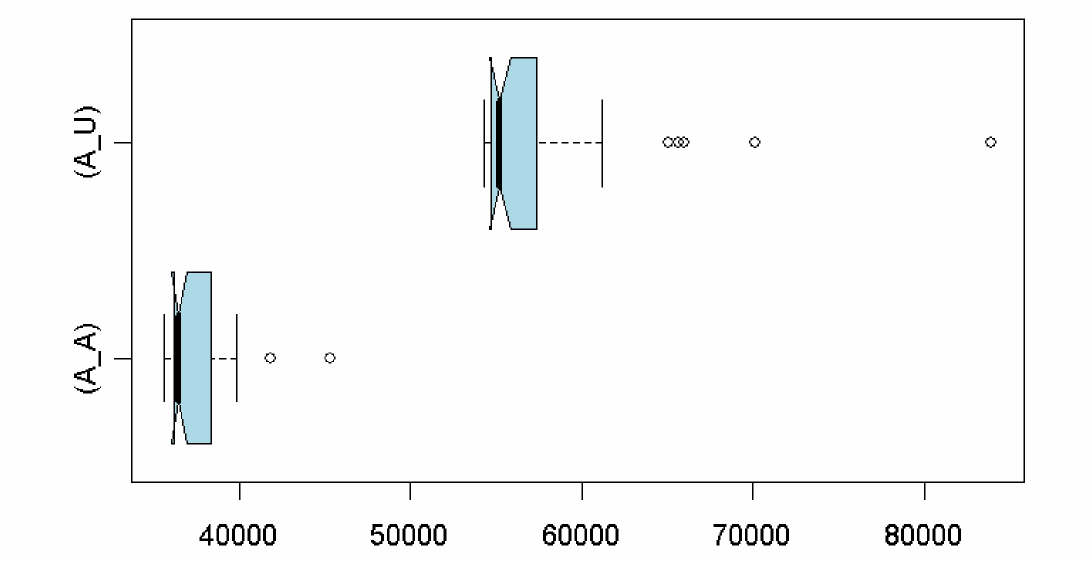
TQP_80121: Use the set operator `UNION ALL` instead of `UNION`: Projection of a non-indexed column

Median 55,230 µs (A_U, per 1,000 resulting rows):

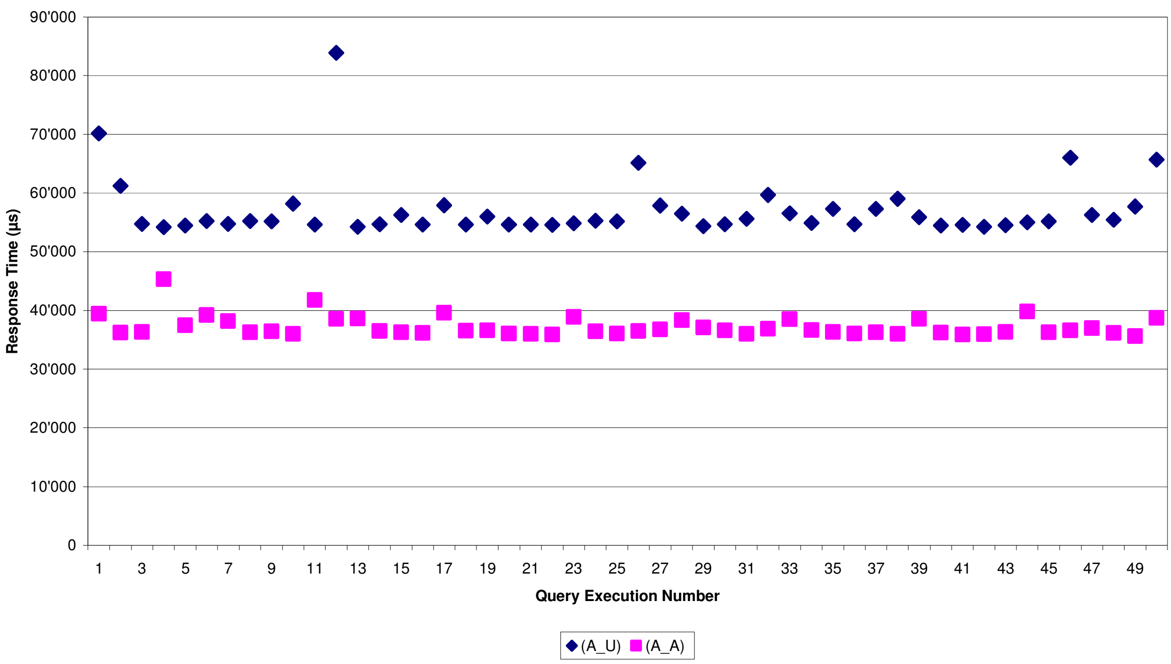
```
SELECT AMOUNT_SERVICE AS CHARGING, NULL AS METERING
  FROM TSMT_DETAILED_CHARGING
UNION
SELECT NULL AS CHARGING, AMOUNT_SERVICE AS METERING
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1338      0.32       0.29       2375       2647          0       13379
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1340      0.32       0.29       2375       2647          0       13379

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  13379  SORT UNIQUE (cr=2647 pr=2375 pw=0 time=283322 us)
 135766   UNION-ALL  (cr=2647 pr=2375 pw=0 time=271656 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=631 pw=0 time=33301 us)
 102583    TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1744 pw=0 time=102804
us)
```

Median 36,470 µs (A_A, per 1,000 resulting rows):

```
SELECT AMOUNT_SERVICE AS CHARGING, NULL AS METERING
  FROM TSMT_DETAILED_CHARGING
UNION ALL
SELECT NULL AS CHARGING, AMOUNT_SERVICE AS METERING
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    13577      0.39       0.36       2494      15959          0      135766
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    13579      0.39       0.36       2494      15959          0      135766

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 135766  UNION-ALL  (cr=15959 pr=2494 pw=0 time=408059 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=677 pw=0 time=66701 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=12007 pr=1817 pw=0 time=102827
us)
```



Response times per 1,000 resulting rows



Response times per 1,000 resulting rows

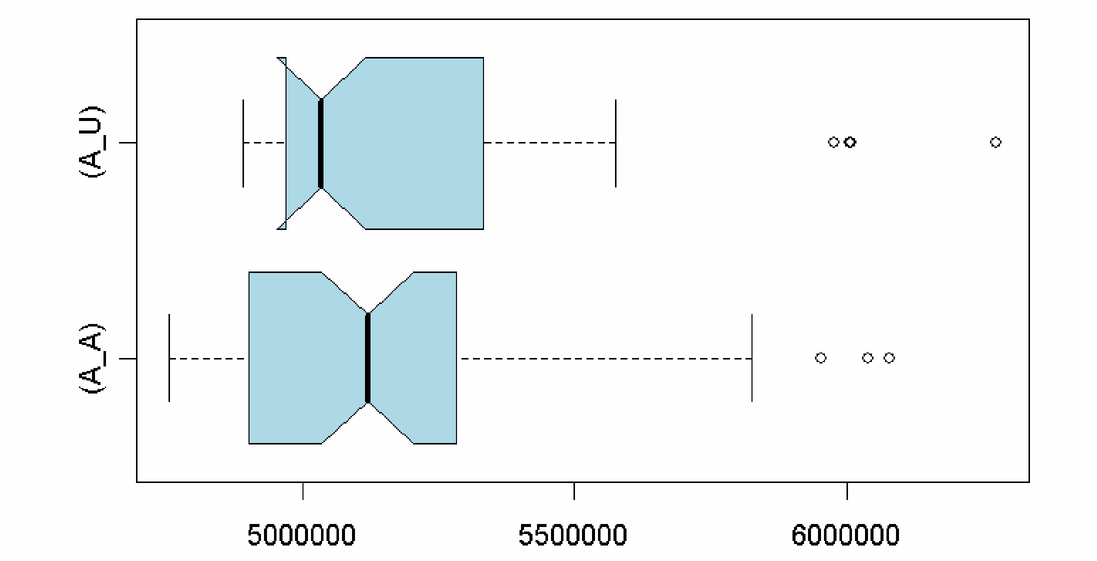
TQP_80122: Use the set operator `UNION ALL` instead of `UNION`: Projection of an indexed column

Median 5,034,000 µs (A_U):

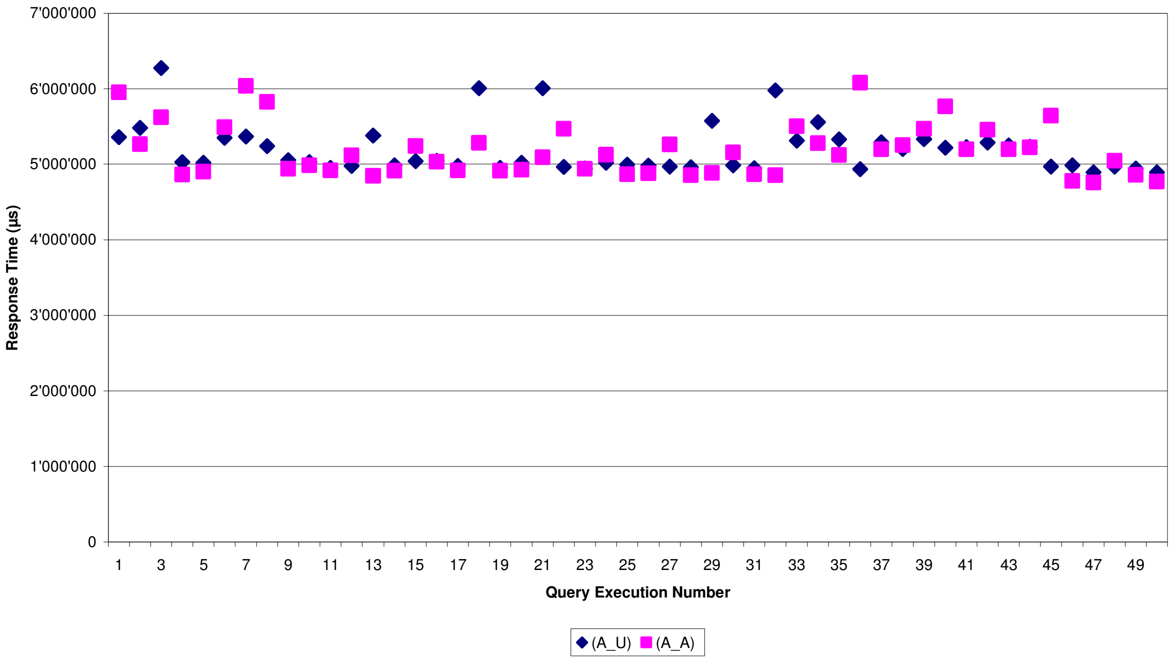
```
SELECT IDENT AS CHARGING_ID, NULL AS METERING_ID
  FROM TSMT_DETAILED_CHARGING
UNION
SELECT NULL AS CHARGING_ID, IDENT AS METERING_ID
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    13577      0.51       0.59        230        303          2      135766
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    13579      0.51       0.59        230        303          2      135766

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 135766  SORT UNIQUE (cr=303 pr=230 pw=230 time=354236 us)
 135766   UNION-ALL  (cr=303 pr=0 pw=0 time=135803 us)
  33183    INDEX FAST FULL SCAN PK_T_SMT_DC (cr=75 pr=0 pw=0 time=30 us)(object id 
83890)
 102583    INDEX FAST FULL SCAN PK_T_SMT_DM (cr=228 pr=0 pw=0 time=27 us)(object id 
83892)
```

Median 5,119,000 µs (A_A):

```
SELECT IDENT AS CHARGING_ID, NULL AS METERING_ID
  FROM TSMT_DETAILED_CHARGING
UNION ALL
SELECT NULL AS CHARGING_ID, IDENT AS METERING_ID
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    13577      0.26       0.47          0      13850          0      135766
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    13579      0.26       0.47          0      13850          0      135766

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 135766  UNION-ALL  (cr=13850 pr=0 pw=0 time=543605 us)
  33183   INDEX FAST FULL SCAN PK_T_SMT_DC (cr=3387 pr=0 pw=0 time=99638 us)(object id 
83890)
 102583   INDEX FAST FULL SCAN PK_T_SMT_DM (cr=10463 pr=0 pw=0 time=102617 us)(object 
id 83892)
```



(A_U)
(A_A)
5000000
5500000
6000000



Response Time (µs)
7'000'000
6'000'000
5'000'000
4'000'000
3'000'000
2'000'000
1'000'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

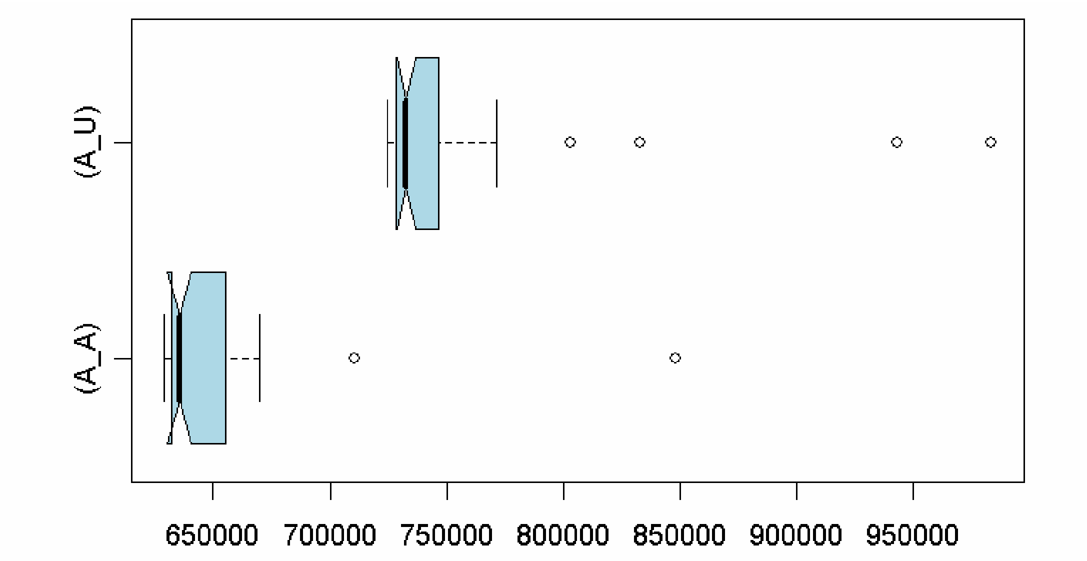
TQP_80123: Use the set operator `UNION ALL` instead of `UNION`: `SELECT DISTINCT` instead of `UNION`

Median 732,800 µs (A_U):

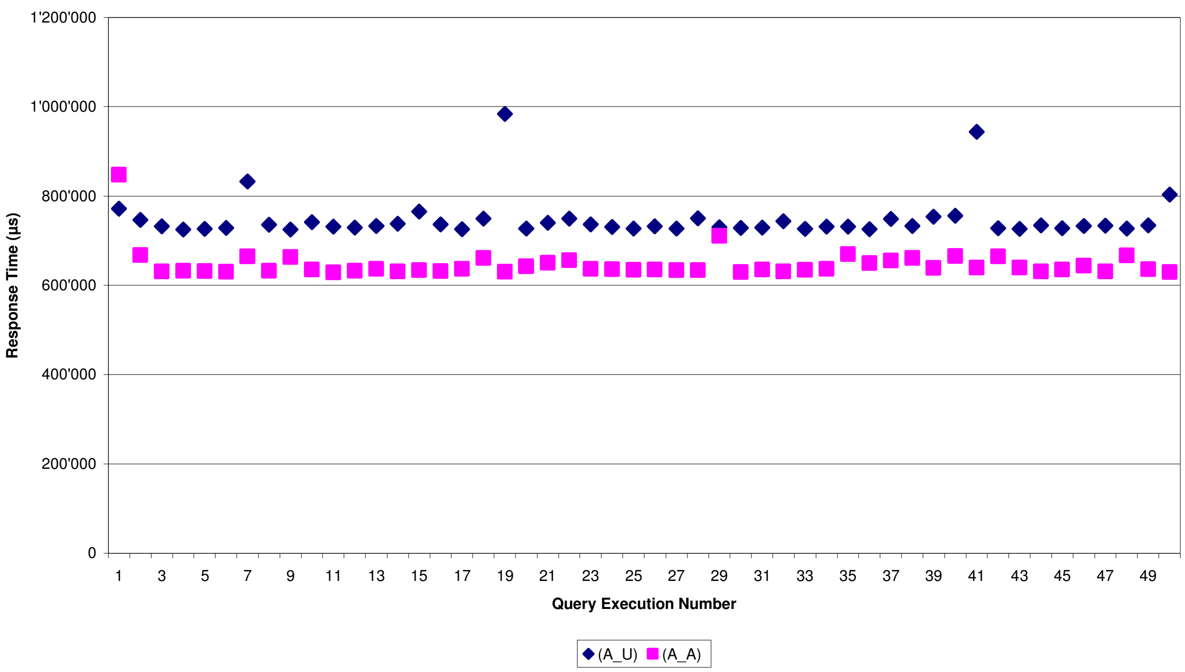
```
SELECT AMOUNT_SERVICE AS CHARGING, NULL AS METERING
  FROM TSMT_DETAILED_CHARGING
 UNION
SELECT NULL AS CHARGING, AMOUNT_SERVICE AS METERING
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1338      0.26       0.28       2640       2647          0       13379
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1340      0.26       0.28       2640       2647          0       13379

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  13379  SORT UNIQUE (cr=2647 pr=2640 pw=0 time=275679 us)
 135766   UNION-ALL  (cr=2647 pr=2640 pw=0 time=271819 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=690 pw=0 time=33463 us)
 102583    TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1950 pw=0 time=102776
us)
```

Median 632,000 µs (A_A):

```
SELECT DISTINCT *
  FROM (SELECT AMOUNT_SERVICE AS CHARGING, NULL AS METERING
          FROM TSMT_DETAILED_CHARGING
         UNION ALL
        SELECT NULL AS CHARGING, AMOUNT_SERVICE AS METERING
          FROM TSMT_DETAILED_METERING)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1338      0.20       0.19       2639       2647          0       13379
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1340      0.20       0.19       2639       2647          0       13379

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  13379  HASH UNIQUE (cr=2647 pr=2639 pw=0 time=185364 us)
 135766   VIEW  (cr=2647 pr=2639 pw=0 time=407591 us)
 135766    UNION-ALL  (cr=2647 pr=2639 pw=0 time=271823 us)
  33183     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=689 pw=0 time=33467
us)
 102583     TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1950 pw=0 time=102774
us)
```



(A_U)
(A_A)
650000
700000
750000
800000
850000
900000
950000



1'200'000
1'000'000
800'000
600'000
400'000
200'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

Avoid Explicit Sorting

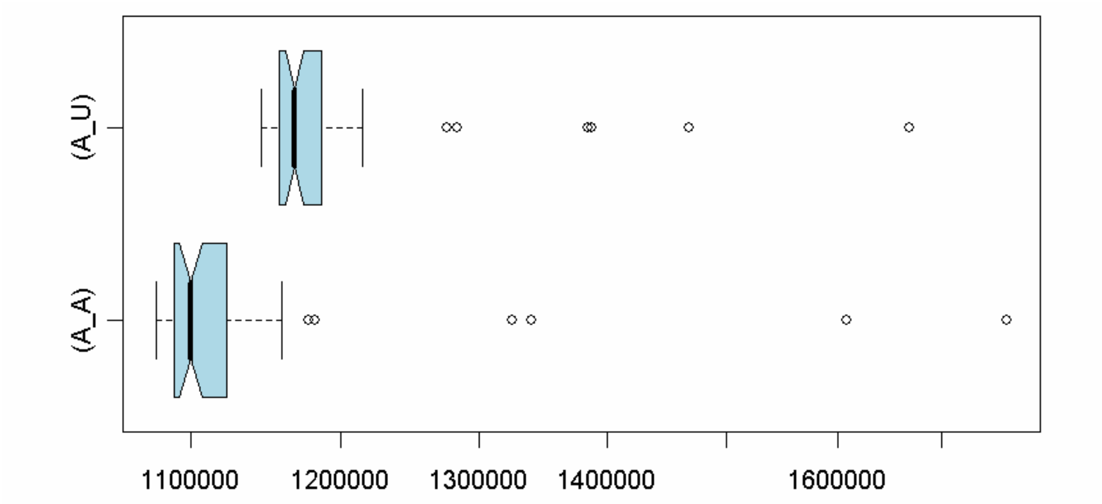
TQP_80901: A `DISTINCT` clause matching an index may cover sorting requirements

Median 1,168,000 µs (A_U):

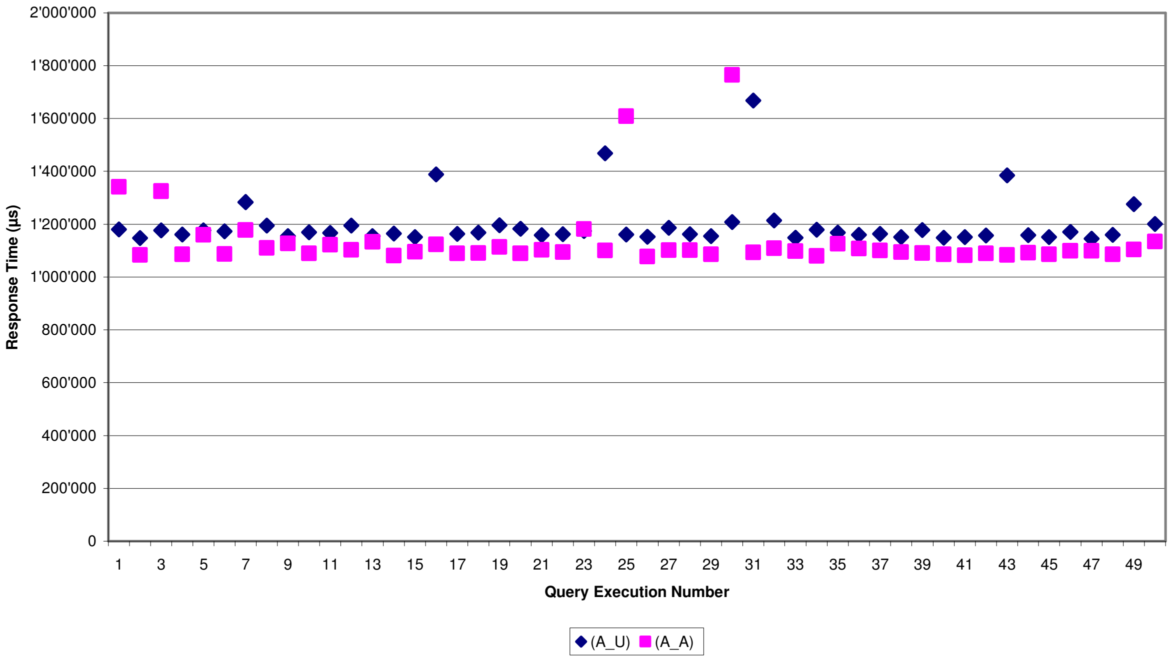
```
SELECT DISTINCT COMPONENT_ID, IDENT
  FROM TSMT_DETAILED_CHARGING
 ORDER BY COMPONENT_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.11          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.11          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  SORT UNIQUE (cr=693 pr=0 pw=0 time=95990 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33227 us)
```

Median 1,100,000 µs (A_A):

```
SELECT DISTINCT COMPONENT_ID, IDENT
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.10       0.06          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.10       0.06          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH UNIQUE (cr=693 pr=0 pw=0 time=55837 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33244 us)
```



(Logarithmic scale)



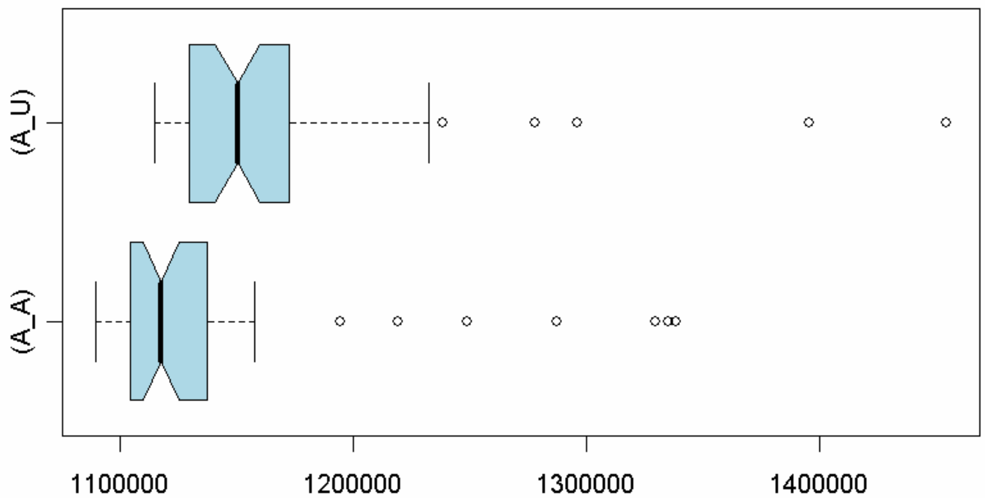
TQP_80902: A `DISTINCT` clause matching an index may cover sorting requirements - indexed column and primary key

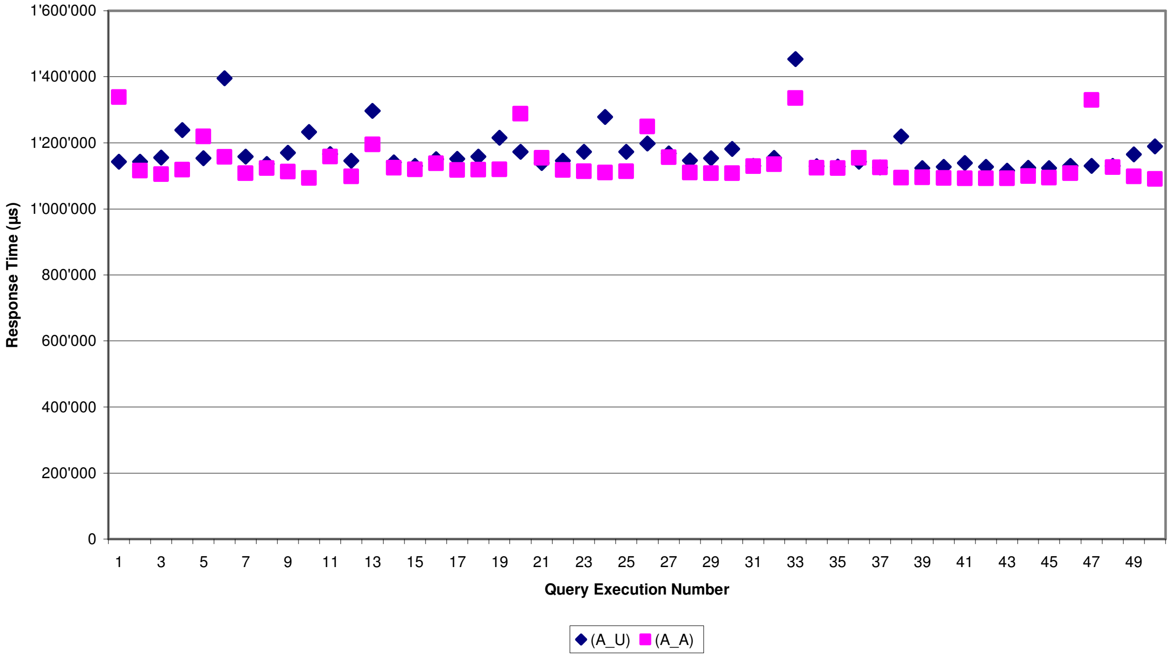
Median 1,150,000 µs (A_U):

```
SELECT DISTINCT SERVICE_RECEIVER_ID, IDENT
  FROM TSMT_DETAILED_CHARGING
 ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.07       0.09          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.07       0.09          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  SORT UNIQUE (cr=693 pr=0 pw=0 time=85237 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33220 us)
```

Median 1,118,000 µs (A_A):

```
SELECT DISTINCT SERVICE_RECEIVER_ID, IDENT
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.07       0.06          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.07       0.06          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH UNIQUE (cr=693 pr=0 pw=0 time=56395 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33254 us)
```





1'600'000
1'400'000
1'200'000
1'000'000
800'000
600'000
400'000
200'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

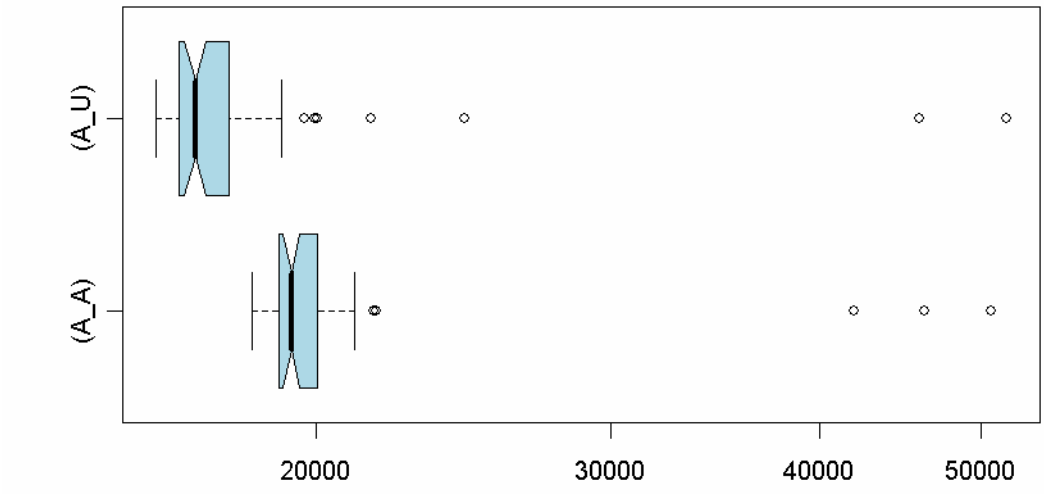
TQP_80903: A `DISTINCT` clause matching an index may cover sorting requirements - indexed column

Median 16,920 µs (A_U):

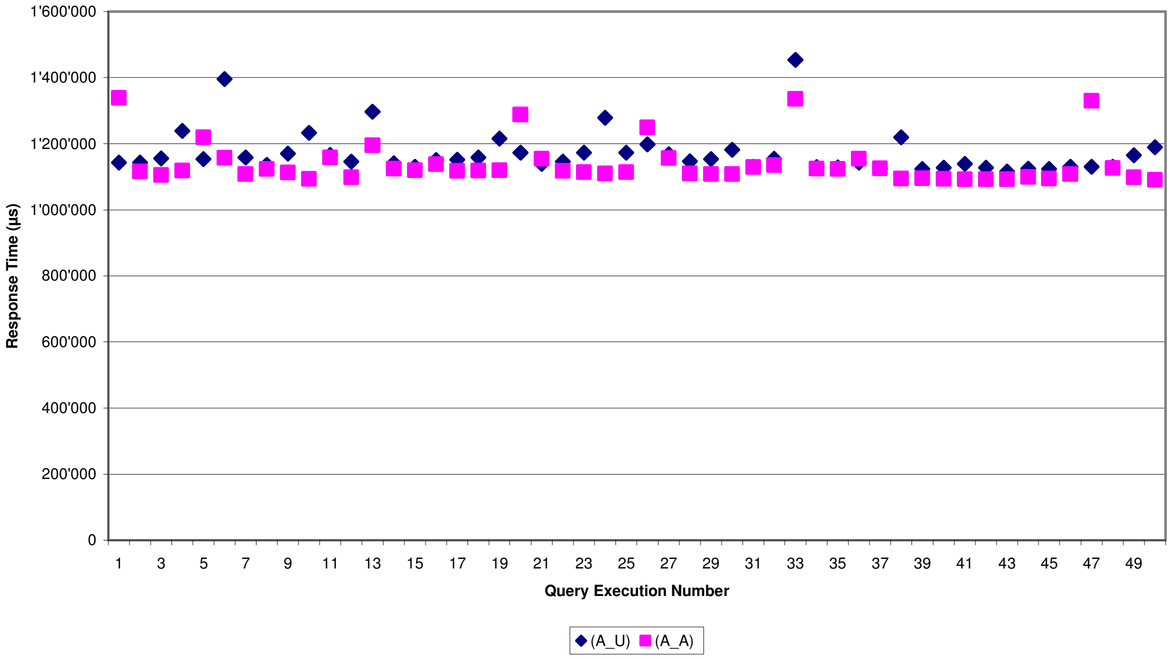
```
SELECT DISTINCT SERVICE_RECEIVER_ID
  FROM TSMT_DETAILED_CHARGING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       14      0.01       0.01          0         73          0         131
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       16      0.01       0.01          0         73          0         131

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    131  HASH UNIQUE (cr=73 pr=0 pw=0 time=11909 us)
  33183   INDEX FAST FULL SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=73 pr=0 pw=0 time=36
us)(object id 89036)
```

Median 19,320 µs (A_A):

```
SELECT DISTINCT SERVICE_RECEIVER_ID
  FROM TSMT_DETAILED_CHARGING
 ORDER BY SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       14      0.00       0.01          0         73          0         131
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       16      0.00       0.01          0         73          0         131

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    131  SORT UNIQUE (cr=73 pr=0 pw=0 time=9977 us)
  33183   INDEX FAST FULL SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=73 pr=0 pw=0 time=37
us)(object id 89036)
```



(Logarithmic scale)



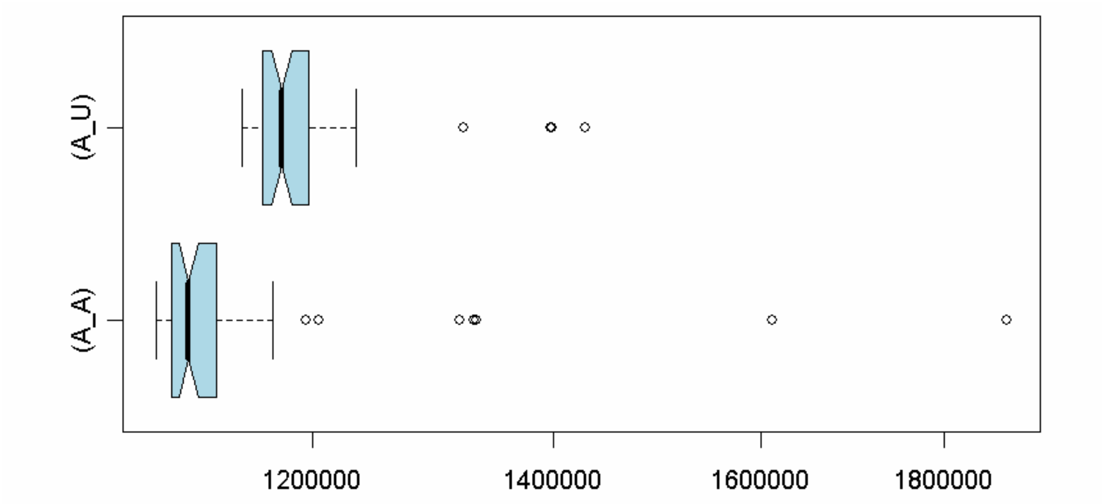
TQP_80911: A GROUP BY clause matching an index may cover sorting requirements

Median 1,177,000 µs (A_U):

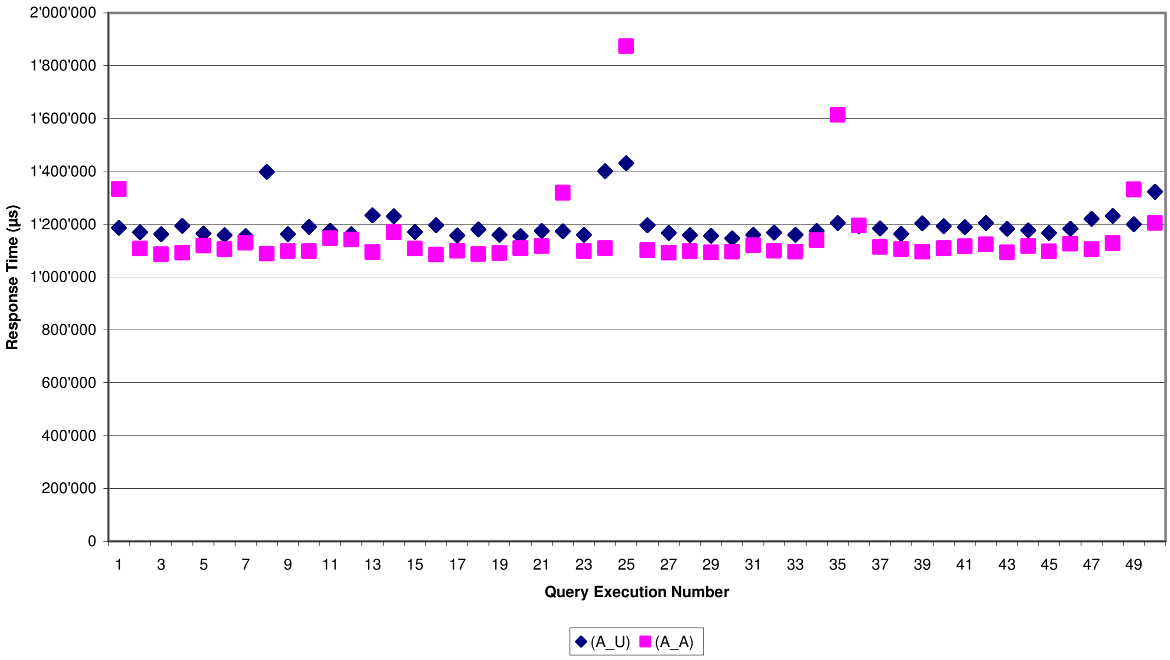
```
SELECT COMPONENT_ID, IDENT
  FROM TSMT_DETAILED_CHARGING
 GROUP BY COMPONENT_ID, IDENT
 ORDER BY COMPONENT_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.14       0.11          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.14       0.11          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  SORT GROUP BY (cr=693 pr=0 pw=0 time=98419 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33220 us)
```

Median 1,108,000 µs (A_A):

```
SELECT COMPONENT_ID, IDENT
  FROM TSMT_DETAILED_CHARGING
 GROUP BY COMPONENT_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.06          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.06          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH GROUP BY (cr=693 pr=0 pw=0 time=54236 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33257 us)
```



(Logarithmic scale)



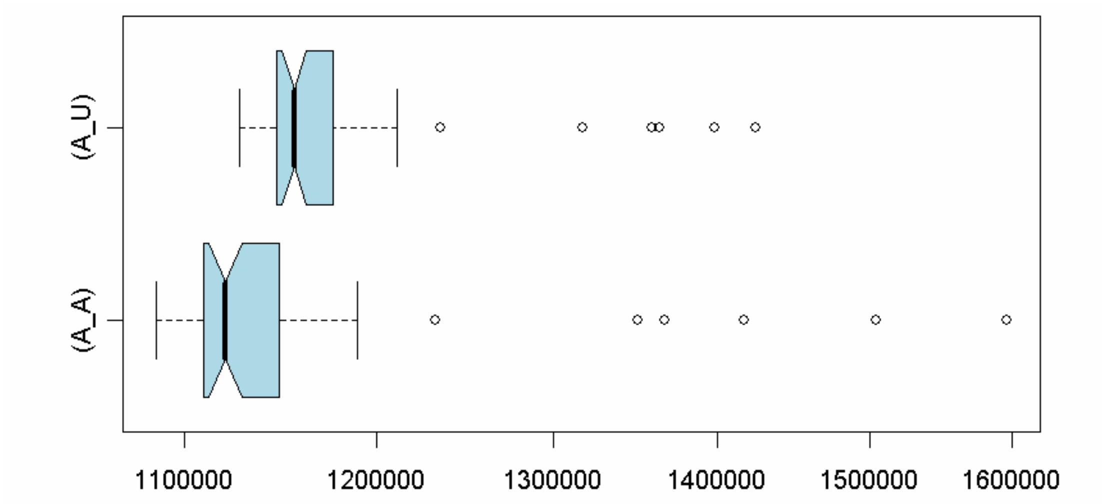
TQP_80912: A GROUP BY clause matching an index may cover sorting requirements - indexed column and primary key

Median 1,156,000 µs (A_U):

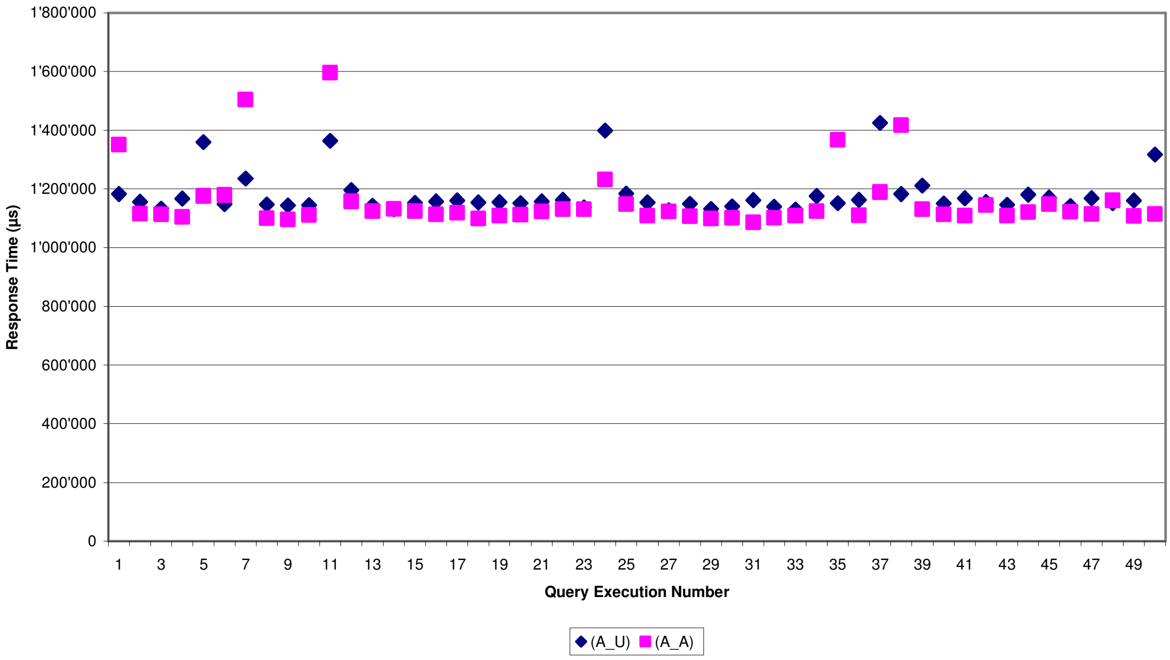
```
SELECT SERVICE_RECEIVER_ID, IDENT
  FROM TSMT_DETAILED_CHARGING
 GROUP BY SERVICE_RECEIVER_ID, IDENT
 ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.09          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.09          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  SORT GROUP BY (cr=693 pr=0 pw=0 time=90006 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33220 us)
```

Median 1,120,000 µs (A_A):

```
SELECT SERVICE_RECEIVER_ID, IDENT
  FROM TSMT_DETAILED_CHARGING
 GROUP BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.04       0.06          0        693          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.04       0.06          0        693          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH GROUP BY (cr=693 pr=0 pw=0 time=55000 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33246 us)
```



(Logarithmic scale)



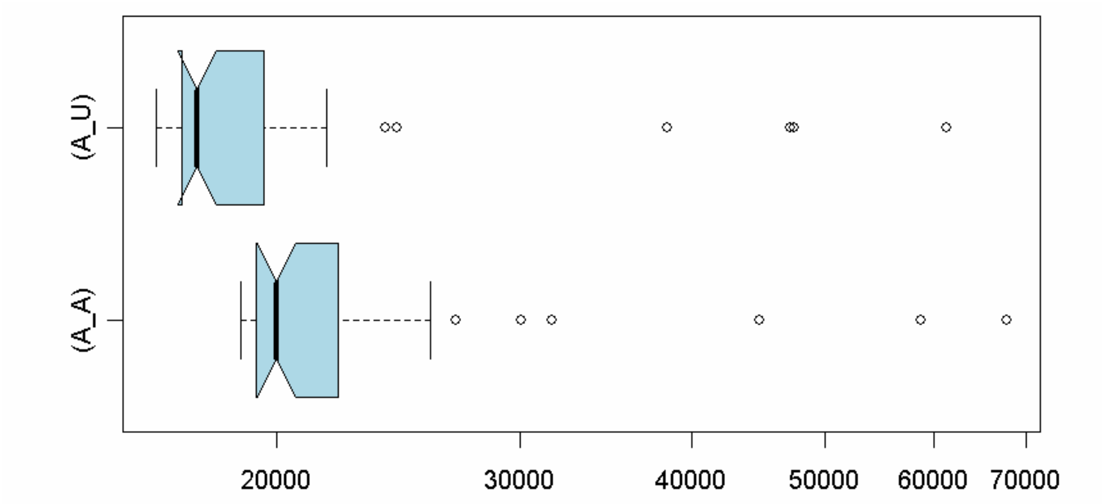
TQP_80913: A GROUP BY clause matching an index may cover sorting requirements - indexed column

Median 17,490 µs (A_U):

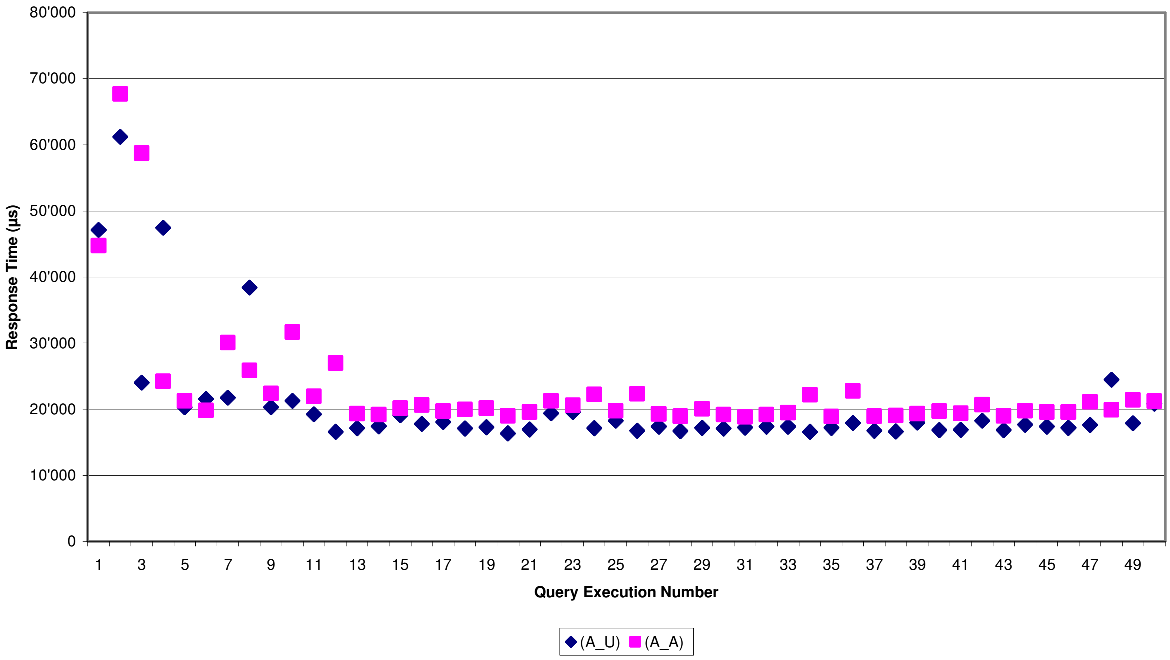
```
SELECT SERVICE_RECEIVER_ID
  FROM TSMT_DETAILED_CHARGING
 GROUP BY SERVICE_RECEIVER_ID
 ORDER BY SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       14      0.01       0.01          0         73          0         131
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       16      0.01       0.01          0         73          0         131

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    131  SORT GROUP BY (cr=73 pr=0 pw=0 time=10209 us)
  33183   INDEX FAST FULL SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=73 pr=0 pw=0 time=38
us)(object id 89036)
```

Median 19,990 µs (A_A):

```
SELECT SERVICE_RECEIVER_ID
  FROM TSMT_DETAILED_CHARGING
 GROUP BY SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       14      0.01       0.01          0         73          0         131
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       16      0.01       0.01          0         73          0         131

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    131  HASH GROUP BY (cr=73 pr=0 pw=0 time=12361 us)
  33183   INDEX FAST FULL SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=73 pr=0 pw=0 time=58
us)(object id 89036)
```



(Logarithmic scale)



Complete `JOIN` Selection

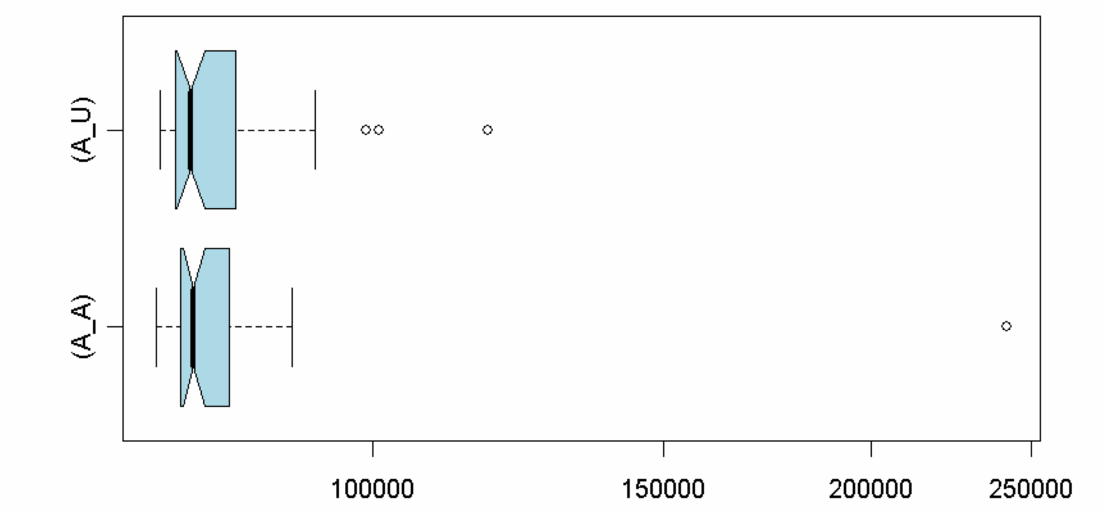
TQP_87401: Maximising the selection information in the `JOIN` and `WHERE` clauses (even if this appears redundant) may support the query optimiser in improving the access plans and in reducing the number of iterations of nested loops - `WHERE` or `JOIN`

Median 77,650 µs (A_U):

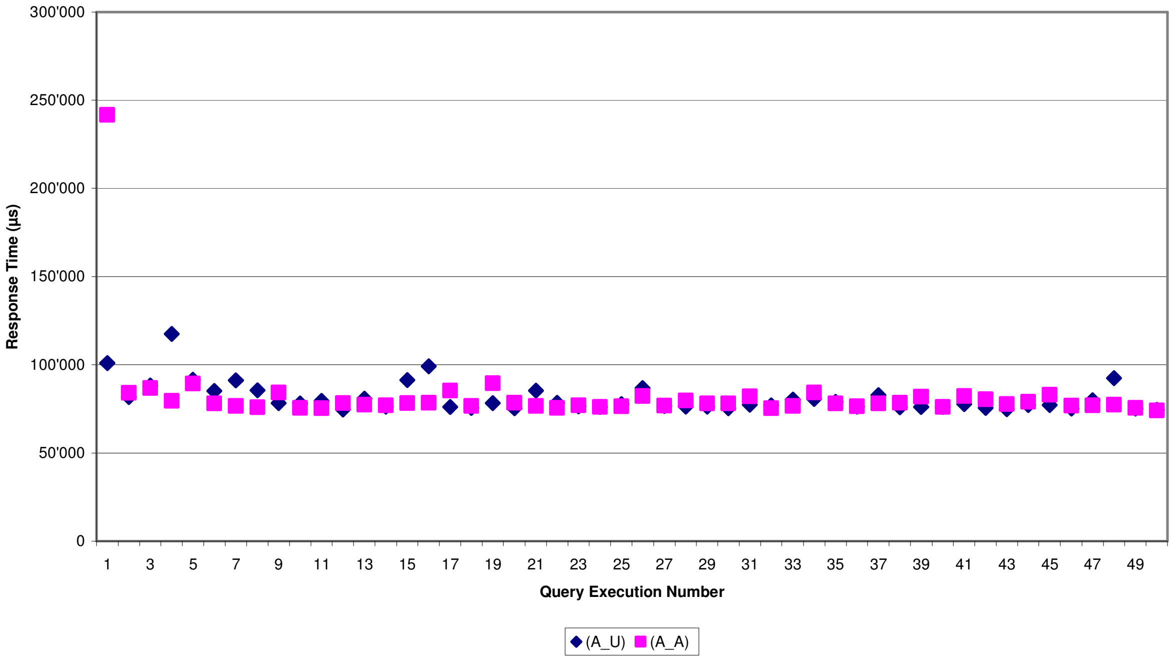
```
SELECT DC.IDENT
  FROM TSMT_CATALOGUE_SERVICE CSE
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON CSE.YEAR = EXTRACT (YEAR FROM DC.PERIOD)
            AND CSE.SERVICE_ID = DC.SERVICE_ID
 WHERE DC.PERIOD = CAST ('01.01.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.02          0       2626          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.02          0       2626          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  NESTED LOOPS  (cr=2626 pr=0 pw=0 time=26756 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=10626 us)
   1611   INDEX UNIQUE SCAN PK_T_SMT_CSE (cr=1774 pr=0 pw=0 time=10005 us)(object id
83844)
```

Median 77,960 µs (A_A):

```
SELECT DC.IDENT
  FROM TSMT_CATALOGUE_SERVICE CSE
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON CSE.YEAR = EXTRACT (YEAR FROM DC.PERIOD)
            AND CSE.SERVICE_ID = DC.SERVICE_ID
            AND DC.PERIOD = CAST ('01.01.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.02          0       2626          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.02          0       2626          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  NESTED LOOPS  (cr=2626 pr=0 pw=0 time=26989 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=10860 us)
   1611   INDEX UNIQUE SCAN PK_T_SMT_CSE (cr=1774 pr=0 pw=0 time=10178 us)(object id
83844)
```



(Logarithmic scale)



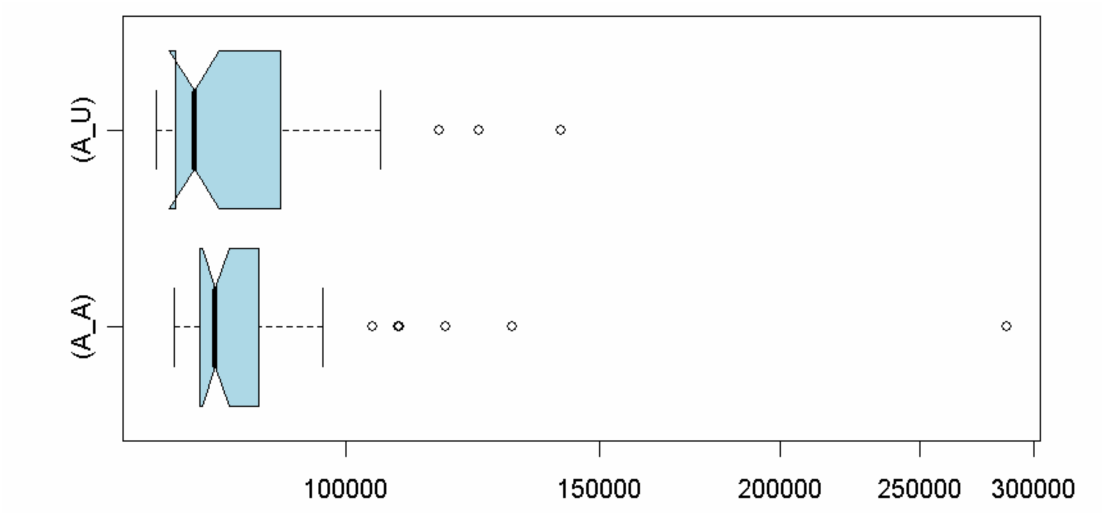
TQP_87402: Maximising the selection information in the `JOIN` and `WHERE` clauses (even if this appears redundant) may support the query optimiser in improving the access plans and in reducing the number of iterations of nested loops - maximum redundancy

Median 78,490 µs (A_U):

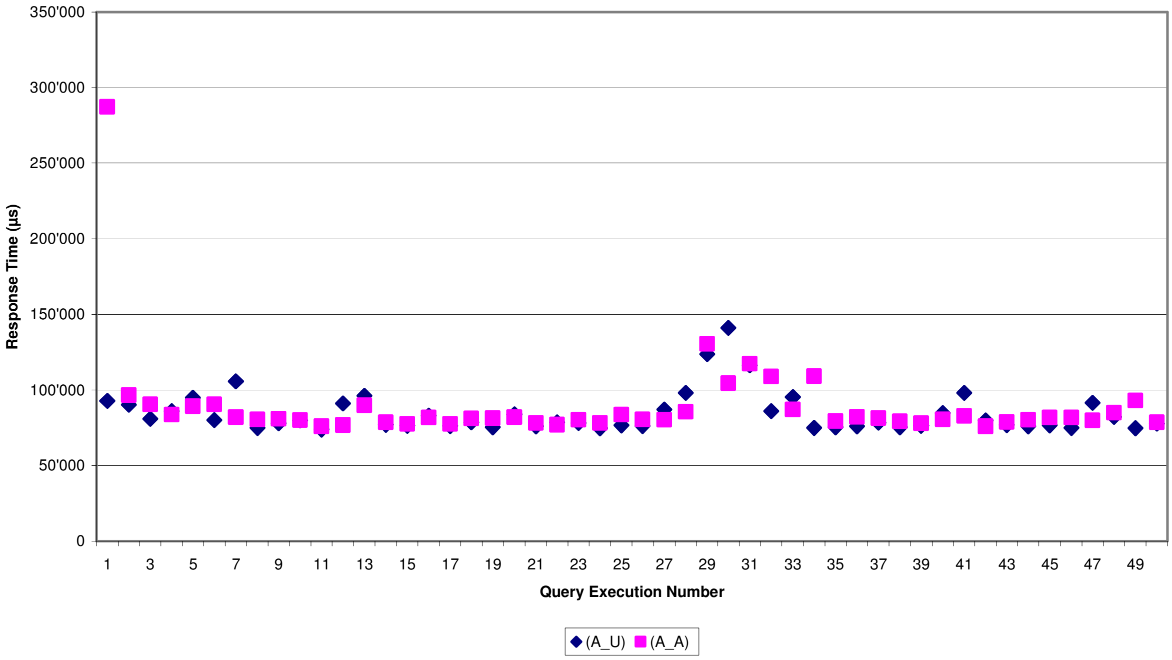
```
SELECT DC.IDENT
  FROM TSMT_CATALOGUE_SERVICE CSE
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON CSE.YEAR = EXTRACT (YEAR FROM DC.PERIOD)
            AND CSE.SERVICE_ID = DC.SERVICE_ID
            AND DC.PERIOD = CAST ('01.01.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.03       0.02          0       2626          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.03       0.02          0       2626          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  NESTED LOOPS  (cr=2626 pr=0 pw=0 time=31985 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=15856 us)
   1611   INDEX UNIQUE SCAN PK_T_SMT_CSE (cr=1774 pr=0 pw=0 time=10123 us)(object id
83844)
```

Median 81,240 µs (A_A):

```
SELECT DC.IDENT
  FROM TSMT_CATALOGUE_SERVICE CSE
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON CSE.YEAR = 2006
            AND CSE.YEAR = EXTRACT (YEAR FROM DC.PERIOD)
            AND CSE.SERVICE_ID = DC.SERVICE_ID
            AND DC.PERIOD = CAST ('01.01.2006' AS DATE)
 WHERE CSE.YEAR = 2006 AND DC.PERIOD = CAST ('01.01.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.02          0       2626          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.02          0       2626          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  NESTED LOOPS  (cr=2626 pr=0 pw=0 time=29604 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=13474 us)
   1611   INDEX UNIQUE SCAN PK_T_SMT_CSE (cr=1774 pr=0 pw=0 time=10082 us)(object id
83844)
```



(Logarithmic scale)



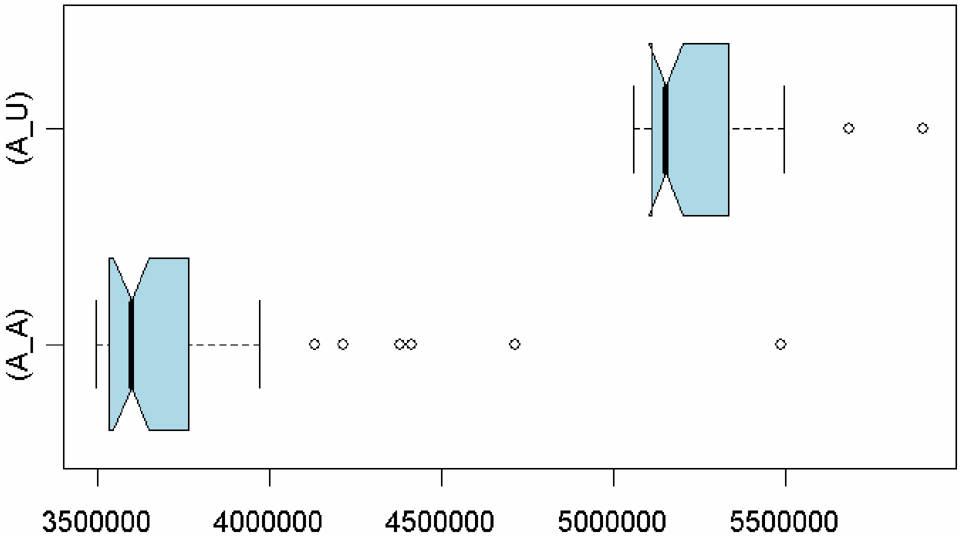
TQP_80301: Be specific and avoid the asterisk in the `SELECT` clause

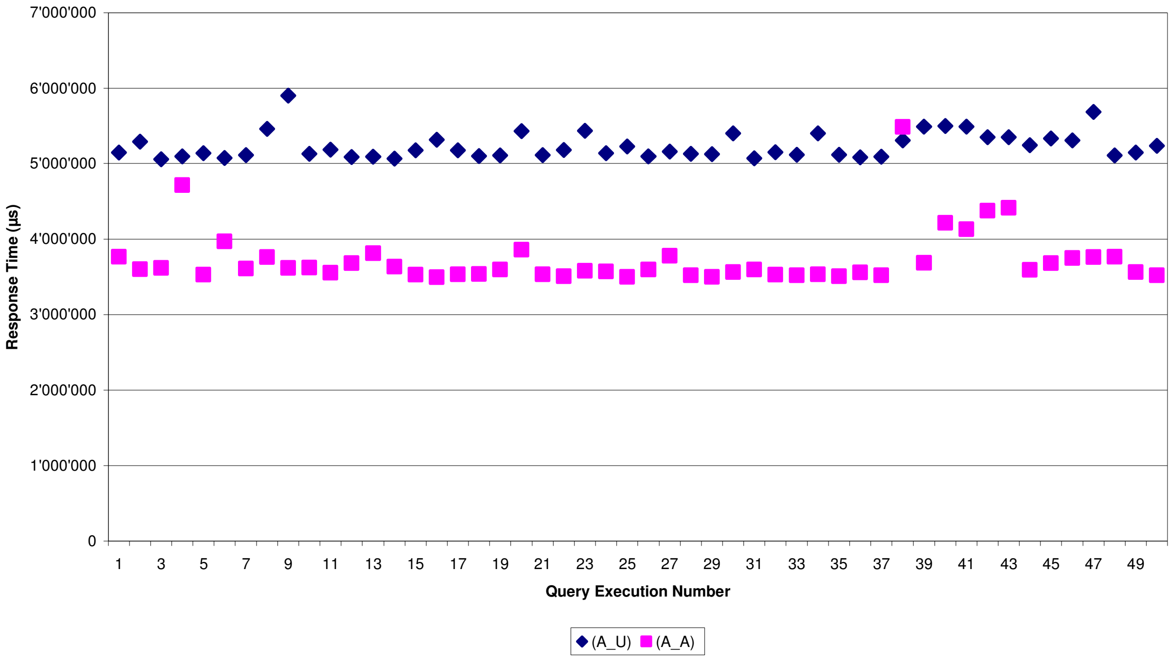
Median 5,153,000 µs (A_U):

```
SELECT *
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.46       0.52       1742      12031          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.46       0.52       1742      12031          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=12031 pr=1742 pw=0 time=308233
us)
```

Median 3,599,000 µs (A_A):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.15       0.20       1749      12031          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.15       0.20       1749      12031          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=12031 pr=1749 pw=0 time=308225
us)
```





7'000'000
6'000'000
5'000'000
4'000'000
3'000'000
2'000'000
1'000'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

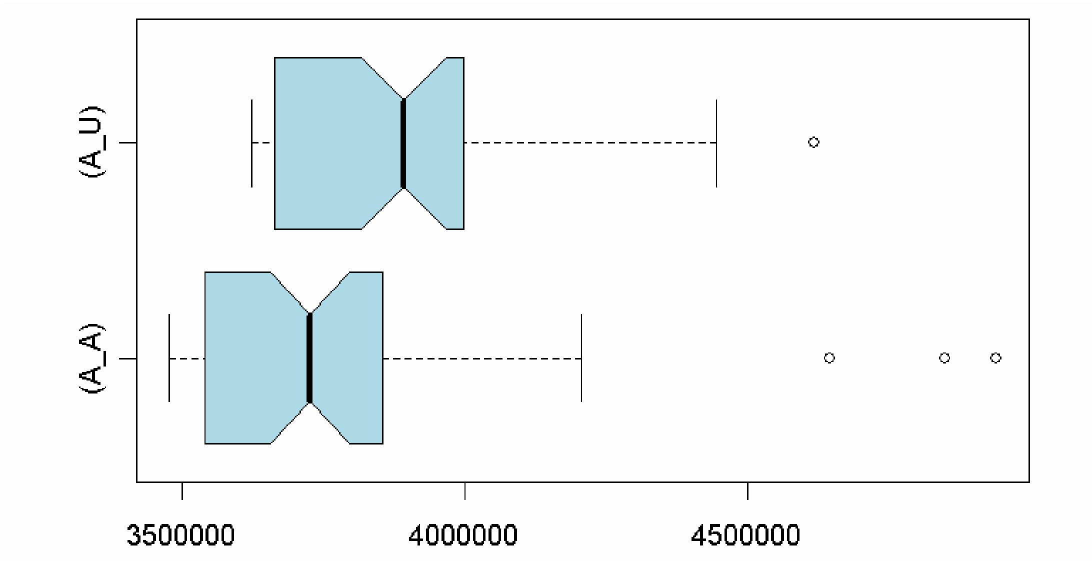
TQP_80311: If possible restrict the columns in the `SELECT` clause to the indexed ones

Median 3,892,000 µs (A_U):

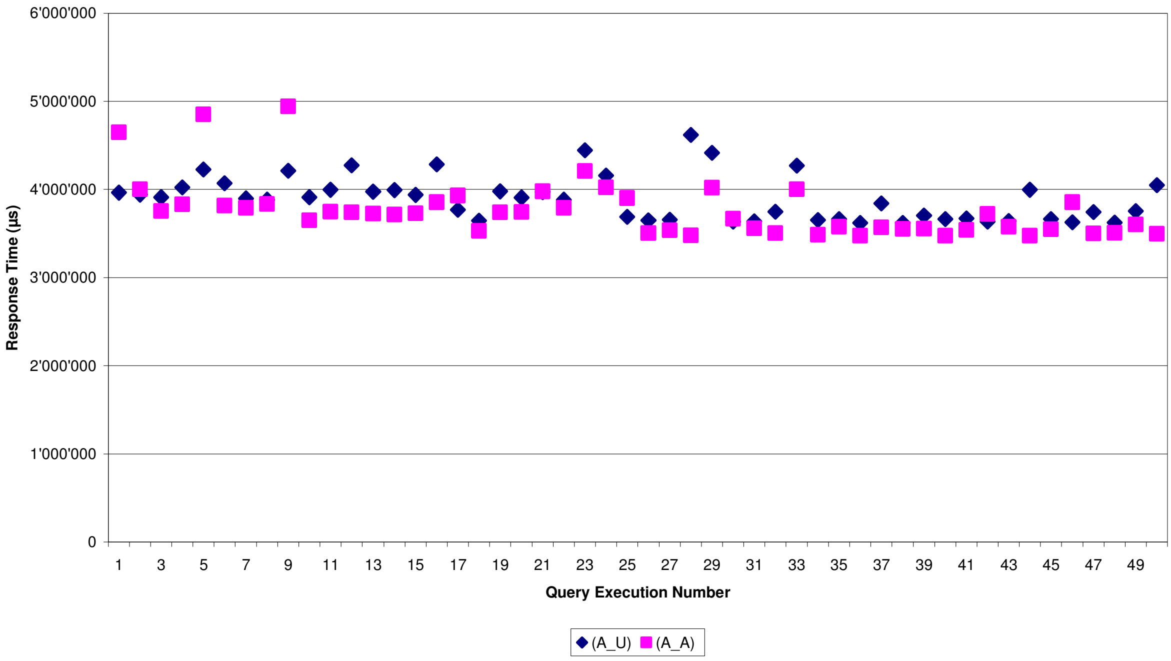
```
SELECT SERVICE_ID, COST_CENTER_CD_DISCHARGING
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.31       0.36       1746      12031          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.31       0.36       1746      12031          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=12031 pr=1746 pw=0 time=410926
us)
```

Median 3,726,000 µs (A_A):

```
SELECT SAPINST_NO, SAP_ORDER_NUMBER
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.29       0.27          0      10480          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.29       0.27          0      10480          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  INDEX FAST FULL SCAN IDX_T_SMT_DM_SAP_ORDER_NUMBER (cr=10480 pr=0 pw=0
time=205416 us)(object id 87254)
```



(A_U)
(A_A)
3500000
4000000
4500000



Response Time (µs)
6'000'000
5'000'000
4'000'000
3'000'000
2'000'000
1'000'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

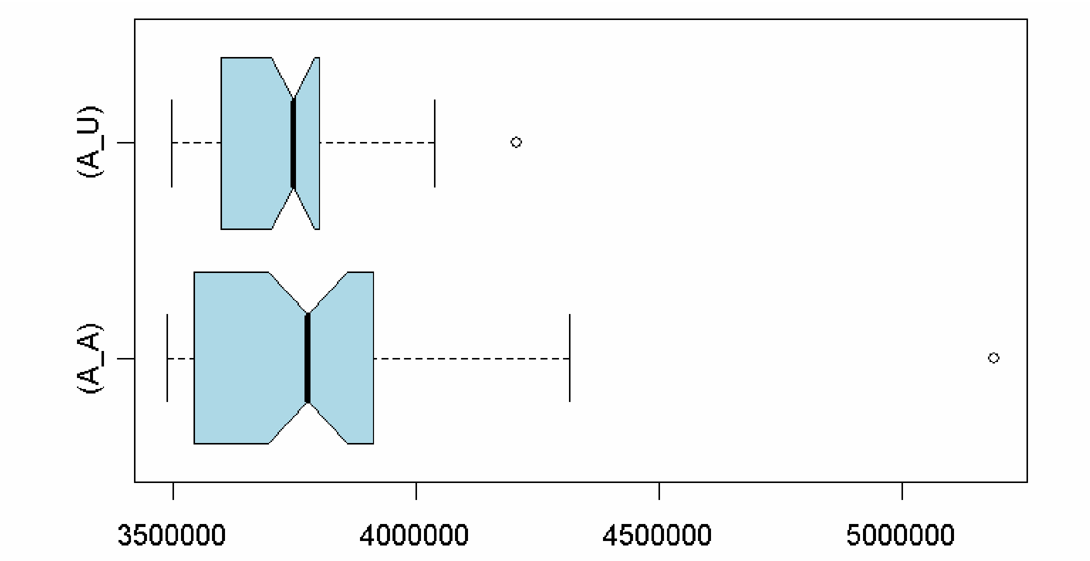
TQP_80321: If possible restrict the columns in the `SELECT` clause to the indexed ones: the order of the columns is irrelevant

Median 3,747,000 µs (A_U):

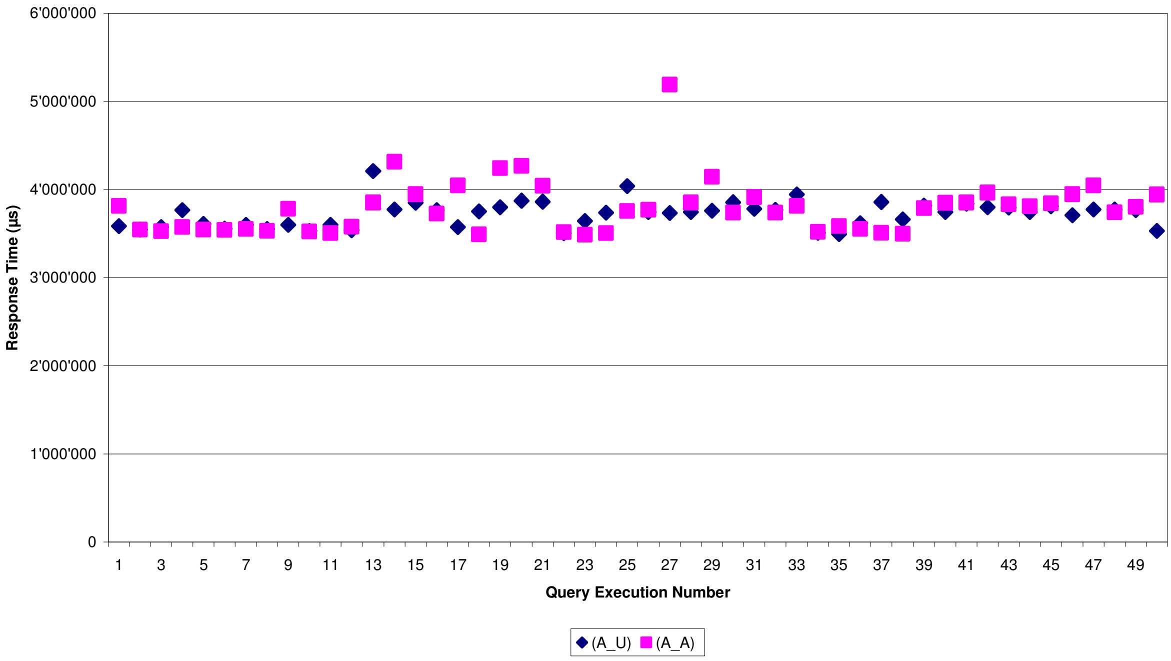
```
SELECT SAPINST_NO, SAP_ORDER_NUMBER
FROM
 TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.20       0.17          0      10480          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.20       0.17          0      10480          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  INDEX FAST FULL SCAN IDX_T_SMT_DM_SAP_ORDER_NUMBER (cr=10480 pr=0 pw=0
time=205373 us)(object id 87254)
```

Median 3,776,000 µs (A_A):

```
SELECT SAP_ORDER_NUMBER, SAPINST_NO
  FROM TSMT_DETAILED_METERING

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.15       0.16          0      10480          0      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.15       0.16          0      10480          0      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  INDEX FAST FULL SCAN IDX_T_SMT_DM_SAP_ORDER_NUMBER (cr=10480 pr=0 pw=0
time=102862 us)(object id 87254)
```



(A_U)
(A_A)
3500000
4000000
4500000
5000000



Response Time (µs)
6'000'000
5'000'000
4'000'000
3'000'000
2'000'000
1'000'000
0
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
Query Execution Number
(A_U)
(A_A)

EXISTS or COUNT

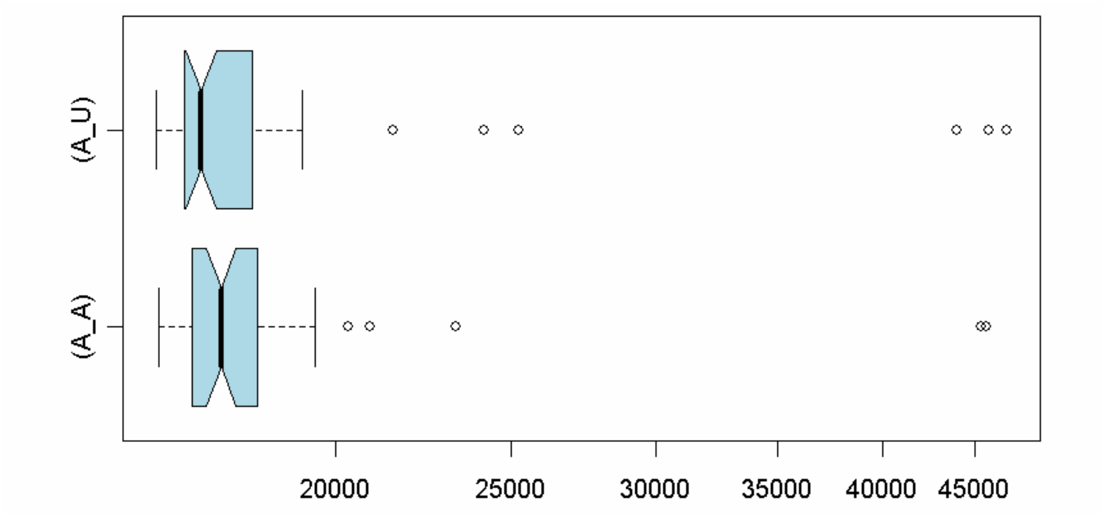
TQP_86401: A query of the form `WHERE EXISTS (SELECT col1 FROM …)` can be rewritten as `0 < (SELECT COUNT(*) FROM …)`

Median 16,870 µs (A_U):

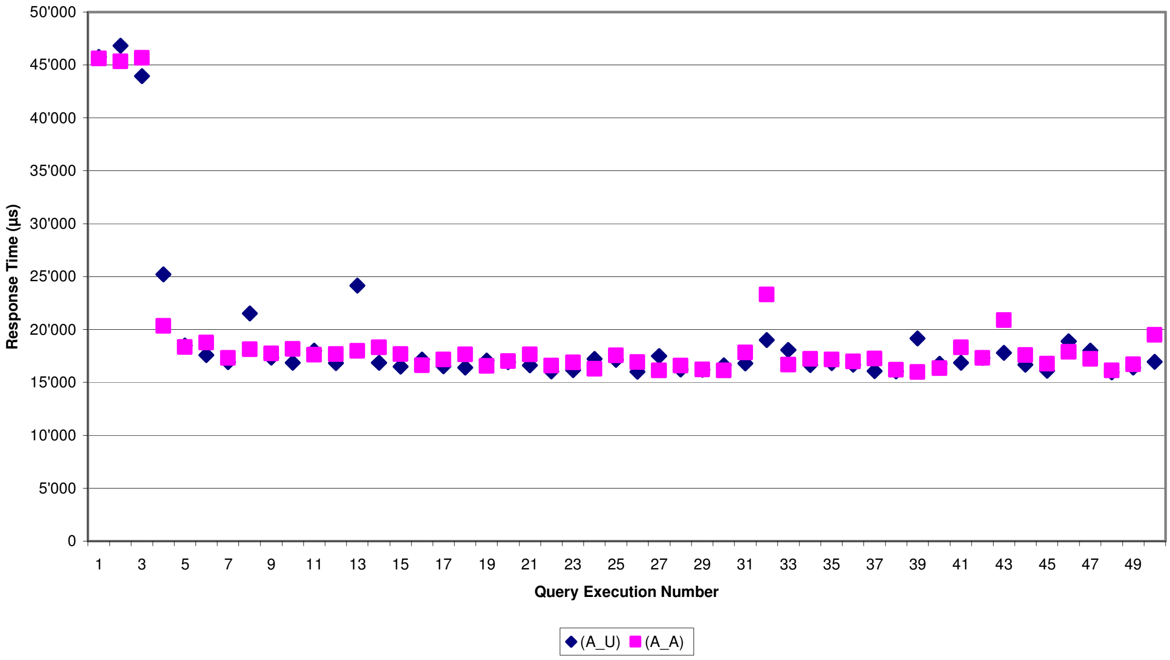
```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND EXISTS (
              SELECT *
                FROM TSMT_DETAILED_CHARGING DC
               WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                 AND DC.SERVICE_ID = CSE.SERVICE_ID
                 AND NOT DC.QUANTITY >= 10000)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        6      0.00       0.01          0        729          0          59
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8      0.00       0.01          0        729          0          59

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     59  HASH JOIN SEMI (cr=729 pr=0 pw=0 time=9907 us)
     95   TABLE ACCESS FULL TSMT_CATALOGUE_SERVICE (cr=31 pr=0 pw=0 time=245 us)
   1572   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=698 pr=0 pw=0 time=9970 us)
```

Median 17,300 µs (A_A):

```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND 0 <
         (SELECT COUNT (*)
            FROM TSMT_DETAILED_CHARGING DC
           WHERE PERIOD = CAST ('01.01.2006' AS DATE)
             AND DC.SERVICE_ID = CSE.SERVICE_ID
             AND NOT DC.QUANTITY >= 10000)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        6      0.00       0.01          0        729          0          59
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8      0.00       0.01          0        729          0          59

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     59  HASH JOIN SEMI (cr=729 pr=0 pw=0 time=9775 us)
     95   TABLE ACCESS FULL TSMT_CATALOGUE_SERVICE (cr=31 pr=0 pw=0 time=244 us)
   1572   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=698 pr=0 pw=0 time=9894 us)
```



(Logarithmic scale)



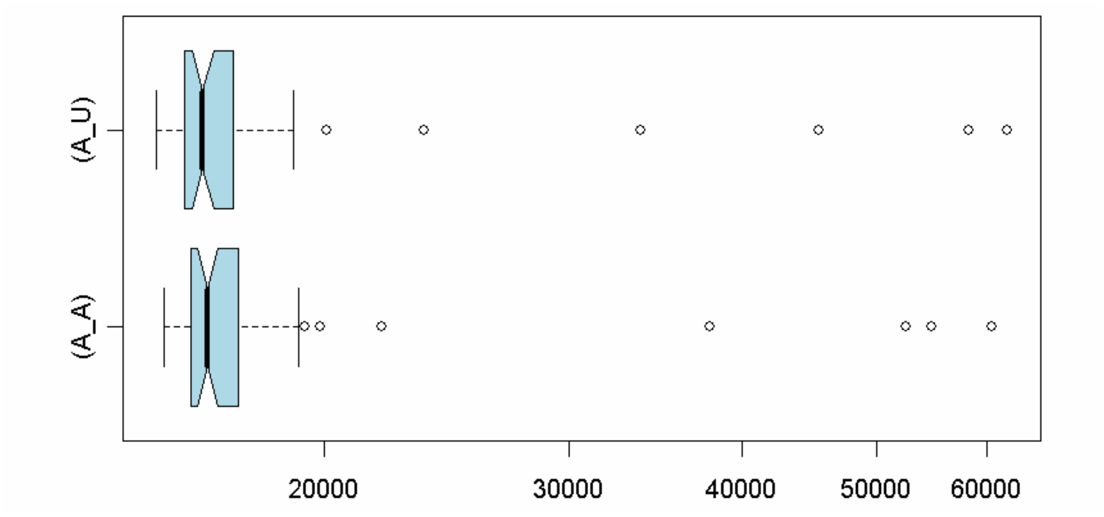
TQP_86411: A query of the form `WHERE NOT EXISTS (SELECT coll FROM …)` can be rewritten as `0 = (SELECT COUNT(*) FROM …)`

Median 16,340 µs (A_U):

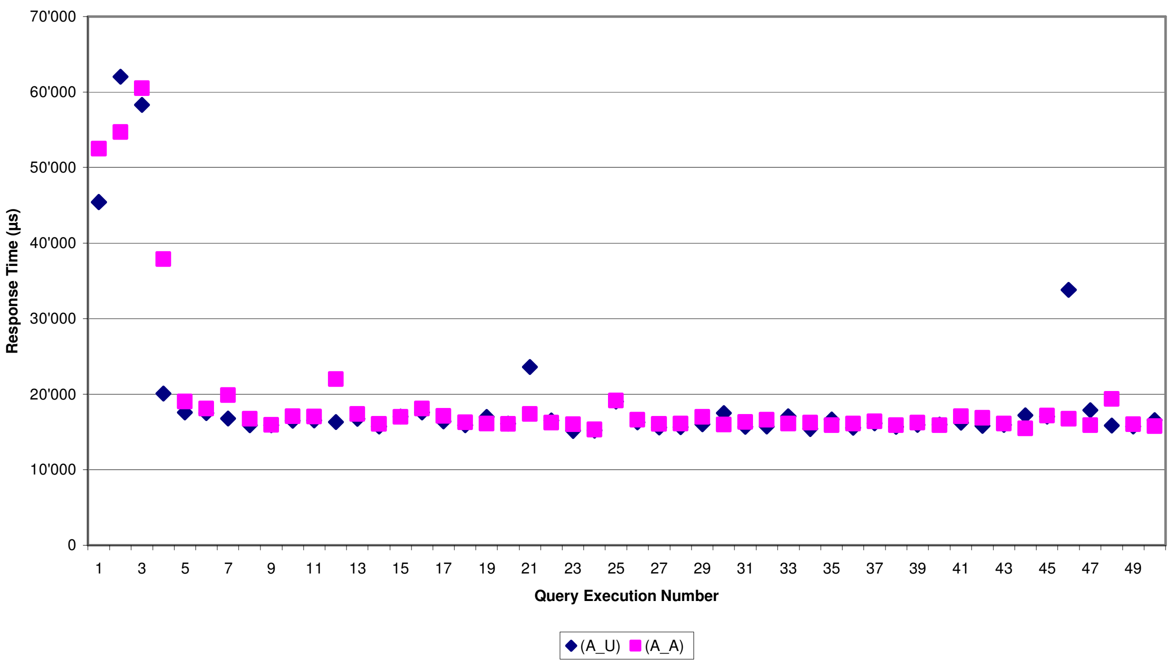
```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND NOT EXISTS (
                  SELECT *
                    FROM TSMT_DETAILED_CHARGING DC
                   WHERE PERIOD = CAST ('01.01.2006' AS DATE)
                     AND DC.SERVICE_ID = CSE.SERVICE_ID
                     AND NOT DC.QUANTITY >= 10000)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        4      0.01       0.01          0        724          0          36
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6      0.01       0.01          0        724          0          36

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     36  HASH JOIN ANTI (cr=724 pr=0 pw=0 time=11712 us)
     95   TABLE ACCESS FULL TSMT_CATALOGUE_SERVICE (cr=31 pr=0 pw=0 time=253 us)
   1572   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=10391 us)
```

Median 16,470 µs (A_A):

```
SELECT CSE.SERVICE_ID, CSE.NAME
  FROM TSMT_CATALOGUE_SERVICE CSE
 WHERE CSE.YEAR = 2006
   AND 0 =
         (SELECT COUNT (*)
            FROM TSMT_DETAILED_CHARGING DC
           WHERE PERIOD = CAST ('01.01.2006' AS DATE)
             AND DC.SERVICE_ID = CSE.SERVICE_ID
             AND NOT DC.QUANTITY >= 10000)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        4      0.01       0.01          0        724          0          36
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6      0.01       0.01          0        724          0          36

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     36  HASH JOIN ANTI (cr=724 pr=0 pw=0 time=12265 us)
     95   TABLE ACCESS FULL TSMT_CATALOGUE_SERVICE (cr=31 pr=0 pw=0 time=319 us)
   1572   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=10990 us)
```



(Logarithmic scale)



EXISTS or IN

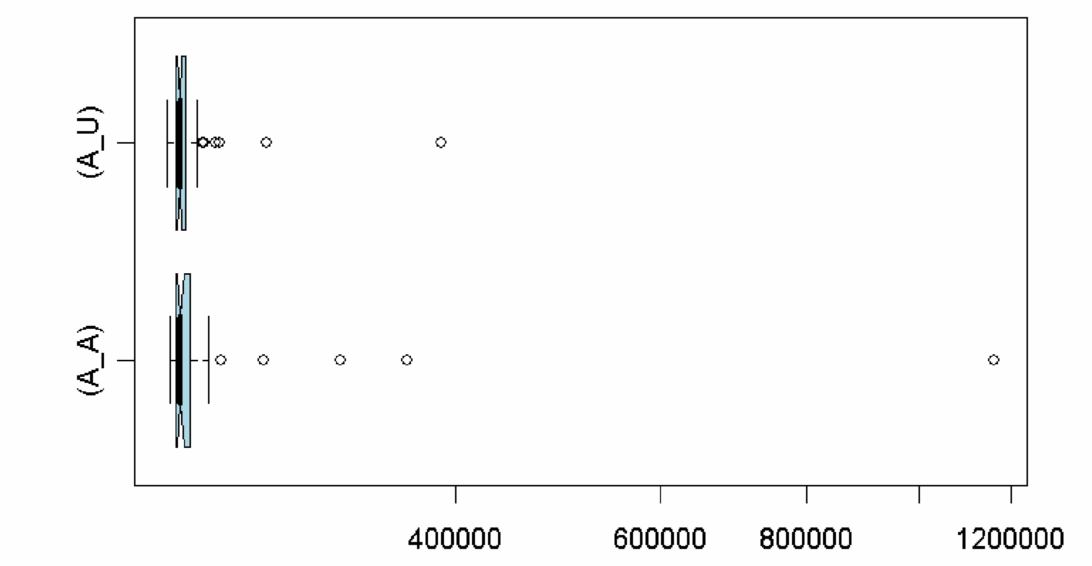
TQP_86301: If the outer table has many rows and the inner table has few rows, then use `IN`

Median 231,600 µs (A_U):

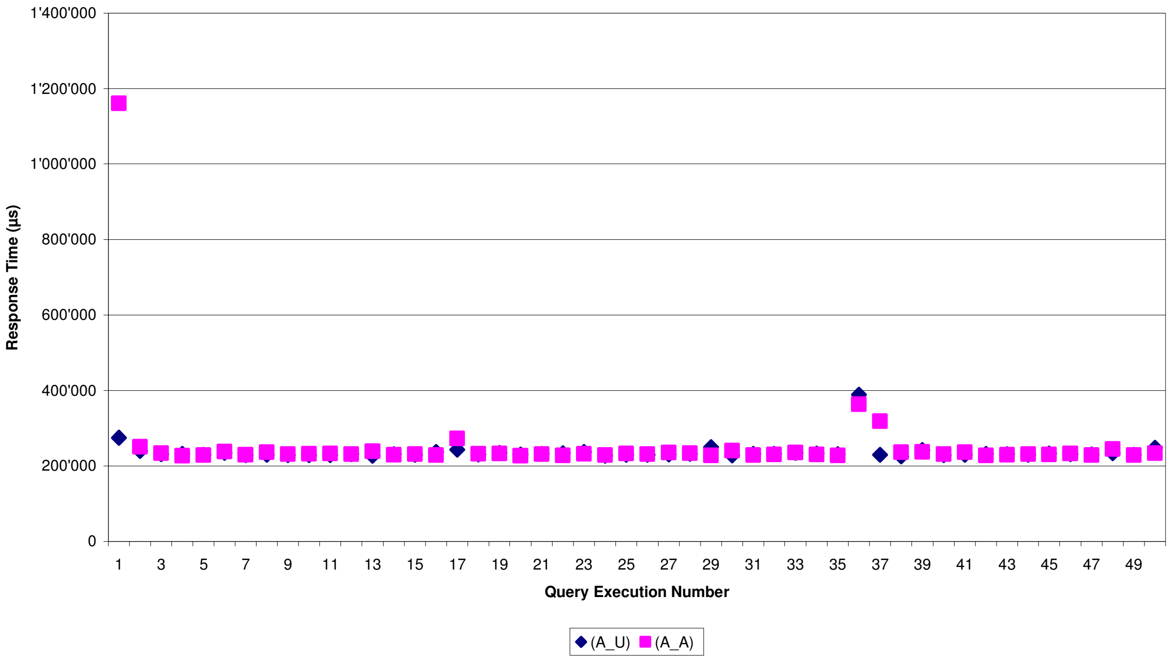
```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE EXISTS (SELECT *
                 FROM TSMT_SERVICE_PROVIDER SEP
                WHERE SEP.HIERARCHY_CD LIKE 'PGIN%'
                  AND SEP.IDENT = DDC.SERVICE_PROVIDER_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      797      0.04       0.03          0       1483          0        7964
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      799      0.04       0.03          0       1483          0        7964

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   7964  HASH JOIN RIGHT SEMI (cr=1483 pr=0 pw=0 time=32486 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=106 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=1476 pr=0 pw=0 time=33228 us)
```

Median 232,000 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE SERVICE_PROVIDER_ID IN (SELECT IDENT
                                 FROM TSMT_SERVICE_PROVIDER
                                WHERE HIERARCHY_CD LIKE 'PGIN%')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      797      0.04       0.03          0       1483          0        7964
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      799      0.04       0.03          0       1483          0        7964

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   7964  HASH JOIN  (cr=1483 pr=0 pw=0 time=32485 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=106 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=1476 pr=0 pw=0 time=33226 us)
```



(Logarithmic scale)



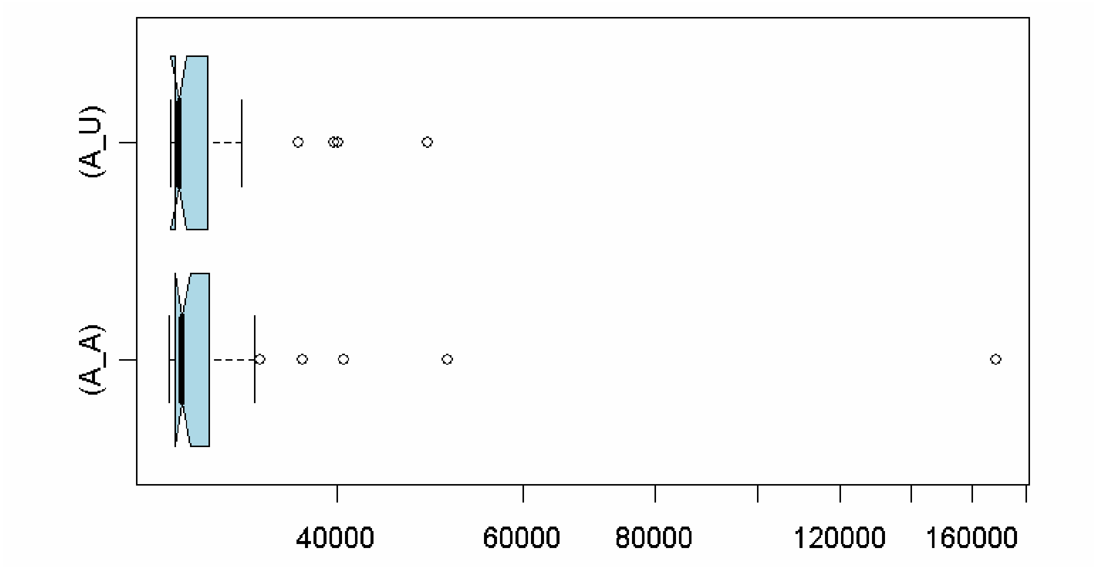
TQP_86302: If the outer table has many rows and the inner table has few rows, then use IN – indexed column

Median 28,290 µs (A_U):

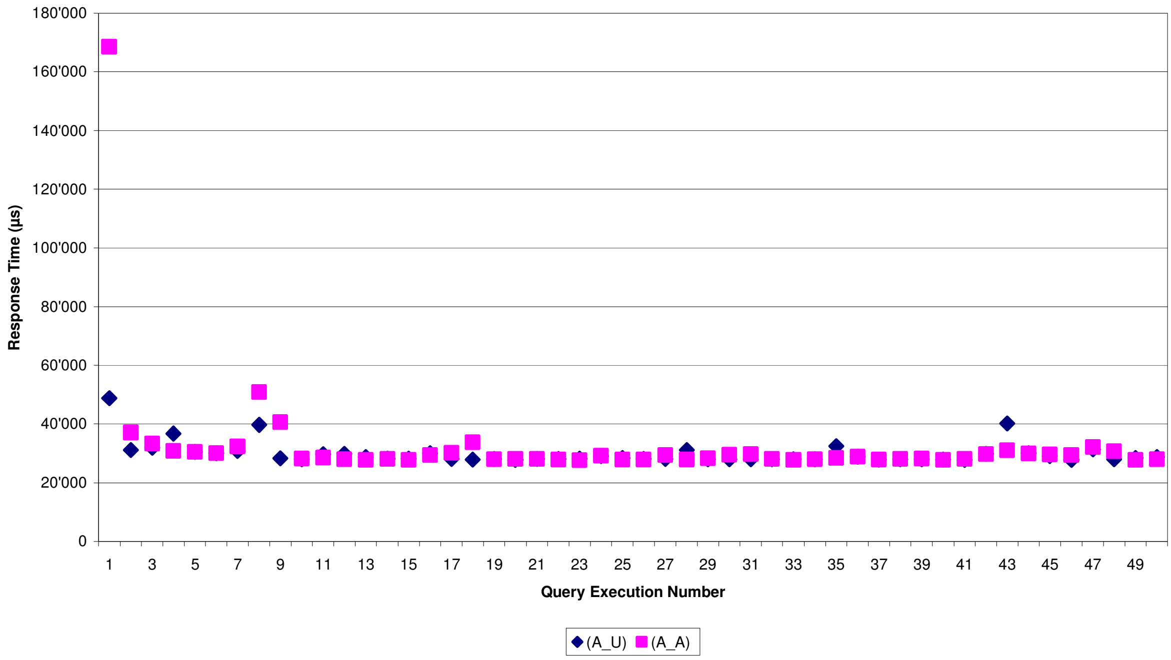
```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE EXISTS (SELECT *
                 FROM TSMT_SERVICE_RECEIVER SER
                WHERE SER.HIERARCHY_CD LIKE 'PGI%'
                  AND SER.IDENT = DDC.SERVICE_RECEIVER_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       92      0.00       0.00          0        324          0         912
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       94      0.00       0.00          0        324          0         912

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    912  TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=324 pr=0 pw=0
time=5806 us)
    963   NESTED LOOPS  (cr=166 pr=0 pw=0 time=29843 us)
     50    SORT UNIQUE (cr=16 pr=0 pw=0 time=279 us)
     50     TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=147 us)
    912    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=150 pr=0 pw=0 time=1245
us)(object id 89036)
```

Median 28,520 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE SERVICE_RECEIVER_ID IN (SELECT IDENT
                                 FROM TSMT_SERVICE_RECEIVER
                                WHERE HIERARCHY_CD LIKE 'PGI%')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       92      0.01       0.00          0        328          0         912
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       94      0.01       0.00          0        328          0         912

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    912  TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=328 pr=0 pw=0
time=5634 us)
    963   NESTED LOOPS  (cr=170 pr=0 pw=0 time=9632 us)
     50    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=20 pr=0 pw=0 time=198 us)
    912    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=150 pr=0 pw=0 time=1270
us)(object id 89036)
```



(Logarithmic scale)



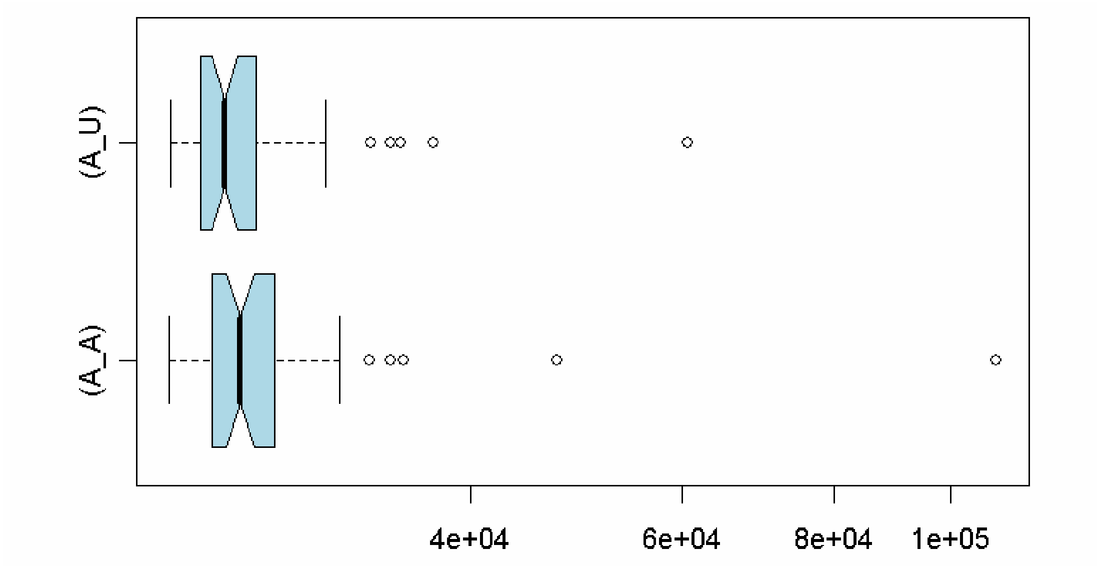
TQP_86303: If most of the rows are filtered by the outer query, then use EXISTS

Median 25,060 µs (A_U):

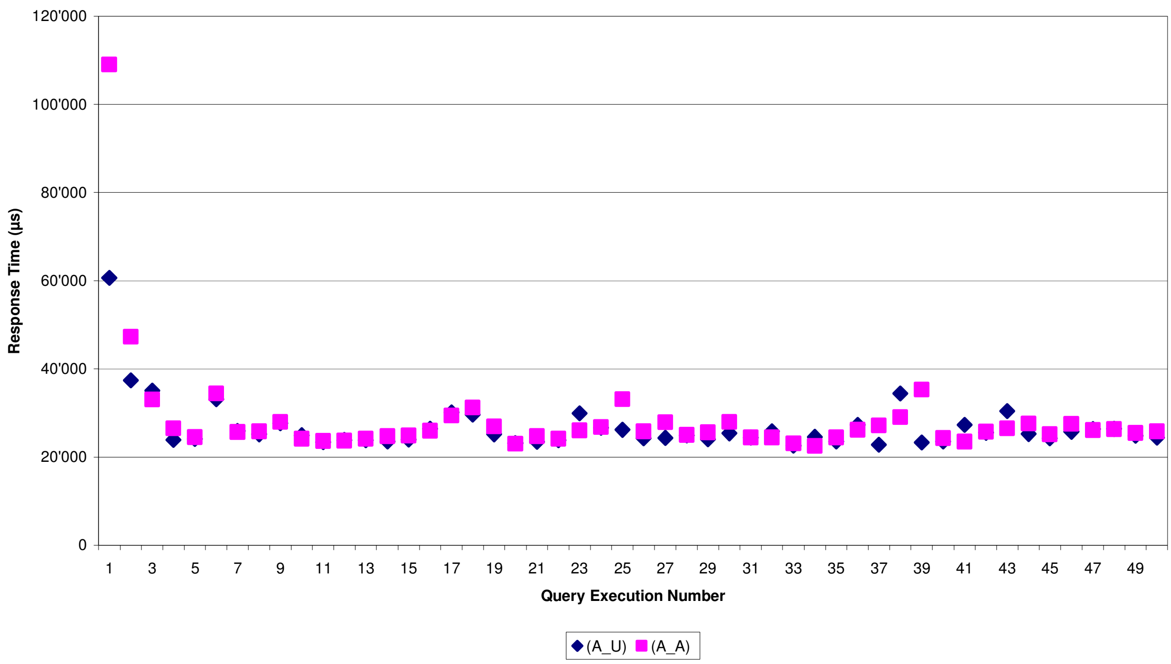
```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND SERVICE_PROVIDER_ID IN (SELECT IDENT
                                 FROM TSMT_SERVICE_PROVIDER
                                WHERE HIERARCHY_CD LIKE 'PGIN%')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       42      0.01       0.01          0        741          0         416
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       44      0.01       0.01          0        741          0         416

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    416  HASH JOIN  (cr=741 pr=0 pw=0 time=9229 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=127 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=734 pr=0 pw=0 time=9759 us)
```

Median 25,850 µs (A_A):

```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE DDC.PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND EXISTS (SELECT *
                 FROM TSMT_SERVICE_PROVIDER SEP
                WHERE SEP.HIERARCHY_CD LIKE 'PGIN%'
                  AND SEP.IDENT = DDC.SERVICE_PROVIDER_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       42      0.01       0.01          0        741          0         416
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       44      0.01       0.01          0        741          0         416

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    416  HASH JOIN RIGHT SEMI (cr=741 pr=0 pw=0 time=9369 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=106 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=734 pr=0 pw=0 time=9421 us)
```



(Logarithmic scale)



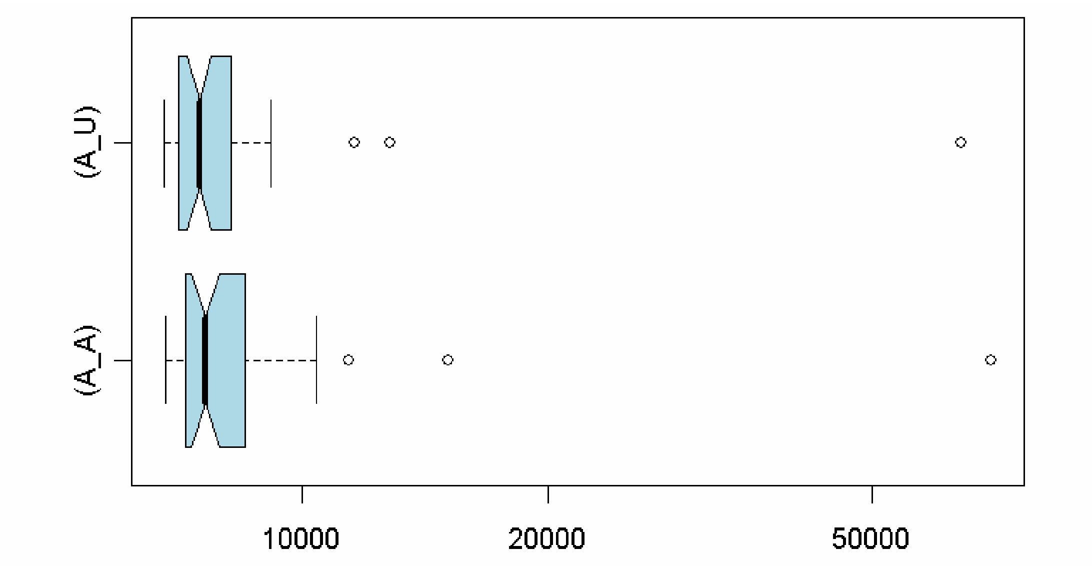
TQP_86304: If most of the rows are filtered by the outer query, then use `EXISTS` – indexed column

Median 7,480 µs (A_U):

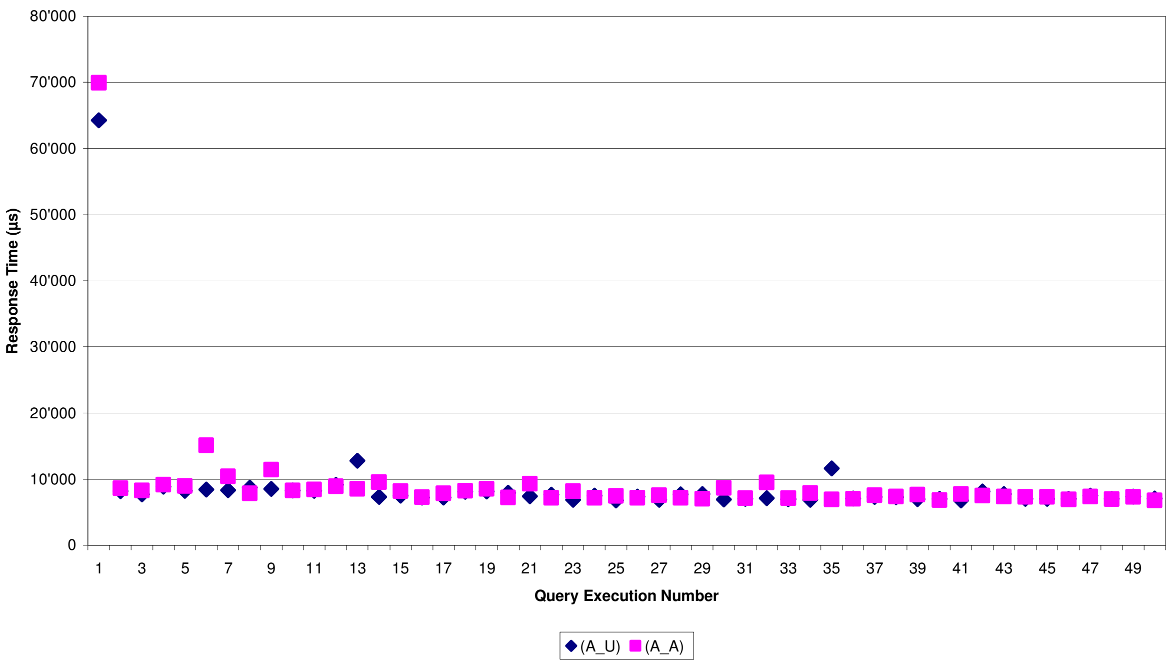
```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND SERVICE_RECEIVER_ID IN (SELECT IDENT
                                 FROM TSMT_SERVICE_RECEIVER
                                WHERE HIERARCHY_CD LIKE 'PGI%')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       10      0.00       0.00          0        175          0          90
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       12      0.00       0.00          0        175          0          90

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     90  TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=175 pr=0 pw=0 time=743
us)
    963   NESTED LOOPS  (cr=88 pr=0 pw=0 time=9631 us)
     50    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=20 pr=0 pw=0 time=198 us)
    912    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=68 pr=0 pw=0 time=1248
us)(object id 89036)
```

Median 7,626 µs (A_A):

```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE DDC.PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND EXISTS (SELECT *
                 FROM TSMT_SERVICE_RECEIVER SER
                WHERE SER.HIERARCHY_CD LIKE 'PGI%'
                  AND SER.IDENT = DDC.SERVICE_RECEIVER_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       10      0.00       0.00          0        171          0          90
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       12      0.00       0.00          0        171          0          90

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     90  TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=171 pr=0 pw=0 time=915
us)
    963   NESTED LOOPS  (cr=84 pr=0 pw=0 time=31765 us)
     50    SORT UNIQUE (cr=16 pr=0 pw=0 time=343 us)
     50     TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=114 us)
    912    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=68 pr=0 pw=0 time=1265
us)(object id 89036)
```



(Logarithmic scale)



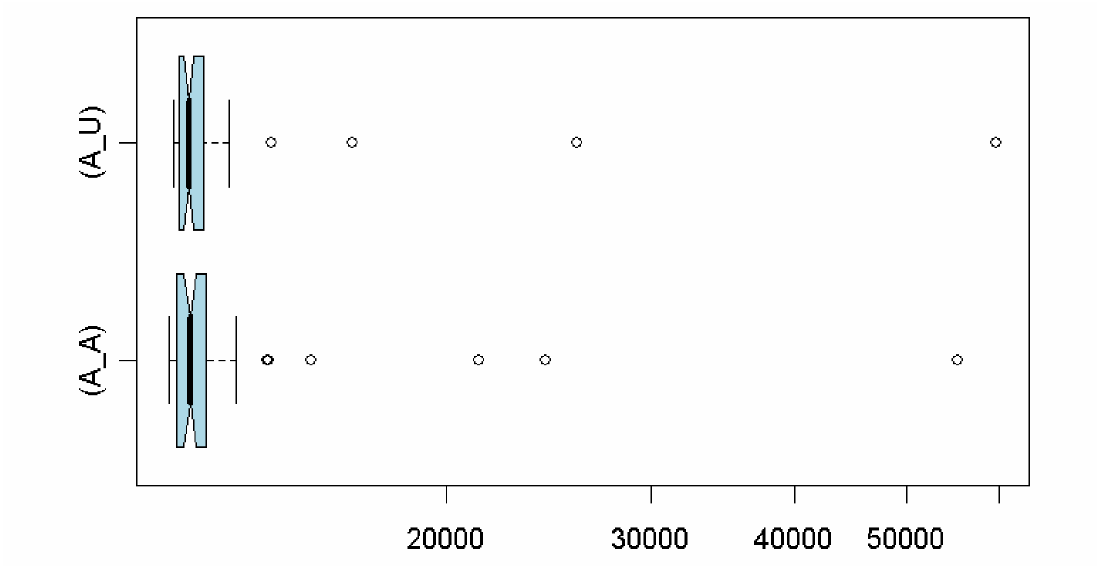
TQP_86305: If most of the rows are filtered by the inner query, then use `IN`

Median 11,980 µs (A_U):

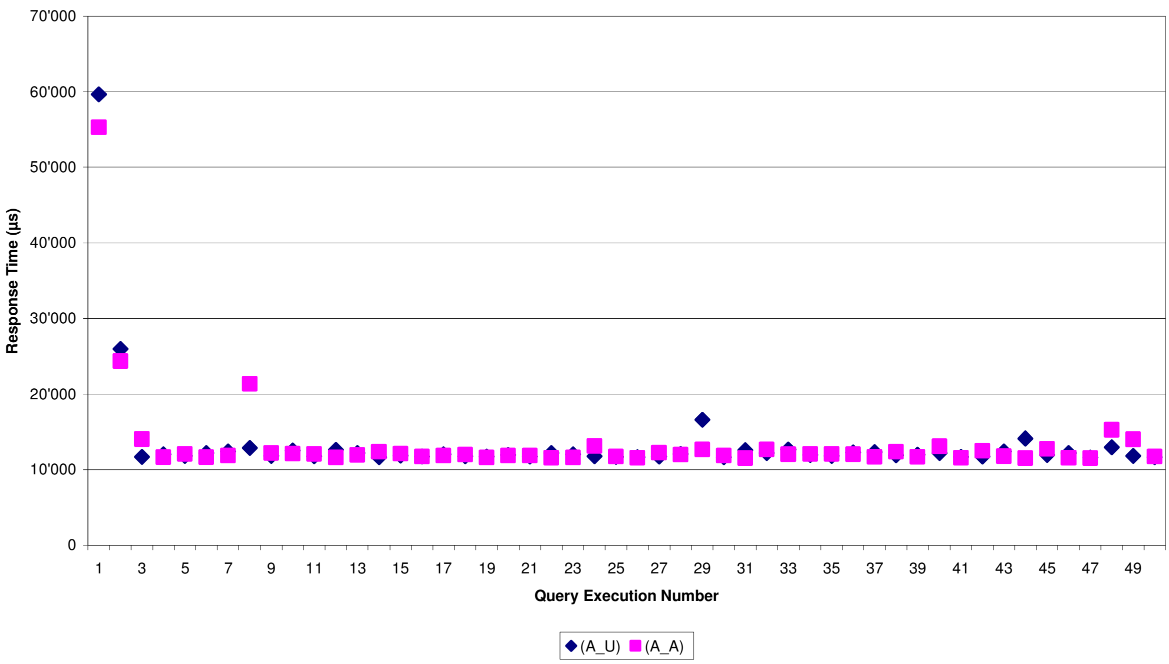
```
SELECT SEP.IDENT
  FROM TSMT_SERVICE_PROVIDER SEP
 WHERE EXISTS (SELECT *
                 FROM TSMT_DETAILED_CHARGING DDC
                WHERE DDC.PERIOD = TO_DATE ('01.06', 'MM.YY')
                  AND DDC.SERVICE_PROVIDER_ID = SEP.IDENT)
   AND SEP.HIERARCHY_CD LIKE 'PGIN%'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.00          0        700          0           7
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.00          0        700          0           7

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      7  HASH JOIN SEMI (cr=700 pr=0 pw=0 time=8100 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=87 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9070 us)
```

Median 12,020 µs (A_A):

```
SELECT IDENT
  FROM TSMT_SERVICE_PROVIDER
 WHERE IDENT IN (SELECT SERVICE_PROVIDER_ID
                   FROM TSMT_DETAILED_CHARGING
                  WHERE PERIOD = TO_DATE ('01.06', 'MM.YY'))
   AND HIERARCHY_CD LIKE 'PGIN%'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.00          0        700          0           7
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.00          0        700          0           7

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      7  HASH JOIN SEMI (cr=700 pr=0 pw=0 time=8425 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=121 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9384 us)
```



(Logarithmic scale)



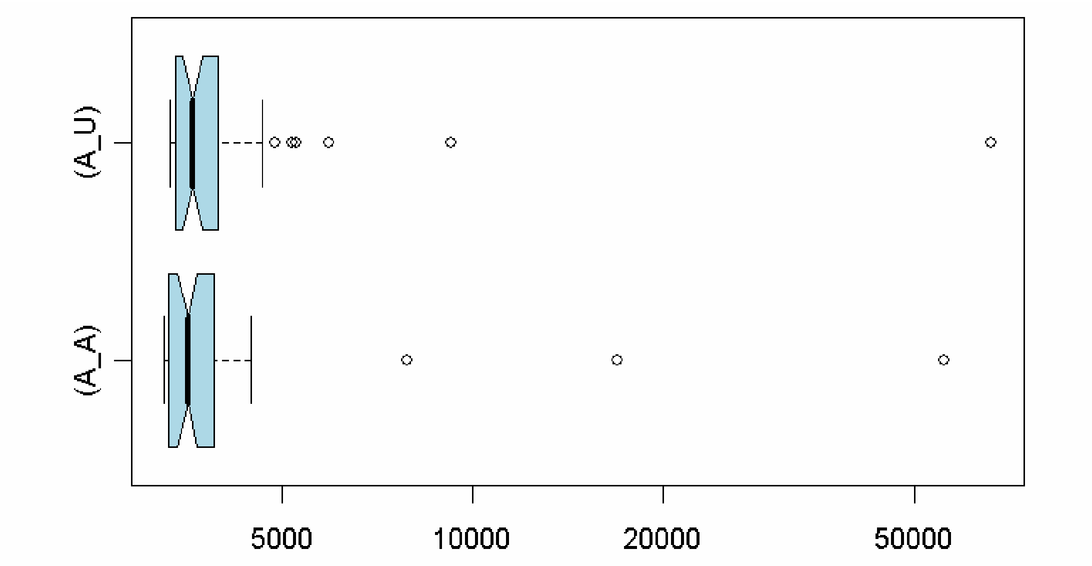
TQP_86306: If most of the rows are filtered by the inner query, then use `IN` – indexed column

Median 3,606 µs (A_U):

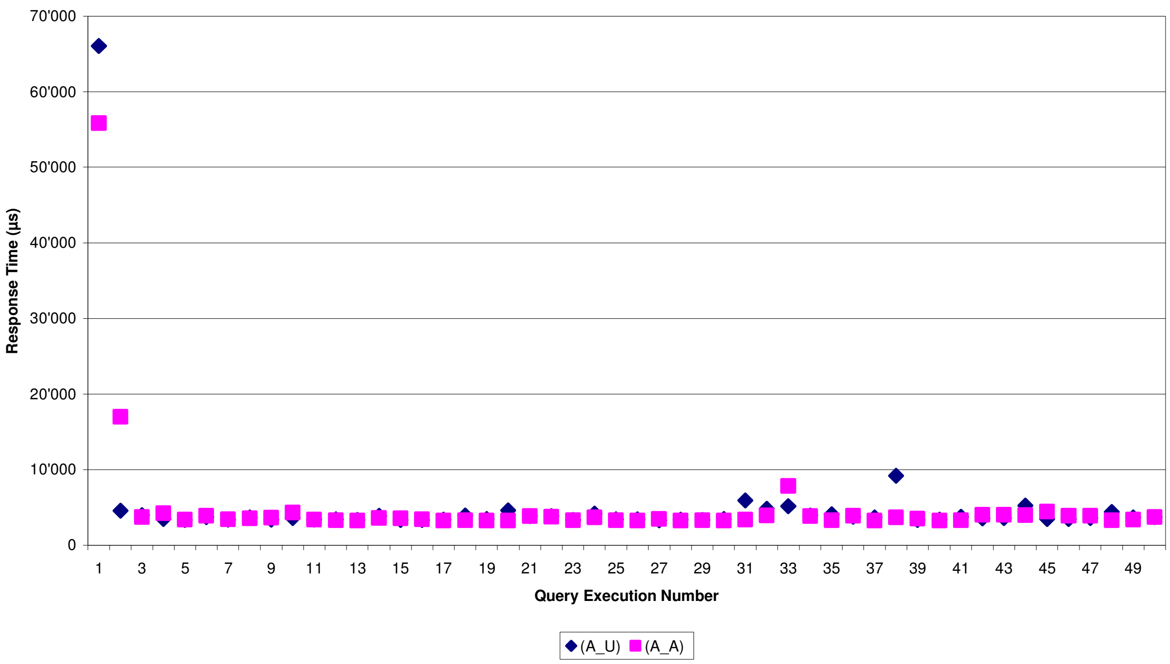
```
SELECT SER.IDENT
  FROM TSMT_SERVICE_RECEIVER SER
 WHERE EXISTS (SELECT *
                 FROM TSMT_DETAILED_CHARGING DDC
                WHERE DDC.PERIOD = TO_DATE ('01.06', 'MM.YY')
                  AND DDC.SERVICE_RECEIVER_ID = SER.IDENT)
   AND SER.HIERARCHY_CD LIKE 'PGI%'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0        144          0           4
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0        144          0           4

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      4  NESTED LOOPS SEMI (cr=144 pr=0 pw=0 time=1061 us)
     50   TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=197 us)
      4   TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=128 pr=0 pw=0 
time=1212 us)
    743    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=55 pr=0 pw=0 time=1047 
us)(object id 89036)
```

Median 3,538 µs (A_A):

```
SELECT IDENT
  FROM TSMT_SERVICE_RECEIVER
 WHERE IDENT IN (SELECT SERVICE_RECEIVER_ID
                   FROM TSMT_DETAILED_CHARGING
                  WHERE PERIOD = TO_DATE ('01.06', 'MM.YY'))
   AND HIERARCHY_CD LIKE 'PGI%'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0        144          0           4
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0        144          0           4

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      4  NESTED LOOPS SEMI (cr=144 pr=0 pw=0 time=1160 us)
     50   TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=223 us)
      4   TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=128 pr=0 pw=0 
time=1240 us)
    743    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=55 pr=0 pw=0 time=1063 
us)(object id 89036)
```



(Logarithmic scale)



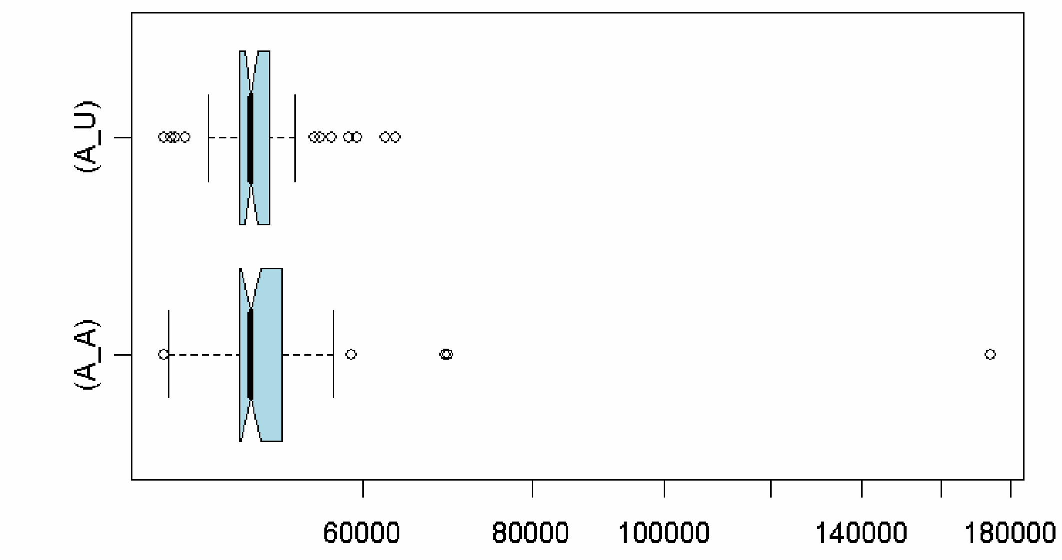
TQP_86307: If the outer query is of format "WHERE NOT …", then use NOT EXISTS

Median 49,660 µs (A_U):

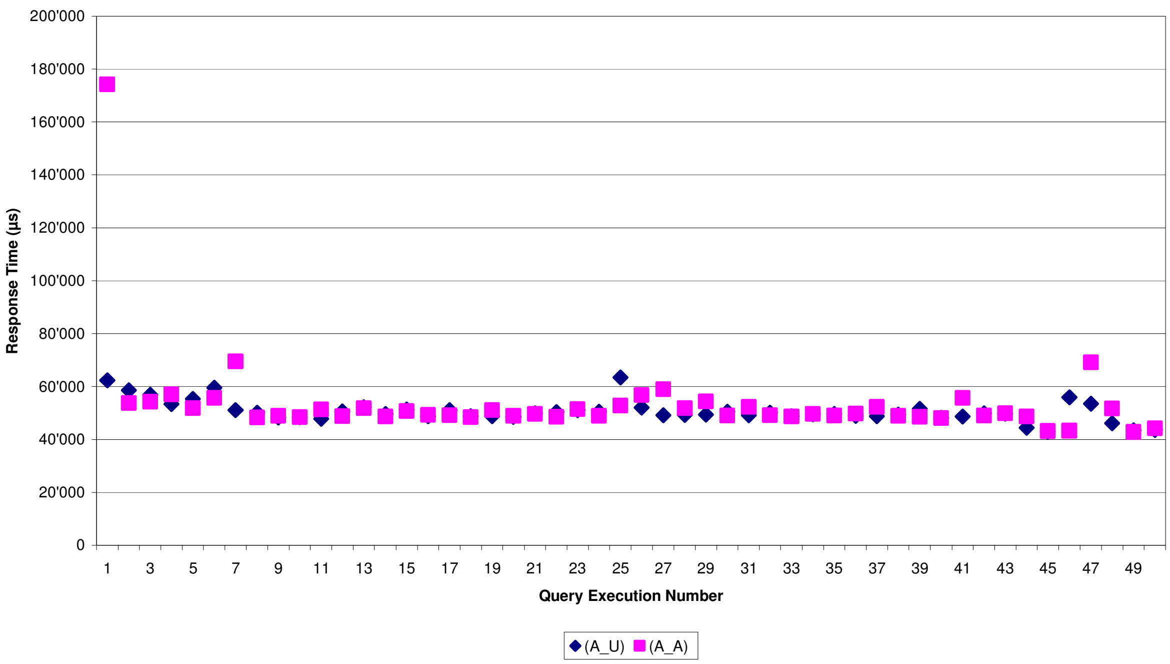
```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE DDC.PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND NOT EXISTS (SELECT *
                     FROM TSMT_SERVICE_PROVIDER SEP
                    WHERE SEP.HIERARCHY_CD LIKE 'PGIN%'
                      AND SEP.IDENT = DDC.SERVICE_PROVIDER_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      120      0.01       0.01          0        815          0        1195
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      122      0.01       0.01          0        815          0        1195

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1195  HASH JOIN RIGHT ANTI (cr=815 pr=0 pw=0 time=10852 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=116 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=808 pr=0 pw=0 time=9465 us)
```

Median 49,590 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND SERVICE_PROVIDER_ID NOT IN (SELECT IDENT
                                     FROM TSMT_SERVICE_PROVIDER
                                    WHERE HIERARCHY_CD LIKE 'PGIN%')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      120      0.01       0.01          0        815          0        1195
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      122      0.01       0.01          0        815          0        1195

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1195  HASH JOIN RIGHT ANTI (cr=815 pr=0 pw=0 time=13119 us)
     34   TABLE ACCESS FULL TSMT_SERVICE_PROVIDER (cr=7 pr=0 pw=0 time=143 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=808 pr=0 pw=0 time=9245 us)
```



(Logarithmic scale)



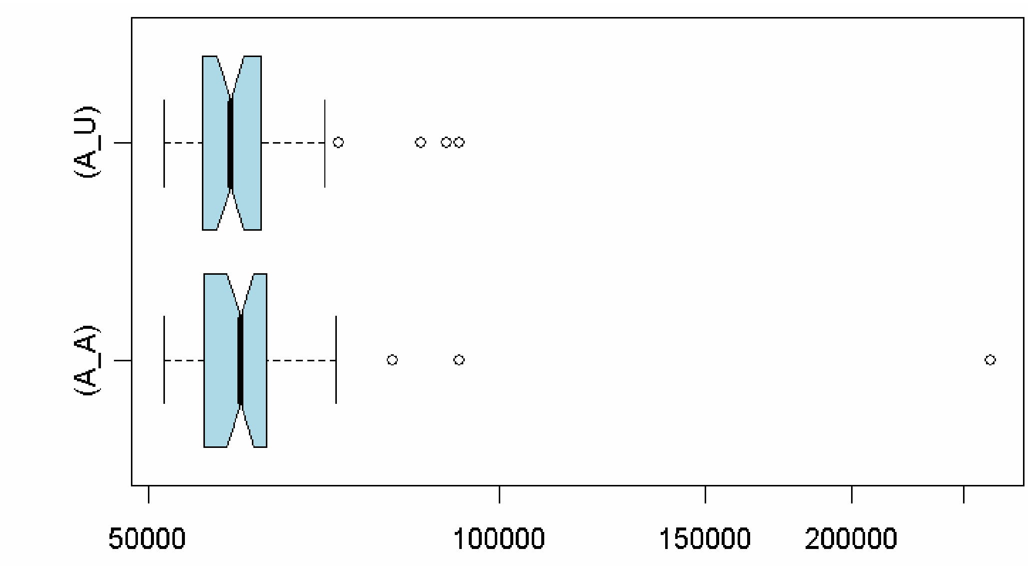
TQP_86308: If the outer query is of format "`WHERE NOT …`", then use `NOT EXISTS` – indexed column

Median 60,820 µs (A_U):

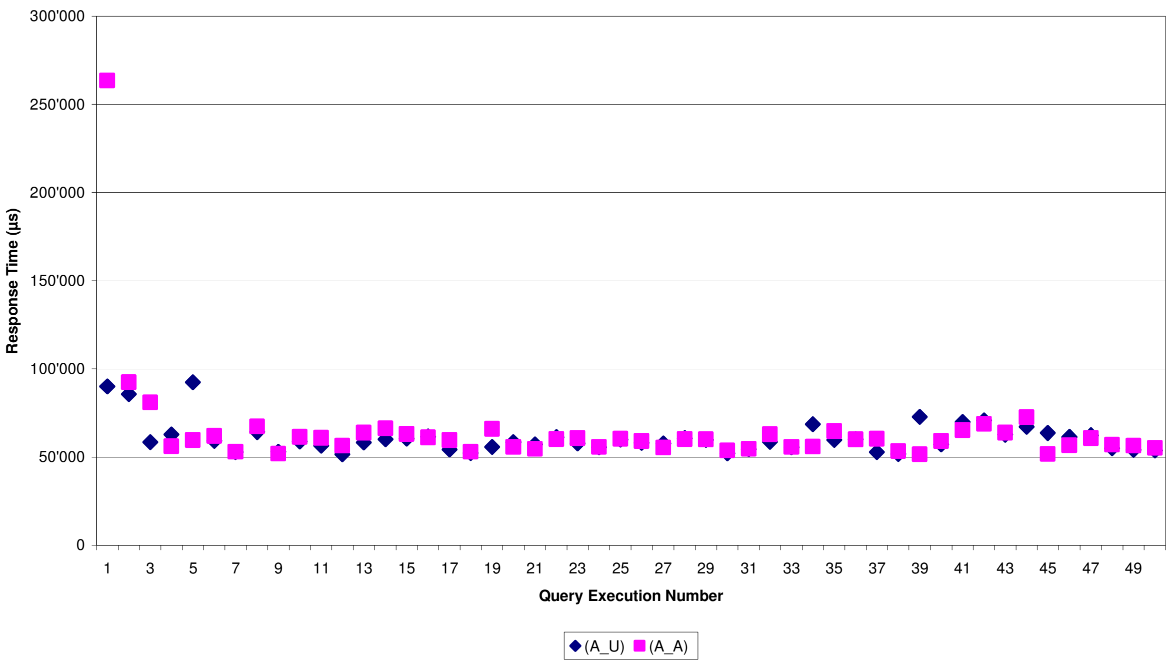
```
SELECT DDC.IDENT
  FROM TSMT_DETAILED_CHARGING DDC
 WHERE DDC.PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND NOT EXISTS (SELECT *
                      FROM TSMT_SERVICE_RECEIVER SER
                     WHERE SER.HIERARCHY_CD LIKE 'PGI%'
                       AND SER.IDENT = DDC.SERVICE_RECEIVER_ID)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      153      0.03       0.01          0        856          0        1521
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      155      0.03       0.01          0        856          0        1521

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1521  HASH JOIN RIGHT ANTI (cr=856 pr=0 pw=0 time=11774 us)
     50   TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=170 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=840 pr=0 pw=0 time=11157 us)
```

Median 64,440 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD = TO_DATE ('01.06', 'MM.YY')
   AND SERVICE_RECEIVER_ID NOT IN (SELECT IDENT
                                     FROM TSMT_SERVICE_RECEIVER
                                    WHERE HIERARCHY_CD LIKE 'PGI%')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      153      0.03       0.01          0        856          0        1521
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      155      0.03       0.01          0        856          0        1521

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1521  HASH JOIN RIGHT ANTI (cr=856 pr=0 pw=0 time=16054 us)
     50   TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=169 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=840 pr=0 pw=0 time=12392 us)
```



(Logarithmic scale)



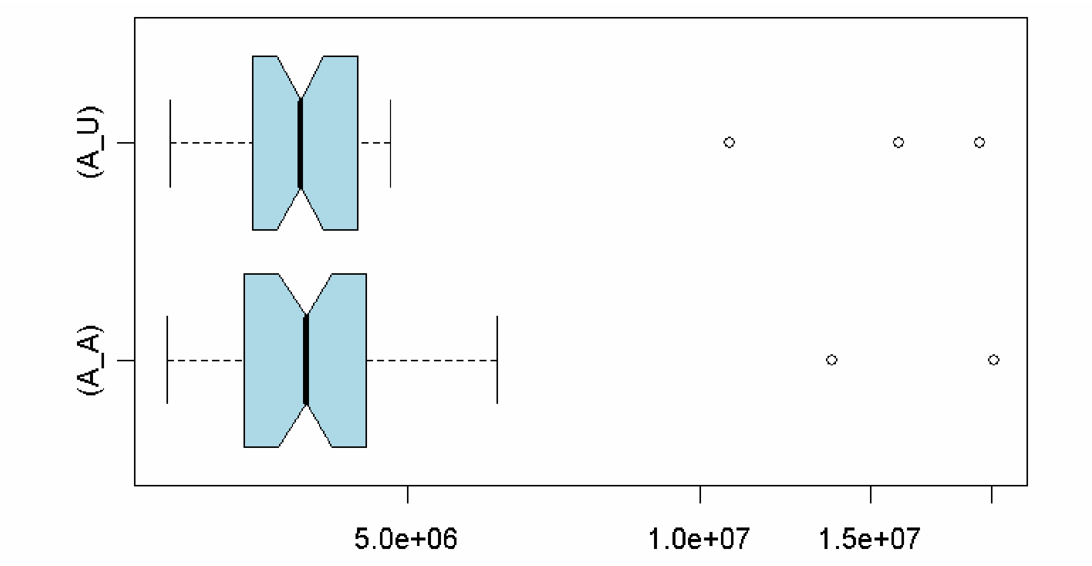
TQP_86309: Order the list of values of an `IN` predicate by frequency of probable usage

Median 4,579,000 µs (A_U):

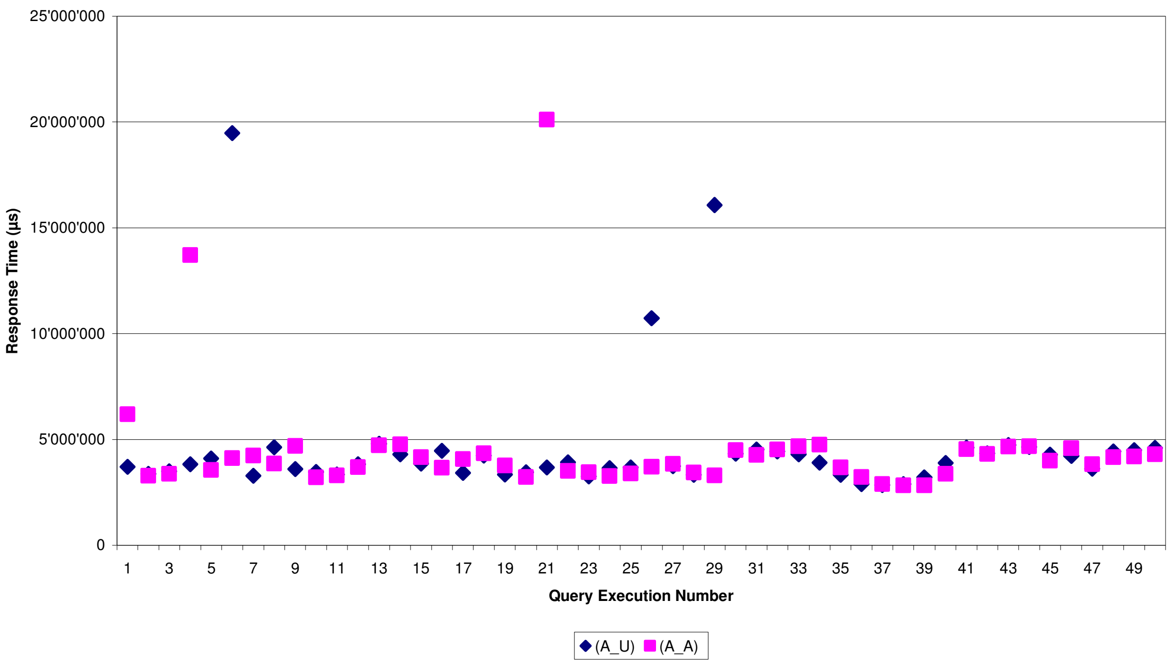
```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                 FROM TSMT_DETAILED_CHARGING)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.31       0.34       2389      12656          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.31       0.34       2389      12656          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN  (cr=12656 pr=2389 pw=0 time=434870 us)
     26   SORT UNIQUE (cr=693 pr=631 pw=0 time=25969 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=631 pw=0 time=33518 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1758 pw=0 time=205437
us)
```

Median 4,459,000 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                 FROM TSMT_DETAILED_CHARGING_86309)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.28       0.32       1733      11970          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.28       0.32       1733      11970          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN RIGHT SEMI (cr=11970 pr=1733 pw=0 time=408857 us)
     26   TABLE ACCESS FULL TSMT_DETAILED_CHARGING_86309 (cr=7 pr=0 pw=0 time=87 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1733 pw=0 time=205448
us)
```



(Logarithmic scale)



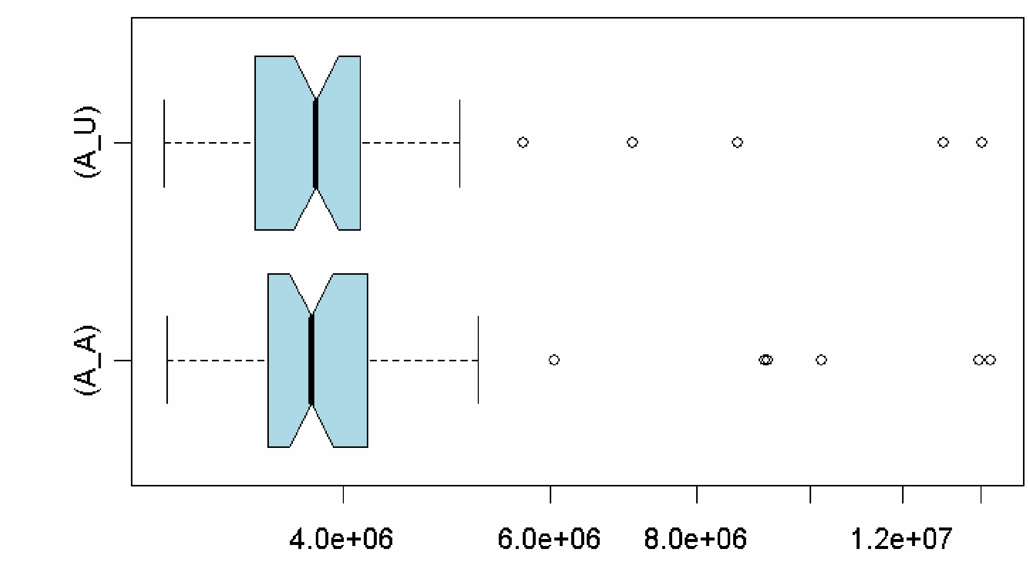
TQP_86310: Try to avoid duplicates in the list of values of an IN predicate – applying DISTINCT

Median 3,794,000 µs (A_U):

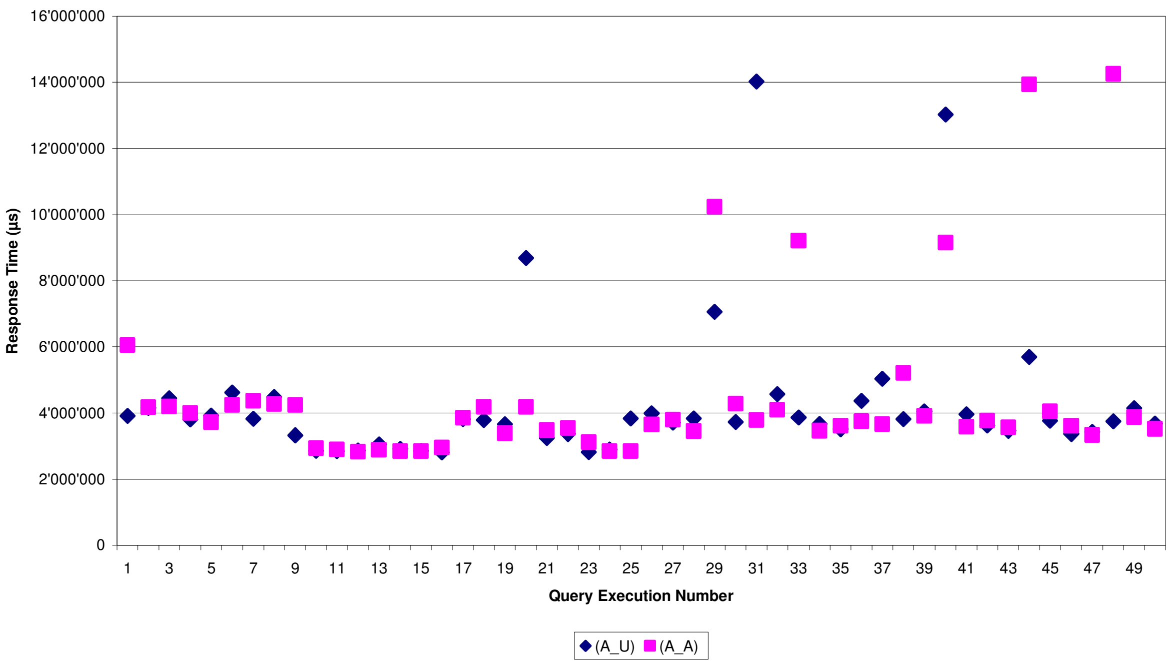
```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                  FROM TSMT_DETAILED_CHARGING)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.32       0.31       2393      12656          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.32       0.31       2393      12656          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN  (cr=12656 pr=2393 pw=0 time=339476 us)
     26   SORT UNIQUE (cr=693 pr=632 pw=0 time=32321 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=632 pw=0 time=33451 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1761 pw=0 time=103105
us)
```

Median 3,756,000 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT DISTINCT SERVICE_PROVIDER_ID
                                  FROM TSMT_DETAILED_CHARGING)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.39       0.30       2394      12656          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.39       0.30       2394      12656          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN  (cr=12656 pr=2394 pw=0 time=333167 us)
     26   SORT UNIQUE (cr=693 pr=632 pw=0 time=26354 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=632 pw=0 time=33464 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1762 pw=0 time=102840
us)
```



(Logarithmic scale)



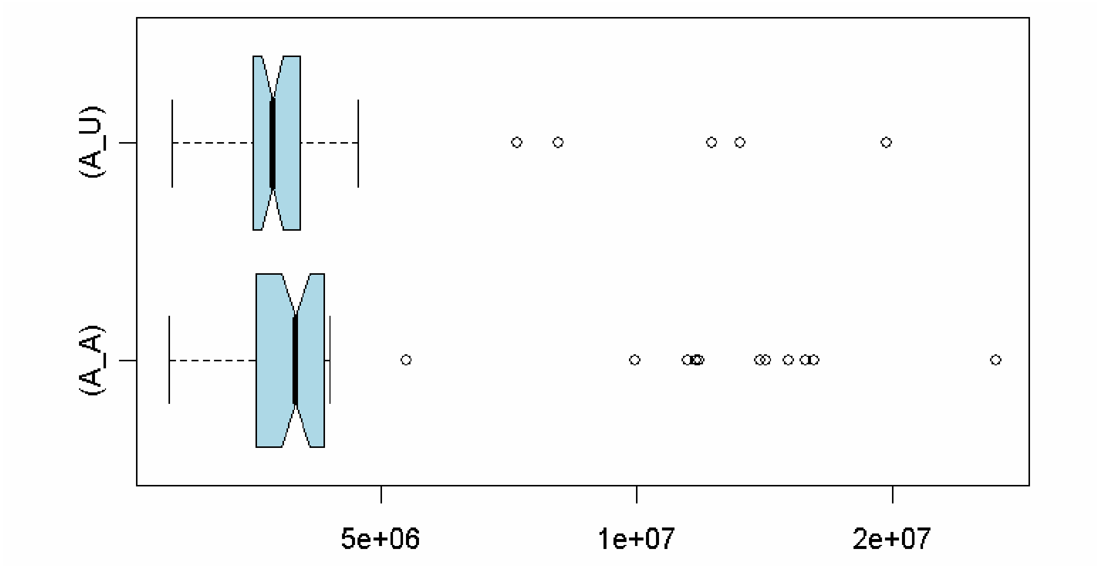
TQP_86311: Try to avoid duplicates in the list of values of an `IN` predicate

Median 3,546,000 µs (A_U):

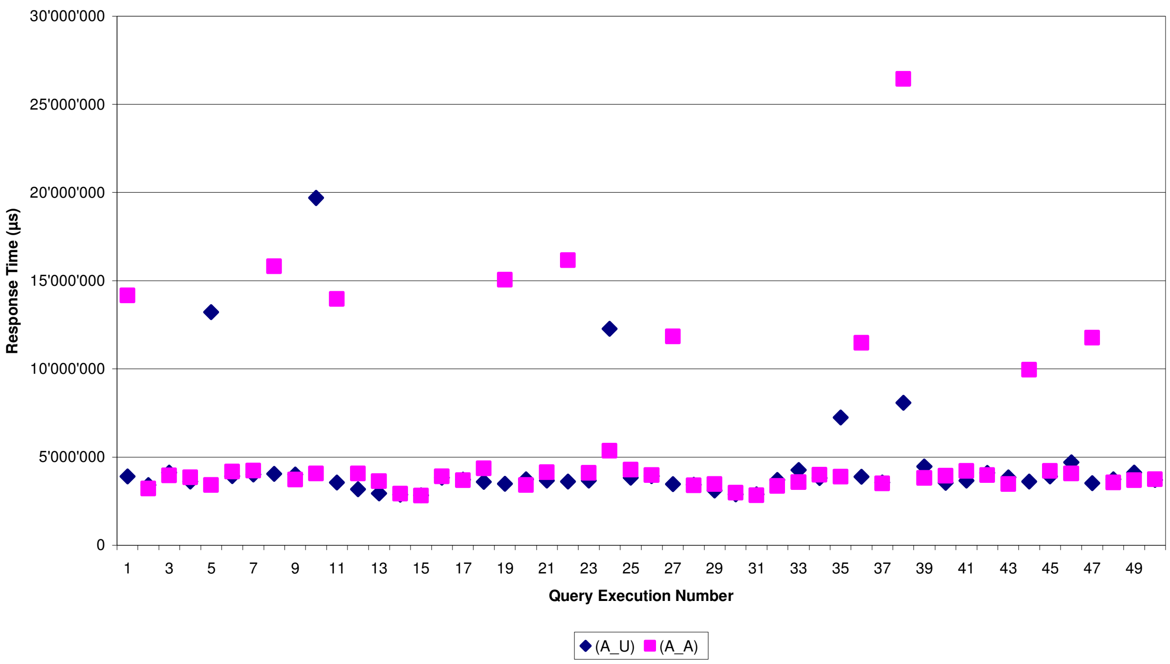
```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                 FROM TSMT_DETAILED_CHARGING)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.34       0.27       2392      12656          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.34       0.27       2392      12656          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN  (cr=12656 pr=2392 pw=0 time=332498 us)
     26   SORT UNIQUE (cr=693 pr=631 pw=0 time=25674 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=631 pw=0 time=33492 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1761 pw=0 time=102847
us)
```

Median 3,571,000 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                 FROM TSMT_DETAILED_CHARGING_86311)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.40       0.26       2447      12652          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.40       0.26       2447      12652          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN RIGHT SEMI (cr=12652 pr=2447 pw=0 time=316104 us)
     26   TABLE ACCESS FULL TSMT_DETAILED_CHARGING_86311 (cr=689 pr=684 pw=0 time=419
us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1763 pw=0 time=102857
us)
```



(Logarithmic scale)



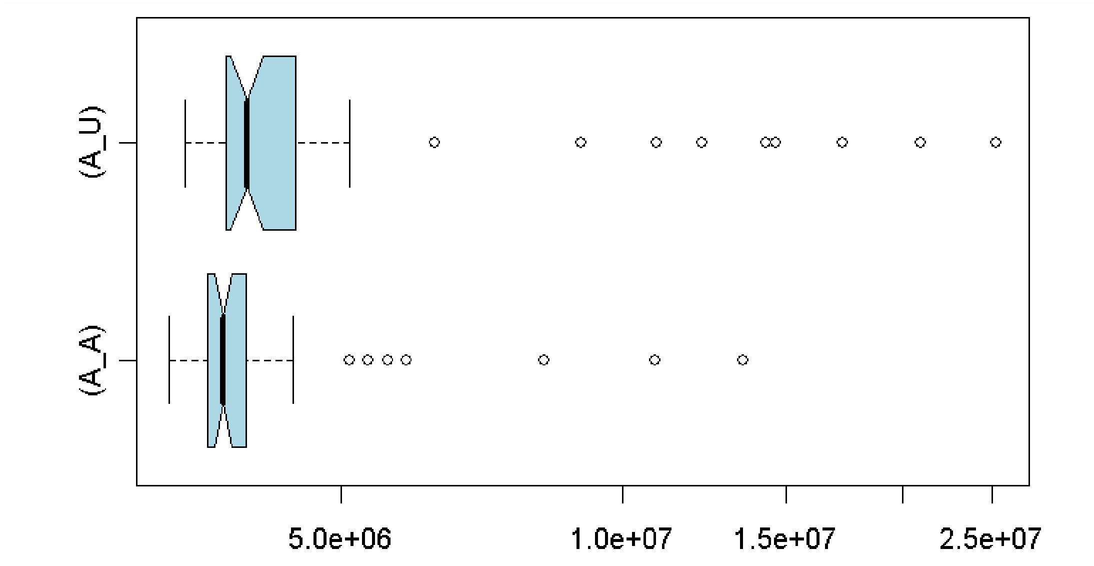
TQP_86312: Order the list of values of an `IN` predicate by frequency of probable usage versus try to avoid duplicates in the list of values of an `IN` predicate

Median 3,957,000 µs (A_U):

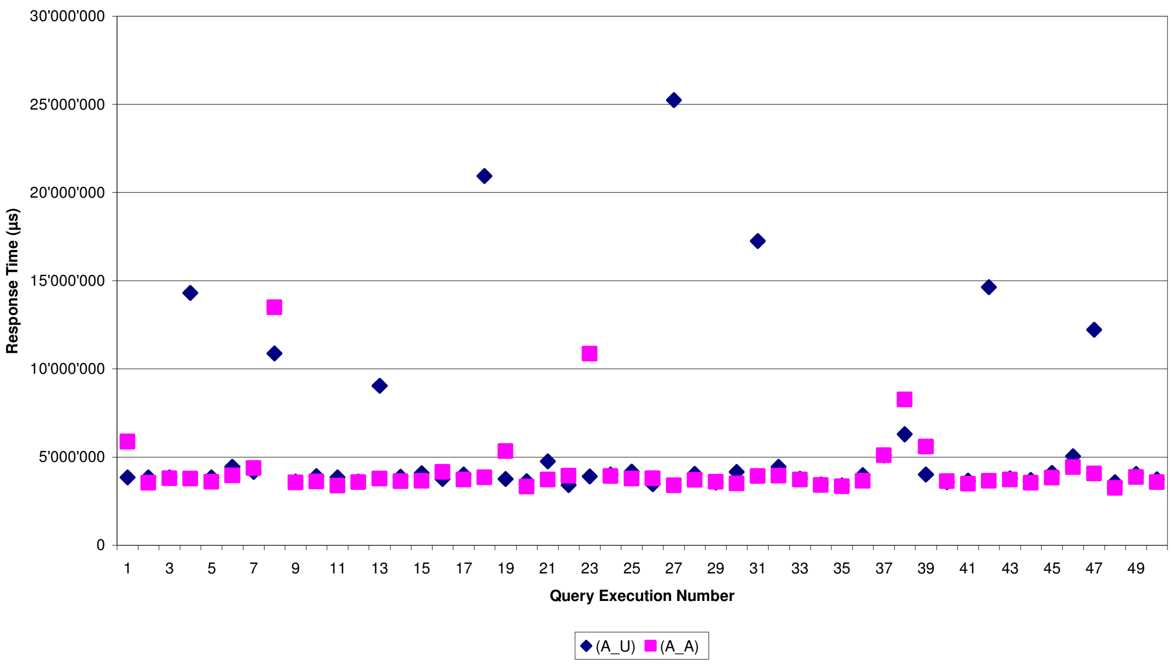
```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                 FROM TSMT_DETAILED_CHARGING_86309)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.35       0.40       1750      11970          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.35       0.40       1750      11970          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN RIGHT SEMI (cr=11970 pr=1750 pw=0 time=510836 us)
     26   TABLE ACCESS FULL TSMT_DETAILED_CHARGING_86309 (cr=7 pr=0 pw=0 time=97 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1750 pw=0 time=205452
us)
```

Median 3,730,000 µs (A_A):

```
SELECT IDENT
  FROM TSMT_DETAILED_METERING
 WHERE SERVICE_PROVIDER_ID IN (SELECT SERVICE_PROVIDER_ID
                                 FROM TSMT_DETAILED_CHARGING_86311)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10197      0.39       0.40       2442      12652          0      101962
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10199      0.39       0.40       2442      12652          0      101962

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 101962  HASH JOIN RIGHT SEMI (cr=12652 pr=2442 pw=0 time=519893 us)
     26   TABLE ACCESS FULL TSMT_DETAILED_CHARGING_86311 (cr=689 pr=684 pw=0 time=416
us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=11963 pr=1758 pw=0 time=205421
us)
```



(Logarithmic scale)



GROUPING SETS beats UNION

TQP_80501: The use of a `GROUPING SETS` clause instead of the application of a `UNION` should result in a significant performance improvement - `CUBE`

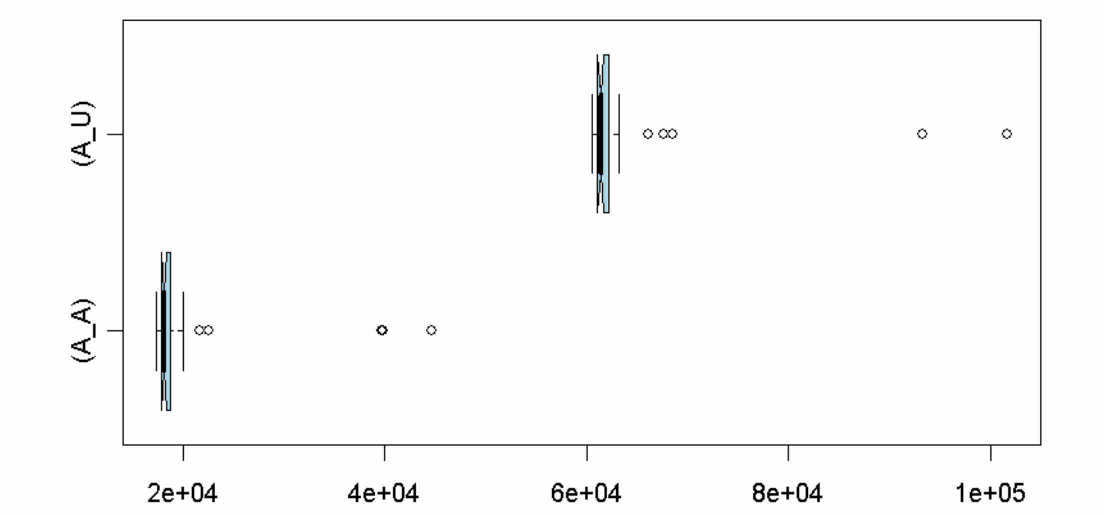
Median 61,380 µs (A_U):

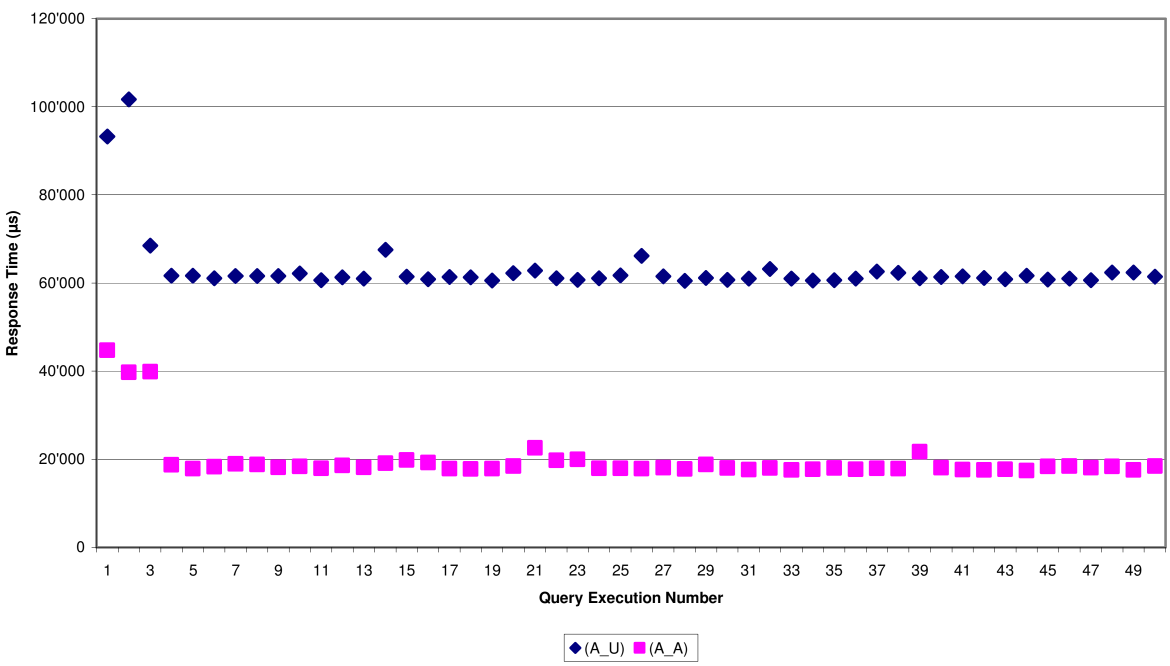
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD, SIGN
UNION ALL
SELECT PERIOD, NULL, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD
UNION ALL
SELECT NULL, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY SIGN
UNION ALL
SELECT NULL, NULL, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.06       0.05          0       2772          0          12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.06       0.05          0       2772          0          12

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  UNION-ALL  (cr=2772 pr=0 pw=0 time=16823 us)
      6   HASH GROUP BY (cr=693 pr=0 pw=0 time=16689 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=14099 us)
      3   HASH GROUP BY (cr=693 pr=0 pw=0 time=14364 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13108 us)
      2   HASH GROUP BY (cr=693 pr=0 pw=0 time=14085 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13215 us)
      1   SORT AGGREGATE (cr=693 pr=0 pw=0 time=11936 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13189 us)
```

Median 18,120 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY CUBE (PERIOD, SIGN)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
```

```
Execute       1      0.00        0.00          0          0          0            0
Fetch         2      0.01        0.01          0        693          0           12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total         4      0.01        0.01          0        693          0           12

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  SORT GROUP BY (cr=693 pr=0 pw=0 time=14824 us)
     24   GENERATE CUBE (cr=693 pr=0 pw=0 time=14757 us)
      6    SORT GROUP BY (cr=693 pr=0 pw=0 time=14736 us)
   4835     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13450 us)
```





TQP_80502: The use of a GROUPING SETS clause instead of the application of a UNION should result in a significant performance improvement – GROUPING SETS

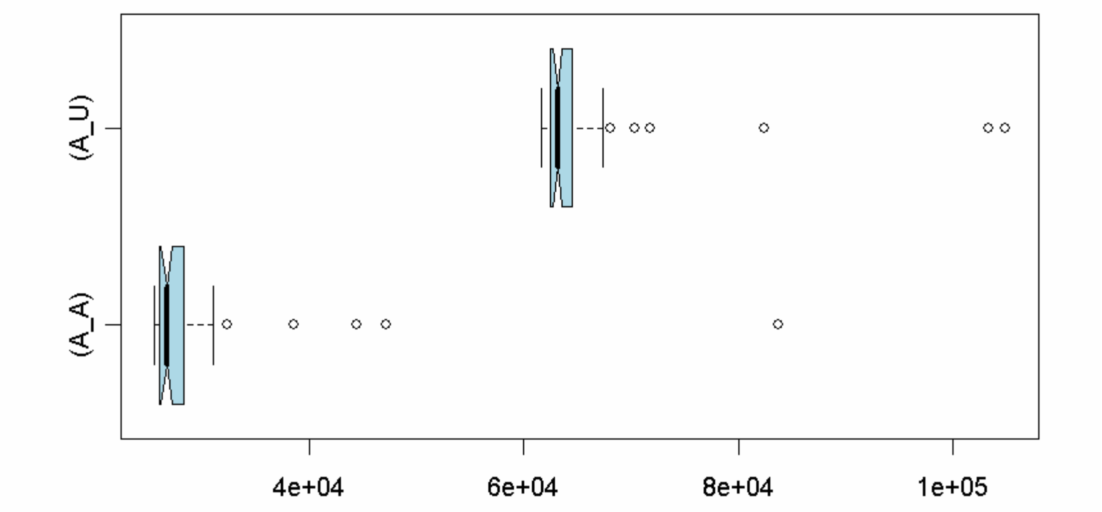
Median 63,130 µs (A_U):

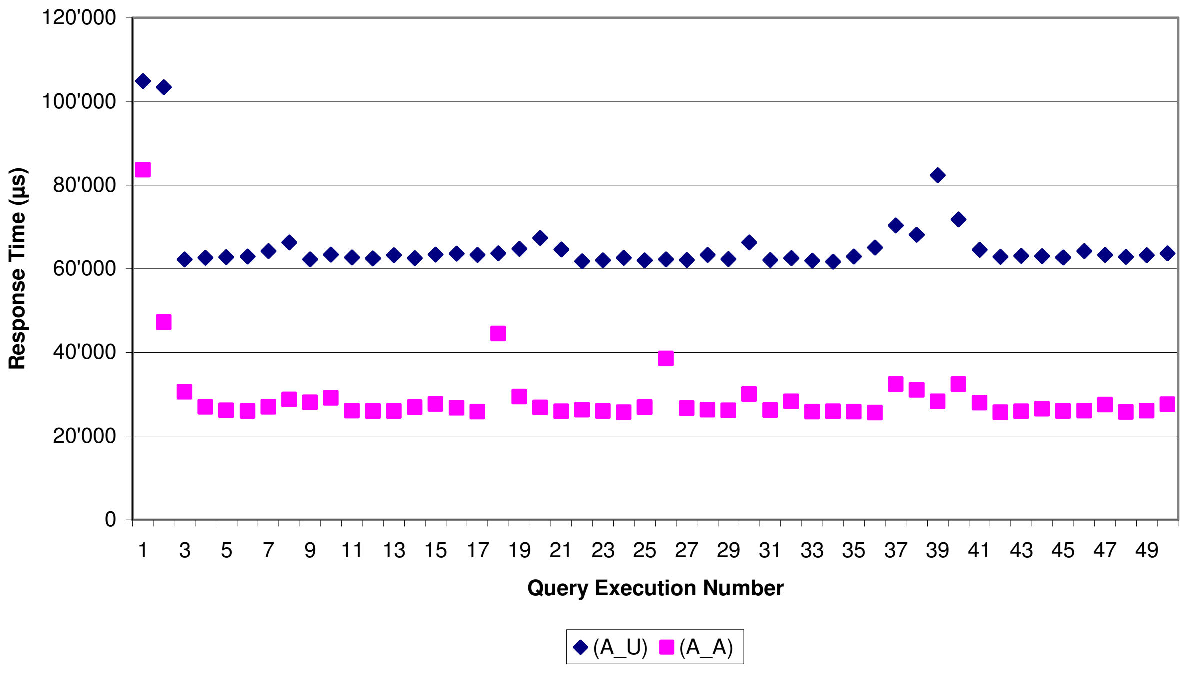
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD, SIGN
UNION ALL
SELECT PERIOD, NULL, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD
UNION ALL
SELECT NULL, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY SIGN
UNION ALL
SELECT NULL, NULL, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.06       0.05          0       2772          0          12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.06       0.05          0       2772          0          12

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  UNION-ALL  (cr=2772 pr=0 pw=0 time=15318 us)
      6   HASH GROUP BY (cr=693 pr=0 pw=0 time=15220 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13448 us)
      3   HASH GROUP BY (cr=693 pr=0 pw=0 time=14318 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=8260 us)
      2   HASH GROUP BY (cr=693 pr=0 pw=0 time=14276 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13224 us)
      1   SORT AGGREGATE (cr=693 pr=0 pw=0 time=12124 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=8490 us)
```

Median 26,710 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY GROUPING SETS ((PERIOD, SIGN), PERIOD, SIGN, ())

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.03       0.02          1        698         20           0
Fetch        2      0.00       0.00          2         10          1          12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
```

```
total           4      0.03        0.02            3         708          21            12

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  TEMP TABLE TRANSFORMATION  (cr=708 pr=3 pw=3 time=20527 us)
      2   MULTI-TABLE INSERT  (cr=693 pr=0 pw=2 time=18142 us)
      9    SORT GROUP BY ROLLUP (cr=693 pr=0 pw=0 time=14732 us)
   4835     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13728 us)
      1   LOAD AS SELECT  (cr=5 pr=1 pw=1 time=1948 us)
      3    SORT GROUP BY ROLLUP (cr=5 pr=1 pw=0 time=151 us)
      6     TABLE ACCESS FULL SYS_TEMP_0FD9D660C_69BE73 (cr=5 pr=1 pw=0 time=97 us)
     12   VIEW  (cr=10 pr=2 pw=0 time=135 us)
     12    VIEW  (cr=10 pr=2 pw=0 time=109 us)
     12     UNION-ALL  (cr=10 pr=2 pw=0 time=95 us)
      6      TABLE ACCESS FULL SYS_TEMP_0FD9D660C_69BE73 (cr=3 pr=0 pw=0 time=48 us)
      6      TABLE ACCESS FULL SYS_TEMP_0FD9D660D_69BE73 (cr=7 pr=2 pw=0 time=111 us)
```





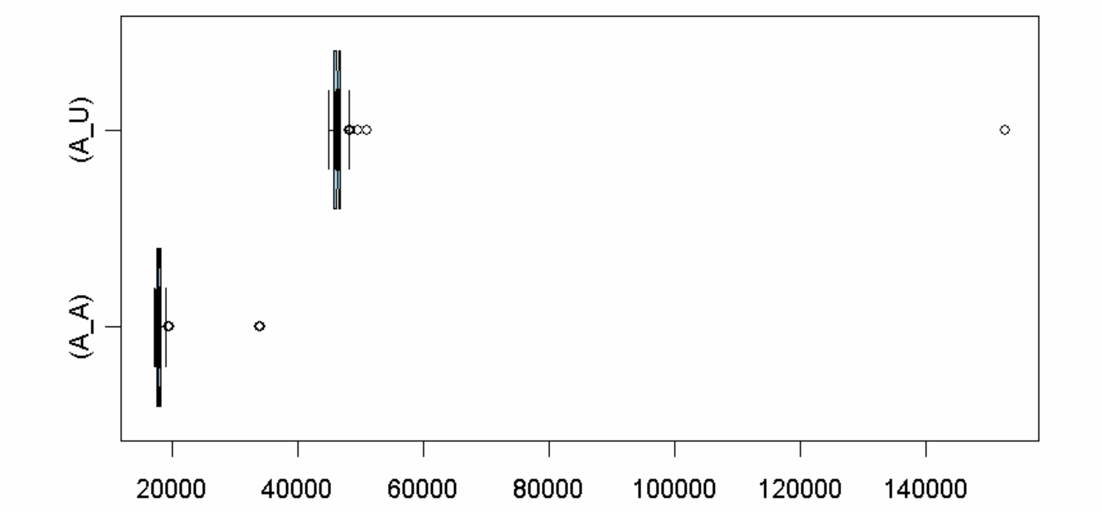
TQP_80503: The use of a GROUPING SETS clause instead of the application of a UNION should result in a significant performance improvement - ROLLUP

Median 46,250 µs (A_U):

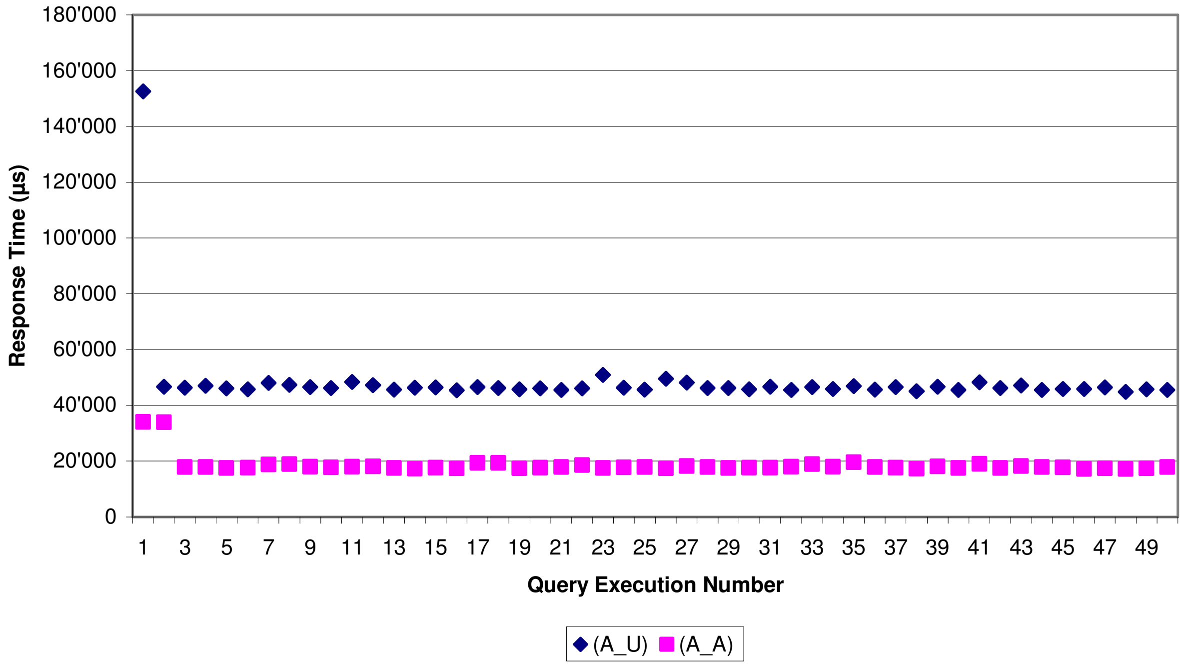
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD, SIGN
UNION ALL
SELECT PERIOD, NULL, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD
UNION ALL
SELECT NULL, NULL, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.03       0.04          0       2079          0          10
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.03       0.04          0       2079          0          10

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     10  UNION-ALL  (cr=2079 pr=0 pw=0 time=15391 us)
      6   HASH GROUP BY (cr=693 pr=0 pw=0 time=15321 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13455 us)
      3   HASH GROUP BY (cr=693 pr=0 pw=0 time=14260 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13227 us)
      1   SORT AGGREGATE (cr=693 pr=0 pw=0 time=11543 us)
   4835    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=8328 us)
```

Median 17,770 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY ROLLUP (PERIOD, SIGN)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.01          0        693          0          10
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.00       0.01          0        693          0          10

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     10  SORT GROUP BY ROLLUP (cr=693 pr=0 pw=0 time=13722 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13319 us)
```



(A_U)
(A_A)
20000
40000
60000
80000
100000
120000
140000



Response Time (µs)
180'000
160'000
140'000
120'000
100'000
80'000
60'000
40'000
20'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

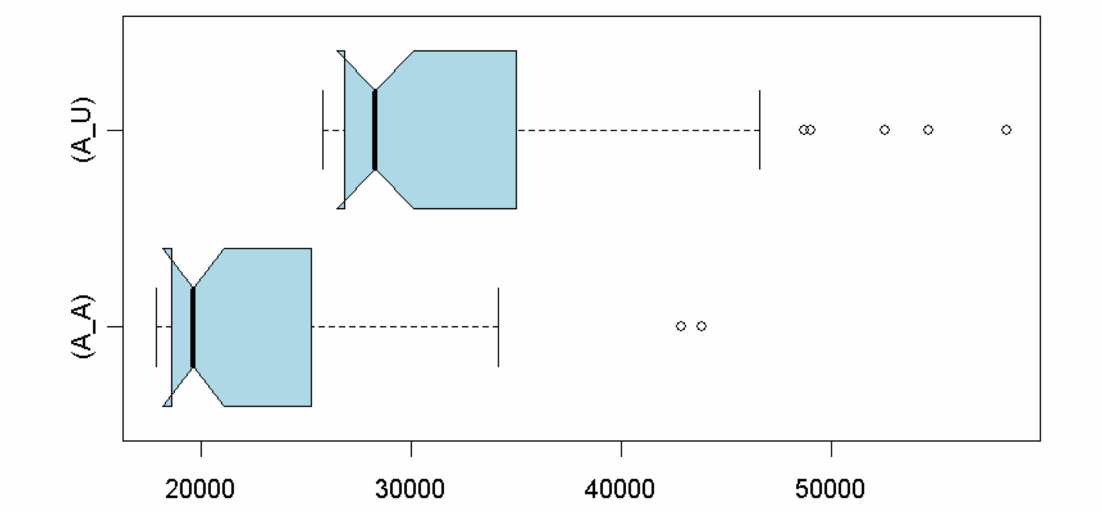
TQP_80511: `CUBE` and `ROLLUP` are shortcuts that can be used to simplify the application of certain variants of the `GROUPING SETS` clause - `CUBE`

Median 28,310 µs (A_U):

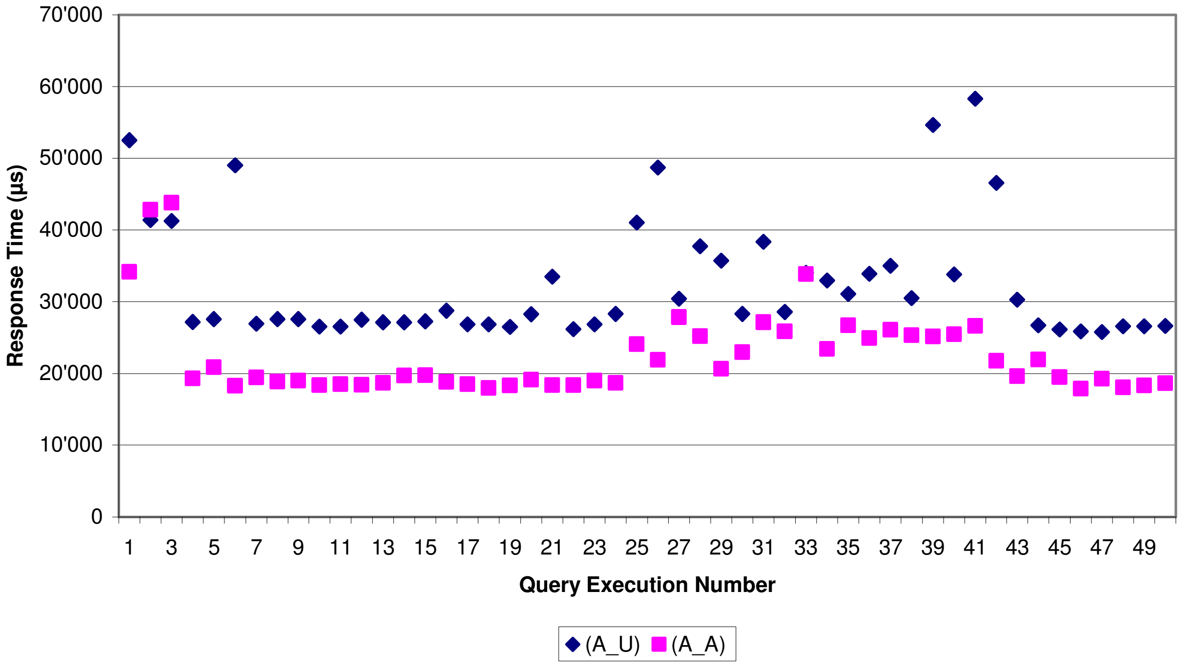
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY GROUPING SETS ((PERIOD, SIGN), PERIOD, SIGN, ())

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.01       0.02          1        698         20           0
Fetch        2      0.00       0.00          2         10          1          12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.02          3        708         21          12

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  TEMP TABLE TRANSFORMATION  (cr=708 pr=3 pw=3 time=20472 us)
      2   MULTI-TABLE INSERT  (cr=693 pr=0 pw=2 time=18500 us)
      9    SORT GROUP BY ROLLUP (cr=693 pr=0 pw=0 time=15115 us)
   4835     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13813 us)
      1   LOAD AS SELECT  (cr=5 pr=1 pw=1 time=1572 us)
      3    SORT GROUP BY ROLLUP (cr=5 pr=1 pw=0 time=135 us)
      6     TABLE ACCESS FULL SYS_TEMP_0FD9D660C_69BE73 (cr=5 pr=1 pw=0 time=81 us)
     12   VIEW  (cr=10 pr=2 pw=0 time=130 us)
     12    VIEW  (cr=10 pr=2 pw=0 time=104 us)
     12     UNION-ALL  (cr=10 pr=2 pw=0 time=89 us)
      6      TABLE ACCESS FULL SYS_TEMP_0FD9D660C_69BE73 (cr=3 pr=0 pw=0 time=43 us)
      6      TABLE ACCESS FULL SYS_TEMP_0FD9D660D_69BE73 (cr=7 pr=2 pw=0 time=112 us)
```

Median 19,660 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY CUBE (PERIOD, SIGN)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.01       0.01          0        693          0          12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.01          0        693          0          12

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  SORT GROUP BY (cr=693 pr=0 pw=0 time=14736 us)
     24   GENERATE CUBE (cr=693 pr=0 pw=0 time=14650 us)
      6    SORT GROUP BY (cr=693 pr=0 pw=0 time=14606 us)
   4835     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=14280 us)
```



(A_U)
(A_A)
20000
30000
40000
50000



Response Time (µs)
70'000
60'000
50'000
40'000
30'000
20'000
10'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

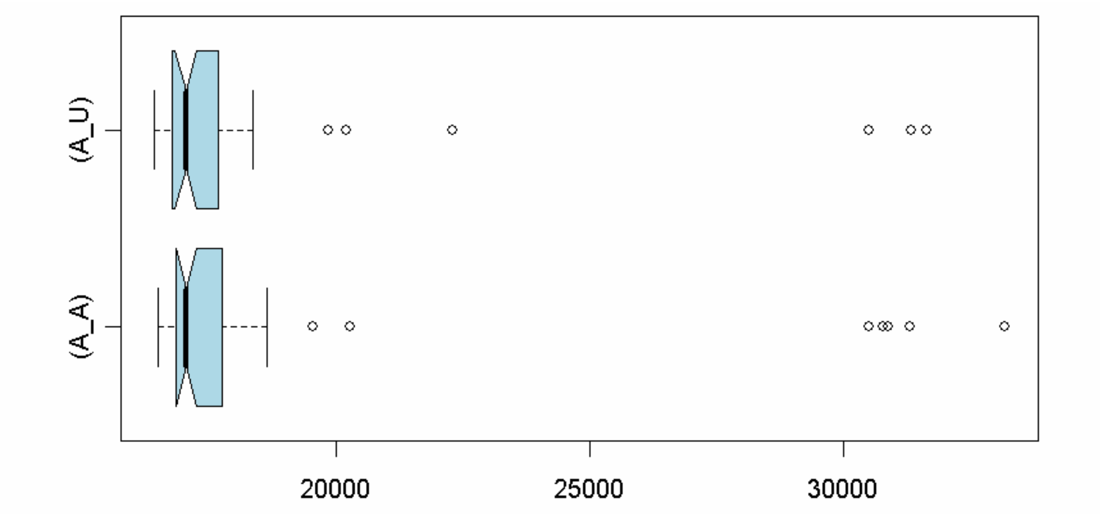
TQP_80512: CUBE and ROLLUP are shortcuts that can be used to simplify the application of certain variants of the GROUPING SETS clause - ROLLUP

Median 17,040 µs (A_U):

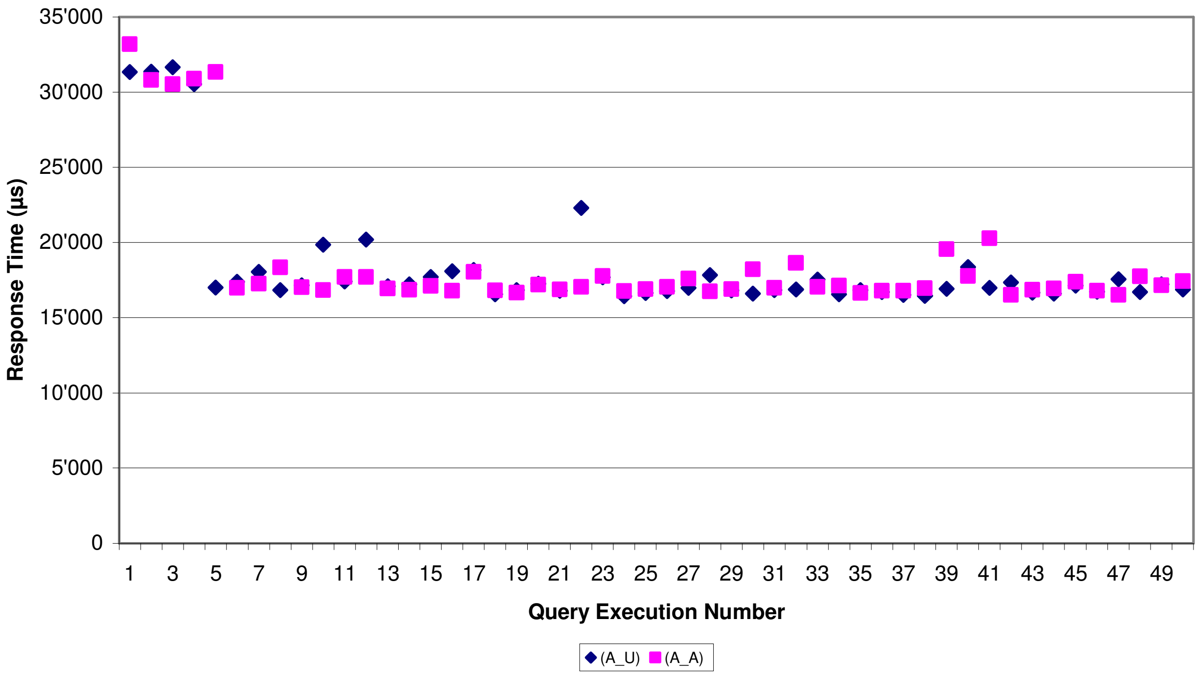
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY GROUPING SETS ((PERIOD, SIGN), PERIOD, ())

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.01       0.01          0        693          0          10
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.01          0        693          0          10

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     10  SORT GROUP BY ROLLUP (cr=693 pr=0 pw=0 time=14491 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13627 us)
```

Median 17,050 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY ROLLUP (PERIOD, SIGN)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.01       0.01          0        693          0          10
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.01          0        693          0          10

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     10  SORT GROUP BY ROLLUP (cr=693 pr=0 pw=0 time=13945 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13419 us)
```



(A_U)
(A_A)
20000
25000
30000



35'000
30'000
25'000
20'000
15'000
10'000
5'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U)
(A_A)

Index Exploitation

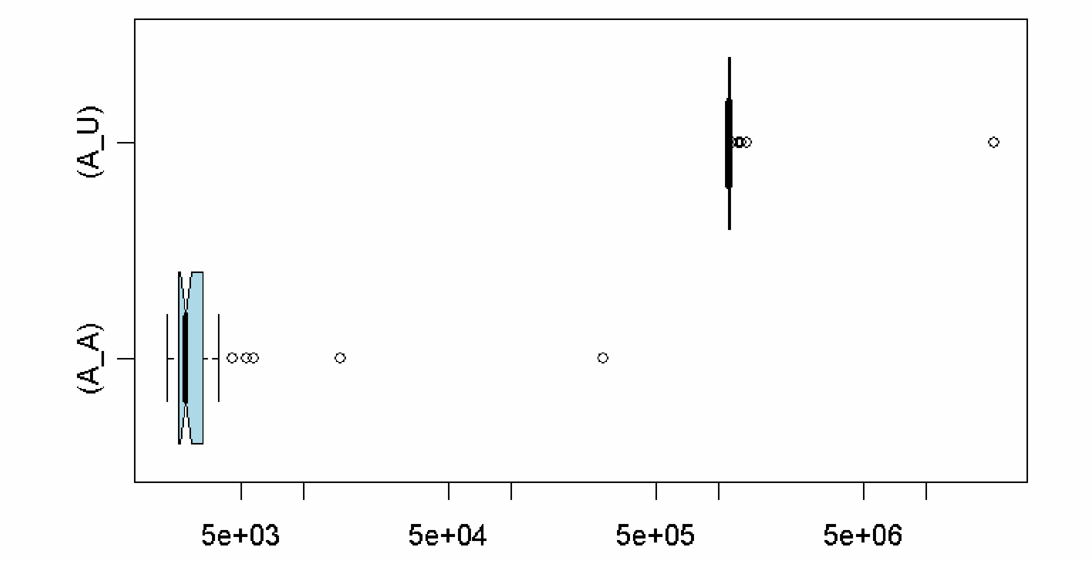
TQP_80401: Don't apply functions to indexed columns – result set cardinality 1 of 1,945,431 rows

Median 1,115,700 µs (A_U):

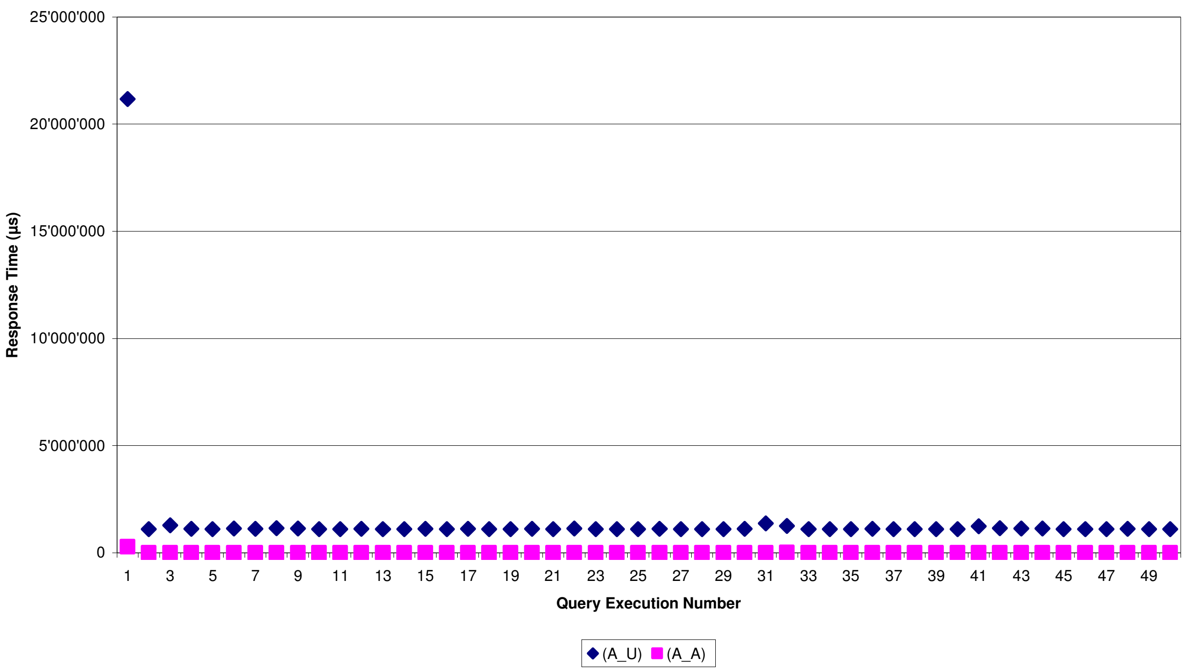
```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE TO_NUMBER (COMPANY_CD) = 2786

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      1.10       1.11      25155      25163          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      1.10       1.11      25155      25163          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=25163 pr=25155 pw=0 time=1110289
us)
```

Median 2,712 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD = '2786'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          4          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0          4          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID TSMT_MDB_ACCOUNT_RUD (cr=4 pr=0 pw=0 time=62 us)
      1   INDEX RANGE SCAN T_SMT_MAR_COMPANY_CD (cr=3 pr=0 pw=0 time=24 us)(object id
87336)
```



(Logarithmic scale)



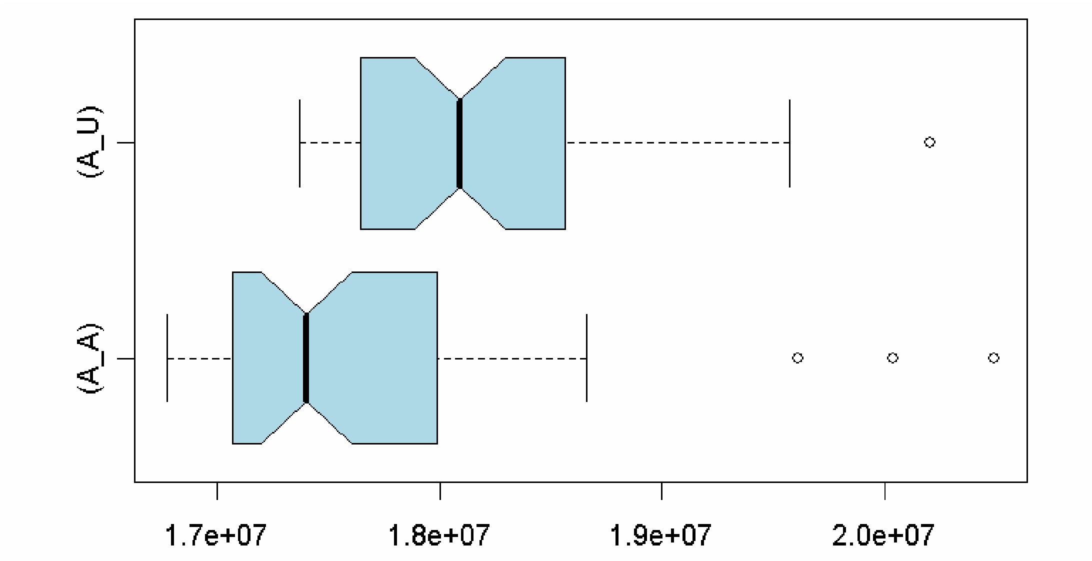
TQP_80402: Don't apply functions to indexed columns – result set cardinality 20% of 1,945,431 rows

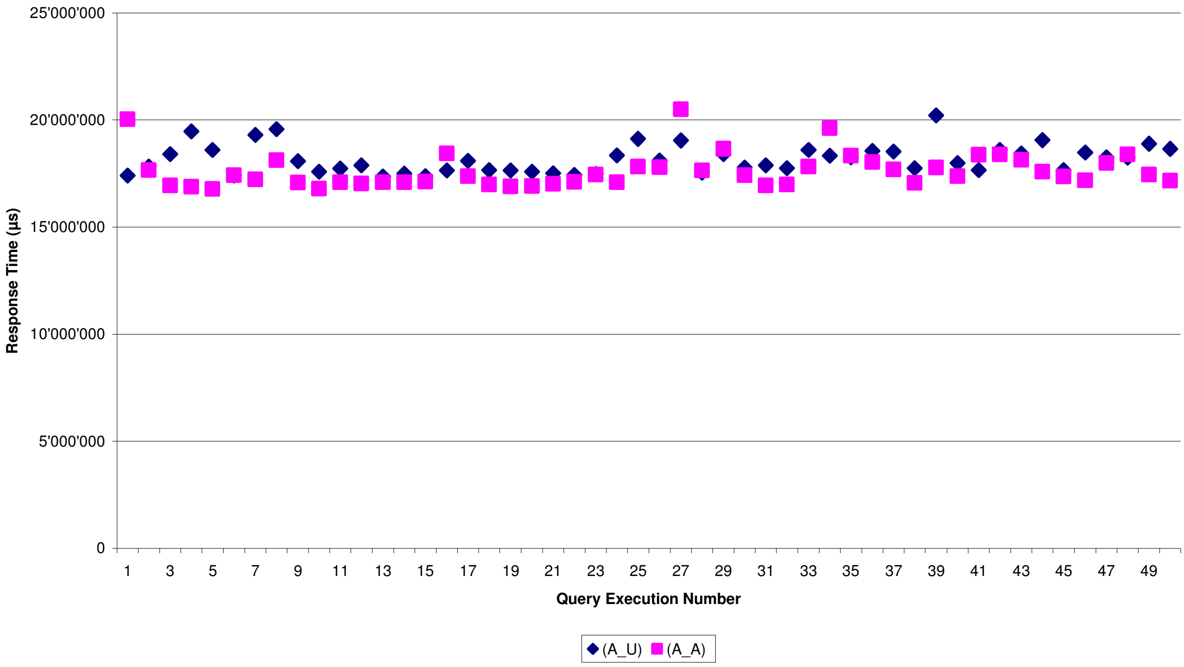
Median 18,090,000 µs (A_U):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE TO_NUMBER (COMPANY_CD) = 1201

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    39043      2.21       2.35      23081      63665          0      390428
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    39045      2.21       2.35      23081      63665          0      390428

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 390428  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=63665 pr=23081 pw=0 time=1562752
us)
```

Median 17,400,000 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD = '1201'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    39043      1.71       1.96       6777      85133          0      390428
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    39045      1.71       1.96       6777      85133          0      390428

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 390428  TABLE ACCESS BY INDEX ROWID TSMT_MDB_ACCOUNT_RUD (cr=85133 pr=6777 pw=0
time=2733771 us)
 390428   INDEX RANGE SCAN T_SMT_MAR_COMPANY_CD (cr=39828 pr=866 pw=0 time=784186
us)(object id 87336)
```


(A_U)
(A_A)
1.7e+07
1.8e+07
1.9e+07
2.0e+07


25'000'000
20'000'000
15'000'000
10'000'000
5'000'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

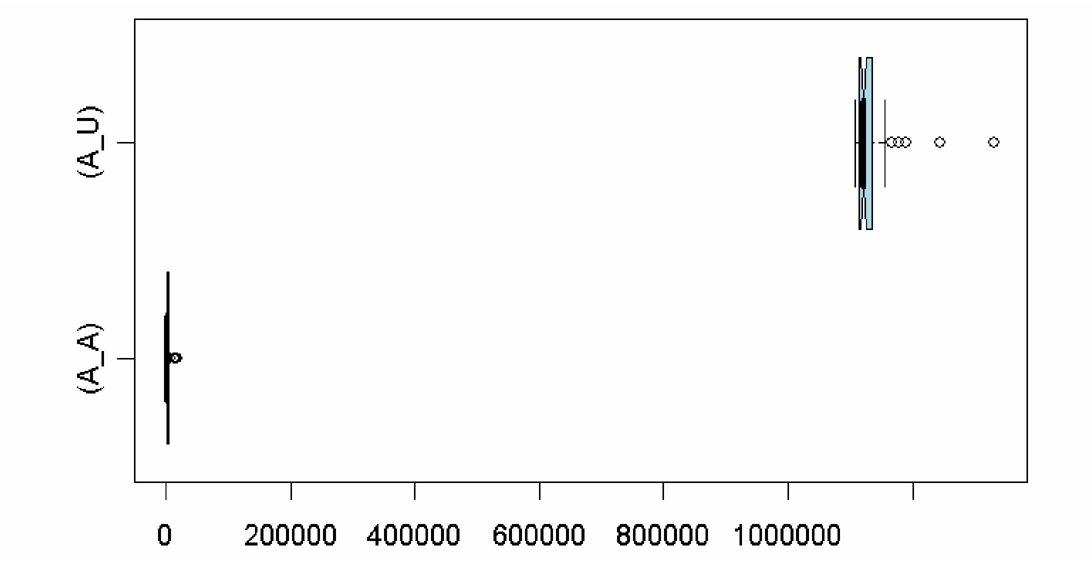
TQP_80411: Move function calls to the non-indexed components

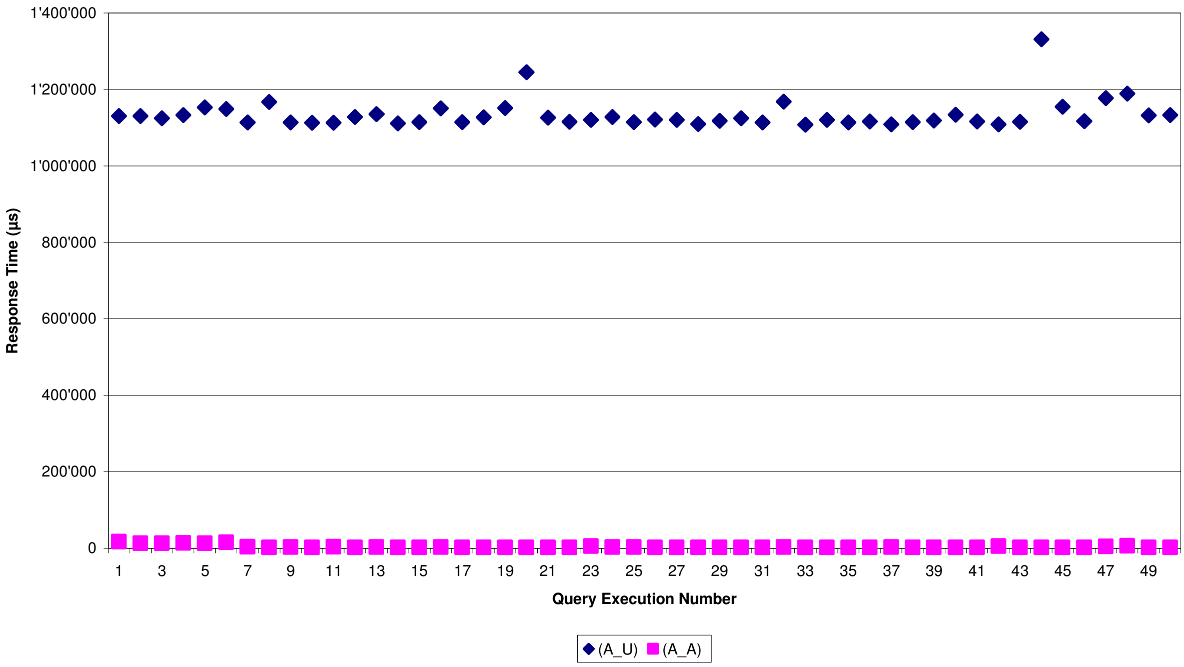
Median 1,114,000 µs (A_U):

```
SELECT *
FROM TSMT_MDB_ACCOUNT_RUD
WHERE TO_NUMBER (COMPANY_CD) = 2786

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      1.09       1.11      21226      25163          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      1.09       1.11      21226      25163          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=25163 pr=21226 pw=0 time=1115344
us)
```

Median 2,515 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD = TO_CHAR(2786)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          4          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0          4          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID TSMT_MDB_ACCOUNT_RUD (cr=4 pr=0 pw=0 time=76 us)
      1   INDEX RANGE SCAN T_SMT_MAR_COMPANY_CD (cr=3 pr=0 pw=0 time=27 us)(object id
87336)
```


(A_U)
(A_A)
0
200000
400000
600000
800000
1000000


1'400'000
1'200'000
1'000'000
800'000
600'000
400'000
200'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

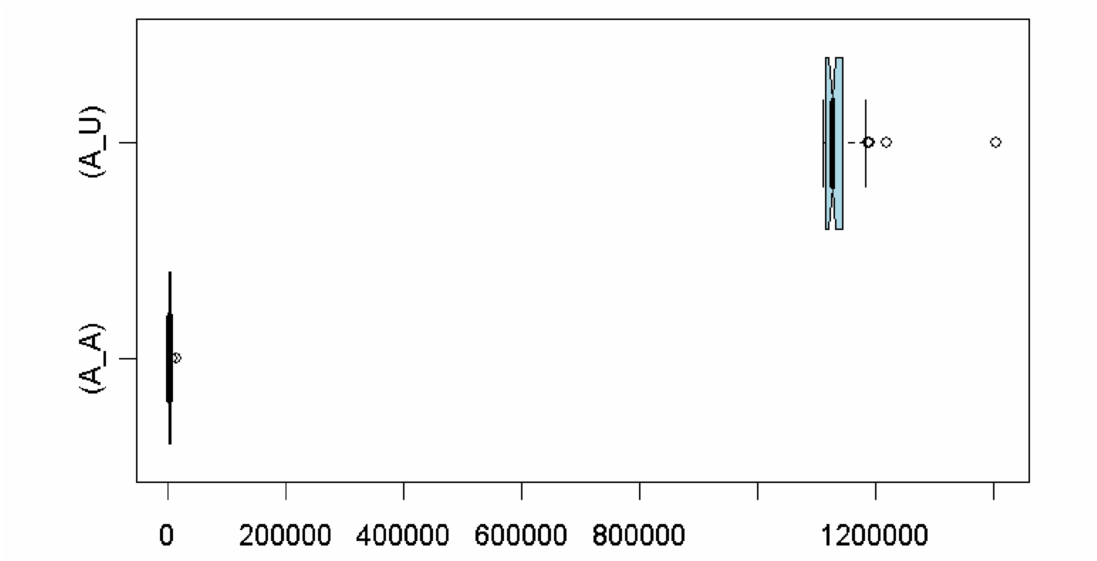
TQP_80421: Avoid implicit applied functions like type conversions

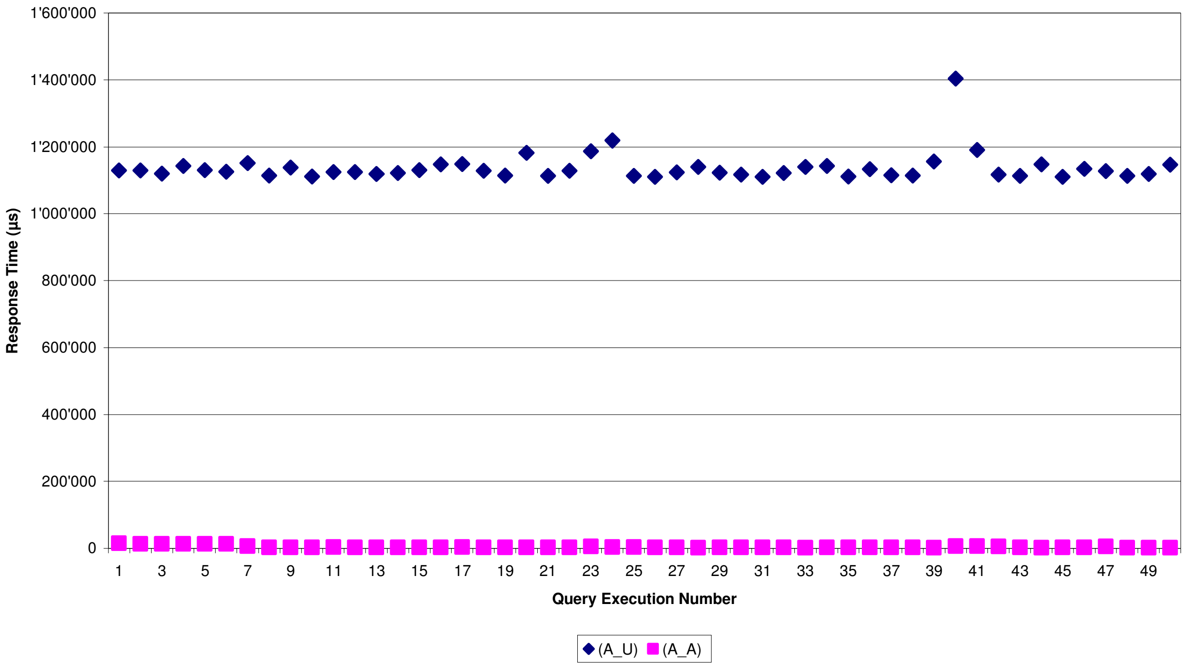
Median 1,127,000 µs (A_U):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD = 2786

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      1.10       1.11      21304      25163          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      1.10       1.11      21304      25163          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=25163 pr=21304 pw=0 time=1114131
us)
```

Median 2,822 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD = '2786'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          4          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0          4          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID TSMT_MDB_ACCOUNT_RUD (cr=4 pr=0 pw=0 time=72 us)
      1   INDEX RANGE SCAN T_SMT_MAR_COMPANY_CD (cr=3 pr=0 pw=0 time=27 us)(object id
87336)
```


(A_U)
(A_A)
0
200000
400000
600000
800000
1200000


1'600'000
1'400'000
1'200'000
1'000'000
800'000
600'000
400'000
200'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

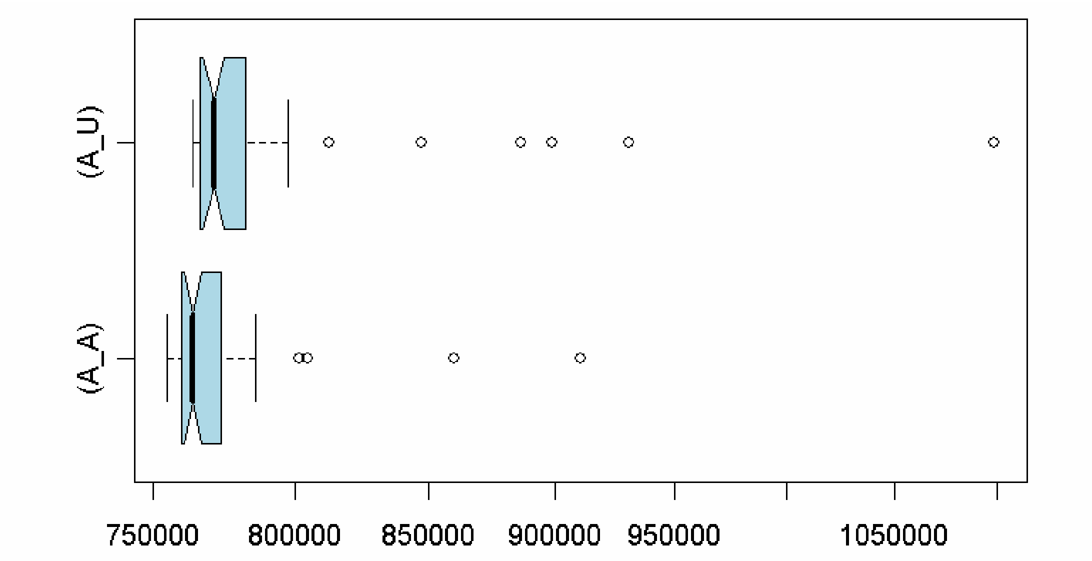
TQP_80431: Avoid leading wildcards - comparing % and _ (result set cardinality 1 of 1,945,431 rows)

Median 770,900 µs (A_U):

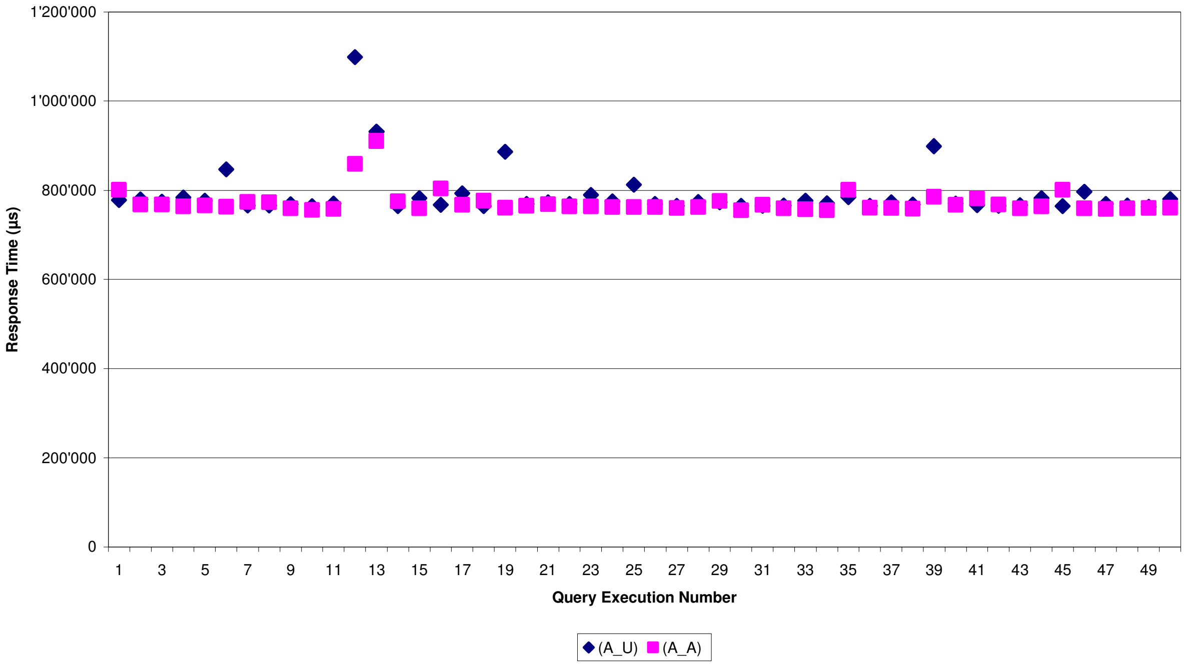
```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD LIKE '_786'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.76       0.76      21310      25163          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.76       0.76      21310      25163          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=25163 pr=21310 pw=0 time=765838
us)
```

Median 763,500 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD LIKE '%786'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.75       0.75      21314      25163          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.75       0.75      21314      25163          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=25163 pr=21314 pw=0 time=754625
us)
```



(Logarithmic scale)



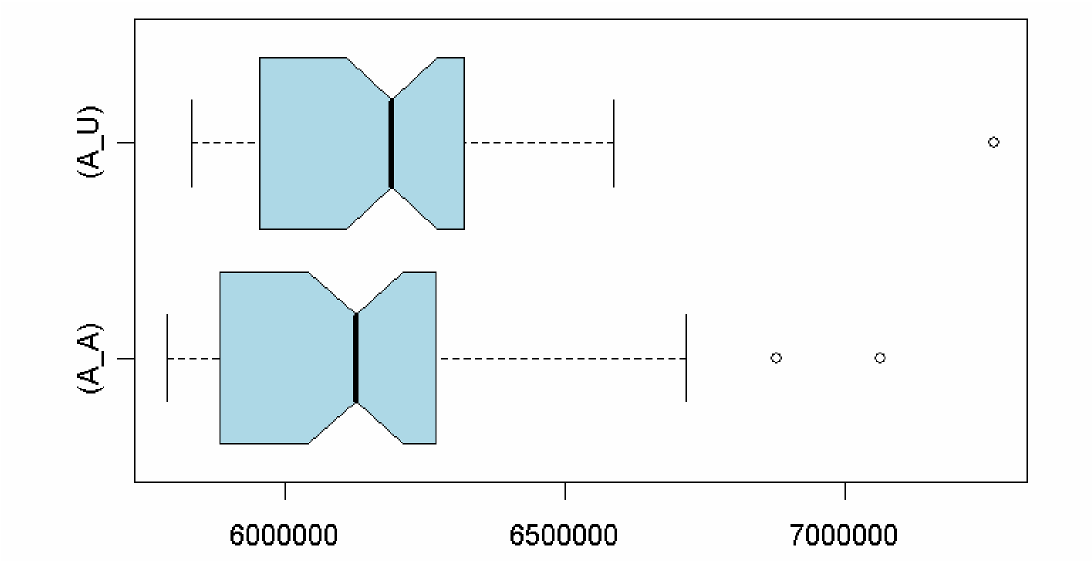
TQP_80432: Avoid leading wildcards - prefer selection with range if possible

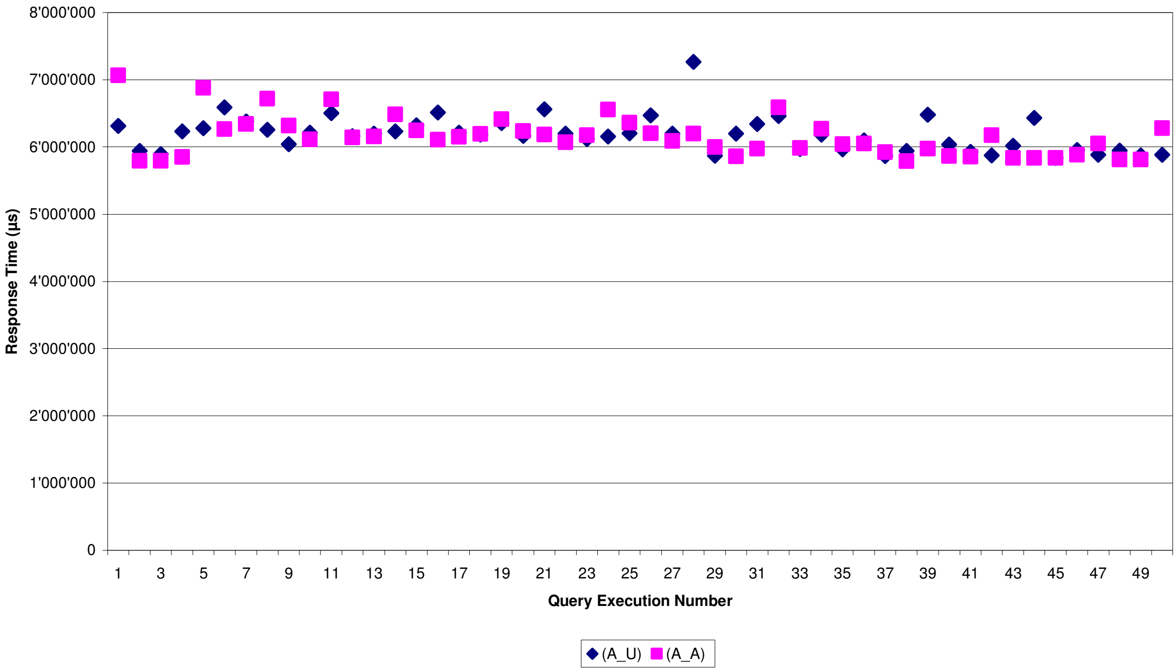
Median 6,189,000 µs (A_U):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD LIKE '27%'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    13891      0.57       0.64          0      31661          0      138908
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    13893      0.57       0.64          0      31661          0      138908

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 138908  TABLE ACCESS BY INDEX ROWID TSMT_MDB_ACCOUNT_RUD (cr=31661 pr=0 pw=0
time=972639 us)
 138908   INDEX RANGE SCAN T_SMT_MAR_COMPANY_CD (cr=14171 pr=0 pw=0 time=278960
us)(object id 87336)
```

Median 6,125,000 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD >= '2700'
   AND COMPANY_CD <= '2799'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    13891      0.73       0.59          0      31661          0      138908
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    13893      0.73       0.59          0      31661          0      138908

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 138908  TABLE ACCESS BY INDEX ROWID TSMT_MDB_ACCOUNT_RUD (cr=31661 pr=0 pw=0
time=833703 us)
 138908   INDEX RANGE SCAN T_SMT_MAR_COMPANY_CD (cr=14171 pr=0 pw=0 time=278918
us)(object id 87336)
```


(A_U)
(A_A)
6000000
6500000
7000000


8'000'000
7'000'000
6'000'000
5'000'000
4'000'000
3'000'000
2'000'000
1'000'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

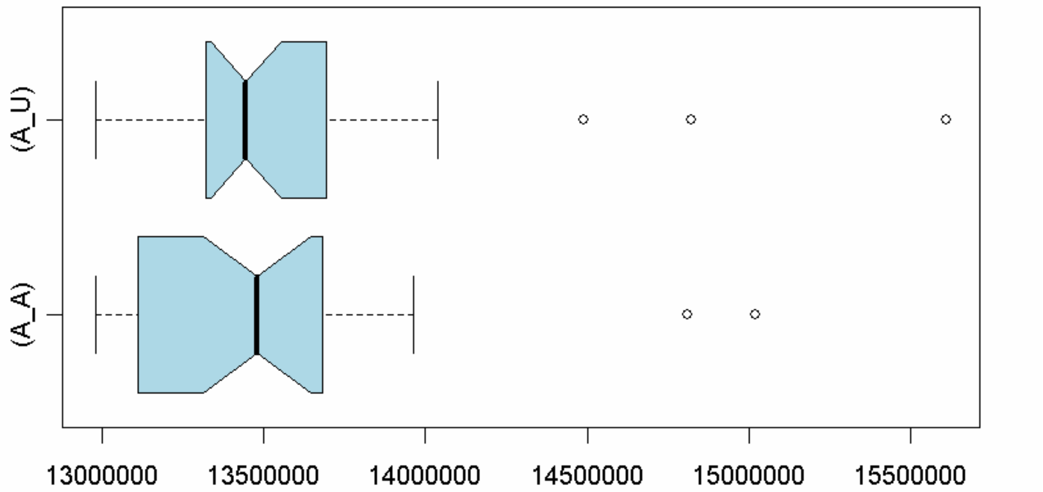
TQP_80433: Avoid leading wildcards - comparing % and _ (result set cardinality 20% of 1,945'431 rows)

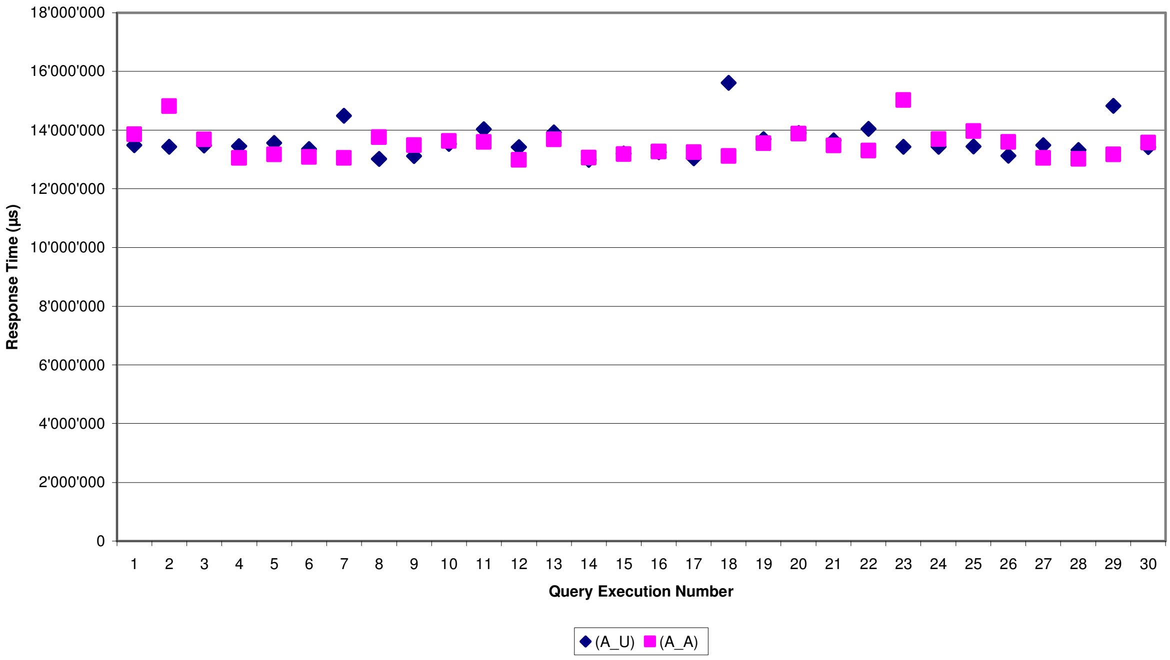
Median 13,450,000 µs (A_U):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD LIKE '_201'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    39043      2.14       1.97      24585      63665          0      390428
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    39045      2.14       1.97      24585      63665          0      390428

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 390428  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=63665 pr=24585 pw=0 time=1562709
us)
```

Median 13,480,000 µs (A_A):

```
SELECT *
  FROM TSMT_MDB_ACCOUNT_RUD
 WHERE COMPANY_CD LIKE '%201'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    39043      1.98       2.01      24585      63665          0      390428
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    39045      1.98       2.01      24585      63665          0      390428

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 390428  TABLE ACCESS FULL TSMT_MDB_ACCOUNT_RUD (cr=63665 pr=24585 pw=0 time=1562680
us)
```





18'000'000
16'000'000
14'000'000
12'000'000
10'000'000
8'000'000
6'000'000
4'000'000
2'000'000
0
Response Time (µs)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Query Execution Number
(A_U) (A_A)

JOIN and DISTINCT

TQP_87301: If the projection of a `JOIN` contains a unique key of one table and the key columns of all other tables are joined by an equi-join, then the `DISTINCT` clause in the projection is unnecessary

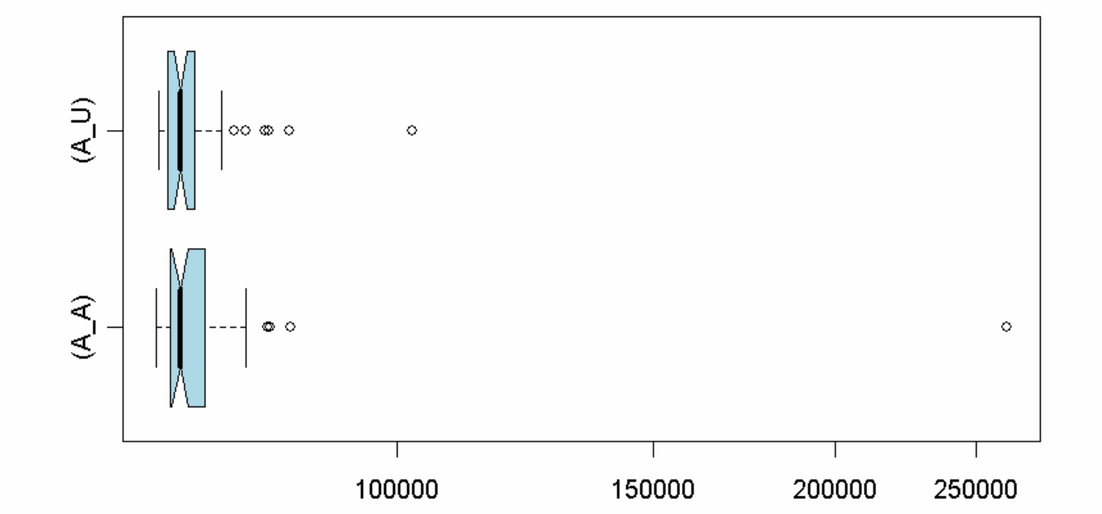
Median 70,930 µs (A_U):

```
SELECT DISTINCT DC.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
              ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_COUNTRY COUY
              ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_REGION_IT RN
              ON COUY.REGION_IT_ID = RN.IDENT
 WHERE DC.PERIOD = CAST ('01.01.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.01          0        734          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.01          0        734          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH UNIQUE (cr=734 pr=0 pw=0 time=18016 us)
   1611   HASH JOIN  (cr=734 pr=0 pw=0 time=19817 us)
    831    HASH JOIN  (cr=41 pr=0 pw=0 time=6786 us)
    308     HASH JOIN  (cr=25 pr=0 pw=0 time=3515 us)
    250      NESTED LOOPS  (cr=9 pr=0 pw=0 time=2572 us)
    250       TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=313 us)
    250       INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=847 us)(object id
83971)
    308      TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=327 us)
    831     TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=870 us)
   1611    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=9678 us)
```

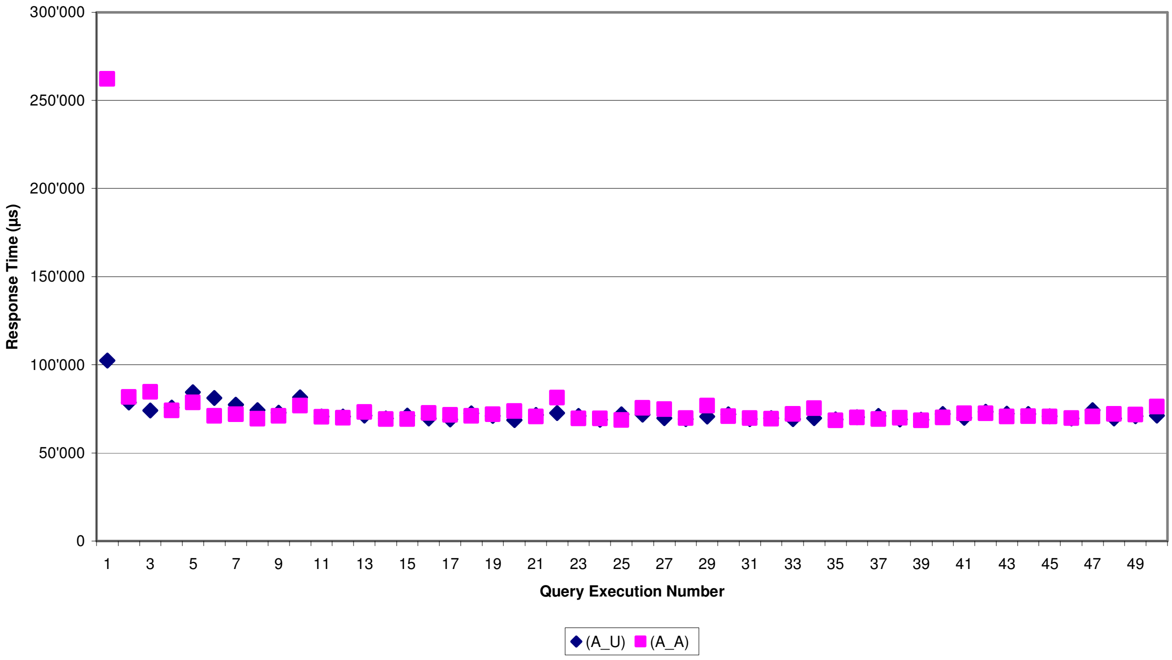
Median 70,840 µs (A_A):

```
SELECT DC.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
              ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_COUNTRY COUY
              ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_REGION_IT RN
              ON COUY.REGION_IT_ID = RN.IDENT
 WHERE DC.PERIOD = CAST ('01.01.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.00       0.01          0        893          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
```

```
total      164      0.00      0.01          0       893          0       1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH JOIN  (cr=893 pr=0 pw=0 time=21238 us)
    831   HASH JOIN  (cr=41 pr=0 pw=0 time=5803 us)
    308    HASH JOIN  (cr=25 pr=0 pw=0 time=3082 us)
    250     NESTED LOOPS  (cr=9 pr=0 pw=0 time=2634 us)
    250      TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=380 us)
    250      INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=854 us)(object id
83971)
    308     TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=327 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=867 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=9385 us)
```



(Logarithmic scale)



Minimise Grouping Columns

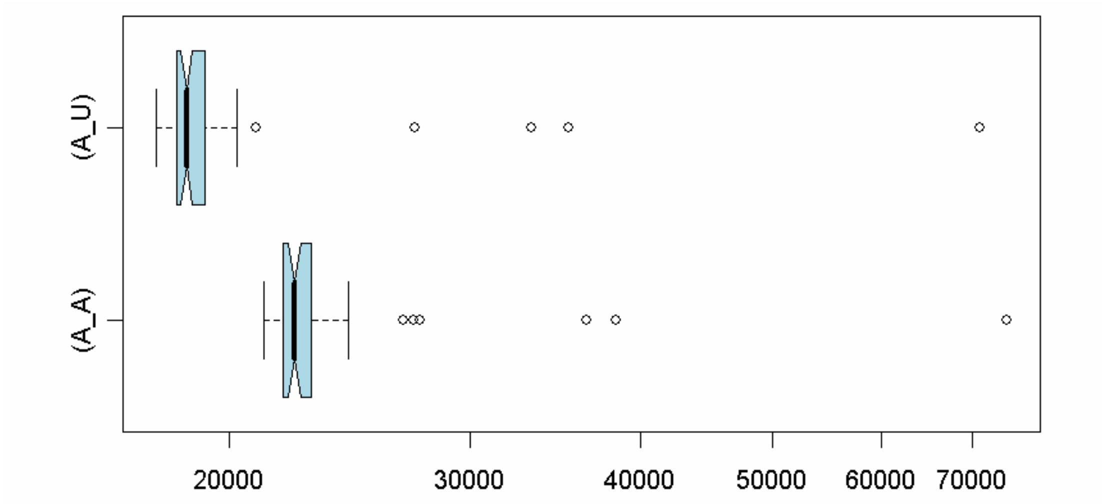
TQP_80701: Keep the number of grouping columns small – combine columns

Median 18,600 µs (A_U):

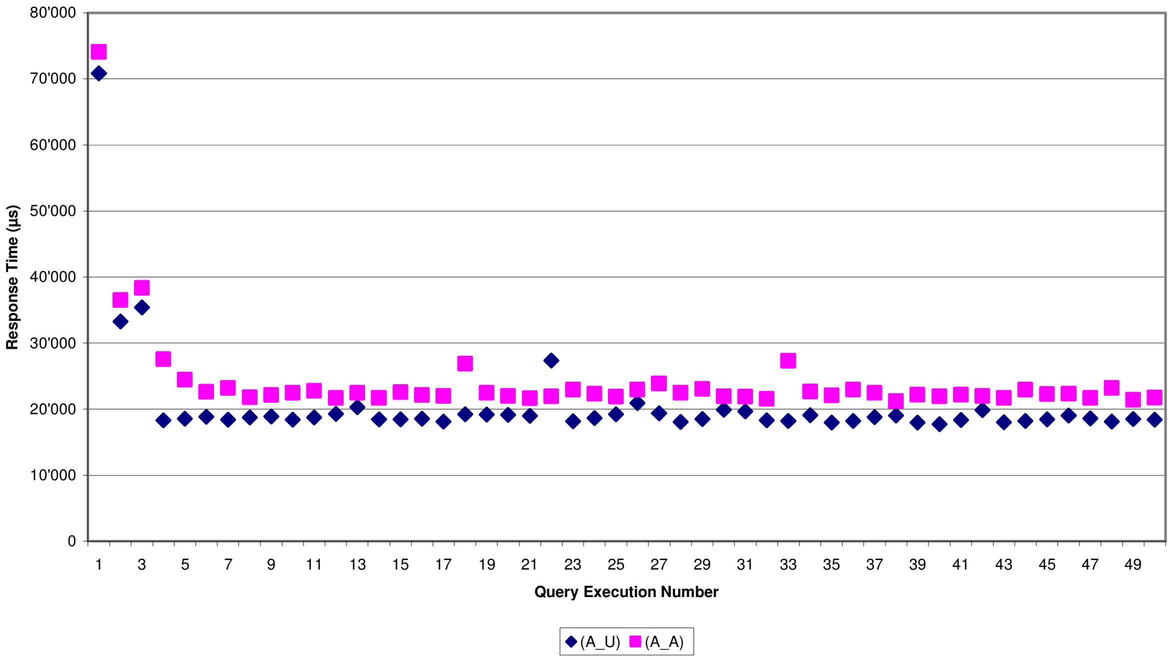
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD, SIGN

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           6

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      6  HASH GROUP BY (cr=693 pr=0 pw=0 time=15769 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13605 us)
```

Median 22,310 µs (A_A):

```
SELECT PERIOD || ' ' || SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD || ' ' || SIGN

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           6

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      6  HASH GROUP BY (cr=693 pr=0 pw=0 time=19138 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13703 us)
```



(Logarithmic scale)



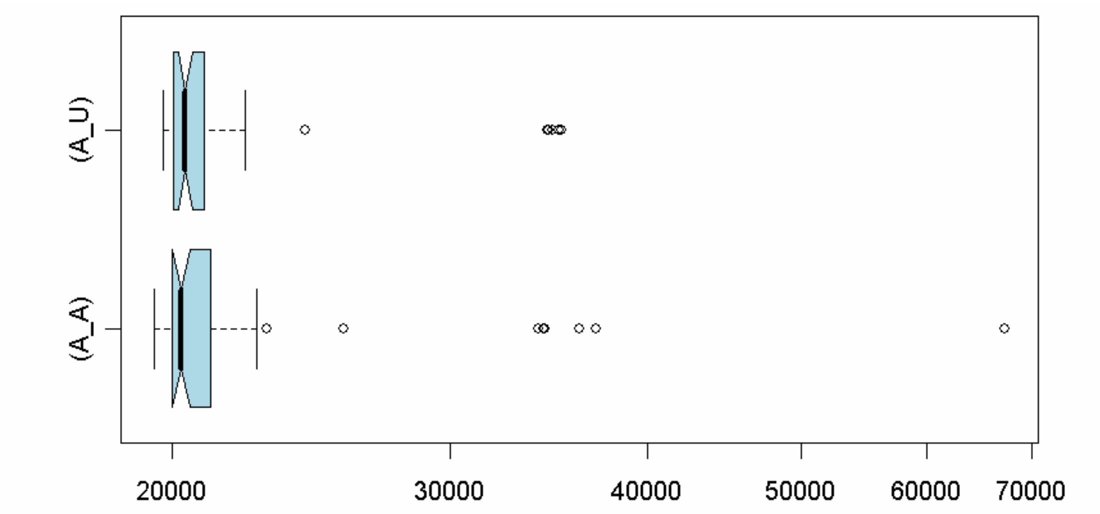
TQP_80702: Keep the number of grouping columns small - eliminate superfluous columns

Median 20,390 µs (A_U):

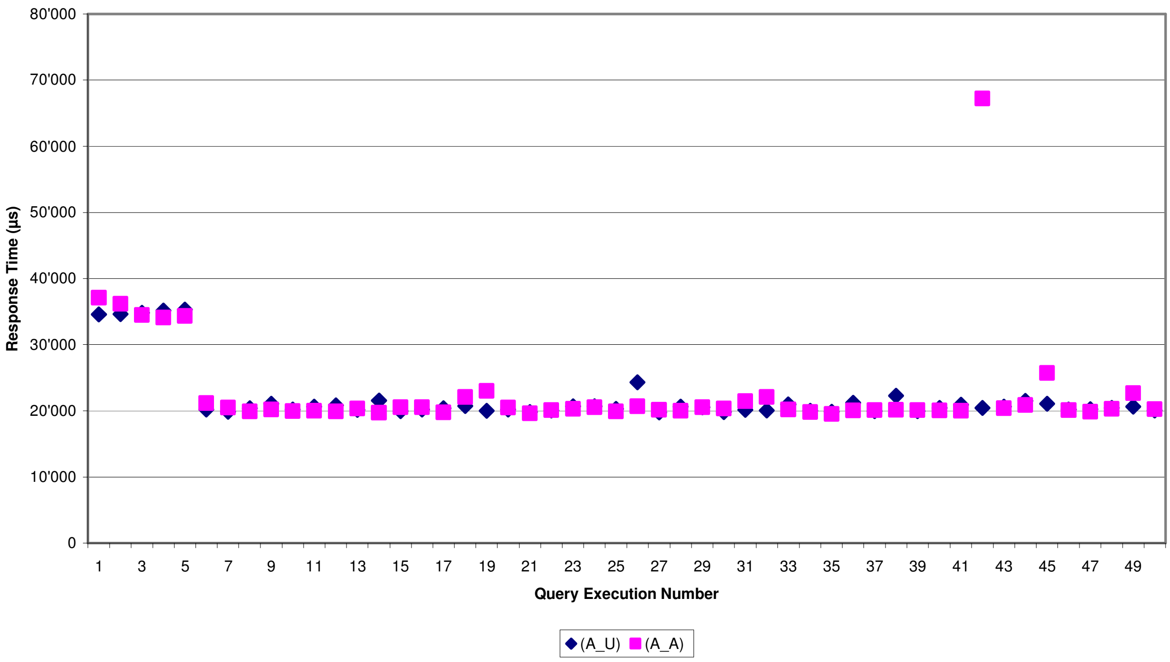
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
   AND SIGN = '+'
 GROUP BY PERIOD, SIGN

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.01          0        693          0           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.01          0        693          0           3

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      3  HASH GROUP BY (cr=693 pr=0 pw=0 time=17647 us)
   4740   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=18691 us)
```

Median 20,270 µs (A_A):

```
SELECT PERIOD, '+', SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
   AND SIGN = '+'
 GROUP BY PERIOD

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.03       0.01          0        693          0           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.03       0.01          0        693          0           3

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      3  HASH GROUP BY (cr=693 pr=0 pw=0 time=17323 us)
   4740   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=18864 us)
```



(Logarithmic scale)



Minimise Sorting Columns

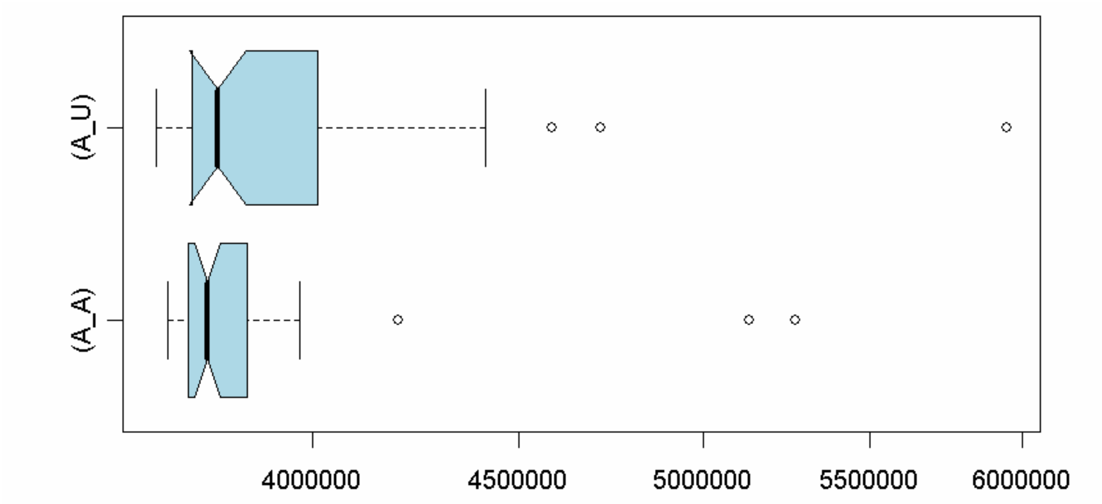
TQP_81001: Minimise the number of columns (expressions) in the `ORDER BY` clause – combine columns

Median 3,791,000 µs (A_U):

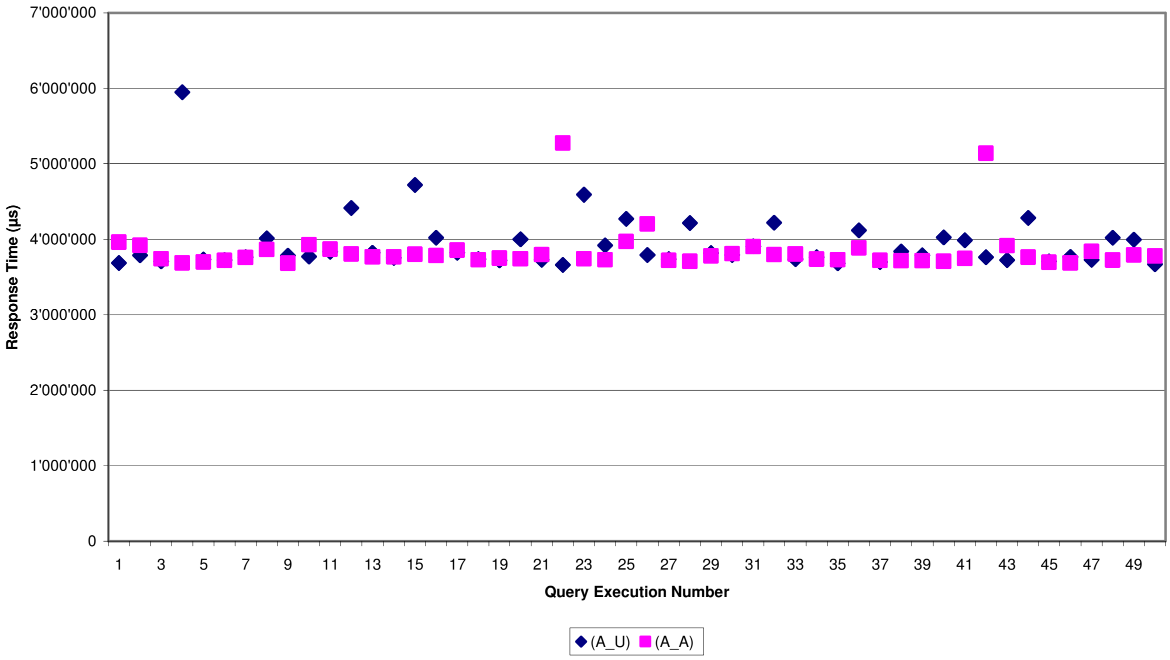
```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.39       0.45       2205       1954          5      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.39       0.45       2205       1954          5      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  SORT ORDER BY (cr=1954 pr=2205 pw=272 time=385269 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1933 pw=0 time=102901
us)
```

Median 3,768,000 µs (A_A):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 ORDER BY SERVICE_RECEIVER_ID * 100000000 + IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.54       0.49       2190       1954          4      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.54       0.49       2190       1954          4      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  SORT ORDER BY (cr=1954 pr=2190 pw=250 time=420611 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1940 pw=0 time=102851
us)
```



(Logarithmic scale)



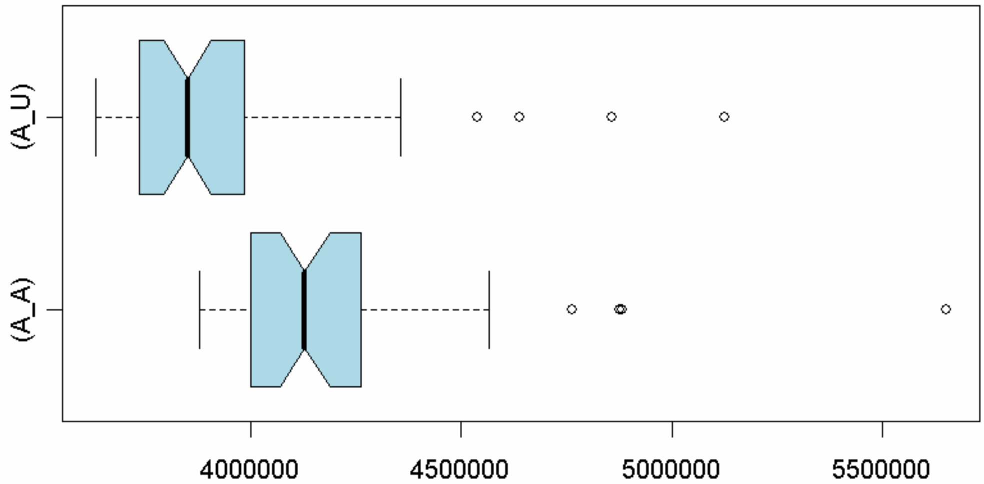
TQP_81002: Minimise the number of columns (expressions) in the `ORDER BY` clause – combine columns

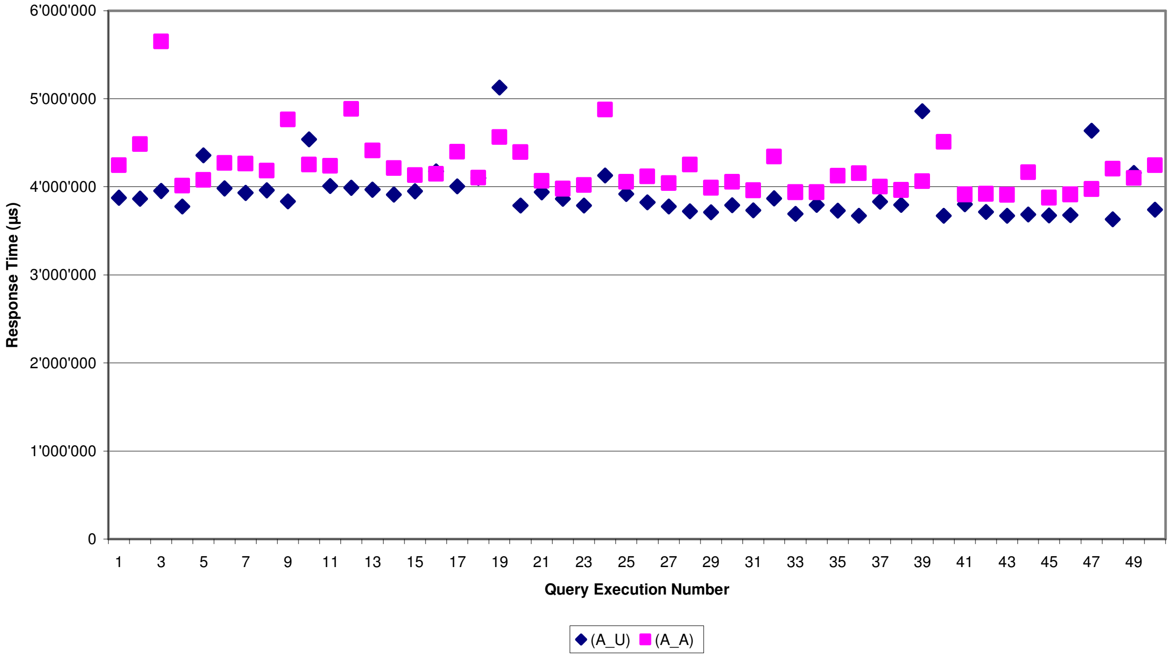
Median 3,850,000 µs (A_U):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.48       0.50       2212       1954          5      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.48       0.50       2212       1954          5      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  SORT ORDER BY (cr=1954 pr=2212 pw=272 time=377555 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1940 pw=0 time=102860
us)
```

Median 4,128,000 µs (A_A):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 ORDER BY SERVICE_RECEIVER_ID || IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.76       0.70       2353       1954          6      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.76       0.70       2353       1954          6      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  SORT ORDER BY (cr=1954 pr=2353 pw=423 time=617917 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1930 pw=0 time=102847
us)
```




6'000'000
5'000'000
4'000'000
3'000'000
2'000'000
1'000'000
0
Response Time (µs)
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
Query Execution Number
(A_U)
(A_A)

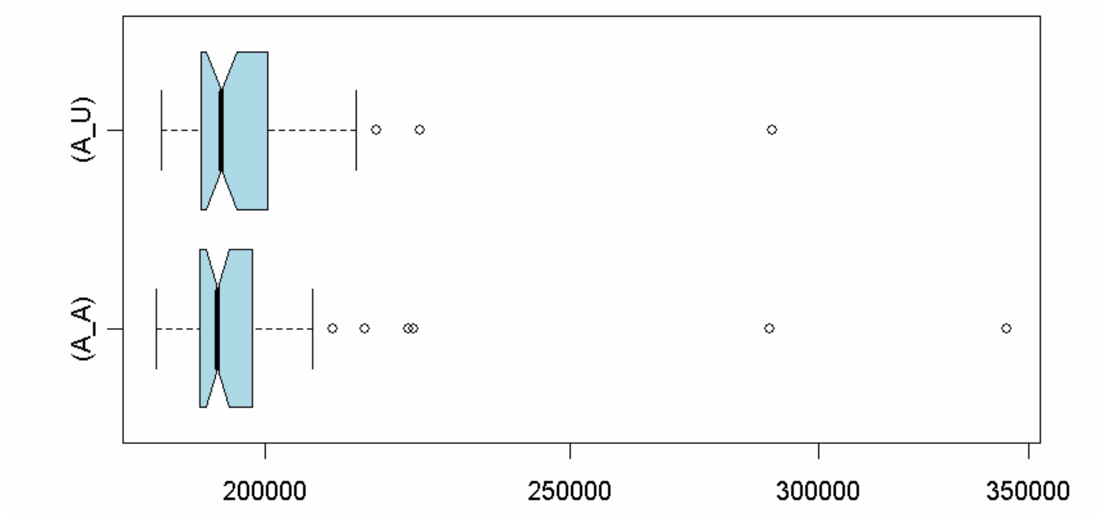
TQP_81003: Minimise the number of columns (expressions) in the `ORDER BY` clause – eliminate superfluous columns

Median 193,700 µs (A_U):

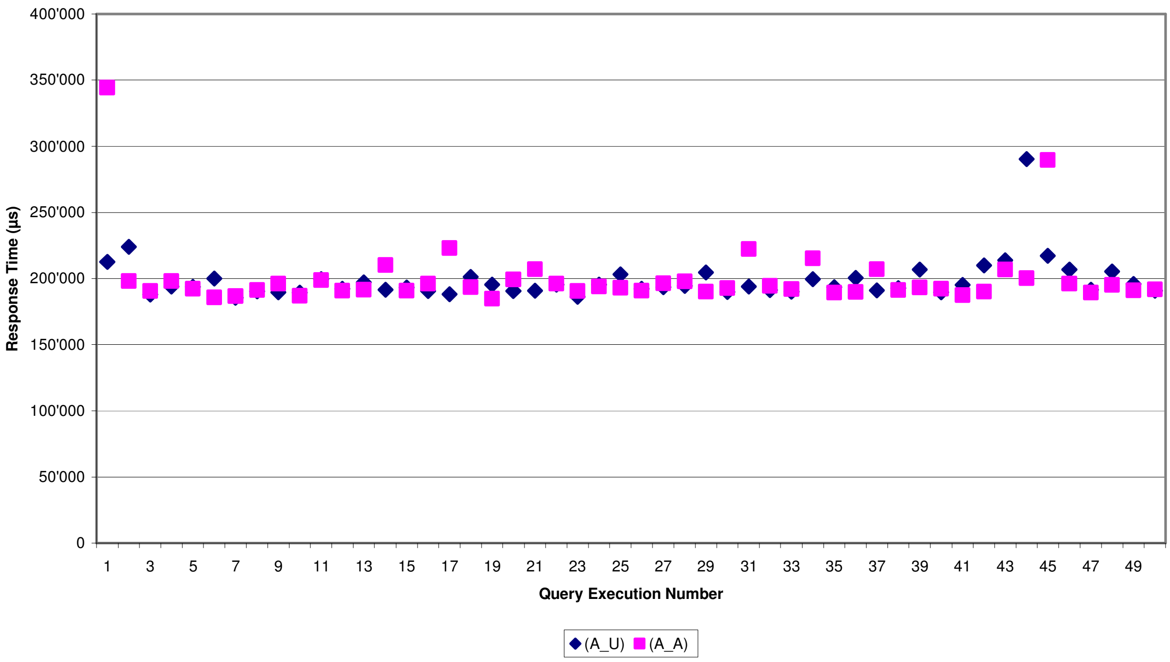
```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
 ORDER BY PERIOD, SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      440      0.03       0.05       1935       1954          0        4392
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      442      0.03       0.05       1935       1954          0        4392

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   4392  SORT ORDER BY (cr=1954 pr=1935 pw=0 time=55265 us)
   4392   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1935 pw=0 time=4677 us)
```

Median 193,200 µs (A_A):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
 ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      440      0.04       0.05       1934       1954          0        4392
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      442      0.04       0.05       1934       1954          0        4392

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   4392  SORT ORDER BY (cr=1954 pr=1934 pw=0 time=55705 us)
   4392   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1934 pw=0 time=4694 us)
```



(Logarithmic scale)



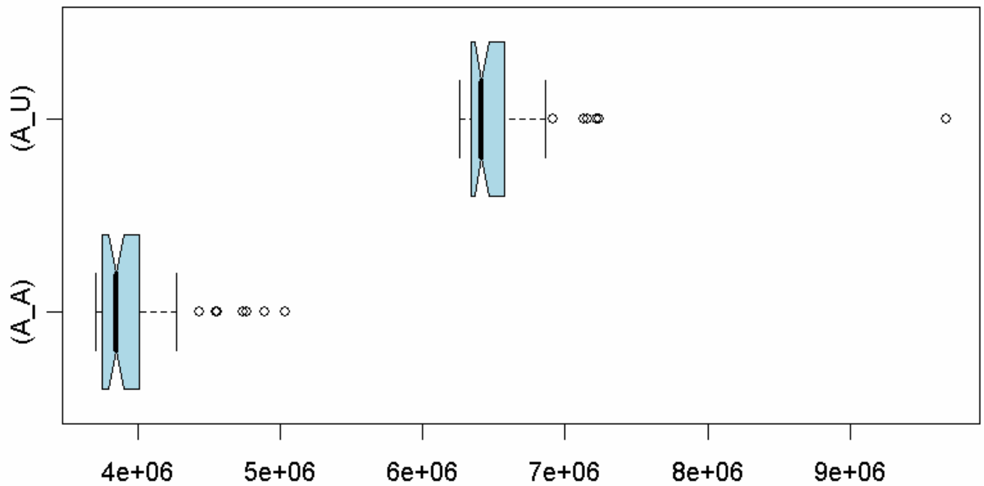
TQP_81011: Minimise the number of columns to be projected

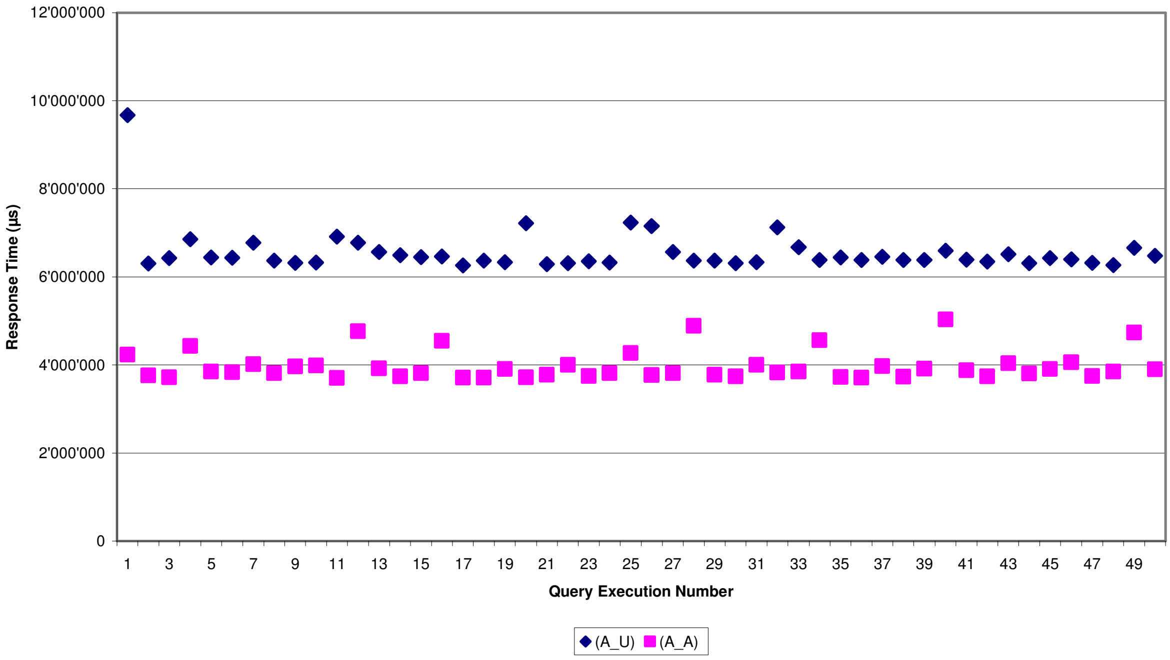
Median 6,415,000 µs (A_U):

```
SELECT *
  FROM TSMT_DETAILED_METERING
ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      1.68       1.78       4037       1954         15      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      1.68       1.78       4037       1954         15      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  SORT ORDER BY (cr=1954 pr=4037 pw=2124 time=1556130 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1913 pw=0 time=102867
us)
```

Median 3,845,000 µs (A_A):

```
SELECT AMOUNT_SERVICE
  FROM TSMT_DETAILED_METERING
 ORDER BY SERVICE_RECEIVER_ID, IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    10259      0.32       0.54       2199       1954          5      102583
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    10261      0.32       0.54       2199       1954          5      102583

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
 102583  SORT ORDER BY (cr=1954 pr=2199 pw=272 time=482085 us)
 102583   TABLE ACCESS FULL TSMT_DETAILED_METERING (cr=1954 pr=1927 pw=0 time=102849
us)
```





12'000'000
10'000'000
8'000'000
6'000'000
4'000'000
2'000'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

Ordering `INNER JOIN`

TQP_87201: The smallest table should be the inner table

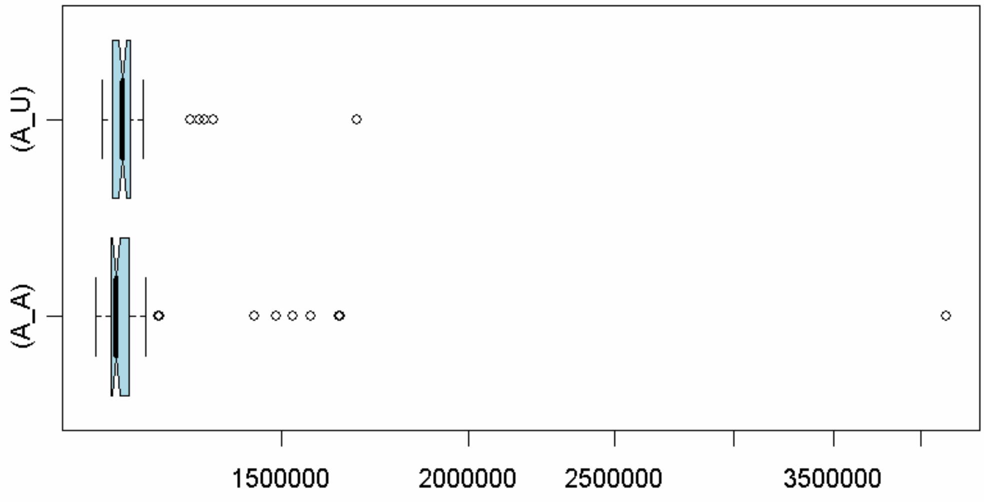
Median 1,175,000 µs (A_U):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT, COUY.IDENT, RN.IDENT
  FROM TSMT_REGION_IT RN
       INNER JOIN TSMT_COUNTRY COUY
             ON RN.IDENT = COUY.REGION_IT_ID
       INNER JOIN TSMT_COMPANY COMY
             ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON DC.SERVICE_RECEIVER_ID = SR.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.08          0       3993          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.08          0       3993          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3993 pr=0 pw=0 time=171126 us)
    831   HASH JOIN  (cr=41 pr=0 pw=0 time=6508 us)
    308    HASH JOIN  (cr=25 pr=0 pw=0 time=3281 us)
    250     NESTED LOOPS  (cr=9 pr=0 pw=0 time=2561 us)
    250      TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=555 us)
    250      INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=848 us)(object id
83971)
    308     TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=326 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=854 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33280 us)
```

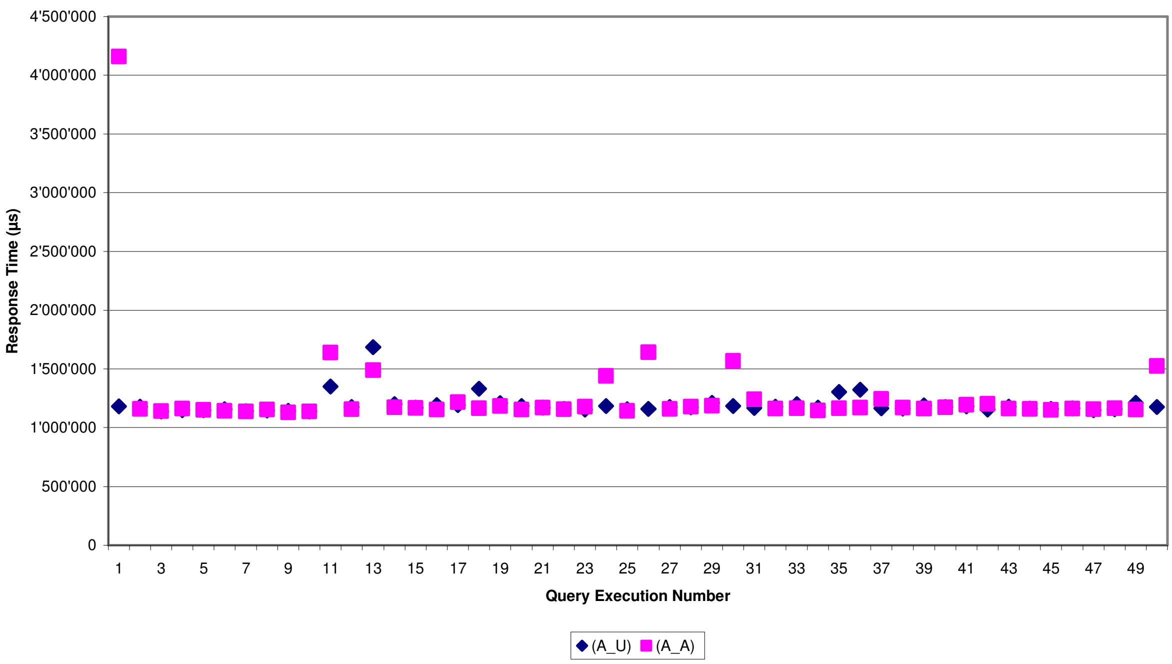
Median 1,164,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT, COUY.IDENT, RN.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_COUNTRY COUY
             ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_REGION_IT RN
             ON COUY.REGION_IT_ID = RN.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.08          0       3993          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.08          0       3993          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75
```

```
Rows       Row Source Operation
-------    ---------------------------------------------------
  33183    HASH JOIN  (cr=3993 pr=0 pw=0 time=171269 us)
    831     HASH JOIN  (cr=41 pr=0 pw=0 time=6683 us)
    308      HASH JOIN  (cr=25 pr=0 pw=0 time=8196 us)
    250       NESTED LOOPS  (cr=9 pr=0 pw=0 time=2556 us)
    250        TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=550 us)
    250        INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=859 us)(object id
83971)
    308       TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=325 us)
    831      TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=863 us)
  33183     TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33277 us)
```



(Logarithmic scale)



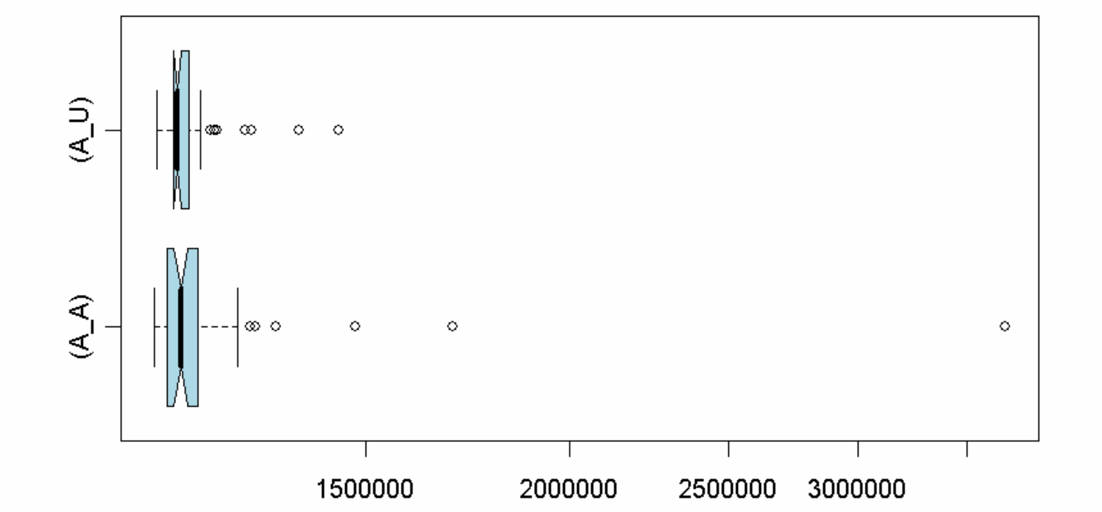
TQP_87211: The table with the best index should be the inner table

Median 1,150,000 µs (A_U):

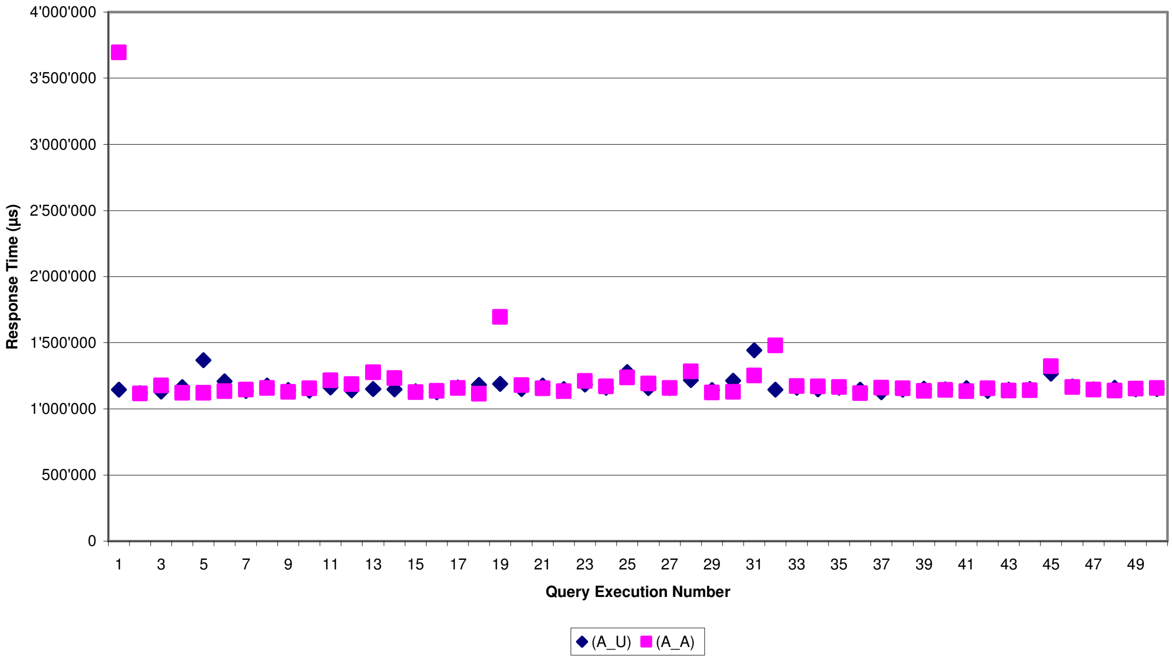
```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
              ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171601 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6700 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=875 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2896 us)(object id 
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=66441 us)
```

Median 1,156,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_SERVICE_RECEIVER SR
       INNER JOIN TSMT_DETAILED_CHARGING DC
              ON SR.IDENT = DC.SERVICE_RECEIVER_ID
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.01       0.00          0          0          0           0
Fetch     3319      0.01       0.07          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.03       0.07          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171757 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6722 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=896 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2896 us)(object id 
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33260 us)
```



(Logarithmic scale)



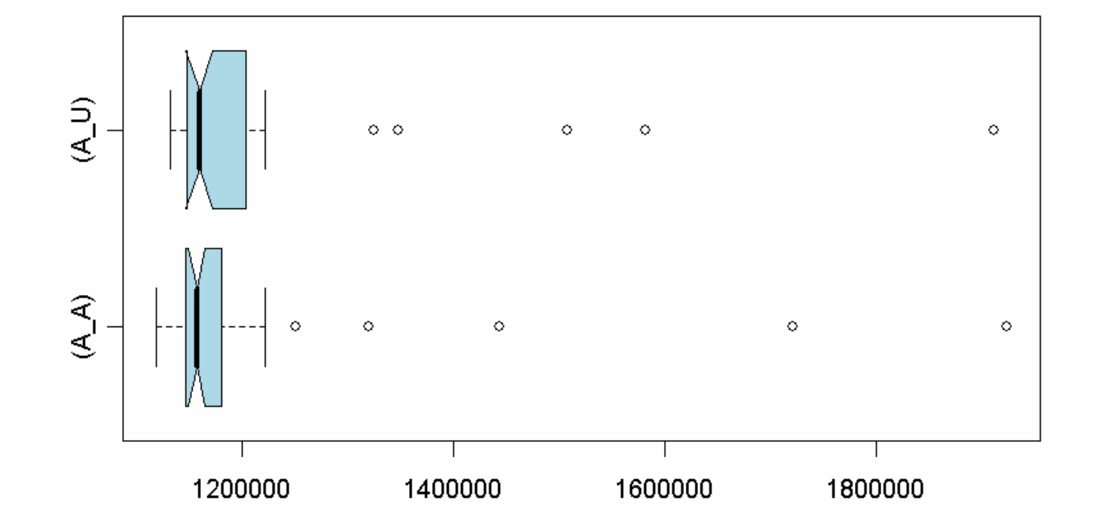
TQP_87212: The table with the best index should be the inner table

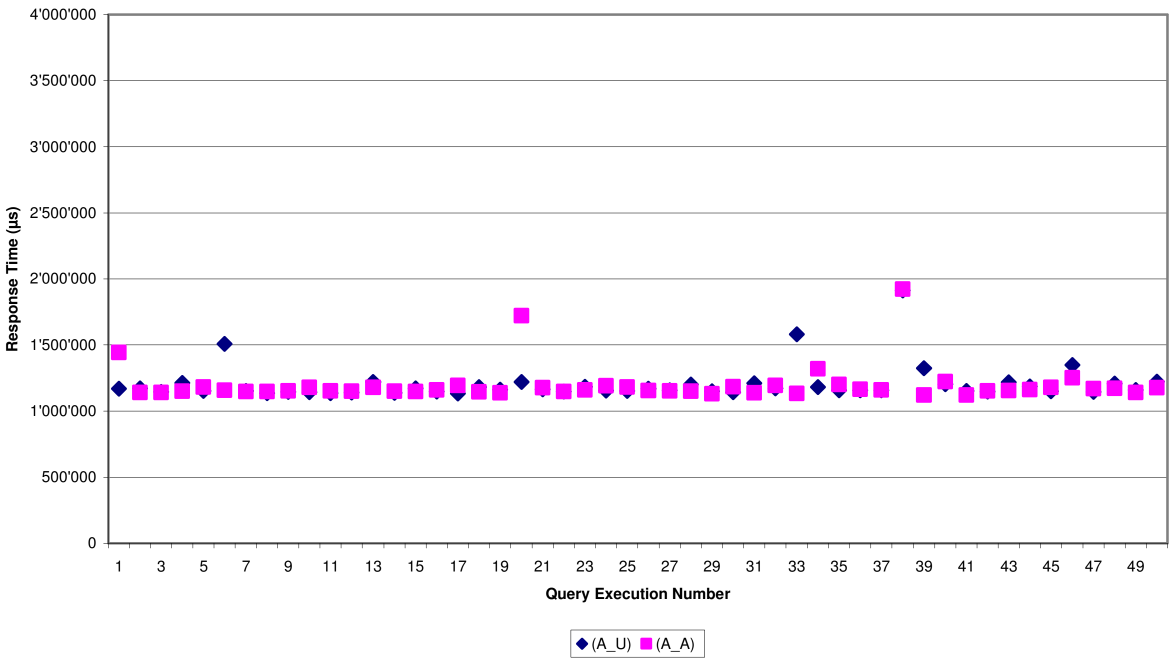
Median 1,159,000 µs (A_U):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
              ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.10       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.10       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171533 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6702 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=876 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2911 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33259 us)
```

Median 1,146,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_SERVICE_RECEIVER SR
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_DETAILED_CHARGING DC
              ON SR.IDENT = DC.SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.01       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.01       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171643 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6703 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=876 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2941 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33261 us)
```


(A_U)
(A_A)
1200000
1400000
1600000
1800000


4'000'000
3'500'000
3'000'000
2'500'000
2'000'000
1'500'000
1'000'000
500'000
0
Response Time (µs)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

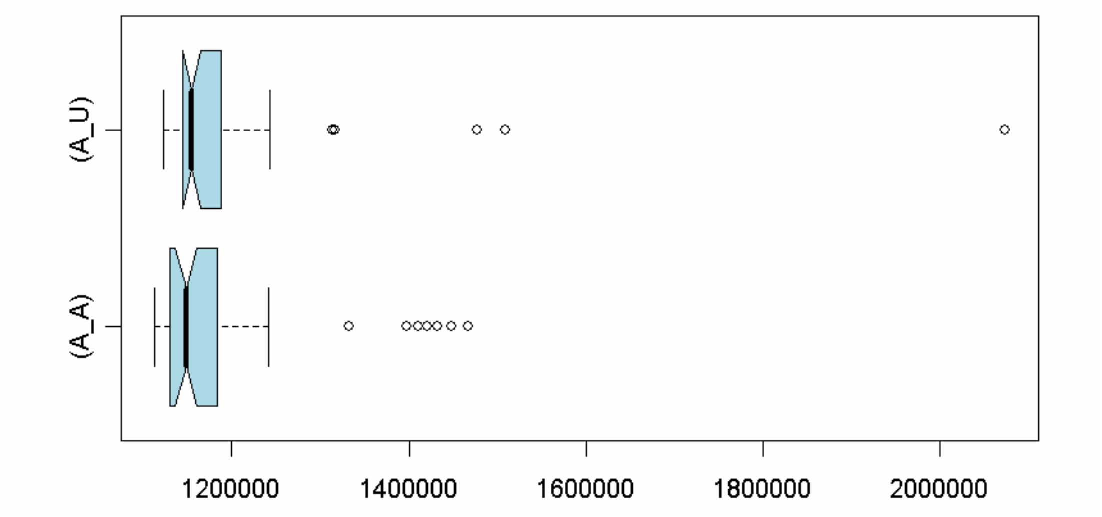
TQP_87213: The table with the best index should be the inner table

Median 1,154,000 µs (A_U):

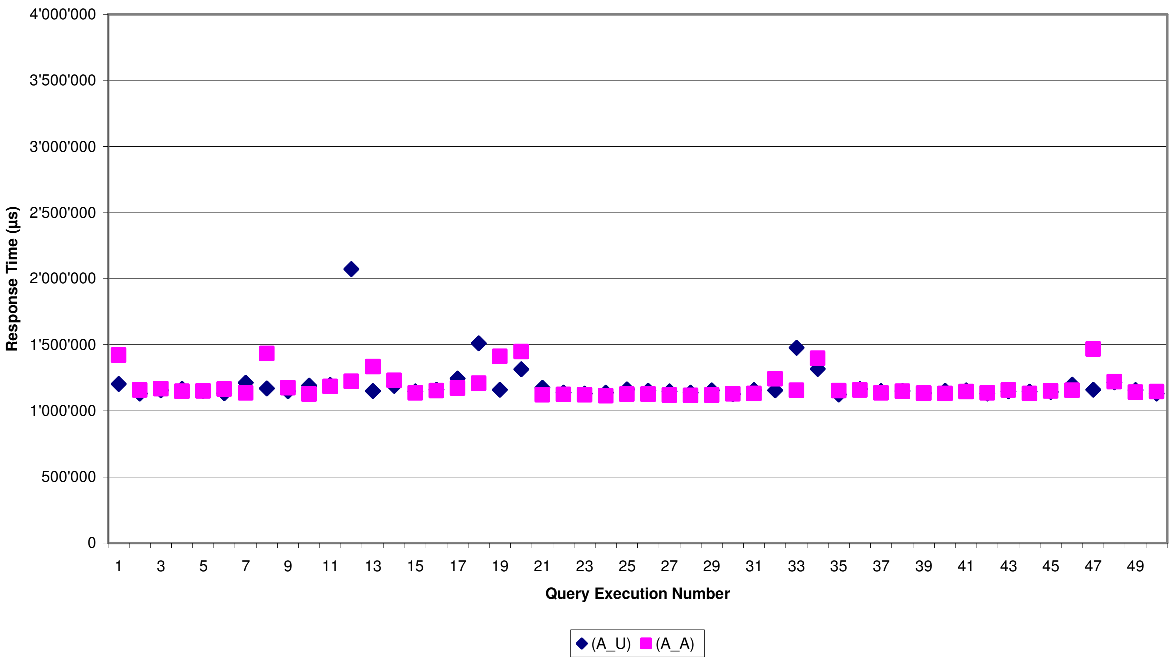
```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
              ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.07       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.07       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171554 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6707 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=880 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2894 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33262 us)
```

Median 1,149,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_COMPANY COMY
       INNER JOIN TSMT_SERVICE_RECEIVER SR
              ON COMY.IDENT = SR.COMPANY_CD
       INNER JOIN TSMT_DETAILED_CHARGING DC
              ON SR.IDENT = DC.SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.12       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.12       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171720 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6713 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=884 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2877 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33262 us)
```



(A_U)
(A_A)
1200000
1400000
1600000
1800000
2000000



Response Time (µs)
4'000'000
3'500'000
3'000'000
2'500'000
2'000'000
1'500'000
1'000'000
500'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

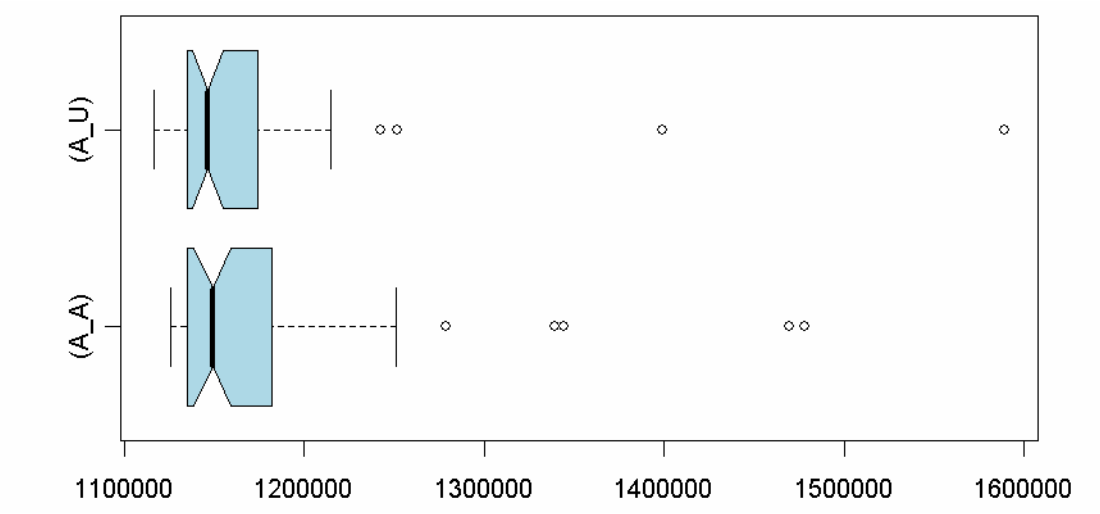
TQP_87214: The table with the best index should be the inner table

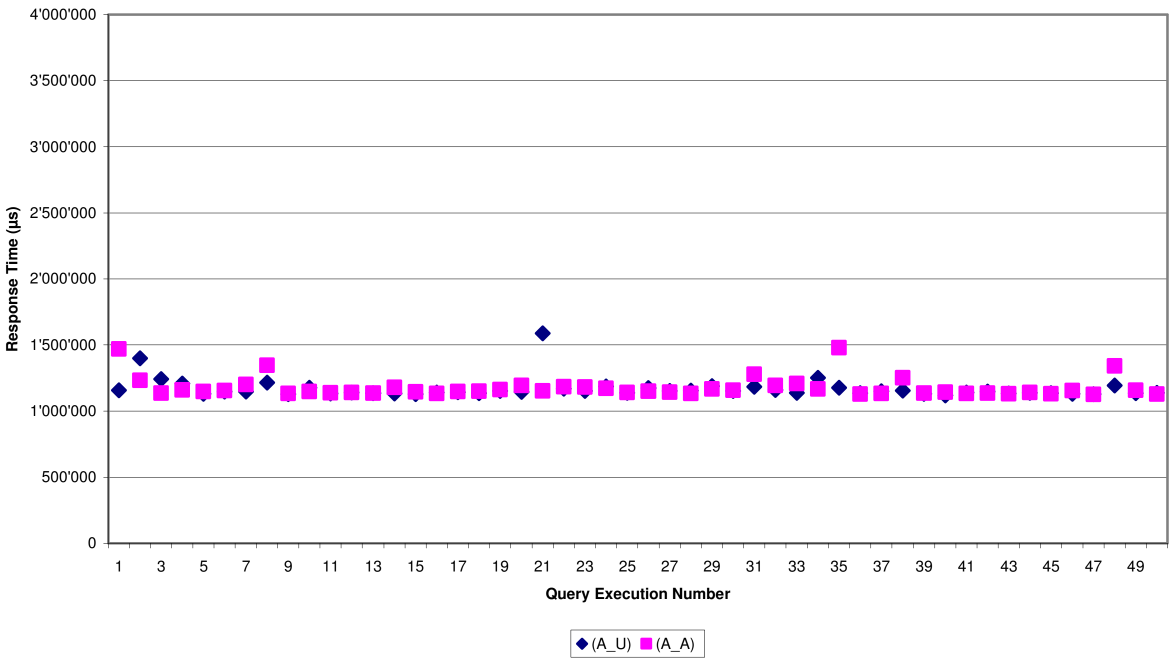
Median 1,146,000 µs (A_U):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_SERVICE_RECEIVER SR
       INNER JOIN TSMT_DETAILED_CHARGING DC
              ON SR.IDENT = DC.SERVICE_RECEIVER_ID
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.09          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.09          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171566 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6704 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=879 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2901 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=66438 us)
```

Median 1,149,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_SERVICE_RECEIVER SR
       INNER JOIN TSMT_COMPANY COMY
              ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_DETAILED_CHARGING DC
              ON SR.IDENT = DC.SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171753 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6700 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=876 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2938 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33260 us)
```


(A_U)
(A_A)
1100000
1200000
1300000
1400000
1500000
1600000


4'000'000
3'500'000
3'000'000
2'500'000
2'000'000
1'500'000
1'000'000
500'000
0
Response Time (µs)
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
Query Execution Number
(A_U)
(A_A)

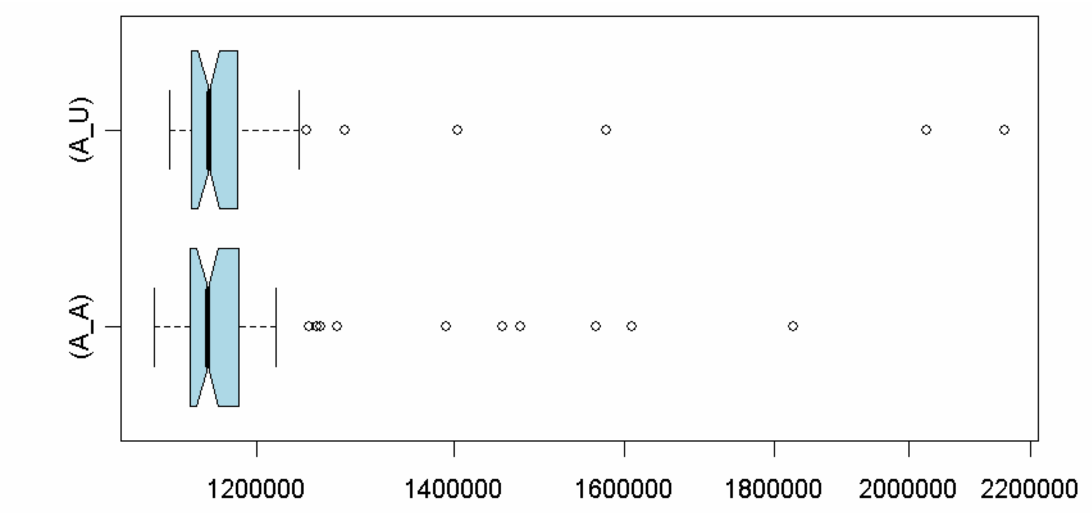
TQP_87215: The table with the best index should be the inner table

Median 1,155,000 µs (A_U):

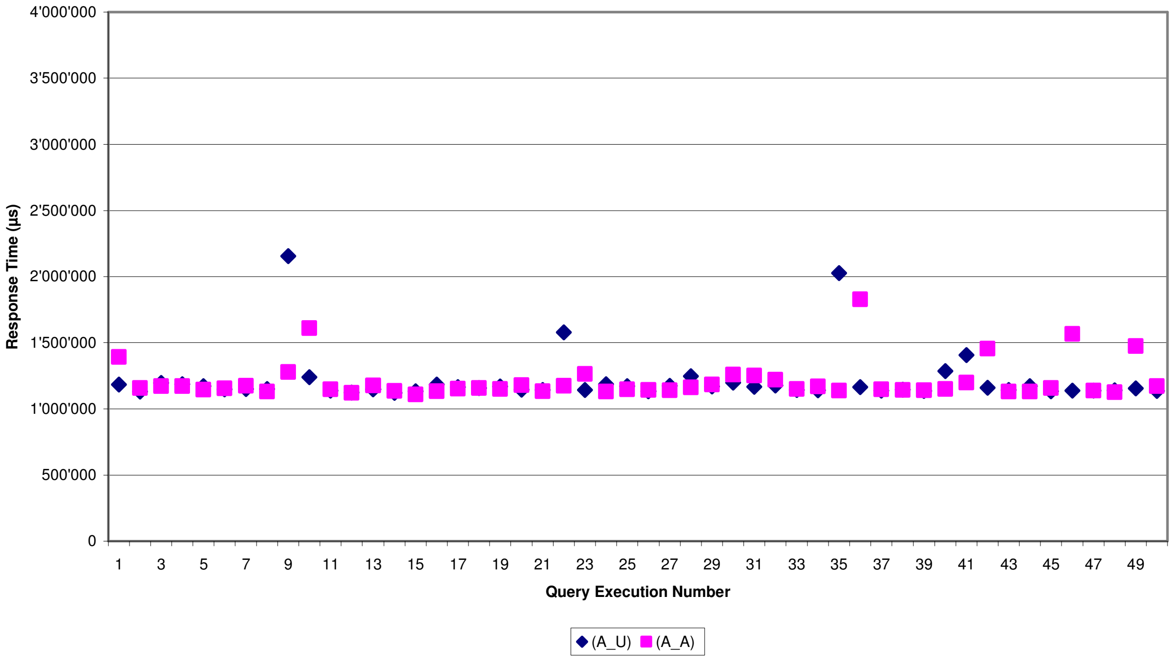
```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_SERVICE_RECEIVER SR
        INNER JOIN TSMT_DETAILED_CHARGING DC
               ON SR.IDENT = DC.SERVICE_RECEIVER_ID
        INNER JOIN TSMT_COMPANY COMY
               ON SR.COMPANY_CD = COMY.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.18       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.18       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171762 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6702 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=876 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=2942 us)(object id 
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33277 us)
```

Median 1,155,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_COMPANY COMY
        INNER JOIN TSMT_SERVICE_RECEIVER SR
               ON COMY.IDENT = SR.COMPANY_CD
        INNER JOIN TSMT_DETAILED_CHARGING DC
               ON SR.IDENT = DC.SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.09       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.09       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=172639 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=17496 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=10010 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=3284 us)(object id 
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33278 us)
```



(Logarithmic scale)



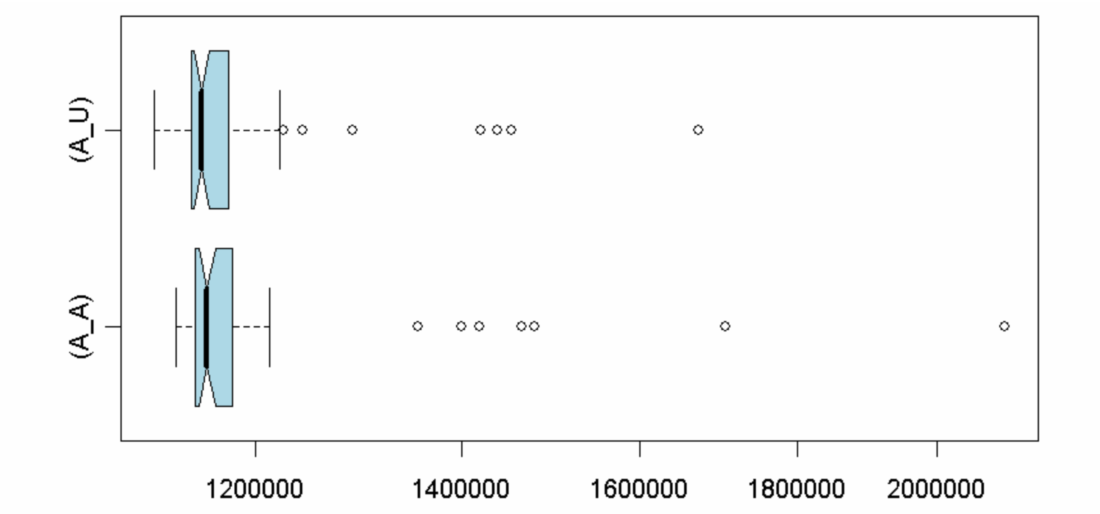
TQP_87216: The table with the best index should be the inner table

Median 1,153,000 µs (A_U):

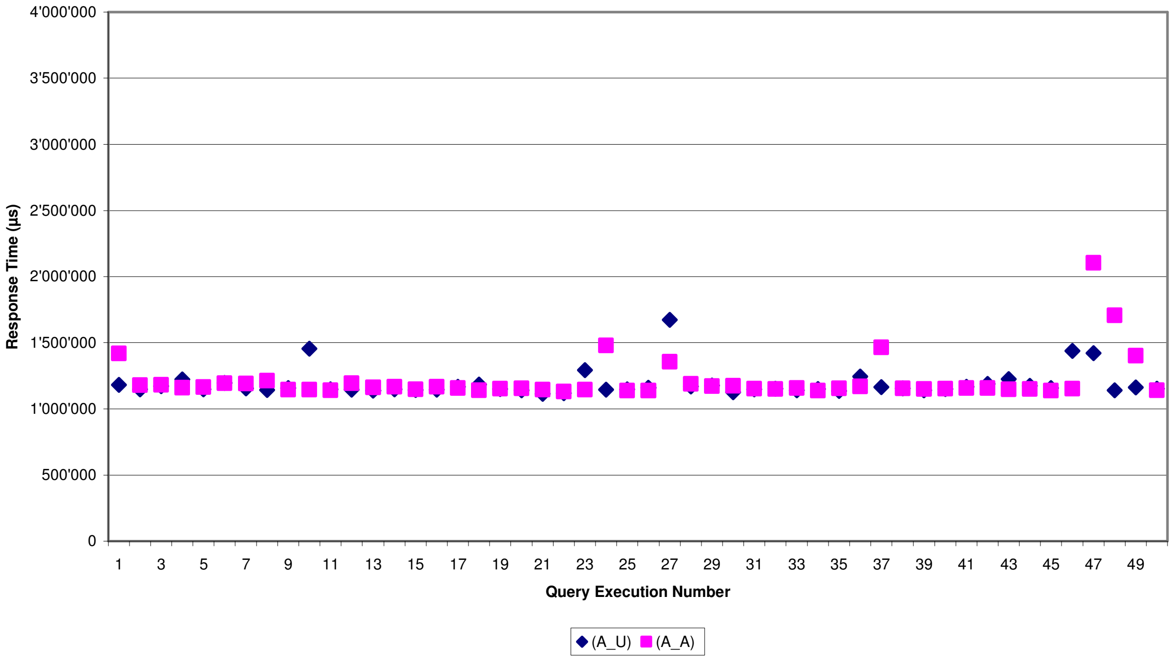
```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_SERVICE_RECEIVER SR
       INNER JOIN TSMT_COMPANY COMY
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON SR.IDENT = DC.SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.14       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.14       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=171798 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6712 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=887 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=3022 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33267 us)
```

Median 1,157,000 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT
  FROM TSMT_COMPANY COMY
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON COMY.IDENT = SR.COMPANY_CD
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON SR.IDENT = DC.SERVICE_RECEIVER_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     3319      0.04       0.08          0       3970          0       33183
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3321      0.04       0.08          0       3970          0       33183

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
  33183  HASH JOIN  (cr=3970 pr=0 pw=0 time=172800 us)
    831   NESTED LOOPS  (cr=18 pr=0 pw=0 time=6706 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=881 us)
    831    INDEX UNIQUE SCAN PK_T_SMT_COMY (cr=2 pr=0 pw=0 time=3369 us)(object id
83857)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=3952 pr=0 pw=0 time=33268 us)
```



(Logarithmic scale)



TQP_87221: The table with the most restrictive clause should be the outer table - selection of a small table

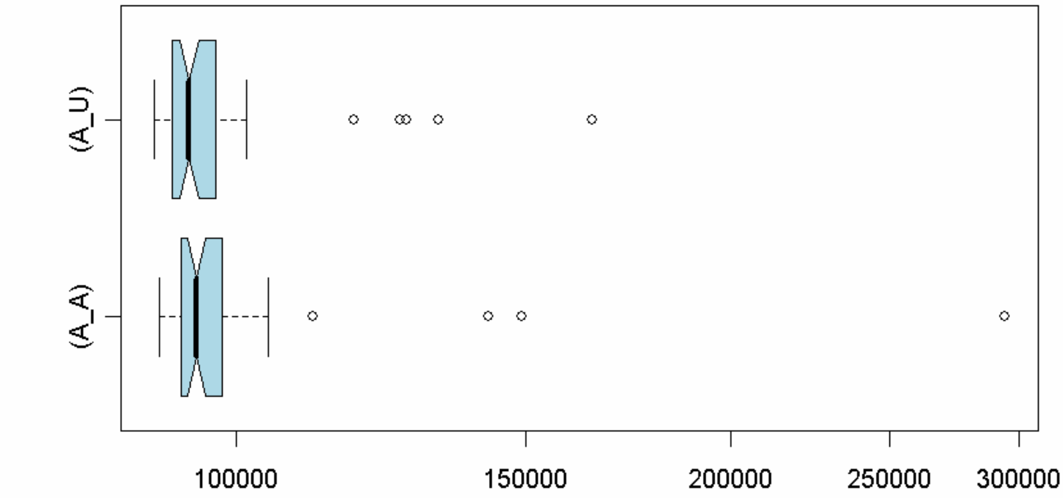
Median 93,600 µs (A_U):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT, COUY.IDENT, RN.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_COUNTRY COUY
             ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_REGION_IT RN
             ON COUY.REGION_IT_ID = RN.IDENT
            AND NOT RN.NAME = 'EMEA'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      191      0.03       0.02          0       1172          0        1909
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      193      0.03       0.02          0       1172          0        1909

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1909  HASH JOIN  (cr=1172 pr=0 pw=0 time=9875 us)
    260   HASH JOIN  (cr=291 pr=0 pw=0 time=5150 us)
    158    HASH JOIN  (cr=275 pr=0 pw=0 time=3927 us)
    132     NESTED LOOPS  (cr=259 pr=0 pw=0 time=4027 us)
    250      TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=303 us)
    132      TABLE ACCESS BY INDEX ROWID TSMT_REGION_IT (cr=252 pr=0 pw=0 time=2415
us)
    250       INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=951 us)(object id
83971)
    308     TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=326 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=868 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=881 pr=0 pw=0 time=33214 us)
```

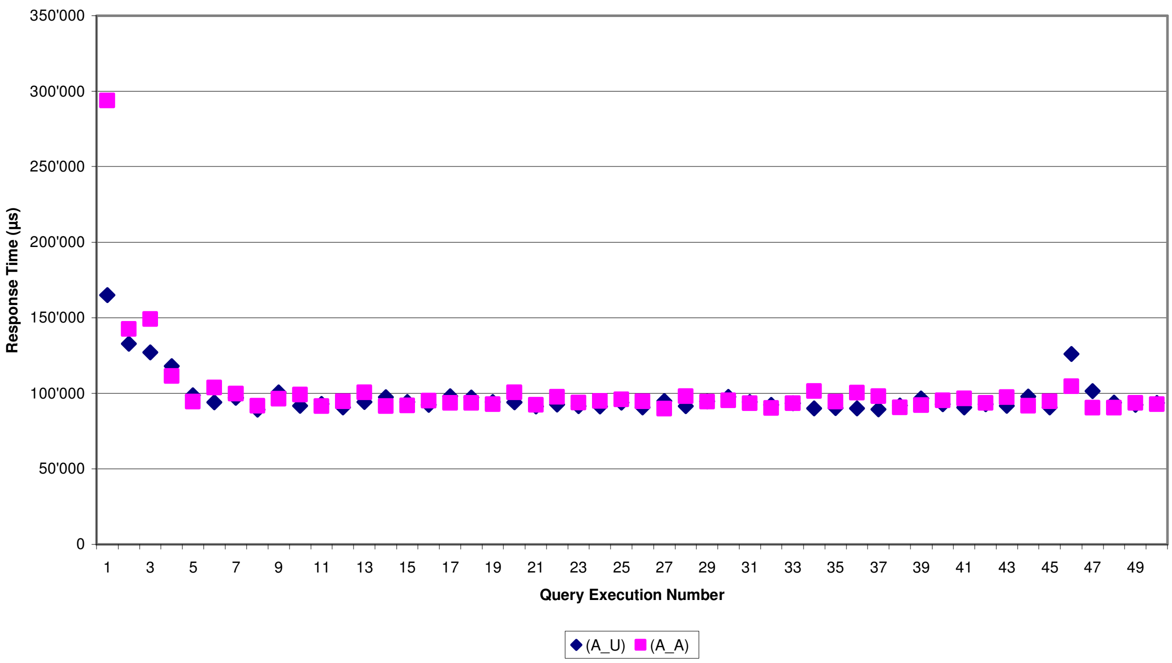
Median 94,600 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT, COUY.IDENT, RN.IDENT
  FROM TSMT_REGION_IT RN
       INNER JOIN TSMT_COUNTRY COUY
             ON RN.IDENT = COUY.REGION_IT_ID
            AND NOT RN.NAME = 'EMEA'
       INNER JOIN TSMT_COMPANY COMY
             ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON DC.SERVICE_RECEIVER_ID = SR.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      191      0.03       0.02          0       1172          0        1909
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      193      0.03       0.02          0       1172          0        1909

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75
```

```
Rows     Row Source Operation
-------  ---------------------------------------------------
   1909  HASH JOIN  (cr=1172 pr=0 pw=0 time=11912 us)
    260   HASH JOIN  (cr=291 pr=0 pw=0 time=5627 us)
    158    HASH JOIN  (cr=275 pr=0 pw=0 time=4221 us)
    132     NESTED LOOPS  (cr=259 pr=0 pw=0 time=4156 us)
    250      TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=550 us)
    132      TABLE ACCESS BY INDEX ROWID TSMT_REGION_IT (cr=252 pr=0 pw=0 time=2482
us)
    250       INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=1026 us)(object id
83971)
    308     TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=341 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=38 us)
  33183   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=881 pr=0 pw=0 time=33213 us)
```



(Logarithmic scale)



TQP_87222: The table with the most restrictive clause should be the outer table - selection of a big table

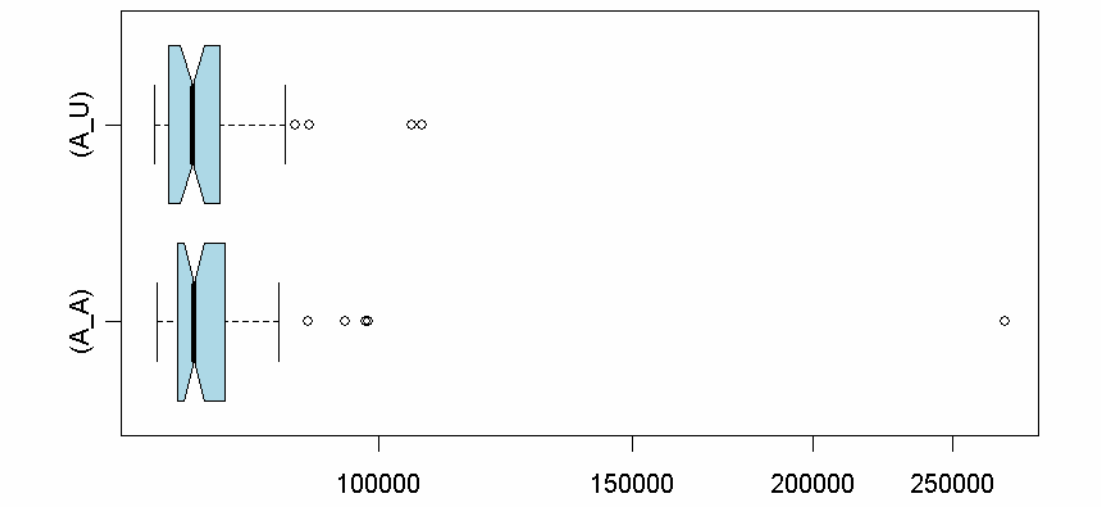
Median 74,270 µs (A_U):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT, COUY.IDENT, RN.IDENT
  FROM TSMT_REGION_IT RN
       INNER JOIN TSMT_COUNTRY COUY
             ON RN.IDENT = COUY.REGION_IT_ID
       INNER JOIN TSMT_COMPANY COMY
             ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_DETAILED_CHARGING DC
             ON DC.PERIOD = CAST ('01.01.2006' AS DATE)
            AND DC.SERVICE_RECEIVER_ID = SR.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.01          0        893          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.01          0        893          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
   1611  HASH JOIN  (cr=893 pr=0 pw=0 time=20435 us)
    831   HASH JOIN  (cr=41 pr=0 pw=0 time=6474 us)
    308    HASH JOIN  (cr=25 pr=0 pw=0 time=3279 us)
    250     NESTED LOOPS  (cr=9 pr=0 pw=0 time=2554 us)
    250      TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=300 us)
    250      INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=852 us)(object id
83971)
    308     TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=327 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=876 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=9082 us)
```

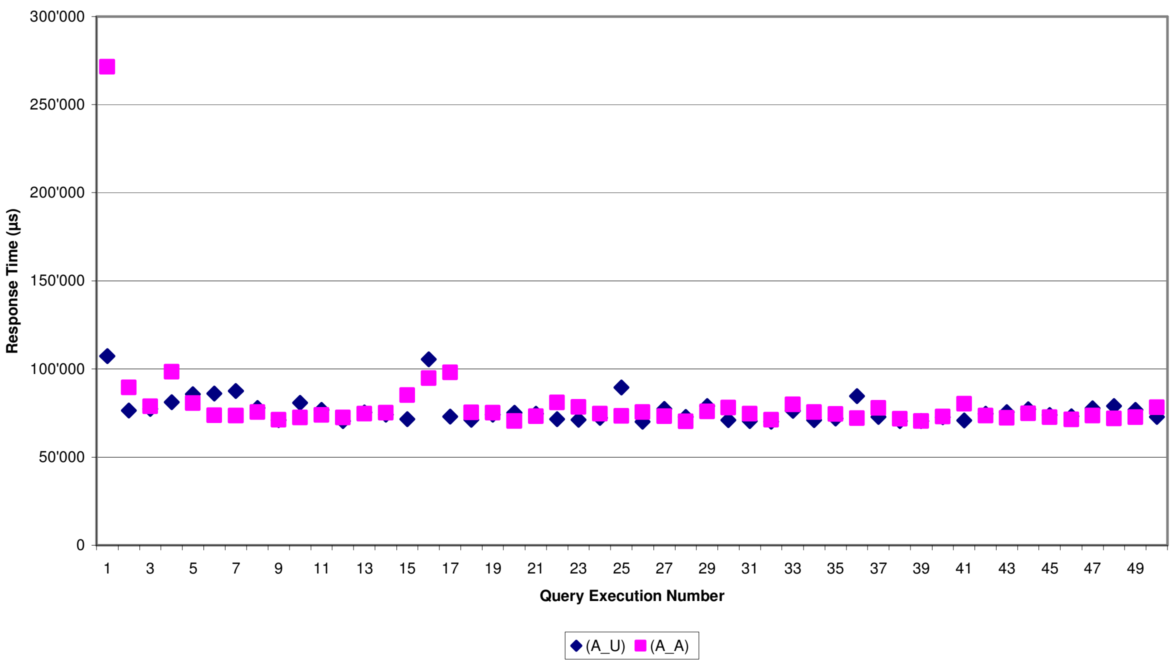
Median 74,490 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT, COMY.IDENT, COUY.IDENT, RN.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON DC.PERIOD = CAST ('01.01.2006' AS DATE)
            AND DC.SERVICE_RECEIVER_ID = SR.IDENT
       INNER JOIN TSMT_COMPANY COMY
             ON SR.COMPANY_CD = COMY.IDENT
       INNER JOIN TSMT_COUNTRY COUY
             ON COMY.COUNTRY_ID = COUY.IDENT
       INNER JOIN TSMT_REGION_IT RN
             ON COUY.REGION_IT_ID = RN.IDENT

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      162      0.01       0.01          0        893          0        1611
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      164      0.01       0.01          0        893          0        1611

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
```

```
-------   ---------------------------------------------------
   1611  HASH JOIN  (cr=893 pr=0 pw=0 time=20487 us)
    831   HASH JOIN  (cr=41 pr=0 pw=0 time=6427 us)
    308    HASH JOIN  (cr=25 pr=0 pw=0 time=3245 us)
    250     NESTED LOOPS  (cr=9 pr=0 pw=0 time=2560 us)
    250      TABLE ACCESS FULL TSMT_COUNTRY (cr=7 pr=0 pw=0 time=554 us)
    250      INDEX UNIQUE SCAN PK_T_SMT_RI (cr=2 pr=0 pw=0 time=838 us)(object id
83971)
    308     TABLE ACCESS FULL TSMT_COMPANY (cr=16 pr=0 pw=0 time=326 us)
    831    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=16 pr=0 pw=0 time=45 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=852 pr=0 pw=0 time=10771 us)
```



(Logarithmic scale)



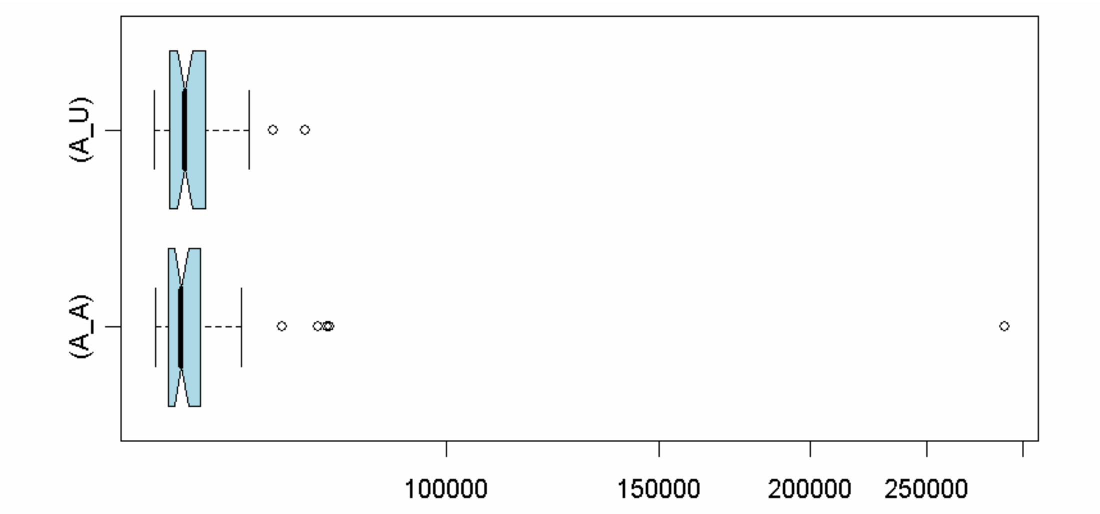
TQP_87231: A JOIN in the FROM clause may beat JOIN in the WHERE clause

Median 60,820 µs (A_U):

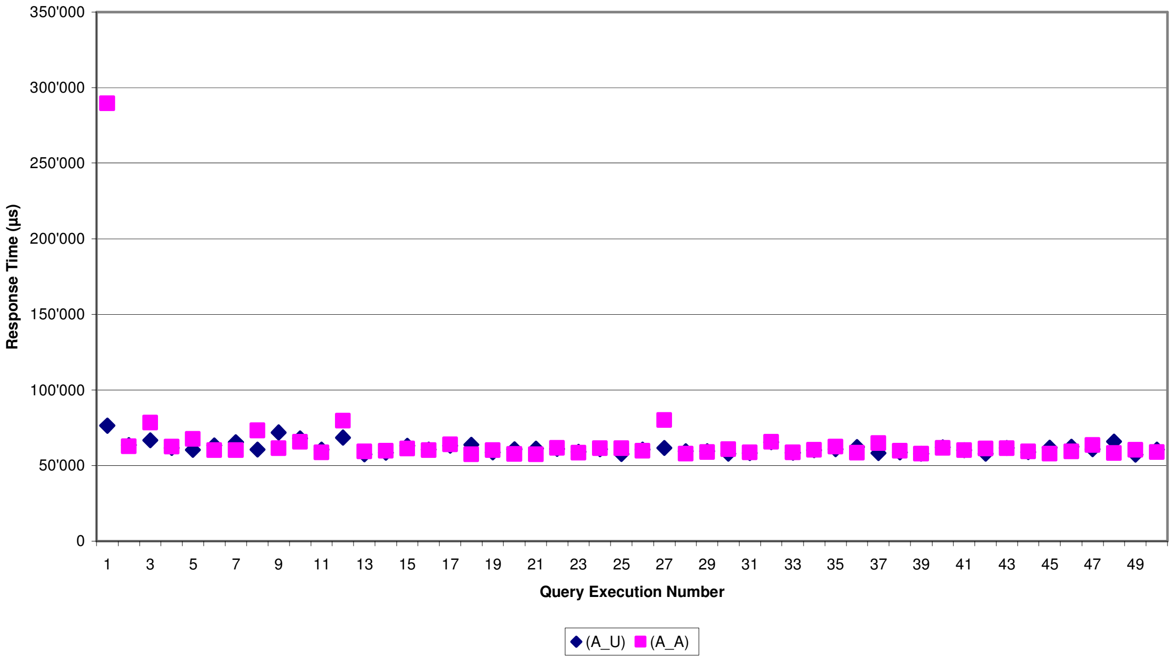
```
SELECT DC.IDENT, SR.IDENT
  FROM TSMT_DETAILED_CHARGING DC, TSMT_SERVICE_RECEIVER SR
 WHERE DC.PERIOD = CAST ('01.01.2006' AS DATE)
   AND DC.SERVICE_RECEIVER_ID = SR.IDENT
   AND SR.COMPANY_CD = '1201'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       82      0.03       0.03          0       2096          0         819
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       84      0.03       0.03          0       2096          0         819

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    819  TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=2096 pr=0 pw=0
time=4413 us)
  16907   NESTED LOOPS  (cr=578 pr=0 pw=0 time=17045 us)
    364    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=56 pr=0 pw=0 time=762 us)
  16542    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=522 pr=0 pw=0 time=18889
us)(object id 89036)
```

Median 60,400 µs (A_A):

```
SELECT DC.IDENT, SR.IDENT
  FROM TSMT_DETAILED_CHARGING DC
       INNER JOIN TSMT_SERVICE_RECEIVER SR
             ON DC.PERIOD = CAST ('01.01.2006' AS DATE)
            AND DC.SERVICE_RECEIVER_ID = SR.IDENT
            AND SR.COMPANY_CD = '1201'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       82      0.01       0.03          0       2096          0         819
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       84      0.01       0.03          0       2096          0         819

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
    819  TABLE ACCESS BY INDEX ROWID TSMT_DETAILED_CHARGING (cr=2096 pr=0 pw=0
time=3608 us)
  16907   NESTED LOOPS  (cr=578 pr=0 pw=0 time=17050 us)
    364    TABLE ACCESS FULL TSMT_SERVICE_RECEIVER (cr=56 pr=0 pw=0 time=399 us)
  16542    INDEX RANGE SCAN IDX_T_SMT_DC_SERVICE_RECEIVER (cr=522 pr=0 pw=0 time=18915
us)(object id 89036)
```



(Logarithmic scale)



WHERE Outperforms HAVING

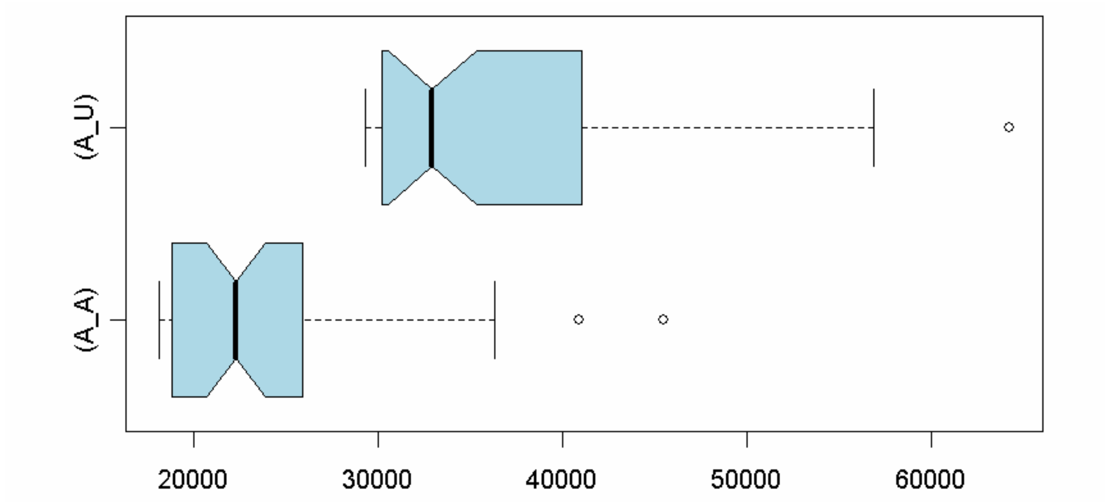
TQP_80801: The selection based on the WHERE clause happens before the selection based on the HAVING clause. Therefore the WHERE clause should always be chosen if the aggregated values are not affected by the selection. – Propagate the condition from the HAVING clause to the WHERE clause

Median 32,930 µs (A_U):

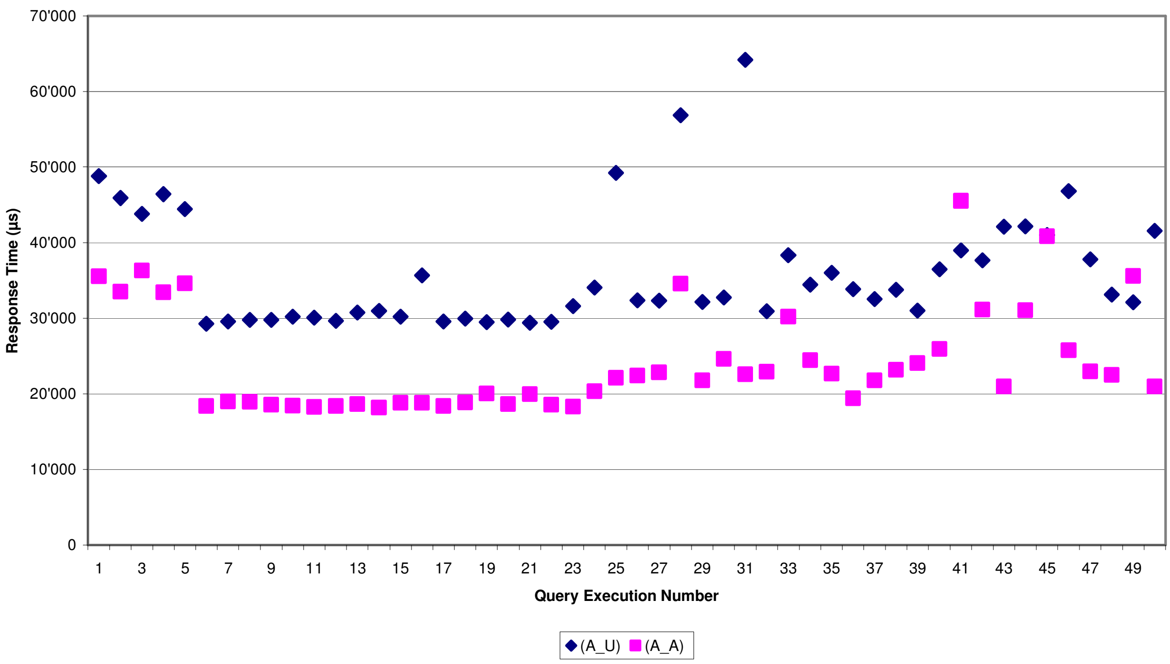
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 GROUP BY PERIOD, SIGN
HAVING PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.03       0.02          0        693          0           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.03       0.02          0        693          0           6

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      6  FILTER  (cr=693 pr=0 pw=0 time=26657 us)
     32   HASH GROUP BY (cr=693 pr=0 pw=0 time=26632 us)
  33183    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=33264 us)
```

Median 22,260 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD BETWEEN CAST ('01.01.2006' AS DATE)
                  AND CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD, SIGN

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           6

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      6  HASH GROUP BY (cr=693 pr=0 pw=0 time=15067 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13449 us)
```



(A_U)
(A_A)
20000
30000
40000
50000
60000



Response Time (µs)
70'000
60'000
50'000
40'000
30'000
20'000
10'000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Query Execution Number
(A_U) (A_A)

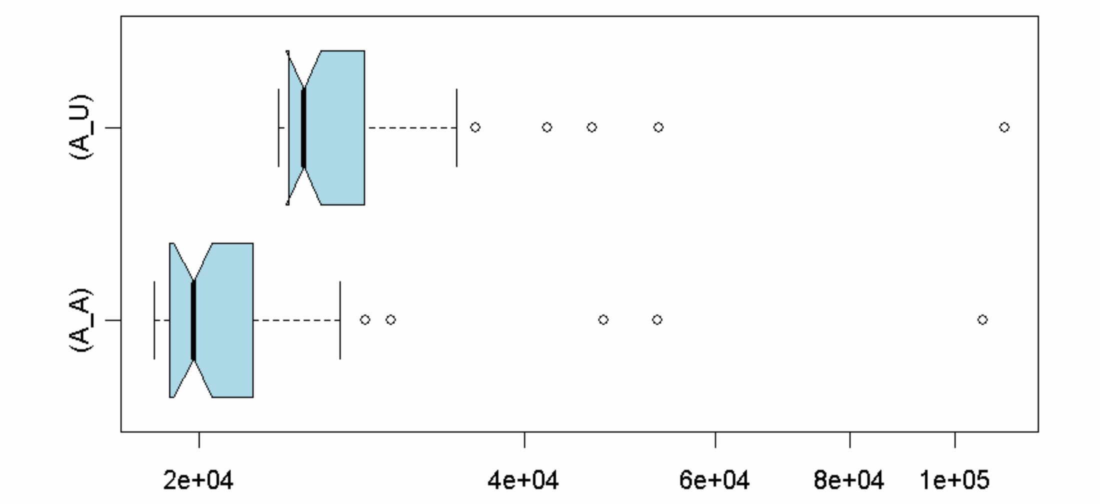
TQP_80802: The selection based on the `WHERE` clause happens before the selection based on the `HAVING` clause. Therefore the `WHERE` clause should always be chosen if the aggregated values are not affected by the selection. – Combine the condition from the `HAVING` clause and the `WHERE` clause

Median 24,980 µs (A_U):

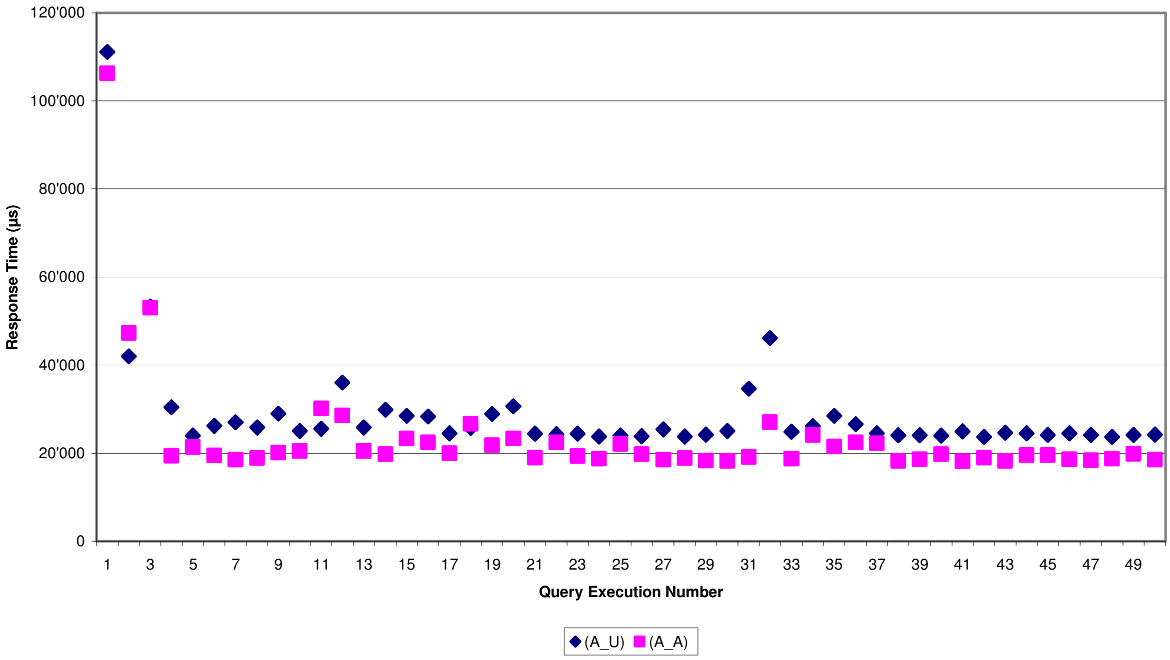
```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD >= CAST ('01.01.2006' AS DATE)
 GROUP BY PERIOD, SIGN
HAVING PERIOD <= CAST ('31.03.2006' AS DATE)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.03       0.02          0        693          0           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.03       0.02          0        693          0           6

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      6  FILTER  (cr=693 pr=0 pw=0 time=21563 us)
     20   HASH GROUP BY (cr=693 pr=0 pw=0 time=21529 us)
  16119    TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=16208 us)
```

Median 18,790 µs (A_A):

```
SELECT PERIOD, SIGN, SUM (QUANTITY)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD >= CAST ('01.01.2006' AS DATE)
   AND PERIOD <= CAST ('31.03.2006' AS DATE)
 GROUP BY PERIOD, SIGN

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0        693          0           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0        693          0           6

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
      6  HASH GROUP BY (cr=693 pr=0 pw=0 time=15648 us)
   4835   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=13666 us)
```



(Logarithmic scale)



Appendix C – Other Detailed Measurement Results

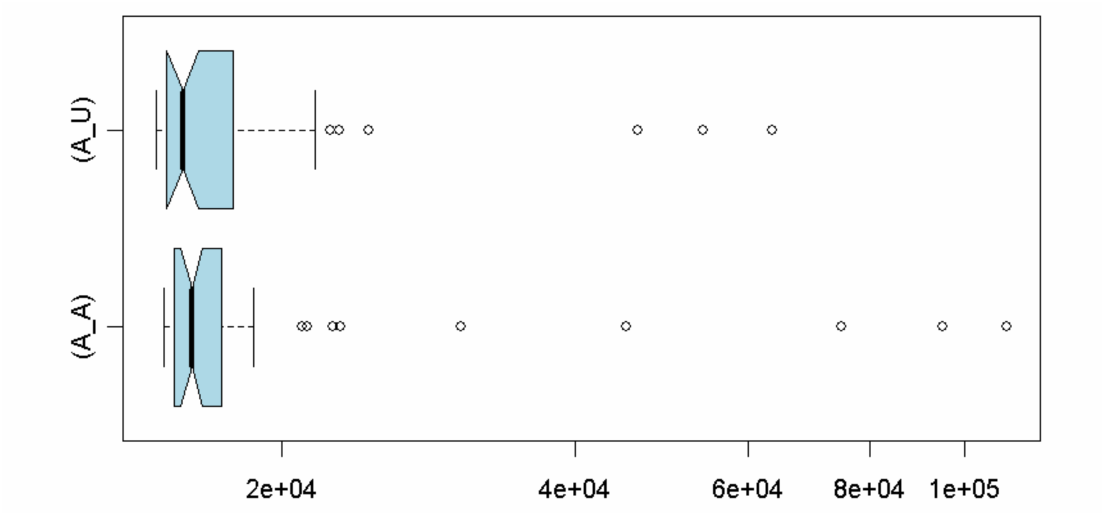
TQP_89901: Subquery in `FROM` clause

Median 15,870 µs (A_U):

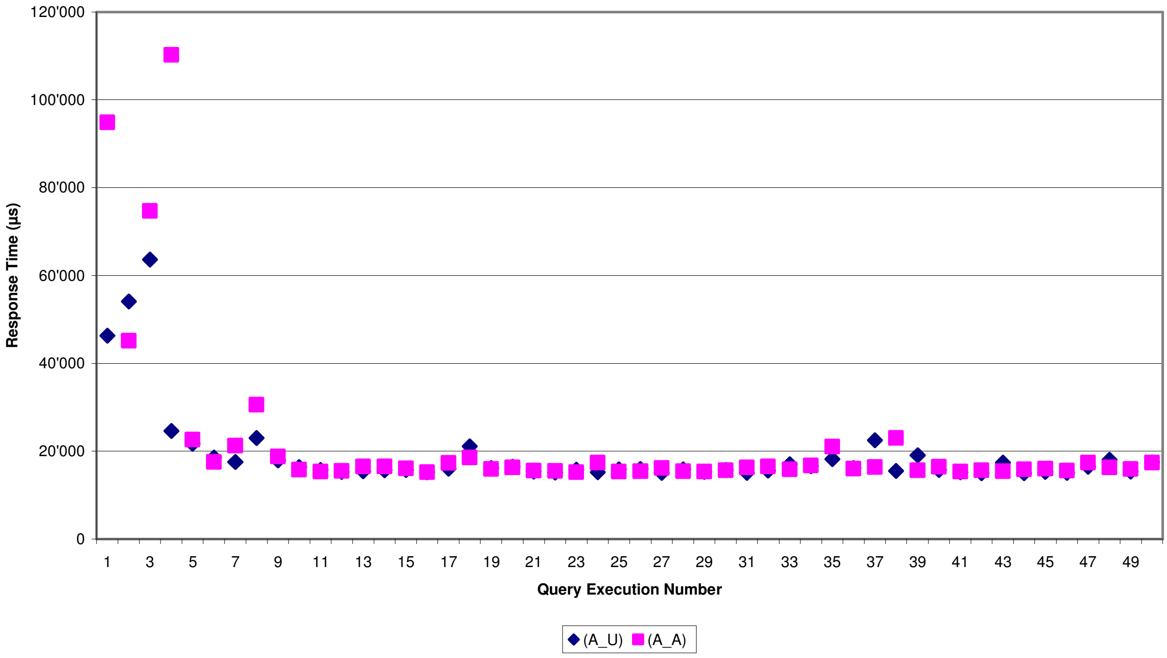
```
SELECT SERVICE_ID, COUNT (*)
  FROM TSMT_DETAILED_CHARGING
 WHERE PERIOD = CAST ('01.01.2006' AS DATE)
  GROUP BY SERVICE_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        6      0.01       0.01          0        693          0          59
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8      0.01       0.01          0        693          0          59

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     59  HASH GROUP BY (cr=693 pr=0 pw=0 time=11904 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=10421 us)
```

Median 16,200 µs (A_A):

```
SELECT SERVICE_ID, COUNT (*)
  FROM (SELECT SERVICE_ID
          FROM TSMT_DETAILED_CHARGING
         WHERE PERIOD = CAST ('01.01.2006' AS DATE))
 GROUP BY SERVICE_ID

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        6      0.01       0.01          0        693          0          59
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8      0.01       0.01          0        693          0          59

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 75

Rows     Row Source Operation
-------  ---------------------------------------------------
     59  HASH GROUP BY (cr=693 pr=0 pw=0 time=12638 us)
   1611   TABLE ACCESS FULL TSMT_DETAILED_CHARGING (cr=693 pr=0 pw=0 time=11444 us)
```



(Logarithmic scale)



Appendix D – Results Summary

<table>
<tr><th colspan="2">Pattern</th><th colspan="2">Test Query Pair (TQP)</th><th colspan="3">Pattern not applied</th><th colspan="3">Pattern applied</th><th></th><th></th><th colspan="3">Significant Improved</th></tr>
<tr><th>Id</th><th>Description</th><th>Id</th><th>Description</th><th>Hits</th><th>Median</th><th>Factor Slower</th><th>Hits</th><th>Median</th><th>Improved</th><th>Same Result</th><th>Access Plans</th><th>Box Plot</th><th>Dot Chart</th><th>Total</th></tr>
<tr><td>862</td><td>ALL / ANY or EXISTS</td><td>86201</td><td>A query of the form col1 comp ALL (SELECT col2 FROM ... WHERE cond) can be rewritten as NOT EXISTS (SELECT ... FROM ... WHERE cond AND col1 negatived_comp col2)</td><td>36</td><td>656'700</td><td>37.55</td><td>36</td><td>17'490</td><td>97%</td><td>yes</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
<tr><td>862</td><td>ALL / ANY or EXISTS</td><td>86211</td><td>A query of the form col1 comp ANY (SELECT col2 FROM ... WHERE cond) can be rewritten as EXISTS (SELECT ... FROM ... WHERE cond AND col1 negatived_comp col2)</td><td>16</td><td>692'200</td><td>37.72</td><td>16</td><td>18'350</td><td>97%</td><td>yes</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
<tr><td>861</td><td>ALL / ANY or MAX / MIN</td><td>86101</td><td>The operators < ANY or <= ANY may be replaced by < or <= along with a MAX function</td><td>1'611</td><td>77'150</td><td>1.01</td><td>1'611</td><td>76'100</td><td>1%</td><td>yes</td><td>different</td><td>no</td><td>no</td><td>no</td></tr>
<tr><td>861</td><td>ALL / ANY or MAX / MIN</td><td>86111</td><td>The operators < ALL or <= ALL may be replaced by < or <= along with a MIN function</td><td>61</td><td>28'000</td><td>1.01</td><td>61</td><td>27'830</td><td>1%</td><td>yes</td><td>different</td><td>no</td><td>no</td><td>no</td></tr>
<tr><td>861</td><td>ALL / ANY or MAX / MIN</td><td>86121</td><td>The operators > ANY or >= ANY may be replaced by > or >= along with a MIN function</td><td>1'611</td><td>75'540</td><td>0.99</td><td>1'611</td><td>76'360</td><td>-1%</td><td>yes</td><td>different</td><td>no</td><td>no</td><td>no</td></tr>
<tr><td>861</td><td>ALL / ANY or MAX / MIN</td><td>86131</td><td>The operators > ALL or >= ALL may be replaced by > or >= along with a MAX function</td><td>61</td><td>27'800</td><td>1.03</td><td>61</td><td>26'920</td><td>3%</td><td>yes</td><td>different</td><td>yes</td><td>no</td><td>no</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80101</td><td>Within a query use SELECT [ALL | DISTINCT] - Projection of a non-indexed column</td><td>1'774</td><td>44'590</td><td>1.33</td><td>33'183</td><td>33'640</td><td>25%</td><td>n/a</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80102</td><td>Within a query use SELECT [ALL | DISTINCT]: Projection of an indexed column</td><td>33'183</td><td>1'122'000</td><td>1.00</td><td>33'183</td><td>1'118'000</td><td>0%</td><td>yes</td><td>different</td><td>no</td><td>no</td><td>no</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80103</td><td>Within a query use SELECT [ALL | DISTINCT]: Projection of all columns</td><td>33'183</td><td>2'057'000</td><td>1.26</td><td>33'183</td><td>1'638'000</td><td>20%</td><td>n/a</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80104</td><td>Within a query use SELECT [ALL | DISTINCT]: Projection of a non-indexed column – bigger table</td><td>11'605</td><td>40'170</td><td>1.17</td><td>102'583</td><td>34'420</td><td>14%</td><td>n/a</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80105</td><td>Within a query use SELECT [ALL | DISTINCT]: Projection of a non-indexed column – GROUP BY instead of DISTINCT</td><td>1'774</td><td>78'870</td><td>1.00</td><td>1'774</td><td>79'070</td><td>0%</td><td>yes</td><td>different</td><td>no</td><td>no</td><td>no</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80111</td><td>Within certain aggregate functions use <aggregate function> ([ALL | DISTINCT] <value expression>): Function AVG</td><td>1</td><td>39'590</td><td>2.68</td><td>1</td><td>14'790</td><td>63%</td><td>n/a</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
<tr><td>801,802</td><td>All Rows (801) / Distinct Rows (802)</td><td>80112</td><td>Within certain aggregate functions use <aggregate function> ([ALL | DISTINCT] <value expression>): Function COUNT</td><td>1</td><td>20'300</td><td>2.67</td><td>1</td><td>7'610</td><td>63%</td><td>n/a</td><td>different</td><td>yes</td><td>yes</td><td>yes</td></tr>
</table>

Pattern		Test Query Pair (TQP)		Pattern not applied			Pattern applied			Same Result	Access Plans	Significant Improved		
Id	Description	Id	Description	Hits	Median	Factor Slower	Hits	Median	Improved			Box Plot	Dot Chart	Total
801,802	All Rows (801) / Distinct Rows (802)	80113	Within certain aggregate functions use <aggregate function> ([ALL | DISTINCT] <value expression>): Function MAX	1	13'800	1.00	1	13'740	0%	n/a	identical	no	no	no
801,802	All Rows (801) / Distinct Rows (802)	80114	Within certain aggregate functions use <aggregate function> ([ALL | DISTINCT] <value expression>): Function MIN	1	13'900	1.00	1	13'840	0%	n/a	identical	no	no	no
801,802	All Rows (801) / Distinct Rows (802)	80115	Within certain aggregate functions use <aggregate function> ([ALL | DISTINCT] <value expression>): Function SUM	1	39'810	2.70	1	14'720	63%	n/a	different	yes	yes	yes
801,802	All Rows (801) / Distinct Rows (802)	80121	Use the set operator UNION ALL instead of UNION: Projection of a non-indexed column	13'379	55'230	1.51	135'766	36'470	34%	n/a	different	yes	yes	yes
801,802	All Rows (801) / Distinct Rows (802)	80122	Use the set operator UNION ALL instead of UNION: Projection of an indexed column	135'766	5'034'000	0.98	135'766	5'119'000	-2%	yes	different	no	no	no
801,802	All Rows (801) / Distinct Rows (802)	80123	UNION: SELECT DISTINCT instead of UNION	13'379	732'800	1.16	13'379	632'000	14%	yes	different	yes	yes	yes
809	Avoid Explicit Sorting	80901	A DISTINCT clause matching an index may cover sorting requirements	33'183	1'168'000	1.06	33'183	1'100'000	6%	no	different	yes	yes	no
809	Avoid Explicit Sorting	80902	A DISTINCT clause matching an index may cover sorting requirements - indexed column and primary key	131	1'150'000	1.03	131	1'118'000	3%	no	different	yes	yes	no
809	Avoid Explicit Sorting	80903	A DISTINCT clause matching an index may cover sorting requirements - indexed column	33'183	16'920	0.88	33'183	19'320	-14%	no	different	worse	worse	no
809	Avoid Explicit Sorting	80911	A GROUP BY clause matching an index may cover sorting requirements	33'183	1'177'000	1.06	33'183	1'108'000	6%	no	different	yes	yes	no
809	Avoid Explicit Sorting	80912	A GROUP BY clause matching an index may cover sorting requirements - indexed column and primary key	33'183	1'156'000	1.03	33'183	1'120'000	3%	no	different	yes	yes	no
809	Avoid Explicit Sorting	80913	A GROUP BY clause matching an index may cover sorting requirements - indexed column	131	17'490	0.87	131	19'990	-14%	no	different	worse	worse	no
874	Complete JOIN Selection	87401	Maximising the selection information in the JOIN and WHERE clauses (even if this appears redundant) may support the query optimiser in improving the access plans and in reducing the number of iterations of nested loops - WHERE or JOIN	1'611	77'650	1.00	1'611	77'960	0%	yes	identical	no	no	no

Pattern		Test Query Pair (TQP)		Pattern not applied			Pattern applied					Significant Improved		
Id	Description	Id	Description	Hits	Median	Factor Slower	Hits	Median	Improved	Same Result	Access Plans	Box Plot	Dot Chart	Total
874	Complete JOIN Selection	87402	Maximising the selection information in the JOIN and WHERE clauses (even if this appears redundant) may support the query optimiser in improving the access plans and in reducing the number of iterations of nested loops - maximum redundancy	1'611	78'490	0.97	1'611	81'240	-4%	yes	identical	no	no	no
803	Defensive Projection	80301	Be specific and avoid the asterisk in the SELECT clause	102'583	5'153'000	1.43	102'583	3'599'000	30%	n/a	identical	yes	yes	yes
803	Defensive Projection	80311	If possible restrict the columns in the SELECT clause to the indexed ones	102'583	3'892'000	1.04	102'583	3'726'000	4%	n/a	different	yes	yes	yes
803	Defensive Projection	80321	If possible restrict the columns in the SELECT clause to the indexed ones: the order of the columns is irrelevant	102'583	3'747'000	0.99	102'583	3'776'000	-1%	n/a	identical	no	no	no
864	EXISTS or COUNT	86401	A query of the form WHERE EXISTS (SELECT col1 FROM ...) can be rewritten as 0 < (SELECT COUNT(*) FROM ...)	59	16'870	0.98	59	17'300	-3%	yes	identical	no	no	no
864	EXISTS or COUNT	86411	A query of the form WHERE NOT EXISTS (SELECT col1 FROM ...) can be rewritten as 0 = (SELECT COUNT(*) FROM ...)	36	16'340	0.99	36	16'470	-1%	yes	identical	no	no	no
863	EXISTS or IN	86301	If the outer table has many rows and the inner table has few rows, then use IN	7'964	231'600	1.00	7'964	232'000	0%	yes	different	no	no	no
863	EXISTS or IN	86302	if the outer table has many rows and the inner table has few rows, then use IN – indexed column	912	28'290	0.99	912	28'520	-1%	yes	different	no	no	no
863	EXISTS or IN	86303	If most of the rows are filtered by the outer query, then use EXISTS	416	25'060	0.97	416	25'850	-3%	yes	different	no	no	no
863	EXISTS or IN	86304	If most of the rows are filtered by the outer query, then use EXISTS – indexed column	90	7'480	0.98	90	7'626	-2%	yes	different	no	no	no
863	EXISTS or IN	86305	If most of the rows are filtered by the inner query, then use IN	7	11'980	1.00	7	12'020	0%	yes	identical	no	no	no
863	EXISTS or IN	86306	If most of the rows are filtered by the inner query, then use IN – indexed column	4	3'606	1.02	4	3'538	2%	yes	identical	no	no	no
863	EXISTS or IN	86307	If the outer query is of format "WHERE NOT ...", then use NOT EXISTS	1'195	49'660	1.00	1'195	49'590	0%	yes	identical	no	no	no
863	EXISTS or IN	86308	If the outer query is of format "WHERE NOT ...", then use NOT EXISTS – indexed column	1'521	60'820	0.94	1'521	64'440	-6%	yes	identical	no	no	no

Pattern		Test Query Pair (TQP)		Pattern not applied			Pattern applied					Significant Improved		
Id	Description	Id	Description	Hits	Median	Factor Slower	Hits	Median	Improved	Same Result	Access Plans	Box Plot	Dot Chart	Total
863	EXISTS or IN	86309	Order the list of values of an IN predicate by frequency of probable usage	101'962	4'579'000	1.03	101'962	4'459'000	3%	yes	different	no	no	no
863	EXISTS or IN	86310	Try to avoid duplicates in the list of values of an IN predicate – applying DISTINCT	101'962	3'794'000	1.01	101'962	3'756'000	1%	yes	identical	no	no	no
863	EXISTS or IN	86311	Try to avoid duplicates in the list of values of an IN predicate	101'962	3'546'000	0.99	101'962	3'571'000	-1%	yes	different	no	no	no
863	EXISTS or IN	86312	Order the list of values of an IN predicate by frequency of probable usage versus try to avoid duplicates in the list of values of an IN predicate	101'962	3'957'000	1.06	101'962	3'730'000	6%	yes	identical	no	no	no
805	GROUPING SETS beats UNION	80501	The use of a GROUPING SETS clause instead of the application of a UNION should result in a significant performance improvement - CUBE	12	61'380	3.39	12	18'120	70%	yes	different	yes	yes	yes
805	GROUPING SETS beats UNION	80502	The use of a GROUPING SETS clause instead of the application of a UNION should result in a significant performance improvement – GROUPING SETS	12	63'130	2.36	12	26'710	58%	yes	different	yes	yes	yes
805	GROUPING SETS beats UNION	80503	The use of a GROUPING SETS clause instead of the application of a UNION should result in a significant performance improvement - ROLLUP	10	46'250	2.60	10	17'770	62%	yes	different	yes	yes	yes
805	GROUPING SETS beats UNION	80511	CUBE and ROLLUP are shortcuts that can be used to simplify the application of certain variants of the GROUPING SETS clause - CUBE	12	28'310	1.44	12	19'660	31%	yes	different	yes	yes	yes
805	GROUPING SETS beats UNION	80512	CUBE and ROLLUP are shortcuts that can be used to simplify the application of certain variants of the GROUPING SETS clause - ROLLUP	10	17'040	1.00	10	17'050	0%	yes	identical	no	no	no
804	Index Exploitation	80401	Don't apply functions to indexed columns – result set cardinality 1 of 1,945,431 rows	1	1'115'700	411.39	1	2'712	100%	n/a	different	yes	yes	yes
804	Index Exploitation	80402	Don't apply functions to indexed columns – result set cardinality 20% of 1,945,431 rows	390'428	18'090'000	1.04	390'428	17'400'000	4%	n/a	different	yes	yes	yes
804	Index Exploitation	80411	Move function calls to the non-indexed components	1	1'114'000	442.94	1	2'515	100%	n/a	different	yes	yes	yes
804	Index Exploitation	80421	Avoid implicit applied functions like type conversions	1	1'127'000	399.36	1	2'822	100%	n/a	different	yes	yes	yes

Pattern		Test Query Pair (TQP)		Pattern not applied			Pattern applied					Significant Improved		
Id	Description	Id	Description	Hits	Median	Factor Slower	Hits	Median	Improved	Same Result	Access Plans	Box Plot	Dot Chart	Total
804	Index Exploitation	80431	Avoid leading wildcards - comparing % and _ (result set cardinality 1 of 1,945,431 rows)	1	770'900	1.01	1	763'500	1%	n/a	identical	yes	no	no
804	Index Exploitation	80432	Avoid leading wildcards - prefer selection with range if possible	138'908	6'189'000	1.01	138'908	6'125'000	1%	n/a	identical	no	no	no
804	Index Exploitation	80433	Avoid leading wildcards - comparing % and _ (result set cardinality 20% of 1,945'431 rows)	390'428	13'450'000	1.00	390'428	13'480'000	0%	n/a	identical	no	no	no
873	JOIN and DISTINCT	87301	If the projection of a JOIN contains a unique key of one table and the key columns of all other tables are joined by an equi-join, then the DISTINCT clause in the projection is unnecessary	1'611	70'930	1.00	1'611	70'840	0%	yes	different	no	no	no
807	Minimise Grouping Columns	80701	Keep the number of grouping columns small - combine columns	6	18'600	0.83	6	22'310	-20%	n/a	identical	worse	worse	no
807	Minimise Grouping Columns	80702	Keep the number of grouping columns small - eliminate superfluous columns	3	20'390	1.01	3	20'270	1%	yes	identical	no	no	no
810	Minimise Sorting Columns	81001	Minimise the number of columns (expressions) in the ORDER BY clause - combine columns	102'583	3'791'000	1.01	102'583	3'768'000	1%	yes	identical	no	no	no
810	Minimise Sorting Columns	81002	Minimise the number of columns (expressions) in the ORDER BY clause - combine columns	102'583	3'850'000	0.93	102'583	4'128'000	-7%	n/a	identical	worse	worse	no
810	Minimise Sorting Columns	81003	Minimise the number of columns (expressions) in the ORDER BY clause – eliminate superfluous columns	4'392	193'700	1.00	4'392	193'200	0%	yes	identical	no	no	no
810	Minimise Sorting Columns	81011	Minimise the number of columns to be projected	102'853	6'415'000	1.67	102'583	3'845'000	40%	n/a	identical	yes	yes	yes
999	No Pattern	99901	Subquery in FROM clause	59	15'870	0.98	59	16'200	-2%	yes	identical	no	no	no
872	Ordering INNER JOIN	87201	The smallest table should be the inner table	33'183	1'175'000	1.01	33'183	1'164'000	1%	yes	identical	no	no	no
872	Ordering INNER JOIN	87211	The table with the best index should be the inner table	33'183	1'150'000	0.99	33'183	1'156'000	-1%	yes	identical	no	no	no
872	Ordering INNER JOIN	87212	The table with the best index should be the inner table	33'183	1'159'000	1.01	33'183	1'146'000	1%	yes	identical	no	no	no
872	Ordering INNER JOIN	87213	The table with the best index should be the inner table	33'183	1'154'000	1.00	33'183	1'149'000	0%	yes	identical	no	no	no
872	Ordering INNER JOIN	87214	The table with the best index should be the inner table	33'183	1'146'000	1.00	33'183	1'149'000	0%	yes	identical	no	no	no

Pattern		Test Query Pair (TQP)		Pattern not applied			Pattern applied					Significant Improved		
Id	Description	Id	Description	Hits	Median	Factor Slower	Hits	Median	Improved	Same Result	Access Plans	Box Plot	Dot Chart	Total
872	Ordering INNER JOIN	87215	The table with the best index should be the inner table	33'183	1'155'000	1.00	33'183	1'155'000	0%	yes	identical	no	no	no
872	Ordering INNER JOIN	87216	The table with the best index should be the inner table	33'183	1'153'000	1.00	33'183	1'157'000	0%	yes	identical	no	no	no
872	Ordering INNER JOIN	87221	The table with the most restrictive clause should be the outer table - selection of a small table	1'909	93'600	0.99	1'909	94'600	-1%	yes	identical	no	no	no
872	Ordering INNER JOIN	87222	The table with the most restrictive clause should be the outer table - selection of a big table	1'611	74'270	1.00	1'611	74'490	0%	yes	identical	no	no	no
872	Ordering INNER JOIN	87231	A JOIN in the FROM clause may beat JOIN in the WHERE clause	819	60'820	1.01	819	60'400	1%	yes	identical	no	no	no
808	WHERE Outperforms HAVING	80801	The selection based on the WHERE clause happens before the selection based on the HAVING clause. Therefore the WHERE clause should always be chosen if the aggregated values are not affected by the selection	6	32'930	1.48	6	22'260	32%	yes	different	yes	yes	yes
808	WHERE Outperforms HAVING	80802	The selection based on the WHERE clause happens before the selection based on the HAVING clause. Therefore the WHERE clause should always be chosen if the aggregated values are not affected by the selection	6	24'980	1.33	6	18'790	25%	yes	different	yes	yes	yes

Appendix E – REPSI Tool: Documentation and Software

Please see enclosed CD-ROM or link http://developer.berlios.de/projects/repsi/

(Version 2.00) at the Web

References

Abiteboul, S., Hull, R. and Vianu, V. (1995) *Foundations of Databases,* Addison-Wesley, Reading, MA, USA.

Alexander, C. (1979) *The timeless way of building,* Oxford University Press, New York, USA.

Alexander, C., Ishikawa, S. and Silverstein, M. (1977) *A pattern language : towns, buildings, construction,* Oxford University Press, New York, USA.

Appleton, B. (2000) *Patterns and software: essential concepts and terminology*. Available from: http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html [Accessed 24.10.2006].

Arnout, K. and Meyer, B. (2004) *From Patterns to Components: The Factory Library Example*. Available from: http://se.inf.ethz.ch/people/arnout/ongoing/arnout_meyer_factory.pdf [Accessed 20.10.2006].

Avnur, R. and Hellerstein, J. M. (2000) 'Eddies: continuously adaptive query processing', *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* Dallas, TX, USA2000. ACM Press New York, NY, USA pp. 261-272

Barabanov, M. (1997) *A linux-based real-time operating system*, New Mexico Institute of Mining and Technology. Available from: http://www.ee.ryerson.ca/~courses/ee8205/projects/RT-Linux-Report.pdf [Accessed 15.10.2006].

Beck, K. (1997) *Smalltalk best practice patterns,* Prentice Hall, Upper Saddle River, NJ, USA.

Bhargava, G., Goel, P. and Iyer , B. R. (1995) 'Simplification of outer joins', *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research* Toronto, Ontario, Canada 1995. IBM Press, pp. 1-13.

Brosky, S. and Rotolo, S. (2003) *Shielded processors: guaranteeing sub-millisecond response in standard Linux*. Available from: http://www.ccur.com/isddocs/wp-shielded-cpu.pdf [Accessed 15.10.2006].

Burleson, D. K. (2001) *Oracle high-performance SQL tuning,* (1st Edn), McGraw-Hill, Berkeley, CA, USA.

Burleson, D. K. and Gogala, M. (2005) *Oracle tuning : the definitive reference,* Rampant TechPress, Kittrell, NC, USA.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. (1996) *Pattern-oriented software architecture : a system of patterns,* Wiley, New York, USA.

Celis, P. and Zeller, H. (1997) 'Subquery elimination: A complete unnesting algorithm for an extended relational algebra', *Proceedings of the 1997 IEEE 13th International Conference on Data Engineering,* Birmingham, England1997. IEEE, Los Alamitos, USA, pp. 321-321.

Celko, J. (2005a) *Joe Celko's SQL for smarties : advanced SQL programming,* (3rd Edn), Morgan Kaufmann, Boston, MA, USA.

Celko, J. (2005b) *Joe Celko's SQL programming style,* Morgan Kaufmann, Amsterdam, Netherlands.

Chakravarthy, U. S., Grant, J. and Minker, J. (1990) 'Logic-based approach to semantic query optimization', *ACM Transactions on Database Systems (TODS),* 15 (2), pp. 162-207.

Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., Lorie, R., Mehl, J. W., Price, T. G., Putzolu, F., Selinger, P. G., Schkolnick, M., Slutz, D. R., Traiger, I. L. and Wade, B. W. (1981) 'HISTORY AND EVALUATION OF SYSTEM R', *Communications of the ACM,* 24 (10), pp. 632-646.

Chan, I. (2005) *Oracle Database Performance Tuning Guide 10g Release 2 (10.2)*, Oracle Corporation. Available from: http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14211.pdf [Accessed 27.08.2006].

Chaudhuri, S. (1998) 'An overview of query optimization in relational systems', *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems,* Seattle, WA, USA1998. ACM Press New York, NY, USA pp. 34-43.

Chaudhuri, S. and Vardi, M. Y. (1993) 'Optimization of real conjunctive queries', *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* Washington, D.C., USA1993. ACM Press New York, NY, USA, pp. 59-70.

Cherniack, M. and Zdonik, S. B. (1996) 'Rule languages and internal algebras for rule-based optimizers', *Proceedings of the 1996 ACM SIGMOD international conference on Management of data,* Montreal, Quebec, Canada1996. ACM Press New York, NY, USA, pp. 401-412.

Chu, F., Halpern, J. Y. and Seshadri, P. (1999) 'Least expected cost query optimization: an exercise in utility', *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* Philadelphia, PA, USA 1999. ACM Press New York, NY, USA pp. 138-147

Cline, M. P. (1996) 'The pros and cons of adopting and applying design patterns in the real world', *Communications of the ACM,* 39 (10), pp. 47-49.

Codd, E. F. (1970) 'A relational model of data for large shared data banks', *Communications of the ACM,* 13 (6), pp. 377-387.

Coplien, J. O. (1992) *Advanced C++ programming styles and idioms,* Addison-Wesley, Reading, MA, USA.

Coplien, J. O. (1997) 'Idioms and patterns as architectural literature', *IEEE Software,* 14 (1), pp. 36-42.

Cumming, A. and Russell, G. (2007) *SQL Hacks : Tips and Tools for Digging into Your Data,* O'Reilly, Sebastopol, CA, USA.

Cunningham, W. and Beck, K. (1987) *Using pattern languages for object-oriented programs*. Available from: http://c2.com/doc/oopsla87.html [Accessed 30.10.2006].

Cyran, M. (2005) *Oracle Database Concepts, 10g Release 2 (10.2)*, Oracle Corporation. Available from: http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14220.pdf [Accessed 17.10.2006].

Date, C. J. and Darwen, H. (1997) *A guide to the SQL standard : a user's guide to the standard database language SQL,* (4th Edn), Addison-Wesley, Reading, MA, USA.

Dayal, U. (1987) 'Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers', *Proceedings of the 13th International Conference on Very Large Data Bases*1987. pp. 197-208.

Derntl, M. (2003) *The Pattern Approach – Capturing Proven Solutions with Special Focus on Architecture, Software Engineering, and Pedagogy*, University of Vienna. Available from: http://www.pri.univie.ac.at/~derntl/papers/LitSE-Patterns.pdf [Accessed 21.03.2006].

Dietrich, S. W. (2001) *Understanding Relational Database Query Languages,* Prentice Hall, Upper Saddle River, NJ, USA.

Donahoo, M. J. and Speegle, G. D. (2005) *SQL : practical guide for developers,* Morgan Kaufmann, Amsterdam, Netherlands.

Elmasri, R. and Navathe, S. (2007) *Fundamentals of database systems,* (5th Edn), Pearson/Addison Wesley, Boston, MA, USA.

Evans Data Corporation (2007) *Open Source Databases Currently Used.* Available from: http://www.evansdata.com/n2/surveys/database/2005_1/db_05_1_xmp1.shtml [Accessed 11.02.2007].

Faroult, S. and Robson, P. (2006) *The Art of SQL,* (1st Edn), O'Reilly, Sebastopol, CA, USA.

Ferdinandi, P. L. (2002) *A requirements pattern : succeeding in the Internet economy,* Addison-Wesley, Boston, MA, USA.

Floss, K. (2004) *Oracle SQL tuning & CBO internals,* (1st Edn), Rampant TechPress, Kittrell, NC, USA.

Fowler, M. (1997) *Analysis patterns : reusable object models,* Addison-Wesley, Menlo Park, CA, USA.

Fowler, M. and Beck, K. (1999) *Refactoring : improving the design of existing code,* Addison-Wesley, Reading, MA, USA.

Freytag, J. C. (1987) 'A rule-based view of query optimization', *Proceedings of the 1987 ACM SIGMOD international conference on Management of data,* San Francisco, CA, USA1987. ACM Press New York, NY, USA pp. 173-180.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design patterns : elements of reusable object-oriented software,* Addison-Wesley, Reading, MA, USA.

Ganski, R. A. and Wong, H. K. T. (1987) 'Optimization of nested SQL queries revisited', *Proceedings of the 1987 ACM SIGMOD international conference on Management of data,* San Francisco, CA, USA 1987. ACM Press New York, NY, USA pp. 23-33.

Garcia-Molina, H., Ullman, J. D. and Widom, J. (2002) *Database Systems : The Complete Book,* (International Edn), Prentice Hall, Upper Saddle River, NJ, USA.

Gartner (2002) *Gartner Dataquest Says Worldwide DBMS Industry Experienced Flat Growth in 2001*. Available from: http://www.gartner.com/5_about/press_releases/2002_05/pr20020507a.jsp [Accessed 13.03.2006].

Gartner (2007) *Gartner Says Worldwide Relational Database Market Increased 8 Percent in 2005*. Available from: http://www.gartner.com/press_releases/asset_152619_11.html [Accessed 08.02.2007].

Goldstein, J. and Larson, P. A. (2001) 'Optimizing queries using materialized views: A practical, scalable solution', Santa Barbara, CA, USA2001. ACM Press New York, NY, USA pp. 331-342.

Graefe, G. (1993) 'Query evaluation techniques for large databases', *ACM Computing Surveys,* 25 (2), pp. 73-170.

Graefe, G. (1995) 'The Cascades Framework for Query Optimization', *Bulletin of the Technical Committee on Data Engineering,* 18 (3), pp. 19-28.

Graefe, G. and DeWitt, D. J. (1987) 'The EXODUS optimizer generator', *Proceedings of the 1987 ACM SIGMOD international conference on Management of data* San Francisco, CA, USA1987. ACM Press New York, NY, USA pp. 160-172

Graefe, G. and McKenna, W. J. (1993) 'Volcano optimizer generator: Extensibility and efficient search', Vienna, Austria1993. Publ by IEEE, Los Alamitos, CA, USA, pp. 209-218.

Grand, M. (2001) *Java Enterprise design patterns,* Wiley, New York, USA.

Grier, D. A. (1992) 'Graphical techniques for output analysis', *Proceedings of the 24th conference on Winter simulation* Arlington, VA, USA1992. ACM Press New York, NY, USA pp. 314-319

Gulutzan, P. and Pelzer, T. (2003) *SQL performance tuning,* Addison-Wesley, Boston, MA, USA.

Haas, F. (2005) *Oracle Tuning in der Praxis : Rezepte und Anleitungen für Datenbankadministratoren und -entwickler,* Carl Hanser Verlag, München, Germany.

Haas, L. M., Freytag, J. C., Lohman, G. M. and Pirahesh, H. (1989) 'Extensible query processing in starburst', *Proceedings of the 1989 ACM SIGMOD international conference on Management of data* Portland, OR, USA 1989. ACM Press New York, NY, USA pp. 377-388

IBM (2004) *IBM® DB2 Universal Database: Administration Guide: Performance Version 8.2*, International Business Machines Corporation. Available from: ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2d3e81.pdf [Accessed 19.02.2007].

Ioannidis, Y. E. (1996) 'Query optimization', *ACM Computing Surveys,* 28 (1), pp. 121-123.

ISO/IEC JTC 1/SC 32/WG 3 (2003) *Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation) [International standard - working draft under consideration - 22.07.2005]*. Available from: http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=34133&ICS1=35&ICS2=60&ICS3= [Accessed 14.03.2006].

Jarke, M. and Koch, J. (1984) 'Query optimization in database systems', *ACM Computing Surveys (CSUR),* 16 (2), pp. 111-152.

Kim, W. (1982) 'On Optimizing an SQL-like Nested Query', *ACM Transactions on Database Systems (TODS),* 7 (3), pp. 443-469.

Kline, K. E., Kline, D. and Hunt, B. (2004) *SQL in a nutshell : a desktop quick reference,* (2nd Edn), O'Reilly, Sebastopol, CA, USA.

Kossmann, D. and Stocker, K. (2000) 'Iterative Dynamic Programming: A

New Class of Query Optimization Algorithms', *ACM Transactions on Database Systems,* 25 (1), pp. 43-82.

Langr, J. (2000) *Essential Java style : patterns for implementation,* Prentice Hall, Upper Saddle River, NJ, USA.

Larman, C. (2002) *Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process,* (2nd Edn), Prentice Hall, Upper Saddle River, NJ, USA.

Lea, D. (1994) 'Christopher Alexander: an introduction for object-oriented designers', *ACM SIGSOFT Software Engineering Notes* 19 (1), pp. 39-46

Lorentz, D. (2005) *Oracle Database SQL Reference 10g Release 2 (10.2)*, Oracle Corporation. Available from: http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14200.pdf [Accessed 23.09.2006].

McGill, R., Tukey, J. W. and Larsen, W. A. (1978) 'Variations of Box Plots', *The American Statistician,* 32 (1), pp. 12-16.

Melton, J. and Simon, A. R. (2002) *SQL:1999 : Understanding Relational Language Components,* Morgan Kaufmann, San Francisco, CA, USA.

Meszaros, G. and Doble, J. (2005) *A Pattern Language for Pattern Writing*, Hillside Group Available from: http://hillside.net/patterns/writing/patterns.htm#1.0 [Accessed 07.04.2006].

Meyer, B. and Arnout, K. (2006) *Componentization: the Visitor example*. Available from: http://se.ethz.ch/~meyer/publications/computer/visitor.pdf [Accessed 20.10.2006].

Moerkotte, G. (2006) *Building Query Compilers (Under Construction) [expected time to completion: 5 years]*. Available from: http://pi3.informatik.uni-mannheim.de/~moer/querycompiler.pdf [Accessed 25.01.2007].

Molinaro, A. (2005) *SQL cookbook,* O'Reilly, Sebastopol, CA, USA.

Moody, D. L. (2000) 'Building links between IS research and professional practice: improving the relevance and impact of IS research', *Proceedings of the twenty first international conference on Information systems* Brisbane, Australia 2000. Association for Information Systems Atlanta, GA, USA pp. 351-360

Muralikrishna, M. (1992) 'Improved Unnesting Algorithms for Join Aggregate SQL Queries', *Proceedings of the 18th International Conference on Very Large Data Bases,* Vancouver, Canada 1992. pp. 91-102.

Negri, M., Pelagatti, G. and Sbattella, L. (1991) 'Formal semantics of SQL queries', *ACM Transactions on Database Systems,* 16 (3), pp. 513-534

Nock, C. (2004) *Data access patterns : database interactions in object-oriented applications,* Addison-Wesley, Boston, MA, USA.

Oracle Corporation (2005) *Query Optimization in Oracle Database10g Release 2 : An Oracle White Paper*, Oracle Corporation. Available from: http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_general_query_optimization_10gr2_0605.pdf [Accessed 17.11.2006].

Powell, G. (2005) *Oracle Data Warehouse Tuning for 10g,* Elsevier, Oxford, England.

Powell, G. (2007) *Oracle Performance Tuning for 10gR2,* (2nd Edn), Elsevier, Oxford, England.

Pressman, R. S. (2005) *Software engineering : a practitioner's approach,* (6th Edn), McGraw-Hill, Boston, MA, USA.

Ramakrishnan, R. and Gehrke, J. (2003) *Database Management Systems,* (3rd, International Edn), McGraw-Hill, Boston, MA, USA.

Rampant TechPress (2006) *SQL Design Patterns*. Available from: http://www.rampant-books.com/book_2006_1_sql_coding_styles.htm [Accessed 01.12.2006].

Rich, K. (2005) *Oracle Database Reference, 10g Release 2 (10.2)*, Oracle Corporation. Available from: http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14237.pdf [Accessed 15.12.2006].

Roy, P., Seshadri, S., Sudarshan, S. and Bhobe, S. (2000) 'Efficient and extensible algorithms for multi query optimization', *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* Dallas, TX, USA 2000. ACM Press New York, NY, USA pp. 249-260

Saunders, K. and Anderson, J. (2006) *Cloudscape Version 10: A technical overview*, IBM Corporation. Available from: http://www-

128.ibm.com/developerworks/db2/library/techarticle/dm-0408anderson/index.html [Accessed 03.09.2006].

Shasha, D. E. and Bonnet, P. (2003) *Database tuning : principles, experiments, and troubleshooting techniques,* ([Rev. Edn), Morgan Kaufmann, San Francisco, CA, USA.

Silberschatz, A. (2006) *Database system concepts,* (5th Edn), McGraw-Hill, Maidenhead, England.

Smith, J. M. and Chang, P. Y.-T. (1975) 'Optimizing the performance of a relational algebra database interface', *Communications of the ACM* 18 (10), pp. 568-579.

Sun Microsystems Inc. (2004) *Java(tm) 2 platform standard edition 5.0 API specification*. Available from: http://java.sun.com/j2se/1.5.0/docs/api/ [Accessed 16.10.2006].

Tao, H., Yuan, S. and Sunzi (2000) *Sun Tzu's art of war : the modern Chinese interpretation,* ([Rev. Edn), Sterling Pub., New York, USA.

The R Foundation for Statistical Computing (2006) *The R Project for Statistical Computing*. Available from: http://www.r-project.org/ [Accessed 17.10.2006].

Tidwell, J. (2006) *Designing interfaces,* O'Reilly, Sebastopol, CA, USA.

Tow, D. (2003) *SQL tuning,* (1st Edn), O'Reilly, Sebastopol, CA, USA.

Ullman, J. D. (1987) 'Database theory—past and future', *Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* San Diego, CA, USA1987. ACM Press New York, NY, USA pp. 1-10.

Visa International Service Association (2007) *Statistics*. Available from: http://corporate.visa.com/md/st/main.jsp?src=VQSU06 [Accessed 11.02.2007].

Warshaw, L. B. and Miranker, D. P. (1999) 'Rule-based query optimization, revisited', *Proceedings of the eighth international conference on Information and knowledge management,* Kansas City, MO, USA1999. ACM Press New York, NY, USA pp. 267-275.

Weinmann, W. (2006) *REPSI (Recording the Efficiency of Patterns / SQL*

Idioms), BerliOS Developer. Available from:
http://developer.berlios.de/projects/repsi/ [Accessed 04.01.2007].

Widom, J. (1996) 'Starburst active database rule system', *IEEE Transactions on Knowledge and Data Engineering,* 8 (4), pp. 583-595.

Wiki Wiki Web (2006) *Are Design Patterns Missing Language Features*. Available from:
http://www.c2.com/cgi/wiki?AreDesignPatternsMissingLanguageFeatures
[Accessed 09.01.2007].

Winslett, M. (2002) 'David DeWitt speaks out', *SIGMOD Record,* 31 (2), pp. 50-62.

Index