BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **19**, No 2
Sofia • 2019 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2019-0019

Performance Analysis and Optimization Techniques for Oracle Relational Databases

Fernando Almeida[1], *Pedro Silva*[2], *Fernando Araújo*[2]

[1]*Faculty of Engineering, Oporto University & INESC TEC, Porto, Portugal*
[2]*Higher Polytechnic Institute of Gaya, ISPGaya, V. N. Gaia, Portugal*
E-mails: *almd@fe.up.pt ispg3881@ispgaya.pt fernando.jopinheiro@gmail.com*

***Abstract*:** *Databases provide an efficient way to store, retrieve and analyze data. Oracle relational database is one of the most popular database management systems that is widely used in a different variety of industries and businesses. Therefore, it is important to guarantee that the database access and data manipulation is optimized for reducing database system response time. This paper intends to analyze the performance and the main optimization techniques* (*Forall, Returning, and Bulk Collect*) *that can be adopted for Oracle Relational Databases. The results have shown that the adoption of Forall and Bulk Collect approaches bring significant benefits in terms of execution time. Furthermore, the growth rate of the average execution time is lower for Bulk Collect than Forall. However, adoption of Returning approach doesn't bring significant statistical benefits.*

***Keywords*:** *Databases, performance analysis, optimization techniques, Oracle, information systems.*

1. Introduction

The demand for high performance processing is requested by several data-intensive applications. In this sense processing techniques and data organization such as grid computing, cloud and OLAP that have allowed the exploitation of a large volume of data in feasible time [1, 2]. One of the fundamental components in data-intensive applications is a data storage structure. Databases are a ubiquitous part of today's computing environment. DataBase Management Systems (DBMSs) are typically complex and used also in mission-critical software systems [3]. In fact, DBMSs are used in a wide variety of business and scientific applications, and also in the internet and electronic commerce applications. Over the last three decades, relational DBMS technology has proven to be highly adaptable and have evolved to accommodate new application requirements and the ever-increasing size and complexity of data [4]. However, and due to the emergence of data-intensive and mobile applications during the last years, the Object-Oriented DataBase Management Systems (OODBMSs) have gained significant market share [5]. The idea behind the concept of OODBMSs

is that by supporting data as objects in databases, the overhead of converting between objects and relations can be avoided, resulting in higher development efficiency and better performance. Furthermore, OODBMSs offer new features which are not present in the relational paradigm, such as the concepts of temporal evaluation of data, derived attributes, polymorphism, dynamic binding, among others [4, 6, 7].

Oracle database management system is one of the earliest, robust and most widely used in the context of storing and managing enterprise data. It incorporates numerous features both in terms of functionality and in terms of performance and scalability [8]. This characteristic of Oracle DBMS turns it specifically adequate for professional applications that require advanced scalability and reliability. In order to offer a development immersive environment that could take advantage of the features offered by SQL, Oracle created in 1991 a Procedural Language (PL) that extends the SQL language, which is called PL/SQL. PL/SQL is a standard data access language for Oracle relational databases that offers features like data encapsulation, exception handling, information hiding, and object orientation [9].

This paper focuses its analysis on looking for performance and optimization techniques that could be adopted in Oracle relational databases. In fact, most major database systems, including Oracle, started a few years ago to support the object-oriented paradigm. However, and due to the importance of the relational model and because it will remain the mainstream database model for many years, this work considers only the relational database paradigm. The paper is organized as follows: First, we perform a revision of literature in the field of relational databases by looking for performance analysis perspectives and introducing optimization techniques in Oracle PL/SQL. Subsequently, we present the adopted methodology and analyze the main results for three considered scenarios (write data, write & read data, and read data). Finally, we draw the conclusions of our work.

2. Related work

2.1. Relational databases and performance

Relational databases can be represented in tabular form consisting of rows and columns. The data must be an elementary table and is organized into tuples. A DBMS responds to commands given by application programs (e.g., SQL) in form of queries results, messages and completion codes [10]. The nature of the relational model does not require that users understand the representation of data in storage to retrieve it, but they need to know SQL syntax. A DBMS system provides a Data Definition Language (DDL) to specify and change the database schema, a Data Manipulation Language (DML) to express database queries, and a Data Control Language (DCL) to control the security and permission access. In practice, these three languages are not separate, instead, they simply form part of a single database language, such as the SQL [11].

Relational programming is nonprocedural. This allows that multiple tuples can be selected simultaneously without the adoption of cycles created by the programmer. Greenwald, Stackowiak and Stern [12] state “in a master-detail

relationship between tables, there can be one or many detail rows for each individual master row, yet the statements used to access, insert, or modify the data simply describe the set of results". Due to this property, programs access more than one record in a relational database more easily which causes those relational databases can be used more productively to extract large groups of data at once [13].

The activity of analyzing and improveing the system performance in DBMS is called database tuning [14]. In order to make a system quicker, the database tuning process involves typically change the data structures and parameters of a database system, the configuration of the operating system, or including the hardware. Shasha and Bonnet [15] summarize the five principles of performance considerations in a database environment:

- Think globally but fix locally – effective tuning requires a proper identification of the problem and a minimalist intervention.
- Partitioning breaks bottlenecks – a slow system is rarely slow because all its components are saturated. Usually, one part of the system limits its overall performance. That part is called a bottleneck.
- Start-up costs are high but running costs are low – most objects devote a substantial part of their resources to starting up. Therefore, it is expensive to begin a read operation on a disk, but once the read begins, the disk can deliver data at high speed.
- Render into Server side what is due to the Server – an important design question is the allocation of work between the database system (the server) and the application program (the client). Shasha and Bonnet [15] refer that the allocation of a specific task depends on three main factors: (i) the relative computing resources of client and server; (ii) where the relevant information is located; and (iii) whether the database task interacts with the screen.
- Be prepared for trade-offs – increasing the speed of an application often requires some combination of memory, disk and other computational resources. These resources typically cannot be individually optimized.

Other authors contributed to this discussion by exposing the main components of a database management system that have impact in the performance of a DBMS. Table 1 summarizes the main performance issues identified by most predominant authors in this field.

Table 1. Summary of the main identified performance issues

Performance issues	Ziauddin et al. [16]	Pavlo, Paulson and Rasin [17]	Corlatan et al. [18]	Slasha and Bonnet [15]	Pons [19]
Concurrency control and bottlenecks			×	×	×
Execution plan	×	×	×	×	×
Indexing		×	×	×	×
Programming model and language		×			
Recovery and logging				×	

Concurrency control is a process to ensure that data is updated correctly and appropriately when multiple transactions are concurrently executed in DBMS [20]. In general, concurrency control is an essential part of a DBMS. It is a mechanism for correctness when two or more database transactions that access the same data or data set are executed concurrently with time overlap. When multiple transactions are executed serially or sequentially, data is consistent in a database. However, if concurrent transactions with interleaving operations are executed, some unexpected data and inconsistent result may occur [15]. The concurrency control must also deal and resolve deadlock situations in database environments. Deadlock refers to a particular situation where two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain [21]. Deadlock is typically a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource. P o n s [19] establishes three types of deadlocks:

- Disk input/output (i/o) bottlenecks – i/o operations require read/write disk drive heads to physically move across the drive platters, potentially incurring a significant time penalty in the process;
- Central Processing Unit (CPU) bottlenecks – occur when too many resources compete for computer processing time at once;
- Random Access Memory (RAM) bottlenecks – RAM, like the CPU, is a physical resource of the server itself and, therefore, any other processes running on the server will take away from the amount of RAM available to the database system.

Execution plan is considered by all identified authors as the main performance issue related to database performance. The performance of SQL statements depends heavily on the optimality of execution plans generated by the query optimizer. Z i a u d d i n et al. in [16] state that the query optimizer has the unenviable task of generating efficient plans for SQL statements of varied characteristics: simple vs. complex, lightweight vs. resource intensive, recursive vs. non-recursive. It must be highlighted that the way an SQL statement is written affects the manner how hidden implementation details are performed by the DBMS. P o n s [19] refers that poorly constructed SQL is a major cause of database system performance degradation, and usually is one of the first factors addressed in performance tuning.

Indexing process intends to enhance the performance of select statements against a database table. When missing indexing of a table, the system has to make full table scan in order to find the searchable item. This leads to overloading RAM and CPU, thus considerably increasing the execution time of a query. However, the use of queries in all attributes of a table may not be considered a feasible solution. Since indexes must be modified to reflect table changes, their use incurs a significant amount of time overhead. The definitions of indexes on tables will typically slow down the database system as inserts and updates are performed. In order to deal with this issue, DataBase Administrator (DBA) must continuously monitor DBMS performance statistics to re-evaluate the creation and deletion of indexes [22].

Programming model and language is also referred by P a v l o, P a u l s o n and R a s i n in [17] as a factor that affects the database performance. In fact, programs in high-level languages, such as SQL, are easier to write, modify and understand. On

the other hand, when adopting procedural languages, such as C/C++ or Java, describing tasks in a declarative language like SQL can be challenging. Additionally, each language has its own approach to embed static SQL statements. As a consequence, the performance of an application is not only affected by the SQL code, but also includes the SQL embedded process adopted by the programming language.

Recovery and logging are mentioned by Shasha and Bonnet [15] as a complementary factor in concurrency control that affects the database performance. The recovery process consists of an integrated physical and logical recovery mechanism that protects database integrity in case of hardware and software failure, such as deadlock situations. Furthermore, Pratt and Last in [23] state that deadlock handling involves the adoption of detection techniques, prevention mechanisms and avoidance schemes.

Finally, the optimization process must take into account the various factors in a system that determines its response time. Tiwari [24] advocates that performance analysis should consider all technological infrastructure (e.g., CPU, disk, network, I/O), since very often performance problems can be in the network response capacity and not in the database response time.

2.2. Optimization techniques in Oracle PL/SQL

Procedural Language/Structured Query Language (PL/SQL) is Oracle Corporation's procedural language extension to SQL. Hellström [25] advocates that one of the main reasons why PL/SQL is so important in the context of application database environments is that SQL itself doesn't offer a robust construction to apply logical processing to DML statements. Due to this limitation of SQL, the PL/SQL has emerged as an important programming language that lets programmers to access common 3GL constructs such as conditional blocks, loops and exceptions. Oracle [26] summarizes the main benefits of using the PL/SQL programming language with an Oracle database, namely in terms of: (i) integration of procedural constructs with SQL; (ii) modularized program development; (iii) improved performance; (iv) integration with Oracle tools; (v) portability; and (vi) exception handling.

The central purpose of PL/SQL is to provide a portable, fast, easy way to write and execute SQL against an Oracle database. Oracle PL/SQL has been used in several applications that require data-intensive analysis [27-29]. Poljak, Poscic and Jaksic [30] identify some of the main advantages of an Oracle database when compared to other relational databases, unleashing its robustness, reliability and performance optimization for large data volume. Oracle uses two engines to process PL/SQL code: all procedural code is handled by the PL/SQL engine; while all SQL code is handled by the SQL statement executor or SQL engine. There is an overhead associated with each context switch between the two engines [31]. Therefore, goals are simply straightforward by reducing the number of context switches in order to improve the performance [32]. For that, Oracle offers three additional PL/SQL statements: (i) Bulk Collect; (ii) Forall; and (iii) Returning.

2.2.1. Bulk collect

Bulk processing features of PL/SQL change the way the PL/SQL engine communicates with the SQL layer. Bulk binds can improve the performance when loading collections from queries. The Bulk Collect construct binds the output of the query to the collection. When PL/SQL processes this statement, the whole collection, instead of each individual collection element, is sent to the database server for processing [33]. Bulk binds also allow similar DML statements to be executed with one call instead of requiring a separate call for each. Gupta [34] emphasizes the importance of bulk binds in business scenarios where several cursors are loaded with hundreds, thousands or millions of records causing significant degradations in the performance of web applications.

2.2.2. Forall

The Forall construct allows the user to gain the same type of efficiency offered by BULK COLLECT when performing write operations. This construct packages up multiple write statements and sends them off to the Oracle Database in a single message, increasing the overall performance of the operation [35]. The Forall is usually much faster than an equivalent For or While loop [31].

2.2.3. Returning

One of the characteristic operations in an IT application is the need to check whether the data sent to the database has actually been saved. Thus, typically, the developers execute a new query to the database to verify the correct insertion of the information [36]. However, this approach necessarily has performance costs that can be avoided. The Returning statement increases the performance by allowing the prompt return in column the “Insert”, “Update” and “Delete” instructions. This also eliminates the need of using a “Select” statement to get the content of a given table. By default, programmers can use this clause only when operating on exactly one row. However, when using bulk SQL, they can use the form Returning Bulk Collect Into to store the results in one or more collections [31].

3. Methodology

Quantitative research techniques were adopted in order to study the performance of Oracle relational databases. Quantitative research method is characterized as a systematic approach of investigation during which numerical data are observed, collected, transformed and analyzed by the researcher [37-38]. This approach tries to find evidences that could support or contradict an idea or hypothesis [39].

The adopted quantitative methodology is composed by four steps:

1. Determine the basic question that is intended to be answered with the research study. In our study, we want to determine whether the optimization techniques provided by PL/SQL (Forall, Returning and Bulk Collect) improve the performance in an Oracle database.

2. Identify variables, measures, and the research design to use in formulating the research question. In our work, we adopted a relational model composed by just

one table (TestTable) containing one attribute of integer type. Moreover, we define three test scenarios and, for each scenario, we change the number of records in the database (from 1000 to 1,000,000), and we measure the average time of five execution attempts in order to decrease the volatility of memory management policies and CPU utilization.

3. Choose statistical analysis tools for analyzing the data collected. In our study we adopted Stata Software v13.0 in order to perform a statistical analysis of the data.

4. Interpretation of the results of the analysis based on the statistical significance determined. In our work, the interpretation of results is done by looking for the hypothesis testing and also considering the literature review in the field.

The first scenario (Scenario I, Table 2) considers the process of writing data in an Oracle database. For that, we initially start by cleaning all the existence records in the database and we create a new table of integers that contains values from 1 up to *N* (number of records). Then, in SA1 approach we use a traditional for loop to insert values in the database; Forall approach inserts the same elements using a Forall loop.

Table 2. PL/SQL code for Scenario I

Standard Approach 1 (SA1)	Forall Approach

DECLARE
type v_type_a IS TABLE OF Integer index by binary_integer;
v_a v_type_a;
timeStart TIMESTAMP;
timeEnd TIMESTAMP;
BEGIN
DELETE FROM TestTable;
For i In 1..1000
Loop
v_a(i):=i;
End Loop;
timeStart := SYSTIMESTAMP;
FOR i in 1..v_a.count
LOOP
Insert into TestTable values (v_a(i));
END LOOP;
timeEnd := SYSTIMESTAMP;
dbms_output.put_line(timeEnd-timeStart);
END;
	DECLARE
	type v_type_a IS TABLE OF Integer index by binary_integer;
	v_a v_type_a;
	timeStart TIMESTAMP;
	timeEnd TIMESTAMP;
	BEGIN
	DELETE FROM TestTable;
	For i In 1..1000
	Loop
	v_a(i):=i;
	End Loop;
	timeStart := SYSTIMESTAMP;
	FORALL i in 1..v_a.count
	Insert into TestTable values (v_a(i));
	timeEnd := SYSTIMESTAMP;
	dbms_output.put_line(timeEnd-timeStart);
	END;

The second scenario (Scenario II, Table 3) considers the process of writing and immediately reading data from an Oracle database. For that, and like in the Scenario I, we start by creating a new table of integers and, then, we write the data in the database. Finally, we extract these data by using a loop cursor (SA2 approach) and using the returning instruction (Returning approach).

Table 3. PL/SQL code for Scenario II

Standard Approach 2 (SA2)	Returning Approach

DECLARE
type v_type_a IS TABLE OF Integer index by binary_integer;
v_a v_type_a;
l_tab v_type_a;
i integer;
i_rec integer;
timeStart TIMESTAMP;
timeEnd TIMESTAMP;
Cursor c1 IS Select * from TestTable;
BEGIN
DELETE FROM TestTable;
For i In 1..1000
Loop
v_a(i):=i;
End Loop;
timeStart := SYSTIMESTAMP;
FOR i in 1..v_a.count
LOOP
Insert into TestTable values (v_a(i));
END LOOP;
Open c1;
FETCH c1 INTO i_rec;
WHILE c1%FOUND
LOOP
FETCH c1 INTO i_rec;
i:=i+1;
End Loop;
Close c1;
timeEnd := SYSTIMESTAMP;
dbms_output.put_line(timeEnd-timeStart);
END;
	DECLARE
	type v_type_a IS TABLE OF Integer index by binary_integer;
	v_a v_type_a;
	l_tab v_type_a;
	j integer;
	timeStart TIMESTAMP;
	timeEnd TIMESTAMP;
	BEGIN
	DELETE FROM TestTable;
	For i In 1..1000
	Loop
	v_a(i):=i;
	End Loop;
	timeStart := SYSTIMESTAMP;
	FORALL i in 1..v_a.count
	Insert into TestTable values (v_a(i))
	RETURNING a BULK COLLECT INTO l_tab;
	timeEnd := SYSTIMESTAMP;
	dbms_output.put_line(timeEnd-timeStart);
	END;

The third scenario (Scenario III, Table 4) considers the process of reading data from an Oracle database. For that, we initially start by declaring a new cursor that will read all data from TestTable. Then, we read the data by using a loop cursor (SA3 approach) and using the bulk collect approach. In this last approach all the content of the database is immediately stored in a table, which content is read in the client side using a for loop.

The tests were performed in three laptops with Intel Core 2.50 GHz processors and 8 GB of RAM. The operating system was different for each machine: Windows 8.1 Professional, Windows 10 Home Edition and Linux – Ubuntu 15.10 64 bits, using Oracle Database 11g Release 2 as DBMS.

Table 4. PL/SQL code for Scenario III

Standard Approach 3 (SA3)	Bulk Collect Approach

DECLARE
type v_type_a IS TABLE OF Integer index by binary_integer;
v_a v_type_a;
l_tab v_type_a;
i_rec integer;
timeStart TIMESTAMP;
timeEnd TIMESTAMP;
i integer;
Cursor c1 IS Select * from TestTable;
BEGIN
i:=0;
timeStart := SYSTIMESTAMP;
Open c1;
FETCH c1 INTO i_rec;
WHILE c1%FOUND
LOOP
FETCH c1 INTO i_rec;
i:=i+1;
End Loop;
Close c1;
timeEnd := SYSTIMESTAMP;
dbms_output.put_line(timeEnd-timeStart);
dbms_output.put_line('Number of records:' || i);
END;
	DECLARE
	type v_type_a IS TABLE OF TestTable%ROWTYPE index by binary_integer;
	v_a v_type_a;
	j integer;
	timeStart TIMESTAMP;
	timeEnd TIMESTAMP;
	Cursor c1 IS Select * from TestTable;
	BEGIN
	timeStart := SYSTIMESTAMP;
	Open C1;
	Fetch c1 BULK COLLECT INTO v_a;
	For i In 1..v_a.count
	Loop
	j:=j+1;
	End Loop;
	Close c1;
	timeEnd := SYSTIMESTAMP;
	dbms_output.put_line(timeEnd-timeStart);
	END;

4. Results and discussion

Table 5 shows the average measured time for the three considered scenarios: Scenario I – Write Data, Scenario II – Write & Read Data, and Scenario III – Read Data. For each scenario, we measured the time, in milliseconds (ms), using a standard approach and adopting the optimization techniques provided by Oracle (Forall, Returning and Bulk Collect).

Table 5. Comparative analysis of the average execution time

N	Scenario I – Write Data		Scenario II – Write & Read Data		Scenario III – Read Data	
	SA1	Forall	SA2	Returning	SA3	Bulk Collect
1000	61.80	1.13	65.73	52.73	11.33	3.33
5000	284.33	4.00	312.40	225.47	37.20	3.47
10,000	547.20	7.33	620.93	451.73	64.00	6.40
50,000	2466.33	36.07	3072.00	2242.07	299.87	21.40
100,000	6203.93	69.33	5702.20	4505.33	600.80	38.07
500,000	26565.13	357.53	30815.73	24603.80	2593.87	174.67
1,000,000	56381.60	1112.13	62744.20	51884.67	5150.13	341.87

A comparative graphical analysis regarding the evolution of the execution time in milliseconds (*y* axis) along the increment of the number of records (*x* axis) is depicted in Fig. 1.

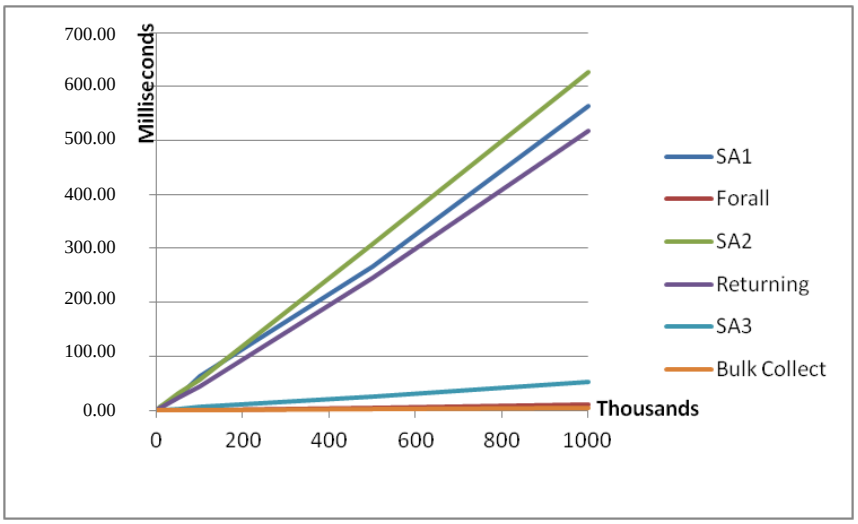

Fig. 1. Graphical comparative analysis for three scenarios

The adoption of optimization techniques provided by Oracle doesn't have the same impact for all scenarios. The use of Forall instruction decreases significantly the total execution time. On the other hand, adoption of Returning and Bulk Collect instructions don't have the same impact on the total execution time.

4.1. Scenario I – write data

The adoption of the Forall instruction brings significant benefits in terms of runtime. These benefits are greater as long the value of *N* is increased. For a small *N* (*N*=1000) the average execution time decreases from 61.80 ms to 1.13 by using Forall, which represents a difference of 60.67 ms. However, this difference increases significantly for more than 55,000 ms, when *N* is equal to 1 million of records. The variance and standard deviation also follow a similar pattern of behavior to the average, and its value increases with *N*. However, when we calculate the growth rate of the average execution time of both approaches, we reach to the conclusion that this value is 0.06% for SA1 and 0.07% for Forall. Therefore, it can be concluded that as *N* increases the growth of execution time in percentage is similar for both approaches.

Table 6 presents the hypothesis test conducted for Scenario I.

It is possible to verify that *p*-values from *t*-score and *f*-score are in all cases lower than 0.00001. In this way, if we consider a significance level of 5% (0.05), these results indicate a strong evidence against the null hypotheses. Therefore we reject the null hypothesis and it is possible to conclude that the execution time is lower using the Forall instruction.

Table 6. Hypothesis test for Scenario I

N	Approach	*t*-score	*f*-score	*p*-value from *t*-score	*p*-value from *f*-ratio (ANOVA)	95% Conf. interval	
						Min	Max
1000	SA1	7.6513	7.60×10^{3}	< 0.00001	< 0.00001	44.80	78.80
	Forall					0.94	1.33
5000	SA1	7.9825	2.20×10^{4}	< 0.00001	< 0.00001	209.16	359.91
	Forall					3.49	4.51
10,000	SA1	7.9985	1.20×10^{4}	< 0.00001	< 0.00001	402.44	691.96
	Forall					6.02	8.65
50,000	SA1	5.5486	1.90×10^{4}	< 0.00001	< 0.00001	1526.94	3405.72
	Forall					29.26	42.87
100,000	SA1	9.9646	1.10×10^{4}	< 0.00001	< 0.00001	4874.17	7509.70
	Forall					56.74	81.93
500,000	SA1	9.8408	5.00×10^{3}	< 0.00001	< 0.00001	20853.81	32276.46
	Forall					276.95	438.12
1,000,000	SA1	10.7063	728.044	< 0.00001	< 0.00001	45317.13	67446.07
	Forall					702.07	1522.2

4.2. Scenario II – write data & read data

The executed tests confirm a minor improvement by the adoption of Returning instruction. The difference between two approaches is not so relevant like in Scenario I. For instance, the difference between two means is only 13 ms for *N* equal to 1 thousand, and increases to more than 10,000 ms for *N* equal to 1 million. It is possible to check a similar behavior for variance and standard derivation. The growth rate of the average execution time of both approaches is similar (0.06% for SA2 and 0.07% for Returning). Other relevant aspect that could be highlighted is that all measured values are between the smallest and largest for both approaches. This situation happens because tests were performed in three machines with different operating systems.

Table 7 presents the hypothesis test conducted for Scenario II.

It is possible to verify that the hypothesis is inconclusive in order to determine the behavior of the mean and variance. The *p*-value from *t*-score is only 3/7 (three in seven situations) lower than the significance level of 0.05. ANOVA values are also similar and only 1/7 (one in seven situations) is lower than the significance level of 0.05. Therefore, we cannot reject the null hypothesis and it is not possible to conclude that the execution time and variances are different.

Table 7. Hypothesis test for Scenario II

N	Approach	*t*-score	*f*-score	*p*-value from *t*-score	*p*-value from *f*-ratio (ANOVA)	95% Conf. Interval	
						Min	Max
1000	SA2					48.02	83.45
	Returning	1.1744	1.2548	0.125063	0.338454	36.92	68.54
5000	SA2					228.60	396.20
	Returning	1.8661	2.3728	0.036267	0.058831	171.06	279.87
10,000	SA2					456.75	785.65
	Returning	1.8453	2.3005	0.037797	0.065502	343.31	560.16
50,000	SA2					2266.88	3877.12
	Returning	1.8498	2.3338	0.037462	0.062332	1715.04	2769.09
100,000	SA2					4089.53	7314.87
	Returning	1.3463	2.5135	0.094502	0.047886	3488.14	5522.53
500,000	SA2					23699.46	37932.01
	Returning	1.5401	2.0931	0.067381	0.089677	19684.96	29522.64
1,000,000	SA2					48740.01	76748.53
	Returning	1.3416	1.8627	0.095253	0.12834	41623.67	62145.66

4.3. Scenario III – read data

The adoption of Bulk Collect operation brings also significant benefits in terms of runtime. This benefits increases with the value of *N*. For a small *N* (*N*=1000) the average execution time decreases from 11.33 ms to 3.33 ms by using Bulk Collect, which represents a difference of 8 ms. However, this difference increases significantly for more than 4800 ms, when *N* is equal to 1 million of records. The variance and standard derivation are also lower when adopting the Bulk Collect approach. The growth rate of the average execution time of both approaches is similar to both scenarios (0.05% for SA3 and 0.04% for Bulk Collect), but decreased when compared to Scenario I (Table 8).

Table 8. Comparative growth rate for three scenarios

Growth Rate					
Scenario I		Scenario II		Scenario III	
SA1	Forall	SA2	Returning	SA3	Bulk Collect
0.06%	0.07%	0.06%	0.07%	0.05%	0.04%

In this phase is also relevant to analyze the comparative behavior of Forall (Scenario I) and Bulk Collect (Scenario III). Descriptive statistics of both scenarios are given in Table 9.

Table 9. Comparative analysis between scenarios I and III (descriptive statistics)

Forall		Bulk Collect	
Obs.	105	Obs.	105
Mean	226.791	Mean	84.171
Std. Dev.	471.744	Std. Dev.	121.669
Variance	222542.7	Variance	14803.45
Skewness	3.409	Skewness	1.428
Kurtosis	16.547	Kurtosis	3.641

The minimum execution value times are equal for both scenarios, but significantly different when looking for maximum values (it is bigger in case of Forall). In fact, there is no difference also for quartile 1. However, the differences start to appear for median and quartile 3. This situation can be easily verified looking for the box plot representation (Fig. 2). The variance and standard derivation are also lower for Bulk Collect scenario.

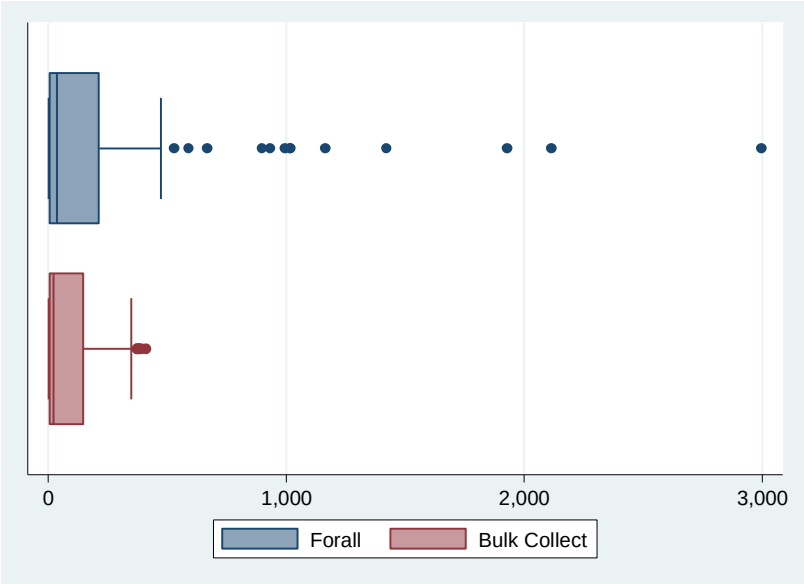

Fig 2. Box plot representation of forall and Bulk Collect Scenarios

Finally, Table 10 presents the hypothesis test conducted for Scenario III. It is possible to verify that *p*-values from *t*-score and *f*-score are in all situations lower than the significance level of 0.05. These results indicate strong evidence against the null hypotheses, and let us conclude that the execution time/variance averages are lower using the Bulk Collect approach.

Table 10. Hypothesis test for Scenario III

N	Approach	*t*-score	*f*-score	*p*-value from *t*-score	*p*-value from *f*-ratio (ANOVA)	95% conf. Interval	
						Min	Max
1000	SA3	5.0326	14.0189	0.000013	< 0.00001	8.04	14.63
	Bulk Collect					2.45	4.21
5000	SA3	8.6577	88.2201	< 0.00001	< 0.00001	28.89	45.51
	Bulk Collect					2.58	4.35
10,000	SA3	8.88	104.6890	< 0.00001	< 0.00001	50.15	77.85
	Bulk Collect					5.05	7.75
50,000	SA3	8.6506	549.0691	< 0.00001	< 0.00001	230.89	368.85
	Bulk Collect					18.46	24.34
100,000	SA3	8.5196	688.4231	< 0.00001	< 0.00001	459.24	742.36
	Bulk Collect					32.67	43.46
500,000	SA3	6.835	2.6000×10^3	< 0.00001	< 0.00001	1834.88	3352.85
	Bulk Collect					159.83	189.50
1,000,000	SA3	6.7639	3.4000×10^3	< 0.00001	< 0.00001	3625.69	6674.58
	Bulk Collect					315.7	368.03

5. Conclusion

Bulk binds (e.g., Forall, Returning, and Bulk Collect) are PL/SQL techniques where, instead of multiple individual SQL statements (e.g., Select, Insert, Update, or Delete), it guarantees that all of the operations are carried out at once, in bulk. This avoids the context switching between the SQL engine and PL/SQL engine. These optimization techniques can be relatively easy to implement and have the advantage to increase the performance in Oracle relational databases.

The tests performed in the Oracle relational database let us conclude that Forall and Bulk Collect instructions bring significant benefits in term of execution time. In fact, the use of these two approaches decreased the average execution time and variance. Additionally, it was possible to conclude that the growth rate of the average execution time is lower for Bulk Collect instruction than Forall approach. This difference has impact when the number of records in the database increases. Finally, the tests performed revealed that there aren't significant statistical benefits by the use of Returning.

Some limitation of this work can also be emphasized. Firstly, we only use integer values in our tests. Therefore, it is not proven that the behavior of optimization PL/SQL techniques will remain the same, if we use other data types such as varchar, date/time or records. Secondly, since the execution time of some tests is low for a small number of records, the overall result is influenced by memory buffers that can represent a bottleneck in most database operations.

References

1. Blaheta, R., K. Georgiev, O. Jaki, R. Kohut, S. Margenov, J. Stary. High Performance Computing in Micromechanics with an Application. – Cybernetics and Information Technologies, Vol. **17**, 2017, No 5, pp. 5-16.
   https://doi.org/10.1515/cait-2017-0050
2. Dimitrov, D., E. Atanassov. Accounting Services for Heterogeneous Computing Resources. – Cybernetics and Information Technologies, Vol. **17**, 2017, No 5, pp. 81-88.
   https://doi.org/10.1515/cait-2017-0057
3. Hellerstein, J., M. Stonebraker, J. Hamilton. Architecture of a Database System. – Foundations and Trends in Databases, Vol. **1**, 2007, No 2, pp. 141-259.
   https://doi.org/10.1561/1900000002
4. Feuerlicht, G. Database Trends and Directions: Current Challenges and Opportunities. – In: J. Pokorný, V. Snásel, K. Richta, Eds. Dataeso. 2010, pp. 163-174.
   CEUR-WS.org.
5. Moniruzzaman, A., S. Hossain. NoSQL Database: New Era of Databases for Big Data Analytics – Classification, Characteristics and Comparison. – International Journal of Database Theory and Application, Vol. **6**, 2013, No 4, pp. 1-14.
6. Nayak, A., A. Poriya, D. Poojary. Type of NOSQL Databases and Its Comparison with Relational Databases. – International Journal of Applied Information Systems (IJAIS), Vol. **5**, 2013, No 4, pp. 16-19.
   https://doi.org/ 10.5120/ijais12-450888
7. Damesha, H. Object Oriented Database Management Systems-Concepts, Advantages, Limitations and Comparative Study with Relational Database Management Systems. – Global Journal of Computing Science and Technology: Software & Data Engineering, Vol. **15**, 2015, No 3, pp. 1-9.
8. Liu, H. Oracle Database Performance and Scalability: A Quantitative Approach. Wiley-IEEE Computer Society, New Jersey, 2011.
9. Feuerstein, S., B. Pribyl. Oracle PL/SQL Programming. Sebastopol, O'Reilly Media, 2014.
10. Darwen, H. An Introduction to Relational Database Theory. London, Bookboon, 2010.
11. Silberschatz, A., H. Korth. Database System Concepts. Columbus, McGraw-Hill Education, 2010.
12. Greenwald, R., R. Stackowiak, J. Stern. Oracle Essentials – Oracle Database 11g. Sebastopol, O'Reilly Media, 2008.
13. Medhi, S., H. Baruah. Relational Database and Graph Database: A Comparative Analysis. – Journal of Process Management and New Technologies, Vol. **5**, 2017, No 2, pp. 1-9.
   https://doi.org/ 10.5937/jouproman5-13553
14. Colley, D., C. Stanier. Identifying New Directions in Database. – Performance Tuning, Vol. **121**, 2017, pp. 260-265.
   https://doi.org/10.1016/j.procs.2017.11.036
15. Shasha, D., P. Bonnet. Database Tuning: Principles, Experiments, and Troubleshooting Techniques. Burlington, Morgan Kaufmann Publishers, 2002.
16. Ziauddin, M., D. Das, H. Su, Y. Zhu, K. Yagoub. Optimizer Plan Change Management: Improved Stability and Performance in Oracle 11g. – In: Proc. of PVLBD'08 Conference, Auckland, New Zealand, 2008, pp. 1346-1355.
17. Pavlo, A., E. Paulson, A. Rasin. A Comparison of Approaches to Large-Scale Data Analysis. – In: Proc. of PVLBD'09 Conference, New York, USA, 2009, pp. 165-178.
18. Corlatan, C. M. Lazar, V. Luca, O. Petricica. Query Optimization Techniques in Microsoft SQL Server. – Database Systems Journal, Vol. **5**, 2014, No 2, pp. 33-48.
19. Pons, A. Database Tuning and Its Role in Information Technology Education. – Journal of Information Systems Education, Vol. **14**, 2003, No 4, pp. 381-387.
20. Connolly, T., C. Begg. Database Systems: A Practical Approach to Design, Implementation, and Management. New York, Pearson Education, 2014.
21. Coronel, C., S. Morris. Database Systems: Design, Implementation, & Management. New Hampshire, Cengage Learning, 2018.

22. M u l l i n s, C. Database Administration: The Complete Guide to DBA Practices and Procedures. Boston, Addison-Wesley Professional, 2012.
23. P r a t t, P., M. L a s t. Concepts of Database Management. New Hampshire, Cengage Learning, 2014.
24. T i w a r i, V. Oracle Performance Tuning (for Oracle DBAs). – International Journal of Human Computer Interaction and Data Mining, Vol. **1**, 2018, No 1-2, pp. 30-36.
25. H e l l s t r ö m, I. Oracle SQL & PL/SQL Optimization for Developers Documentation. Oracle Media Networks. Retrieved 10 November 2017.
https://media.readthedocs.org/pdf/oracle/latest/oracle.pdf
26. O r a c l e. Benefits of PL/SQL. Oracle Academy. Retrieved 22 November 2017.
http://web.cerritos.edu/hohly/SitePages/Oracle/PL_SQL/PLSQL_course_materials%206_07/PLSQL_s01_l02.pdf
27. B i s s e t, K., J. C h e n, S. D e o d h a r, X. F e n g, Y. M a, M. M a r a t h e. Indemics: An Interactive High-Performance Computing Framework for Data Intensive Epidemic Modeling. – ACM Transaction on Modelling and Computer Simulation, Vol. **24**, 2014, No 1, pp. 1-54.
https://dx.doi.org/10.1145%2F2501602
28. K a u l a, R. Operational Intelligence through Performance Trends: An Oracle Prototype. – International Journal of Business Intelligence and Systems Engineering, Vol. **1**, 2017, No 2, pp. 140-165.
https://doi.org/10.1504/IJBISE.2017.088686
29. A l m e i d a, F., J. M o n t e i r o, J. F e r r e i r a. Building an Effective Data Warehousing for Financial Sector. – Automatic Control and Information Sciences, Vol. **3**, 2017, No 1, pp. 16-25.
http://dx.doi.org/10.12691/acis-3-1-4
30. P o l j a k, R., P. P o s c i c, D. J a k s i c. Comparative Analysis of the Selected Relational Database Management Systems. – In: Proc. of International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'17), Opatija, 2017, pp. 1496-1500.
https://doi.org/ 10.23919/MIPRO.2017.7973658
31. A l m e i d a, F. Developing Effective PL/SQL. Palo Alto, ISSUU Publishing, 2016.
32. F e u e r s t e i n, S. The Best of Oracle PL/SQL: Must Know Features, Best Practices and New Features in Oracle Database 11g. Toad World. Retrieved 21 December 2017.
https://stage.toadworld.com/cfs-file/__key/communityserver-wikis-components-files/00-00-00-00-03/Best-of-Oracle-PLSQL-11g.pdf
33. F e u e r s t e i n, S., B. P r i b y l, C. D a w e s. Oracle PL/SQL Language: Pocket Reference. Sebastopol, O'Reilly Media, 2008.
34. G u p t a, S. Advanced Oracle PL/SQL Developer's Guide. Birmingham, Packt Publishing, 2016.
35. G r e e n w a l d, R., R. S t a c k o w i a k, G. D o d g e, D. K l e i n, B. S h a p i r o, C. C h e l l i a h. Professional Oracle Programming. New Jersey, Wrox, 2005.
36. C a m p b e l l, L., C. M a j o r s. Database Reliability Engineering: Designing and Operating Resilient Database Systems. Newton, Massachusetts, O'Reilly Media, 2017.
37. M a r t i n, W., K. B r i d g m o n. Quantitative and Statistical Research Methods: From Hypothesis to Results. London, Jossey-Bass, 2012.
38. C r e s w e l l, J. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. New York, SAGE Publications, 2013.
39. Q u e i r ó s, A., D. F a r i a, F. A l m e i d a. Strengths and Limitations of Qualitative and Quantitative Research Methods. – European Journal of Education Studies, Vol. **3**, 2017, No 9, pp. 369-387.
https://doi.org/10.5281/zenodo.887089

Received: 04.04.2018; Second Version: 04.05.2019; Accepted: 14.05.2019