**Michał APOLINARSKI**
POZNAN UNIVERSITY OF TECHNOLOGY, INSTITUTE OF CONTROL AND INFORMATION ENGINEERING
pl. M. Skłodowskiej-Curie 5, 60-965 Poznań

# Influence of using cryptography on data processing in RDBMS Oracle 10g

**M.Sc. eng. Michał APOLINARSKI**

Michał Apolinarski received an M.Sc. degree in computer science from the Poznań University of Technology in 2008. His research interests include data security in information systems and influence of using cryptology on data processing.

*e-mail: michal.apolinarski@put.poznan.pl*

**Abstract**

Ensuring the confidentiality, privacy and integrity of data is a major issue for the security of database systems. In this paper the author investigates the efficiency of data processing in relational database management system Oracle 10g when built-in mechanism called Transparent Data Encryption (TDE) is used to encrypt table columns in order to increase data confidentiality and for data integrity control. Transparent Data Encryption supports table columns encryption using 3DES and AES algorithms with 128-, 192, 256-bits key length and data integrity using cryptographic hash function SHA-1.

**Keywords**: database security, cryptography in database, Oracle 10g, Oracle advanced security, transparent data encryption.

## Wpływ stosowania mechanizmów kryptograficznych na przetwarzanie danych w SZBD Oracle 10g

**Streszczenie**

Zapewnienie poufności, integralności i prywatności danych ma bardzo duże znaczenie dla bezpieczeństwa systemów informatycznych, a w szczególności dla bezpieczeństwa systemów baz danych. W tym artykule autor przedstawia wyniki doświadczenia badającego wpływ mechanizmów kryptograficznych na wydajność przetwarzania danych w systemie zarządzania relacyjną bazą danych (SZBD) Oracle 10g z wykorzystaniem wbudowanego narzędzia Transparent Data Encryption (TDE). Mechanizm TDE przeznaczony jest do szyfrowania kolumn tabeli w celu podwyższenia stopnia poufności danych i kontroli integralności danych. Transparent Data Encryption obsługuje szyfrowanie kolumn tabeli przy użyciu algorytmów kryptograficznych 3DES lub AES z kluczem 128-bitowym, 192-bitowym lub 256-bitowym. Do kontroli integralności danych wykorzystywana jest kryptograficzna funkcja skrótu SHA-1 generująca skrót o długości 160-bitów. Przeprowadzone testy obejmowały pomiary wydajności operacji SELECT, INSERT oraz UPDATE na przygotowanej kolekcji 10000 krotek danych.

**Słowa kluczowe**: bezpieczeństwo baz danych, kryptografia w bazach danych, transparent data encryption.

## 1. Introduction

Oracle Advanced Security was introduced in Oracle 8i and combines database encryption, strong authentication and network encryption.

In this paper the author investigates influence of cryptography on data processing in Oracle 10g when built-in Oracle Advance Security Transparent Data Encryption is used to increase data security on selected table columns. The first section presentss general conception of Transparent Data Encryption, supported algorithms and data types that can be encrypted.

## 2. Transparent Data Encryption

Transparent Data Encryption provides an easy to use encryption mechanism to increase data privacy on selected tables and columns in the relational database management system Oracle. All encryption keys are managed automatically by TDE. Database administrators do not have to manually create any additional database structures like triggers or views. TDE creates 3 additional database structures: dba_encrypted_columns, all_encrypted_columns, user_encrypted_columns.

Database administrators only need to create and open Oracle Wallet (to do this ALTER SYSTEM privilege and correct password defined in *sqlnet.ora* are needed) and define tables (create table or alter existing table) with correct clauses to use TDE. TDE mechanisms are completely transparent for a database user or application, only successful user authentication in database is needed. Keys used in column encryption process are stored in a cipher form secured by the database server master key. The server master key is stored outside the database in Oracle Wallet (Fig. 1 shows a general conception of TDE). Oracle Wallet contains authentication information, passwords, PKI private keys, the TDE master key, certificates and other confidential data required, for example, by SSL protocol.

The default encryption method in TDE is AES with 192-bit key length. TDE also supports AES with 128-bit or 256-bit key length and 3DES with 168-bit key length. Using salt (pseudorandom initialization value) is also default and can be disabled if needed. For data integrity check the TDE uses the cryptographic hash function SHA-1.

According to the Oracle 10g documentation, the TDE supports encryption for the following data types: CHAR, DATA, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, NCHAR, NUMBER, NVARCHAR2, RAW, ZONE, VARCHAR2. Encryption cannot be used with index types other than B-tree, range scan search through an index, external large objects (BFILE), materialized view logs, original import/export utilities [1, 2, 3].

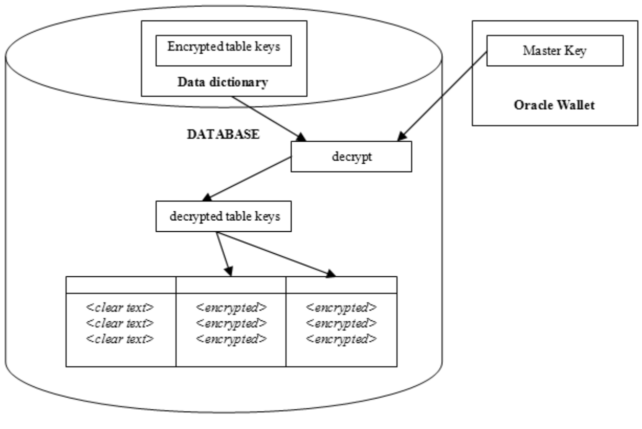

Fig. 1. General concept of TDE [2]
Rys. 1. Koncepcja TDE [2]

## 3. The Influence of TDE on Data Processing

This section explains the purpose and scope of the research, as well as its environment and methods used. PL/SQL procedures used for tests are presented and the obtained results are given.

### 3.1. Purpose and Scope of Tests

Cryptographic algorithms are complex and influence the performance of the system in which they are used, for example computer networks, operating systems and relational database management systems. This fact is often the reason for not using cryptographic methods to secure data. They are not used in order to improve (or at least not to decrease) the performance of a given environment. The cost of such a decision may cause weaker or no data security.

According to the Oracle 10g documentation, there is storage overhead between 1 and 52 bytes for each encrypted value using TDE column encryption. Reasons for this storage overhead is: padding encrypting value to the next 16 byte (for AES; with 3DES168, to the next 8 bytes), 20 byte for integrity check (message authentication code, MAC) and if 'SALT' is specified on the encrypted value, additional 16 bytes are required for a pseudorandom value in the plain form. This overhead is important in storage planning but a database administrator or a user does not have to expand columns manually [1, 2].

This document presents the performance tests related to the efficiency of SQL queries like SELECT, INSERT and DELETE depending on which the cryptographic method and key length was used.

### 3.2. Environment of Tests

The test environment was a virtual machine (MS Virtual PC 2007). The virtual machine runs on a computer with the following parameters: CPU Intel Core 2 Duo (T7500, 2,2Ghz, 4MB Cache) , 4GB RAM, HDD Seagate 300 (7200 rpm).

Software installed on the virtual machine was MS Windows XP Professional (with 2GB RAM assigned) and relational database management system Oracle 10g Enterprise (version 10.2.0.1.0). The database instance had the following parameters: percentage allocation of available physical memory was 40% (818 MB), including 584 MB for SGA, 194 MB for PGA and 40 MB for processes, block size was 8192 B.

Additional tools used in tests were [4]:

- PL/SQL – extension of SQL language that supports variables, conditions, loops, exceptions and arrays etc.
- DBMS_RANDOM – package to generate pseudorandom values.
- DBMS_OUTPUT – package to send messages from stored procedures, packages, and triggers..
- DBMS_UTILITY – package that contains programs that perform a wide variety of operations.
- Oracle Wallet Manager – program for creating and manage cryptographic keys of TDE.

### 3.3. Description of Tests

There were 9 variants of tests: testing performance when encryption was not used, testing performance when 3DES168 was used with and without salt, testing performance when AES128, AES192, AES256 was used with and without salt.

The first step for all tests was preparing database structures for tests like creating new table namespace as a database administrator. The next step was creating a new user with correct privileges. After this a new Oracle Wallet was created in default location ($ORACLE_BASE\admin\$ORACLE_SID\wallet) to store encryption keys. In the next step the mentioned 9 tables for the tests were created. All columns were the same in each test, 'id' field was always in a plain form, the remaining fields were encrypted with different method on each test:

- id NUMBER(6) – row identification.
- pass VARCHAR2(25) – ex. password with 25 chars.
- cc NUMBER(15) – ex. credits card number with 15 digits.

The command used for creating tables had different table name and encryption methods parameters for each test (see table). The example for AES 128-bit key and no salt:

```
SQL > CREATE TABLE tde_aes128_salt (
     id NUMBER(10),
     pass VARCHAR2(25) ENCRYPT USING 'AES128' NO SALT,
     cc NUMBER(15) ENCRYPT USING 'AES128' NO SALT
) TABLESPACE <tablespace>;
```

Tab. 1. Variants of tests
Tab. 1. Warianty testów

| Test variant | description |
|---|---|
| plaintext | Table without encryption |
| 3des_salt | Table with encryption using 3DES 168-bit key with salt. |
| 3des_nosalt | Table with encryption using 3DES 168-bit key without salt. |
| aes128_salt | Table with encryption using AES 128-bit key with salt. |
| aes128_nosalt | Table with encryption using AES 128-bit key without salt. |
| aes192_salt | Table with encryption using AES 192-bit key with salt |
| aes192_nosalt | Table with encryption using AES 192-bit key without salt. |
| aes256_salt | Table with encryption using AES 256-bit key with salt. |
| aes256_nosalt | Table with encryption using AES 256-bit key without salt. |

The single test was measuring the time of inserting 10 000 records, modifying 10 000 records and reading 10 000 records. Each of the 9 tests was performed 10 times and the obtained results were averaged.

The testing procedure for insertion included two loops. The external loop was used to repeat the test 10 times, while the inner loop was used to generate pseudorandom values and perform an insert operation into a proper table. The general code of the insertion testing procedure:

```
DECLARE
  v_loops  NUMBER := 10000;
  v_start  NUMBER;
  v_stop NUMBER;
BEGIN
FOR j IN 1 .. 10 LOOP
  EXECUTE IMMEDIATE 'TRUNCATE TABLE <table_name>';
  v_start := DBMS_UTILITY.GET_TIME;
  FOR i IN 1 .. v_loops LOOP
    INSERT INTO <table_name> (id, pass, cc) VALUES (i,
DBMS_RANDOM.STRING('a', 25),
ABS(DBMS_RANDOM.RANDOM));
  END LOOP;
    v_stop:= DBMS_UTILITY.GET_TIME - v_start;
   DBMS_OUTPUT.PUT_LINE (v_stop);
END LOOP;
  DBMS_RANDOM.TERMINATE;
END;
```

The testing procedure for updating rows was similar and included two loops. The external loop for repeating the test 10 times, and the inner loop for generating new pseudorandom values and update operation. The general code of the update testing procedure:

```
DECLARE
  v_loops  NUMBER := 10000;
  v_start  NUMBER;
  v_stop NUMBER;
BEGIN
  FOR j IN 1 .. 10 LOOP
    v_start := DBMS_UTILITY.GET_TIME;
    FOR i IN 1 .. v_loops LOOP
      UPDATE <table_name> SET pass =
DBMS_RANDOM.STRING('a', 25), cc =
ABS(DBMS_RANDOM.RANDOM) WHERE id = i;
  END LOOP;
  v_stop:= DBMS_UTILITY.GET_TIME - v_start;
  DBMS_OUTPUT.PUT_LINE (v_stop);
END LOOP;
  DBMS_RANDOM.TERMINATE;
END;
```

The testing procedure for selecting rows again included two loops. The external loop for repeating test 10 times, and the inner loop for selecting rows. The general code of the select testing procedure:

```
DECLARE
  v_loops  NUMBER := 10000;
  v_start  NUMBER;
  v_stop NUMBER;
  v_pass VARCHAR2(25);
  v_cc NUMBER(15);
BEGIN
FOR j IN 1 .. 15 LOOP
  v_start := DBMS_UTILITY.GET_TIME;
  FOR i IN 1 .. v_loops LOOP
    SELECT pass, cc INTO v_pass, v_cc FROM <table_name>
WHERE id = i;
  END LOOP;
  v_stop:= DBMS_UTILITY.GET_TIME - v_start;
  DBMS_OUTPUT.PUT_LINE (v_stop);
END LOOP;
END;
```

## 3.4. Tests Results

This section contains the results of all tests variations. Table 2 presents times needed for data processing of 10 000 insert operations.

The first test with result 1.92s was obtained without encryption. The time of data processing when columns were encrypted by 3DES with pseudorandom value oscillated between 2.29s to 2.39s, giving the average time needed to write 10 000 tuples of 2.34s. It was about 20% longer than that for plain-text operations. When AES with 128-bit key and salt was used, the time of data processing was in the range 2.38 ÷ 2.46s, which was 23% longer than without encryption. The results for the default TDE encryption method AES with 192-bit key and salt oscillated between 2.45s to 2.51s, giving the average time 2.48s, which was about 0.53s longer compared to the situation when encryption was not used. AES with 256-bit key (which is the strongest available cryptographic algorithm in TDE) gave results with the average time of 2.56s insert operations of 10 000 rows, so 31% longer compared to the processing of the plaintext.

The time overhead associated with the data encryption algorithm can be partially reduced by a non-compliance pseudo-random value in the encryption process. This leads to a reduction in the data security level, because two identical non-confidential data will be transformed in the same cryptogram. In addition, using the pseudorandom value causes the storage overhead mentioned in the previous section.

The time needed to perform write operations of 10 000 rows without using pseudorandom values with 3DES algorithm was 2.17s. It was 9% shorter than using the 3DES algorithm with salt. The average time needed to write data when using AES without salt and 128-, 192-, 256-bit keys was: 2.23s, 2.31s, 2.39s respectively, which is longer by 14%, 18% and 22% when compared to the processing of the plaintext and by 9% shorter then AES with same key length but with pseudorandom value.

Tab. 2. Execution time of 10 000 INSERT operations
Tab. 2. Czas wykonania 10 000 operacji INSERT

| **Test variant** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **avg[s]** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| plaintext | 1,96 | 1,99 | 1,93 | 2,03 | 1,97 | 1,92 | 1,94 | 1,86 | 1,95 | 1,95 | 1,950 |
| 3des_salt | 2,35 | 2,32 | 2,29 | 2,37 | 2,33 | 2,28 | 2,34 | 2,38 | 2,36 | 2,39 | 2,341 |
| 3des_nosalt | 2,19 | 2,22 | 2,16 | 2,21 | 2,21 | 2,09 | 2,12 | 2,16 | 2,15 | 2,19 | 2,170 |
| aes128_salt | 2,46 | 2,39 | 2,44 | 2,38 | 2,39 | 2,41 | 2,37 | 2,41 | 2,42 | 2,39 | 2,406 |
| aes128_nosalt | 2,23 | 2,24 | 2,20 | 2,23 | 2,19 | 2,26 | 2,24 | 2,22 | 2,21 | 2,24 | 2,226 |
| aes192_salt | 2,46 | 2,48 | 2,50 | 2,47 | 2,51 | 2,51 | 2,45 | 2,48 | 2,49 | 2,46 | 2,481 |
| aes192_nosalt | 2,30 | 2,31 | 2,32 | 2,38 | 2,33 | 2,24 | 2,28 | 2,31 | 2,33 | 2,25 | 2,305 |
| aes256_salt | 2,59 | 2,57 | 2,61 | 2,58 | 2,53 | 2,54 | 2,57 | 2,61 | 2,52 | 2,51 | 2,563 |
| aes256_nosalt | 2,44 | 2,31 | 2,39 | 2,40 | 2,43 | 2,38 | 2,38 | 2,44 | 2,33 | 2,35 | 2,385 |

Table 3 presents the times needed for data processing of 10 000 update operations depending on the cryptographic algorithm used. The reference time obtained in update tests, was for the case without encryption and it was 7.60s. The test which gave the longest average duration of 10.02s was the test of AES algorithm with 256-bit key and the pseudorandom modifier (this was a time of 31% longer than the time which is a reference point).

Tab. 3. Execution time of 10 000 UPDATE operations
Tab. 3. Czas wykonania 10 000 operacji UPDATE

| **Test variant** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | avg[s] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| plaintext | 7,65 | 7,60 | 7,64 | 7,61 | 7,59 | 7,60 | 7,59 | 758 | 7,57 | 7,58 | 7,601 |
| 3des_salt | 9,23 | 9,18 | 9,19 | 9,20 | 9,21 | 9,22 | 9,19 | 920 | 9,17 | 9,19 | 9,180 |
| 3des_nosalt | 8,51 | 8,58 | 8,55 | 8,54 | 8,51 | 8,48 | 8,49 | 851 | 8,43 | 8,58 | 8,518 |
| aes128_salt | 9,43 | 9,42 | 9,44 | 9,41 | 9,39 | 9,46 | 9,37 | 943 | 9,41 | 9,42 | 9,418 |
| aes128_nosalt | 8,83 | 8,77 | 8,90 | 8,81 | 8,84 | 8,75 | 8,81 | 880 | 8,83 | 8,76 | 8,810 |
| aes192_salt | 9,73 | 9,69 | 9,67 | 9,81 | 9,69 | 9,72 | 9,72 | 973 | 9,77 | 9,70 | 9,727 |
| aes192_nosalt | 9,12 | 9,01 | 9,05 | 9,00 | 9,08 | 9,09 | 9,10 | 910 | 9,11 | 9,12 | 9,078 |
| aes256_salt | 9,91 | 9,98 | 10,0 | 9,97 | 9,99 | 9,87 | 1,01 | 995 | 10,0 | 9,89 | 9,958 |
| aes256_nosalt | 9,41 | 9,39 | 9,35 | 9,42 | 9,36 | 9,38 | 9,40 | 939 | 9,31 | 9,32 | 9,373 |

Table 4 shows the results of 10 000 read operations when different TDE methods were used. A full reading of the plaintext data lasted from 4.87s to 4.95s.

Executing 10 000 read operations that requires decrypting using the 3DES algorithm with a pseudorandom modifier was about 29% slower and lasted on average 6.37s. Data processing with the same algorithm but without using salt lasted on average 5.93s, which was about 20% slower than reading non-confidential data.

Not using the pseudo-modifier with the AES algorithm improved the reading time by 9% compared to the reading of encrypted data using AES with pseudo-random value.

The average time of reading 10 000 tuples, in which two columns were encrypted using AES with a random modifier and keys with a length of 128 bits, was 6.49s (32% slower than reference time), the key length of 192 bits was 6,71s (36% slower than reference time), the key length of 256 bits was 6.85s (39% slower than reference time).

Tab. 4. Execution time of 10 000 SELECT operations
Tab. 4. Czas wykonania 10 000 operacji SELECT

| Test variant | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | avg[s] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| plaintext | 4,89 | 4,92 | 4,88 | 4,92 | 4,90 | 4,92 | 4,93 | 4,87 | 4,95 | 4,89 | 4,907 |
| 3des_salt | 6,30 | 6,48 | 6,47 | 6,33 | 6,45 | 6,37 | 6,33 | 6,31 | 6,32 | 6,30 | 6,366 |
| 3des_nosalt | 5,98 | 5,96 | 5,99 | 5,91 | 5,92 | 5,90 | 5,88 | 5,95 | 5,93 | 5,88 | 5,930 |
| aes128_salt | 6,51 | 6,54 | 6,49 | 6,50 | 6,39 | 6,49 | 6,51 | 6,50 | 6,48 | 6,47 | 6,488 |
| aes128_nosalt | 6,07 | 6,04 | 6,05 | 6,13 | 6,01 | 6,11 | 6,01 | 6,00 | 6,08 | 6,07 | 6,057 |
| aes192_salt | 6,71 | 6,70 | 6,69 | 6,74 | 6,79 | 6,71 | 6,67 | 6,65 | 6,73 | 6,72 | 6,711 |
| aes192_nosalt | 6,25 | 6,31 | 6,29 | 6,24 | 6,30 | 6,28 | 6,26 | 6,32 | 6,32 | 6,31 | 6,288 |
| aes256_salt | 6,91 | 6,85 | 6,84 | 6,89 | 6,94 | 6,88 | 6,87 | 6,91 | 6,76 | 6,72 | 6,857 |
| aes256_nosalt | 6,46 | 6,50 | 6,49 | 6,37 | 6,38 | 6,48 | 6,46 | 6,39 | 6,44 | 6,45 | 6,442 |

Figure 2 depicts the test results for all cryptographic algorithms and operation types available in TDE.

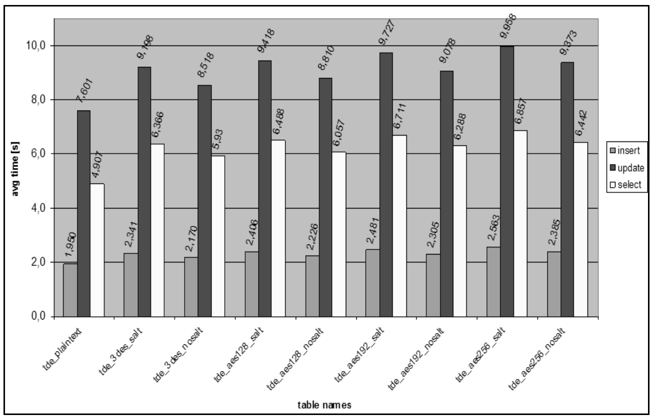

Fig. 2. Average execution times for INSERT, UPDATE, SELECT on 10 000 tuples when different cryptographic algorithms were used
Rys. 2. Średnie czasy wykonania operacji INSERT, UPDATE, SELECT na 10 000 krotkach z wykorzystaniem różnych algorytmów kryptograficznych

## 4. Conclusions

Based on the presented results it may be noted that using cryptography algorithms in database affects its performance. The parameters influenced by use of encryption include execution times of DML operations and SELECT queries on encrypted data as well as the size of disk space needed to store data.

The execution time of data processing for the encrypted data increases with the complexity of the cryptographic algorithms used.

Use of the 3DES cryptographic algorithm resulted in decreased performance during write and update operations by 11% or 20% (depending on whether the pseudo-modifier was used), while the decrease in performance during read operations was 20% or 29% also depending on whether or not pseudorandom values were used.

When the AES with a key 256-bit was used, the strongest method available in the TDE, the execution time needed to perform DML operations increased by almost 22% or 31% and almost 31% or 39% (depending on whether or not the salt were used).

The observed increases in execution times were significant and can be a reason for not using cryptography, even if their use improves the security of stored data.

However, it should be noted that the sensitive data in production systems, which require special security, may only constitute a few percent of the total data processed in the system and so the observed performance loss would only apply to a relatively small amount of data (data processing).

The author wishes to draw attention to the fact that the presented results were just one of many possible cases. The tests presented a situation in which the encryption process was applied to two of the three columns. The database did not undergo a process of tuning and no additional structure (like indexes) of the processed data was applied. Therefore, the results of tests carried out in other environments (with different configurations) can differ from the results presented in this paper.

The essence of the experiment was to study the effect of using different variants of cryptographic algorithms in Oracle 10g system. Therefore the obtained single time results were less important than the differences in the time of data processing between different variations of algorithms.

The advantage of the TDE is the transparency for the end-user. In addition, TDE is responsible for the management of cryptographic material (cryptographic keys). This solution makes it possible to use cryptographic mechanisms without need to modify the application that works with the database. End-users get decrypted data without even knowing that this data was encrypted.

A database administrator should consider potential impact of using cryptography on the parameters of the database system and to consider what is important from the standpoint of the system, the security of stored data or the performance of selected queries.

It should be noted that TDE works only on database server side and this approach poses a potential risk, i.e. the data between the Oracle RDBMS and the remote end user or application is sent through the network in plaintext where data can be captured. Therefore, the author wants to draw attention to this fact. In order to comprehensively protect sensitive data in a distributed environment (SQL*Net) Network Data Encryption and Integrity (element of Oracle Advance Security) should also be used.

The research on the impact of cryptography in a distributed environment goes beyond this work (as well the new TDE features available from Oracle 11g called tablespace encryption). However, in the author's opinion this is a topic worth discussing in subsequent works about the security of data in relational database management systems based on Oracle system.

*This work was partially supported by the Polish Ministry of Education and Science as a 2010-2013 research project.*

## 5. Literatura

[1] Jeloka S.: Oracle Database Advanced Security Administrator's Guide 10g Release 2 (10.2). B14268-02, Oracle, November 2005.

[2] Jeloka S.: Oracle Database Security Administrator's Guide 10g Release 2 (10.2). B14266-04, Oracle, November 2008.

[3] Loney K., Bryla B.: Oracle Database 10g DBA Handbook. Oracle Press, 2008.

[4] Raphaely D.: Oracle Database PL/SQL Packages and Types Reference 10g Release 2 (10.2). Part Number B14258-01, Oracle, June 2005, chapter 24.

*otrzymano / received: 10.09.2010*
*przyjęto do druku / accepted: 01.11.2010* *artykuł recenzowany*