TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Informatics

IDU70LT

Ott Jalakas
143810IAPM

DATA MASKING AND USER RIGHTS IN DATA WAREHOUSE TO PROTECT DATA

Master's thesis

Supervisor: Eduard Ševtšenko
PhD
Associate professor

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDU70LT

OTT JALAKAS
143810IAPM

ANDMETE MASKIMINE JA KASUTAJAÕIGUSED ANDMEAIDAS ANDMETE KAITSMISEKS

Magistritöö

Juhendaja:	Eduard Ševtšenko
	Doktorikraad
	Dotsent

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Signature:

Date:

Abstract

Data masking and user rights in data warehouse to protect data

Working with databases means working with information which means sensitive information. People who are maintaining and developing databases have access to sensitive information which is personally identifiable because they are building the database itself and therefore it is necessary to see the data inside the database. But if it is a case when developing database and data is needed it is commonly taken from production, live, real data. Not generated or meaningless data. If we have a common development process which means development, testing and production environment then we need data also in development and testing environment which leads us to a point that data used in development and testing environment shouldn't be real data so that someone could use it in his/her own interest. This is security and it means that not all data should be available for everybody. This is a place where it is possible to improve security by masking data and managing user rights.

The goal of the thesis is to explain some ways of data masking and user rights management. After that make a conclusion which one of the explained methods would be suitable for a data warehouse and then implement a method of data masking and user rights management in Teradata database system which is a database used for data warehouses.

This thesis is written in English and is 52 pages long, including 4 chapters, 35 figures and 15 tables.

Annotatsioon

Andmete maskimine ja kasutajaõigused andmeaidas andmete kaitsmiseks

Töötamine andmebaasidega tähendab töötamist informatsiooniga, mis omakorda tähendab, et on võimalik, et tegmist on ka tundlike andmetega. Inimesed kes haldavad ning arendavad andmebaase omavad ligipääsu tundlikele andmetele mille põhjal on võimalik identifitseerida isikuid sest nad ise ehitavad seda andmebaasi, seega on oluline ja elementaarne, et nad näevad andmeid andmebaasis. Üldjuhul kui arendatakse andmebaasi siis kasutatakse andmeid otse toodangubaasist, kus on reaalsed ehk õiged andmed, mitte suvalised, genereeritud andmed. Kui on tegemist tavapärase arendusprotsessiga, mis tähendab hõlmab arendus-, testimis- ja toodangkeskkonda siis on vaja andmeid arendus- ja testimiskeskkonda, mis omakorda viib meid faktini, et arendus- ja testimiskeskkonnas ei peaks olema reaalsed ehk toodangkeskkonna andmed, mida keegi võib kasutada oma huvides ära. See on turvalisus ehk kõik andmed ei peaks olema kõikidele saadavad. See on koht kus on võimalik turvalisust tõsta maskeerides andmeid ning hallates kasutajate õigusi.

Selle töö eesmärk on kirjeldada mõningaid olemasolevaid viise kuidas maskeerida andmeid ning hallata kasutajate õigusi. Peale seda on välja pakkuda andmeaidale sobiv meetod ning siis implementeerida näide andmete maskimiseks ning kasutajaõiguste haldamiseks Teradata andmebaasisüsteemis, mida kasutakse andmeaitades.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 52 leheküljel, 4 peatükki, 35 joonist, 15 tabelit.

List of abbreviations and terms

DWH	Data warehouse
TUT	Tallinn University of Technology
Database	Collection of organized information
Schema	Structure of database
Data mart	Access layer of data warehouse, used to get data out to the users
CPU	Central Processing Unit

Table of contents

List of figures

List of tables

1 Introduction

1.1 About DWH

Data warehouse is a system used for reporting and data analyses. In this work I am thinking specially of an enterprise data warehouse. DWH is like a central database system in enterprise where there can be several different or separate databases. For example if we take a bank then there are separate databases for loans, leasing, customers, cards data. In Figure 1 is shown a basic example of different databases combined to one common data warehouse.

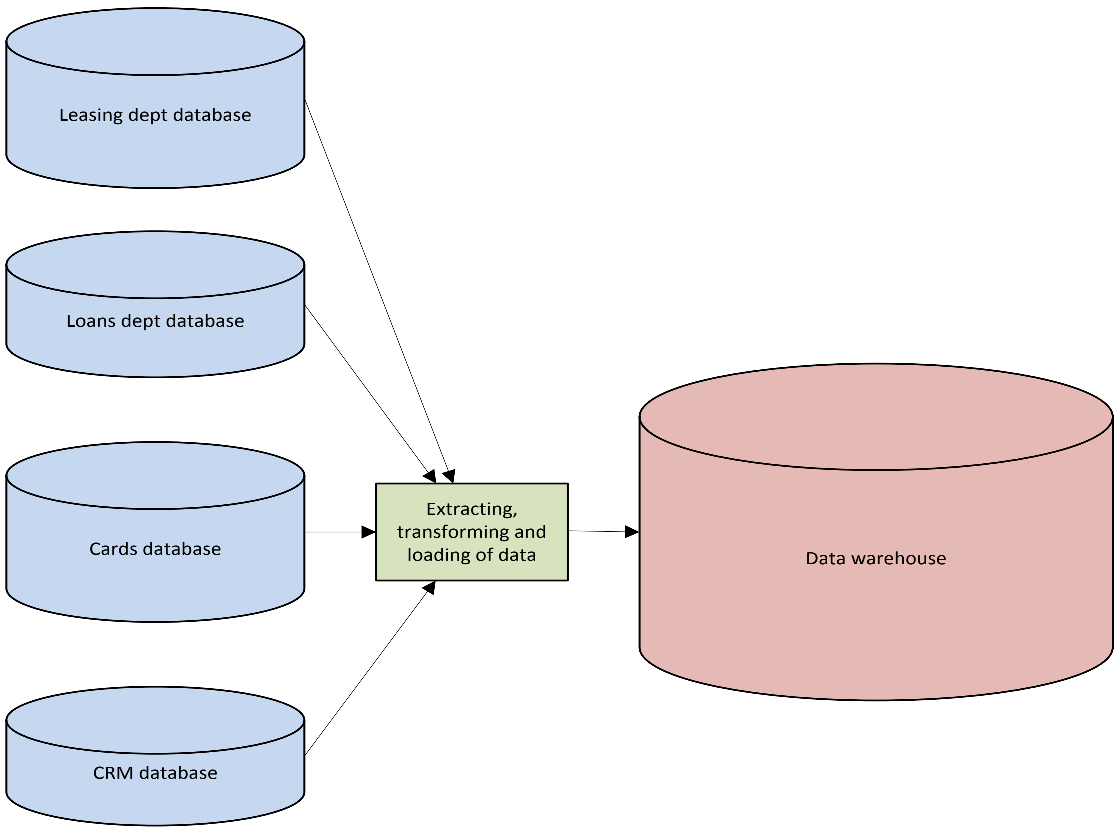


Figure 1. Source database to data warehouse.

Now if the bank has to report something to central bank or want to do some analysis and data mining then this separate department database data needs to be moved to one common database to have the data available in one common place. It simplifies very much reporters and analysts job. The data is historical because data is loaded from source systems to DWH for example the next day but this depends mostly on the architecture of source and warehouse database systems and also tools used for extracting, transforming and loading data.

A common data warehouse architecture is using multi layered schemas so that there is the main layer of tables where is kept all the data physically. The main layer is used mostly by database developers and tools which are moving and transforming data. On top of main layer is built a layer which includes views. Each view has basically one to one relation with main model table. That layer is called presentation layer and is used by business users, people who actually need this data, analysts and reporters. Also there can be separate layers built on presentation layer for special purposes like tables for reporting data to financial institutions or for data mining. Described warehouse data layer structure is shown on Figure 2.

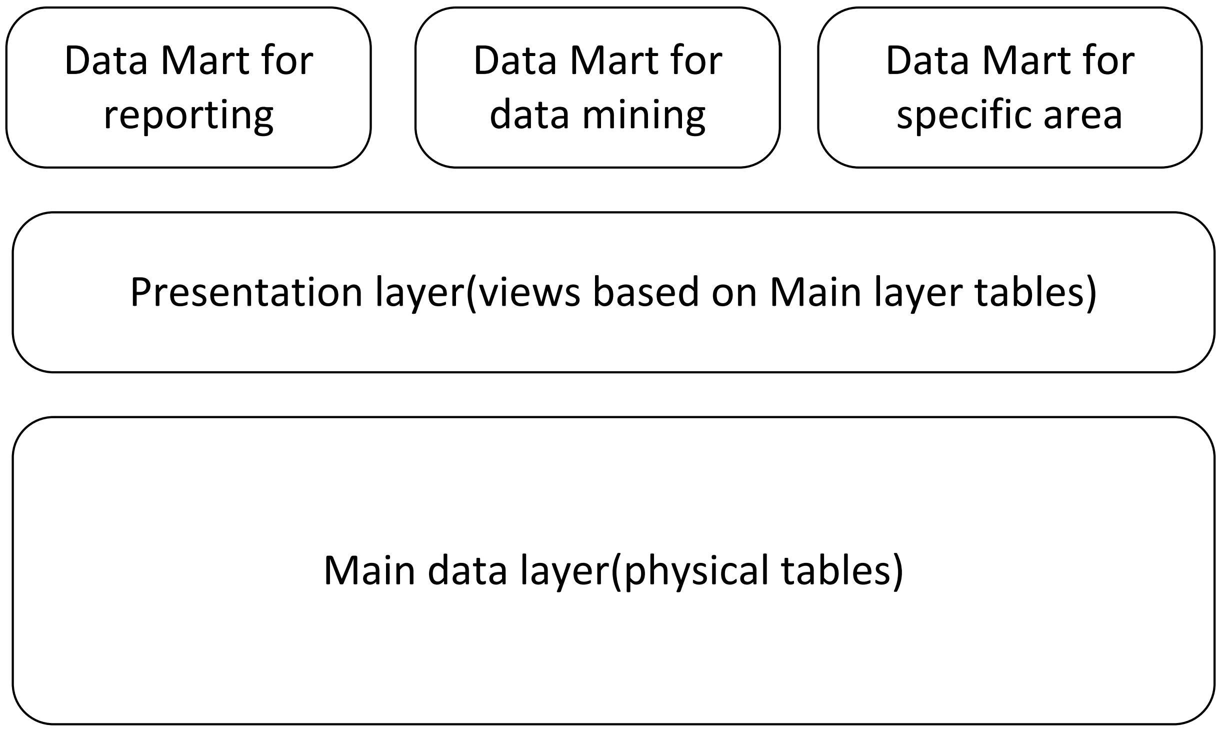

Data Mart for reporting
Data Mart for data mining
Data Mart for specific area
Presentation layer(views based on Main layer tables)
Main data layer(physical tables)

Figure 2. Data warehouse layer structure.

1.2 Data security in DWH

Data security is critical for all organizations and also home users. Personal files, bank account details or client information is such data which is hard to replace without backup. If the data is lost due to some disasters like flood or fire then we can describe it as bad luck but if the data is lost to hackers then the consequences are much worse. Securing database means knowing the risks which can happen physically or virtually, caused by an earthquake, floods, fires, electricity loss, hackers who are trying to get through firewall, a bug in software or just simply when database administrator gets sick. It is also very important to have competent people who are working with data by training and giving instructions how they should work and deal with data. All these cases need to have a plan: what to do when there is a security risk.

Securing data in DWH is complicated mostly because of a very large system which is used by hundreds of users. As DWH is holding data for basically everything in organization then it can contain sensitive information which is, for example personal information of customers and employees. In different countries there are laws for privacy how personal information can be used and presented. For example in Estonia there is an act called "Personal Data Protection Act" [1] which means violating the obligation to register the processing of sensitive personal data or violating requirements regarding security measures to protect personal data or violating other requirements for the processing of personal data if violated, it is punishable by a fine of up to 32000 euros for legal person and 300 fine units for private person. These rules must be followed strictly as they are to protect all private and legal persons but also the company which database is dealing with sensitive personal data.

The following are sensitive personal data [1]:

1. data revealing political opinions or religious or philosophical beliefs, except data relating to being a member of a legal person in private law registered pursuant to the procedure provided by law;
2. data revealing ethnic or racial origin;
3. data on the state of health or disability;
4. data on genetic information;
5. biometric data (above all fingerprints, palm prints, eye iris images and genetic data);

6. information on sex life;
7. information on trade union membership;
8. information concerning commission of an offence or falling victim to an offence before a public court hearing, or making of a decision in the matter of the offence or termination of the court proceeding in the matter.

Each reporter and analyst who has access to data warehouse should have access only to the data he or she needs for working. This means managing access to DWH by creating roles for different layers and objects to access for each employee role. If we are talking about database developers who are working in development environment and testers who are working in test environment then they see basically all DWH data because they must make sure that the system works technically and the data which is in database is correct. It is very common that data for development and test environment is taken from production database because real data is the best to work with. But in this case all the sensitive, personally identifiable information is available in test or development environment.

What to do, so that sensitive information would not be understandable and the data would not be personally identifiable, yet database could be used with real data? For this problem it is possible to use different kind of data masking methods and managing user rights correctly, which is the topic which will be described in this writing.

1.3 The task of this thesis

The following tasks are accomplished as the result of this thesis:

1) We describe the problems with data security in data warehouses.

2) We describe data masking methods, compare them, propose a solution and implement an example of data masking in Teradata database system.

3) We describe user rights management in data warehouse, propose a solution and implement it on databases created in point 2.

2 Data masking

2.1 Why is data masking needed in DWH

When we have data which is sensitive and personally identifiable then this data needs special attention because it is vulnerable for data breaches. Data breach can happen by system error or malfunctioning or when someone is stealing data as an employee having access to data or a thief who is attacking from outside the organization. To increase the risk of data breaches it is possible to use data masking methods.

Data masking in DWH is most commonly needed for creating test data. Test data because in non-production environments employees have usually greater rights to work with data and also greater access to data. For example testers access usually all data because testers can’t test if all data is not available. So if testers are curious or are having an interest in using the data for their own good and copying the data and taking out of the organization and then the data is accidentally leaked means a loss of money and trust from clients and also penalties. Losing clients respect is probably the biggest loss in case of data breach because it is known that up to 40% of customers would consider discontinuing their relationship with a company who exposed their personal data [2].

Masking data gives also better overview where is sensitive and personally identifiable data and who are using it. If we register sensitive info then it is possible to maintain the usage of this data. Having an overview of sensitive information and possibility to maintain the usage of sensitive data should be must have in every organization because it must be ensured that sensitive info is stored and secured according to laws of privacy and regulations. Also knowing who uses sensitive data helps to prevent data breaches by inside users who might use the data for their own good. Dealing after data breach needs additional work and resources, human and financial:

- Technical overview and organizational audit why data breach happened
- Possible new development to fix the reason why data breach happened
- Explaining to media and customers the reason of data breach
- Regaining organization reputation
- Fines and legal judgements

In table 1 below are described positive and negative aspects of data masking

Positive	Negative
Increase the possibility of data breaches	Needs additional resource to develop and maintain
Follow the laws of using of personal data	Takes additional time in existing process
Create structurally similar but not personally identifiable data for test and development environments	Negative impact to performance

Table 1. Data masking positive and negative.

2.2 Data masking possibilities

There are different ways how to mask data as masking is an operation how to change the original value into something else. Masking also depends on the type and need for data. For example values can be substituted or replaced, redacted by just showing for example * for value, shuffled by shuffling individual values in a column, blurred by taking a value and turning it into a certain range of values, averaged, tokenized by replacing data with random elements.

In table 2 below are listed masking techniques that I know and have read about:

Technique	Description
Shuffling	Values in column are shuffled randomly
Randomizing	Generating random values
Hiding	Hiding value completely by using views
Substitution	Each character or number is replaced by a given value

Scrambling	Value has some part scrambled with a symbol
Blurring	Turning a value into a certain range of values
Encryption	Value is encrypted with some encrypting algorithm using a secret key

Table 2. Data masking techniques.

2.2.1 Data masking by using views on physical tables

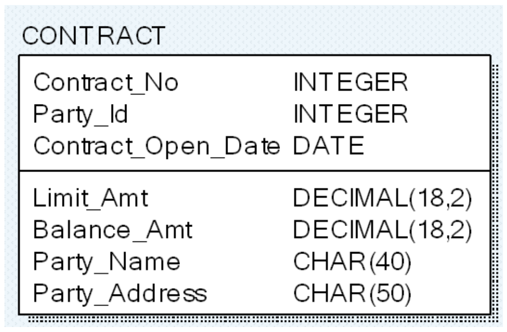
When we need to hide some data from users we can create a layer on top of main data layer by creating views on top of physical tables. In view we can define which table we are using and how and what columns we select from table. An easy solution to hide data from users is by creating views without columns which must be hidden. To do this we must know that data for other columns would be still usable after that. For example if we hide amount value from transaction data it is probably no use. But when we hide clients social security number from transaction table it is probably acceptable. Hiding values need analyze of the data and certain requirements to understand the need of it.

For example there is CONTRACT with following columns, see table 3.

Contract_No	Party_Id	Contract_Open_Date	Limit_Amt	Balance_Amt	Party_Name	Party_Address

Table 3. Contract columns.

Model of the table:



CONTRACT

Contract_No	INTEGER
Party_Id	INTEGER
Contract_Open_Date	DATE
Limit_Amt	DECIMAL(18,2)
Balance_Amt	DECIMAL(18,2)
Party_Name	CHAR(40)
Party_Address	CHAR(50)

Figure 3. Contract model.

Business users requests for certain people to have access for all columns except Party_Name and Party_Address. Then we should create a view with access to certain people only which means that those certain people do not have access to table but only view. In this view will be selected all columns except Party_Name and Party_Address:

```
CREATE VIEW CONTRACT_VW AS
      SELECT Contract_No,
            Party_Id,
            Contract_Open_Date,
            Limit_Amt,
            Balance_Amt,
      FROM CONTRACT;
```

Figure 4. Create view script for contract_vw.

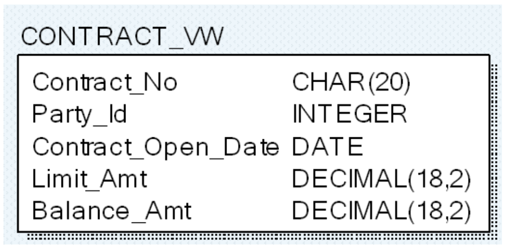
With views can be used also redaction method as replacing some part of value with * for example. This method is called scrambling. For this can be used replace and substring functions by selecting certain part of value and replacing it with meaningless values.

For example we have a column 10 characters long and we have a requirement to replace first 5 characters with *. In view we can use following solution:

```
CREATE VIEW CONTRACT_VW AS
      SELECT '*****' || substr(CAST(Contract_No AS CHAR(20)),5)
      AS Contract_No,
      Party_Id,
      Contract_Open_Date,
      Limit_Amt,
      Balance_Amt,
FROM CONTRACT;
```

Figure 5. Create view script for contract_vw, scrambling.

Model of created view see figure 6.



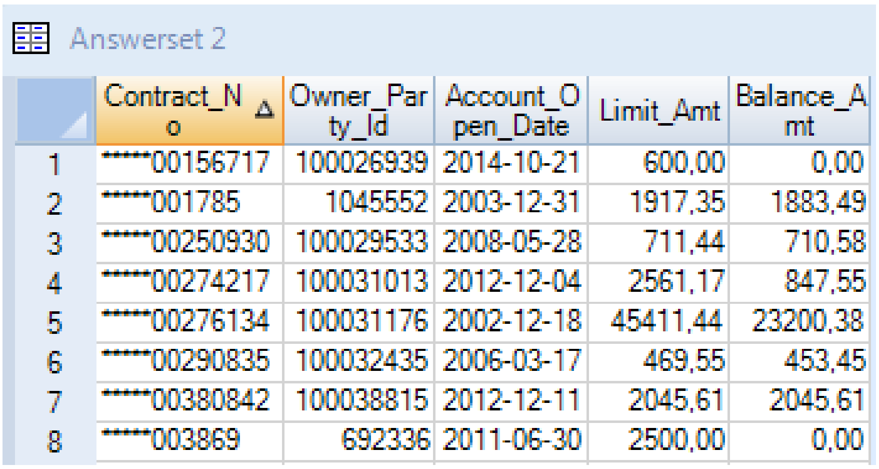
CONTRACT_VW

Contract_No	CHAR(20)
Party_Id	INTEGER
Contract_Open_Date	DATE
Limit_Amt	DECIMAL(18,2)
Balance_Amt	DECIMAL(18,2)

Figure 6. Model of contract_vw.

In this example we selected Contract_No from table CONTRACT. First five letters are * and then is added a substring of value from fifth character to the end.

In Figure 7 is shown how data will look like after scrambling.


Answerset 2

	Contract_No	Owner_Party_Id	Account_Open_Date	Limit_Amt	Balance_Amt
1	******00156717	100026939	2014-10-21	600,00	0,00
2	******001785	1045552	2003-12-31	1917,35	1883,49
3	******00250930	100029533	2008-05-28	711,44	710,58
4	******00274217	100031013	2012-12-04	2561,17	847,55
5	******00276134	100031176	2002-12-18	45411,44	23200,38
6	******00290835	100032435	2006-03-17	469,55	453,45
7	******00380842	100038815	2012-12-11	2045,61	2045,61
8	******003869	692336	2011-06-30	2500,00	0,00

Figure 7. Data after scrambling in contract_vw.

2.2.2 Data masking in MSSQL database system using mapping of replaceable values

When searching for articles about MSSQL database masking solution I found a lot of software which could be integrated with database but these solutions cost and need external support. So I searched for a solution which could be implemented in database without needing any external support. I found an article by Rick Dobson from website called www.mmsqltips.com with a title called “Masking personal identifiable SQL Server data” [3].

The basic idea of this solution is to use a mapping table where is defined value for each replaced character. For example if we have social security number which in Estonia is 11 number long integer type number then we store for each position a replaceable value for each number. For example first 2 numbers are 82 and in masking table for position 1 and number 8 we replace it with number 2 and for position 2 number 2 we replace it with 0 and we have number 20 as masked value for number 82 in this example.

Example of mapping table, see table 4.

PositionNumber	OriginaValue	MaskValue
1	0	4
1	1	3
1	2	2
1	3	5
1	4	6
1	5	1
1	6	2
1	7	0
1	8	2
1	9	9
2	0	5
2	1	4
2	2	0
2	3	9
2	4	6
2	5	3
2	6	4
2	7	6
2	8	5
2	9	8

Table 4. Mapping table.

This kind of mapping table works for social security, telephone, address numbers, also it can be used in dates but for character strings this table gets much bigger in that case.

The use of this table should be very restricted because with this table data can be unmasked and it would be no use if everyone can see it. So it should be accessible basically only by database administrator who creates this table, adds mapping values and the procedure which uses the table to mask data.

To start actually masking data with mapping table there has to be created a procedure which takes the selected table column and replaces each value with a value from mapping table. Procedure should use as input the value which needs to be masked and procedure should return after masking the masked value. Procedure can be called for each value then. For this should be done another procedure which goes through all values in database column and updates the value in target column.

This method needs some manual work preparation and data analysis to make sure that masking works for each masked value but still this method can be automated for data generation in test and development environments for example.

2.2.3 Data masking in Oracle database system using DBMS_REDACT package

Arup Nanda has written an article with title “Hide from Prying Eyes” which was published in Oracle Magazine January/February 2014 [4]. In this article he is describing a solution for Oracle database system version 12c for hiding sensitive data automatically by using data redaction. This is meant to be used specially on production environment but also other. Using this solution can be possible to mask the data for example in production environment and export it to test environment to create test data. I will give a brief overview how it is done.

The goal in this particular article is to hide sensitive data for visitors and this must not be done by user interface tools but on database level. Data in tables must remain intact but information which is shown must be redacted. Arup Nanda says that traditionally these requirements mean creating views on tables and assigning user certain privileges to have access only to created view. But this kind of solution is complex, error-prone and subject to performance issues. There is a feature in Oracle database security for data redaction and this is the way data will be redacted.

Data is stored in schema named TSBS and data where the data must be redacted is called SAVINGS. The structure of the table SAVINGS is described in table 5.

Name	Type
ACCNO	NUMBER
ACCNAME	VARCHAR2 (20)
ID_NO	VARCHAR2 (9)
LAST_DEP_DT	DATE
FOLIOID	NUMBER
EMAIL	VARCHAR2 (200)

Table 5. Table SAVINGS structure.

Requirements for masking are that when a certain user named TSBS selects from table then all values must be shown but when any other user selects from table then following values must be masked:
ID_NO – Social Security Number which first five numbers must be hidden with *;
LAST_DEP_DT – Date of last deposit, show only day and month and replace year with 1900;
FOLIOID – Replace with any random number;
EMAIL – Replace name before @ with x-s and keep the domain name.
After requirements it is explained how these rules are traditionally applied:

Redaction is done in application which means complexity in application and means changes needs to be done in application also when changes are done in database level.

A view is created on table, certain masking is done based on columns and user is given right to access the view. In this case when user wants to modify the data it is not possible because inserting, deleting, modifying a view is not possible. This means creating triggers and means more job and maintaining.

But for Oracle 12c there is a feature to create a set of rule for redacting data on table level. The set of rules are called redaction policy. There is a PL/SQL package DBMS_REDACT for this in Oracle 12c that is for creating and maintaining policies on a table.

In figure 8 is an example script how to create and apply policies on table columns for certain users.

```
begin
    dbms_redact.add_policy (
         object_schema => 'TSBS',
         object_name => 'SAVINGS',
         policy_name => 'Savings_Redaction',
         expression => 'USER!=''TSBS''',
         column_name => 'ID_NO',
         function_type => dbms_redact.partial,
         function_parameters     => 'VVVVVVVVV,VVVVVVVVV,*,1,5'
     );
     -- subsequent columns will need to be added
     dbms_redact.alter_policy (
        object_schema => 'TSBS',
        object_name => 'SAVINGS',
        policy_name => 'Savings_Redaction',
        action  => dbms_redact.add_COLUMN,
        column_name => 'FOLIOID',
        function_type => dbms_redact.random
     );
     dbms_redact.alter_policy (
        object_schema => 'TSBS',
        object_name => 'SAVINGS',
        policy_name => 'Savings_Redaction',
        action  => dbms_redact.add_COLUMN,
        column_name => 'LAST_DEP_DT',
        function_type => dbms_redact.partial,
        function_parameters     => 'MDy1900'
     );
     dbms_redact.alter_policy (
        object_schema => 'TSBS',
        object_name => 'SAVINGS',
        policy_name => 'Savings_Redaction',
        action  => dbms_redact.add_COLUMN,
        column_name => 'EMAIL',
        function_type => dbms_redact.regexp,
        regexp_pattern => dbms_redact.re_pattern_email_address,
        regexp_replace_string => dbms_redact.re_redact_email_name,
        regexp_position => dbms_redact.re_beginning,
         regexp_occurrence => dbms_redact.re_all
     );
  end;
```

Figure 8. Oracle script for creating and applying policies

Procedure add_policy creates a policy on SAVINGS table column ID_NO for all users expect TSBS. Function type for column ID_NO is marked as partial because we want to replace first five numbers with * which is defined in function parameters. With procedure alter_policy s possible to add more columns to be redacted. As for column FOLIOID it was needed to replace values with random number here is used function type random. For column LAST_DEP_DT we needed to replace year part of date with 1900 so here we are using parameter MDy1900, capital M and D means that they should not be redacted and small y means that year should be replaced with 1900. For column EMAIL is used regex functions to replace email name before @ letter.

Now if we select data from table SAVINGS as user TSBS then we see data shown in figure 9.

```
SQL> conn tsbs/tsbs

SQL> select * from tsbs.savings;

ACCNO ACCNAME    ID_NO     LAST_DEP_DT FOLIOID EMAIL
_____ __________ _________ ___________ _______ ____________________

  101 John Smith 123456789 21-SEP-13   1234567 john.smith@proligence.com
  102 Jane Smith 234567890 20-SEP-13   2345678 jane.smith@proligence.com
  103 Jane Doe   345678901 19-SEP-13   3456789 jane.doe@proligence.com
```

Figure 9. Data from table SAVINGS as user TSBS

And when we select data from table SAVINGS as some other user like app then we see masked data shown in figuure 10.

```
SQL> conn app/app

SQL> select * from tsbs.savings;

ACCNO ACCNAME    ID_NO     LAST_DEP_DT FOLIOID EMAIL
_____ __________ _________ ___________ _______ ____________________

  101 John Smith *****6789 21-SEP-00   3434562 xxxx@proligence.com
  102 Jane Smith *****7890 20-SEP-00   3452092 xxxx@proligence.com
  103 Jane Doe   *****8901 19-SEP-00   4529012 xxxx@proligence.com
```

Figure 10. Data from table SAVINGS as user app.

Creating these policies makes data masking quick and easy to manage the formats of masked columns as we have possibility to mask data with different techniques. For user who is allowed to see data without masking is all the same and nothing is changed. This is done all on table level and not by creating views or doing masking in application. This solution in Oracle database is very good and works well. Also there are many more possibilities how to mask data using different function types which are all listed in oracles documentation:

http://docs.oracle.com/database/121/ARPLS/d_redact.htm#ARPLS73800

2.3 Comparison of existing data masking realizations

When reviewing these 3 different ways how masking has been solved by creating views on physical tables, creating mapping table by using replacing technique and Oracles built in function with many different type of masking functions then it is quite clear that Oracles data redaction package is the best for this usage but when we are not using Oracles database system then it is no use.

All these realizations need first analyze which data needs to be masked and how it should be masked. Now we need to know for what we need this masking for. If it is for creating test data for test environment then we can’t do hiding of data or replacing with * because the data needs to be functional. In this case we should use mapping table by replacing sensitive values. When we need masking for hiding data from users then best solution is Oracles data redaction package but if we are not using Oracles database system then a good solution is also using views built on physical tables and then hiding or scrambling sensitive columns. When we are dealing with data warehouse it means huge data amounts and in this case we need to make sure that the way of masking we are using would also perform. Creating views on tables means a lot of manual work and if we need to do it for all tables it takes time, also it increase complexity of system by creating another layer. Creating a mapping table and a procedure for masking data will probably take more resource because replacing data in big tables is time consuming but this is quite well working solution because if we have no extra layers and we have fully functional dataset where sensitive column data is not actual data then it is the result we actually needed for creating test data without sensitive information.

In the tables listed below are described positive and negative aspects of described masking methods:

Data masking by using views on physical tables:

Positive	Negative
Separate object with only allowed data	Increases complexity of as basically duplicating objects
Suits well for hiding certain columns	Needs additional resource from database

Easy to produce	Does not suit for generating test data

Table 6. Positive and negative aspects for data masking using views on physical tables.

Data masking using mapping of replaceable values:

Positive	Negative
When realization is finished easy to execute	Additional step in data generation to test environment
Suits well for hiding values and generating test data	Time consuming

Table 7. Positive and negative aspects for data masking using mapping of replaceable values.

Data masking using Oracles data redaction package:

Positive	Negative
Easy to create rules	Need to define each user for policy
Easy to maintain	Only available for Oracle database systems
Suits well for hiding and scrambling values	

Table 8. Positive and negative aspects for data masking using Oracles data redaction package.

2.4 Using data masking in Teradata database system

As mentioned before I will be thinking of a data warehouse in a financial company where is used Teradata database system. It is very important first to know what kind of system we have architecturally. In basic there are 3 environments: development, test and production. Test and development environment are basically a copy of production but with much limited data amounts. For example in production there is data from 2005 and to test and development environment we bring data only for last 2 years. Development environment and test environment structures and data should be synchronized with production quite often. Synchronization means copying data from production backup to another environment in this case test or development. If we are taking a big financial company where usually is used waterfall process then synchronization can happen every month. Described data warehouse has the main physical layer and on top of that is built presentation layer where every view is basically with a 1 to 1 relation with physical table from main data layer. Masking data requires first to collect and analyze which data in system is sensitive and personally identifiable, after that we can think about masking the data. When knowing what data columns needs to be masked it is very important to do impact analyze: where columns are used in query result sets, join keys, where and join conditions. And after impact analyze it is possible to decide the masking pattern or methods what to use. In this scenario I can see 2 ways how to mask data.

First one is to create for production environment a security layer where in each object is created a view where are taken out all sensitive columns. Doing this requires that sensitive columns are not used as primary or unique keys. But this is basically a must be case always when masking data. When building a system we should not use sensitive data in primary and unique keys.

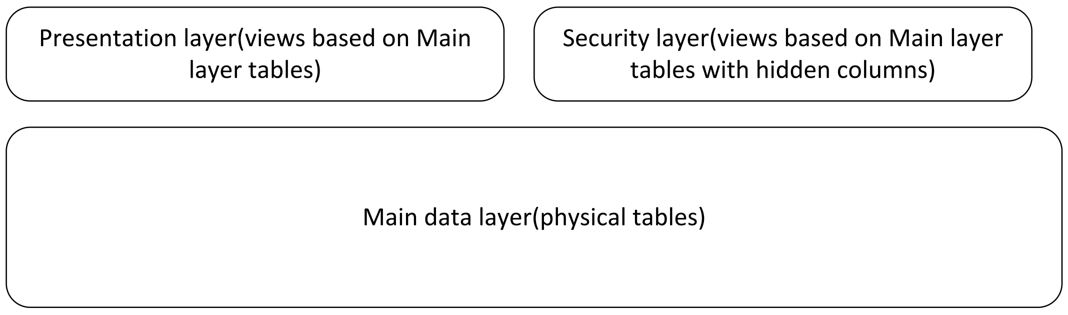

Presentation layer(views based on Main layer tables)
Security layer(views based on Main layer tables with hidden columns)
Main data layer(physical tables)

Figure 11. Security layer next to presentation layer.

Second way to use masking in data warehouse is to use mapping and replacing method combined with scrambling and random methods to mask sensitive data which is synchronized to test and development environments from production environment. This requires rules how data will be mapped and replaced. As data is taken from production environment backup and is copied to test environment we should consider the way how it will be done because amounts of data are big in warehouse kind of systems and also it should be done as automatically as possible. We need to create mapping tables and procedures which will map and replace the data with restricted access in target environment. Data should be first copied from production database backup to target environment. In that moment no users should have access to database. Then after data copying should be executed procedures for masking data. I expect that in current system the process for getting data to test environment already exists and adding to that process a step for masking data is described in figure 12.

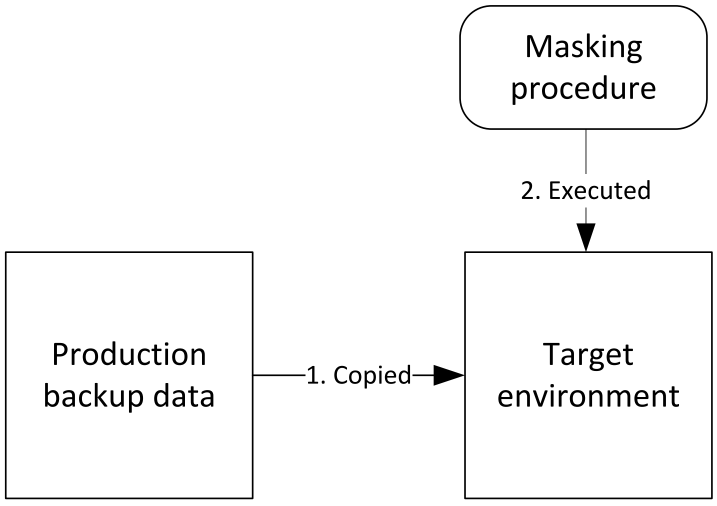


Figure 12. Process for preparing test/development environment data.

2.5 Example of data masking in Teradata database system

In this paragraph will be created an example of financial institutions data warehouse with some example data. The example database is set up on my personal computer using VMware 12 Workstation Player to create a virtual machine with operating system Suse Linux where is installed Teradata Studio Express 14.0. Information how to install it was founded from Teradata forum [5]. The reason why I am using Teradata database system is because I am working with Teradata database system myself in a financial organization and I am more familiar with the syntax and systems specifics compared to other data warehouse database systems.

First will be described the data model of physical table layer where the data is like in production environment. The model is very simplified comparing to actual data warehouses but the aim of this work is creating a working masking solution not creating a warehouse model.

2.5.1 Views on physical tables in different schema

The process should be following:

1. Impact analyze on masked columns
2. Developer creates scripts for installation
3. Database administrator installs scripts

Second point will be described in detail. We will implement a layer on top of main layer. This is meant specially for business users in production environment.

We have a physical layer called financial where are 4 tables: customer, accts, credit_acct and credit_tran. In figure 13 are described physical table models.

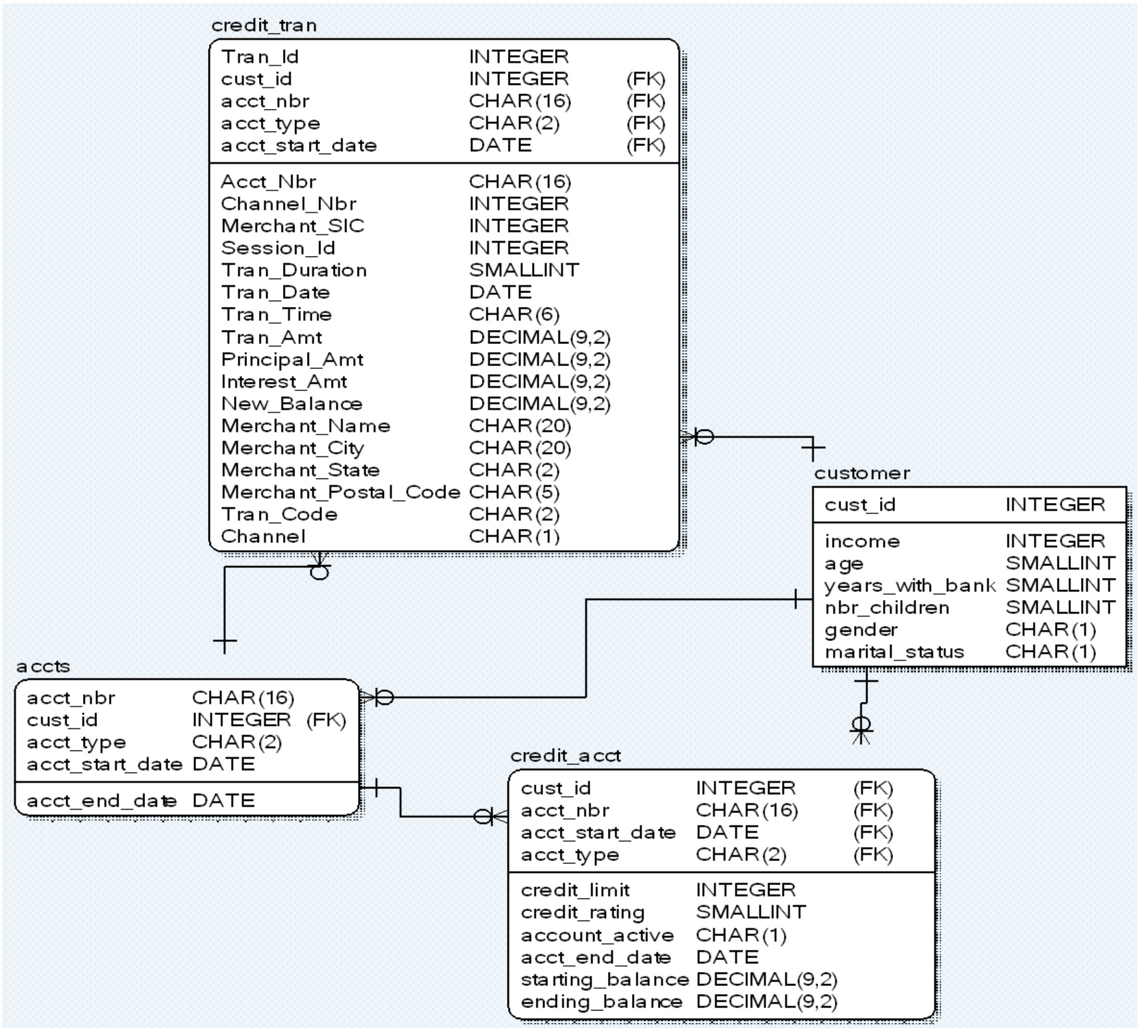


Figure 13. Model of financial layer physical tables.

Physical table generation scripts are in Appendix 1.

We have requirements to hide following columns described in table 9.

TableName	ColumnName
customer	age
customer	nbr_children
customer	gender
customer	marital_status
credit_tran	Merchant_SIC
credit_tran	Merchant_Name
credit_tran	Merchant_City
credit_tran	Merchant_State
credit_tran	Merchant_Postal_Code

Table 9. Requirements for hiding columns

Model of the views look like described in figure 14.

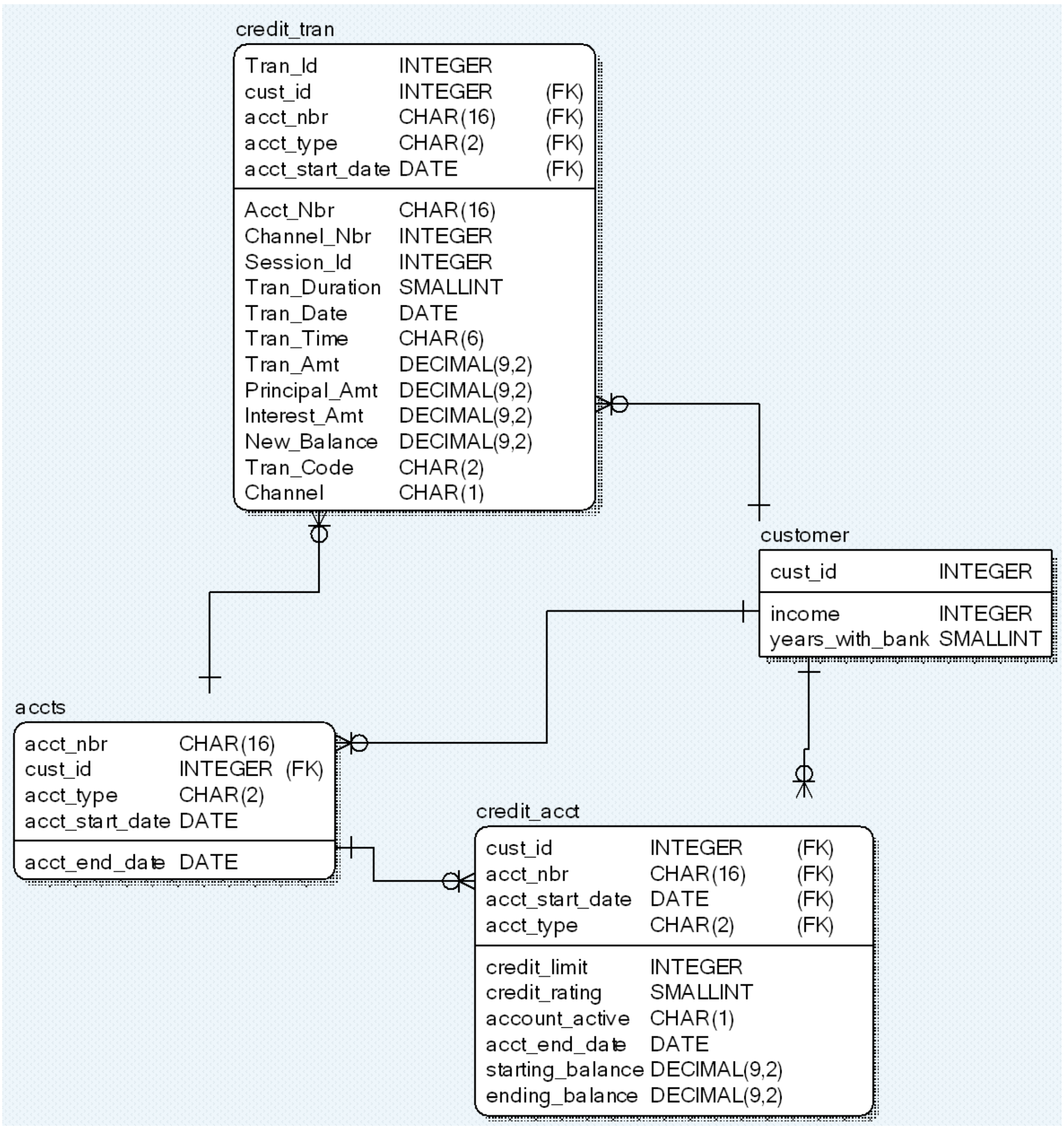


Figure 14. Model of views.

To hide these columns we create a new schema called security shown in figure and create for each table in financial schema a view without columns that were requested to hide shown in figure .

```
CREATE VIEW security.accts_VW
AS
SELECT cust_id
       ,acct_type
       ,acct_nbr
       ,acct_start_date
       ,acct_end_date
FROM financial.accts;
CREATE VIEW security.customer_VW
AS
SELECT cust_id
       ,income
       ,years_with_bank
FROM financial.customer;
CREATE VIEW security.credit_acct_VW
AS
SELECT cust_id
       ,acct_nbr
       ,credit_limit
       ,credit_rating
       ,account_active
       ,acct_start_date
       ,acct_end_date
       ,starting_balance
       ,ending_balance
FROM financial.credit_acct;
CREATE VIEW security.credit_tran_VW
AS
SELECT Tran_Id
       ,Cust_Id
       ,Acct_Nbr
       ,Channel_Nbr
       ,Session_Id
       ,Tran_Duration
       ,Tran_Date
       ,Tran_Time
       ,Tran_Amt
       ,Principal_Amt
       ,Interest_Amt
       ,New_Balance
       ,Tran_Code
       ,Channel
FROM financial.credit_tran;
```

Figure 15. Script for creating views in security schema.

```
CREATE DATABASE security AS PERMANENT = 10000
       ,spool = 10000
       ,fallback protection;
```

Figure 16. Script for creating new security schema.

Also we need to grant select rights for view shown in figure 17.

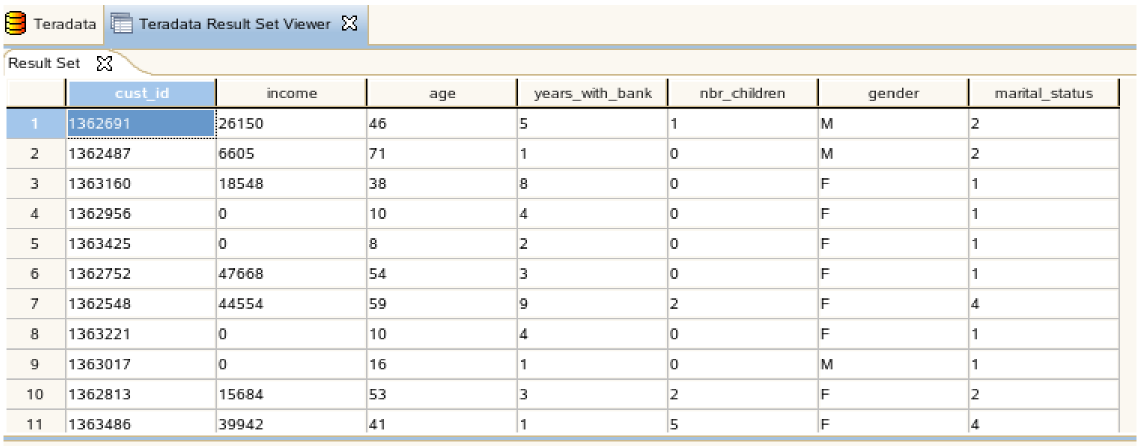
```
GRANT SELECT ON security to user;
```

Figure 17. Grant user rights to select security layer objects.

Now when we do a select from physical table financial.customer we should see all columns with data inside with query shown in figure 18. Data shown in figure 19.

```
SELECT *
FROM financial.customer;
```

Figure 18. Script to select data from table customer.



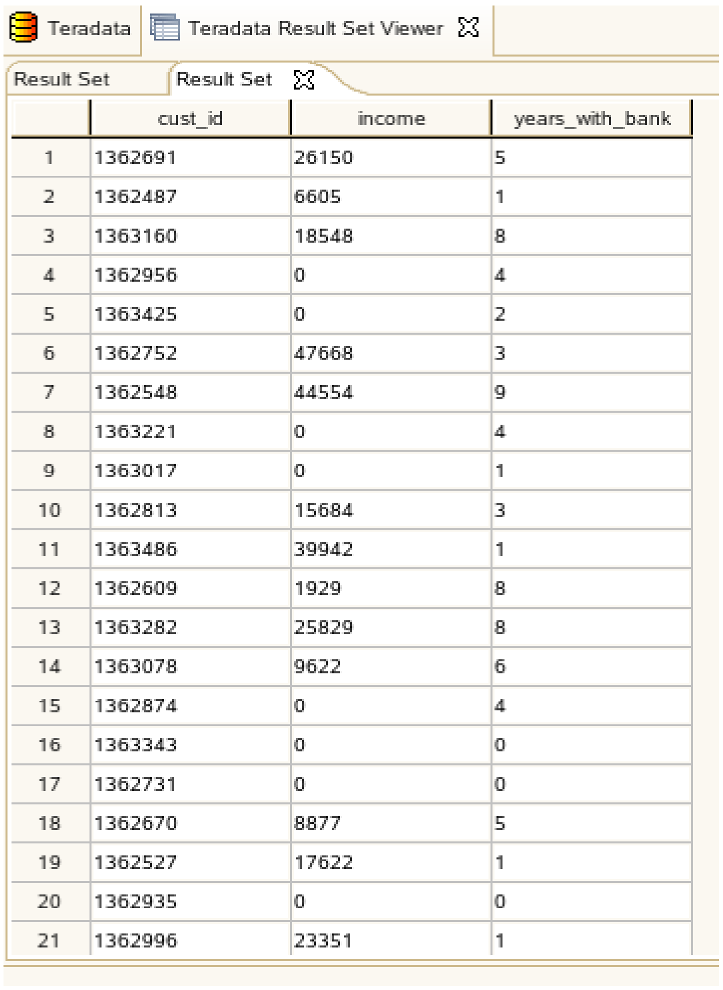
	cust_id	income	age	years_with_bank	nbr_children	gender	marital_status
1	1362691	26150	46	5	1	M	2
2	1362487	6605	71	1	0	M	2
3	1363160	18548	38	8	0	F	1
4	1362956	0	10	4	0	F	1
5	1363425	0	8	2	0	F	1
6	1362752	47668	54	3	0	F	1
7	1362548	44554	59	9	2	F	4
8	1363221	0	10	4	0	F	1
9	1363017	0	16	1	0	M	1
10	1362813	15684	53	3	2	F	2
11	1363486	39942	41	1	5	F	4

Figure 19. Data shown in table customer.

And from security view we can see only columns cust_id, income and years_with_bank with query shown in figure 20. Data shown in figure 21.

```
SELECT *
FROM security.customer_VW;
```

Figure 20. Script to select data from view customer_vw.



	cust_id	income	years_with_bank
1	1362691	26150	5
2	1362487	6605	1
3	1363160	18548	8
4	1362956	0	4
5	1363425	0	2
6	1362752	47668	3
7	1362548	44554	9
8	1363221	0	4
9	1363017	0	1
10	1362813	15684	3
11	1363486	39942	1
12	1362609	1929	8
13	1363282	25829	8
14	1363078	9622	6
15	1362874	0	4
16	1363343	0	0
17	1362731	0	0
18	1362670	8877	5
19	1362527	17622	1
20	1362935	0	0
21	1362996	23351	1

Figure 21. Data shown in view customer_vw.

Now as there are some group of usernames which should have access only to security schema these users must be granted only specific user rights to access data. This part will be described in chapter 3.

2.5.2 Masking by replacing and using mapping table

Secondly we will create a procedure that is able to mask certain columns and so mask production environment data so that we have no sensitive information in test environment or development environment.

As was described before we have a data model with physical tables and data inside the tables. Model is described in Figure 1 and physical table creating scripts are in Appendix 1.

We have requirements that in test environment should columns be masked as not real data shown in table 10.

TableName	ColumnName
customer	age
credit_tran	Merchant_Name

Table 10. Columns which have to be masked in table.

Now we need to analyze each column data type and make a decision how are we going to mask them.

Customer.age – This column in SMALLINT type, for this we can use Teradata random number generator by giving the interval for example from 1 to 99.

Credit_tran.Merchant_Name – This column is CHAR(20), Here we can use mapping table by replacing each character with a character from mapping table

In mapping table we need to create mapping type code which will define for which table column this mapping row is for.

For creating mapping table we need to execute script shown in figure 22.

```
CREATE SET TABLE financial.masking_mapping
      (
      mapping_type_code INTEGER NOT NULL,
      pos INTEGER,
      orig_val_INT INTEGER,
      orig_val_CHR CHAR(1),
      mask_val_INT INTEGER,
      mask_val_CHR CHAR(1))
PRIMARY INDEX (mapping_type_code);
```

Figure 22. Script for creating mapping table.

We need to create also another table for defining mapping_type_code and how many positions column has and the type of column and in which table the column is.

In figure 23 is shown query for creating mapping type table.

```
CREATE SET TABLE financial.mapping_type
       (
       mapping_type_code INTEGER NOT NULL,
       pos_count INTEGER,
       column_type SMALLINT,
       tablename CHAR(20),
       columnname CHAR(20))
PRIMARY INDEX (mapping_type_code);
```

Figure 23. Script for creating mapping_type table

We need to insert into tables values for mapping type codes and the replaceable values which will be in mapping table. Example of inserting values to target tables is shown on figure 24 and 25:

```
INSERT INTO financial.mapping_type (
mapping_type_code,
pos_count,
column_type,
table,
column)
VALUES (1, 20, 2, credit_tran, 'Merchant_Name');
```

Figure 24. Script for inserting values to mapping_type table.

```
INSERT INTO financial.masking_mapping (
mapping_type_code,
position,
orig_val_INT,
orig_val_CHR,
mask_val_INT,
mask_val_CHR)
VALUES (1, 1, NULL, ‘a’, NULL, ‘x’);
```

Figure 25. Script for inserting values to masking_mapping table.

For doing masking all at once will be created a macro which contains all necessary updates for masking data so that it is all in one place and database administrator can do only one execution for masking all data.

Creating macro is shown in figure 26.

```
CREATE MACRO financial.mask AS (
UPDATE financial.customer
SET age = random(1,99);
CALL mask_MerchName;
);
```

Figure 26. Script for creating macro.

Inside the macro first we update age column by setting the value for each value using Teradata random function between values 1 and 99. Next will be called stored procedure which will mask Merchant_Name in credit_tran table. This is in separate stored procedure to keep the logic separate and make the main macro better to maintain.

Stored procedure contains update query where for each letter in the name is replaced with character from mapping table. This is done using Teradata oreplace and substr functions.

Oreplace function takes 3 parameters:

1) String where something will be replaced
2) Character which will be replaced
3) Character which is the replaceable.

Substr function takes 2 parameters:

1) String from which substring will be taken
2) starting index of string and length of substring which will be taken for example 2nd character is 2 FOR 1.

In figure 27 is shown an example of first character replacement using mapping table.

```
oreplace(substr(Merchant_Name, 1 FOR 1), (
SEL orig_val_CHR
FROM   financial.masking_mapping
WHERE mapping_type_code = 3
AND pos = 1
       AND orig_val_CHR = substr(Merchant_Name, 1 FOR 1)
), (
SEL mask_val_CHR
FROM   financial.masking_mapping
WHERE mapping_type_code = 3
AND pos = 1
       AND orig_val_CHR = substr(Merchant_Name, 1 FOR 1)
))
```

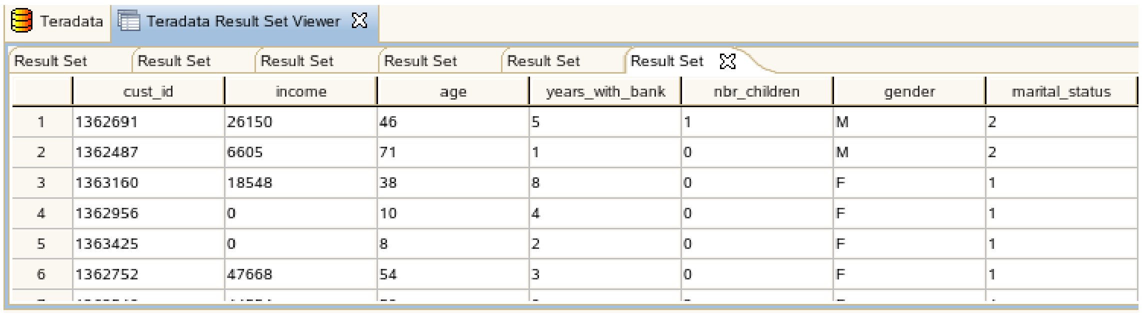
Figure 27. Script for first character replacement using mapping table.

Macro must be executed and 2 columns in 2 tables will be masked with query shown in figure 28.

```
Exec financial.mask;
```

Figure 28. Script to execute macro mask.

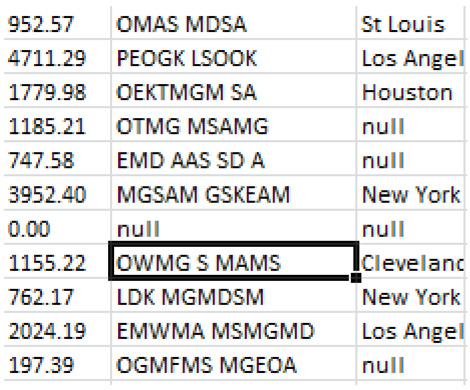
Generated random values for column age in table customer data are shown in figure 29.



	cust_id	income	age	years_with_bank	nbr_children	gender	marital_status
1	1362691	26150	46	5	1	M	2
2	1362487	6605	71	1	0	M	2
3	1363160	18548	38	8	0	F	1
4	1362956	0	10	4	0	F	1
5	1363425	0	8	2	0	F	1
6	1362752	47668	54	3	0	F	1

Figure 29. Random values generated for column age in table customer.

Generated mapped values for column Merchant_Name in table credit_trans data are shown in figure 30(second column in figure).



952.57	OMAS MDSA	St Louis
4711.29	PEOGK LSOOK	Los Angel
1779.98	OEKTMGM SA	Houston
1185.21	OTMG MSAMG	null
747.58	EMD AAS SD A	null
3952.40	MGSAM GSKEAM	New York
0.00	null	null
1155.22	OWMG S MAMS	Clevelanc
762.17	LDK MGMDSM	New York
2024.19	EMWMA MSMGMD	Los Angel
197.39	OGMFMS MGEOA	null

Figure 30. Mapped values for column Merchant_Name in table credit_trans.

Using mapping table and for masking long characters is time consuming but as this process is supposed to be right after test database synchronization then performance is not the first priority.

2.5.3 Benefits from created masking solution

If we take into account that currently copying all tables from backup takes 1 minute and when adding into this process execution of mapping macro then this will need additional time of 5 minutes. By amount of time this not big number but when considering that the dataset which will be copied takes an hour to copy then by this calculation execution of mapping macro will take 5 hours. Considering that environment maintenance, data generation is done by administrators on weekends this is still reasonable time. The time estimation is done based on before described solution where dataset amount was not big, about 200MB, also the test database was created on a laptop with a virtual machine with not a powerful CPU. In real database system hardware is a lot more powerful. As copying data from production data backup is done usually automatically to simplify administrators work and automate the process, masking macro execution can be added to the data copying job and it will be an additional step in automated job but for administrator it will be as 1 step to execute the job.

Masking data in test and development environment and using views to hide certain columns decreases risk of having data breach. Data breach can cost about 5,5 million USD dollars [6] depending on the size of organization and number of customers. When we have database with 200 users where is data for 10000 customers and production data is used in test and development environments and users do not have special roles but possibility to see all data, then after implementing data masking solution the possibility of data breach decreases significantly and this is financial win for company. When using data breach cost calculator by hub international we can estimate that leaked 50000 records with social security number means cost of around 2 million dollars, each record 40 dollars [7]. After using masking solution which decreases the chance of data breach approximately 30% [6] we can say that actual win in money for company with 200 database users and data of 10000 customers is 600000 dollars. Meaning of this is that possible data breach from inside users decreases as users have less access to data and in test and development environments there is no sensitive data which could be personally identifiable.

3 User rights in data warehouse

Database is the center for organization as keeping sensitive financial, customer and company data. When this valuable data gets under attack and there is a data breach it is calculated that an average cost for data breach in US for an organization is approximately 5,5 million US dollars [6]. So for every organization it should be very critical to keep the database secure from data breaches as this is direct risk of losing money and trust of customers. Keeping database secure means to have good knowledge of who is using database, what can users access in database and what should user not have access in database. Having a small organization with 20 workers having access to database is usually not a problem and manageable easily without needing any special need for user rights management but when we have an organization where database is accessed by hundreds of users who can be with very different needs of data from database then it is necessary to have some common way for user rights management.

In my experience of working in a financial organization data warehouse department ordering user rights is quite a complicated process and takes a lot of time. It starts with filling a form where and what rights I will need. Then the request needs to be accepted by direct manager of the person who ordered rights. Very often getting the accept from direct manager can take days because of big number of emails, business trip or vacation. After that the task is forwarded by IT helpdesk to database administrator who will manually give rights to user for certain objects. Database administrator needs to maintain user rights also later if user has left the organization or doesn't need the rights anymore. This process for ordering user rights is shown in figure 31.

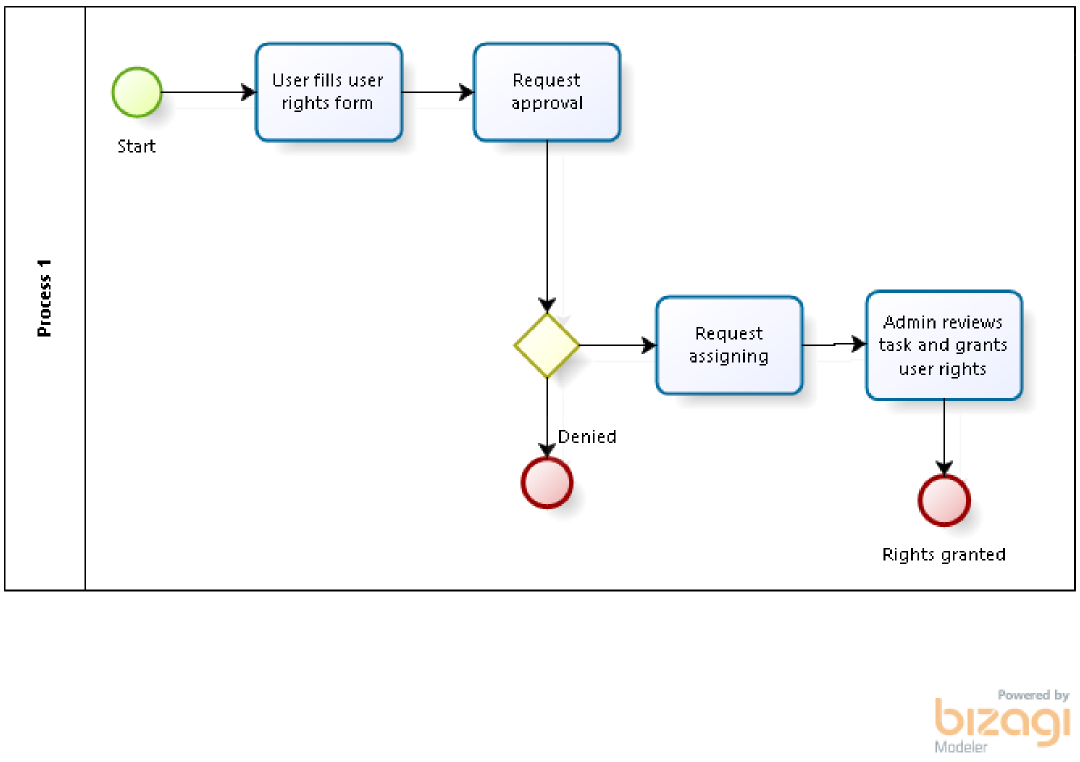

Process 1
Start
User fills user rights form
Request approval
Request assigning
Admin reviews task and grants user rights
Denied
Rights granted
Powered by bizagi Modeler

Figure 31. Process for ordering user rights.

With described process I can see here a way to improve and speed up the process. First idea is to use project based user rights. Users usually need access rights related to some new or existing project. Usually when a new project is started a lot of users need to order rights. For example we have 10 testers who need to order user rights to access new data related to new project. Each user needs to fill form which takes approximately 5-10 minutes, getting approval might take a day, if there is different manager for users it can take even more time, each request needs to be assigned and administrator needs to do each task separately. A proposal for such case is that ordering rights could be done by project manager for all users. In this case we can make the process better by adding details how long the rights will be needed, project manager will order rights for certain period and if project is longer than expected user rights can be extended because project manager is the key person of project and is responsible that all project members have access to all the project related data.

Example form for ordering user rights for multiple users is shown on table 11.

Person who orders:	Project manager name
Users needing rights:	User1, user2, user3
Database:	Target database
Name of user role:	Project_XX11
Access rights applied till:	2016-09-30

Table 11. Example form for ordering user rights for multiple users.

We could skip the approval part as project manager is responsible for the project and so for knowing what people can or cannot do, later on reordering rights when rights are needed longer than expected and the most time consuming part: every user ordering rights by themselves. Process when ordering rights by project manager is shown on figure 32.

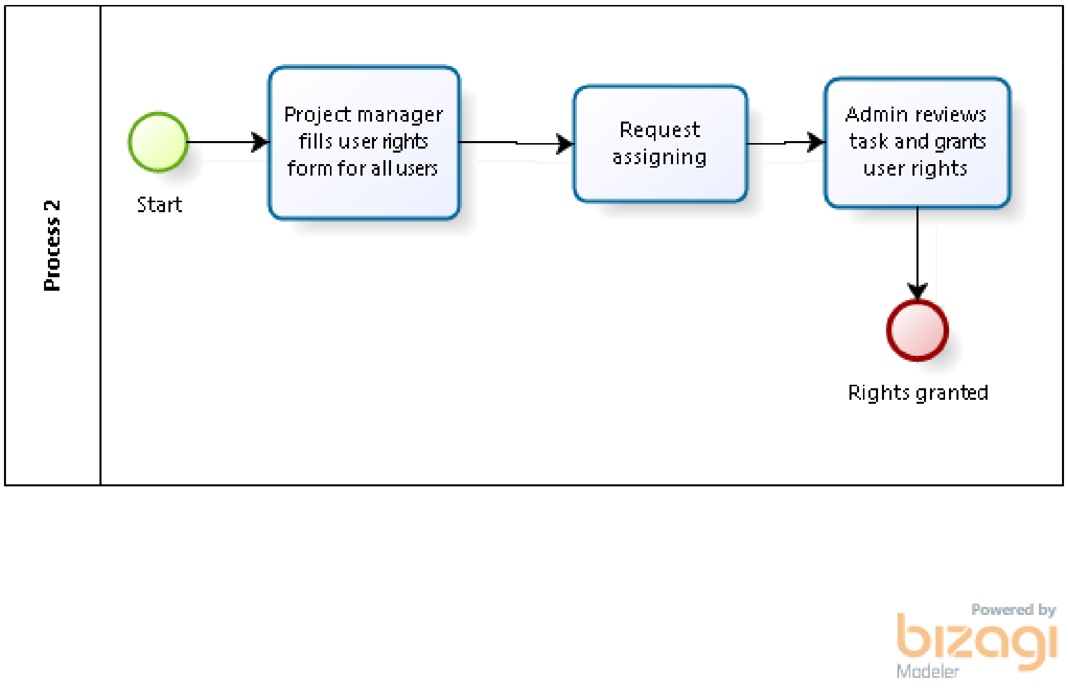

Process 2
Start
Project manager fills user rights form for all users
Request assigning
Admin reviews task and grants user rights
Rights granted
Powered by bizagi Modeler

Figure 32. Ordering rights process by project manager.

In the proposed process for user rights ordering and managing I can see also an automation possibility for database administrator when maintaining user rights. Each user right has the date how long it will be active and after that it will be removed. For this can be created a table where data is inserted for each given user right and the project related to it. The table columns are shown in table 12.

Username	Role	Start_Date	End_Date

Table 12. Columns for table where are kept user rights related data.

With this solution the number of steps will decrease in process and the ordering part will be faster as it will be done by 1 person. Also user rights have date till valid to make sure that users will not have rights they do not need later.

Time estimations for ordering rights for 10 users separately and with ordering once by project manager by my own personal experience working in financial organization warehouse department are shown in table 13.

	Separately	Once by project manager
Filling form	10min*10=100min	10 min
Getting approval	3600min	0
Request assigning	5min*10=50min	5min
Granting rights	10min*10=100min	10min
TOTAL	3850min	25min

Table 13. Time estimations for ordering user rights separately and by project manager.

When not ordering rights separately for new project time win can be 100 times bigger as getting approval from managers can take days. This also reduces work for IT helpdesk who are assigning requests and creating tasks and the work of administrators who can do 1 task instead of 10.

Also greater use of the date how long user rights are needed reduces the work of managers and administrators who have to maintain rights which are not needed anymore because of finished project, user leaving the organization or change of role as a employee. Administrator can easily check when some users rights are starting to end and notice the user or just remove the rights.

3.1 Approach for user rights management

In a small database meaning small number of database objects we can grant user rights by each object. For example user x can select data from table t1 and t2 and user x can insert to table t1. In data warehouse there is a common way that there is a system table where all rights for each object and user is stored. Now when user is doing some kind of query the system checks first if the user has all access rights necessary for the query. Granting user rights by object is simple and very risk free if we know exactly what the user can and cannot access but if user is leaving the organization it must be made sure that user rights are removed. Having huge number of users and objects in database causes problems in such way when granting rights for users by objects.

An useful solution which is possible develop in most databases is by creating roles for different user groups [7]. Each role has its own definition of rights what is possible to do in database. When having good architectural solution by different layers this can be done very simply. Unfortunately usually users don’t need access to all tables in one layer. Possible approach how to get to know how many roles and what rights each role should have is by analyzing users and their needs. Should be some common way also when a new user comes then what roles is needed and when a user leaves or doesn’t need access to database any more then what roles should be removed from user. This is useful to automate in bigger organizations.

For example we have 5 business users of which 2 are from customer management and 3 from accounting department also we have a service manager, a developer and a tester. Business users don’t need all access to accounting so we would create for business users 2 roles: customer_dept and accounting_dept having only access to select data from customer department users customer related data and accounting department accounting related data in production environment. Service manager should have access in all environments by doing basically everything except dropping database. Developer should have access only in development environment but to all data and rights to create and drop objects as much as needed as this is development environment. And tester should have access to select all data in test environment.

3.2 Example of user rights management in Teradata database

In this paragraph will be created example users and roles for them. Database and model we are using is the example created in paragraph 2.5.

There will be business users, all of them need to have access to security layer just for selecting data, not modifying, inserting or deleting. Then there are testers who will need access to all structures but without possibility to modify actual structures of objects. And then there will be developer users who should have access to all objects and rights to also modify structures of objects. These roles are specific to each environment, business users in production environment, testers in test environment and developers in development environment.

In Teradata database there is a system table called DBC.AccessRights. All rights which each role has are inserted there and when query is sent to database it will be first checked from there if the user has rights on the object selected. In Teradata there are 60 different access rights which can be granted [8].

The usernames, roles and access rights are described in table 14.

Username	Role	Access rights	Schema
bsns1	business	SELECT	security
bsns2	business	SELECT	security
dvlpr1	developer	INSERT, SELECT, DELETE, TABLE, CREATE VIEW, CREATE MACRO, CREATE TRIGGER	financial, security
tstr1	tester	SELECT, INSERT, DELETE	financial, security

Table 14. Usernames, roles and access rights description.

Crate new roles by function CREATE ROLE script is shown in figure 33.

```
CREATE ROLE business;
CREATE ROLE developer;
CREATE ROLE test;
```

Figure 33. Script for creating new roles.

To give each role needed access rights based on schema is used function GRANT and each access right has its own name. The script for this is shown in figure 34.

```
GRANT SELECT ON security to business;
GRANT INSERT, SELECT, DELETE, TABLE, CREATE VIEW, CREATE MACRO, CREATE
             TRIGGER ON financials to developer;
GRANT INSERT, SELECT, DELETE, TABLE, CREATE VIEW, CREATE MACRO, CREATE
             TRIGGER ON security to developer;
GRANT SELECT, INSERT, DELETE ON financial to tester;
GRANT SELECT, INSERT, DELETE ON security to tester;
```

Figure 34. Script for granting rights to roles.

To grant each user a role is used function GRANT <role_name> to <user_name>. The script for this is shown in figure 35.

```
GRANT BUSINESS TO bsns1;
GRANT business TO bsns2;
GRANT developer TO dvlpr1;
GRANT tester TO tstr1;
```

Figure 35. Script for granting user certain role rights.

To test if all users have needed rights as defined in table 14 and no rights to some other schema will be created test cases which are shown in table 15.

Test Case Id	Test description	SQL	Expected Result	Actual Result	Result
1	Check if user bsns1 can access security schema views.	--Logged in as user bsns1 SELECT * FROM security.customer;	Rows returned, no error messages	Rows returned, no error	Passed
2	Check if user bsns1 can access financial schema objects	--Logged in as user bsns1 SELECT * FROM financial.customer;	Query fails as user has no rights to access selected table	SELECT Failed. 3523: The user does not have SELECT access to financial.customer.	Passed
3	Check if user dvlpr1 can create table in financial schema	--Logged in as user dvlpr1 CREATE TABLE financial.testtable (testcolumn INTEGER NOT NULL) PRIMARY INDEX (testcolumn);	Table created successfully	Table created successfully	Passed

4	Check if user dvlpr1 can create macro in financial schema	--Logged in as user dvlpr1
		CREATE MACRO financial.testmacro AS(
		DROP TABLE financial.testtable;);
			Macro created successfully	Macro created successfully	Passed

5	Check if user dvlpr1 can create view in security schema.	--Logged in as user dvlpr1
		CREATE VIEW security.testview
		AS
		SELECT *
		FROM security.accts_VW;
			View created successfully	View created successfully	Passed

6	Check if user tstr1 can delete data from financial schema table	--Logged in as user tstr1
		DELETE FROM financial.accts
		where acct_type='CK';
			Query executed successfully, some rows are deleted	Delete Statement completed. 8 rows processed	Passed

7	Check if user tstr1 can insert data to financial schema table	--Logged in as user tstr1 INSERT INTO financial.customer (cust_id, income, age, years_with_bank, nbr_children, gender, marital_status) VALUES(2,2222,33,1,2,'F','M')	Query executed successfully, new row is inserted	Query executed successfully, new row is inserted	Passed
8	Check if user tstr1 can select data from security schema views	--Logged in as user tstr1 SELECT * FROM security.customer;	Rows returned, no error messages	Rows returned, no error	Passed

Table 15. Testcases for checking if all users have the correct rights described in table 14.

4 Summary

Data warehouse kind of database systems contains a lot of information and this data going accidentally public is a great financial risk for every organization. This is the topic which needs more attention and especially when taking into account test and development environments where is often used production data. Every organization that is using production data without modifying or masking it in development or test environment should very much consider doing something about it because risk by having data breach because of this is quite big. Also knowing that every user has the correct data access rights is very closely related with data security. The aim of this thesis was to introduce some ways of masking data and managing user rights and implementing in Teradata database system.

The result of this implementation is an example database where is used different techniques for hiding sensitive data by creating security layer using views and using data masking by replacing and mapping sensitive personally identifiable columns and example users and roles for managing user roles.

As this example is very general then must be said that every database is unique having its own architecture and models which means that every database needs its own analyze to make sure what data needs to be secured, how it could be done.

By creating this example database it is possible to say that using masking technologies is a great way to increase data security in data warehouse and using user rights by roles decrease chances of data breaches and reduces human resource need for managing rights by objects.

Kokkuvõte

Andmeaida tüüpi andmebaas hõlmab endas palju informatsiooni ning kui peaks juhtuma, et see info saab kogemata avalikuks, on see suur rahaline risk iga organisatsiooni jaoks. See on teema, mis vajab rohkem tähelepanu ning eriti arvestades test ja arendus keskkondi kus kasutatakse tihtipeale toodangukeskkonna andmeid. Iga organisatsiooni kes kasutab toodangu andmeid test või arenduskeskkonnas ilma neid muutmata või maskimata peaks tõsiselt kaaluma midagi muuta sest sellisel juhul on võimalus andmelekkeks kordades suurem. Lisaks teades, et igal andmebaasi kasutajal on õige ligipääs andmetele ja mitte rohkem on selle teemaga samuti tihedalt seotud. Selle töö eesmärk on tutvustada maskeerimise viise ning kasutajaõiguste haldamist ning implementeerida see Teradata andmebaasi süsteemi peal.

Implementatsiooni tulemus on näidis andmebaas kus on kasutatud erinevad tehnikad peitmaks sensitiivsed andmeid luues turvalise kihi kasutades vaateid ning kasutades andmete maskimist asendamise ja kaardistamise viisil ning luues näidis kasutajad ja rollid nende haldamiseks.

Kuna see näide on üldine ja tehtud lihtsa näite põhjal siis peab toonitama, et iga andmebaas on unikaalne ning omab isesugust arhitektuuri ja mudeleid, mis tähendab, et iga andmebaasi jaoks on vaja teha eraldi põhjalik analüüs, et olla kindel, mis andmeid on vaja kaitsta ning kuidas seda oleks kõige parem teha.

Loodud näidisandmebaasi põhjal võib öelda, et kasutades maskeerimise tehnoloogiaid on suurepärane võimalus suurendada andmete turvalisust andmeaidas ning kasutades kasutajaõiguseid rollide näol kahandab andmelekke võimalust ning vähendab inimressurssi haldamaks õigusi objektipõhiselt.

References

[1] "Personal Data Protection Act," 15 February 2007. [Online]. Available: https://www.riigiteataja.ee/en/eli/507032016001/consolide. [Accessed 9 May 2016].

[2] S. V. Camp, "Hacking at LinkedIn Highlights PR’s Critical Role in Data Breaches," 6 June 2012. [Online]. Available: http://www.prnewsonline.com/featured/2012/06/18/hacking-at-linkedin-highlights-prs-critical-role-in-data-breaches/. [Accessed 9 May 2016].

[3] R. Dobson, "Masking Personal Identifiable SQL Server Data," 13 November 2013. [Online]. Available: https://www.mssqltips.com/sqlservertip/3091/masking-personal-identifiable-sql-server-data/. [Accessed 21 April 2016].

[4] A. Nanda, "Hide from Prying Eyes," 14 January 2014. [Online]. Available: http://www.oracle.com/technetwork/issue-archive/2014/14-jan/o14dba-2045565.html. [Accessed 21 April 2016].

[5] L. Cliff, "Teradata Express 14.0 for VMware User Guide," 17 May 2012. [Online]. Available: http://developer.teradata.com/database/articles/teradata-express-14-0-for-vmware-user-guide. [Accessed 9 May 2016].

[6] Ponemon Institute, "2015 Cost of Data Breach Study: United States," May 2015. [Online]. Available: http://public.dhe.ibm.com/common/ssi/ecm/se/en/sew03055usen/SEW03055USEN.PDF?. [Accessed 9 May 2016].

[7] "Data Breach Cost Calculator," [Online]. Available: https://www.hubinternational.com/business-insurance/cyber-risk-solutions/tools/data-breach-cost-calculator/. [Accessed 9 May 2016].

[8] J. Browning, "Hardening a Teradata database," October 2013. [Online]. Available: http://assets.teradata.com/resourceCenter/downloads/WhitePapers/EB7452.pdf?processed=1. [Accessed 9 May 2016].

[9] "Teradata Database, Tools and Utilities Release 14.10," [Online]. Available: http://www.info.teradata.com/HTMLPubs/DB_TTU_14_10/index.html. [Accessed 9 May 2016].

5 Appendix 1 – Teradata financial schema table generation scripts

```
CREATE SET TABLE financial.accts
     (
      cust_id INTEGER NOT NULL,
      acct_type CHAR(2) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
      acct_nbr CHAR(16) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
      acct_start_date DATE FORMAT 'YYYYMMDD' NOT NULL,
      acct_end_date DATE FORMAT 'YYYYMMDD')
PRIMARY INDEX ( cust_id ,acct_type );


CREATE SET TABLE financial.customer
     (
      cust_id INTEGER NOT NULL,
      income INTEGER,
      age SMALLINT,
      years_with_bank SMALLINT,
      nbr_children SMALLINT,
      gender CHAR(1) CHARACTER SET LATIN NOT CASESPECIFIC,
      marital_status CHAR(1) CHARACTER SET LATIN NOT CASESPECIFIC)
UNIQUE PRIMARY INDEX ( cust_id );

CREATE SET TABLE financial.credit_acct
     (
      cust_id INTEGER NOT NULL,
      acct_nbr CHAR(16) CHARACTER SET LATIN NOT CASESPECIFIC,
      credit_limit INTEGER,
      credit_rating SMALLINT,
      account_active CHAR(1) CHARACTER SET LATIN NOT CASESPECIFIC,
      acct_start_date DATE FORMAT 'YYYYMMDD',
      acct_end_date DATE FORMAT 'YYYYMMDD',
      starting_balance DECIMAL(9,2) FORMAT '--Z(6)9.99',
      ending_balance DECIMAL(9,2) FORMAT '--Z(6)9.99')
UNIQUE PRIMARY INDEX ( cust_id );
```

```
CREATE SET TABLE financial.credit_tran
     (
      Tran_Id INTEGER NOT NULL,
      Cust_Id INTEGER NOT NULL,
      Acct_Nbr CHAR(16) CHARACTER SET LATIN NOT CASESPECIFIC,
      Channel_Nbr INTEGER,
      Merchant_SIC INTEGER,
      Session_Id INTEGER,
      Tran_Duration SMALLINT,
      Tran_Date DATE FORMAT 'YYYY-MM-DD',
      Tran_Time CHAR(6) CHARACTER SET LATIN NOT CASESPECIFIC,
      Tran_Amt DECIMAL(9,2),
      Principal_Amt DECIMAL(9,2),
      Interest_Amt DECIMAL(9,2),
      New_Balance DECIMAL(9,2),
      Merchant_Name CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
      Merchant_City CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
      Merchant_State CHAR(2) CHARACTER SET LATIN NOT CASESPECIFIC,
      Merchant_Postal_Code CHAR(5) CHARACTER SET LATIN NOT CASESPECIFIC,
      Tran_Code CHAR(2) CHARACTER SET LATIN NOT CASESPECIFIC,
      Channel CHAR(1) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( Tran_Id );
```