Database Security – master, 2nd year
Laboratory 2

Auditing activities in the database

Keywords: • Audit • Database audit trail • Operating system audit trail	• Audit triggers • Audit policies • The package DBMS_FGA

- Auditing the activity on the database has two components: monitoring and persistent recording of a set of activities and events, established a priori, from the database.
- The objectives of auditing the activities on the database include: non-repudiation, investigation of suspicious activities, detection of problems generated by the current configurations regarding the authorization (access to resources), compliance with the current legislation, control.

1. Standard audit

1.1 What activities can be audited?

Starting and stopping the database, connection of the administrator to the database	***They are audited by default by the Oracle system; the data is automatically stored in the OS***

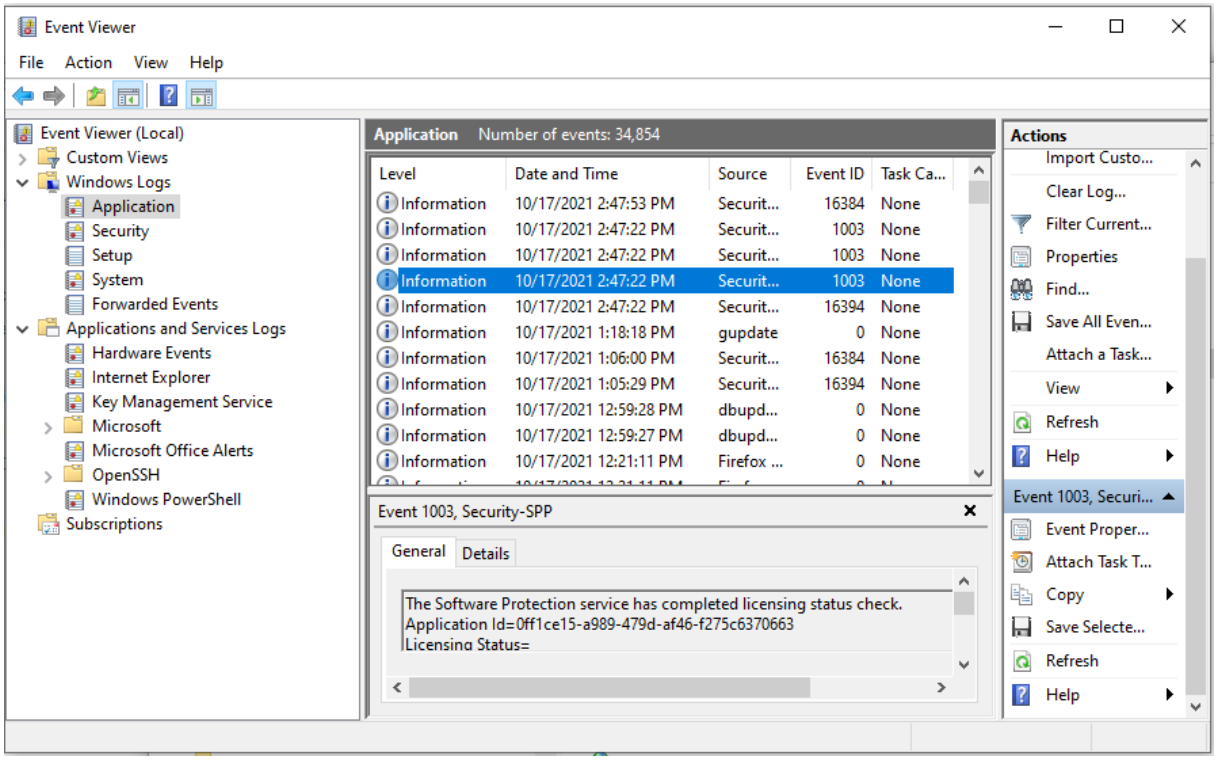
	For all users	For user Tom
SQL statements		
- DDL : *all statements CREATE TABLE, DROP TABLE, TRUNCATE TABLE*	***AUDIT TABLE***	***AUDIT TABLE BY Tom***
- DML : *all statements INSERT, UPDATE, DELETE*	***AUDIT INSERT TABLE*** ***AUDIT DELETE TABLE*** ***AUDIT UPDATE TABLE***	***AUDIT INSERT TABLE BY Tom*** ***etc.***
- SELECT: *all queries on all tables and all views*	***AUDIT SELECT TABLE***	***AUDIT SELECT TABLE BY Tom***
SQL statements on a specified object (schema.object) of the database		
- only when the statement fails	***AUDIT SELECT, INSERT, UPDATE,*** ***DELETE ON Tom.employees BY ACCESS WHENEVER NOT SUCCESSFUL;***	***AUDIT SELECT, INSERT, UPDATE,*** ***DELETE ON Tom.employees BY Tom WHENEVER NOT SUCCESSFUL;***
- anytime	***AUDIT SELECT, INSERT, UPDATE,***	***AUDIT SELECT, INSERT, UPDATE,***

	DELETE ON Tom.employees;	***DELETE ON Tom.employees BY Tom;***
- default audit for the objects that will be created	***AUDIT ALTER, GRANT, INSERT, UPDATE, DELETE ON DEFAULT;***	-
Network activity	***AUDIT NETWORK***	-
Exercise of privileges - *whenever a privilege is used to perform a database action*	***Example:*** ***AUDIT CREATE ANY VIEW (in any schema)*** ***AUDIT CREATE VIEW (in own schema)***	***AUDIT CREATE ANY VIEW BY Tom*** ***etc.***
Database session	***AUDIT SESSION***	***AUDIT SESSION BY Tom***

1.2 Where do we record the monitored information?

- In the database – *database audit trail*:
  → ***audit_trail = DB*** (the table *SYS.AUD$*, the views *DBA_AUDIT_TRAIL*, *DBA_COMMON_AUDIT_TRAIL*)
  `alter system set audit_trail=db scope=spfile;`
  → ***audit_trail = DB,EXTENDED*** (the same table and the same views, but the statements' text is also stored in the field *SQLTEXT* of type *CLOB***)**

- External to the database – *operating system audit trail*. Options:
  → ***audit_trail = OS*** (in *Windows, Control Panel → Administrative Tools → Event Viewer →* the "Application" zone in *Windows Event Viewer*)
  `alter system set audit_trail=os scope=spfile;`



→ ***audit_trail = XML, AUDIT_FILE_DEST* = path to the file (**by default, this

path is $ORACLE_BASE/admin/$ORACLE_SID/adump.**)**

```
alter system set audit_trail=xml scope=spfile;
```

1.3 Starting and stopping the standard audit

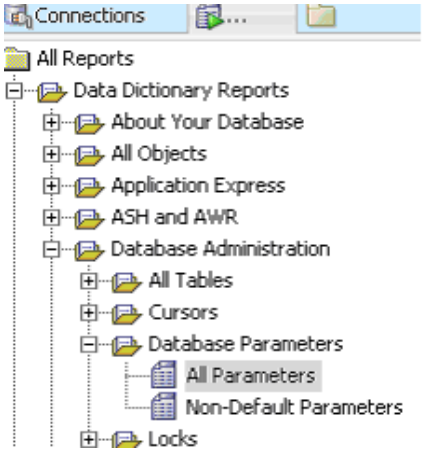
- To find out the current configuration regarding the storage location of the monitored data (with lowercase!):
  - → The *SELECT* statement:

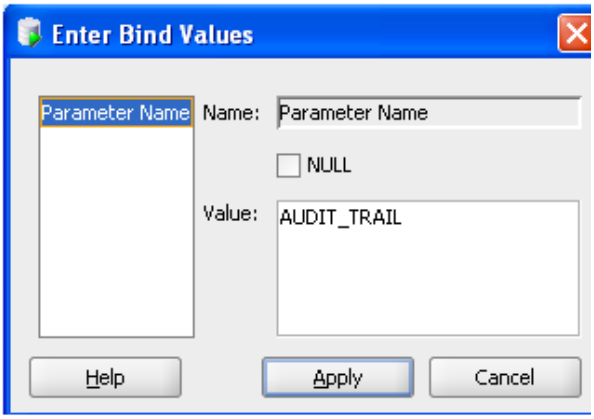
    ```
    select value from v$parameter where name='audit_trail';
    ```

  - → or from SQLPlus:

    ```
    show PARAMETER audit_trail
    ```

  - → or from SQLDeveloper: the menu *View → Reports → All Reports → Data Dictionary Reports → Database Administration → Database Parameters → All Parameters* → select the connection to use, and enter the value AUDIT_TRAIL in the dialog box, then press the "Apply" button.





- ***Starting the audit for activity X (see the table on the first page): AUDIT x***
- ***Stopping the audit for activity X: NOAUDIT x***
- ***Mass stopping of the audit for all SQL commands unrelated to a specific object: NOAUDIT ALL***
- ***Mass stopping of the audit for the exercise of privileges: NOAUDIT ALL PRIVILEGES***
- ***Mass stopping of the audit for all SQL commands related to a specific object OBJ: NOAUDIT ALL ON obj***
- ***Mass stopping of the default audit, for all SQL commands related to objects: NOAUDIT ALL ON DEFAULT***

1.4 Delete monitoring information after archiving it

- Depending on the number of audited activities and their daily frequency, the volume of monitoring data can become very large and thus take up useful disk space. That is why it is recommended to periodically archive the monitored data and delete it from the

production system.

- If data is recorded in the database (database audit trail), then you can use delete commands (remember, after archiving the data in advance!):

```
DELETE FROM SYS.AUD$;
```

- You can choose to delete the monitored information for a specific database object, for example for the *EMPLOYEES* table:

```
DELETE FROM SYS.AUD$ WHERE OBJ$NAME='EMPLOYEES';
```

2. Triggers for auditing

2.1 Recap *triggers*

- We recall from the DBMS course that "*a trigger is a PL/SQL block or CALL of a PL/SQL procedure that runs automatically whenever a certain trigger event occurs*".
- There are two types of triggers: database triggers (database operations) and application triggers (for example, pressing a button on a form in Oracle Forms). The category of interest for us in this material is represented by the database triggers.
- **Database triggers** are, in turn, classified in 3 categories:
  - → *DML* triggers – they are triggered by the DML statements on a table. They can be either executed only once at the level of a statement regardless of the number of affected rows (statement-level triggers) or they can be executed *FOR EACH ROW* (row-level triggers). They correspond to the types of triggers *BEFORE STATEMENT, AFTER STATEMENT, BEFORE EACH ROW, AFTER EACH ROW*;
  - → *INSTEAD OF* triggers – they are triggered by DML statements on views;
  - → *SYSTEM* triggers – they are triggered by events such as starting/ stopping the database, DDL commands, user login/logout. They correspond to the types of triggers *AFTER EVENT, BEFORE EVENT*.
- Querying the ***SYS.TRIGGER$*** table or the ***ALL_TRIGGERS*** view provides information about all the database triggers.

```
SELECT DISTINCT TRIGGER_TYPE FROM ALL_TRIGGERS;
```

```
TRIGGER_TYPE
----------------
BEFORE STATEMENT
BEFORE EACH ROW
AFTER EACH ROW
BEFORE EVENT
AFTER STATEMENT
AFTER EVENT
INSTEAD OF

7 rows selected.
```

- The ***DBA_TRIGGERS*** view provides information about the triggers automatically created when installing the *Oracle* products. Right after the creation of a database we find

617 DBA triggers. In order to find out information about the *SYSTEM* triggers (of type '***BEFORE EVENT***' **and** '***AFTER EVENT***'**)** that were created automatically during the installation, we have to execute a query like the following:

```
SELECT SUBSTR(OWNER,1,20) OWNER ,
       SUBSTR(TRIGGER_NAME,1,30) TRIGGER_NAME,
       SUBSTR(TRIGGERING_EVENT,1,30) TRIGGERING_EVENT,
       TRIGGER_TYPE
FROM DBA_TRIGGERS
WHERE TRIGGER_TYPE='BEFORE EVENT' OR TRIGGER_TYPE='AFTER EVENT'
ORDER BY TRIGGER-TYPE DESC;
```

OWNER	TRIGGER_NAME	TRIGGERING_EVENT	TRIGGER_TYPE
SYS	XDB_PI_TRIG	DROP OR TRUNCATE	BEFORE EVENT
SYS	CDC_ALTER_CTABLE_BEFORE	ALTER	BEFORE EVENT
MDSYS	SDO_ST_SYN_CREATE	CREATE	BEFORE EVENT
MDSYS	SDO_TOPO_DROP_FTBL	DROP	BEFORE EVENT
EXFSYS	EXPFIL_RESTRICT_TYPEEVOLVE	CREATE OR ALTER	BEFORE EVENT
EXFSYS	EXPFIL_DROPOBJ_MAINT	DROP	BEFORE EVENT
SYS	CDC_DROP_CTABLE_BEFORE	DROP	BEFORE EVENT
MDSYS	SDO_GEOR_BDDL_TRIGGER	DDL	BEFORE EVENT
SYS	CDC_CREATE_CTABLE_BEFORE	CREATE	BEFORE EVENT
EXFSYS	RLMGR_TRUNCATE_MAINT	TRUNCATE	BEFORE EVENT
WMSYS	NO_VM_DDL	CREATE OR ALTER OR DROP OR REN	BEFORE EVENT

OWNER	TRIGGER_NAME	TRIGGERING_EVENT	TRIGGER_TYPE
SYS	OLAPISHUTDOWNTRIGGER	SHUTDOWN	BEFORE EVENT
MDSYS	SDO_NETWORK_DROP_USER	DROP	AFTER EVENT
MDSYS	SDO_GEOR_ADDL_TRIGGER	DDL	AFTER EVENT
MDSYS	SDO_DROP_USER	DROP	AFTER EVENT
SYS	OLAPISTARTUPTRIGGER	STARTUP	AFTER EVENT
MDSYS	SDO_GEOR_ERR_TRIGGER	ERROR	AFTER EVENT
EXFSYS	EXPFIL_DROPUSR_MAINT	DROP	AFTER EVENT
EXFSYS	EXPFIL_ALTEREXPTAB_MAINT	ALTER OR RENAME	AFTER EVENT
SYS	CDC_CREATE_CTABLE_AFTER	CREATE	AFTER EVENT
WMSYS	NO_VM_DROP_A	DROP	AFTER EVENT
SYS	AW_REN_TRG	RENAME	AFTER EVENT

OWNER	TRIGGER_NAME	TRIGGERING_EVENT	TRIGGER_TYPE
SYSMAN	MGMT_STARTUP	STARTUP	AFTER EVENT
SYS	AW_DROP_TRG	DROP	AFTER EVENT
SYS	AW_TRUNC_TRG	TRUNCATE	AFTER EVENT

25 rows selected

- Also, during the installation, DML triggers are automatically created in the HR user schema:

```
SELECT SUBSTR(TABLE_NAME,1,20) TABLE_NAME,
SUBSTR(TRIGGER_TYPE,1,30) TRIGGER_TYPE,TRIGGER_BODY FROM
DBA_TRIGGERS
WHERE  OWNER='HR';
```

TABLE_NAME	TRIGGER_TYPE	TRIGGER_BODY
EMPLOYEES	BEFORE STATEMENT	BEGIN secure_dml; END secure_employees;
EMPLOYEES	AFTER EACH ROW	BEGIN add_job_history(:old.employee_id, :old.hire_date, sysdate,

2.2 Using triggers in auditing

- For auditing, we can create custom triggers that record certain information of interest. In general, we will create a special table for storing monitored information.
- The triggers we build will be found when querying the *TRIGGER*$ table and the *ALL_TRIGGERS*, *USER_TRIGGERS* views.

- Some useful recap related to trigger processing, useful in auditing:

1) **The triggers we build must not influence the normal activity of the database.** The

purpose of the audit is to passively monitor and record the activity for further analysis. Therefore, we will NOT define *INSTEAD OF* triggers that misplace the results from the targeted tables to the audit table!

2) **The DML statement-level and row-level triggers can coexist. They will be called in the following order:**

BEFORE statement-level trigger
For each affected row
BEFORE row-level trigger
DML operation
AFTER row-level trigger
AFTER statement-level trigger

From the perspective of the audit, the granularity of the monitoring must be carefully established, because the purpose is not to clone the basic tables, but to record the activity on them.

3) **User-defined triggers will be executed only if, from Oracle's point of view, the statement is correct and can take place.** For an incorrectly built DML statement or one that violates some constraints, for example, the user-defined trigger will not be reached, but rather the error will be returned first.

In conclusion, statement-level DML triggers are especially suitable for auditing.

3. Audit policies

- The third audit method refers to Fine Grain Audit through audit policies. The structure of an audit policy is as follows:
  - → Specification of the object (schema, object name, columns) to be monitored;
  - → Specification of the monitored actions on the object (*SELECT*, *INSERT*, *UPDATE*, *DELETE*); the default is *SELECT*;
  - → Specification of the conditions under which the monitored information is recorded; it is the correspondent of the *WHEN* clause in a trigger and it is optional;
  - → Specification of an event handler to further address the event; it is optional.
- An audit policy can be enabled or disabled. No more than 256 audit policies can be defined for a database object.

  The list of enabled audit policies is obtained by querying the *ALL_AUDIT_POLICIES* view as follows:

```
SELECT POLICY_TEXT,ENABLED FROM ALL_AUDIT_POLICIES
WHERE OBJECT_NAME='DEPARTMENTS';
```

- For the management of audit policies we can use the *DBMS_FGA* package (it is necessary to grant privilege to the users who will write PL/SQL code that uses this package: *grant execute on dbms_fga to username;*).

- **Syntax:**

```
DBMS_FGA.ADD_POLICY (
        object_schema=>'schema name',
        object_name=>'audited object',
        policy_name=>'unique policy name',
        audit_column=>'col1,col2,.. in the audited object',
        enable=>false,
        statement_types=>'select,insert,update,delete'
        handler_schema=>'schema containing the handler'
        handler_module=>'handler name');
```

- The *handler* module must be a PL/SQL procedure with the following signature: **`CREATE OR REPLACE PROCEDURE <fname> (object_schema VARCHAR2, object_name VARCHAR2, policy_name VARCHAR2) AS ...`**

- The results of the audit can be obtained from the table *SYS.FGA_LOG$* and the view ***dba_fga_audit_trail***

- To enable or disable an audit policy, we use the following procedure:

```
DBMS_FGA.ENABLE_POLICY / DBMS_FGA.DISABLE_POLICY (
    object_schema=>'nume schema de care apartine obiectul',
    object_name=>'obiect auditat',
    policy_name=>'nume unic de politica');
```

Remark: The administrator's actions (*as SYSDBA*) are not audited (a modification in the file ini.ora is required)!

4. Exercises

1. **Configure the standard audit for the database, with the monitored data stored in the database.**

   All query activities performed in the database will be monitored and stored, together with the text of the user's queries.

   Display a report of these activities for the tables you query. Stop the configured audit.

2. **Configure the standard audit for the database by storing the monitored data in an XML file in the default path.**

   All failed DML commands on the *HR.EMPLOYEES* table will be monitored.

   Consult the resulting XML files. Stop the configured audit.

3. **For audit purposes, create trigger(s) that record in an audit table *(TAB_AUDIT_EMP*) the information about the DML delete operations on the *EMPLOYEES* table, including the number of affected rows.**

4. **For audit purposes, create a trigger that records in an audit table (*TAB_AUDIT_EMP*) the information about the DML operations that set salaries above the value of 20000.**

5. **Create an audit policy to record the DML statements for changing department managers (*MANAGER_ID*) in the *DEPARTMENTS* table.**