You can only change the initialization parameters as system DBA. Run the following steps:

1. Start SQL *Plus and log in as sys/password as sysdba.

2. Set the auditing parameter with the command:

   SQL> **ALTER SYSTEM set audit_trail = DB_EXTENDED SCOPE =SPFILE;**

   The DB_EXTENDED parameter stores the SQL statement to make it easier to read the log file. If you just use the DB value, the log file is smaller, but it keeps only the type of statement, not the actual command.

3. The parameter change will not take effect until you restart the Oracle instance. In SQL*Plus type **SHUTDOWN IMMEDIATE** to close the database and shut down your instance.

4. Restart the database. In SQL*Plus, type the **STARTUP** command. When the instance restarts and the database is open, verify that the change was made.

```
2018-10-05 18:33:01 SYS AS SYSDBA> ALTER SYSTEM set audit_trail = DB_EXTENDED SCOPE =SPFILE;

System altered.

2018-10-05 18:37:22 SYS AS SYSDBA> SHUTDOWN IMMEDIATE ;
Database closed.
Database dismounted.
ORACLE instance shut down.
2018-10-05 18:37:55 SYS AS SYSDBA> STARTUP;
ORACLE instance started.

Total System Global Area  456146944 bytes
Fixed Size                  1344840 bytes
Variable Size             369101496 bytes
Database Buffers           79691776 bytes
Redo Buffers                6008832 bytes
Database mounted.
Database opened.
05-OCT-18 SYS AS SYSDBA>
```

SQL> **show parameter audit_trail**
You should see the audit_trail value as DB_EXTENDED

```
05-OCT-18 SYS AS SYSDBA> show parameter audit_trail;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------------
audit_trail                          string      DB_EXTENDED
05-OCT-18 SYS AS SYSDBA>
```

Audit Command

1. Open a SQL*Plus session and log in as the sys user as sysdba. You are concerned about changes to the EMP and DEPT tables owned by Scott, so set audits for any insert or delete commands on those tables.

   SQL> **AUDIT INSERT on scott.EMP;**
   SQL> **AUDIT DELETE on scott.DEPT;**

```
05-OCT-18 SYS AS SYSDBA> AUDIT INSERT on scott.EMP;

Audit succeeded.

05-OCT-18 SYS AS SYSDBA> AUDIT DELETE on scott.DEPT;

Audit succeeded.

05-OCT-18 SYS AS SYSDBA>
```

   Notice that these commands do not have the WHENEVER clause, so they will record all insert and delete commands on the tables whether they are successful or not.

2. Connect with the SCOTT user account.

```
05-OCT-18 SYS AS SYSDBA> ALTER USER scott IDENTIFIED BY scott1;

User altered.

05-OCT-18 SYS AS SYSDBA> CONNECT scott/scott1;
Connected.
```

3. Create a new record in the EMP table and delete a record from the DEPT table as follows:

   SQL> **INSERT INTO EMP VALUES**
   (7500, 'SMITH', 'SALESMAN', 7698, TO_DATE('10-FEB-1985', 'DD-MON-YYYY'), 1600, 300, 30);
   SQL> **COMMIT;**

   SQL> **DELETE FROM DEPT WHERE DEPTNO=40;**
   SQL> **COMMIT;**

```
2018-10-05 18:57:00 SCOTT > INSERT INTO EMP VALUES (7500, 'SMITH', 'SALESMAN', 7698, TO_DATE('10-FEB-1985', 'DD-MON-YYYY'), 1600, 300, 30);

1 row created.

2018-10-05 18:58:30 SCOTT > COMMIT;

Commit complete.

2018-10-05 18:59:21 SCOTT > DELETE FROM DEPT WHERE DEPTNO=40;

1 row deleted.

2018-10-05 19:01:56 SCOTT > COMMIT;

Commit complete.

2018-10-05 19:02:14 SCOTT >
```

4. Connect as the sys user as sysdba again and stop the individual audits you created using the NoAudit command:

   SQL> **NOAUDIT INSERT on scott.EMP;**
   SQL> **NOAUDIT DELETE on scott.DEPT;**

```
2018-10-05 19:04:18 SYS AS SYSDBA> NOAUDIT INSERT on scott.EMP;

Noaudit succeeded.

2018-10-05 19:05:31 SYS AS SYSDBA> NOAUDIT DELETE on scott.DEPT;

Noaudit succeeded.

2018-10-05 19:05:55 SYS AS SYSDBA>
```

Viewing Audit Trails

When you choose the option to store audit data to the database, the records are stored in the sys.aud$ table.

1. Log back in as the sys user as sysdba and view the events that were logged. Issue the following SELECT statement:

   SQL> **SET linesize 150**
   SQL> **COLUMN UserID FORMAT A10**
   SQL> **COLUMN OBJ$NAME FORMAT A10**
   SQL> **COLUMN SQLTEXT FORMAT A40**
   SQL> **SELECT TO_CHAR (NTimeStamp#,**
   'YYYY-MM-DD HH24:MI:SS') AS Time, UserID, OBJ$NAME, SQLTEXT
   FROM sys.aud$
   WHERE OBJ$NAME IN ('DEPT','EMP');

```
Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 19:12:50 SYS AS SYSDBA> SET linesize 150
2018-10-05 19:13:31 SYS AS SYSDBA> COLUMN UserID FORMAT A10
2018-10-05 19:13:51 SYS AS SYSDBA> COLUMN OBJ$NAME FORMAT A10
2018-10-05 19:14:09 SYS AS SYSDBA> COLUMN SQLTEXT FORMAT A40
2018-10-05 19:14:29 SYS AS SYSDBA> SELECT TO_CHAR (NTimeStamp#,'YYYY-MM-DD HH24:MI:SS') AS Time, UserID, OBJ$NAME, SQLTEXT FROM sys.aud$ WHERE OBJ$NAME IN ('DEPT','EMP'
);

TIME                USERID     OBJ$NAME   SQLTEXT
------------------- ---------- ---------- ----------------------------------------
2018-10-06 01:58:30 SCOTT      EMP        INSERT INTO EMP VALUES (7500, 'SMITH', '
                                          SALESMAN', 7698, TO_DATE('10-FEB-1985',

2018-10-06 02:01:56 SCOTT      DEPT       DELETE FROM DEPT WHERE DEPTNO=40

2018-10-05 19:15:31 SYS AS SYSDBA>
```

2. Run the following commands:

SQL> **ALTER SYSTEM set audit_trail =NONE SCOPE=SPFILE;**
SQL> **DELETE FROM sys.aud$;**
SQL> **COMMIT;**

```
2018-10-05 19:15:31 SYS AS SYSDBA> ALTER SYSTEM set audit_trail =NONE SCOPE=SPFILE;

System altered.

2018-10-05 19:19:00 SYS AS SYSDBA> DELETE FROM sys.aud$;

687 rows deleted.

2018-10-05 19:19:24 SYS AS SYSDBA> COMMIT;

Commit complete.

2018-10-05 19:19:49 SYS AS SYSDBA>
```

Fine-grained auditing gives you even more control over data that is collected. However, it is not handled through the AUDIT_TRAIL at all. The DBA grants permissions by adding or removing FGA policies. But these details are beyond the scope of this exercise.

Creating Triggers for Auditing

A common example is to record changes to employee salaries-particularly large increases. That way, even if someone does circumvent security, the changes will still be flagged and stored in a separate table. You can monitor this table on a regular basis to identify potential problems. Assuming that the database has an Employees table with a column for Salary, the trigger code is straightforward.

1. To test this code, use the existing HR EMPLOYEES table.

2. Log in as the sys user as sysdba and create a table to hold audit values. Call it Salary_Audit, with columns for EmployeeID, OldSalary, NewSalary, Username, and ChangeDate.

```
Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:01:40 SYS AS SYSDBA> CREATE TABLE Salary_Audit(EmployeeID DECIMAL (16),
  2  OldSalary DECIMAL (16),
  3  NewSalary DECIMAL (16),
  4  UserName VARCHAR (50),
  5  ChangeDate DATE);

Table created.

2018-10-05 20:09:49 SYS AS SYSDBA>
```

3. Create a TRIGGER as follows (as sys user as sysdba):

```
SQL> CREATE OR REPLACE TRIGGER Audit_Salaries
       AFTER UPDATE ON HR.EMPLOYEES
       FOR EACH ROW
       BEGIN
         IF (:NEW.Salary > :OLD.Salary*1.20) THEN
         INSERT INTO Salary_Audit (EmployeeID, OldSalary, NewSalary,
                                   Username, ChangeDate)
            VALUES (:NEW.employee_id, :OLD.Salary,:NEW.Salary, user,
sysdate);
          END IF;
       END;
       /
```

```
2018-10-05 20:09:49 SYS AS SYSDBA> CREATE OR REPLACE TRIGGER Audit_Salaries
          AFTER UPDATE ON HR.EMPLOYEES
          FOR EACH ROW
          BEGIN
            IF (:NEW.Salary > :OLD.Salary*1.20) THEN
            INSERT INTO Salary_Audit (EmployeeID, OldSalary, NewSalary,
                                                        Username, ChangeDate)
                 VALUES (:NEW.employee_id, :OLD.Salary,:NEW.Salary, user, sysdate);
            END IF;
         END;
          /  2    3    4    5    6    7    8    9   10   11

Trigger created.

2018-10-05 20:12:12 SYS AS SYSDBA>
```

4. Now connect as the HR user and change one of the salaries by more than 20 percent.

```
SQL> UPDATE Employees
```

SET Salary=Salary*2
WHERE employee_id=100;
SQL> **COMMIT;**

```
2018-10-05 20:12:12 SYS AS SYSDBA> CONNECT hr/hr;
Connected.

Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:15:42 HR > UPDATE Employees
           SET Salary=Salary*2
         WHERE employee_id=100;  2    3

1 row updated.

2018-10-05 20:16:34 HR > COMMIT;

Commit complete.

2018-10-05 20:16:54 HR >
```

5. Connect again as the sys user as sysdba and check the Salary_Audit table to ensure that it recorded the changes.

   SQL> **SELECT * FROM Salary_Audit;**

```
2018-10-05 20:22:07 SYS AS SYSDBA> SELECT * FROM Salary_Audit;

EMPLOYEEID  OLDSALARY  NEWSALARY USERNAME                                                  CHANGEDATE
---------- ---------- ---------- -------------------------------------------------- -------------------
       100      24000      48000 HR                                                 2018-10-05 20:16:34

2018-10-05 20:22:20 SYS AS SYSDBA>
```

Study the changes listed in the audit table. If you do not get expected results, go back and check the trigger code carefully. For security purposes, the code really needs to be protected so that no one can see it. The solution is to wrap or "hide" your PL/SQL source code. "You can wrap PL/SQL source code with either the wrap utility or DBMS_DDL subprograms. The wrap utility wraps a single source file, such as a SQL*Plus script. The DBMS_DDL subprograms wrap a single dynamically generated PL/SQL unit, such as a single CREATE PROCEDURE statement." (“Oracle® Database PL/SQL Language Reference 11g Release 1 (11.1),” n.d.). Also remember that you will have to set access rights carefully for both the Employees and Salary_Audit tables. In fact, you should simply give users Execute permissions on the new procedure and make sure that they have no access to the underlying table.

Note: You can restore the record you updated in the HR EMPLOYEES table by running the following commands (as HR):

SQL> **UPDATE Employees**
SET Salary=24000
WHERE employee_id=100;
SQL> **COMMIT;**

```
2018-10-05 20:22:20 SYS AS SYSDBA> CONNECT hr/hr;
Connected.

Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:24:40 HR > UPDATE Employees
           SET Salary=24000
           WHERE employee_id=100;  2    3

1 row updated.

2018-10-05 20:25:23 HR > COMMIT;

Commit complete.

2018-10-05 20:25:43 HR >
```

Auditing Scott

Run the following commands in SQL*Plus. Start by connecting as the sys user as sysdba. Your results should be similar to the results shown below each command.

1. SQL> **show parameter aud**

```
2018-10-05 20:26:59 SYS AS SYSDBA> show parameter aud

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
audit_file_dest                      string      /home/oracle/app/oracle/admin/
                                                 orcl/adump
audit_sys_operations                 boolean     TRUE
audit_syslog_level                   string
audit_trail                          string      DB_EXTENDED
2018-10-05 20:27:34 SYS AS SYSDBA>
```

NAME TYPE VALUE
---

audit_file_dest	string	/home/oracle/app/oracle/admin/orcl/adump
audit_sys_operations	boolean	TRUE
audit_syslog_level	string	
audit_trail	string	DB_EXTENDED

2. SQL> **audit all by scott;**

```
2018-10-05 20:27:34 SYS AS SYSDBA> audit all by scott;

Audit succeeded.

2018-10-05 20:33:07 SYS AS SYSDBA>
```

Audit succeeded.

3. SQL> **connect scott/tiger**

Connected.

Note: Unlock the Scott account or change the password if needed.

4. SQL> **create table test2(col1 int not null);**

```
2018-10-05 20:36:14 SYS AS SYSDBA> CONNECT scott/scott1
Connected.

Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:36:34 SCOTT > create table test2(col1 int not null);

Table created.

2018-10-05 20:38:10 SCOTT >
```

Table created.

5. SQL> **connect system/<password>**

```
2018-10-05 20:38:10 SCOTT > CONNECT system/oracle;
Connected.

Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:39:57 SYSTEM > 
```

Connected.

6. SQL> **column obj_name format a20**
   SQL> **column action_name format a30**
   SQL> **column obj_name format a30**
   SQL> **select username, obj_name, action_name**
   from dba_audit_trail
   where username='SCOTT';

```
2018-10-05 20:39:57 SYSTEM > column obj_name format a20
2018-10-05 20:41:53 SYSTEM > column action_name format a30
2018-10-05 20:42:06 SYSTEM > column obj_name format a30
2018-10-05 20:42:21 SYSTEM > select username, obj_name, action_name from dba_audit_trail where username='SCOTT';

USERNAME                       OBJ_NAME                       ACTION_NAME
------------------------------ ------------------------------ ------------------------------
SCOTT                                                         LOGON
SCOTT                                                         LOGON
SCOTT                          TEST2                          CREATE TABLE
SCOTT                                                         LOGOFF

2018-10-05 20:42:50 SYSTEM > 
```

USERNAME	OBJ_NAME	ACTION_NAME
SCOTT		LOGON
SCOTT	TEST2	CREATE TABLE

7. **ON YOUR OWN - please do a NOAUDIT on SCOTT**.

```
2018-10-05 20:50:39 SYSTEM > NOAUDIT all BY scott;

Noaudit succeeded.

2018-10-05 20:51:53 SYSTEM > 
```

Fine-Grained Auditing

Try the following example dealing with FGA. Fine-Grained Auditing is similar to writing trigger code, but it requires less effort and provides additional controls. Audit trail records can be retrieved from the DBA_FGA_AUDIT_TRAIL view. You need DBA permissions to configure FGA policies and to view the audit trail. Most operations are handled through the DBMS_FGA package. For example, you could create a policy to monitor SQL operations on the salary field of the Employees table.

1. Create the policy.

```
SQL> BEGIN
     DBMS_FGA.ADD_POLICY(
     Object_schema => 'HR',
     Object_name => 'Employees',
     Policy_name => 'Monitor_Employee_Table',
     Audit_column => 'Salary',
     Statement_types => 'select, insert, update, delete');
     END;
     /
```

```
2018-10-05 20:51:53 SYSTEM > BEGIN
DBMS_FGA.ADD_POLICY(
Object_schema => 'HR',
Object_name => 'Employees',
Policy_name => 'Monitor_Employee_Table',
Audit_column => 'Salary',
Statement_types => 'select, insert, update, delete');
END;
/  2    3    4    5    6    7    8    9

PL/SQL procedure successfully completed.

2018-10-05 20:54:21 SYSTEM >
```

---

2. Log in as the hr user.

   SQL> **connect hr/<password>**

```
2018-10-05 20:54:21 SYSTEM > CONNECT hr/hr
Connected.

Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:55:37 HR >
```

3. Test the policy. Select the first name, last name and salary of the employee with the employee id '200'.

SQL> **select first_name, last_name, salary**
from employees where
employee_id=200;

```
2018-10-05 20:55:37 HR > select first_name, last_name, salary
from employees where
employee_id=200;
  2    3
FIRST_NAME           LAST_NAME                     SALARY
-------------------- ------------------------- ----------
Jennifer             Whalen                          4400

2018-10-05 20:56:50 HR >
```

4. Connect again as the sys user as sysdba.

SQL> **connect sys as sysdba**
Enter password:

5. Query the audit trail and view the audit records.

SQL> **SELECT policy_name, object_name, statement_type, db_user**
FROM dba_fga_audit_trail;

```
2018-10-05 20:56:50 HR > CONNECT sys as sysdba
Enter password:
Connected.

Session altered.

You are running SQL*Plus in directory /home/oracle

2018-10-05 20:58:04 SYS AS SYSDBA> SELECT policy_name, object_name, statement_type, db_user FROM dba_fga_audit_trail;

POLICY_NAME                  OBJECT_NAME                                                                                                  STATEME
DB_USER
---------------------------- -------------------------------------------------------------------------------------------------------------- -------
----------------------------
MONITOR_EMPLOYEE_TABLE       EMPLOYEES                                                                                                    SELECT H
R

2018-10-05 20:58:52 SYS AS SYSDBA>
```

6. Drop the FGA policy

SQL> **exec dbms_fga.drop_policy(-**
object_schema => 'HR', -
object_name => 'Employees', -
policy_name => 'Monitor_Employee_Table' -
);

```
2018-10-05 20:58:52 SYS AS SYSDBA> exec dbms_fga.drop_policy( -
object_schema => 'HR', -
object_name => 'Employees', -
policy_name => 'Monitor_Employee_Table' -
);> > > >

PL/SQL procedure successfully completed.

2018-10-05 21:00:54 SYS AS SYSDBA>
```

References:

1. Perry, J. T., & Post, G. V. (2007). Introduction to Oracle 10G. Pearson/Prentice Hall.

2. Oracle® Database PL/SQL Language Reference 11g Release 1 (11.1). (n.d.). Retrieved from http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/wrap.htm