Oracle Database Coding Practices for Security and Data Security

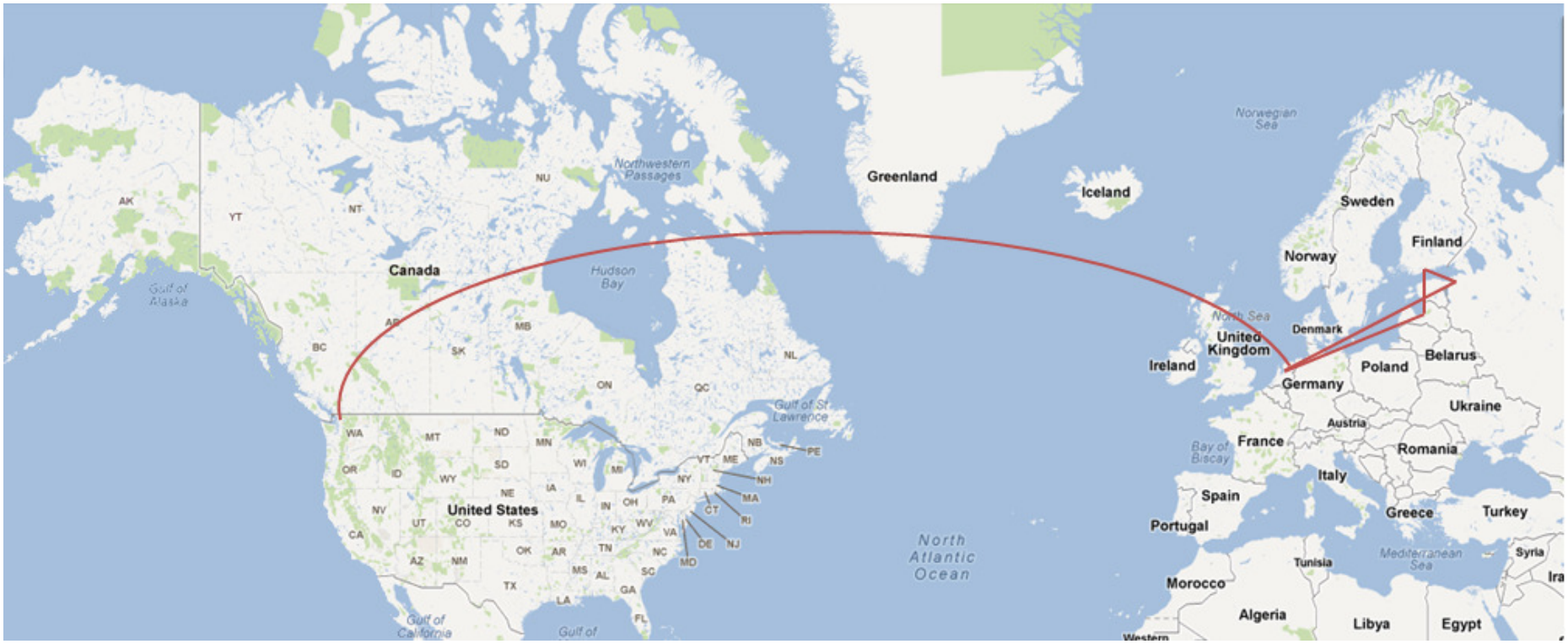
Daniel A. Morgan | damorgan12c@gmail.com | www.morganslibrary.org

Daniel A. Morgan

- Oracle ACE Director
- Consultant to Harvard University
- University of Washington Oracle Instructor, ret.
- The Morgan of Morgan's Library on the web
- Board Member: Western Washington OUG
- Upcoming Presentations
  - May 30-31: OUG Harmony Finland
  - Jun 1: OUG Harmony Latvia
  - Jun 20: Vancouver Canada
  - Jun 21: Victoria Canada
  - Sep: OpenWorld 2012: San Francisco

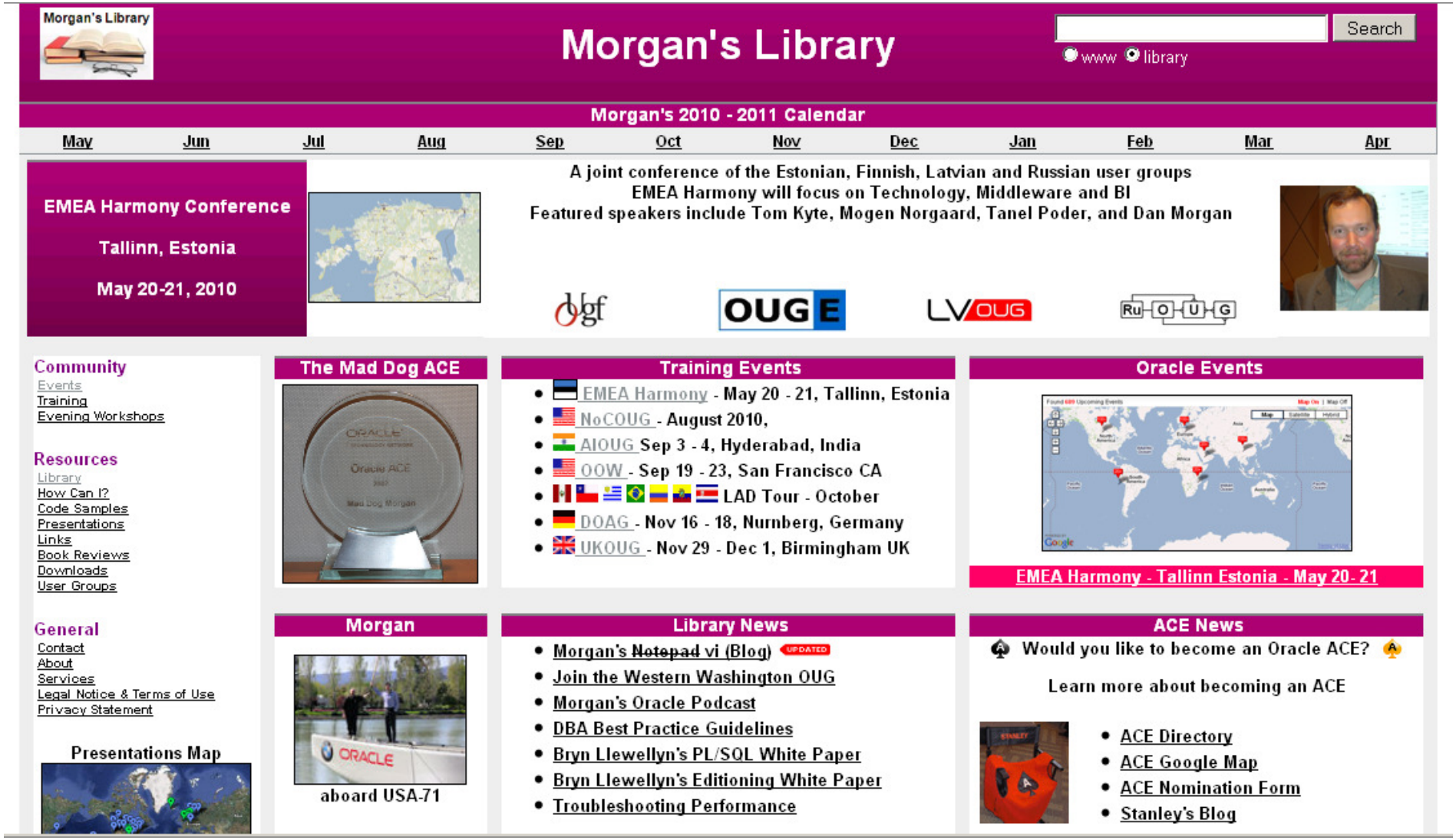
Official Beta Site ORACLE 11g DATABASE ORACLE RACSIG International zSeries Oracle SIG

cd $MORGAN_HOME



cd $MORGAN_HOME

Morgan's Library: www.morganslibrary.org



Syllabus

- SQL*NET
- System Event Triggers
  - SYS_CONTEXT Function
  - System Event Functions
- Fine Grained Access Control
- Fine Grained Auditing
- DBMS_ASSERT
- Password Verify Function
- Dynamic Password Generation
- Dynamic Transparent Encryption

Discussion

SQLNET Options

SQL*Net Connections

Inactive Session Expiration Time: Kills inactive SQLNET sessions.
If set to a non-zero value enables DCD (Dead Connection Detection)

```
sqlnet.expire_time = <integer_minutes>
sqlnet.expire_time=10
```

Encryption Client: Default: accepted

```
sqlnet.encryption_client = <accepted | rejected | requested | required>
sqlnet.encryption_client=required
```

Encryption Server: Default: accepted

```
sqlnet.encryption_server = <accepted | rejected | requested | required>
sqlnet.encryption_server=required
```

Encryption Type: Server
3des112 for triple DES with a two-key (112 bit) option, 3des168 for triple DES with a three-key (168 bit) option, des for standard 56 bit key size, des40 for 40 bit key size, rc4_40 for 40 bit key size, rc4_56 for 56 bit key size, rc4_128 for 128 bit key size, rc4_256 for 256 bit key size

```
sqlnet.encryption_types_server = <value>
sqlnet.encryption_types_server=(rc4_256)
```

Invited and Excluded Nodes

```
TCP.VALIDNODE_CHECKING=<yes/no>

TCP.VALIDNODE_CHECKING=yes


TCP.INVITED_NODES=<hostname | ip_address, hostname | ip_address, ...>

TCP.INVITED_NODES=(sales.us.acme.com, hr.us.mlib.com, 144.185.5.73)


TCP.EXCLUDED_NODES=<hostname | ip_address, hostname | ip_address,...>
TCP.EXCLUDED_NODES=(spammer.hacker.com, mktg.us.acme.com, 144.25.5.25)
```

Discussion

System Event Triggers

System Event Triggers

- Can fire on:
  - AFTER STARTUP
  - BEFORE SHUTDOWN
  - AFTER LOGON
  - BEFORE LOGOFF
  - AFTER DB_ROLE_CHANGE (Data Guard fail/switch-over)
  - AFTER SUSPEND
  - AFTER SERVERERROR

ON SERVERERROR Trigger

```
CREATE OR REPLACE TRIGGER logon_failures
 AFTER SERVERERROR
 ON DATABASE

 BEGIN
   IF (IS_SERVERERROR(1017)) THEN
     INSERT INTO connection_audit
     (login_date, user_name)
     VALUES
     (SYSDATE, 'ORA-1017');
   END IF;
 END logon_failures;
 /
```

SYS_CONTEXT

- SYS_CONTEXT (partial list)
  - AUTHENTICATED_IDENTITY
  - AUTHENTICATION_DATA
  - AUTHENTICATION_METHOD
  - CURRENT_SCHEMA and CURRENT_SCHEMAID
  - CURRENT_USER and CURRENT_USERID
  - ENTERPRISE_IDENTITY
  - GLOBAL_UID
  - HOST
  - IDENTIFICATION_TYPE
  - IP_ADDRESS
  - ISDBA
  - OS_USER
  - PROXY_GLOBAL_UID
  - PROXY_USER
  - PROXY_USERID

```
SELECT sys_context('USERENV', 'OS_USER')
FROM dual;
```

System Events

- Oracle created functions that work within the context of event triggers (partial list)
  - ORA_CLIENT_IP_ADDRESS
  - ORA_DES_ENCRYPTED_PASSWORD
  - ORA_LOGIN_USER
  - ORA_PRIVILEGE_LIST
  - ORA_REVOKEE
  - ORA_WITH_GRANT_OPTION

Discussion

Fine Grained Access Control

FGAC / Virtual Private Database / Row Level Security

- Uses the DBMS_RLS built-in package
  - ADD_GROUPED_POLICY
  - ADD_POLICY
  - ADD_POLICY_CONTEXT
  - CREATE_POLICY_GROUP
  - DELETE_POLICY_GROUP
  - DISABLE_GROUPED_POLICY
  - DROP_GROUPED_POLICY
  - DROP_POLICY
  - DROP_POLICY_CONTEXT
  - ENABLE_GROUPED_POLICY
  - ENABLE_POLICY
  - REFRESH_GROUPED_POLICY
  - REFRESH_POLICY

FGAC Policy Function

```
CREATE OR REPLACE FUNCTION vpd_sec(p_owner IN VARCHAR2, p_name IN VARCHAR2)
AUTHID DEFINER RETURN VARCHAR2 IS
BEGIN
  IF sys_context('userenv', 'session_user') IN ('UWCLASS') THEN
    RETURN NULL;
  ELSE
    RETURN '1=0';
  END IF;
END vpd_sec;
/
```

```
SELECT DISTINCT object_owner, object_name, predicate
FROM sys.v$vpd_policy
WHERE predicate IS NOT NULL
ORDER BY 1,2;
```

Discussion

Fine Grained Auditing

FGA

- Based on the DBMS_FGA built-in package
  - ADD_POLICY
  - DISABLE_POLICY
  - DROP_POLICY
  - ENABLE_POLICY

FGA_LOG$

```
SQL> desc fga_log$
 Name                          Null?    Type
 ----------------------------- -------- ----------------
 SESSIONID                     NOT NULL NUMBER
 TIMESTAMP#                             DATE
 DBUID                                  VARCHAR2(30)
 OSUID                                  VARCHAR2(255)
 OSHST                                  VARCHAR2(128)
 CLIENTID                               VARCHAR2(64)
 EXTID                                  VARCHAR2(4000)
 OBJ$SCHEMA                             VARCHAR2(30)
 OBJ$NAME                               VARCHAR2(128)
 POLICYNAME                             VARCHAR2(30)
 SCN                                    NUMBER
 SQLTEXT                                VARCHAR2(4000)
 LSQLTEXT                               CLOB
 SQLBIND                                VARCHAR2(4000)
 COMMENT$TEXT                           VARCHAR2(4000)
 PLHOL                                  LONG
 STMT_TYPE                              NUMBER
 NTIMESTAMP#                            TIMESTAMP(6)
 PROXY$SID                              NUMBER
 USER$GUID                              VARCHAR2(32)
 INSTANCE#                              NUMBER
 PROCESS#                               VARCHAR2(16)
 XID                                    RAW(8)
 AUDITID                                VARCHAR2(64)
 STATEMENT                              NUMBER
 ENTRYID                                NUMBER
 DBID                                   NUMBER
 LSQLBIND                               CLOB
 OBJ$EDITION                            VARCHAR2(30)
```

Discussion

DBMS_ASSERT

DBMS_ASSERT Built-in Package

- Used to counteract SQL Injection attacks
  - ENQUOTE_LITERAL
  - ENQUOTE_NAME
  - QUALIFIED_SQL_NAME
  - SCHEMA_NAME
  - SIMPLE_SQL_NAME
  - SQL_OBJECT_NAME

Discussion

Password Verify Function

$ORACLE_HOME/rdbms/admin/utlpwdmg.sql

```
-- strip the password of numbers and characters
lPwdTest := NLS_LOWER(
  TRANSLATE(password,'A!"#$%&()``*+,-/:;<=>?_0123456789', 'A'));

 -- test the stripped password is not based on a dictionary word
SELECT /* password_verify_dscm1 */ COUNT(*)
INTO I
FROM dbadmin.verify_function_dictionary vfd
WHERE INSTR(lPwdTest, vfd.source_word, 1, 1) <> 0;

IF i <> 0 THEN
  RAISE_APPLICATION_ERROR(-20008,
    'Password Must Be Substantially Different From A Dictionary Word');
END IF;

SELECT /* password_verify_dscm2 */ COUNT(*)
INTO I
FROM dice.verify_function_dictionary vfd
WHERE INSTR(REVERSE(lPwdTest), vfd.source_word, 1, 1) <> 0;

IF i <> 0 THEN
  RAISE_APPLICATION_ERROR(-20009,
   'Password Must Be Substantially Different From A Dictionary Word');
END IF;
```

Discussion

Dynamic Password Generation

Passwords

- We tend to think of security as relating to userids and passwords
- The biggest single problem with passwords: Database passwords, operating system passwords, PIN Number, etc. is that some human may choose it, knows it, write it down, and can pass it along to others. Take the human out of the equation and you get a lot closer to security
- On my current assignment DOT NET developers are hard-coding passwords into Web Application servers: Security by appearance only
- So here is a simple implementation of a solution to the problem. My actual production implementation is far more complex and if not unbreakable, nothing is, a lot closer to the goal than where most database implementations are today.

Passwords

- The heart of the solution is that any application, or user, wishing to connect to the database must first obtain the current password for a different schema by calling an "open" schema that provides access to nothing other than this one function
- The password received must then be used to log into a different schema, very quickly, because the password is again changed to a password unknown to any person

- Warning: This demo does not make use of bind variables or `DBMS_ASSERT` ... any serious implementation should do so
- It might also make use of `DBMS_DDL.CREATE_WRAPPED`

Dynamic Password Generator

```
CREATE OR REPLACE FUNCTION dbadmin.randombytes(byteType IN VARCHAR2)
RETURN VARCHAR2 AUTHID CURRENT_USER IS
 rStr VARCHAR2(50);
 btLen PLS_INTEGER;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  -- retrieve a 25 byte random string
  rStr := dbms_crypto.randombytes(25);

  -- retrieve the length of the input string
  btLen := length(byteType);
```

```
SELECT dbms_crypto.randombytes(25) FROM dual;
```

Dynamic Password Generator

```
    -- if the input string is 3 bytes
    IF btLen = 3 THEN
      -- set a new password that has never previously existed and is unknown
      -- to any humans
      execute immediate 'ALTER USER abc IDENTIFIED BY "' ||
                        SUBSTRB(rStr, btLen,30) || '"';

      -- schedule a job to run in 1 second that will change the password
      -- whether the current password is used or unused
      dbms_scheduler.create_job(
        job_name   => 'J' || SUBSTRB(rStr,40,5),
        start_date => SYSDATE + 1/86400,
        enabled    => TRUE,
        auto_drop  => TRUE,
        job_type   => 'PLSQL_BLOCK',
        job_action =>
'DECLARE x VARCHAR2(30); BEGIN x := dbadmin.randombytes(''XXX''); END; ');
```

Dynamic Password Generator

```
  -- if the input string is 4 bytes
  ELSIF btLen = 4 THEN
  ...
  -- if the input string is 5 bytes
  ELSIF btLen = 5 THEN
  ...
  END IF;
   RETURN rStr;
END randombytes;
/
```

Note the function returns the full 50 byte string ... the application trying to log in must know how to parse the string to obtain the actual password, something no human can possibly accomplish fast enough, especially if one locks accounts based on the profile setting for FAILED LOGIN ATTEMPTS and uses an AFTER SERVERERROR system event trigger to trap ORA-01017s and notify security administration.

Dynamic Password Generator Test

```
-- run the function
SELECT dbadmin.randombytes('ABC')
FROM dual;

-- substring out the password portion of the returned string
SELECT substr('3496D98355A39400D4FF7D5BF2EFB4379198DC8A80FFF32DC5',3,30)
FROM dual;

-- use it very quickly to log on
conn scott/"96D98355A39400D4FF7D5BF2EFB437"

-- monitor the job that will reset the password to an unknown value
col job_name format a10
col next_run_date format a35
select job_name, state, next_run_date, sysdate
FROM user_scheduler_jobs;

-- attempt to re-use the password that seconds before had been valid
conn scott/"96D98355A39400D4FF7D5BF2EFB437"
```

Discussion

Dynamic Transparent Encryption

The story of a table named "CREDIT_CARDS"

- - -

not be to confused with Oracle's Transparent Data Encryption

Step 1: Create Application Users

```
CREATE USER lunar
IDENTIFIED BY lunar
DEFAULT TABLESPACE mydata
TEMPORARY TABLESPACE temp
QUOTA 1G ON mydata;

GRANT create session, create table, create procedure,
create trigger, create view TO lunar;

GRANT execute ON dbms_crypto TO lunar;

GRANT insert, references, select, update, delete
ON website.credit_cards TO lunar WITH GRANT OPTION;

CREATE USER svc_lunar
IDENTIFIED BY svc_lunar
TEMPORARY TABLESPACE temp;

GRANT create session;
```

Step 2: Create and Load Seed Table

```
-- create seed table
CREATE TABLE lunar.async_hashmap (rid NUMBER(4,0),
checksums VARCHAR2(1000));

ALTER TABLE lunar.async_hashmap
ADD CONSTRAINT pk_async_hashmap
PRIMARY KEY(rid);

-- load seed table
BEGIN
  FOR i in 1 .. 1000 LOOP
    INSERT INTO lunar.async_hashmap
    VALUES (i, dbms_crypto.randombytes(500));
  END LOOP;
  COMMIT;
END;
/

ALTER TABLE lunar.async_hashmap READ ONLY;
```

Step 3 Create Application Data Objects

```
CREATE OR REPLACE VIEW website.payments AS
SELECT *
FROM website.credit_cards;

CREATE TABLE lunar.eclipse (
member_id CHAR(32) NOT NULL,
seed      RAW(128) NOT NULL,
cc_no     RAW(128) NOT NULL);

ALTER TABLE lunar.eclipse
ADD CONSTRAINT pk_eclipse
PRIMARY KEY (member_id);
```

Step 4: Create The Control Package Specification

```
CREATE OR REPLACE PACKAGE lunar.craters AUTHID DEFINER IS
 PROCEDURE get_checksum(pCcid IN CHAR, pSrc IN VARCHAR2);
 FUNCTION  set_checksum(pSrc IN RAW, pOffset1 IN RAW) RETURN VARCHAR2;
 PROCEDURE put_checksum;
END craters;
/
```

Step 5a: Create The Control Package Body

```
CREATE OR REPLACE PACKAGE BODY lunar.craters wrapped
a000000
b2
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
b
d3d 526
YWTFdT+Rsl2oEz3eQdVpsiofYvMwg80rBUoF3y/NA52sIPerlUtMimG6Fx1QXYVmWfdWoZUI
ty+ORKMOLEtPmu7pbxChg8qFDAUQpRhqWIJqgh8laZ8OEsra165d/pr55IaGZKJTlRVU16ko
b4Uc5IV7cFt40o04WbhmjjikDbJ//8JLI655L378cAxdFomALQcAtHlULI+iVgArHteuMo7S
7K6eXcEXFb/t/VgAiGScDgbGWq95HBz6Hm0u0mZ3hZ+K8kgWLNc3vgfj4gU8Nw+cHNgcjmRR
9GMYHj86QZf75TIDhzsrMgPlmdnXJAMUrtAkQWIIMJXzH+OpJ6xbrKUoInx8hxENTIpwlnes
L7PTvSe7mMVLVLZjHRcaoyQhHsbszsusbzVp7g6phCA2jiWxOXiRrAypkuffhu0QgIftNDQ0
7Pda/3Qclx7MsnNICRnUc70N53iX0OdnIRVU6qwQZksEjR29R7ZlzoVH8czBgPWrBMBq/0K6
LipD7Fj+QORLJh5nSpt7PbLRowoSBqAan6FVoS0tjOIeJpPbUEuiYjxdTm7+BDWXhXOD5bkC
lPQRCXnf+r9ZFNQG3ZJ2Xp4ddY7U+0grADargDffy6aqiP2NyxfjAQas3IrsGwIhhJwUgBht
8Tp+M7uL78VKdwQ+BLJN71QKeYHJAyMjIxWPP/cWfdYPWccxdzCFn2lWtI/zl8lyVaSM38de
RDUuVs6v/ztP2rX93zPDKTEIMsuwZ+UlboTyMFf6TphXZKsGrhtWwOhqfp66Os+B7UdoyJDm
adaJ9K6/+hd/Itd8u6JKtHCQYRqVtVIADZf+bcN9gN0zUG+nR+EfEVligA3DKULgjrq+0B7e
ny92HONGHDlqJgdYxMr2YtuJYLsJ9rbQan82KP4AF8ZNIPsXB0iMeasEvoLgC27wsgCBTSlR
1Okaa1uXKov7UbhJGo4nJAn/XkmUW/3aRiAlhTcoYLTDaO0rvPby+r0V8I5W2mKtq8WDao2E
EKaNLRi87QG2AdPu/wqiaQnp3MZnftShVkm93mxVhnrHQZKvTOG7RuWMlUJWytqopNn2GEhG
F0EG/7KTobUiC1K7p2H2Uga5dPU1j3lXRG1SAw6bYELgfHlxyKvpF8lJ2dUjjM7xhsr6osa7
9M39mjebKWF0DRy8WLkS7l7rpzu+7T+bXdroZYObbUkoPMQlOJDGaLAPNbW+6nIh3vlDxLXM
FhziMthfcTNUtjApqrlELwoR5UGdqp7Pc81Bk0W8aHIBc7VV+Wy/+EfJyPXKRAEFu/nhL5AR
uA==
/
```

Step 5b: Encryption Code

```
PROCEDURE get_checksum(pCcid IN CHAR, pSrc IN VARCHAR2) IS
 l_int INTEGER;
 l_key RAW(128);
 l_mod INTEGER := dbms_crypto.encrypt_aes256 + dbms_crypto.chain_cbc + dbms_crypto.pad_pkcs5;
 l_offset1 INTEGER := ROUND(dbms_random.value(1,1000),0);
 l_offset2 INTEGER := ROUND(dbms_random.value(1, 967),0);
 l_raw RAW(128);

 InvalidSrc    EXCEPTION;
BEGIN
  IF LENGTH(pSrc) NOT IN (15,16) THEN
    RAISE InvalidSrc;
  END IF;

  BEGIN
    l_int := TO_NUMBER(pSrc);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE InvalidSrc;
  END;

  SELECT /* LUNAR.CRATER1 */ utl_raw.cast_to_raw(SUBSTR(checksums, l_offset2, 32))
  INTO l_key
  FROM lunar.async_hashmap
  WHERE rid = l_Offset1;

  l_raw := dbms_crypto.encrypt(utl_raw.cast_to_raw(pSrc), l_mod, l_key);

  INSERT /* LUNAR.CRATER2 */ INTO lunar.eclipse
  (member_id, seed, cc_no)
  VALUES
  (pCcid, l_raw, l_key);
  ...
END get_checksum;
```

Step 6: Instead-Of Trigger On View

```
CREATE OR REPLACE TRIGGER lunar.rover
INSTEAD OF INSERT
ON website.payments
FOR EACH ROW
BEGIN
  INSERT /* LUNAR.ROVER */ INTO website.credit_cards
  (credit_card_id, credit_card_type_id, billing_addr_id, card_number_hint,
   expiration_month, expiration_year, cardholder_name, credit_card_token,
   member_id, credit_card_number, clear_number, deleted, create_date,
   update_date, flags, status, credit_card_bin)
  VALUES
  (:NEW.credit_card_id, :NEW.credit_card_type_id, :NEW.billing_addr_id,
   :NEW.card_number_hint, :NEW.expiration_month, :NEW.expiration_year,
   :NEW.cardholder_name, :NEW.credit_card_token,
   :NEW.member_id,
   CASE WHEN (:NEW.credit_card_type_id IN (6, 9, 10, 11, 12, 13, 14)) THEN
   :NEW.credit_card_number ELSE '???' END,
   :NEW.clear_number, :NEW.deleted, :NEW.create_date,
   :NEW.update_date, :NEW.flags, :NEW.status, :NEW.credit_card_bin);

  IF :NEW.credit_card_type_id NOT IN (6, 9, 10, 11, 12, 13, 14) THEN
    lunar.craters.get_checksum(:NEW.credit_card_id, :NEW.credit_card_number);
  END IF;
END rover;
/
```

Questions

ERROR at line 1:
ORA-00028: your session has been killed

All demo code at morganslibrary.org

Thank you