Eggshell Home Builder (Final Project)

Database User Guide:

**Advanced Database Management:** December 9, 2013

(Group 3)

Gummoe, Benjamin

Malavia, Sonal

Shaw, Katarina

Zhai, Xuefeng

Table of Contents

[1. Objective 1](#_Toc374331501)

[2. Requirements 1](#_Toc374331502)

[3. Set-up 2](#_Toc374331503)

[4. Tests 2](#_Toc374331504)

[a. Test Set 1: Owner Role: 2](#_Toc374331505)

[b. Test Set 2: sales\_agent Role: 3](#_Toc374331506)

[c. Test Set 3: project\_manager Role: 3](#_Toc374331507)

# Objective

This user guide will walk you through the testing process for the Eggshell Home Builder database. The Eggshell Homebuilder database allows employees to track the Eggshell inventory of subdivisions and houses, sales, and construction projects. The database also allows customers to make design decisions and view the progress of their construction project.

# Requirements

* 1. Oracle Database 11g XE 11. 2 or newer
  2. Windows OS.
  3. Group 3’s Eggshell Homebuilder files.

# Set-up

*Setup is used to create the database with dummy data. The following database objects are created in this step: tables, table constraints, sequences and sequence triggers, views, procedures, functions, package, database triggers, scheduled jobs, roles, and alternate indexes.*

* 1. Download all files into a single directory.
  2. Generate and populate the Eggshell Database by running the following line in SQL Plus:

START <filepath>/G3\_eggshell.sql;

* 1. Be sure to run the following command before beginning the tests:

set serveroutput on

# Tests

*Each role and the objects that the role has access to are tested independently.*

*Note: The grey box are the commands that should type into the SQL Plus.*

# Test Set 1: Database Owner:

* + 1. **Function 1**: (cal\_employee\_no) Calculate the number of employees that currently work at Eggshell using a procedure that calls this function.

SQL>EXEC employee\_number;

* + 1. **Report 1:** View the project manager assignment report.

SQL>START <filepath>/G3\_Report1.SQL;

SQL>EXEC PM\_assignment;

* + 1. **Job 1:** The table automatic update about what construction projects have been or will be completed today. (Currently, the frequency of the job is set to 30 seconds for testing)

SQL>SELECT \* FROM CONSTRUCTION\_END\_TODAY;

* + 1. Repot2: Start the report for project manager to execute later;

SQL>START <filepath>/G3\_Report2.SQL;

# Test Set 2: sales\_agent Role:

* + 1. **Log** into Sales Manager role.
       1. Username: sa\_example
       2. Password: sa\_example\_pw
       3. SQL>ALTER SESSION SET current\_schema = <database owner role>;
       4. SET SERVEROUTPUT ON;
    2. **View 1:** View all empty lots.

SQL>SELECT \* FROM VIEW\_EMPTYLOT;

* + 1. **Package 1:** List all the sales for the second sales agent.

SQL>EXEC list\_sales.sales(2);

* + 1. Run the following statement. It should fail, because the Sales Agent role does not have access to this table:

SQL>SELECT \* FROM TASK;

# Test Set 3: project\_manager Role:

* + 1. **Log** into Project Manager role.
       1. Username: pm\_example
       2. Password: pm\_example\_pw
       3. SQL>ALTER SESSION SET current\_shema = <database owner role>;
       4. SET SERVEROUTPUT ON;
    2. **View 2**: View the customer information for all unfinished houses.

SQL>SELECT \* FROM VIEW\_UNFINISHED\_HOUSES;

* + 1. **Report 2**: View the construction status report of the first home. The house is unfinished.

SQL>EXEC display\_home\_status(1);

* + 1. **Report 2:** View the construction status report of the second home. The house is complete.

EXEC display\_home\_status(2);

* + 1. Run the following statement. It should fail, because the Project manager role does not have access to this table:

SELECT \* FROM CONTRACT;

# Test Set 4: test triggers:

**Return to your database owner role before testing these 2 triggers.**

set serveroutput on

i. **Trigger 1:** Trigger ‘selected\_stage\_option\_check’ ensures that options with a last allowed stage value that are too far behind the current project stage aren’t added to the selected\_stage\_option table.

Run the following statement. It should fail, because the option has a last allowed stage of 4, but the construction project is in stage 7.

INSERT INTO selected\_stage\_option

(selected\_date, option\_choice\_id, construction\_project\_stage\_id)

VALUES(sysdate, 1, 11);

**i**i. **Trigger 2:** Trigger ‘selected\_stage\_option\_markup’ ensures that options with a last allowed stage values that are only one stage behind the current project stage are added, but with a 15% price markup. Run the following statement:

INSERT INTO selected\_stage\_option

(selected\_date, option\_choice\_id, construction\_project\_stage\_id)

VALUES(sysdate, 1, 9);

Next, run the following statement to confirm that the option was added and compare the entered price against the original price:

SELECT C.construction\_project\_stage\_id,

C.stage\_id,

O.last\_allowed\_stage\_id,

S.customer\_price,O.price,

S.option\_choice\_id,

S.selected\_stage\_option\_id

FROM CONSTRUCTION\_PROJECT\_STAGE c

JOIN SELECTED\_STAGE\_OPTION S ON

C.CONSTRUCTION\_PROJECT\_STAGE\_ID=S.CONSTRUCTION\_PROJECT\_STAGE\_ID

JOIN OPTION\_CHOICE O ON

O.option\_choice\_id = S.option\_choice\_id;