**Object Description**

**Overview**

|  |  |
| --- | --- |
| **Object Type** | **Object Name** |
| Sequences | style\_id\_seq  elevation\_id\_seq |
| Views | option\_view  construction\_stage |
| Procedures | ongoing\_task  house\_price\_query (house\_id) |
| Function | avg\_price\_by\_style (style\_name) |
| Package | Procedure: fire\_employee (employee\_id)  Procedure:hire\_employee (employee\_id) |
| Triggers | contract\_completion\_date  constructionsheet\_stage |
| Scheduled Job | task\_schedule |
| Roles | sales\_representative  construction\_mgr  buyers |
| De-normalization | Materialized view: construction\_progress |
| Alternate Indexes | subdivision\_idx  room\_idx |

**2 Sequences**

* **style\_id\_seq**: a sequence of consecutive numbers starting from 1 and increasing by 1; can serve as surrogate key for table style (the primary key is style\_name).

ALTER TABLE style ADD style\_id NUMBER(4,0);

DROP SEQUENCE style\_id\_seq;

CREATE SEQUENCE style\_id\_seq

INCREMENT BY 1;

UPDATE style

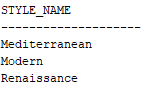
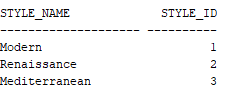
SET style\_id = style\_id\_seq.NEXTVAL;

COMMIT;

SELECT \*

FROM style;

Before sequence was added After sequence was added

* **elevation\_id\_seq**: a sequence of consecutive numbers starting from 1 and increasing by 1; can serve as surrogate key for table elevation (the primary key is elevation\_name).

ALTER TABLE elevation ADD elevation\_id NUMBER(4,0);

DROP SEQUENCE elevation\_id\_seq;

CREATE SEQUENCE elevation\_id\_seq

INCREMENT BY 1;

UPDATE elevation

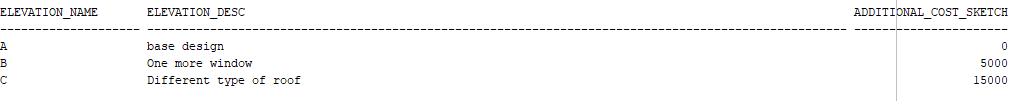
SET elevation\_id = elevation\_id\_seq.NEXTVAL;

COMMIT;

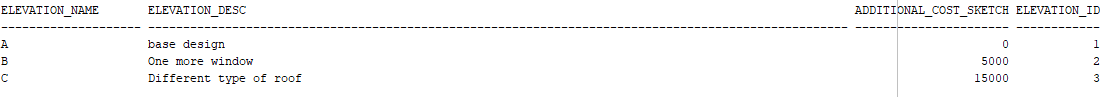
SELECT \*

FROM elevation;

Before sequence was added



After sequence was added

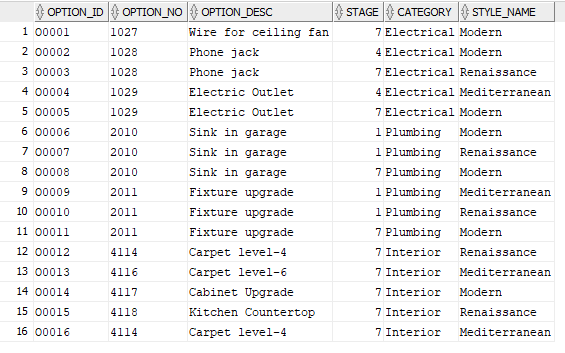
****

**2 Views**

* **option\_view:** A view based on table “Option” that restricts users(house buyers) from seeing the cost and revision date of options.

CREATE OR REPLACE VIEW option\_view AS

SELECT option\_id, option\_no, option\_desc, stage, category, style\_name FROM "Option";

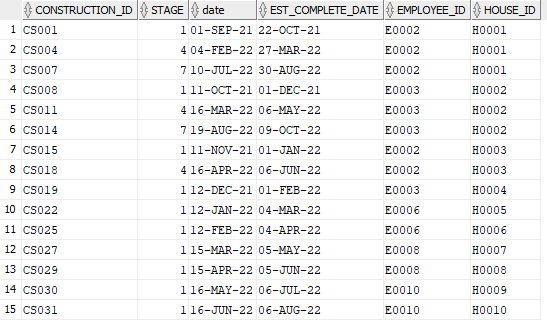


* **construction\_stage:** A view based on table constructionsheet that only shows users the constructions on stage 1, 4, or 7.

CREATE OR REPLACE VIEW construction\_view AS

SELECT \* FROM constructionsheet

WHERE stage = 1 OR stage = 4 OR stage = 7;



These view create a restricted subset of data to show users part of the dataset. It will not affect the performance of the original tables.

**2 Procedures**

* **ongoing\_task:** a procedure that shows all construction managers who have ongoing constructions (percentage completed less than 100), their ongoing task, and the percentage completed.

CREATE OR REPLACE

PROCEDURE ongoing\_task

AS TYPE ongoing\_type IS RECORD

(

construction\_id1 constructionsheet.construction\_id%TYPE,

employee\_id1 constructionsheet.employee\_id%TYPE,

fullname employee.fname%TYPE,

house\_id1 constructionsheet.house\_id%TYPE,

task\_id1 taskprogress.task\_id%TYPE,

percent\_complete1 taskprogress.percent\_complete%TYPE,

task\_desc1 task.task\_desc%TYPE);

ongoing\_task\_record ongoing\_type;

CURSOR task\_cursor IS

SELECT c.construction\_id, c.employee\_id, CONCAT(CONCAT(e.fname,' '),

e.lname) AS fullname, c.house\_id, tp.task\_id, tp.percent\_complete, t.task\_desc

FROM constructionsheet c, taskprogress tp, task t, employee e

WHERE c.construction\_id = tp.construction\_id

AND tp.task\_id = t.task\_id

AND c.employee\_id = e.employee\_id

AND tp.percent\_complete < 100;

BEGIN

OPEN task\_cursor;

LOOP

FETCH task\_cursor INTO ongoing\_task\_record;

EXIT WHEN task\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('+ +');

DBMS\_OUTPUT.PUT\_LINE('Employee ID: '||ongoing\_task\_record.employee\_id1);

DBMS\_OUTPUT.PUT\_LINE('Employee Name: '||ongoing\_task\_record.fullname);

DBMS\_OUTPUT.PUT\_LINE('Construction ID: '||ongoing\_task\_record.construction\_id1);

DBMS\_OUTPUT.PUT\_LINE('House ID: '||ongoing\_task\_record.house\_id1);

DBMS\_OUTPUT.PUT\_LINE('Task ID: '||ongoing\_task\_record.task\_id1);

DBMS\_OUTPUT.PUT\_LINE('Task Description: '||ongoing\_task\_record.task\_desc1);

DBMS\_OUTPUT.PUT\_LINE('Percent Complete: '||ongoing\_task\_record.percent\_complete1);

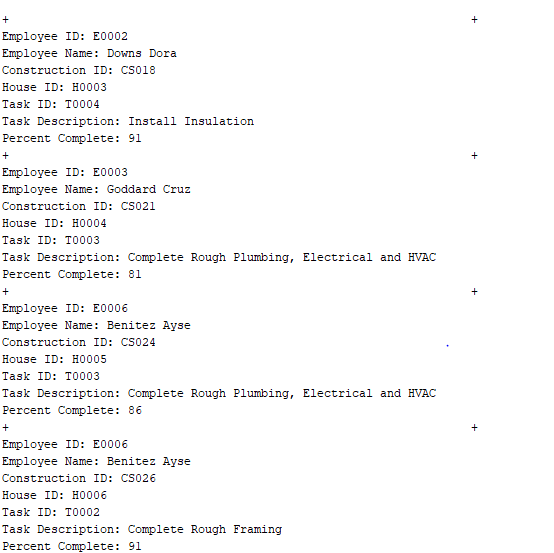
END LOOP;

CLOSE task\_cursor;

END;

/

Sample Output:



* **house\_price\_query:** Sa procedure that shows house base price, elevation price, each decorator choice price and the total price of the chosen house. It takes input parameter = house\_id

CREATE OR REPLACE

PROCEDURE house\_price\_query(houseid1 IN house.house\_id%TYPE)

AS

house\_id house.house\_id%TYPE;

house\_price house.base\_price%TYPE;

elevation\_price elevation.Additional\_cost\_sketch%TYPE;

decoration\_count NUMBER(2) :=1; -- use to number decoration 1, 2, ...

total\_cost NUMBER(15); -- sum the total price

is\_empty boolean := true; -- to check is the cursor (decoration record is empty)

-- Use cursor because a house could have several kinds of decoration

-- Note: a house can only have 1 base price and elevation price

CURSOR c1(houseid2 house.house\_id%TYPE)

IS

SELECT o.option\_desc, d.item\_price

FROM house h, "Option" o, decoratorchoice d

WHERE h.house\_id(+) = d.house\_id

AND d.option\_id = o.option\_id

AND h.house\_id = houseid2;

BEGIN

-- if a price is null, change it to 0

SELECT h.house\_id, COALESCE(h.base\_price,0), COALESCE(e.Additional\_cost\_sketch,0)

INTO house\_id, house\_price, elevation\_price

FROM house h, lot l, elevation e

WHERE e.elevation\_name = l.elevation\_name

AND l.house\_id = h.house\_id

AND h.house\_id = houseid1

GROUP BY h.house\_id, h.base\_price, e.Additional\_cost\_sketch;

DBMS\_OUTPUT.PUT\_LINE('House ID: '||house\_id);

DBMS\_OUTPUT.PUT\_LINE('House Base Price: $'||house\_price);

DBMS\_OUTPUT.PUT\_LINE('Elevation Price: $'||elevation\_price);

total\_cost := house\_price + elevation\_price; -- do the sum

-- get all the decorations and print them out

FOR item in c1(houseid1)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Decoration '||decoration\_count||': '||item.option\_desc);

DBMS\_OUTPUT.PUT\_LINE(' Price: $'||item.item\_price);

total\_cost := total\_cost + item.item\_price; -- do the sum

decoration\_count := decoration\_count+1; -- add the decoration number

is\_empty := false; -- if the cursor is not empty, change to false

END LOOP;

IF is\_empty THEN

DBMS\_OUTPUT.PUT\_LINE('No decoration yet');

END IF;

DBMS\_OUTPUT.PUT\_LINE('Total: $'|| total\_cost);

EXCEPTION

WHEN no\_data\_found THEN

DBMS\_OUTPUT.PUT\_LINE('NO such house');

ROLLBACK;

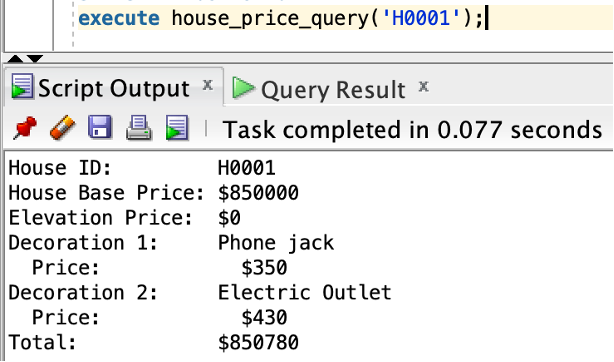
END;

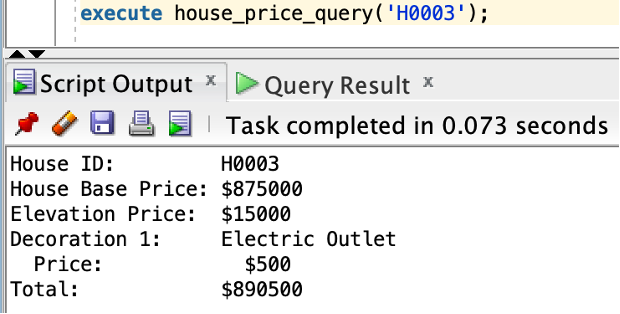
-- execute house\_price\_query('H0001');

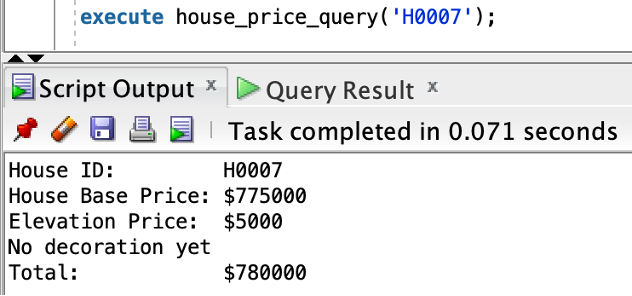
-- execute house\_price\_query('H0003');

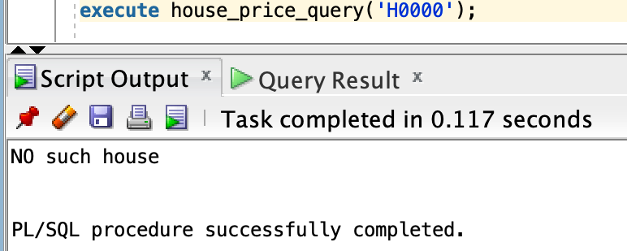
-- execute house\_price\_query('H0007');

-- execute house\_price\_query('H0000');









**1 Function**

* **avg\_price\_by\_style**: a function that takes in style name as one parameter, calculates the average base price of all houses on file of the given style, and returns the price.

CREATE OR REPLACE FUNCTION avg\_price\_by\_style (style\_n VARCHAR)

RETURN NUMBER

IS

avg\_price NUMBER;

BEGIN

SELECT AVG(base\_price)

INTO avg\_price

FROM style JOIN lot USING(style\_name)

JOIN house h USING(house\_id)

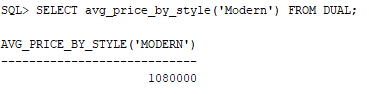
GROUP BY style\_name

HAVING style\_name = style\_n;

RETURN avg\_price;

END avg\_price\_by\_style;

Sample Output:



**1 Package**:

* **Hiring and Firing employees**

By inserting a new employee’s Imployee\_id, firstname, lastname, position, licences, we can successfully add those information to employee table in our database.

By inserting employee’ ID, the company can fire that person and delete related information in the database if the employee no longer serving any jobs.

CREATE OR REPLACE PACKAGE c\_package AS

PROCEDURE fire\_employee(employeeid IN employee.employee\_id%TYPE);

PROCEDURE hire\_employee(employeeid IN employee.employee\_id%TYPE,

lastname IN employee.lname%TYPE,

firstname IN employee.fname%TYPE,

title IN employee.title%TYPE,

license IN employee.license\_no%TYPE);

END c\_package;

/

CREATE OR REPLACE PACKAGE BODY c\_package AS

-- first procedure starts

PROCEDURE hire\_employee(employeeid IN employee.employee\_id%TYPE,

lastname IN employee.lname%TYPE,

firstname IN employee.fname%TYPE,

title IN employee.title%type,

license IN employee.license\_no%TYPE)

IS

BEGIN

INSERT INTO employee

VALUES(employeeid, lastname, firstname, title, license);

DBMS\_OUTPUT.PUT\_LINE('Employee hired: '||employeeid||' '||firstname||' '||lastname);

END hire\_employee;

-- first procedure ends

-- second procedure starts

PROCEDURE fire\_employee(employeeid IN employee.employee\_id%TYPE)

IS

constructor\_manager\_ornot NUMBER;

sells\_manager\_ornot NUMBER;

decorator\_choice\_ornot NUMBER;

no\_number EXCEPTION;

BEGIN

SELECT count(\*) INTO decorator\_choice\_ornot

FROM employee e, decoratorchoice d

WHERE e.employee\_id = d.employee\_id

AND e.employee\_id = employeeid;

SELECT count(\*) INTO constructor\_manager\_ornot

FROM employee e, constructionsheet c

WHERE e.employee\_id = c.employee\_id

AND e.employee\_id = employeeid;

SELECT count(\*) INTO sells\_manager\_ornot

FROM employee e, contract c

WHERE e.employee\_id = c.employee\_id

AND e.employee\_id = employeeid;

IF constructor\_manager\_ornot != 0 THEN

DBMS\_OUTPUT.PUT\_LINE('You can not fire a construction manager!');

ELSIF sells\_manager\_ornot != 0 THEN

DBMS\_OUTPUT.PUT\_LINE('You can not fire a sales manager!');

ELSIF decorator\_choice\_ornot != 0 THEN

DBMS\_OUTPUT.PUT\_LINE('You can not fire a decorator choice manager!');

ELSE

DBMS\_OUTPUT.PUT\_LINE('FIRE '||employeeid);

DELETE FROM employee

WHERE employee.employee\_id = employeeid;

IF SQL%NOTFOUND THEN

RAISE no\_number;

END IF;

END IF;

EXCEPTION

WHEN no\_number THEN

DBMS\_OUTPUT.PUT\_LINE('NO such employee');

ROLLBACK;

END fire\_employee;

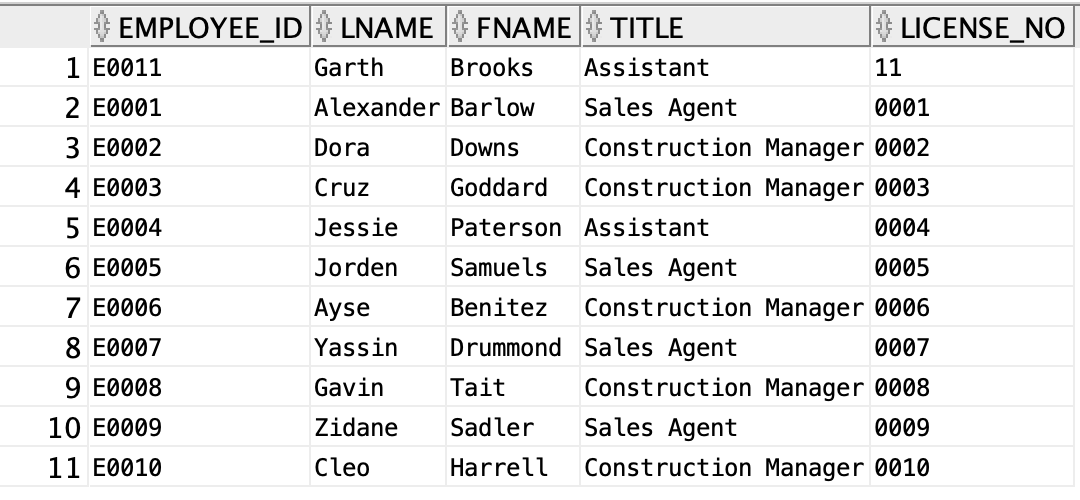
--second procedure ends

END c\_package;

/

Sample output:

* Adding a new employee named Garth Brooks with a title of assitant.
* Deleting employee E0010 and results are the same in that this employee is still in charge of some jobs and can not be fired.



**2 Triggers**

* **contract\_completion\_date**: This trigger will check if the new records inserted to and records updated in the table “contract” are valid before insertion and update. Valid records will have est\_complete\_date within 1 year after signed\_date. If the record is valid, it will be successfully inserted or updated. A message will be printed out to inform users about the success of insert/update. And, for insertion, only new est\_complete\_date will be printed out. For update, both new record’s and old record’s est\_complete\_date and signed\_date will be printed out. If the record is not valid (i.e., est\_complete\_date is beyond 1 year of signed\_date), an exception will be raised and the error message will be printed out.

REM Trigger1: check contract's estimate completion date

CREATE OR REPLACE TRIGGER

contract\_completion\_date

BEFORE INSERT OR UPDATE ON contract /\* contract table\*/

FOR EACH ROW

DECLARE

unacceptable\_est\_completion EXCEPTION;

BEGIN

IF (TO\_DATE(:new.est\_complete\_date) < add\_months(TO\_DATE(:new.signed\_date), 12))

THEN

DBMS\_OUTPUT.PUT\_LINE('Contract Added/Updated');

DBMS\_OUTPUT.PUT\_LINE('New Estimated Complete Date: ' || :new.est\_complete\_date);

DBMS\_OUTPUT.PUT\_LINE('New Signed Date: ' || :new.signed\_date);

DBMS\_OUTPUT.PUT\_LINE('Old Estimated Complete Date: ' || :old.est\_complete\_date);

DBMS\_OUTPUT.PUT\_LINE('Old Signed Date: ' || :old.signed\_date);

ELSE

RAISE unacceptable\_est\_completion;

END IF;

EXCEPTION

WHEN unacceptable\_est\_completion THEN RAISE\_APPLICATION\_ERROR (-20326,'There is an error in input estimated completion date. It must be within 1 year of signed date. ');

END contract\_completion\_date;

/

ALTER TRIGGER contract\_completion\_date ENABLE;

REM Insert a house with no contract to test on the trigger

INSERT INTO house

VALUES ('H0011', 549500.00, NULL,500.00, NULL,1);

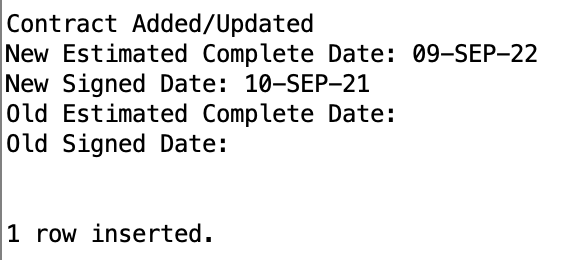
1 row inserted.

REM insert into contract with est\_completion\_date within required date

INSERT INTO contract

VALUES

('C0011', 'H0011', 'B0001', 'E0001', 'Portfolio Loan', 850000.00, 'EA001', 'BA003', '9-Sep-2022', '10-Sep-2021', 1, 1, 1);



REM delete from contract if the record to be inserted already exist

DELETE FROM contract

WHERE contract\_id = 'C0011';

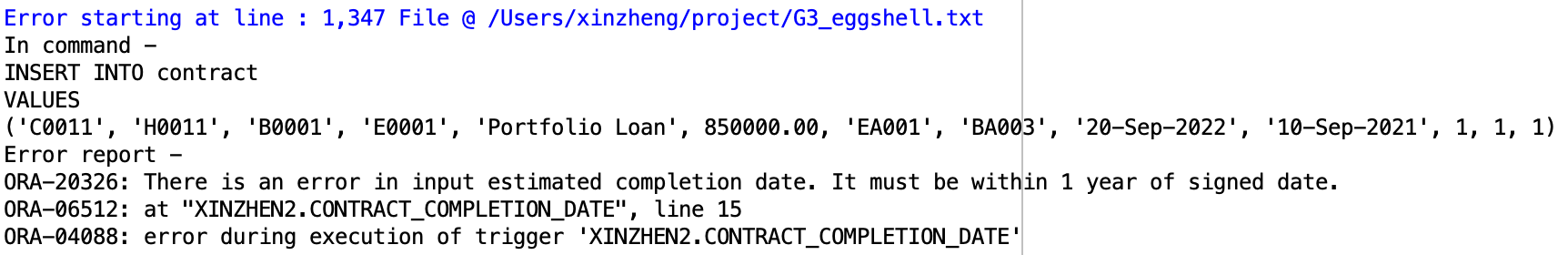
1 row deleted.

REM insert into contract with est\_completion\_date outside required date

INSERT INTO contract

VALUES

('C0011', 'H0011', 'B0001', 'E0001', 'Portfolio Loan', 850000.00, 'EA001', 'BA003', '20-Sep-2022', '10-Sep-2021', 1, 1, 1);



delete from house

WHERE house\_id = 'H0011';

1 row deleted.

* **Constructionsheet\_stage**: This trigger will check if the new record inserted to and the record updated in the table “constructionsheet” have a valid stage before insertion and update. The stage updated or inserted will need to be greater than the nearest stage by 1 according to the corresponding house number. For example, if the most recent stage of any task is 2, the newly inserted/updated stage must be 3. If the record is valid, a message informing the user about the success will be printed out. And, for insertion, the new stage will be printed out. For update, both the new stage and old stage will be printed out. If the record is invalid, an exception will be raised.

REM Trigger2

CREATE OR REPLACE TRIGGER

constructionsheet\_stage

BEFORE INSERT OR UPDATE ON constructionsheet

FOR EACH ROW

DECLARE

v\_max\_stage constructionsheet.stage%TYPE;

unacceptable\_stage EXCEPTION;

BEGIN

SELECT MAX(stage) INTO v\_max\_stage

FROM constructionsheet

WHERE :new.house\_id = constructionsheet.house\_id;

IF :new.stage = v\_max\_stage + 1 THEN

DBMS\_OUTPUT.PUT\_LINE('ConstructionSheet Added/Updated');

DBMS\_OUTPUT.PUT\_LINE('New Stage: ' || :new.stage);

DBMS\_OUTPUT.PUT\_LINE('Old Stage: ' || :old.stage);

ELSE

RAISE unacceptable\_stage;

END IF;

EXCEPTION

WHEN unacceptable\_stage THEN RAISE\_APPLICATION\_ERROR (-20326,'There is an error in the input stage. Stage must be greater than the current stage by 1.');

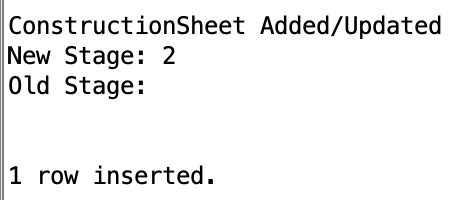
END constructionsheet\_stage;

/

INSERT INTO constructionsheet

VALUES

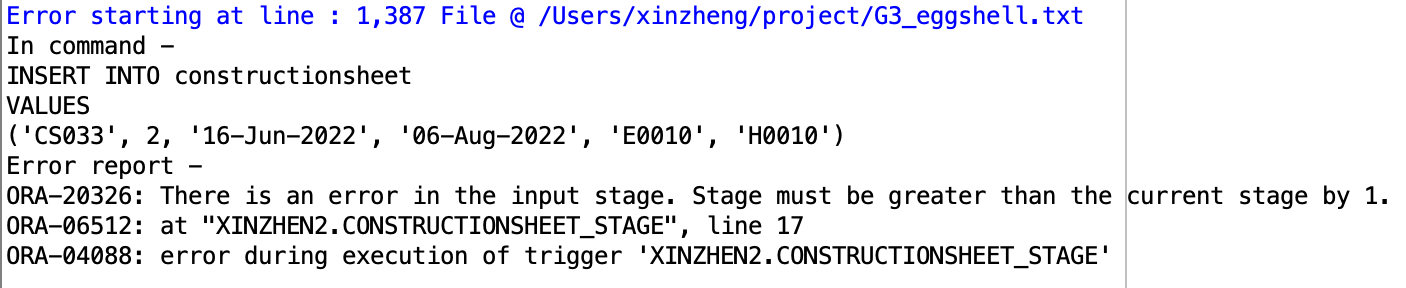
('CS032', 2, '16-Jun-2022', '06-Aug-2022', 'E0010', 'H0010');



INSERT INTO constructionsheet

VALUES

('CS033', 2, '16-Jun-2022', '06-Aug-2022', 'E0010', 'H0010');



DELETE FROM constructionsheet where construction\_id = 'CS032';

**1 Scheduled Job**

* **task\_schedule:** A schedule job to update the column percent\_complete in tabletaskprogress. It is used to update the task progress on a daily basis. The scheduled job adds 1 to all percentage completed that are less than 100 every day starts from the job created date.

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB (

job\_name => 'task\_schedule',

job\_type => 'PLSQL\_BLOCK',

job\_action => 'UPDATE taskprogress

SET percent\_complete = percent\_complete+1

WHERE percent\_complete <100;',

start\_date => sysdate,

repeat\_interval => 'FREQ=DAILY',

enabled => true);

END;



**3 Roles**

* **sales\_representative:** The role of sales representative will be able to see all the tables except for the table “employee”. This role will be able to select, insert, update and delete the tables “decoratorchoice”, “contract”, “house”, and “buyer”. A sales representative needs to create, modify or delete contracts and houses. The sales representative also needs to create, modify, or delete a buyer and create, modify or delete decorator choice sheets. This role will be able to select from “subdivision”, “"Subdivision-Style", “style”, “Option”, “Style-Elevation”, “schooldistrict”, “school”, “house”, “lot”, “elevation”, “room”, “constructionsheet”, “task”, “agent”, “bank”, and “employee”.   
    
  As the role **sales\_representative** has been created, during the actual test, it is possible that an error will be raised since the role has been created. Please change the name of the role if you would like to correctly create the role and grant it to a new user. In our run, the role is successfully created and the result is attached below along with the granting process to the new user and how to test it from the new user’s account.

REM create a role of sales representative

CREATE ROLE sales\_representative;

GRANT select, insert, update, delete ON decoratorchoice TO sales\_representative;

GRANT select, insert, update, delete ON contract TO sales\_representative;

GRANT select, insert, update, delete ON buyer TO sales\_representative;

GRANT SELECT, insert, update, delete ON house TO sales\_representative;

GRANT SELECT ON subdivision TO sales\_representative;

GRANT SELECT ON "Subdivision-Style" TO sales\_representative;

GRANT SELECT ON style TO sales\_representative;

GRANT SELECT ON "Option" TO sales\_representative;

GRANT SELECT ON "Style-Elevation" TO sales\_representative;

GRANT SELECT ON schooldistrict TO sales\_representative;

GRANT SELECT ON school TO sales\_representative;

GRANT SELECT ON lot TO sales\_representative;

GRANT SELECT ON elevation TO sales\_representative;

GRANT SELECT ON room TO sales\_representative;

GRANT SELECT ON constructionsheet TO sales\_representative;

GRANT SELECT ON task TO sales\_representative;

GRANT SELECT ON agent TO sales\_representative;

GRANT SELECT ON bank TO sales\_representative;

GRANT SELECT ON employee TO sales\_representative;

To test the role, we grant the role “sales\_representative” to the user advdb. To ensure the role has been successfully granted, open a new window and switch to the user advdb. Try to update the table “sudivision” on the other user’s schema. An error will be raised indicating that the user advdb does not have privilege to do so. Then, test to insert a new record into the table “house” on the other user’s scheme. The row will be successfully inserted.

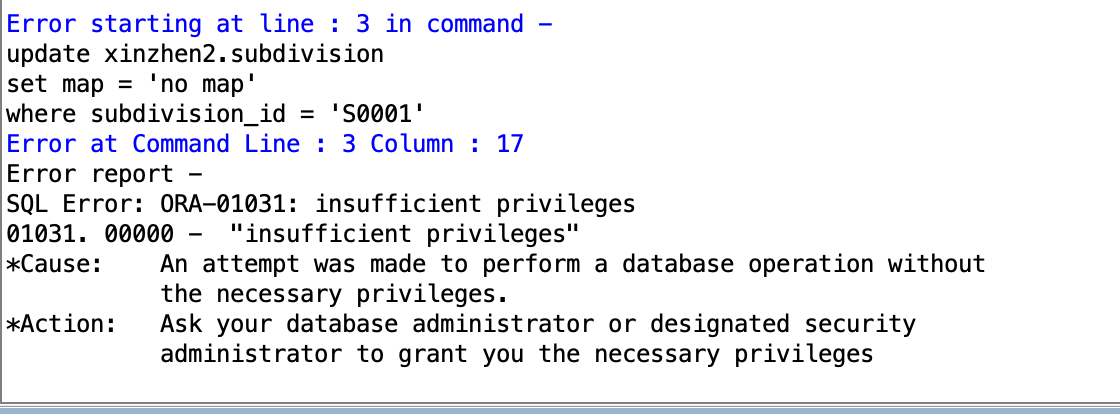
GRANT sales\_representative TO advdb;

REM update with NO priviledge

update SCHEMA\_USERNAME.subdivision

set map = 'no map'

where subdivision\_id = 'S0001';



REM Insertion with privilege

INSERT INTO SCHEMA\_USERNAME.house

VALUES

('H0011', 789000, NULL, 800, NULL, 1);

1 row inserted.

DELETE FROM SCHEMA\_USERNAME.house

WHERE house\_id = 'H0011';

1 row deleted.

* **Construction\_Mgr:** The role of construction will be able to see all the tables except for the table “employee”. This role will be able to select, insert, update and delete the tables “constructionsheet”, “task”, “room”, and “house”, “elevation”, “subdivision”, “Subdivision-Style”, “style”, “Option”, and “Style-Elevation”. A construction manager needs to create, modify or delete houses. The construction manager also needs to create, modify, or delete any construction related table with the house. This role will be able to select from “lot”, “schooldistrict”, “school”, “agent”, “bank”, “decoratorchoice”, and “contract”.

As the role **Construction\_Mgr** has been created, during the actual test, it is possible that an error will be raised since the role has been created. Please change the name of the role if you would like to correctly create the role and grant it to a new user. In our run, the role is successfully created and the result is attached below along with the granting process to the new user and how to test it from the new user’s account.

REM create role of construction manager

CREATE ROLE construction\_mgr;

GRANT select, insert, update, delete ON constructionsheet TO construction\_mgr;

GRANT select, insert, update, delete ON task TO construction\_mgr;

GRANT select, insert, update, delete ON room TO construction\_mgr;

GRANT select, insert, update, delete ON house TO construction\_mgr;

GRANT select, insert, update, delete ON elevation TO construction\_mgr;

GRANT select, insert, update, delete ON subdivision TO construction\_mgr;

GRANT select, insert, update, delete ON "Subdivision-Style" TO construction\_mgr;

GRANT select, insert, update, delete ON style TO construction\_mgr;

GRANT select, insert, update, delete ON "Option" TO construction\_mgr;

GRANT select, insert, update, delete ON "Style-Elevation" TO construction\_mgr;

GRANT select ON lot TO construction\_mgr;

GRANT select ON schooldistrict TO construction\_mgr;

GRANT select ON school TO construction\_mgr;

GRANT select ON agent TO construction\_mgr;

GRANT select ON bank TO construction\_mgr;

GRANT select ON decoratorchoice TO construction\_mgr;

GRANT select ON contract TO construction\_mgr;

GRANT select ON buyer TO construction\_mgr;

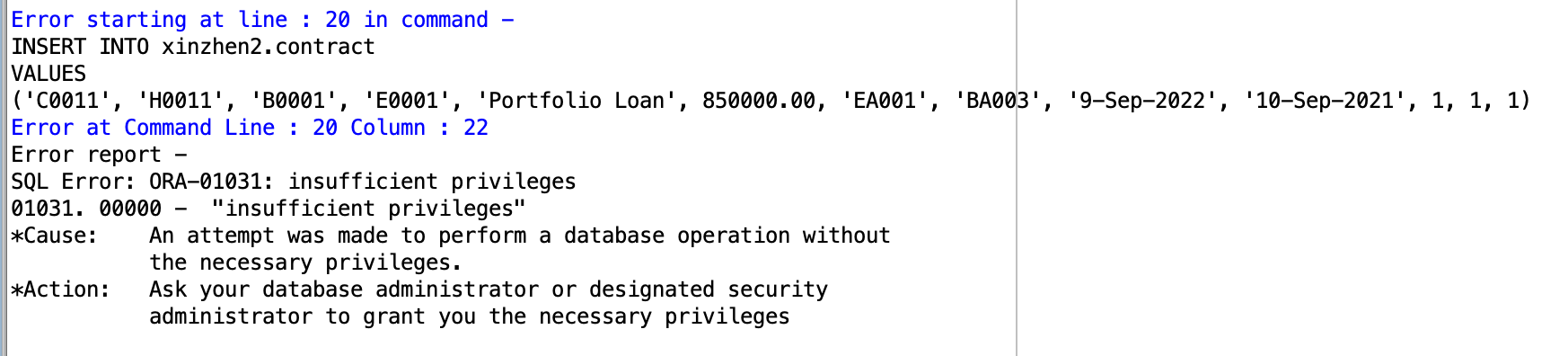
To test the role, we grant the role “construction\_mgr” to the user advdb. To ensure the role has been successfully granted, open a new window and switch to the user advdb. Try to update the table “contract” on the other user’s schema. An error will be raised indicating that the user advdb does not have privilege to do so. Then, test to insert a new record into the table “style” on the other user’s scheme. The row will be successfully inserted.

GRANT construction\_mgr TO advdb;

INSERT INTO SCHEMA\_USERNAME.contract

VALUES

('C0011', 'H0011', 'B0001', 'E0001', 'Portfolio Loan', 850000.00, 'EA001', 'BA003', '9-Sep-2022', '10-Sep-2021', 1, 1, 1);



INSERT INTO SCHEMA\_USERNAME.style

VALUES

('new style', 4);

1 row inserted.

* **Buyers:** The role of buyers will be able to see the tables related to house. They will only be able to see construction stage 1, 4, and 7. Two views are created for users to see the construction progress. The buyers will be able to see tables “subdivision”, “Subdivision-Style”, “ style”, “Style-Elevation”, “schooldistrict”, “school”, “house”, “lot”, “elevation”, and “room”. The buyers will be able to see two views: “construction\_stage” and “option\_view”.

As the role **Buyers** has been created, during the actual test, it is possible that an error will be raised since the role has been created. Please change the name of the role if you would like to correctly create the role and grant it to a new user. In our run, the role is successfully created and the result is attached below along with the granting process to the new user and how to test it from the new user’s account.

REM create role of buyer

CREATE ROLE buyers;

GRANT SELECT ON subdivision TO buyers;

GRANT SELECT ON "Subdivision-Style" TO buyers;

GRANT SELECT ON style TO buyers;

GRANT SELECT ON "Style-Elevation" TO buyers;

GRANT SELECT ON schooldistrict TO buyers;

GRANT SELECT ON school TO buyers;

GRANT SELECT ON house TO buyers;

GRANT SELECT ON lot TO buyers;

GRANT SELECT ON elevation TO buyers;

GRANT SELECT ON room TO buyers;

GRANT select ON construction\_stage to buyers;

GRANT SELECT ON option\_view to buyers;

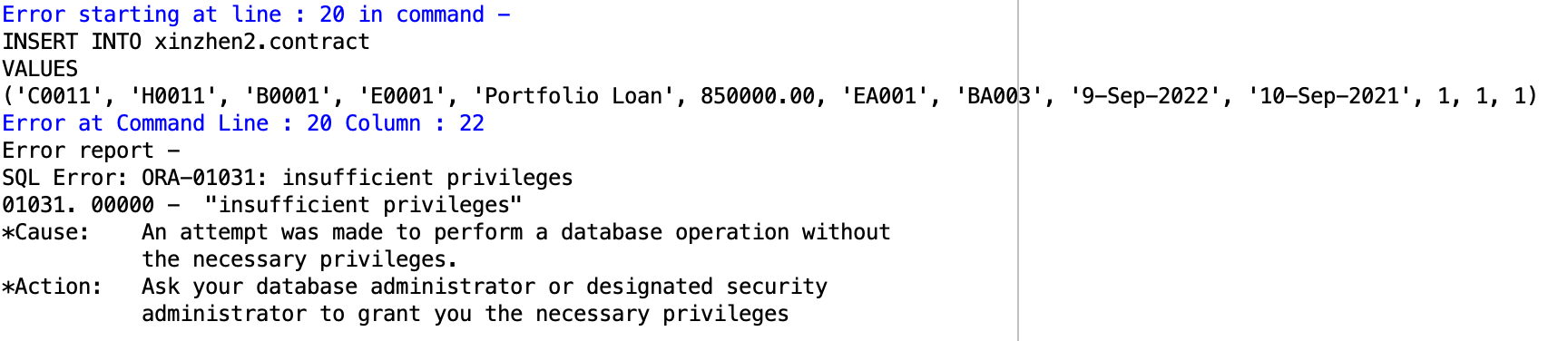
To test the role, we grant the role “buyers” to the user advdb. To ensure the role has been successfully granted, open a new window and switch to the user advdb. Try to update the table “contract” on the other user’s schema. An error will be raised indicating that the user advdb does not have privilege to do so. Then, test to select a new record to see the view “construction\_stage” on the other user’s scheme.

GRANT buyers TO advdb;

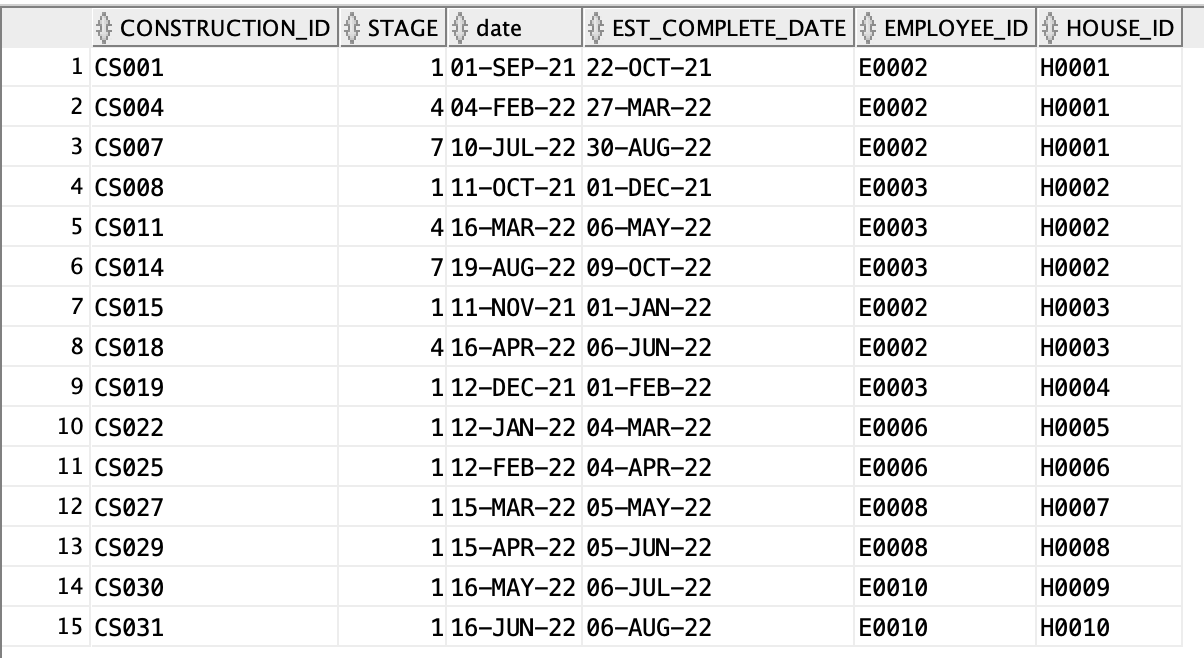
INSERT INTO SCHEMA\_USERNAME.contract

VALUES

('C0011', 'H0011', 'B0001', 'E0001', 'Portfolio Loan', 850000.00, 'EA001', 'BA003', '9-Sep-2022', '10-Sep-2021', 1, 1, 1);



SELECT \* FROM xinzhen2.construction\_stage;



**1 De-normalization**

* We denormalized the originally normalized entities ConstructionSheet, TaskProgress and Task by creating a materialized view construction\_progress. This view joined three tables to aggregate all tasks and task progress for constructions, which can lead to repeated/redundant information for construction\_id and task\_id but can also improve the read performance of our database, although at the expense of losing some write performance.
* In this case, when database users want to know the the tasks progress of one specific house, either in one specific stage or all stages, they don’t have to join all three tables mentioned above but directly retrieve the information from the materialized view construction\_progress, increasing query efficiency. This view will be updated whenever there’s an update in any of the three tables.

CREATE MATERIALIZED VIEW construction\_progress

REFRESH ON COMMIT

AS

SELECT c.house\_id, c.stage, t.task\_desc,tp.percent\_complete

FROM constructionsheet c, taskprogress tp, task t

WHERE c.construction\_id = tp.construction\_id

AND t.task\_id = tp.task\_id;

Before denormalization

SELECT c.house\_id, c.stage, t.task\_desc, tp.percent\_complete

FROM constructionsheet c JOIN taskprogress tp

USING(construction\_id)

JOIN task t USING(task\_id)

WHERE c.house\_id = 'H0003';

After denormalization

SELECT \*

FROM construction\_progress

WHERE house\_id = 'H0003';

**2 Alternate Indexes**

* **subdivision\_idx**: unique alternate index for attribute subdivision\_name in table subdivision because subdivision\_name is a candidate key

CREATE INDEX subdivision\_idx ON subdivision (subdivision\_name);

This alternate index can help users retrieve information about subdivisions much more quickly without scanning through the whole table if they only know the subdivision names but not subdivision id, which is the primary key, thus making the whole process more time-saving and efficient.

Sample Query:

SELECT subdivision\_id,subdivision\_name, map

FROM subdivision

WHERE subdivision\_name = ‘Palm Springs’;

* **room\_idx**: unique alternate index for attribute room\_name in table room because room\_name is a candidate key

CREATE INDEX room\_idx ON room (room\_name);

This alternate index can help users retrieve information about room much more quickly without scanning through the whole table if they only know the room names but not the specific room id, which is the primary key, thus making the whole process more time-saving and efficient..

Sample Query:

SELECT hr.house\_id, r.room\_name, hr.floor,

hr."size", hr.num\_window

FROM room r JOIN "House-Room" hr USING(room\_id)

WHERE r.room\_name = 'Study';