GROUP 1

Tony Eardley 14417434

Josh Challand 14414449

Julius Prakelis 16430034

Ryan Haskins 16442673

Contents

[Schema Design 2](#_Toc511999147)

[Skeleton Tables Design 2](#_Toc511999148)

[Proposed Automation Strategy 5](#_Toc511999149)

[Evidence of Additional Research 5](#_Toc511999150)

[Screenshots 6](#_Toc511999151)

[PROCEDURES 6](#_Toc511999152)

[Procedures Functionality: 6](#_Toc511999153)

[FUNCTIONS 9](#_Toc511999154)

[Functions Functionality: 9](#_Toc511999155)

[TRIGGERS 11](#_Toc511999156)

[Triggers Functionality: 11](#_Toc511999157)

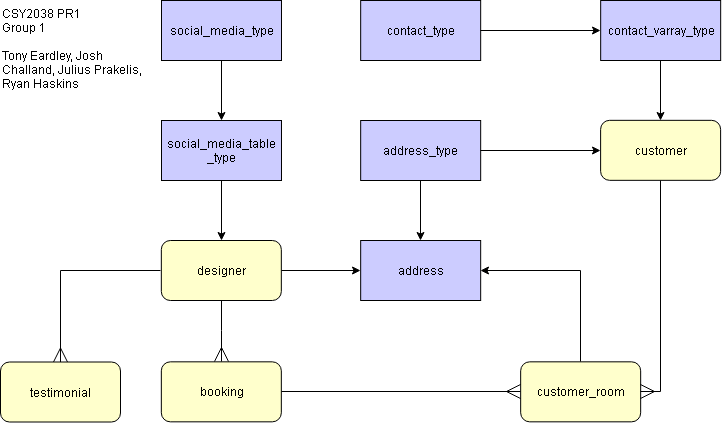
[Test Plan 12](#_Toc511999158)

[References 17](#_Toc511999159)

[Appendix 17](#_Toc511999160)

# Schema Design

One of the initial stage of creating the database is the creation of a schema, detailed below is the schema of which the database system is derived from. Built from the initial ERM, the schema builds on top of the already established entities and their relationship by adding objects to be used by the database and enhance its functionality, efficiency and design.



# Skeleton Tables Design

The following table details the design of the database objects and tables, there attributes, datatypes, and constraints providing the initial groundwork to build the database from scratch. Effectively being a blueprint for the entire project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Attribute | Key | Datatype | Constraints/Defaults |
| DESIGNERS | designer\_id | PK | NUMBER(8) |  |
|  | firstname |  | VARCHAR2(25) | NOT NULL |
|  | lastname |  | VARCHAR2(25) | NOT NULL |
|  | address |  | ref of address\_type |  |
|  | social\_media |  | social\_media\_table\_type |  |
|  | pay\_rate |  | NUMBER(4,2) | DEFAULT 7.50 |
|  | hire\_date |  | DATE | DEFAULT SYSDATE |
|  | dob |  | DATE | NOT NULL |
|  | | | | |
| BOOKINGS | booking\_id | PK | NUMBER(8) |  |
|  | designer\_id | FK | NUMBER(8) | NOT NULL |
|  | cost |  | NUMBER(10,2) |  |
|  | completed |  | CHAR(1) | NOT NULL  CHECK (‘Y‘ || ‘N’’)  DEFAULT ‘N’ |
|  | order\_date |  | DATE | DEFAULT SYSDATE |
|  | arrival\_date |  | DATE |  |
|  | | | | |
| CUSTOMERS | customer\_id | PK | NUMBER(8) |  |
|  | title |  | VARCHAR2(10) |  |
|  | firstname |  | VARCHAR2(25) | NOT NULL |
|  | lastname |  | VARCHAR2(25) | NOT NULL |
|  | username |  | VARCHAR2(25) |  |
|  | address |  | address\_type |  |
|  | contact |  | contact\_varray\_type |  |
|  | | | | |
| CUSTOMER\_ROOMS | customer\_id | PK FK | NUMBER(8) | NOT NULL |
|  | booking\_id | PK  FK | NUMBER(8) | NOT NULL |
|  | room\_width |  | NUMBER(6) | NOT NULL |
|  | room\_length |  | NUMBER(6) | NOT NULL |
|  | address |  | ref of address\_type |  |
|  | | | | |
| TESTIMONIALS | testimonial\_id | PK | NUMBER(8) |  |
|  | designer\_id |  | NUMBER(10) | NOT NULL |
|  | review |  | VARCHAR2(125) |  |
|  | rating |  | NUMBER(2) | NOT NULL  CHECK( rating < 11) |
|  | date\_created |  | DATE | DEFAULT SYSDATE |
|  | | | | |
| ADDRESSES | address\_type: | | | |
|  | house\_number |  | VARCHAR2(25) |  |
|  | street |  | VARCHAR2(25) |  |
|  | city |  | VARCHAR2(25) |  |
|  | postcode |  | VARCHAR2(25) |  |
|  | country |  | VARCHAR2(15) |  |
|  | | | | |
| SOCIAL\_MEDIA\_TABLE\_TYPE | social\_media\_type: | | | |
|  | media\_name |  | VARCHAR2(25) |  |
|  | contact |  | VARCHAR2(50) |  |
|  | | | | |
| CONTACT\_VARRAY\_TYPE | contact\_type: | | | |
|  | contact\_type |  | VARCHAR2(15) |  |
|  | contact\_method |  | VARCHAR2(15) |  |
|  | contact\_info |  | VARCHAR2(50) |  |

The following table details the sequences, functions, procedures, triggers, and Queries that were identified to be useful when implementing the design of the system. They offer key functionality, that enhances the databases maintainability and efficiency whilst providing the means to extract useful information.

|  |  |
| --- | --- |
| **Sequences, Functions, Procedures, Triggers, and Queries** | |
| **Name** | **Description** |
| **Sequences** | |
| designers\_seq | Starts with 10000000 and increments by 1 |
| bookings\_seq | Starts with 40000000 and increments by 1 |
| testimonials\_seq | Starts with 30000000 and increments by 1 |
| customers\_seq | Starts with 20000000 and increments by 1 |
| customer\_rooms\_seq | Starts with 50000000 and increments by 1 |
| **Functions** | |
| func\_retire\_age  (in\_dob designers.dob%TYPE, in\_retire\_age NUMBER)  RETURN NUMBER | Function determines the months left until retirement based upon the dob and retirement age passed in its parameters. |
| func\_booking\_cost  (in\_total\_area NUMBER, in\_pay\_rate designers.pay\_rate%TYPE, in\_avg\_rate testimonials.rating%TYPE)  RETURN NUMBER | Function takes in the total area of a room, a pay rate of a designer, and average rating of said designer. The function then uses the three arguments to generate a total cost which is return to the calling routine. |
| func\_total\_area  (in\_length customer\_rooms.room\_length%TYPE, in\_width customer\_rooms.room\_width%TYPE)  RETURN NUMBER | Function takes in the total area of a room, a pay rate of a designer, and average rating of said designer. The function then uses the three arguments to generate a total cost which is return to the calling routine. |
| **Procedures** | |
| proc\_retire\_param  (in\_designer\_id designers.designer\_id%TYPE, in\_retire\_age NUMBER) | Procedure takes in a designers id and the age at which they plan to retire, it then stores the designers date of birth using a query. the  dob and age they plan to retire is send to a function which returns the number of months they have left until retirement. |
| proc\_higher\_rate  (in\_rating testimonials.rating%Type) | Procedure takes in a rating and creates a cursor which queries all ratings in testimonials which are greater than the input rating. The procedure then opens the cursors and checks if a row was found, displaying  and appropriate message. If a row was found it loops through the results displaying the relevant information from the results found. |
| proc\_booking\_cost  (in\_total\_area NUMBER, in\_booking\_id bookings.booking\_id%TYPE) | Procedure takes in a total area and booking id. The booking id is then used in a cursors to find all customer rooms under the same booking, each customer rooms area is then added to the total area of the booking. It is also used in two queries that extracted the designers pay rate and average rating via joins and the AVG function into two variables. The total area, pay rate, and average rating are then passed to a function which returns a total cost which is used to update the booking associated with the booking id with a new cost. |
| **Triggers** | |
| trig\_designer | Trigger runs before insert, update or delete on the designers table taking appropriate action dependant on the action that triggered the trigger. If inserting it displays the name of who is being inserts. If updating it sends the dob of the record being updated to a function which returns the remaining months until retirements base on a hardcoded age of 67. If deleting it displays the name of who is being deleted. |
| trig\_booking\_cost | Trigger runs when a customer room is inserted. It takes the width and length and works out the total area and passes it into the procedure to update the bookings cost. |
| **Queries** | |
| Highest booking cost using functions. | |
| Customer rooms in which a designer is located. | |
| Find designers and their average ratings. | |
| List designer’s bookings and whether they are completed or not. | |
| Designer’s information and social media. | |
| Designer’s information and address. | |
| Customers address. | |
| Customer emergency contact details. | |
| Customers who have customer rooms and bookings. | |

# Proposed Automation Strategy

As the part of the database build some of the processes were automated to ease the process of the data query and format outputs. As for the complete database build SQL PROCEDURES, FUNCTIONS and TRIGGERS were implemented.

In the below paragraph all procedures, triggers and functions used in the database build will be introduced and briefly explained. Purpose is to comply with the aims and objectives which were set for the successful completion of the database build.

# Evidence of Additional Research

Some of the parts of the project build required additional research which helped to improve database functionality and comply with the up to date conventions.

One of the most informal and reader friendly documents was - **“Oracle SQL & PL/SQL Optimization for Developers Documentation Release 2.1.1”** written by: Ian Hellstrom released: Dec 29, 2017.

Document introducing starting level developer with the best practices and conventions from the very beginning and then going in depth to the structure of the complex queries, procedures etc. Documentation is easy to read and understand due to its nature explaining everything what's happening behind the scenes while running the code.

# Screenshots

## PROCEDURES

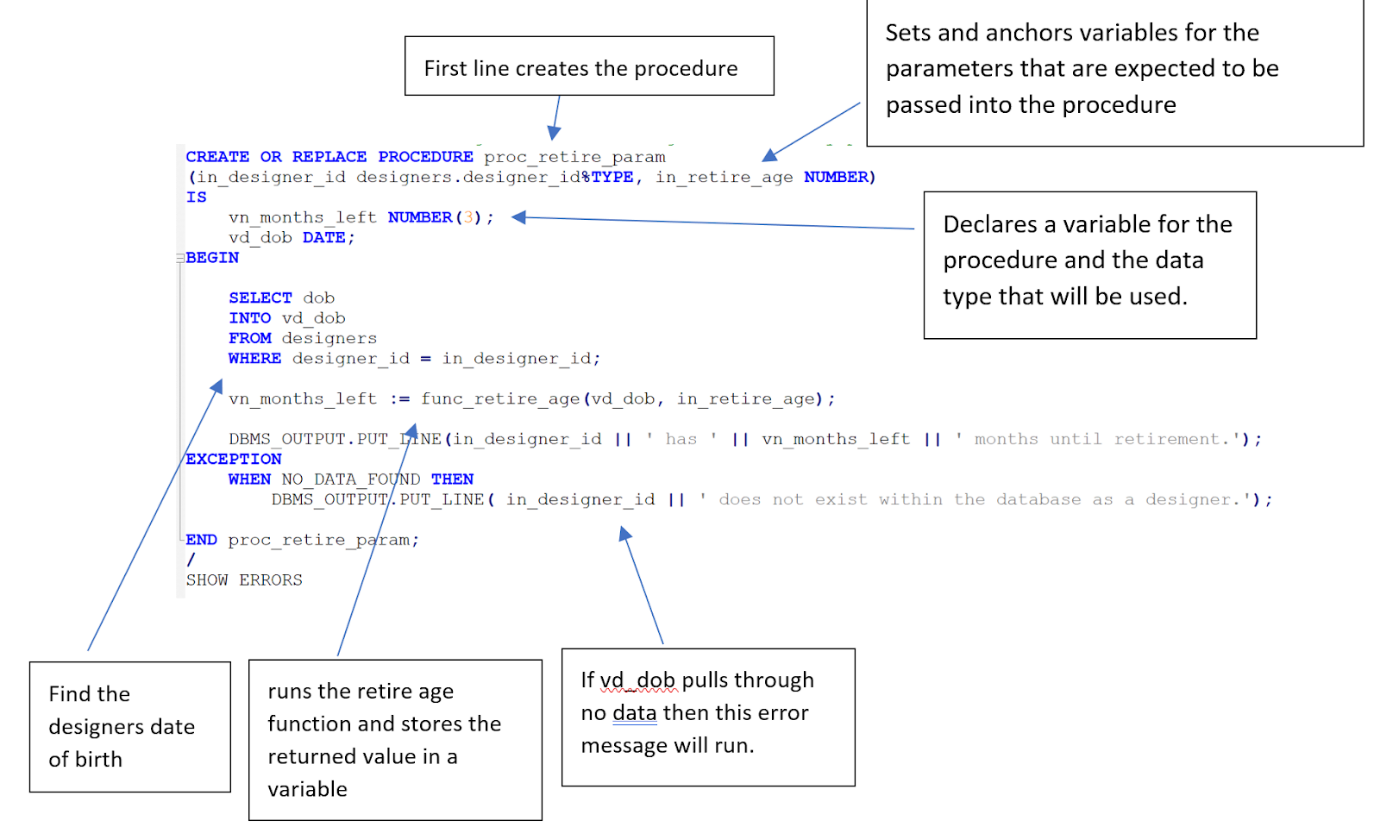
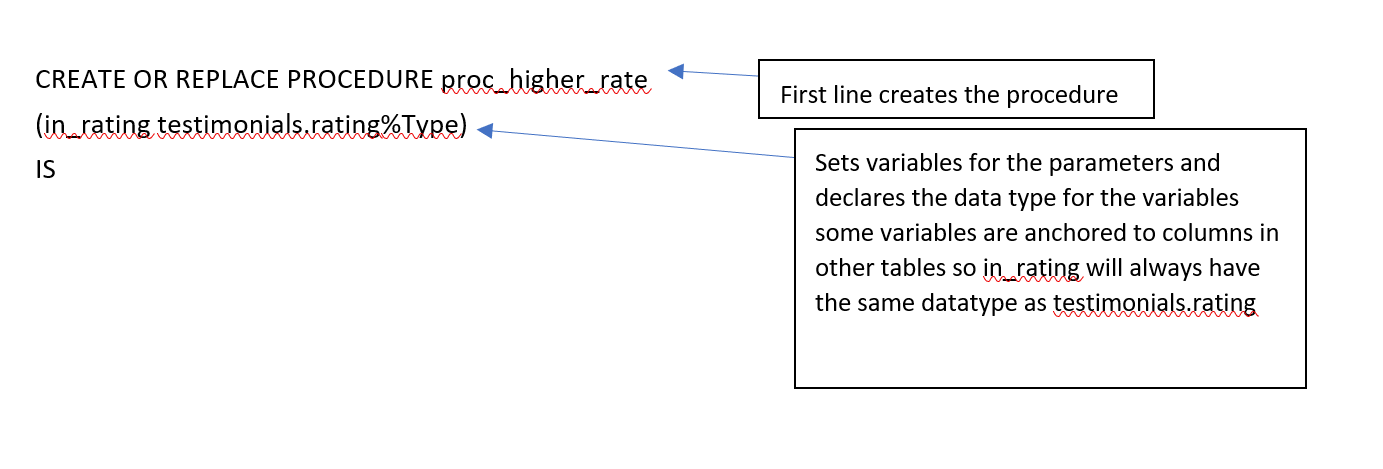
Table below will show procedure which is implemented in the database and brief description.

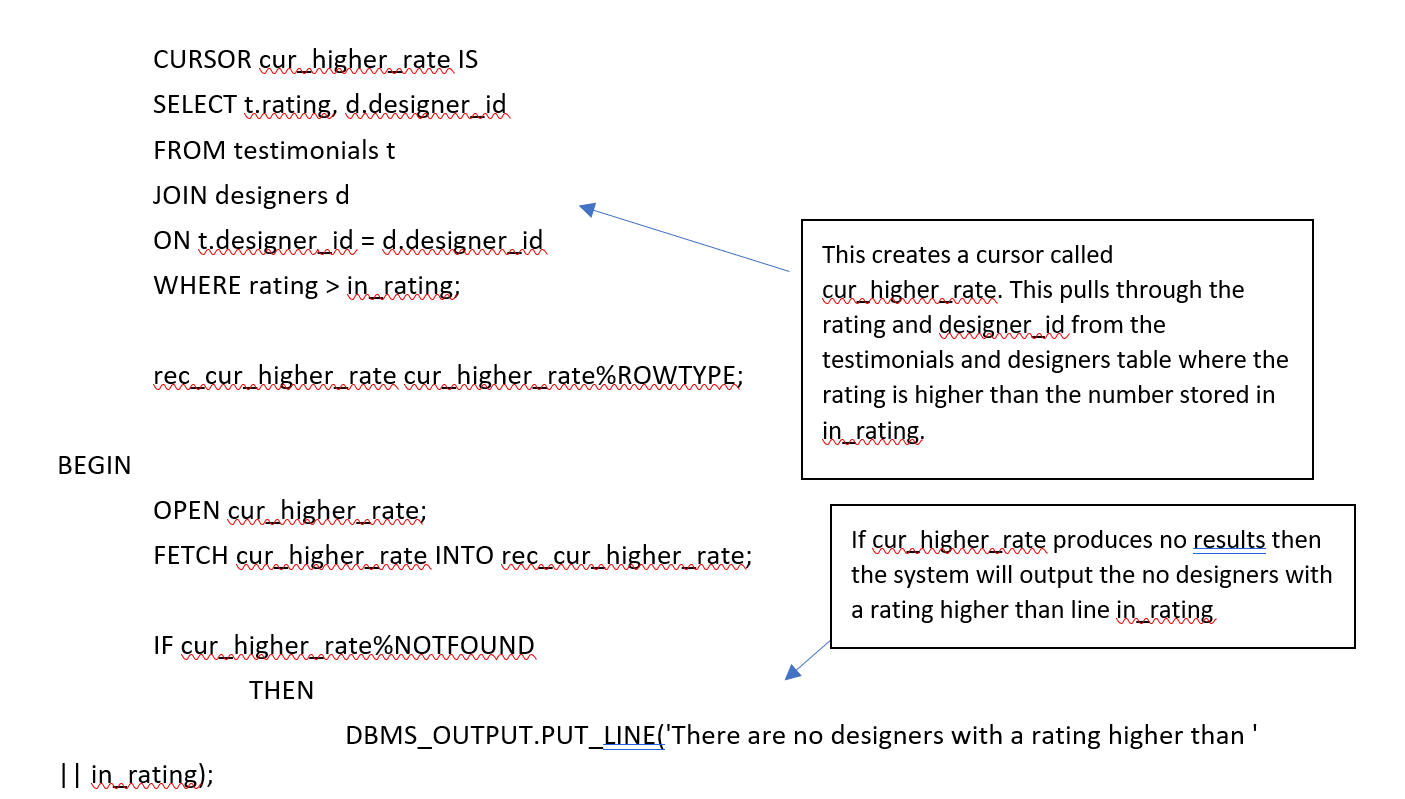
|  |  |  |
| --- | --- | --- |
| **ID.** | **Name** | **description** |
| **1** | proc\_retire\_param | Procedure takes in a designer’s id and the age at which they plan to retire, it then stores the designer’s date of birth using a query. The dob and age they plan to retire is send to a function which returns the number of months they have left until retirement. |
| **2.** | proc\_higher\_rate | Procedure takes in a rating and creates a cursor which queries all  ratings in testimonials which are greater than the input rating. The  procedure then opens the cursors and checks if a row was found, displaying and appropriate message. If a row was found it loops through the results displaying the relevant information from the results found. |
| **3.** | proc\_booking\_cost | Procedure takes in a total area and booking id. The booking id is then used in two queries that extracted the designers pay rate and average rating via joins and the AVG function into two variables. The total area, pay rate, and average rating are then passed to a function which returns a total cost which is used to update the booking associated with the booking id with a new cost |

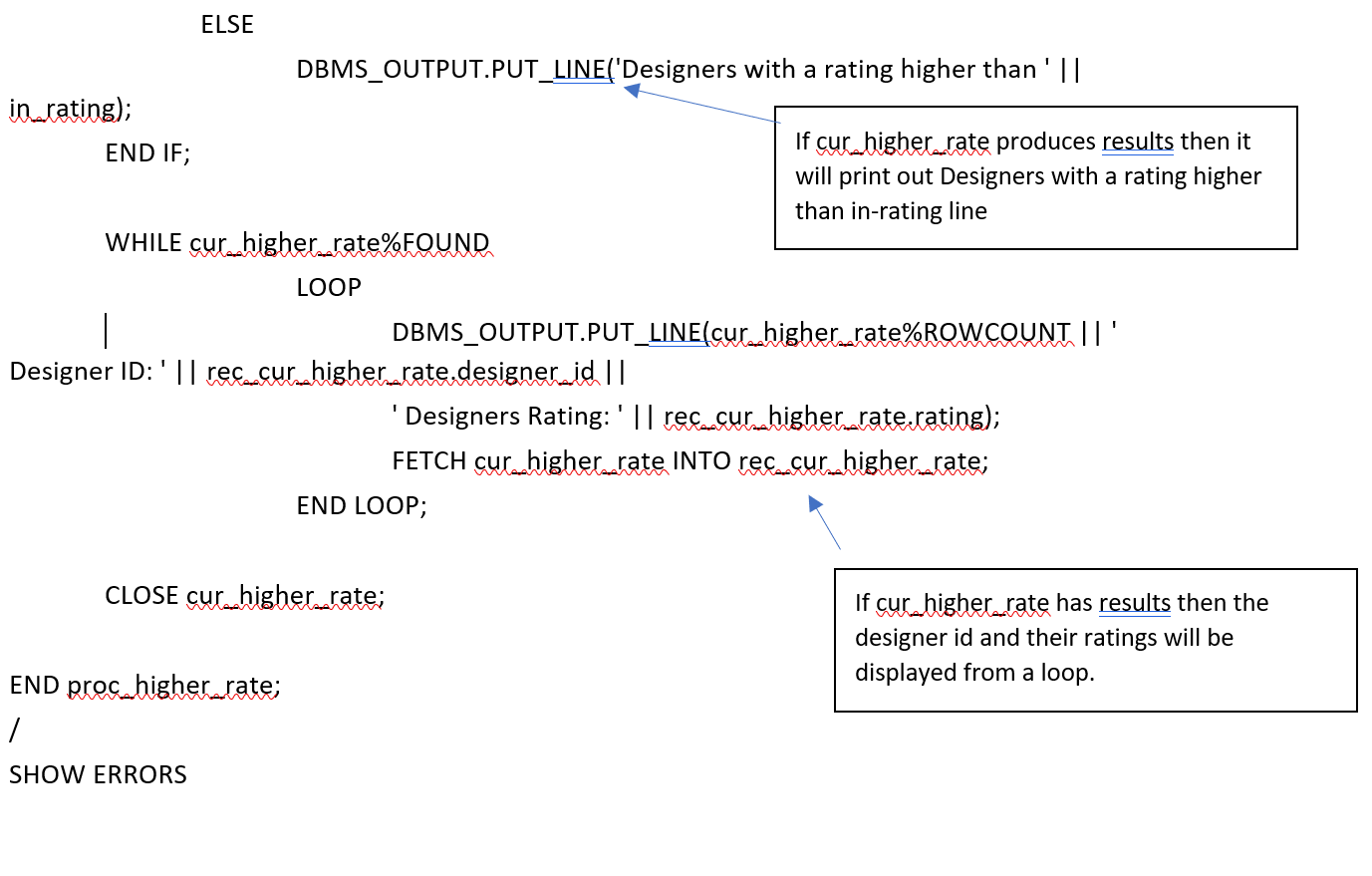
With the help of the procedures we can easy query multiple parameters and use functions to ease data retrieval from the database.

### Procedures Functionality:

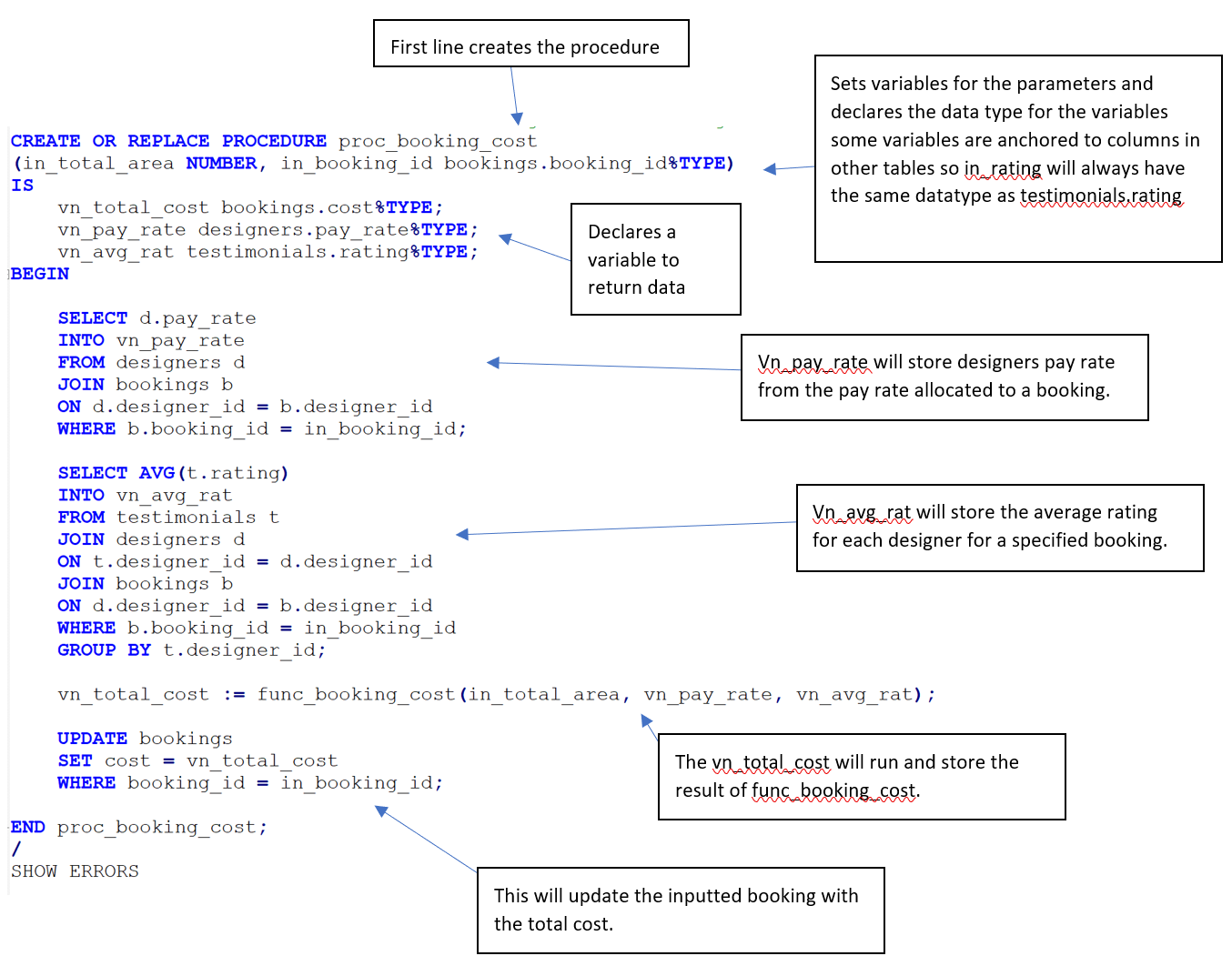
1. proc\_retire\_param

  
  
2. proc\_higher\_rate  






3. proc\_booking\_cost



## FUNCTIONS

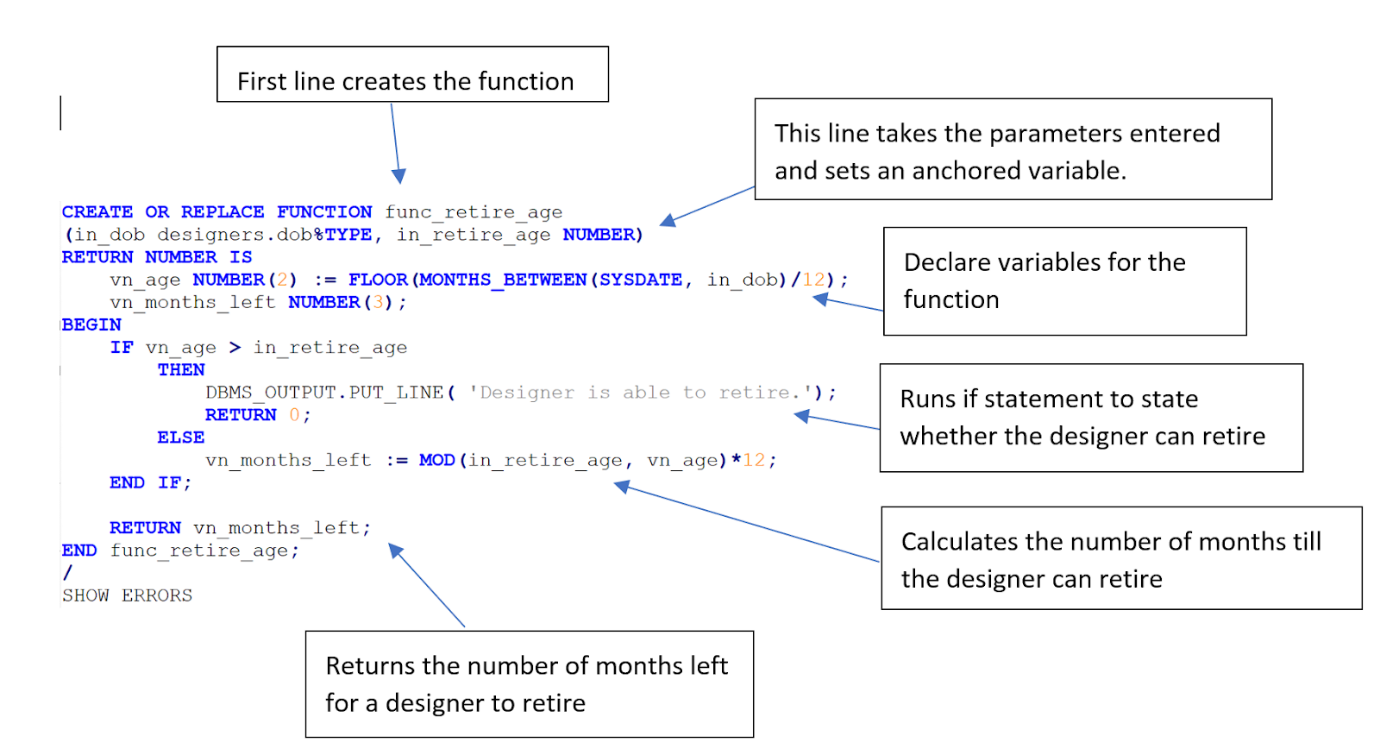
Table below will contain named functions and descriptions on what the functions are doing.

|  |  |  |
| --- | --- | --- |
| **ID.** | **Name** | **description** |
| **1.** | func\_retire\_age | Function determines the months left until retirement based upon the dob and retirement age passed in its parameters. |
| **2.** | func\_booking\_cost | Function takes in the total area of a room, a pay rate of a designer, and average rating of said designer. The function then uses the three arguements to generate a total cost which is return to the calling routine. |
| **3.** | func\_total\_area | Function calculates and returns the total area based upon two parameters passed if which are assumed to be measurements. |

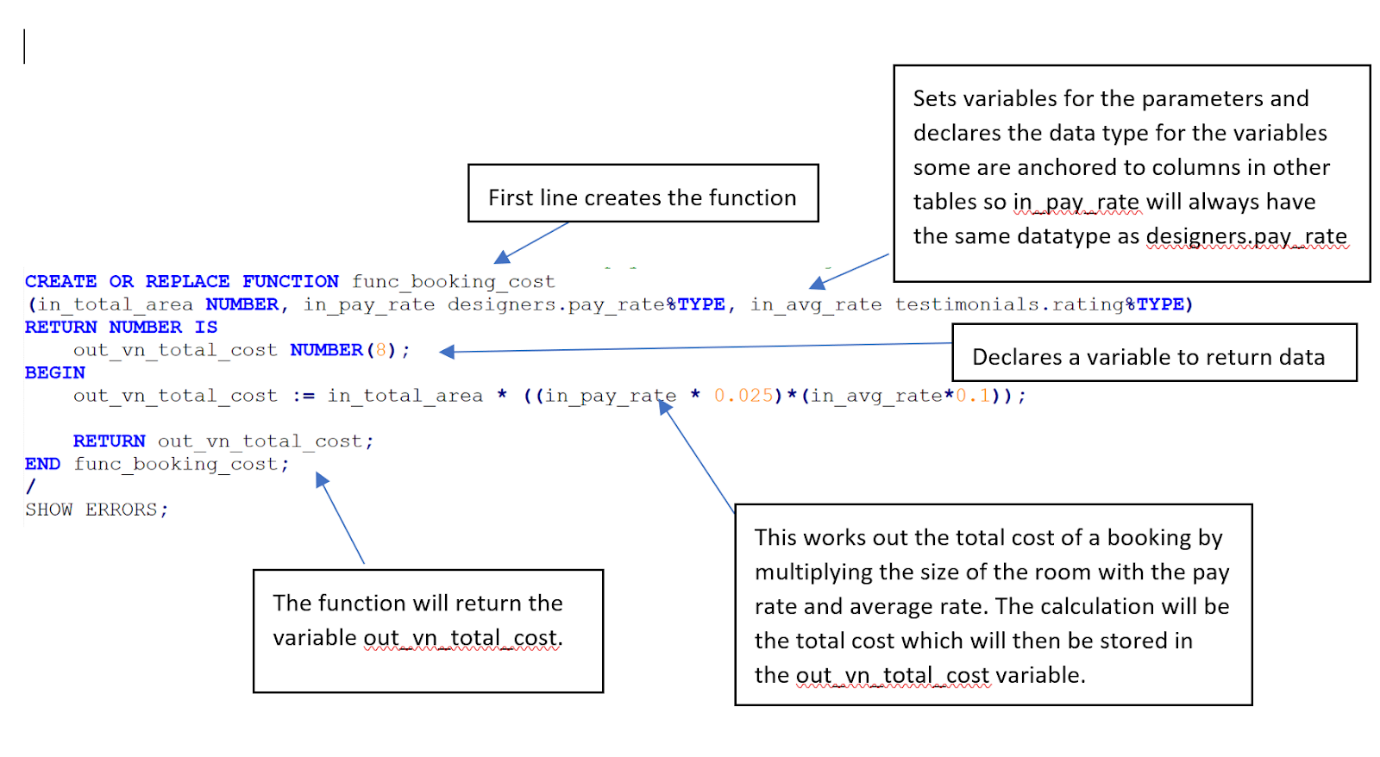
All of the functions explained above is designed to cover 3 different database areas. With the help of these functions it easier and faster to do calculations for the provided criteria.

### **Functions Functionality:**

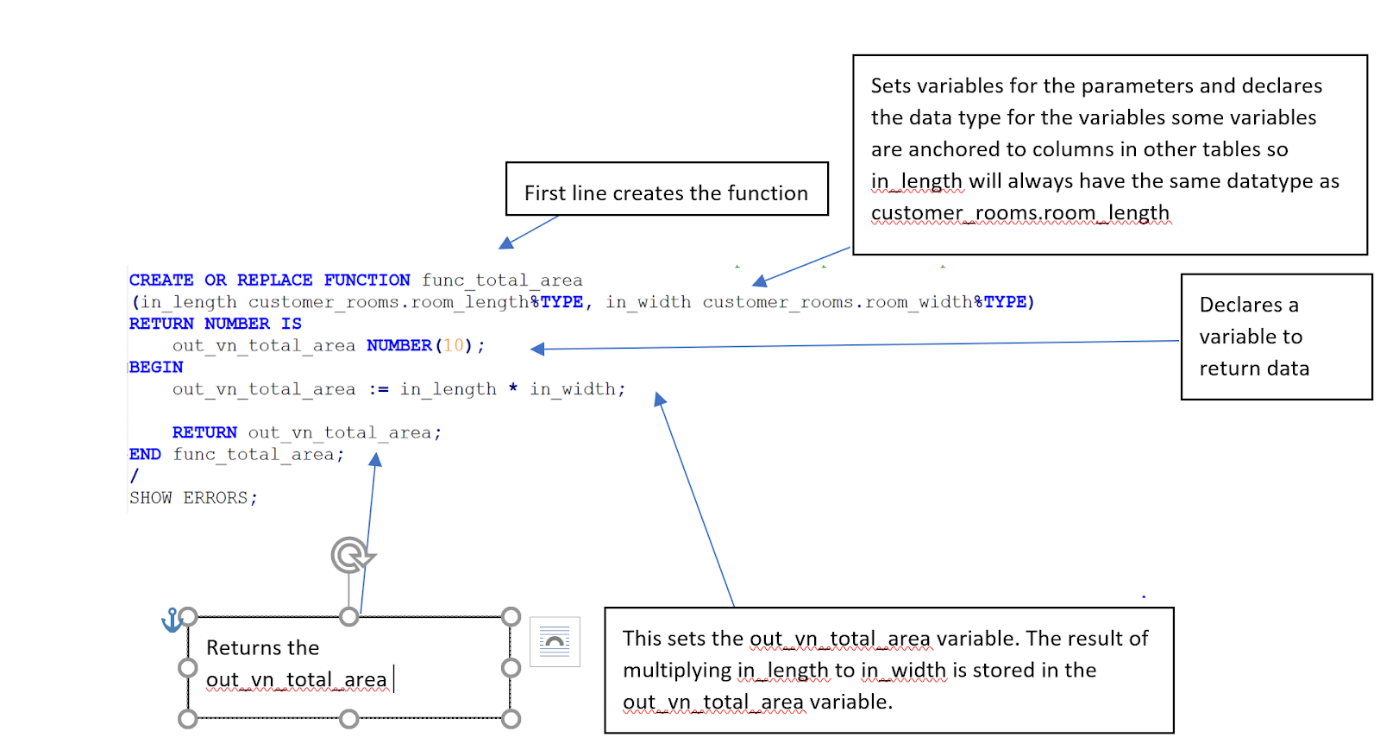
1. func\_retire\_age:



2. func\_booking\_Cost



.3. func\_total\_area



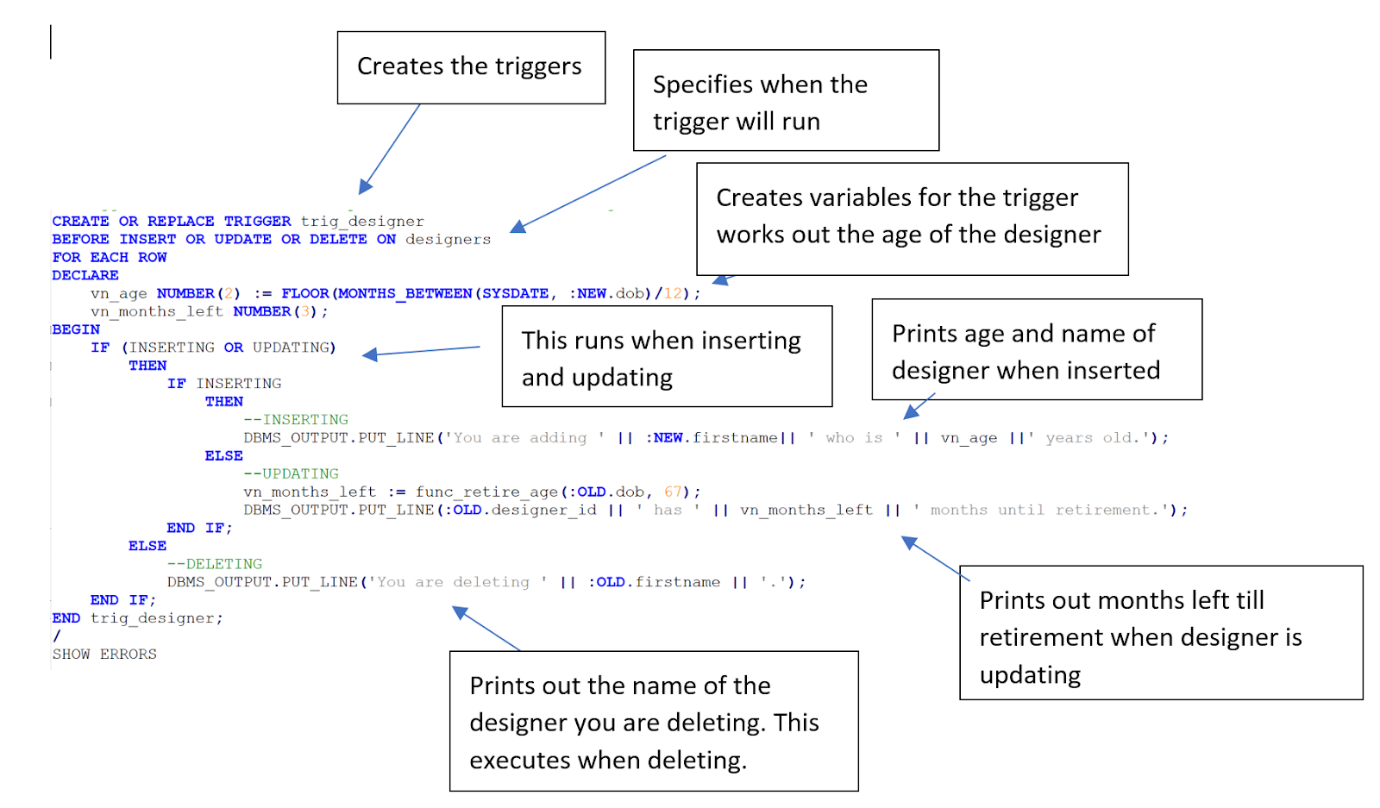
## TRIGGERS

Table below will identify and describe triggers which are going to run if certain parameter is met.

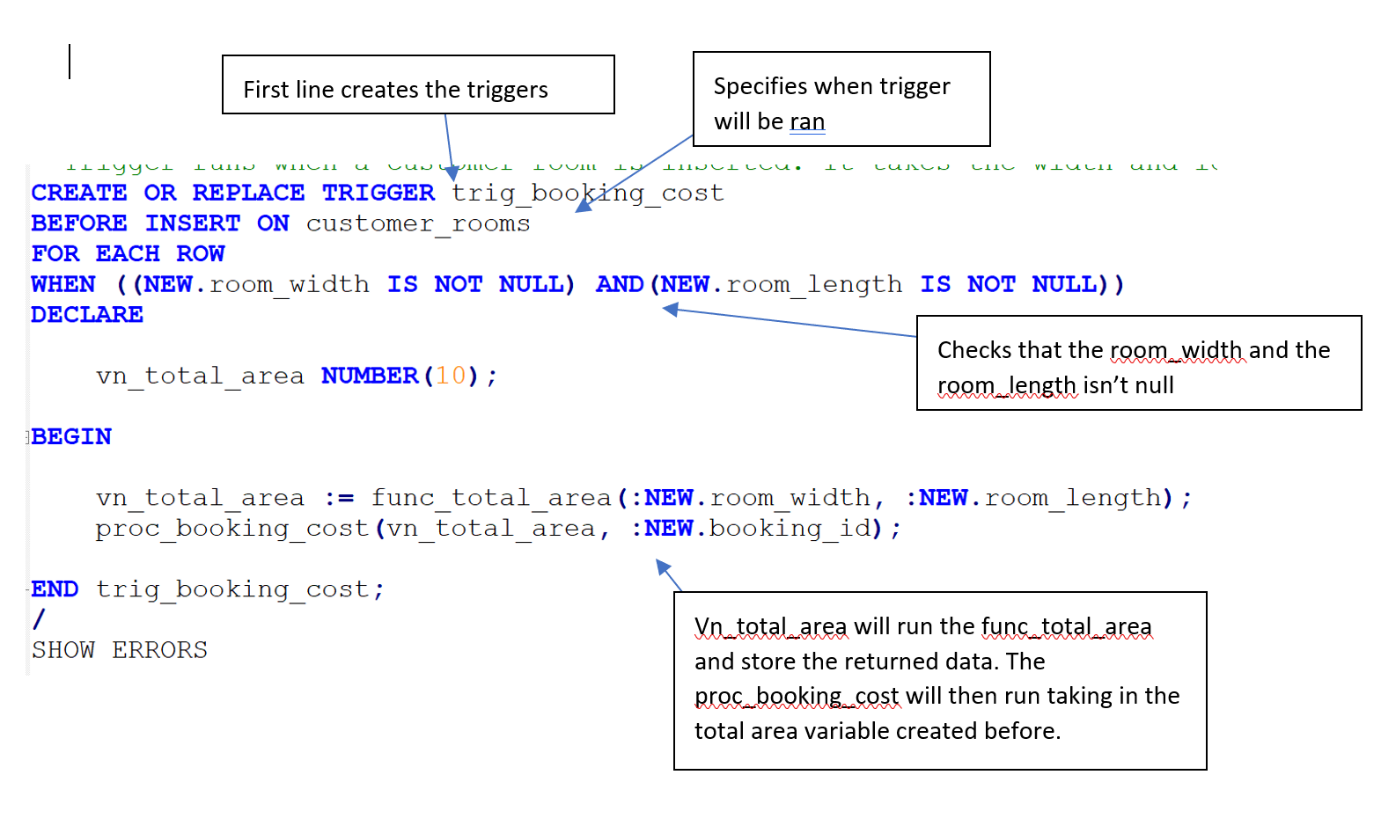
|  |  |  |
| --- | --- | --- |
| **ID.** | **Name** | **description** |
| **1.** | trig\_designer | Trigger runs before insert, update or delete on the designers table  taking appropriate action dependant on the action that triggered the trigger. If inserting it displays the name of who is being inserts. If updating it sends the dob of the record being updated to a function which returns the remaining months until retirement’s base on a hardcoded age of 67. If deleting it displays the name of who is being deleted. |
| **2.** | trig\_booking\_cost | Trigger runs when a customer room is inserted. It takes the width and length and works out the total area and passes it into the procedure to update the bookings cost |

### Triggers Functionality:

1. trig\_designer



2. trig\_booking\_cost



# Test Plan

Testing is fundamental in establishing a program works as intended, as such a test plan was constructed to ensure the queries, functions, procedures, and triggers created in task 1 work as intended and produce accurate/expected results.

The test plan works by assigning ids to individual tests, providing a test description which is the test itself (the query being tested or action required to trigger the action within the system), details the expected results of the action taking place, provides an actual result which either confirms the expected result was correct or details what was output instead, and finally action column details the actions taken to fix the error which caused the result to be different than the expected results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test Description | Expected Results | Actual Results | Action |
| 1 | SELECT booking\_id,  CEIL(cost), FLOOR(cost),  ROUND(cost),  TRUNC(cost), cost  FROM bookings  WHERE cost = (  SELECT MAX(cost)  FROM bookings  ); | 2880000 | ✓ | - |
| 2 | SELECT cr.customer\_id,  cr.booking\_id,  cr.address.street,  cr.address.city,  cr.address.country  FROM customer\_rooms  cr  WHERE cr.address.city  IN (  SELECT d.address.city  FROM designers d  WHERE  d.address.country = 'UK'  ); | 20000003   40000003  KETTERING ROAD  NORTHAMPTON UK   20000002   40000003  KETTERING ROAD  NORTHAMPTON UK  20000001   40000001  MATCHLESS CLOSE  NORTHAMPTON UK | ✓ | - |
| 3 | SELECT d.designer\_id,  AVG(t.rating)  FROM designers d  LEFT JOIN  testimonials t  ON d.designer\_id =  t.designer\_id  GROUP BY  d.designer\_id; | 10000003     6    10000002     3    10000000    10000001     8    10000004 | ✓ |  |
| 4 | SELECT d.designer\_id,  d.firstname, d.lastname,  b.booking\_id,  b.completed  FROM designers d  LEFT JOIN bookings b  ON b.designer\_id =  d.designer\_id; | 6 rows selected. | ✓ |  |
| 5 | SELECT d.designer\_id,  d.firstname, d.lastname,  s.contact  FROM designers d,  TABLE(d.social\_media) s; | 9 rows selected. | ✓ | - |
| 6 | SELECT d.designer\_id,  d.firstname, d.lastname,  d.address.house\_number,  d.address.street,  d.address.city,  d.address.postcode  FROM designers d  ORDER BY designer\_id; | shows the designers  name, id and  address details. | ✓ | - |
| 7 | SELECT customer\_id,  DEREF(address)  FROM customer\_rooms  WHERE customer\_id IN (  SELECT customer\_id  FROM customers  WHERE  customer\_id = '20000001'  ); | 20000001  ADDRESS\_TYPE  ('2A', 'MATCHLESS CLOSE',  'NORTHAMPTON', 'UK',  'NN5 6YE') | ✓ | - |
| 8 | SELECT c.customer\_id,  c.firstname, c.lastname,  co.contact\_type,  co.contact\_info  FROM customers c,  TABLE(c.contact) co  WHERE co.contact\_type =  'EMERGENCY'; | 5 rows selected. | ✓ | - |
| 9 | SELECT customer\_id FROM  customers  INTERSECT  SELECT customer\_id FROM  customer\_rooms  MINUS  SELECT booking\_id FROM  customer\_rooms; | 20000001    20000002    20000003    20000004 | ✓ | - |
| 10 | **Test proc\_retire\_param**  EXEC proc\_retire\_param  (10000000, 67); | 300 months until  retirement. | ✓ | - |
| 11 | **Test proc\_retire\_param**  EXEC proc\_retire\_param  (10000000, 60); | 216 months until  retirement. | ✓ | - |
| 12 | **Test proc\_retire\_param**  EXEC proc\_retire\_param  (40000000, 57); | 40000000 does not exist within the database as a designer. | ✓ | - |
| 13 | **Test proc\_retire\_param**  EXEC proc\_retire\_param  (10000000, 0); | Designer is able to retire. | ✓ | - |
| 14 | **Test proc\_higher\_rate**  EXEC proc\_higher\_rate(5); | Designers with a rating  higher than 5  1 Designer ID: 10000001  Designers Rating: 10  2 Designer ID: 10000003  Designers Rating: 6 | Designers with a rating  higher than 5  1 Designer ID: 10000001  Designers Rating: 7  2 Designer ID: 10000001  Designers Rating: 7  3 Designer ID: 10000001  Designers Rating: 10  4 Designer ID: 10000003  Designers Rating: 6 | Need to account for cartesian product by limiting results to singular designer and then using a function such as average or max to give a rating value. |
| 15 | **Test proc\_higher\_rate**  proc\_higher\_rate(0); | Designers with a rating  higher than 5  1 Designer ID: 10000001  Designers Rating: 10  2 Designer ID: 10000002  Designers Rating: 3  3 Designer ID: 10000003  Designers Rating: 6 | Designers with a rating  higher than 5  1 Designer ID: 10000001  Designers Rating: 7  2 Designer ID: 10000002  Designers Rating: 3  3 Designer ID: 10000001  Designers Rating: 7  4 Designer ID: 10000001  Designers Rating: 10  5 Designer ID: 10000003  Designers Rating: 6 | Need to account for cartesian product by limiting results to singular designer and then using a function such as average or max to give a rating value. |
| 16 | **Test proc\_higher\_rate**  proc\_higher\_rate(10); | There are no designers  with a rating higher than 10 | ✓ | - |
| 17 | **Test trig\_designer**  INSERT INTO designers (designer\_id, firstname, lastname, dob)  VALUES (designers\_seq.NEXTVAL, 'JOSH', 'LAND', '26-JAN-1995'); | You are adding JOSH who is 23 years old. | ✓ | - |
| 18 | **Test trig\_designer**  UPDATE designers SET lastname = 'CHALLAND'  WHERE designer\_id = 10000005; | 252 months until retirement. | ✓ | - |
| 19 | **Test trig\_designer**  DELETE FROM designers WHERE designer\_id = 10000005; | You are deleting JOSH. | ✓ | - |
| 20 | **Test trig\_booking\_cost**  INSERT INTO testimonials (testimonial\_id, designer\_id, rating)  VALUES (testimonials\_seq.NEXTVAL, 10000000, 5);  INSERT INTO customer\_rooms (customer\_room\_id, customer\_id, booking\_id, room\_width, room\_length)  VALUES (customer\_rooms\_seq.NEXTVAL, 20000000, 40000000, 50, 50);  select booking\_id, cost from bookings where booking\_id = 40000000; | --BOOKING\_ID       COST  -------- ----------  --  40000000        281 | ✓ | - |

# References

Media.readthedocs.org. (2018). *Oracle SQL & PL/SQL Optimization for Developers Documentation*. [online] Available at: https://media.readthedocs.org/pdf/oracle/latest/oracle.pdf [Accessed 14 Jan. 2018].

# Appendix

--CSY2038 PR1 Assignment - CSY2038PR1

/\*

Group 1

Josh Challand - 14414449

Julius Prakelis - 16430034

Ryan Haskins - 16442673

Tony Eardley - 14417434

\*/

--Tony Login CSY2038\_296@student/CSY2038\_296

--Josh Login CSY2038\_249@student/CSY2038\_249

--Ryan Login CSY2038\_286@student/16442673

--Julius Login - CSY2038\_251@student/CSY2038\_251

--@K:\scripts\CSY2028PR1.sql

/\* Useful Commands

COLUMN object\_name FORMAT A20;

COLUMN object\_type FORMAT A20;

SELECT object\_name, object\_type FROM user\_objects;

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Create Types \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE OR REPLACE TYPE social\_media\_type AS OBJECT (

media\_name VARCHAR2(25),

contact VARCHAR2(50)

);

/

SHOW ERRORS

CREATE TYPE social\_media\_table\_type AS TABLE OF social\_media\_type;

/

SHOW ERRORS

CREATE OR REPLACE TYPE contact\_type AS OBJECT (

contact\_type VARCHAR2(15),

contact\_method VARCHAR2(15),

contact\_info VARCHAR2(50)

);

/

SHOW ERRORS;

CREATE TYPE contact\_varray\_type AS VARRAY(50) OF contact\_type;

/

SHOW ERRORS;

CREATE OR REPLACE TYPE address\_type AS OBJECT (

house\_number VARCHAR2(25),

street VARCHAR2(25),

city VARCHAR2(25),

country VARCHAR2(25),

postcode VARCHAR2(15)

);

/

SHOW ERRORS;

CREATE TABLE addresses OF address\_type;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Types \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN object\_name FORMAT A20;

SELECT object\_name FROM user\_objects WHERE object\_type = 'TYPE';

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Create Tables \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE TABLE designers (

designer\_id NUMBER(8),

firstname VARCHAR2(25) NOT NULL,

lastname VARCHAR2(25) NOT NULL,

address REF address\_type SCOPE IS addresses,

social\_media social\_media\_table\_type,

pay\_rate NUMBER(4,2) DEFAULT 7.50,

hire\_date DATE DEFAULT SYSDATE,

dob DATE NOT NULL

)

NESTED TABLE social\_media STORE AS nested\_social\_media\_table\_type;

CREATE TABLE bookings (

booking\_id NUMBER(8),

designer\_id NUMBER(8) NOT NULL,

cost NUMBER(10,2),

completed CHAR(1) NOT NULL,

order\_date DATE DEFAULT SYSDATE,

arrival\_date DATE

);

CREATE TABLE customers (

customer\_id NUMBER(8),

title VARCHAR2(10),

firstname VARCHAR2(25) NOT NULL,

lastname VARCHAR2(25) NOT NULL,

username VARCHAR2(25),

address address\_type,

contact contact\_varray\_type

);

CREATE TABLE customer\_rooms (

Customer\_room\_id NUMBER(8),

customer\_id NUMBER(8) NOT NULL,

booking\_id NUMBER(8) NOT NULL,

room\_width NUMBER(6) NOT NULL,

room\_length NUMBER(6) NOT NULL,

address REF address\_type SCOPE IS addresses

);

CREATE TABLE testimonials(

testimonial\_id NUMBER(8),

designer\_id NUMBER(10) NOT NULL,

review VARCHAR2(125),

rating NUMBER(2) NOT NULL,

date\_created DATE DEFAULT SYSDATE

);

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Tables \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN tname FORMAT A20;

SELECT tname FROM TAB;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Create Sequences \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE SEQUENCE designers\_seq START WITH 10000000 INCREMENT BY 1;

CREATE SEQUENCE bookings\_seq START WITH 40000000 INCREMENT BY 1;

CREATE SEQUENCE testimonials\_seq START WITH 30000000 INCREMENT BY 1;

CREATE SEQUENCE customers\_seq START WITH 20000000 INCREMENT BY 1;

CREATE SEQUENCE customer\_rooms\_seq START WITH 50000000 INCREMENT BY 1;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Sequences \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN sequence\_name FORMAT A20;

SELECT sequence\_name FROM user\_sequences;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Primary keys \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER TABLE designers

ADD CONSTRAINT pk\_designers

PRIMARY KEY (designer\_id);

ALTER TABLE bookings

ADD CONSTRAINT pk\_bookings

PRIMARY KEY (booking\_id);

ALTER TABLE customers

ADD CONSTRAINT pk\_customers

PRIMARY KEY (customer\_id);

ALTER TABLE customer\_rooms

ADD CONSTRAINT pk\_customer\_rooms

PRIMARY KEY (customer\_room\_id);

ALTER TABLE testimonials

ADD CONSTRAINT pk\_testimonials

PRIMARY KEY (testimonial\_id);

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Foreign keys \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER TABLE bookings

ADD CONSTRAINT fk\_b\_designers

FOREIGN KEY (designer\_id)

REFERENCES designers(designer\_id);

ALTER TABLE customer\_rooms

ADD CONSTRAINT fk\_customers

FOREIGN KEY (customer\_id)

REFERENCES customers(customer\_id);

ALTER TABLE customer\_rooms

ADD CONSTRAINT fk\_bookings

FOREIGN KEY (booking\_id)

REFERENCES bookings(booking\_id);

ALTER TABLE testimonials

ADD CONSTRAINT fk\_t\_designers

FOREIGN KEY (designer\_id)

REFERENCES designers (designer\_id);

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Other Constraints \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN object\_name FORMAT A20;

SELECT constraint\_name

FROM user\_constraints

WHERE constraint\_type = 'P' OR constraint\_type = 'R';

ALTER TABLE bookings

MODIFY (completed DEFAULT 'N');

ALTER TABLE bookings

ADD CONSTRAINT ck\_completed

CHECK (completed IN ('Y','N'));

ALTER TABLE testimonials

ADD CONSTRAINT ck\_rating

CHECK (rating < 11);

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Constraints \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN object\_name FORMAT A20;

SELECT constraint\_name

FROM user\_constraints

WHERE constraint\_type = 'P' OR constraint\_type = 'R';

SET SERVEROUTPUT ON;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Create Functions \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--Function determines the months left until retirement based upon the dob and retirement age passed in its parameters.

CREATE OR REPLACE FUNCTION func\_retire\_age

(in\_dob designers.dob%TYPE, in\_retire\_age NUMBER)

RETURN NUMBER IS

vn\_age NUMBER(2) := FLOOR(MONTHS\_BETWEEN(SYSDATE, in\_dob)/12);

vn\_months\_left NUMBER(3);

BEGIN

IF vn\_age > in\_retire\_age

THEN

DBMS\_OUTPUT.PUT\_LINE( 'Designer is able to retire.');

RETURN 0;

ELSE

vn\_months\_left := MOD(in\_retire\_age, vn\_age)\*12;

END IF;

RETURN vn\_months\_left;

END func\_retire\_age;

/

SHOW ERRORS

--Function takes in the total area of a room, a pay rate of a designer, and average rating of said designer. The function then uses the three arguements to generate a total cost which is return to the calling routine.

CREATE OR REPLACE FUNCTION func\_booking\_cost

(in\_total\_area NUMBER, in\_pay\_rate designers.pay\_rate%TYPE, in\_avg\_rate testimonials.rating%TYPE)

RETURN NUMBER IS

out\_vn\_total\_cost NUMBER(8);

BEGIN

out\_vn\_total\_cost := in\_total\_area \* ((in\_pay\_rate \* 0.025)\*(in\_avg\_rate\*0.1));

RETURN out\_vn\_total\_cost;

END func\_booking\_cost;

/

SHOW ERRORS;

--Function calculates and returns the total area based upon two parameters passed if which are assumed to be measurements.

CREATE OR REPLACE FUNCTION func\_total\_area

(in\_length customer\_rooms.room\_length%TYPE, in\_width customer\_rooms.room\_width%TYPE)

RETURN NUMBER IS

out\_vn\_total\_area NUMBER(10);

BEGIN

out\_vn\_total\_area := in\_length \* in\_width;

RETURN out\_vn\_total\_area;

END func\_total\_area;

/

SHOW ERRORS;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Create Procedures \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--Procedure takes in a designers id and the age at which they plan to retire, it then stores the designers date of birth using a query. the dob and age they plan to retire is send to a function which returns the number of months they have left until retirement.\*/

CREATE OR REPLACE PROCEDURE proc\_retire\_param

(in\_designer\_id designers.designer\_id%TYPE, in\_retire\_age NUMBER)

IS

vn\_months\_left NUMBER(3);

vd\_dob DATE;

BEGIN

SELECT dob

INTO vd\_dob

FROM designers

WHERE designer\_id = in\_designer\_id;

vn\_months\_left := func\_retire\_age(vd\_dob, in\_retire\_age);

DBMS\_OUTPUT.PUT\_LINE(in\_designer\_id || ' has ' || vn\_months\_left || ' months until retirement.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE( in\_designer\_id || ' does not exist within the database as a designer.');

END proc\_retire\_param;

/

SHOW ERRORS

--test proc\_retire\_param

--EXEC proc\_retire\_param(10000000, 67);

--300 months until retirement.

--Success

--EXEC proc\_retire\_param(10000000, 60);

--216 months until retirement.

--EXEC proc\_retire\_param(40000000, 57);

--40000000 does not exist within the database as a designer.

--EXEC proc\_retire\_param(10000000, 0);

--Designer is able to retire.

--10000000 has months until retirement.

--Procedure takes in a rating and creates a cursor which queries all ratings in testimonials which are greater than the input rating. The procedure then opens the cursors and checks if a row was found, displayingand appropriate message. If a row was found it loops through the results displaying the relevant information from the results found.

CREATE OR REPLACE PROCEDURE proc\_higher\_rate

(in\_rating testimonials.rating%Type)

IS

CURSOR cur\_higher\_rate IS

SELECT t.rating, d.designer\_id

FROM testimonials t

JOIN designers d

ON t.designer\_id = d.designer\_id

WHERE rating > in\_rating;

rec\_cur\_higher\_rate cur\_higher\_rate%ROWTYPE;

BEGIN

OPEN cur\_higher\_rate;

FETCH cur\_higher\_rate INTO rec\_cur\_higher\_rate;

IF cur\_higher\_rate%NOTFOUND

THEN

DBMS\_OUTPUT.PUT\_LINE('There are no designers with a rating higher than ' || in\_rating);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Designers with a rating higher than ' || in\_rating);

END IF;

WHILE cur\_higher\_rate%FOUND

LOOP

DBMS\_OUTPUT.PUT\_LINE(cur\_higher\_rate%ROWCOUNT || ' Designer ID: ' || rec\_cur\_higher\_rate.designer\_id ||

' Designers Rating: ' || rec\_cur\_higher\_rate.rating);

FETCH cur\_higher\_rate INTO rec\_cur\_higher\_rate;

END LOOP;

CLOSE cur\_higher\_rate;

END proc\_higher\_rate;

/

SHOW ERRORS

--test proc\_higher\_rate

--EXEC proc\_higher\_rate(5);

--Designers with a rating higher than 5

--1 Designer ID: 10000001 Designers Rating: 7

--2 Designer ID: 10000001 Designers Rating: 7

--3 Designer ID: 10000001 Designers Rating: 10

--4 Designer ID: 10000003 Designers Rating: 6

--Success

--SELECT designer\_id, rating FROM testimonials WHERE rating > 5;

-- 10000001 7

-- 10000001 7

-- 10000001 10

-- 10000003 6

--Success

--Procedure takes in a total area and booking id. The booking id is then used in two queries that extracted the designers pay rate and average rating via joins and the AVG function into two variables. The total area, pay rate, and average rating are then passed to a function which returns a total cost which is used to update the booking associated with the booking id with a new cost.

CREATE OR REPLACE PROCEDURE proc\_booking\_cost

(in\_total\_area NUMBER, in\_booking\_id bookings.booking\_id%TYPE)

IS

vn\_total\_cost bookings.cost%TYPE;

vn\_pay\_rate designers.pay\_rate%TYPE;

vn\_avg\_rat testimonials.rating%TYPE;

BEGIN

SELECT d.pay\_rate

INTO vn\_pay\_rate

FROM designers d

JOIN bookings b

ON d.designer\_id = b.designer\_id

WHERE b.booking\_id = in\_booking\_id;

SELECT AVG(t.rating)

INTO vn\_avg\_rat

FROM testimonials t

JOIN designers d

ON t.designer\_id = d.designer\_id

JOIN bookings b

ON d.designer\_id = b.designer\_id

WHERE b.booking\_id = in\_booking\_id

GROUP BY t.designer\_id;

vn\_total\_cost := func\_booking\_cost(in\_total\_area, vn\_pay\_rate, vn\_avg\_rat);

UPDATE bookings

SET cost = vn\_total\_cost

WHERE booking\_id = in\_booking\_id;

END proc\_booking\_cost;

/

SHOW ERRORS

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Create Triggers \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--Trigger runs before insert, update or delete on the designers table taking appropriate action dependant on the action that triggered the trigger.If inserting it displays the name of who is being inserts. If updating itsends the dob of the record being updated to a function which returns the remaining months until retirements base on a hardcoded age of 67. If deleteing it displays the name of who is being deleted.

CREATE OR REPLACE TRIGGER trig\_designer

BEFORE INSERT OR UPDATE OR DELETE ON designers

FOR EACH ROW

DECLARE

vn\_age NUMBER(2) := FLOOR(MONTHS\_BETWEEN(SYSDATE, :NEW.dob)/12);

vn\_months\_left NUMBER(3);

BEGIN

IF (INSERTING OR UPDATING)

THEN

IF INSERTING

THEN

--INSERTING

DBMS\_OUTPUT.PUT\_LINE('You are adding ' || :NEW.firstname|| ' who is ' || vn\_age ||' years old.');

ELSE

--UPDATING

vn\_months\_left := func\_retire\_age(:OLD.dob, 67);

DBMS\_OUTPUT.PUT\_LINE(:OLD.designer\_id || ' has ' || vn\_months\_left || ' months until retirement.');

END IF;

ELSE

--DELETING

DBMS\_OUTPUT.PUT\_LINE('You are deleting ' || :OLD.firstname || '.');

END IF;

END trig\_designer;

/

SHOW ERRORS

--test trig\_designer

--INSERT INTO designers (designer\_id, firstname, lastname, dob)

--VALUES (designers\_seq.NEXTVAL, 'JOSH', 'LAND', '26-JAN-1995');

--You are adding JOSH who is 23 years old. - Success

--UPDATE designers SET lastname = 'CHALLAND'

--WHERE designer\_id = 10000005;

--252 months until retirement. - Success

--DELETE FROM designers WHERE designer\_id = 10000005;

--You are deleting JOSH. - Success

--Trigger runs when a customer room is inserted. It takes the width and length and works out the total area and passes it into the procedure to update the bookings cost.

CREATE OR REPLACE TRIGGER trig\_booking\_cost

BEFORE INSERT ON customer\_rooms

FOR EACH ROW

WHEN ((NEW.room\_width IS NOT NULL) AND(NEW.room\_length IS NOT NULL))

DECLARE

vn\_total\_area NUMBER(10);

BEGIN

vn\_total\_area := func\_total\_area(:NEW.room\_width, :NEW.room\_length);

proc\_booking\_cost(vn\_total\_area, :NEW.booking\_id);

END trig\_booking\_cost;

/

SHOW ERRORS

--test trig\_booking\_cost

--original cost: 766.75 --select cost from bookings where booking\_id = 40000000;

--desinger\_id: 10000000 --select designer\_id from bookings where booking\_id = 40000000;

--pay\_rate: 9 --select pay\_rate from designers where designer\_id = 10000000;

--INSERT INTO testimonials (testimonial\_id, designer\_id, rating)

--VALUES (testimonials\_seq.NEXTVAL, 10000000, 5);

--avg(rating): 5 --select AVG(rating) from testimonials where designer\_id = 10000000;

--New cost: 281

--INSERT INTO customer\_rooms (customer\_room\_id, customer\_id, booking\_id, room\_width, room\_length)

--VALUES (customer\_rooms\_seq.NEXTVAL, 20000000, 40000000, 50, 50);

--select booking\_id, cost from bookings;

--BOOKING\_ID COST

---------- ----------

-- 40000000 281

--SUCCESS

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Insert Addresses \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INSERT INTO addresses

VALUES ('2A','MATCHLESS CLOSE', 'NORTHAMPTON', 'UK', 'NN5 6YE');

INSERT INTO addresses

VALUES ('22', 'CARLTON ROAD', 'TOWCESTER', 'UK', 'TT2 7DQ');

INSERT INTO addresses

VALUES ('56', 'BILLING ROAD', 'NORTHAMPTON', 'UK', 'NN7 9PO');

INSERT INTO addresses

VALUES ('62', 'HENSPORT STREET', 'OUNDLE', 'UK', 'OU23 1MG');

INSERT INTO addresses

VALUES ('34', 'GISMOUTH STREET', 'NORTHAMPTON', 'UK', 'NG3 6YE');

INSERT INTO addresses

VALUES ('12', 'HUMMERSBY ROAD', 'NORTHAMPTON', 'UK', 'DV6 2OE');

INSERT INTO addresses

VALUES ('6Q', 'BRILTON ROAD', 'BRACKLEY', 'UK', 'BR10 5DE');

INSERT INTO addresses

VALUES ('6', 'KETTERING ROAD', 'NORTHAMPTON', 'UK', 'NH7 3GG');

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Addresses Inserts \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN street FORMAT A20;

COLUMN city FORMAT A20;

SELECT street, city FROM addresses;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Insert Designers \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INSERT INTO designers

VALUES(

designers\_seq.NEXTVAL,

'AKPAR',

'SALATIAN',

(SELECT REF(a)

FROM addresses a

WHERE street = 'CARLTON ROAD'),

social\_media\_table\_type(

social\_media\_type('AKPARSALATAIN69', 'TWIITER.COM/APKARS69'),

social\_media\_type('AKPARSALATAIN69', 'FACEBOOK.COM/APKARS69'),

social\_media\_type('AKPARSALATAIN69', 'YOUTUBE.COM/APKARS69')),

9.00,

'12-JAN-2004',

'21-FEB-1976'

);

INSERT INTO designers

VALUES (

designers\_seq.NEXTVAL,

'STEVE',

'PERRISON',

(SELECT REF (a)

FROM addresses a

WHERE street = 'KETTERING ROAD'),

social\_media\_table\_type(

social\_media\_type('PERRISON2', 'TWITTER.COM/PERRISON2'),

social\_media\_type('PERRISON2', 'FACEBOOK.COM/PERRISON2')),

12.00,

'17-JAN-2008',

'28-MAY-1998'

);

INSERT INTO designers

VALUES (

designers\_seq.NEXTVAL,

'MYLES',

'GOODE-FOUCHER',

(SELECT REF (a)

FROM addresses a

WHERE street = 'HUMMERSBY ROAD'),

social\_media\_table\_type(

social\_media\_type('MYLESFOUCHER', 'TWITTER.COM/MYLESFOUCHER'),

social\_media\_type('MYLESFOUCHER', 'FACEBOOK.COM/MYLESFOUCHER')),

8.00,

'17-JAN-2005',

'24-NOV-1988'

);

INSERT INTO designers

VALUES (

designers\_seq.NEXTVAL,

'ELIANE',

'PATTERSON',

(SELECT REF (a)

FROM addresses a

WHERE street = 'GISMOUTH STREET'),

social\_media\_table\_type(

social\_media\_type('ELIANEPATTERSON', 'FACEBOOK.COM/ELIANPATTERSON')),

7.50,

'10-AUG-2010',

'14-JAN-1989'

);

INSERT INTO designers

VALUES (

designers\_seq.NEXTVAL,

'BARRY',

'GILMARTIN',

(SELECT REF(a)

FROM addresses a

WHERE street='HENSPORT STREET'),

social\_media\_table\_type(

social\_media\_type('GILMARTIN', 'FACEBOOK.COM/BARRYGILMARTIN')),

20.50,

'20-FEB-2015',

'12-JAN-1977'

);

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Designers Inserts \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN firstname FORMAT A20;

SELECT designer\_id, firstname FROM designers;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Insert Customers \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INSERT INTO customers

VALUES (customers\_seq.NEXTVAL, 'MRS', 'ANDY', 'STAR', 'ONEANDONLY',

address\_type('1', 'MAIN ROAD','NORTHAMPTON', 'UK', 'NN9 5YU'),

contact\_varray\_type(

contact\_type('EMERGENCY', 'MOBILE PHONE', '07458921621'),

contact\_type('HOME', 'HOME PHONE', '01605216874'),

contact\_type('EMAIL', 'EMAIL', 'STARC@GMAIL.COM'))

);

INSERT INTO customers

VALUES (customers\_seq.NEXTVAL , 'MR', 'BEN', 'WALKER', 'WALKINGAWAY',

address\_type('13', 'WRENBRY ROAD', 'NORTHAMPTON', 'UNITED KINGDOM', 'NN5 6UI'),

contact\_varray\_type(

contact\_type('EMERGENCY', 'MOBILE PHONE', '07755787555'),

contact\_type('HOME', 'HOME PHONE', '01605589876'),

contact\_type('EMAIL', 'EMAIL', 'BEN777@HOTMAIL.COM'))

);

INSERT INTO customers

VALUES (customers\_seq.NEXTVAL , 'MS', 'DARKEN', 'SIDEN', 'CHOICEX',

address\_type('1', 'RYELAND ROAD','OUNDLE', 'UK', 'OU3 8GH'),

contact\_varray\_type(

contact\_type('EMERGENCY', 'MOBILE PHONE', '07525512311'),

contact\_type('HOME', 'HOME PHONE', '01605216800'),

contact\_type('EMAIL', 'EMAIL', 'DARKENSIDEN@VERY.DARK.MAIL.COM'))

);

INSERT INTO customers

VALUES (customers\_seq.NEXTVAL , 'MR', 'BILL', 'GATES', 'MILLIONDOOR',

address\_type('300', 'BRIXTON ROAD', 'KETTERING', 'UNITED KINGDOM', 'BX3 99H'),

contact\_varray\_type(

contact\_type('EMERGENCY', 'MOBILE PHONE', '07511223344'),

contact\_type('HOME', 'HOME PHONE', '01405212345'),

contact\_type('EMAIL', 'EMAIL', 'GATESAREOPEN@MICROSOFT.COM'))

);

INSERT INTO customers

VALUES (customers\_seq.NEXTVAL , 'MS', 'RUBY', 'DIAMOND', 'GOLDEN',

address\_type('555', 'BUILDING STREET', 'WELLINGBOROUGH', 'UK', 'DN37 8777'),

contact\_varray\_type(

contact\_type('EMERGENCY', 'MOBILE PHONE', '07501773144'),

contact\_type('HOME', 'HOME PHONE', '01605210000'),

contact\_type('EMAIL', 'EMAIL', 'RUBYTHEDIAMOND@GMAIL.COM'))

);

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Customers Inserts \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN firstname FORMAT A20;

SELECT customer\_id, firstname FROM customers;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Insert Testimonials \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INSERT INTO testimonials

VALUES (testimonials\_seq.NEXTVAL, 10000001, 'Well completed job, very happy!', 7, '12-JAN-2017');

INSERT INTO testimonials

VALUES (testimonials\_seq.NEXTVAL, 10000002, 'Poor Job, unhappy...', 3, '06-MAR-2016');

INSERT INTO testimonials

VALUES (testimonials\_seq.NEXTVAL, 10000001, 'Satisfactory', 7, '19-MAY-2017');

INSERT INTO testimonials

VALUES (testimonials\_seq.NEXTVAL, 10000001, 'Perfect', 10, '27-SEP-2015');

INSERT INTO testimonials

VALUES (testimonials\_seq.NEXTVAL, 10000003, 'Good!', 6, '23-NOV-2017');

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Testimonials Inserts \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT testimonial\_id, designer\_id, rating FROM testimonials;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Inserts Bookings \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INSERT INTO bookings(booking\_id, designer\_id, cost, completed, order\_date, arrival\_date)

VALUES (bookings\_seq.NEXTVAL, 10000000, 766.75, 'N', '14-FEB-2016', '01-FEB-2017');

INSERT INTO bookings(booking\_id, designer\_id, cost, completed, order\_date, arrival\_date)

VALUES (bookings\_seq.NEXTVAL,10000001, 13222.50, 'N', '03-AUG-2017', '01-DEC-2017');

INSERT INTO bookings(booking\_id, designer\_id, cost, completed, order\_date, arrival\_date)

VALUES (bookings\_seq.NEXTVAL,10000002, 988.25, 'N', '17-NOV-2015', '08-JAN-2016');

INSERT INTO bookings(booking\_id, designer\_id, cost, completed, order\_date, arrival\_date)

VALUES (bookings\_seq.NEXTVAL,10000003, 20000.49, 'Y', '06-MAY-2014', '01-JUN-2015');

INSERT INTO bookings(booking\_id, designer\_id, cost, completed, order\_date, arrival\_date)

VALUES (bookings\_seq.NEXTVAL,10000002, 2999.99, 'N', '19-JAN-2017', '01-MAR-2018');

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Bookings Inserts \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT booking\_id, designer\_id, cost FROM bookings;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Inserts Customer\_Rooms \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INSERT INTO customer\_rooms

SELECT customer\_rooms\_seq.NEXTVAL,

20000001,

40000001,

4500,

2300,

REF(a)

FROM addresses a

WHERE street = 'MATCHLESS CLOSE';

INSERT INTO customer\_rooms

SELECT customer\_rooms\_seq.NEXTVAL,

20000002,

40000003,

1400,

2550,

REF(a)

FROM addresses a

WHERE street = 'KETTERING ROAD';

INSERT INTO customer\_rooms

SELECT customer\_rooms\_seq.NEXTVAL,

20000004,

40000002,

6000,

8000,

REF(a)

FROM addresses a

WHERE street = 'BRILTON ROAD';

INSERT INTO customer\_rooms

SELECT customer\_rooms\_seq.NEXTVAL,

20000003,

40000003,

3000,

5025,

REF(a)

FROM addresses a

WHERE street = 'KETTERING ROAD';

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Check Customer\_Rooms Inserts \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

COLUMN c.address.street FORMAT A20;

SELECT c.customer\_room\_id, c.customer\_id, c.booking\_id, c.address.street FROM customer\_rooms c;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Queries \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--Lists the booking with the highest cost with functions (CEIL, FLOOR, ROUND, TRUNC) - Formatted with headings

COLUMN booking\_id HEADING 'Booking'

COLUMN CEIL(cost) HEADING 'CEIL'

COLUMN FLOOR(cost) HEADING 'FLOOR'

COLUMN ROUND(cost) HEADING 'ROUND'

COLUMN TRUNC(cost) HEADING 'TRUNC'

SELECT booking\_id, CEIL(cost), FLOOR(cost), ROUND(cost), TRUNC(cost), cost

FROM bookings

WHERE cost = (

SELECT MAX(cost)

FROM bookings

);

--Test - Expected Results '2880000' to be the MAX cost

--SELECT MAX(cost)

--FROM bookings;

-- Result -- success:

-- 1 row returned with the max cost

--Lists customer rooms located in cities where designers are located within the UK

COLUMN customer\_id HEADING 'Customer ID'

COLUMN booking\_id HEADING 'Booking ID'

COLUMN address.street HEADING 'Street' FORMAT A25

COLUMN address.city HEADING 'City' FORMAT A15

COLUMN address.country HEADING 'Country' FORMAT A10

SELECT cr.customer\_id, cr.booking\_id, cr.address.street, cr.address.city, cr.address.country

FROM customer\_rooms cr

WHERE cr.address.city IN (

SELECT d.address.city

FROM designers d

WHERE d.address.country = 'UK'

);

--Test - Expected Result = 3 rows selected

--SELECT d.address.city

--FROM designers d;

--SELECT cr.address.city

--FROM customer\_rooms cr;

--Result -success:

-- List of the cities as expected is returned

--Pulls through designers id and show their ratings

COLUMN d.designer\_id HEADING 'Designer ID'

COLUMN AVG(t.rating) HEADING 'Rating'

SELECT d.designer\_id, AVG(t.rating)

FROM designers d

LEFT JOIN testimonials t

ON d.designer\_id = t.designer\_id

GROUP BY d.designer\_id;

--Test Expected result will be to pull through all the designers id's and their ratings

-- Results: List of designers returned with and without rating

--Will pull through all designers id first and last name and will show if they have any bookings

COLUMN booking\_id HEADING 'Booking ID'

COLUMN designer\_id HEADING 'Designer ID'

COLUMN firstname HEADING 'First Name' FORMAT A10

COLUMN lastname HEADING 'Last Name' FORMAT A20

SELECT d.designer\_id, d.firstname, d.lastname, b.booking\_id, b.completed

FROM designers d

LEFT JOIN bookings b

ON b.designer\_id = d.designer\_id;

--Expected result - mix of designers with and without bookings assigned to them.

--Results - success: 6 rows returned

--List of designers with and without bookings returned

--Pulls through designer id name and their social networking sites --

COLUMN designer\_id HEADING 'Designer ID'

COLUMN firstname HEADING 'First Name'

COLUMN lastname HEADING 'Last Name'

COLUMN contact FORMAT A30

SELECT d.designer\_id, d.firstname, d.lastname, s.contact

FROM designers d, TABLE(d.social\_media) s;

--Expected result - list all the different social networking url's for each designer

--Result - success:

--9 rows returned.

--Listing all social media contact details

--Pull through designers and their address --

COLUMN designer\_id HEADING 'Designer ID'

COLUMN firstname HEADING 'First Name' FORMAT A10

COLUMN lastname HEADING 'Last Name' FORMAT A10

COLUMN address.house\_number HEADING 'House No.' FORMAT A5

COLUMN address.street HEADING 'Street' FORMAT A20

COLUMN address.city HEADING 'City' FORMAT A10

COLUMN address.postcode HEADING 'Postcode' FORMAT A10

SELECT d.designer\_id, d.firstname, d.lastname, d.address.house\_number, d.address.street, d.address.city, d.address.postcode

FROM designers d

ORDER BY designer\_id;

--Expected results - show the designers name id and address details.

--Result - success:

--List of designer ID and all address details returned

--DREF Query getting the address and matching it to the customer

COLUMN customer\_id HEADING 'Customer ID'

COLUMN DEREF(address) HEADING 'Address'

SELECT customer\_id, DEREF(address)

FROM customer\_rooms

WHERE customer\_id IN (

SELECT customer\_id

FROM customers

WHERE customer\_id = '20000001'

);

--Expected result - query will display the customer address = 20000000

--Result - success:

--List of customer addresses with the matching ID returned

--Pulling through every customers emergency details using a varray query with dot notation

COLUMN customer\_id HEADING 'Customer ID'

COLUMN firstname HEADING 'Firstname'

COLUMN lastname HEADING 'Lastname'

COLUMN contact\_type HEADING 'Contact Type'

COLUMN contact\_info HEADING 'Contact Number'

SELECT c.customer\_id, c.firstname, c.lastname, co.contact\_type, co.contact\_info

FROM customers c, TABLE(c.contact) co

WHERE co.contact\_type = 'EMERGENCY';

--Expected result - list all customers and their emergency contact details

--Result - success:

--List of all customers and emergency contact details returned

--Shows all customer\_id's that have a customer\_room and a Booking

COLUMN customer\_id HEADING 'Customer ID'

SELECT customer\_id FROM customers

INTERSECT

SELECT customer\_id FROM customer\_rooms

MINUS

SELECT booking\_id FROM customer\_rooms;

--Expected result: will display customers that have a customer room and a booking allocated to them

--Result - success:

--List of customer ID`s returned

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Procedure Test(s) \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--test proc\_retire\_param

EXEC proc\_retire\_param(10000000, 67);

--300 months until retirement.

--Success

EXEC proc\_retire\_param(10000000, 60);

--216 months until retirement.

EXEC proc\_retire\_param(40000000, 57);

--40000000 does not exist within the database as a designer.

EXEC proc\_retire\_param(10000000, 0);

--Designer is able to retire.

--10000000 has months until retirement.

--test proc\_higher\_rate

EXEC proc\_higher\_rate(5);

--Designers with a rating higher than 5

--1 Designer ID: 10000001 Designers Rating: 7

--2 Designer ID: 10000001 Designers Rating: 7

--3 Designer ID: 10000001 Designers Rating: 10

--4 Designer ID: 10000003 Designers Rating: 6

--Success

SELECT designer\_id, rating FROM testimonials WHERE rating > 5;

-- 10000001 7

-- 10000001 7

-- 10000001 10

-- 10000003 6

--Success

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Trigger Test(s) \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--test trig\_designer

INSERT INTO designers (designer\_id, firstname, lastname, dob)

VALUES (designers\_seq.NEXTVAL, 'JOSH', 'LAND', '26-JAN-1995');

--You are adding JOSH who is 23 years old. - Success

UPDATE designers SET lastname = 'CHALLAND'

WHERE designer\_id = 10000005;

--252 months until retirement. - Success

DELETE FROM designers WHERE designer\_id = 10000005;

--You are deleting JOSH. - Success

--test trig\_booking\_cost

--booking\_id: 40000000

SELECT booking\_id FROM bookings WHERE booking\_id = 40000000;

--original cost: 766.75

SELECT cost FROM bookings WHERE booking\_id = 40000000;

--desinger\_id: 10000000

--SELECT designer\_id FROM bookings WHERE booking\_id = 40000000;

--pay\_rate: 9

--SELECT pay\_rate FROM designers WHERE designer\_id = 10000000;

INSERT INTO testimonials (testimonial\_id, designer\_id, rating)

VALUES (testimonials\_seq.NEXTVAL, 10000000, 5);

--avg(rating): 5 --select AVG(rating) from testimonials where designer\_id = 10000000;

--New cost: 281

INSERT INTO customer\_rooms (customer\_room\_id, customer\_id, booking\_id, room\_width, room\_length)

VALUES (customer\_rooms\_seq.NEXTVAL, 20000000, 40000000, 50, 50);

SELECT booking\_id, cost FROM bookings;

--BOOKING\_ID COST

---------- ----------

-- 40000000 281

--SUCCESS

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Triggers \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER trig\_designer;

DROP TRIGGER trig\_booking\_cost;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Procedures \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP PROCEDURE proc\_retire\_param;

DROP PROCEDURE proc\_higher\_rate;

DROP PROCEDURE proc\_booking\_cost;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Functions \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP FUNCTION func\_retire\_age;

DROP FUNCTION func\_booking\_cost;

DROP FUNCTION func\_total\_area;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Constraints \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER TABLE bookings

DROP CONSTRAINT ck\_completed;

ALTER TABLE testimonials

DROP CONSTRAINT ck\_rating;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Foreign Keys \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER TABLE customer\_rooms

DROP CONSTRAINT fk\_bookings;

ALTER TABLE customer\_rooms

DROP CONSTRAINT fk\_customers;

ALTER TABLE bookings

DROP CONSTRAINT fk\_b\_designers;

ALTER TABLE testimonials

DROP CONSTRAINT fk\_t\_designers;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Primary Keys \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER TABLE customer\_rooms

DROP CONSTRAINT pk\_customer\_rooms;

ALTER TABLE testimonials

DROP CONSTRAINT pk\_testimonials;

ALTER TABLE customers

DROP CONSTRAINT pk\_customers;

ALTER TABLE bookings

DROP CONSTRAINT pk\_bookings;

ALTER TABLE designers

DROP CONSTRAINT pk\_designers;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Tables \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TABLE customer\_rooms;

DROP TABLE bookings;

DROP TABLE testimonials;

DROP TABLE customers;

DROP TABLE designers;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Sequences \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP SEQUENCE designers\_seq;

DROP SEQUENCE bookings\_seq;

DROP SEQUENCE testimonials\_seq;

DROP SEQUENCE customers\_seq;

DROP SEQUENCE customer\_rooms\_seq;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Object Tables \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TABLE addresses;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Table/Varray Types \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TYPE social\_media\_table\_type;

DROP TYPE contact\_varray\_type;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Types \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TYPE social\_media\_type;

DROP TYPE contact\_type;

DROP TYPE address\_type;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Purge RECYCLEBIN \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PURGE RECYCLEBIN;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Drop Check \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT object\_name FROM user\_objects;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* EXTRAS FROM DEMO \*

\*(HIGHLIGHT BLOCK AND SINGLE LINE UNCOMMENT)\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- /\*###PLSQL\_2\_PROCEDURES###\*/

-- --SET SERVEROUTPUT ON;

-- CREATE OR REPLACE PROCEDURE proc\_add\_customer IS

-- vc\_customer\_name customers.firstname%TYPE := 'CUSTOMER';

-- BEGIN

-- INSERT INTO customers (customer\_id, firstname)

-- VALUES (customers\_seq.NEXTVAL, vc\_customer\_name);

-- END proc\_add\_customer;

-- /

-- SHOW ERRORS

-- /\*test proc\_add\_customer\*/

-- SELECT \* FROM customers;

-- -- 5 rows returned

-- EXEC proc\_add\_customer

-- SELECT \* FROM customers;

-- -- 6 rows returned

-- --Success

-- --DROP PROCEDURE proc\_add\_customer;

-- CREATE OR REPLACE PROCEDURE proc\_delete\_customer IS

-- vn\_customer\_id customers.customer\_id%TYPE;

-- BEGIN

-- SELECT customers\_seq.CURRVAL

-- INTO vn\_customer\_id

-- FROM DUAL;

-- DELETE FROM customers

-- WHERE customer\_id = vn\_customer\_id;

-- END proc\_delete\_customer;

-- /

-- SHOW ERRORS

-- /\*test proc\_delete\_customer\*/

-- SELECT \* FROM customers;

-- --6 rows returned

-- EXEC proc\_delete\_customer;

-- SELECT \* FROM customers;

-- --5 rows returned

-- --Success

-- --DROP PROCEDURE proc\_delete\_customer;

-- CREATE OR REPLACE PROCEDURE proc\_param\_del\_cu

-- (in\_customer\_id customers.customer\_id%TYPE) IS

-- BEGIN

-- DELETE FROM customers

-- WHERE customer\_id = in\_customer\_id;

-- END proc\_param\_del\_cu;

-- /

-- SHOW ERRORS

-- /\*test proc\_param\_del\_cu\*/

-- SELECT \* FROM customers;

-- --5 rows returned

-- EXEC proc\_param\_del\_cu(20000005);

-- SELECT \* FROM customers;

-- --4 rows returned

-- --Success

-- --DROP PROCEDURE proc\_param\_del\_cu;

-- --BONUS ACTIVTY

-- CREATE OR REPLACE PROCEDURE proc\_firstname (in\_firstname customers.firstname%TYPE) IS

-- vn\_length NUMBER(2);

-- vn\_counter NUMBER(2) := 1;

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('----FIRSTNAME----');

-- vn\_length := LENGTH(in\_firstname);

-- LOOP

-- EXIT WHEN vn\_counter = (vn\_length + 1);

-- DBMS\_OUTPUT.PUT\_LINE(SUBSTR(in\_firstname, vn\_counter, 1));

-- vn\_counter := vn\_counter + 1;

-- END LOOP;

-- END proc\_firstname;

-- /

-- SHOW ERRORS

-- /\*test proc\_firstname\*/

-- EXEC proc\_firstname('ARYA')

-- --Success

-- EXEC proc\_firstname('SILVER')

-- --Success

-- --DROP PROCEDURE proc\_firstname;

-- CREATE OR REPLACE PROCEDURE proc\_middlename (in\_middlename VARCHAR2) IS

-- vn\_length NUMBER(2);

-- vn\_counter NUMBER(2) := 1;

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('----MIDDLENAME----');

-- vn\_length := LENGTH(in\_middlename);

-- WHILE vn\_counter <= vn\_length LOOP

-- DBMS\_OUTPUT.PUT\_LINE(SUBSTR(in\_middlename, vn\_counter, 1));

-- vn\_counter := vn\_counter + 1;

-- END LOOP;

-- END proc\_middlename;

-- /

-- SHOW ERRORS

-- /\*test proc\_middlename\*/

-- EXEC proc\_middlename('MARY-ANGEL')

-- --Success

-- EXEC proc\_middlename('FLOWERS')

-- --Success

-- --DROP PROCEDURE proc\_middlename;

-- CREATE OR REPLACE PROCEDURE proc\_lastname (in\_lastname customers.lastname%TYPE) IS

-- vn\_length NUMBER(2);

-- vn\_counter NUMBER(2) := 1;

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('----LASTNAME----');

-- vn\_length := LENGTH(in\_lastname);

-- FOR vn\_counter IN 1 .. vn\_length LOOP

-- DBMS\_OUTPUT.PUT\_LINE(SUBSTR(in\_lastname, vn\_counter, 1));

-- END LOOP;

-- END proc\_lastname;

-- /

-- SHOW ERRORS

-- /\*test proc\_lastname\*/

-- EXEC proc\_lastname('MAULE')

-- --Success

-- EXEC proc\_lastname('EARDLEY')

-- --Success

-- --DROP PROCEDURE proc\_lastname;

-- CREATE OR REPLACE PROCEDURE proc\_fir\_mid\_las (in\_firstname customers.firstname%TYPE, in\_middlename VARCHAR2, in\_lastname customers.lastname%TYPE) IS

-- BEGIN

-- proc\_firstname(in\_firstname);

-- proc\_middlename(in\_middlename);

-- proc\_lastname(in\_lastname);

-- END proc\_fir\_mid\_las;

-- /

-- SHOW ERRORS

-- /\*test proc\_fir\_mid\_las\*/

-- EXEC proc\_fir\_mid\_las('ARYA', 'MARY-ANGEL', 'MAULE')

-- --Success

-- EXEC proc\_fir\_mid\_las('SILVER', 'FLOWERS', 'EARDLEY')

-- --Success

-- --DROP PROCEDURE proc\_fir\_mid\_las;

-- /\*###PLSQL\_3\_FUNCTIONS###\*/

-- CREATE OR REPLACE FUNCTION func\_designer\_ct RETURN NUMBER IS

-- vn\_designer\_ct NUMBER(3);

-- BEGIN

-- SELECT COUNT(designer\_id)

-- INTO vn\_designer\_ct

-- FROM designers;

-- RETURN vn\_designer\_ct;

-- END func\_designer\_ct;

-- /

-- SHOW ERRORS

-- CREATE OR REPLACE PROCEDURE proc\_func\_designer IS

-- vn\_no\_of\_designers NUMBER(3);

-- BEGIN

-- vn\_no\_of\_designers := func\_designer\_ct;

-- DBMS\_OUTPUT.PUT\_LINE('There are ' || vn\_no\_of\_designers || ' designer(s) in the database.');

-- END proc\_func\_designer;

-- /

-- SHOW ERRORS

-- /\*test proc\_func\_designer\*/

-- EXEC proc\_func\_designer;

-- --Success

-- SELECT COUNT(designer\_id) FROM designers;

-- --Success

-- --DROP PROCEDURE proc\_func\_designer;

-- --DROP FUNCTION func\_designer\_ct;

-- CREATE OR REPLACE FUNCTION func\_pay\_desn\_ct (in\_pay\_rate designers.pay\_rate%TYPE) RETURN NUMBER IS

-- vn\_designer\_ct NUMBER(3);

-- BEGIN

-- SELECT COUNT(designer\_id)

-- INTO vn\_designer\_ct

-- FROM designers

-- WHERE pay\_rate > in\_pay\_rate;

-- RETURN vn\_designer\_ct;

-- END func\_pay\_desn\_ct;

-- /

-- SHOW ERRORS

-- CREATE OR REPLACE PROCEDURE proc\_func\_pay\_desn(in\_pay\_rate designers.pay\_rate%TYPE) IS

-- vn\_no\_of\_designers NUMBER(3);

-- BEGIN

-- vn\_no\_of\_designers := func\_pay\_desn\_ct(in\_pay\_rate);

-- DBMS\_OUTPUT.PUT\_LINE('There are ' || vn\_no\_of\_designers || ' designer(s) in the database with a pay rate higher than ' || in\_pay\_rate || '.');

-- END proc\_func\_pay\_desn;

-- /

-- SHOW ERRORS

-- /\*test proc\_func\_pay\_desn\*/

-- EXEC proc\_func\_pay\_desn(10);

-- --Success

-- SELECT COUNT(designer\_id) FROM designers WHERE pay\_rate > 10;

-- --Success

-- EXEC proc\_func\_pay\_desn(5);

-- --Success

-- SELECT COUNT(designer\_id) FROM designers WHERE pay\_rate > 5;

-- --Success

-- --DROP PROCEDURE proc\_func\_pay\_desn;

-- --DROP FUNCTION func\_pay\_desn\_ct;

-- --BONUS ACTIVTY

-- CREATE OR REPLACE FUNCTION func\_cus\_username

-- (in\_customer\_id customers.customer\_id%TYPE)RETURN VARCHAR2 IS

-- vc\_firstname customers.firstname%TYPE;

-- vc\_lastname customers.lastname%TYPE;

-- vc\_username VARCHAR2(7);

-- --Change to customers.username%TYPE;

-- BEGIN

-- SELECT firstname

-- INTO vc\_firstname

-- FROM customers

-- WHERE customer\_id = in\_customer\_id;

-- SELECT lastname

-- INTO vc\_lastname

-- FROM customers

-- WHERE customer\_id = in\_customer\_id;

-- vc\_username := CONCAT(SUBSTR(vc\_firstname, 1, 2), SUBSTR(vc\_lastname, 1, 5));

-- RETURN vc\_username;

-- END func\_cus\_username;

-- /

-- SHOW ERRORS

-- CREATE OR REPLACE PROCEDURE proc\_func\_cus\_username

-- (in\_customer\_id customers.customer\_id%TYPE) IS

-- vc\_username VARCHAR2(7);

-- --Change to customers.username%TYPE;

-- BEGIN

-- vc\_username := func\_cus\_username(in\_customer\_id);

-- DBMS\_OUTPUT.PUT\_LINE('Customers: ' || in\_customer\_id || ' username set to ' || vc\_username || '.');

-- --Change to -¬

-- --UPDATE customers

-- --SET username = vc\_username

-- --WHERE customer\_id = in\_customer\_id;

-- END proc\_func\_cus\_username;

-- /

-- SHOW ERRORS

-- /\*test proc\_func\_cus\_username\*/

-- EXEC proc\_func\_cus\_username(20000000);

-- SELECT customer\_id, firstname, lastname

-- --Change to SELECT customer\_id, username

-- FROM customers

-- WHERE customer\_id = 20000000;

-- --DROP PROCEDURE proc\_func\_cus\_username;

-- --DROP FUNCTION func\_cus\_username

-- /\*###PLSQL\_4\_TRIGGERS###\*/

-- CREATE OR REPLACE TRIGGER trig\_ar\_date\_ck

-- BEFORE INSERT OR UPDATE OF arrival\_date ON bookings

-- FOR EACH ROW

-- WHEN(NEW.arrival\_date < SYSDATE)

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('The arrival date cannot exist in the past');

-- RAISE\_APPLICATION\_ERROR

-- (-20000, 'ERROR - THE ARRIVAL DATE CAN NOT EXIST BEFORE TODAYS DATE!

-- ARRIVAL DATE: ' || :NEW.arrival\_date || ' TODAYS DATE: ' || SYSDATE);

-- END trig\_ar\_date\_ck;

-- /

-- SHOW ERRORS

-- /\*test trig\_ar\_date\_ck\*/

-- INSERT INTO bookings (booking\_id, designer\_id, arrival\_date)

-- VALUES (bookings\_seq.NEXTVAL, 10000000, '01-JUN-2016');

-- --Throws error - Success

-- INSERT INTO bookings (booking\_id, designer\_id, arrival\_date)

-- VALUES (bookings\_seq.NEXTVAL, 10000000, '01-JUN-2019');

-- --Should not throw error - Success

-- --DROP TRIGGER trig\_ar\_date\_ck;

-- CREATE OR REPLACE TRIGGER trig\_hire\_date

-- AFTER INSERT OR UPDATE ON designers

-- FOR EACH ROW

-- WHEN (NEW.hire\_date IS NOT NULL)

-- DECLARE

-- vd\_today designers.hire\_date%TYPE := SYSDATE;

-- BEGIN

-- IF (vd\_today - (365\*5)) >= :NEW.hire\_date

-- THEN

-- DBMS\_OUTPUT.PUT\_LINE('Designer has been hired for or over 5 years.');

-- ELSE

-- DBMS\_OUTPUT.PUT\_LINE('Designer has not been hired for more 5 years.');

-- END IF;

-- END trig\_hire\_date;

-- /

-- SHOW ERRORS

-- /\*test trig\_hire\_date\*/

-- INSERT INTO designers (designer\_id, hire\_date)

-- VALUES (designers\_seq.NEXTVAL, '01-JAN-2013');

-- --Trigger designer has been hired for over 5 years - Success

-- INSERT INTO designers (designer\_id, hire\_date)

-- VALUES (designers\_seq.NEXTVAL, '01-JAN-2014');

-- --Trigger designer has not been hired for 5 years - Success

-- --DROP TRIGGER trig\_hire\_date;

-- --BONUS ACTIVITY

-- CREATE OR REPLACE TRIGGER trig\_desn\_dob\_suc

-- BEFORE INSERT ON designers

-- FOR EACH ROW

-- WHEN ((MONTHS\_BETWEEN(SYSDATE, NEW.dob)/12) > 18)

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('Designer added.');

-- END trig\_desn\_dob\_suc;

-- /

-- SHOW ERRORS

-- /\*test trig\_desn\_dob\_suc \*/

-- INSERT INTO designers (designer\_id, dob)

-- VALUES (designers\_seq.NEXTVAL, '01-JAN-2000');

-- --Designer added. - Success

-- --DROP TRIGGER trig\_desn\_dob\_suc;

-- CREATE OR REPLACE TRIGGER trig\_desn\_dob\_dec

-- BEFORE INSERT ON designers

-- FOR EACH ROW

-- WHEN ((MONTHS\_BETWEEN(SYSDATE, NEW.dob)/12) < 18)

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('Designer is too young.');

-- RAISE\_APPLICATION\_ERROR

-- (-20000, 'ERROR - THE DESIGNER IS TOO YOUNG!');

-- END trig\_desn\_dob\_dec;

-- /

-- SHOW ERRORS

-- /\*test trig\_desn\_dob\_dec\*/

-- INSERT INTO designers (designer\_id, dob)

-- VALUES (designers\_seq.NEXTVAL, '01-JAN-2010');

-- --ERROR - THE DESIGNER IS TOO YOUNG! - Success

-- --DROP TRIGGER trig\_desn\_dob\_dec;

-- /\*###PLSQL\_4.5\_COMBINED--IMPROVED###\*/

-- /\*Function determines the months left until retirement based upon the

-- dob and retirement age passed in its parameters\*/

-- CREATE OR REPLACE FUNCTION func\_retire\_age (in\_dob designers.dob%TYPE, in\_retire\_age NUMBER)

-- RETURN NUMBER IS

-- vn\_age NUMBER(2) := FLOOR(MONTHS\_BETWEEN(SYSDATE, in\_dob)/12);

-- vn\_months\_left NUMBER(3);

-- BEGIN

-- vn\_months\_left := MOD(in\_retire\_age, vn\_age)\*12;

-- RETURN vn\_months\_left;

-- END func\_retire\_age;

-- /

-- SHOW ERRORS

-- /\*Trigger runs before insert, update or delete on the designers table

-- taking appropriate action dependant on the action that triggered the trigger.

-- If inserting it displays the name of who is being inserts. If updating it

-- sends the dob of the record being updated to a function which returns the

-- remaining months until retirements base on a hardcoded age of 67. If deleteing

-- it displays the name of who is being deleted.\*/

-- CREATE OR REPLACE TRIGGER trig\_designer

-- BEFORE INSERT OR UPDATE OR DELETE ON designers

-- FOR EACH ROW

-- DECLARE

-- vn\_age NUMBER(2) := FLOOR(MONTHS\_BETWEEN(SYSDATE, :NEW.dob)/12);

-- vn\_months\_left NUMBER(3);

-- BEGIN

-- IF (INSERTING OR UPDATING)

-- THEN

-- IF INSERTING

-- THEN

-- --INSERTING

-- DBMS\_OUTPUT.PUT\_LINE('You are adding ' || :NEW.firstname|| ' who is ' || vn\_age ||' years old.');

-- ELSE

-- --UPDATING

-- vn\_months\_left := func\_retire\_age(:OLD.dob, 67);

-- DBMS\_OUTPUT.PUT\_LINE(vn\_months\_left || ' months until retirement.');

-- END IF;

-- ELSE

-- --DELETING

-- DBMS\_OUTPUT.PUT\_LINE('You are deleting ' || :OLD.firstname || '.');

-- END IF;

-- END trig\_designer;

-- /

-- SHOW ERRORS

-- /\*test trig\_designer\*/

-- INSERT INTO designers (designer\_id, firstname, dob)

-- VALUES (designers\_seq.NEXTVAL, 'JOSH', '26-JAN-1995');

-- --You are adding JOSH who is 23 years old. - Success

-- UPDATE designers SET lastname = 'CHALLAND'

-- WHERE designer\_id = 10000021;

-- --252 months until retirement. - Success

-- DELETE FROM designers WHERE designer\_id = 10000021;

-- --You are deleting JOSH. - Success

-- /\*Procedure takes in a designers id and the age at which they plan to

-- retire, it then stores the designers date of birth using a query. the

-- dob and age they plan to retire is send to a function which returns the

-- number of months they have left until retirement.\*/

-- CREATE OR REPLACE PROCEDURE proc\_retire\_param

-- (in\_designer\_id designers.designer\_id%TYPE, in\_retire\_age NUMBER)

-- IS

-- vn\_months\_left NUMBER(3);

-- vd\_dob DATE;

-- BEGIN

-- SELECT dob

-- INTO vd\_dob

-- FROM designers

-- WHERE designer\_id = in\_designer\_id;

-- vn\_months\_left := func\_retire\_age(vd\_dob, in\_retire\_age);

-- DBMS\_OUTPUT.PUT\_LINE(vn\_months\_left || ' months until retirement.');

-- END proc\_retire\_param;

-- /

-- SHOW ERRORS

-- /\*test proc\_retire\_param\*/

-- EXEC proc\_retire\_param(10000000, 67);

-- --300 months until retirement. - Success

-- EXEC proc\_retire\_param(10000000, 60);

-- --216 months until retirement. - Success

-- --DROP PROCEDURE proc\_retire\_param;

-- --DROP TRIGGER trig\_designer;

-- --DROP FUNCTION func\_retire\_age;

-- /\*###PLSQL\_5\_CURSORS###\*/

-- CREATE OR REPLACE PROCEDURE proc\_imp\_cursor

-- (in\_firstname designers.firstname%Type)

-- IS

-- BEGIN

-- DELETE FROM designers WHERE firstname = in\_firstname;

-- IF SQL%FOUND

-- THEN

-- DBMS\_OUTPUT.PUT\_LINE(in\_firstname || ' REMOVED.');

-- ELSE

-- DBMS\_OUTPUT.PUT\_LINE(in\_firstname || ' DOES NOT EXIST.');

-- END IF;

-- END proc\_imp\_cursor;

-- /

-- SHOW ERRORS

-- /\*test proc\_imp\_cursor\*/

-- INSERT INTO designers (designer\_id, firstname)

-- VALUES(99999999, 'CANDY');

-- EXEC proc\_imp\_cursor ('CANDY');

-- --CANDY REMOVED. - Success

-- EXEC proc\_imp\_cursor ('DAISY');

-- --DAISY DOES NOT EXIST. - Success

-- SELECT firstname FROM designers;

-- --NO CANDY OR DAISY IN TABLE - Success

-- --DROP PROCEDURE proc\_imp\_cursor;

-- CREATE OR REPLACE PROCEDURE proc\_exp\_cur\_book\_high

-- (in\_cost bookings.cost%Type)

-- IS

-- CURSOR cur\_book\_mor IS

-- SELECT b.booking\_id, b.cost, b.order\_date, d.firstname

-- FROM bookings b

-- JOIN designers d

-- ON b.designer\_id = d.designer\_id

-- WHERE cost > in\_cost;

-- rec\_cur\_book\_mor cur\_book\_mor%ROWTYPE;

-- vn\_row\_count NUMBER(2) := 0;

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('Number bookings with a cost more than ' || in\_cost);

-- FOR rec\_cur\_book\_mor IN cur\_book\_mor LOOP

-- DBMS\_OUTPUT.PUT\_LINE(cur\_book\_mor%ROWCOUNT || ' Booking\_id: ' || rec\_cur\_book\_mor.booking\_id ||

-- ' Booking cost: ' || rec\_cur\_book\_mor.cost || ' Booking date: '

-- || rec\_cur\_book\_mor.order\_date || ' Designer firstname: ' ||

-- rec\_cur\_book\_mor.firstname);

-- vn\_row\_count := vn\_row\_count + 1;

-- END LOOP;

-- IF vn\_row\_count = 0

-- THEN

-- DBMS\_OUTPUT.PUT\_LINE(vn\_row\_count || ' bookings cost more than ' || in\_cost);

-- END IF;

-- END proc\_exp\_cur\_book\_high;

-- /

-- SHOW ERRORS

-- /\*test proc\_exp\_cur\_book\_high\*/

-- EXEC proc\_exp\_cur\_book\_high(900);

-- /\*

-- Bookings with cost more than 900

-- Booking\_id: 40000001 Booking cost: 13222.5 Booking date: 03-AUG-17 Designer

-- firstname: STEVE

-- Booking\_id: 40000002 Booking cost: 988.25 Booking date: 17-NOV-15 Designer

-- firstname: MYLES

-- Booking\_id: 40000003 Booking cost: 20000.49 Booking date: 06-MAY-14 Designer

-- firstname: ELIANE

-- Booking\_id: 40000004 Booking cost: 2999.99 Booking date: 19-JAN-17 Designer

-- firstname: MYLES

-- \*/

-- --Success

-- SELECT booking\_id FROM bookings WHERE cost > 900;

-- /\*

-- 40000001

-- 40000002

-- 40000003

-- 40000004

-- \*/

-- --Success

-- --DROP PROCEDURE proc\_exp\_cur\_book\_high;

-- CREATE OR REPLACE PROCEDURE proc\_exp\_cur\_book\_low

-- (in\_cost bookings.cost%Type)

-- IS

-- CURSOR cur\_book\_low IS

-- SELECT b.booking\_id, b.cost, b.order\_date, d.firstname

-- FROM bookings b

-- JOIN designers d

-- ON b.designer\_id = d.designer\_id

-- WHERE cost < in\_cost;

-- rec\_cur\_book\_low cur\_book\_low%ROWTYPE;

-- vn\_row\_count NUMBER(2) := 0;

-- BEGIN

-- DBMS\_OUTPUT.PUT\_LINE('Bookings with a cost less than ' || in\_cost);

-- FOR rec\_cur\_book\_low IN cur\_book\_low LOOP

-- DBMS\_OUTPUT.PUT\_LINE('Booking\_id: ' || rec\_cur\_book\_low.booking\_id ||

-- ' Booking cost: ' || rec\_cur\_book\_low.cost || ' Booking date: '

-- || rec\_cur\_book\_low.order\_date || ' Designer firstname: ' ||

-- rec\_cur\_book\_low.firstname);

-- vn\_row\_count := vn\_row\_count + 1;

-- END LOOP;

-- IF vn\_row\_count = 0

-- THEN

-- DBMS\_OUTPUT.PUT\_LINE(vn\_row\_count || ' bookings cost less than ' || in\_cost);

-- END IF;

-- END proc\_exp\_cur\_book\_low;

-- /

-- SHOW ERRORS

-- /\*test proc\_exp\_cur\_book\_low\*/

-- EXEC proc\_exp\_cur\_book\_low(900);

-- /\*

-- Bookings with cost less than 900

-- Booking\_id: 40000000 Booking cost: 766.75 Booking date: 14-FEB-16 Designer

-- firstname: AKPAR

-- \*/

-- --Success

-- SELECT booking\_id FROM bookings WHERE cost < 900;

-- /\*

-- 40000000

-- \*/

-- --Success

-- --DROP PROCEDURE proc\_exp\_cur\_book\_low;

-- --BONUS ACTIVITY

-- /\*Procedure takes in a rating and creates a cursor which queries all

-- ratings in testimonials which are greater than the input rating. The

-- procedure then opens the cursors and checks if a row was found, displaying

-- and appropriate message. If a row was found it loops through the results

-- displaying the relevant information from the results found.\*/

-- CREATE OR REPLACE PROCEDURE proc\_higher\_rate

-- (in\_rating testimonials.rating%Type)

-- IS

-- CURSOR cur\_higher\_rate IS

-- SELECT t.rating, d.designer\_id

-- FROM testimonials t

-- JOIN designers d

-- ON t.designer\_id = d.designer\_id

-- WHERE rating > in\_rating;

-- rec\_cur\_higher\_rate cur\_higher\_rate%ROWTYPE;

-- BEGIN

-- OPEN cur\_higher\_rate;

-- FETCH cur\_higher\_rate INTO rec\_cur\_higher\_rate;

-- IF cur\_higher\_rate%NOTFOUND

-- THEN

-- DBMS\_OUTPUT.PUT\_LINE('There are no designers with a rating higher than ' || in\_rating);

-- ELSE

-- DBMS\_OUTPUT.PUT\_LINE('Designers with a rating higher than ' || in\_rating);

-- END IF;

-- WHILE cur\_higher\_rate%FOUND

-- LOOP

-- DBMS\_OUTPUT.PUT\_LINE(cur\_higher\_rate%ROWCOUNT || ' Designer ID: ' || rec\_cur\_higher\_rate.designer\_id ||

-- ' Designers Rating: ' || rec\_cur\_higher\_rate.rating);

-- FETCH cur\_higher\_rate INTO rec\_cur\_higher\_rate;

-- END LOOP;

-- CLOSE cur\_higher\_rate;

-- END proc\_higher\_rate;

-- /

-- SHOW ERRORS

-- /\*test proc\_higher\_rate\*/

-- EXEC proc\_higher\_rate(5);

-- /\*

-- Designers with a rating higher than 5

-- 1 Designer ID: 10000001 Designers Rating: 7

-- 2 Designer ID: 10000001 Designers Rating: 7

-- 3 Designer ID: 10000001 Designers Rating: 10

-- 4 Designer ID: 10000003 Designers Rating: 6

-- \*/

-- --Success

-- SELECT designer\_id, rating FROM testimonials WHERE rating > 5;

-- /\*

-- 10000001 7

-- 10000001 7

-- 10000001 10

-- 10000003 6

-- \*/

-- --Success

-- --DROP PROCEDURE proc\_higher\_rate;

-- /\*###PLSQL\_EXTRAS###\*/

-- /\*Function takes in the total area of a room, a pay rate of a designer,

-- and average rating of said designer. The function then uses the three

-- arguements to generate a total cost which is return to the calling routine.\*/

-- CREATE OR REPLACE FUNCTION func\_booking\_cost

-- (in\_total\_area NUMBER, in\_pay\_rate designers.pay\_rate%TYPE, in\_avg\_rate testimonials.rating%TYPE)

-- RETURN NUMBER IS

-- out\_vn\_total\_cost NUMBER(8);

-- BEGIN

-- out\_vn\_total\_cost := in\_total\_area \* ((in\_pay\_rate \* 0.025)\*(in\_avg\_rate\*0.1));

-- RETURN out\_vn\_total\_cost;

-- END func\_booking\_cost;

-- /

-- SHOW ERRORS;

-- --DROP FUNCTION func\_booking\_cost;

-- /\*Function calculates and returns the total area based upon two parameters passed if which are

-- assumed to be measurements.\*/

-- CREATE OR REPLACE FUNCTION func\_total\_area

-- (in\_length customer\_rooms.room\_length%TYPE, in\_width customer\_rooms.room\_width%TYPE)

-- RETURN NUMBER IS

-- out\_vn\_total\_area NUMBER(10);

-- BEGIN

-- out\_vn\_total\_area := in\_length \* in\_width;

-- RETURN out\_vn\_total\_area;

-- END func\_total\_area;

-- /

-- SHOW ERRORS;

-- --DROP FUNCTION func\_total\_area;

-- /\*Procedure takes in a total area and booking id. The booking id is then

-- used in a cursors to find all customer rooms under the same booking, each

-- customer rooms area is then added to the total area of the booking. It is also

-- used in two queries that extracted the designers pay rate and average rating

-- via joins and the AVG function into two variables. The total area, pay rate,

-- and average rating are then passed to a function which returns a total cost

-- which is used to update the booking associated with the booking id with a

-- new cost.\*/

-- CREATE OR REPLACE PROCEDURE proc\_booking\_cost

-- (in\_total\_area NUMBER, in\_booking\_id bookings.booking\_id%TYPE)

-- IS

-- CURSOR cur\_total\_area IS

-- SELECT cr.room\_length, cr.room\_width

-- FROM customer\_rooms cr

-- WHERE booking\_id = in\_booking\_id;

-- vn\_total\_area NUMBER(10) := in\_total\_area;

-- vn\_total\_cost bookings.cost%TYPE;

-- vn\_pay\_rate designers.pay\_rate%TYPE;

-- vn\_avg\_rat testimonials.rating%TYPE;

-- rec\_cur\_total\_area cur\_total\_area%ROWTYPE;

-- BEGIN

-- FOR rec\_cur\_total\_area IN cur\_total\_area LOOP

-- vn\_total\_area := vn\_total\_area + func\_total\_area(rec\_cur\_total\_area.room\_length, rec\_cur\_total\_area.room\_width);

-- END LOOP;

-- SELECT d.pay\_rate

-- INTO vn\_pay\_rate

-- FROM designers d

-- JOIN bookings b

-- ON d.designer\_id = b.designer\_id

-- WHERE b.booking\_id = in\_booking\_id;

-- SELECT AVG(t.rating)

-- INTO vn\_avg\_rat

-- FROM testimonials t

-- JOIN designers d

-- ON t.designer\_id = d.designer\_id

-- JOIN bookings b

-- ON d.designer\_id = b.designer\_id

-- WHERE b.booking\_id = in\_booking\_id

-- GROUP BY t.designer\_id;

-- vn\_total\_cost := func\_booking\_cost(vn\_total\_area, vn\_pay\_rate, vn\_avg\_rat);

-- UPDATE bookings

-- SET cost = vn\_total\_cost

-- WHERE booking\_id = in\_booking\_id;

-- END proc\_booking\_cost;

-- /

-- SHOW ERRORS

-- --DROP PROCEDURE proc\_booking\_cost;

-- /\*Trigger runs when a customer room is inserted. It takes the width and length

-- and works out the total area and passes it into the procedure to update the

-- bookings cost\*/

-- CREATE OR REPLACE TRIGGER trig\_booking\_cost

-- BEFORE INSERT ON customer\_rooms

-- FOR EACH ROW

-- WHEN ((NEW.room\_width IS NOT NULL) AND(NEW.room\_length IS NOT NULL))

-- DECLARE

-- vn\_total\_area NUMBER(10);

-- BEGIN

-- vn\_total\_area := func\_total\_area(:NEW.room\_width, :NEW.room\_length);

-- proc\_booking\_cost(vn\_total\_area, :NEW.booking\_id);

-- END trig\_booking\_cost;

-- /

-- SHOW ERRORS

-- --original cost: 766.75 --select cost from bookings where booking\_id = 40000000;

-- --desinger\_id: 10000000 --select designer\_id from bookings where booking\_id = 40000000;

-- --pay\_rate: 9 --select pay\_rate from designers where designer\_id = 10000000;

-- /\*

-- INSERT INTO testimonials (testimonial\_id, designer\_id, rating)

-- VALUES (testimonials\_seq.NEXTVAL, 10000000, 5);

-- --DELETE FROM customer\_rooms WHERE customer\_id = 20000000;

-- \*/

-- --avg(rating): 5 --select AVG(rating) from testimonials where designer\_id = 10000000;

-- --New cost: 281.25

-- INSERT INTO customer\_rooms (customer\_room\_id, customer\_id, booking\_id, room\_width, room\_length)

-- VALUES (customer\_rooms\_seq.NEXTVAL, 20000000, 40000000, 50, 50);

-- /\*

-- select booking\_id, cost from bookings;

-- BOOKING\_ID COST

-- ---------- ----------

-- 40000000 281

-- --SUCCESS

-- \*/

-- -- DROP TRIGGER trig\_booking\_cost;