SWD503 - Enterprise Database Development

Jonathan Cauchi

18/01/19

1 - Overview

A report describing the development of the USS Starship Enterprise Database. Database developed in collaboration with Alexei Jarv, Ryan Hinks and Alex Tripp. Report is written in LaTeX, a document preperation tool. The Entity Relationship Diagram (ERD) was designed with *Software Ideas Modeller. Trello*, a project management tool was used to keep track of our progress throughout the weekly sprints and individual workload sessions.

2 - Elicit Requirements and Analysis

Database design encompasses an entire process from the creation of requirements, business processes, logical analysis and physical database constructs to the deployment of the database. In this database design, data modelling was done of which includes the modelling of entities and their relationship with each other. Figuring out the business processes and the activities that will be carried out post deployment was key to set the ground running for a stable, working database system. It was necessary to address issues regarding which information was required, how different parts relate to each other and how the entire system needs to be implemented.

Functional Requirements:

The first thing agreed on the database design is that for each table, there should be a primary key from which all other values inside the table will depend on. It was agreed that the database will satisfy the **Third Normal**

Form requirements. The objective was to cover the business requirements of a spaceship while maintaining a simple and effective design. Many tables may undermine the database' overall performance as well as cost extra implementation time. Therefore, the design was agreed to be made as simple as possible while ensuring it covers all the necessary requirements for the behaviour expected.

Considering the database is for the USS Starship Enterprise, **use cases are as follows:**

- As an operator, I want to be able to calculate the distance between different star systems.
- As an operator, I want to be able to view the captains log.
- As an operator, I want to be able to view and return the location of crew members.
- As an operator, I want to be able to beam crew members onto and out of the enterprise and update beaming log accordingly.
- As an operator, I want to be able to view the ship maintenance status.
- As an operator, I want to be able to view the warp drive status.
- As an operator, I want to be able to keep track and monitor the current and previous crew and update their rank if necessary.
- As an operator, I want to be able to take defensive measures in case of an attack.
- As an operator, I want to be able to calculate the age of a given crew member.
- As an operator, I want to be able to view the details regarding different star systems.
- As an operator, I want to be able to view the subsystems, their function, health level and activity status.
- As an operator, I want to be able to view details regarding the stations that make up the spaceship.
- As an operator, I want to be able to view the subsystems and their activity status.

3 - Logical Database Design

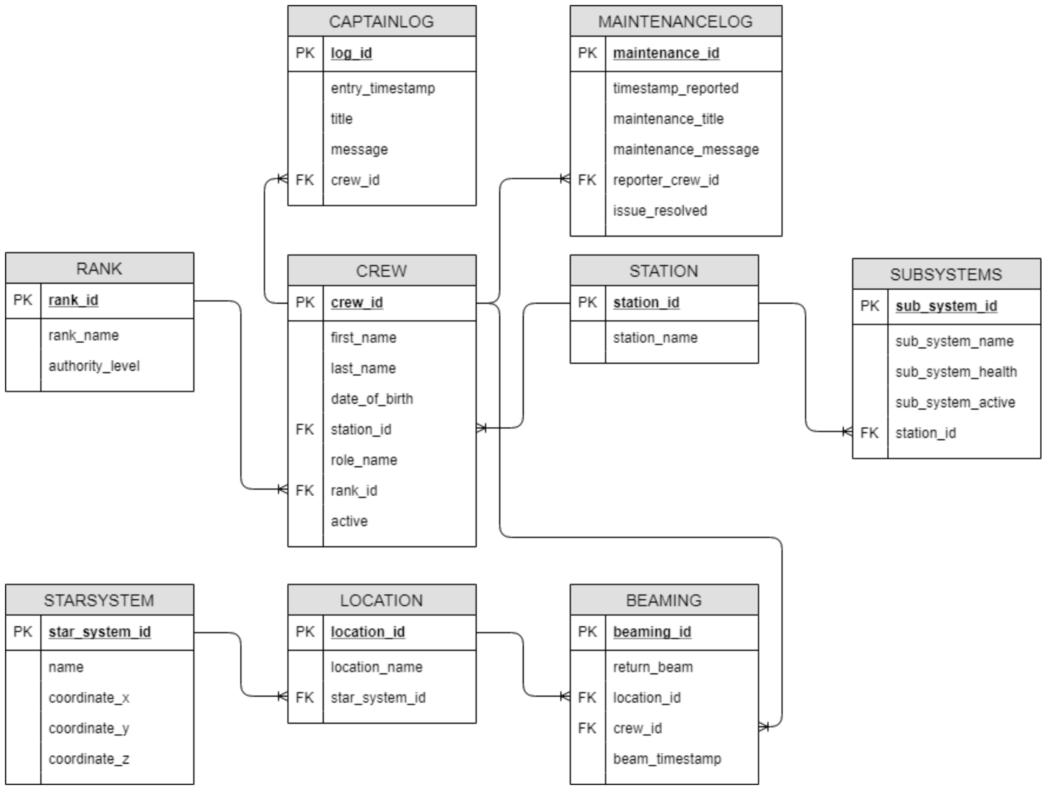


Figure 1: ERD Design for the USS Enterprise Database.

Initially, more of a top-down approach was undertaken to formulate the structure of the database by drawing up obvious required entities such as CREW, LOCATION, BEAMING and CAPTAIN'S LOG. Then naturally, the flow traversed to more of a Normalization approach, consistently adding entities that at the time were thought to be worthy of induction.

First and foremost, a no brainer, a track record of all crew members is required to be existing and updated according to the change of their station location, physical location, activity status and rank. Their station location can change simply from traversal to other stations or to specific surrounding star-systems via a 'beaming' technology. It is necessary to keep records concerning the crew members being beamed, their physical location outside the ship, their returning status, whether they have been promoted or demoted, whether they are highly ranked enough to access the Captain's Log and so on.

Details regarding the surrounding star-systems and their physical coordinates in the discoverable universe are necessary as to track the ship's location as well as crew members location in the cosmos. Details regarding the sub-systems of the ship, their function and location is necessary to be stored as well as their health level, in case of an attack, there needs to be a record of whether the sub-systems are well maintained enough to do a certain duty.

Captain's Log is used for the documentation regarding the changes that have occured in the database, depictions of external scenarios or both simultaneously. Only the individual with the role of Captain can access and input information in the log. Each log input needs to be backed up with the details of its entry date, title and a message portraying the purpose of the log.

4 - PLSQL Functions

A PL/SQL function is similar to a PL/SQL procedure except that a function returns a value. When creating a function, it is necessary to define what the function needs to do and then call the function to operate a specific task. Parameters can be passed over in order to return values according to their relation to the parameters defined/inputted or merely passing through a value that requires arithmetic processing and then return the finished result.

The practical thing about functions in general and their consistent usage within database design is their usability. Once defined, a function can be used time and time again to return values according to a certain input. It can save time in the long run to have functions that return desired results to solve repeating occuring problems or merely just to view a specific chunk of data within possible massive data sets. These sort of routines are at the basis of a fast growing domain regarding Enterprise data management.

Now, considering the structure of the database regarding entities and relationships is defined. There needs to be certain functions that explore many tables that are related via foreign key reference as to obtain data for certain questions that the database may not completely define directly . Four examples of proposable functions for the database are as follows:

- **Return the location of a given crew member.**

- **Distance between two star systems.**

- **Returning age of crew member.**

- **Return the average health of a single station, then calculate the overall (average) health of the ship.**

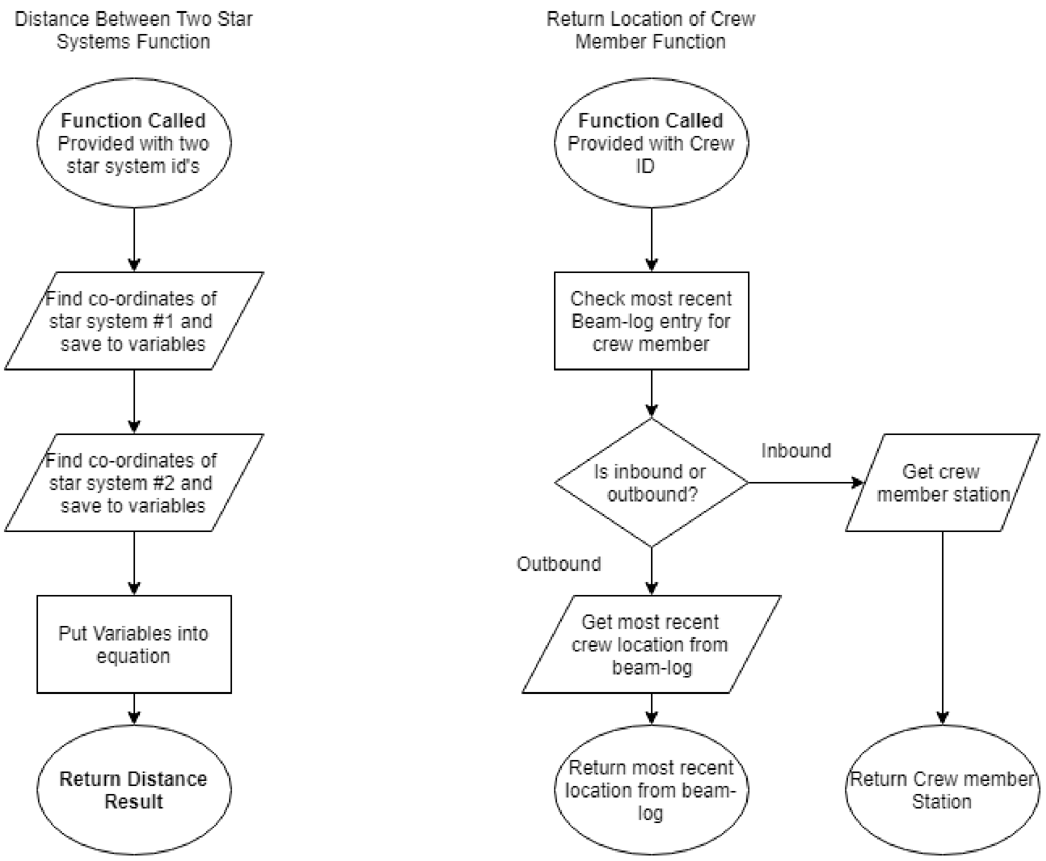
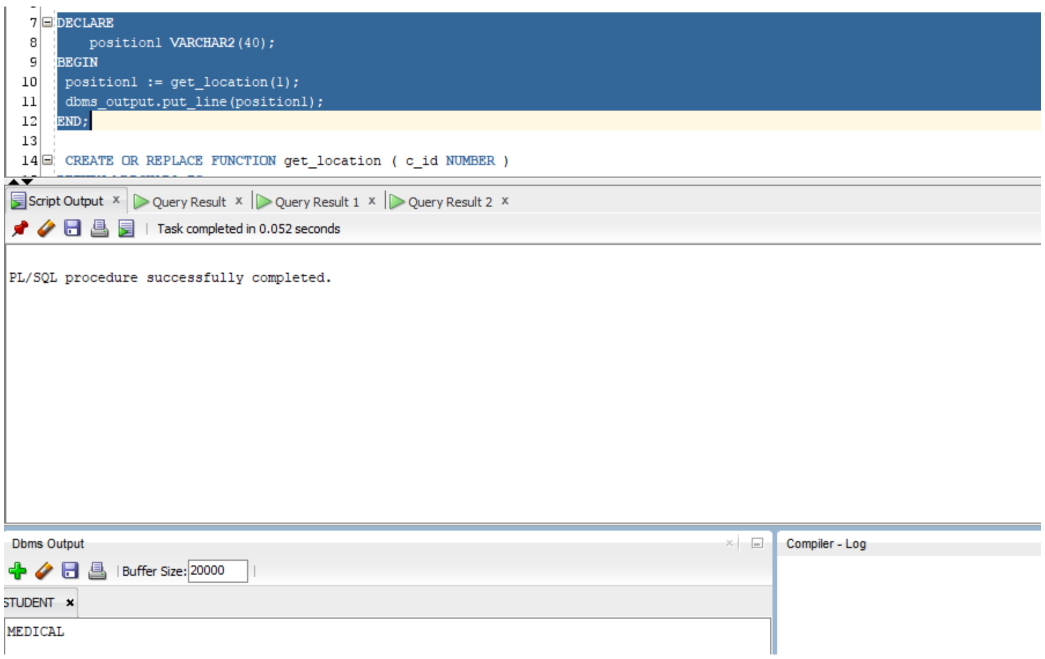

Distance Between Two Star Systems Function
Function Called Provided with two star system id's
Find co-ordinates of star system #1 and save to variables
Find co-ordinates of star system #2 and save to variables
Put Variables into equation
Return Distance Result
Return Location of Crew Member Function
Function Called Provided with Crew ID
Check most recent Beam-log entry for crew member
Is inbound or outbound?
Inbound
Get crew member station
Outbound
Get most recent crew location from beam-log
Return most recent location from beam-log
Return Crew member Station

Figure 2: Data Flow Diagram.

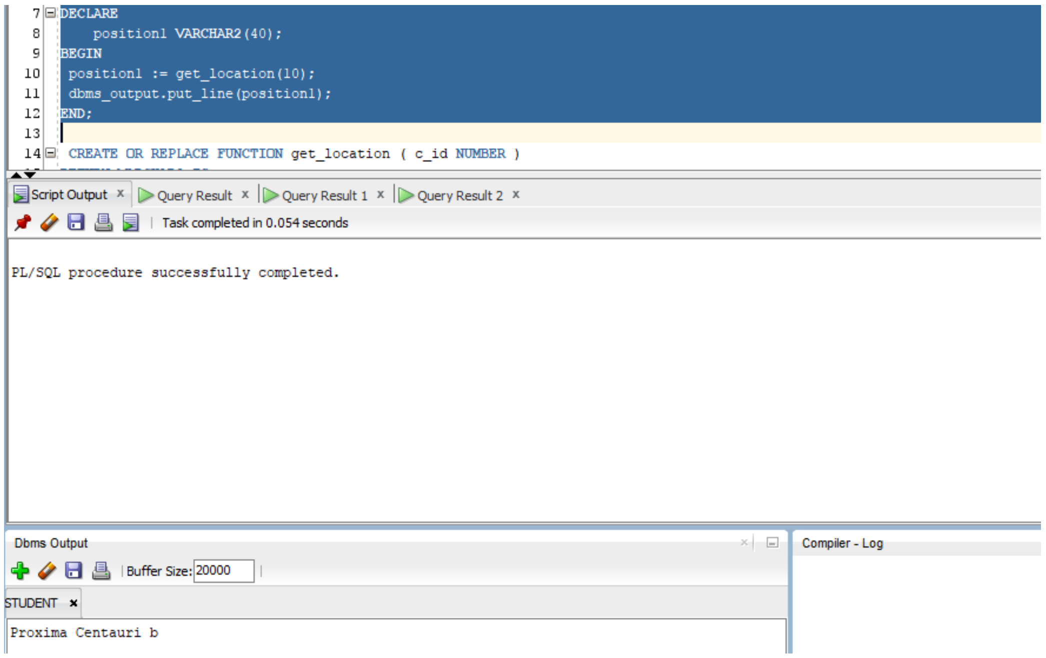


```
DECLARE
    position1 VARCHAR2(40);
BEGIN
 position1 := get_location(1);
 dbms_output.put_line(position1);
END;

 CREATE OR REPLACE FUNCTION get_location ( c_id NUMBER )
```

```
PL/SQL procedure successfully completed.
```

```
MEDICAL
```

Figure 3: Searching for the station location of a crew member that is on the ship using the **get location** function.



```
DECLARE
    position1 VARCHAR2(40);
BEGIN
 position1 := get_location(10);
 dbms_output.put_line(position1);
END;

 CREATE OR REPLACE FUNCTION get_location ( c_id NUMBER )
```

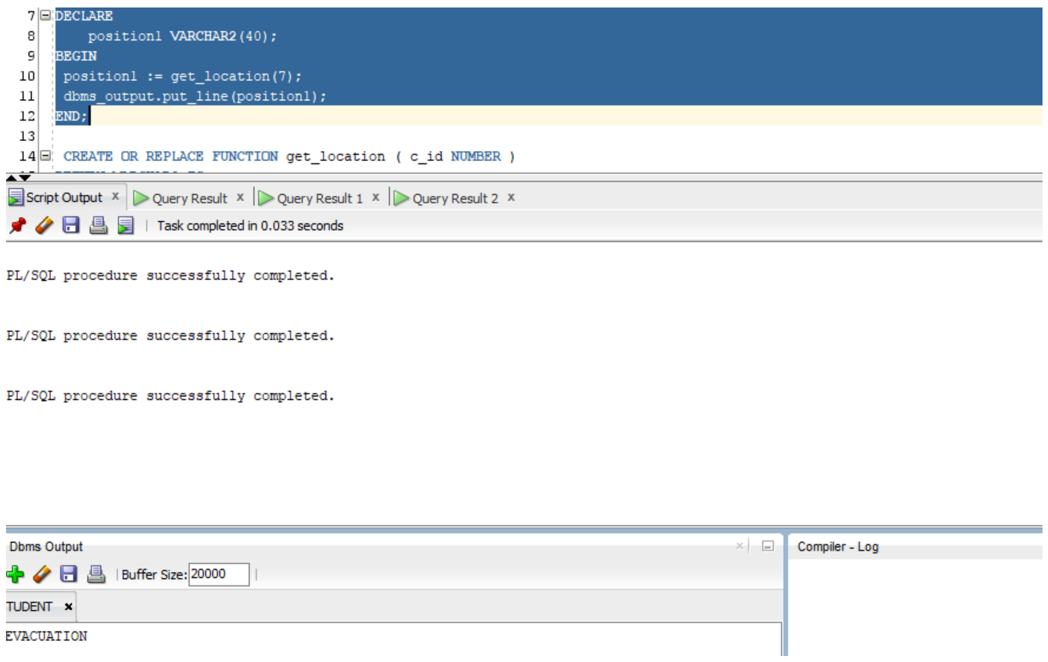
Task completed in 0.054 seconds

```
PL/SQL procedure successfully completed.
```

Buffer Size: 20000

```
Proxima Centauri b
```

Figure 4: Searching for the starsystem location of a crew member that is not on the ship but has records in the beaming table using the get location function.

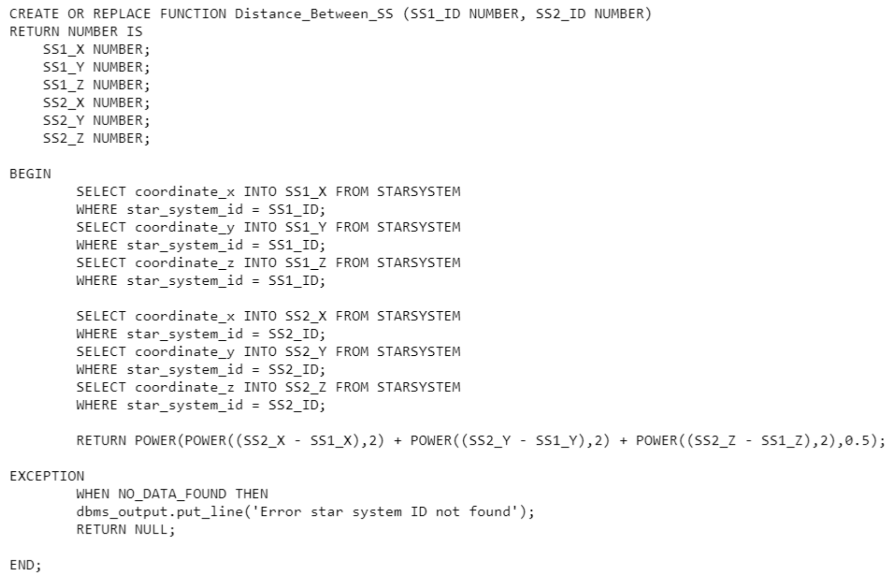


```
DECLARE
    position1 VARCHAR2(40);
BEGIN
 position1 := get_location(7);
 dbms_output.put_line(position1);
END;

 CREATE OR REPLACE FUNCTION get_location ( c_id NUMBER )
```

```
PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.
```

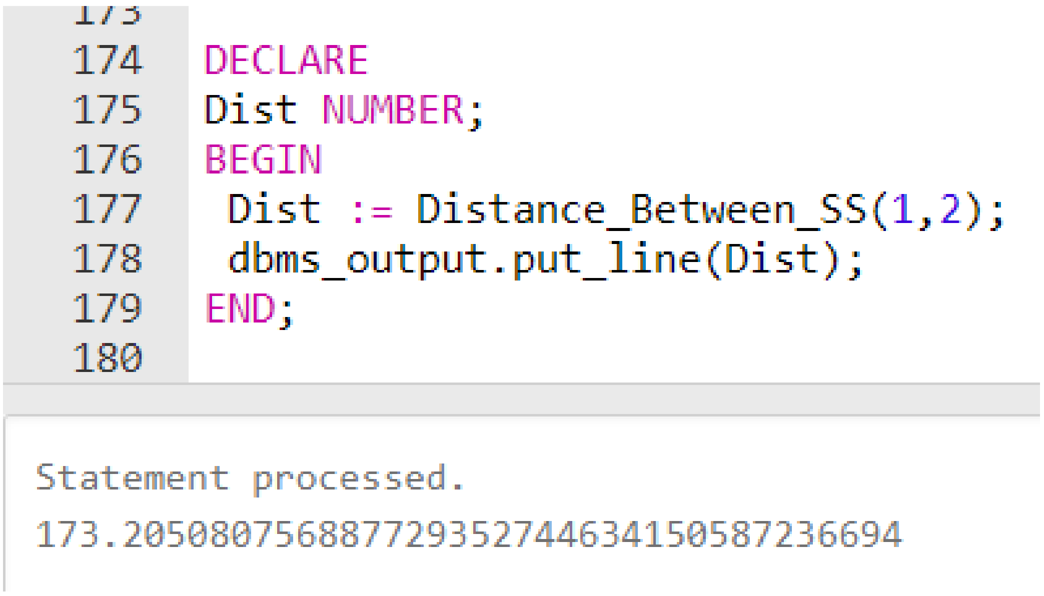
```
EVACUATION
```

Figure 5: Searching for the station location of a crew member that is on the ship but does not have records in the beaming table using the **get location** function.



```
CREATE OR REPLACE FUNCTION Distance_Between_SS (SS1_ID NUMBER, SS2_ID NUMBER)
RETURN NUMBER IS
    SS1_X NUMBER;
    SS1_Y NUMBER;
    SS1_Z NUMBER;
    SS2_X NUMBER;
    SS2_Y NUMBER;
    SS2_Z NUMBER;

BEGIN
        SELECT coordinate_x INTO SS1_X FROM STARSYSTEM
        WHERE star_system_id = SS1_ID;
        SELECT coordinate_y INTO SS1_Y FROM STARSYSTEM
        WHERE star_system_id = SS1_ID;
        SELECT coordinate_z INTO SS1_Z FROM STARSYSTEM
        WHERE star_system_id = SS1_ID;

        SELECT coordinate_x INTO SS2_X FROM STARSYSTEM
        WHERE star_system_id = SS2_ID;
        SELECT coordinate_y INTO SS2_Y FROM STARSYSTEM
        WHERE star_system_id = SS2_ID;
        SELECT coordinate_z INTO SS2_Z FROM STARSYSTEM
        WHERE star_system_id = SS2_ID;

        RETURN POWER(POWER((SS2_X - SS1_X),2) + POWER((SS2_Y - SS1_Y),2) + POWER((SS2_Z - SS1_Z),2),0.5);

EXCEPTION
        WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Error star system ID not found');
        RETURN NULL;

END;
```

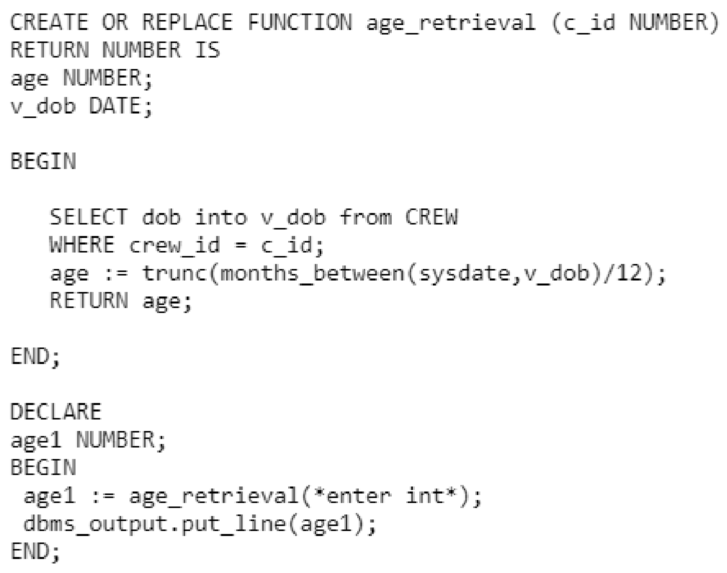
Figure 6: Function for returning the distance between 2 star systems via Cartesian coordinate system.



```
DECLARE
Dist NUMBER;
BEGIN
 Dist := Distance_Between_SS(1,2);
 dbms_output.put_line(Dist);
END;
```

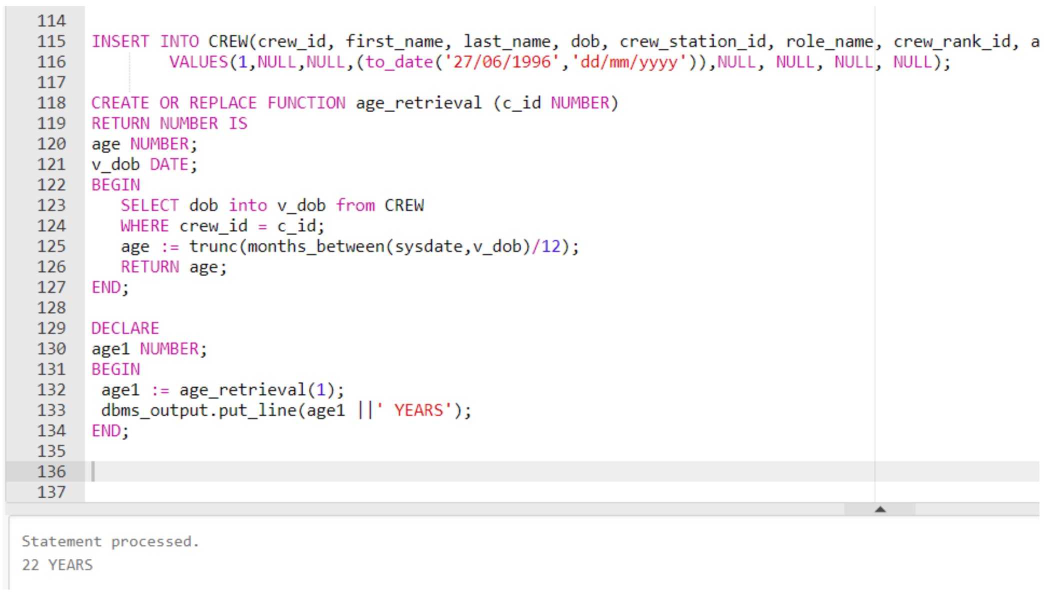
```
Statement processed.
173.2050807568877293527446341505872366 94
```

Figure 7: The distance between 'Solar System' and 'Alpha Centauri' is as shows according to the defined coordinates in the database.



```
CREATE OR REPLACE FUNCTION age_retrieval (c_id NUMBER)
RETURN NUMBER IS
age NUMBER;
v_dob DATE;

BEGIN

   SELECT dob into v_dob from CREW
   WHERE crew_id = c_id;
   age := trunc(months_between(sysdate,v_dob)/12);
   RETURN age;

END;

DECLARE
age1 NUMBER;
BEGIN
 age1 := age_retrieval(*enter int*);
 dbms_output.put_line(age1);
END;
```

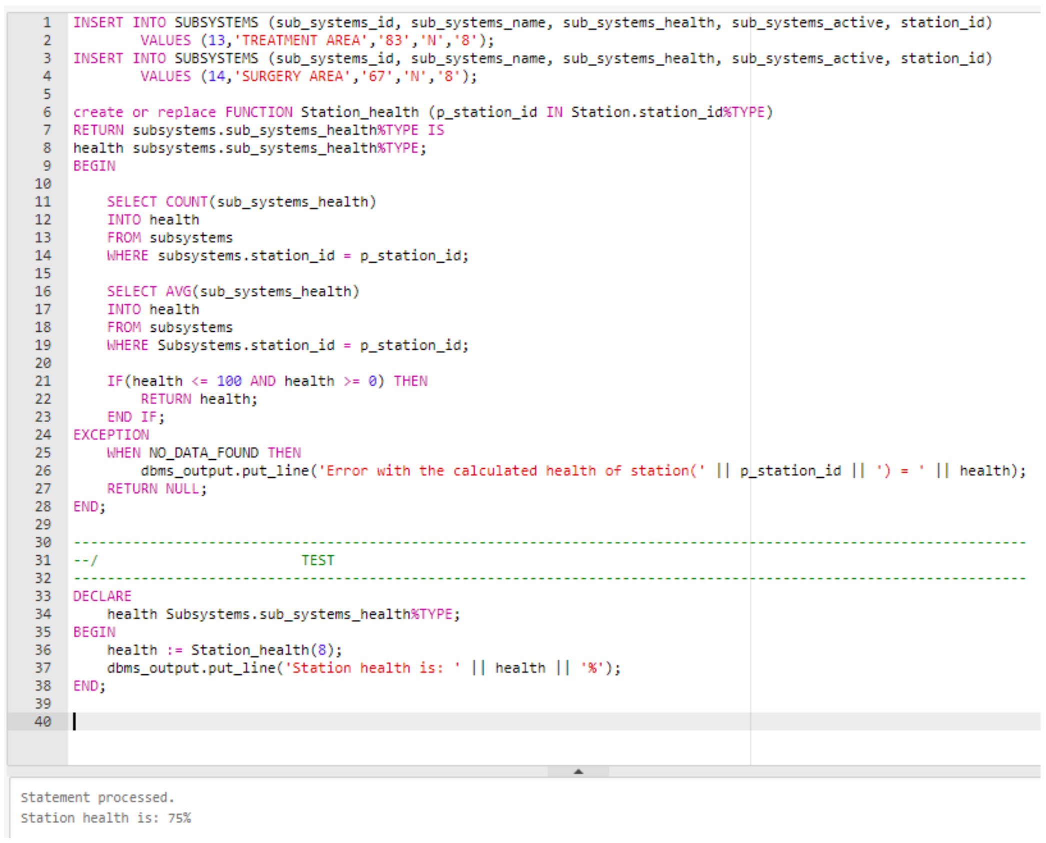
Figure 8: Function for returning age of a specific crew member. Involves calculating the months between the DoB and the current date set for the operating system on which the database resides.



```

INSERT INTO CREW(crew_id, first_name, last_name, dob, crew_station_id, role_name, crew_rank_id, a
        VALUES(1,NULL,NULL,(to_date('27/06/1996','dd/mm/yyyy')),NULL, NULL, NULL, NULL);

CREATE OR REPLACE FUNCTION age_retrieval (c_id NUMBER)
RETURN NUMBER IS
age NUMBER;
v_dob DATE;
BEGIN
   SELECT dob into v_dob from CREW
   WHERE crew_id = c_id;
   age := trunc(months_between(sysdate,v_dob)/12);
   RETURN age;
END;

DECLARE
age1 NUMBER;
BEGIN
 age1 := age_retrieval(1);
 dbms_output.put_line(age1 ||' YEARS');
END;
```

```
Statement processed.
22 YEARS
```

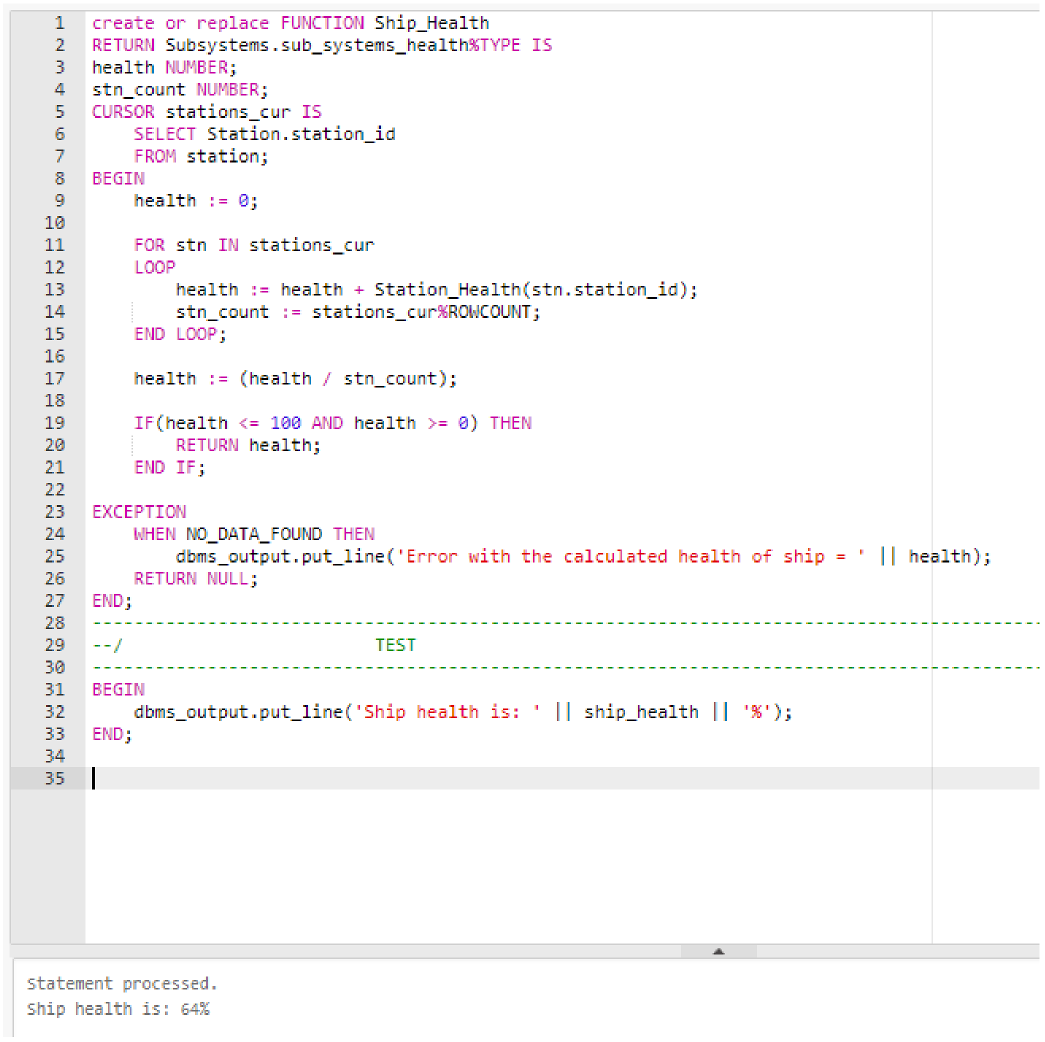
Figure 9: Age returned for a crew member whose DoB is **'27/06/1996'**.



```
INSERT INTO SUBSYSTEMS (sub_systems_id, sub_systems_name, sub_systems_health, sub_systems_active, station_id)
        VALUES (13,'TREATMENT AREA','83','N','8');
INSERT INTO SUBSYSTEMS (sub_systems_id, sub_systems_name, sub_systems_health, sub_systems_active, station_id)
        VALUES (14,'SURGERY AREA','67','N','8');

create or replace FUNCTION Station_health (p_station_id IN Station.station_id%TYPE)
RETURN subsystems.sub_systems_health%TYPE IS
health subsystems.sub_systems_health%TYPE;
BEGIN

    SELECT COUNT(sub_systems_health)
    INTO health
    FROM subsystems
    WHERE subsystems.station_id = p_station_id;

    SELECT AVG(sub_systems_health)
    INTO health
    FROM subsystems
    WHERE Subsystems.station_id = p_station_id;

    IF(health <= 100 AND health >= 0) THEN
        RETURN health;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Error with the calculated health of station(' || p_station_id || ') = ' || health);
    RETURN NULL;
END;

------------------------------------------------------------------------------------------------------------------------
--/                        TEST
------------------------------------------------------------------------------------------------------------------------
DECLARE
    health Subsystems.sub_systems_health%TYPE;
BEGIN
    health := Station_health(8);
    dbms_output.put_line('Station health is: ' || health || '%');
END;

```

```
Statement processed.
Station health is: 75%
```

Figure 10: Function for calculating the health of a station which then can ultimately lead to the calculation of the overall health of the ship. To know the health status of the ship is necessary in order to know when to use certain subsystems to prevent catastrophic measures.



```
create or replace FUNCTION Ship_Health
RETURN Subsystems.sub_systems_health%TYPE IS
health NUMBER;
stn_count NUMBER;
CURSOR stations_cur IS
    SELECT Station.station_id
    FROM station;
BEGIN
    health := 0;

    FOR stn IN stations_cur
    LOOP
        health := health + Station_Health(stn.station_id);
        stn_count := stations_cur%ROWCOUNT;
    END LOOP;

    health := (health / stn_count);

    IF(health <= 100 AND health >= 0) THEN
        RETURN health;
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Error with the calculated health of ship = ' || health);
    RETURN NULL;
END;
-------------------------------------------------------------------------------------------
--/                          TEST
-------------------------------------------------------------------------------------------
BEGIN
    dbms_output.put_line('Ship health is: ' || ship_health || '%');
END;

```

```
Statement processed.
Ship health is: 64%
```

Figure 11: Calculation of the overall health of the ship by taking the average of the health of all stations.

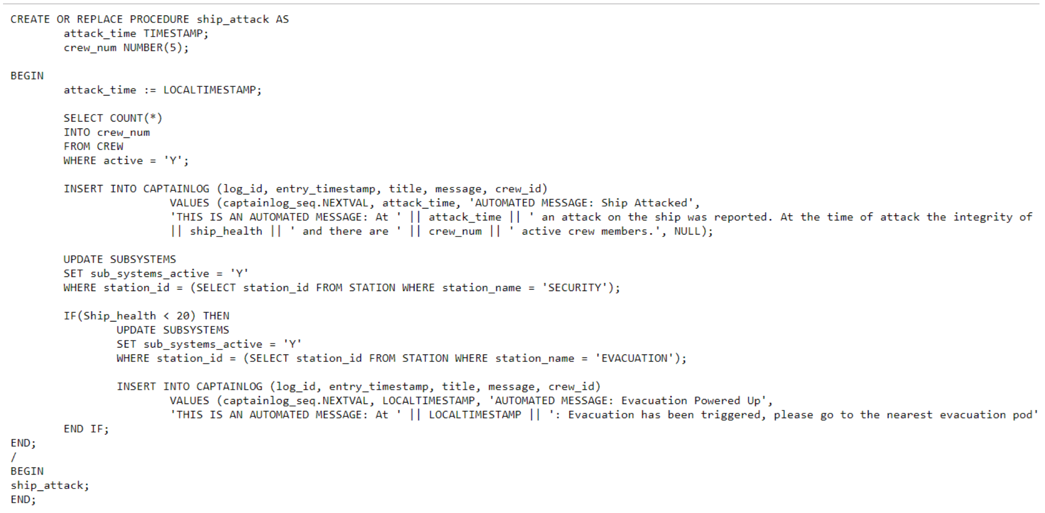
5 - Procedures

Similar to functions, procedures are called from the main program to do a specific task. However, unlike functions, they do not return a value. Instead, procedures will be used to directly affect the database in order to adapt to the external changes that occur.

External changes will occur which then needs to be documented in the database. Therefore the database needs to change to adapt to the following possible scenarios:

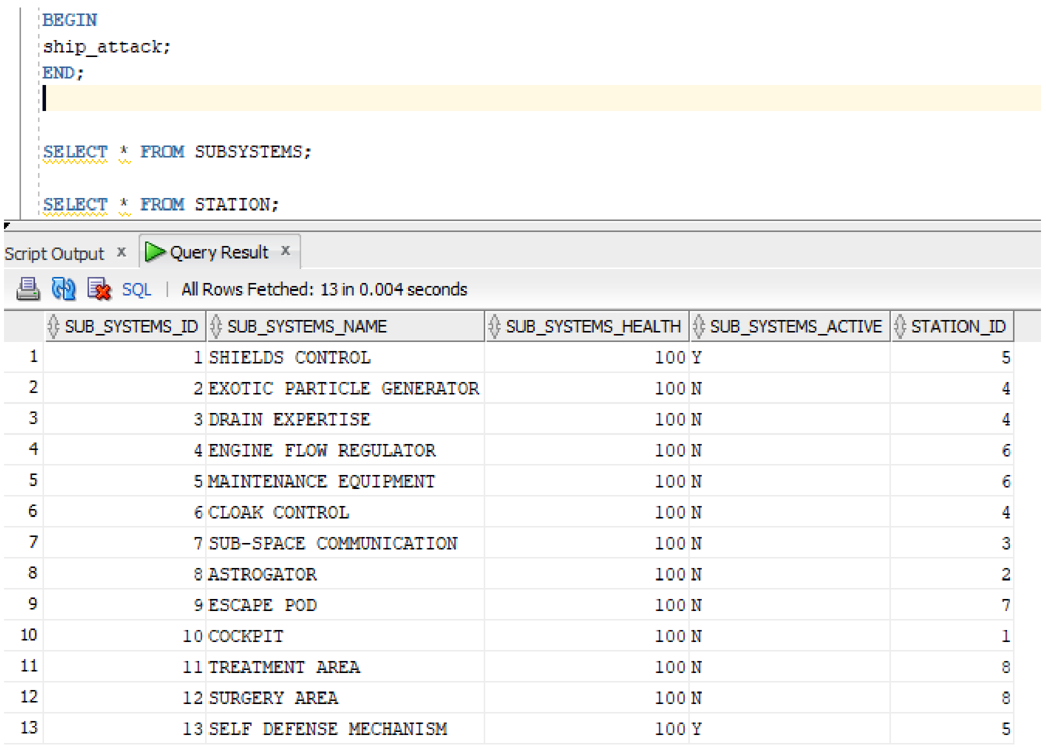
- An attack on the spaceship.

- Beaming a crew member out or in the spaceship.



```
CREATE OR REPLACE PROCEDURE ship_attack AS
        attack_time TIMESTAMP;
        crew_num NUMBER(5);

BEGIN
        attack_time := LOCALTIMESTAMP;

        SELECT COUNT(*)
        INTO crew_num
        FROM CREW
        WHERE active = 'Y';

        INSERT INTO CAPTAINLOG (log_id, entry_timestamp, title, message, crew_id)
                        VALUES (captainlog_seq.NEXTVAL, attack_time, 'AUTOMATED MESSAGE: Ship Attacked',
                        'THIS IS AN AUTOMATED MESSAGE: At ' || attack_time || ' an attack on the ship was reported. At the time of attack the integrity of
                        || ship_health || ' and there are ' || crew_num || ' active crew members.', NULL);

        UPDATE SUBSYSTEMS
        SET sub_systems_active = 'Y'
        WHERE station_id = (SELECT station_id FROM STATION WHERE station_name = 'SECURITY');

        IF(Ship_health < 20) THEN
                UPDATE SUBSYSTEMS
                SET sub_systems_active = 'Y'
                WHERE station_id = (SELECT station_id FROM STATION WHERE station_name = 'EVACUATION');

                INSERT INTO CAPTAINLOG (log_id, entry_timestamp, title, message, crew_id)
                        VALUES (captainlog_seq.NEXTVAL, LOCALTIMESTAMP, 'AUTOMATED MESSAGE: Evacuation Powered Up',
                        'THIS IS AN AUTOMATED MESSAGE: At ' || LOCALTIMESTAMP || ': Evacuation has been triggered, please go to the nearest evacuation pod'
        END IF;
END;
/
BEGIN
ship_attack;
END;
```

Figure 12: The procedure that takes place in case of an attack on the spaceship. The procedure inserts a message into the captain log and turns the defense sub systems active. It also checks the health of the ship and ensures evacuation protocols take place once the health goes below a certain percent

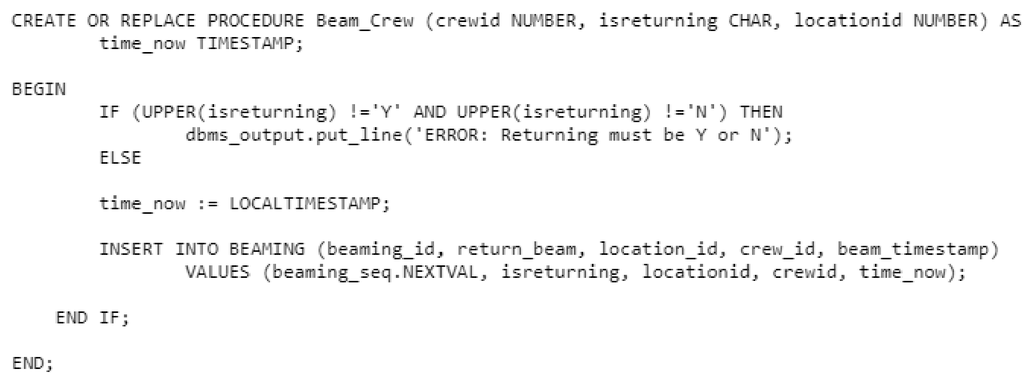


```
BEGIN
ship_attack;
END;


SELECT * FROM SUBSYSTEMS;

SELECT * FROM STATION;
```

All Rows Fetched: 13 in 0.004 seconds

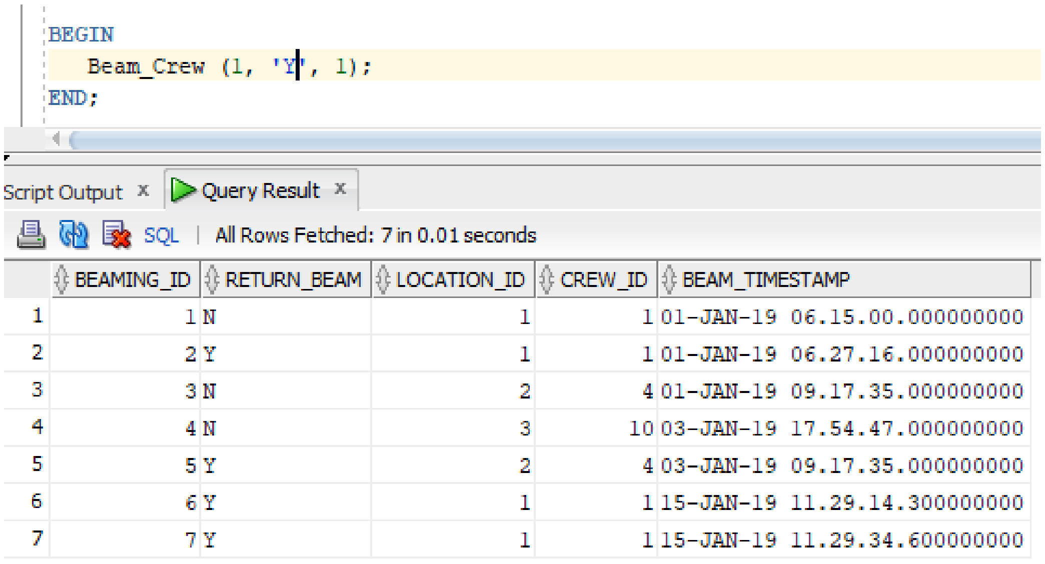
	SUB_SYSTEMS_ID	SUB_SYSTEMS_NAME	SUB_SYSTEMS_HEALTH	SUB_SYSTEMS_ACTIVE	STATION_ID
1	1	SHIELDS CONTROL	100	Y	5
2	2	EXOTIC PARTICLE GENERATOR	100	N	4
3	3	DRAIN EXPERTISE	100	N	4
4	4	ENGINE FLOW REGULATOR	100	N	6
5	5	MAINTENANCE EQUIPMENT	100	N	6
6	6	CLOAK CONTROL	100	N	4
7	7	SUB-SPACE COMMUNICATION	100	N	3
8	8	ASTROGATOR	100	N	2
9	9	ESCAPE POD	100	N	7
10	10	COCKPIT	100	N	1
11	11	TREATMENT AREA	100	N	8
12	12	SURGERY AREA	100	N	8
13	13	SELF DEFENSE MECHANISM	100	Y	5

Figure 13: When tested, the subsystens ’Shield Control’ and ’Self Defense Mechanism’ are activated as shown above.



```
CREATE OR REPLACE PROCEDURE Beam_Crew (crewid NUMBER, isreturning CHAR, locationid NUMBER) AS
        time_now TIMESTAMP;

BEGIN
        IF (UPPER(isreturning) !='Y' AND UPPER(isreturning) !='N') THEN
                dbms_output.put_line('ERROR: Returning must be Y or N');
        ELSE

        time_now := LOCALTIMESTAMP;

        INSERT INTO BEAMING (beaming_id, return_beam, location_id, crew_id, beam_timestamp)
                VALUES (beaming_seq.NEXTVAL, isreturning, locationid, crewid, time_now);

    END IF;

END;
```

Figure 14: Script for creating the procedure to beam member out or in the spaceship.



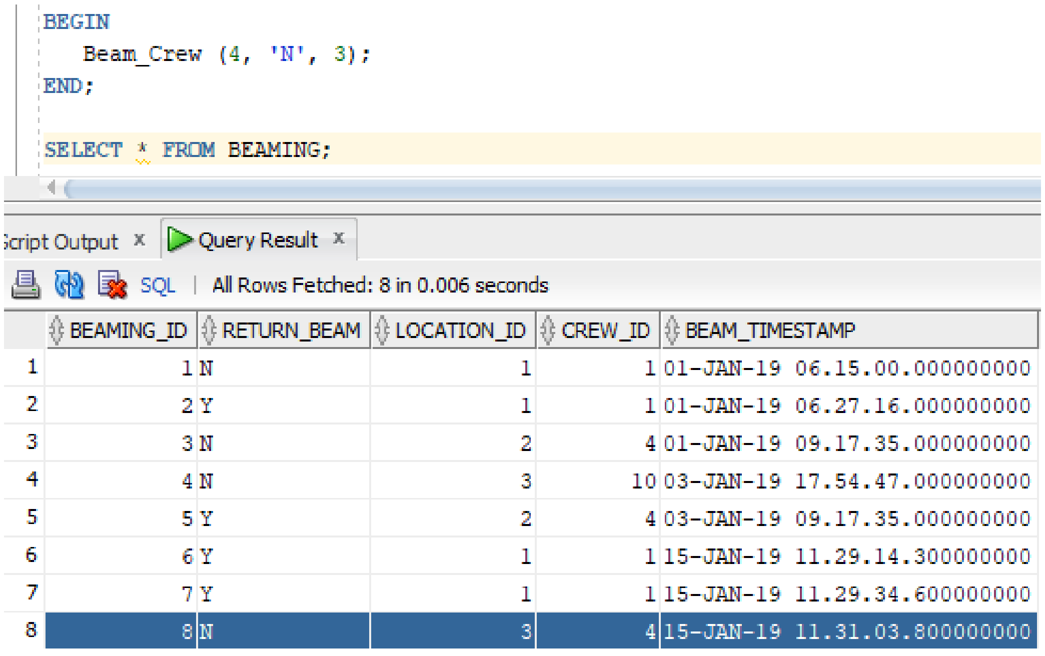
```
BEGIN
   Beam_Crew (1, 'Y', 1);
END;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 7 in 0.01 seconds

	BEAMING_ID	RETURN_BEAM	LOCATION_ID	CREW_ID	BEAM_TIMESTAMP
1	1	N	1	1	01-JAN-19 06.15.00.000000000
2	2	Y	1	1	01-JAN-19 06.27.16.000000000
3	3	N	2	4	01-JAN-19 09.17.35.000000000
4	4	N	3	10	03-JAN-19 17.54.47.000000000
5	5	Y	2	4	03-JAN-19 09.17.35.000000000
6	6	Y	1	1	15-JAN-19 11.29.14.300000000
7	7	Y	1	1	15-JAN-19 11.29.34.600000000

Figure 15: Test 1.



```
BEGIN
   Beam_Crew (4, 'N', 3);
END;

SELECT * FROM BEAMING;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 8 in 0.006 seconds

	BEAMING_ID	RETURN_BEAM	LOCATION_ID	CREW_ID	BEAM_TIMESTAMP
1	1	N	1	1	01-JAN-19 06.15.00.000000000
2	2	Y	1	1	01-JAN-19 06.27.16.000000000
3	3	N	2	4	01-JAN-19 09.17.35.000000000
4	4	N	3	10	03-JAN-19 17.54.47.000000000
5	5	Y	2	4	03-JAN-19 09.17.35.000000000
6	6	Y	1	1	15-JAN-19 11.29.14.300000000
7	7	Y	1	1	15-JAN-19 11.29.34.600000000
8	8	N	3	4	15-JAN-19 11.31.03.800000000

Figure 16: Test 2.

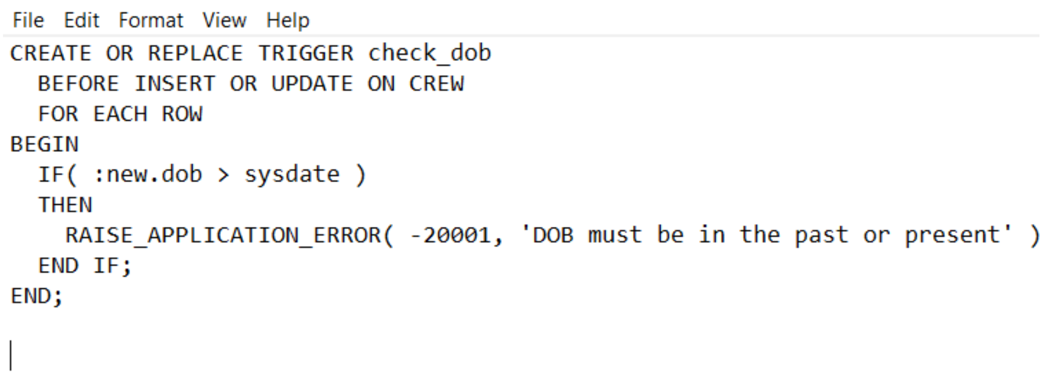
6 - Triggers

Triggers similar to functions and procedures, do a task according to how they are logically defined. Once executed, they are stored in the database and are automatically executed when some events occur. In this case, the triggers implemented for the USS Enterprise database are designed to respond to database manipulation statements DELETE, INSERT or UPDATE.

The main purpose of the triggers that will be implemented are simply for data integrity. When a certain amount of data is being inputted or updated within a database, it is ideal to check if the data manipulation taking place coincides with the underlying business requirements, thus preventing a lot of unwanted data entries. As a demonstration, triggers with the following functions will be implemented as to showcase their relevance to an overall efficient database structure:

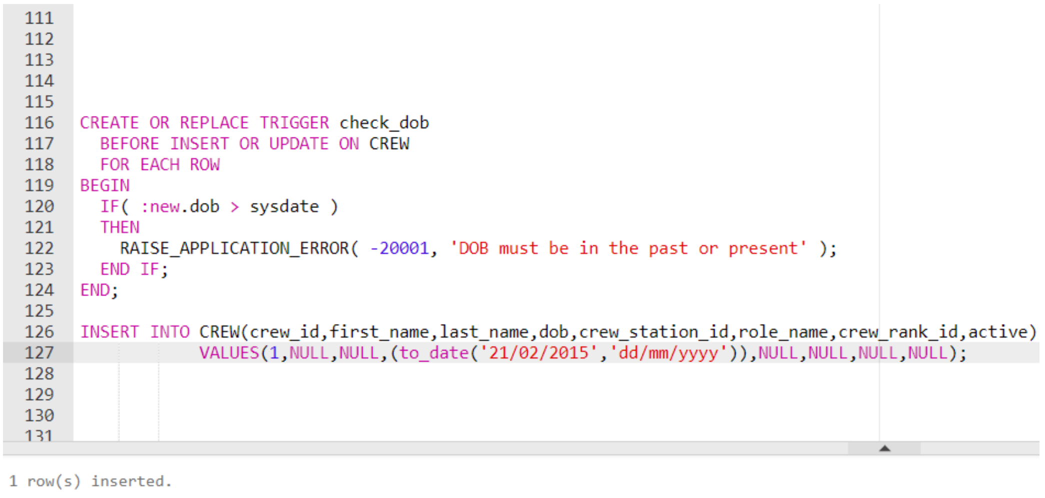
- **Checking crew member's DoB is not in the future.**
- **Check crew's rank authority level when they are listed as author in Captain's Log.**

An unnecessary inaccuracy would be entering any date describing the future as a date of birth for any crew member. Considering values can be inserted, updated or deleted within a database, both trigger implementations will consist of an 'INSERT or UPDATE' statement. Following are images showcasing the PL/SQL scripts necessary to implement the triggers:



```
CREATE OR REPLACE TRIGGER check_dob
  BEFORE INSERT OR UPDATE ON CREW
  FOR EACH ROW
BEGIN
  IF( :new.dob > sysdate )
  THEN
    RAISE_APPLICATION_ERROR( -20001, 'DOB must be in the past or present' )
  END IF;
END;
```

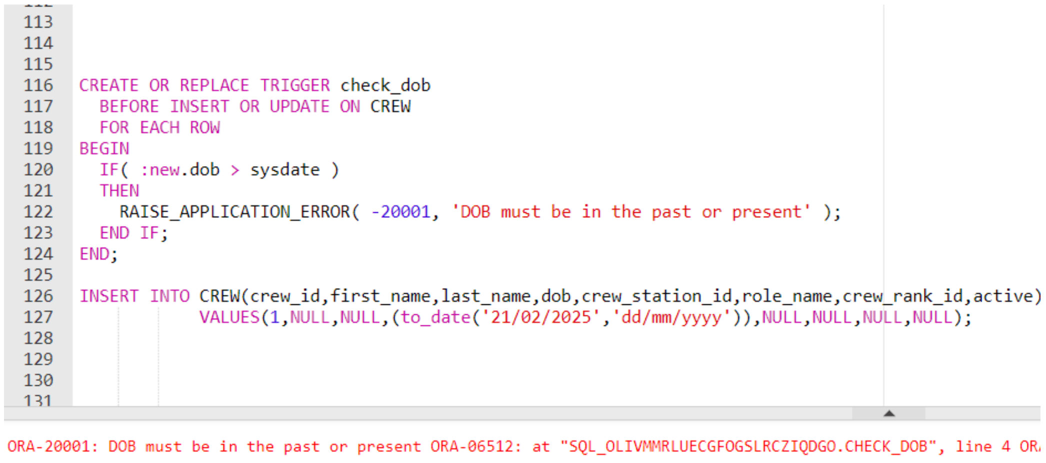
Figure 17: Script for creating the trigger to check DOB of value before entry.



```
CREATE OR REPLACE TRIGGER check_dob
  BEFORE INSERT OR UPDATE ON CREW
  FOR EACH ROW
BEGIN
  IF( :new.dob > sysdate )
  THEN
    RAISE_APPLICATION_ERROR( -20001, 'DOB must be in the past or present' );
  END IF;
END;

INSERT INTO CREW(crew_id,first_name,last_name,dob,crew_station_id,role_name,crew_rank_id,active)
          VALUES(1,NULL,NULL,(to_date('21/02/2015','dd/mm/yyyy')),NULL,NULL,NULL,NULL);
```

```
1 row(s) inserted.
```

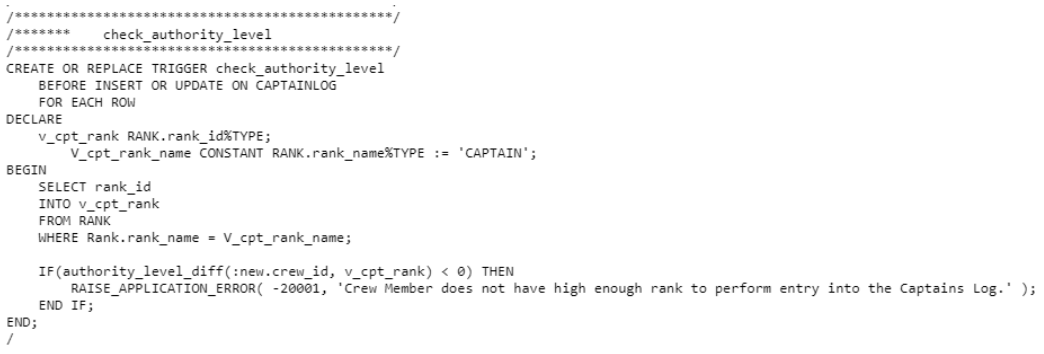
Figure 18: DoB is in the past. Trigger is not executed. Data is accepted.



```
CREATE OR REPLACE TRIGGER check_dob
  BEFORE INSERT OR UPDATE ON CREW
  FOR EACH ROW
BEGIN
  IF( :new.dob > sysdate )
  THEN
    RAISE_APPLICATION_ERROR( -20001, 'DOB must be in the past or present' );
  END IF;
END;

INSERT INTO CREW(crew_id,first_name,last_name,dob,crew_station_id,role_name,crew_rank_id,active)
            VALUES(1,NULL,NULL,(to_date('21/02/2025','dd/mm/yyyy')),NULL,NULL,NULL,NULL);
```

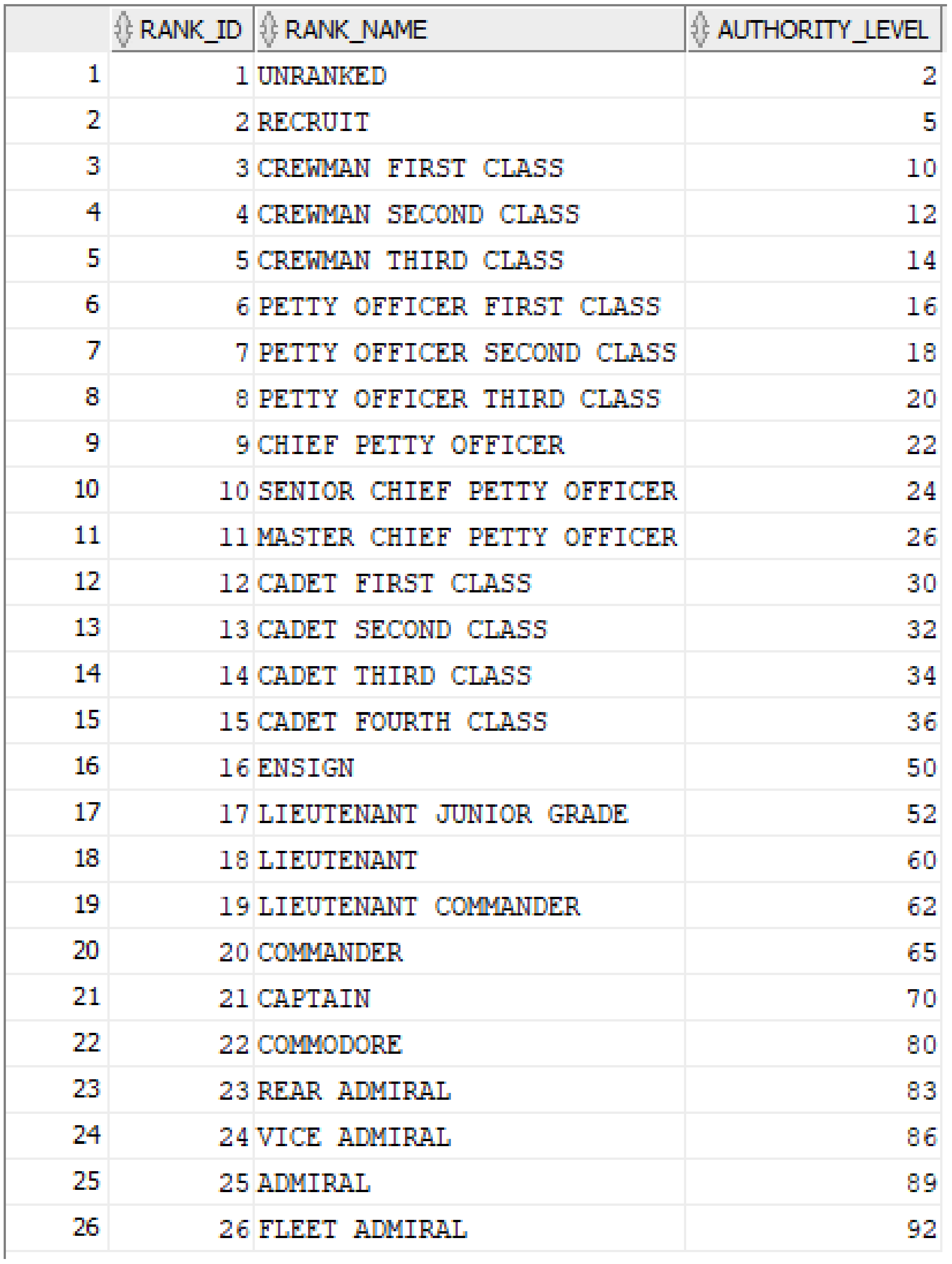
```
ORA-20001: DOB must be in the past or present ORA-06512: at "SQL_OLIVMMRLUECGFOGSLRCZIQDGO.CHECK_DOB", line 4 OR
```

Figure 19: DoB is in the future. Trigger is executed. Data is rejected.



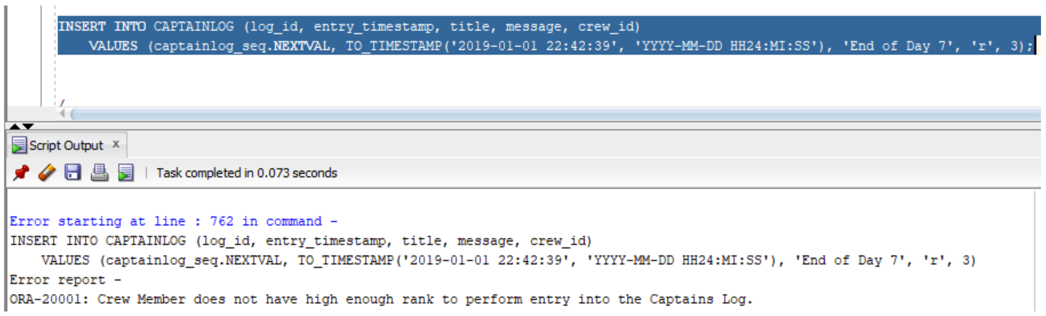
```
/**********************************************/
/*******    check_authority_level
/**********************************************/
CREATE OR REPLACE TRIGGER check_authority_level
    BEFORE INSERT OR UPDATE ON CAPTAINLOG
    FOR EACH ROW
DECLARE
    v_cpt_rank RANK.rank_id%TYPE;
        V_cpt_rank_name CONSTANT RANK.rank_name%TYPE := 'CAPTAIN';
BEGIN
    SELECT rank_id
    INTO v_cpt_rank
    FROM RANK
    WHERE Rank.rank_name = V_cpt_rank_name;

    IF(authority_level_diff(:new.crew_id, v_cpt_rank) < 0) THEN
        RAISE_APPLICATION_ERROR( -20001, 'Crew Member does not have high enough rank to perform entry into the Captains Log.' );
    END IF;
END;
/
```

Figure 20: Script for implementing trigger that checks the authority level of crew member before being listed as author in Captain's Log.



	RANK_ID	RANK_NAME	AUTHORITY_LEVEL
1	1	UNRANKED	2
2	2	RECRUIT	5
3	3	CREWMAN FIRST CLASS	10
4	4	CREWMAN SECOND CLASS	12
5	5	CREWMAN THIRD CLASS	14
6	6	PETTY OFFICER FIRST CLASS	16
7	7	PETTY OFFICER SECOND CLASS	18
8	8	PETTY OFFICER THIRD CLASS	20
9	9	CHIEF PETTY OFFICER	22
10	10	SENIOR CHIEF PETTY OFFICER	24
11	11	MASTER CHIEF PETTY OFFICER	26
12	12	CADET FIRST CLASS	30
13	13	CADET SECOND CLASS	32
14	14	CADET THIRD CLASS	34
15	15	CADET FOURTH CLASS	36
16	16	ENSIGN	50
17	17	LIEUTENANT JUNIOR GRADE	52
18	18	LIEUTENANT	60
19	19	LIEUTENANT COMMANDER	62
20	20	COMMANDER	65
21	21	CAPTAIN	70
22	22	COMMODORE	80
23	23	REAR ADMIRAL	83
24	24	VICE ADMIRAL	86
25	25	ADMIRAL	89
26	26	FLEET ADMIRAL	92

Figure 21: The rank names and authority levels



```
INSERT INTO CAPTAINLOG (log_id, entry_timestamp, title, message, crew_id)
    VALUES (captainlog_seq.NEXTVAL, TO_TIMESTAMP('2019-01-01 22:42:39', 'YYYY-MM-DD HH24:MI:SS'), 'End of Day 7', 'r', 3);
```

```
Error starting at line : 762 in command -
INSERT INTO CAPTAINLOG (log_id, entry_timestamp, title, message, crew_id)
    VALUES (captainlog_seq.NEXTVAL, TO_TIMESTAMP('2019-01-01 22:42:39', 'YYYY-MM-DD HH24:MI:SS'), 'End of Day 7', 'r', 3)
Error report -
ORA-20001: Crew Member does not have high enough rank to perform entry into the Captains Log.
```

Figure 22: Testing the trigger by attempting to enter value into Captain Log with a crew ID of 3 which is not authorizable

7 - Access Controls and Permissions

[h] The crew member roles chosen for testing the database access control are as follows:

- Captain.

- Beam Operator.

- Personnel Manager.

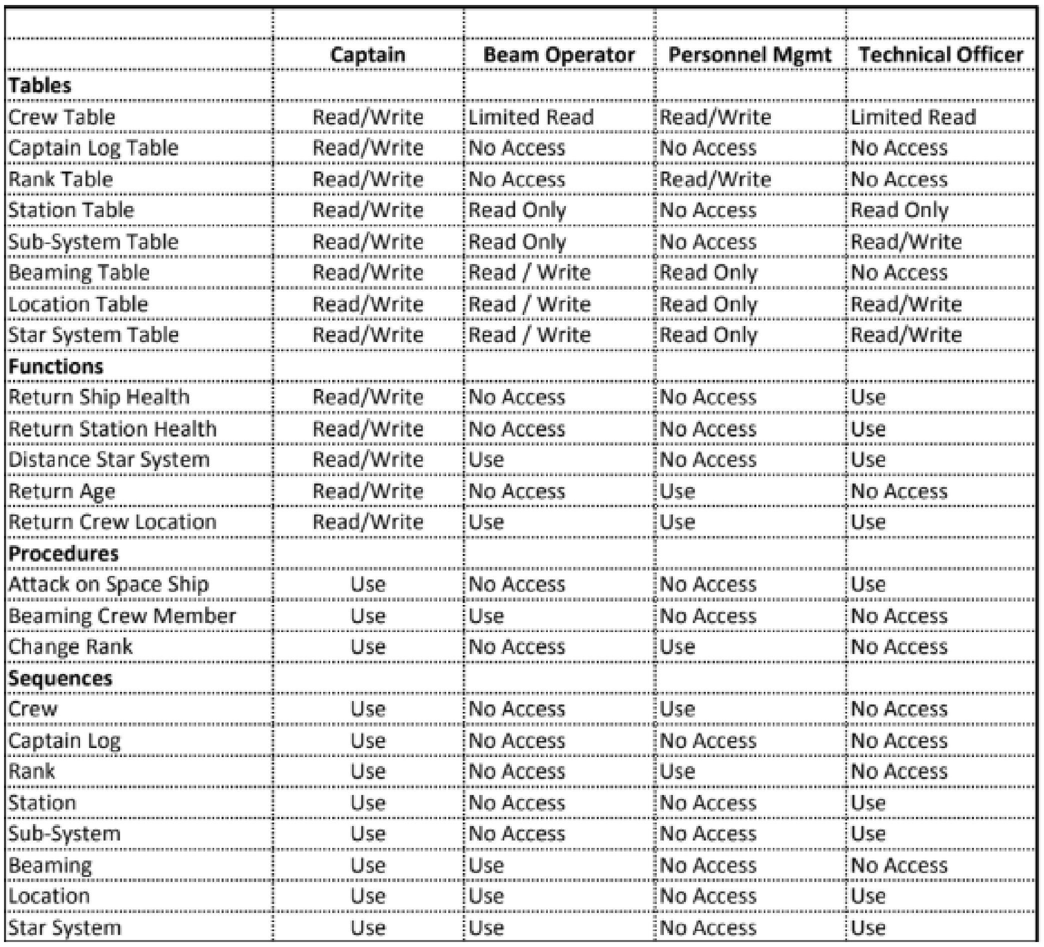
- Technical Officer.

The Captain has read/write access to all database objects. It would be justifiable that at least one crew member has this kind of access and the captain would be the main crew member in contention. The captain can access the captain's log and call any stored function or procedure at his/her disposal.

The beam operator is resposible for ensuring successful beam transfers and returns. Limited access is assigned to the beam operator but full access to the Beaming, Location and Star System. The reason for this is that the beam operator will need access to the star system and location data as to ensure successful transfer from or to the ship as well as adding records of transfers and returns once they are done in the Beam Log.

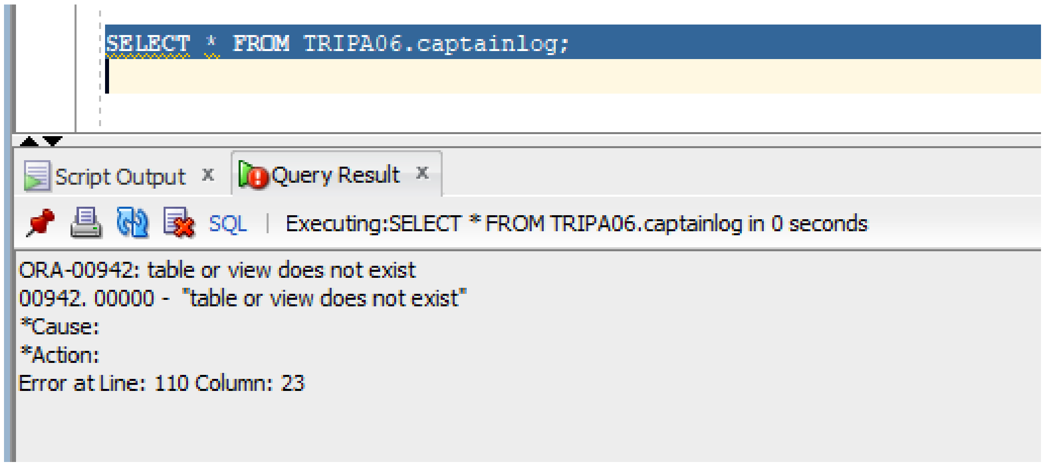
The personnel management are the sort of 'HR' of the USS Enterprise. The purpose of this role is to maximize employee organisation within the spaceship. To do this, the role needs access to crew member data and their whereabouts as well as other data that may be transitively involved.

The technical officer is usually an engineer or a versatile practicioner of many fields. The role involves maintaning the different usage of subsytems such as Weapons, Warp Drive, Space Navigation Astrogator and so on. This role would require access to the sub system data as well as star system and location data. In this report, the testing will show the user '4caucj91' (Jonathan Cauchi) being granted the role of captain and permission to access the database monitored by 'tripa06'(Alex Tripp). The figures below showcase the permissions for the specified roles as well as the access control testing undergone for the Captain role:



	Captain	Beam Operator	Personnel Mgmt	Technical Officer
Tables				
Crew Table	Read/Write	Limited Read	Read/Write	Limited Read
Captain Log Table	Read/Write	No Access	No Access	No Access
Rank Table	Read/Write	No Access	Read/Write	No Access
Station Table	Read/Write	Read Only	No Access	Read Only
Sub-System Table	Read/Write	Read Only	No Access	Read/Write
Beaming Table	Read/Write	Read / Write	Read Only	No Access
Location Table	Read/Write	Read / Write	Read Only	Read/Write
Star System Table	Read/Write	Read / Write	Read Only	Read/Write
Functions				
Return Ship Health	Read/Write	No Access	No Access	Use
Return Station Health	Read/Write	No Access	No Access	Use
Distance Star System	Read/Write	Use	No Access	Use
Return Age	Read/Write	No Access	Use	No Access
Return Crew Location	Read/Write	Use	Use	Use
Procedures				
Attack on Space Ship	Use	No Access	No Access	Use
Beaming Crew Member	Use	Use	No Access	No Access
Change Rank	Use	No Access	Use	No Access
Sequences				
Crew	Use	No Access	Use	No Access
Captain Log	Use	No Access	No Access	No Access
Rank	Use	No Access	Use	No Access
Station	Use	No Access	No Access	Use
Sub-System	Use	No Access	No Access	Use
Beaming	Use	Use	No Access	No Access
Location	Use	Use	No Access	Use
Star System	Use	Use	No Access	Use

Figure 23: Role permission breakdown for Access Control.



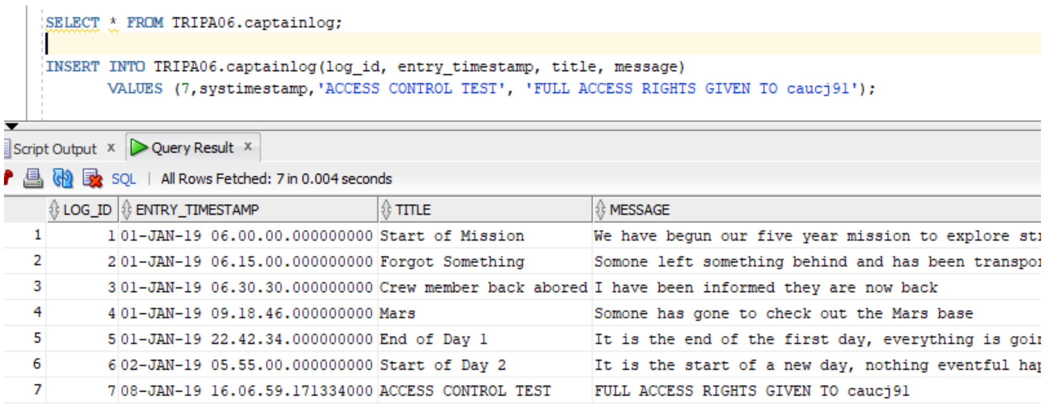
```
SELECT * FROM TRIPA06.captainlog;
```

Script Output | Query Result

SQL | Executing:SELECT * FROM TRIPA06.captainlog in 0 seconds

```
ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
Error at Line: 110 Column: 23
```

Figure 24: Attempt to view table data before permission grant.



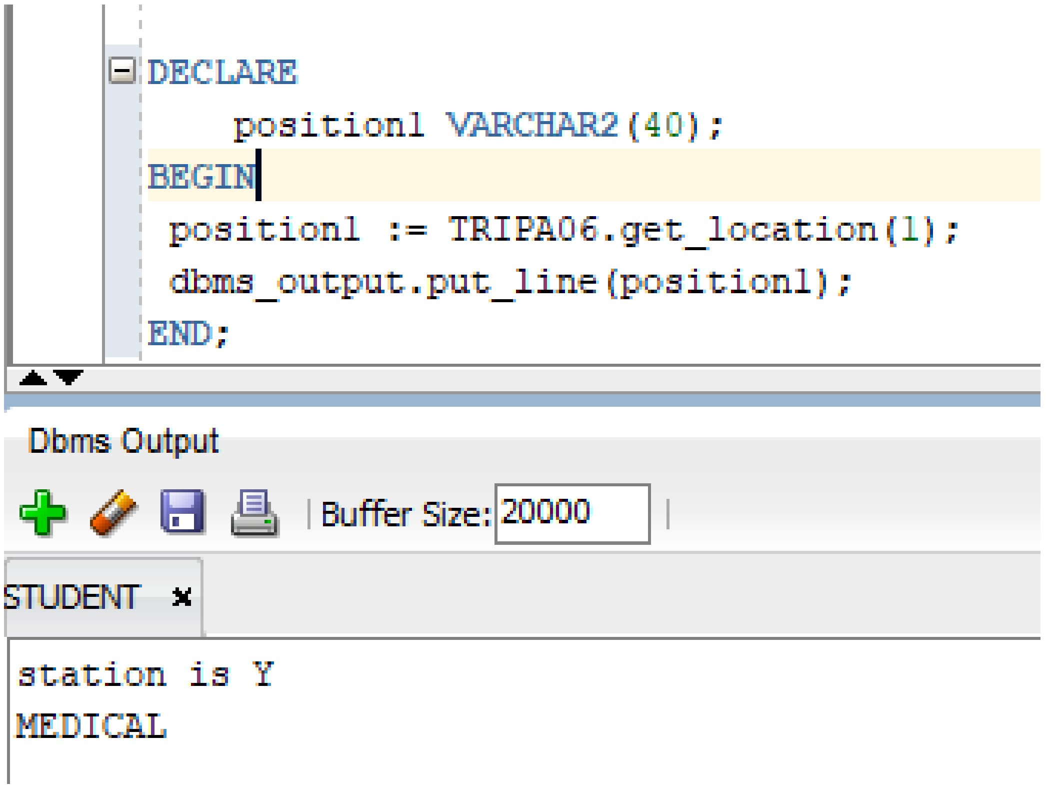
```
SELECT * FROM TRIPA06.captainlog;

INSERT INTO TRIPA06.captainlog(log_id, entry_timestamp, title, message)
       VALUES (7,systimestamp,'ACCESS CONTROL TEST', 'FULL ACCESS RIGHTS GIVEN TO caucj91');
```

Script Output | Query Result

SQL | All Rows Fetched: 7 in 0.004 seconds

	LOG_ID	ENTRY_TIMESTAMP	TITLE	MESSAGE
1	1	01-JAN-19 06.00.00.000000000	Start of Mission	We have begun our five year mission to explore st
2	2	01-JAN-19 06.15.00.000000000	Forgot Something	Somone left something behind and has been transpo
3	3	01-JAN-19 06.30.30.000000000	Crew member back abored	I have been informed they are now back
4	4	01-JAN-19 09.18.46.000000000	Mars	Somone has gone to check out the Mars base
5	5	01-JAN-19 22.42.34.000000000	End of Day 1	It is the end of the first day, everything is goi
6	6	02-JAN-19 05.55.00.000000000	Start of Day 2	It is the start of a new day, nothing eventful ha
7	7	08-JAN-19 16.06.59.171334000	ACCESS CONTROL TEST	FULL ACCESS RIGHTS GIVEN TO caucj91

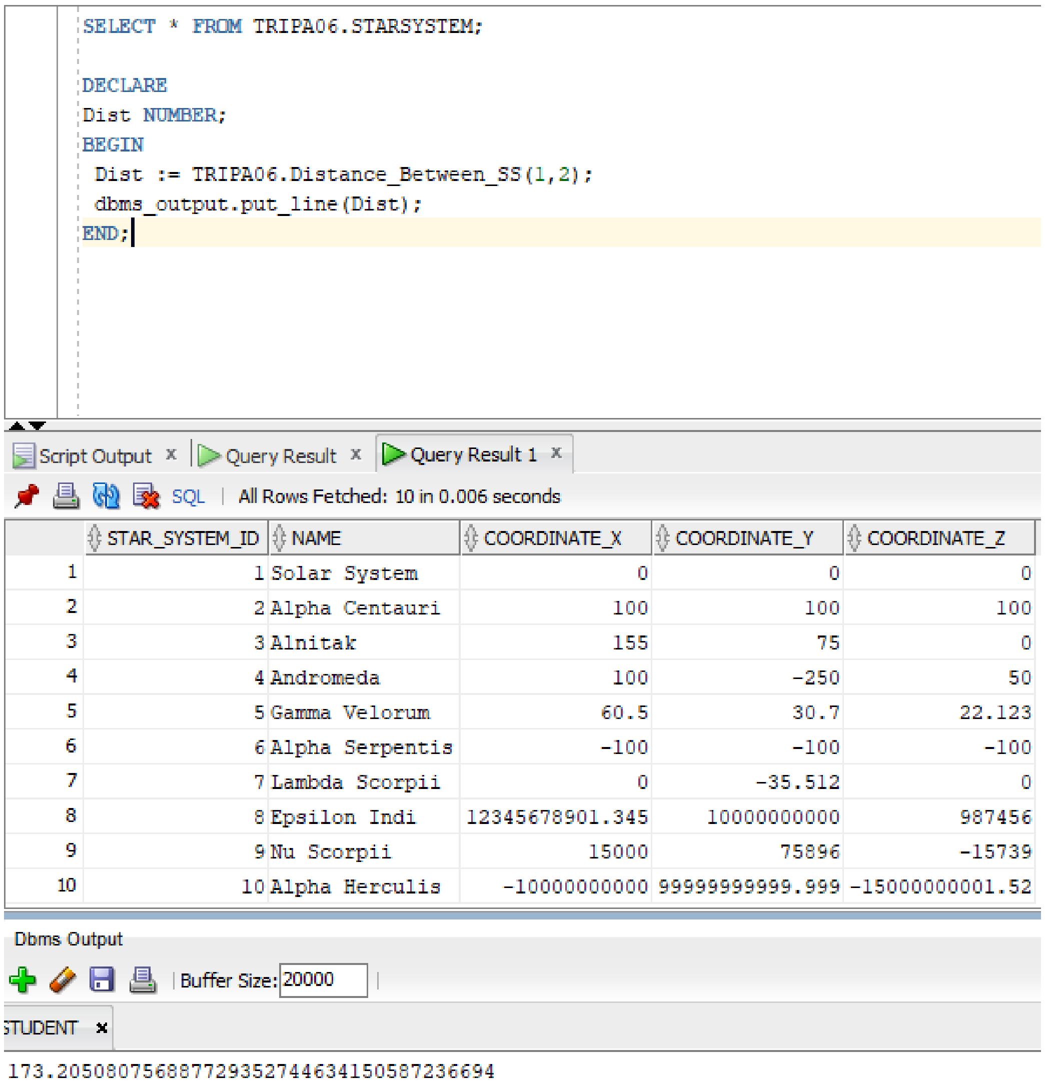
Figure 25: Successful viewing and insertion of data in Captain Log after permission grant. One thing to note is that only the Captain has access to the Captain's Log.



```
DECLARE
    position1 VARCHAR2(40);
BEGIN
 position1 := TRIPA06.get_location(1);
 dbms_output.put_line(position1);
END;
```

```
station is Y
MEDICAL
```

Figure 26: Successful usage of a function that searches the database for a location or station where a certain crew member is residing.



```
SELECT * FROM TRIPA06.STARSYSTEM;

DECLARE
Dist NUMBER;
BEGIN
 Dist := TRIPA06.Distance_Between_SS(1,2);
 dbms_output.put_line(Dist);
END;
```

Script Output × | Query Result × | Query Result 1 ×

SQL | All Rows Fetched: 10 in 0.006 seconds

	STAR_SYSTEM_ID	NAME	COORDINATE_X	COORDINATE_Y	COORDINATE_Z
1	1	Solar System	0	0	0
2	2	Alpha Centauri	100	100	100
3	3	Alnitak	155	75	0
4	4	Andromeda	100	-250	50
5	5	Gamma Velorum	60.5	30.7	22.123
6	6	Alpha Serpentis	-100	-100	-100
7	7	Lambda Scorpii	0	-35.512	0
8	8	Epsilon Indi	12345678901.345	10000000000	987456
9	9	Nu Scorpii	15000	75896	-15739
10	10	Alpha Herculis	-10000000000	99999999999.999	-15000000001.52

Dbms Output

Buffer Size: 20000

STUDENT ×

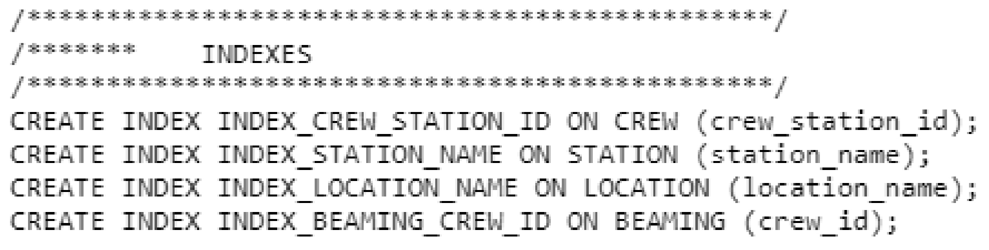
```
173.205080756887729352744634150587236694
```

Figure 27: Allowed access to view the data regarding star systems and to execute the function to calculate the distance between two particular star systems using their ID.

8 - Performance Enhancement

Without indexes, the database would have to scan an entire section of data just to retrieve a small chunk of information. It is ideal to have indexes that allow us to go directly to the section of data we require to view.

After assessing the business processes of the database, it was concluded that the indexes will be implemented in the four more populous tables which are CREW, STATION, LOCATION and BEAMING as shown below:



```
/***********************************************/
/*******    INDEXES
/***********************************************/
CREATE INDEX INDEX_CREW_STATION_ID ON CREW (crew_station_id);
CREATE INDEX INDEX_STATION_NAME ON STATION (station_name);
CREATE INDEX INDEX_LOCATION_NAME ON LOCATION (location_name);
CREATE INDEX INDEX_BEAMING_CREW_ID ON BEAMING (crew_id);
```

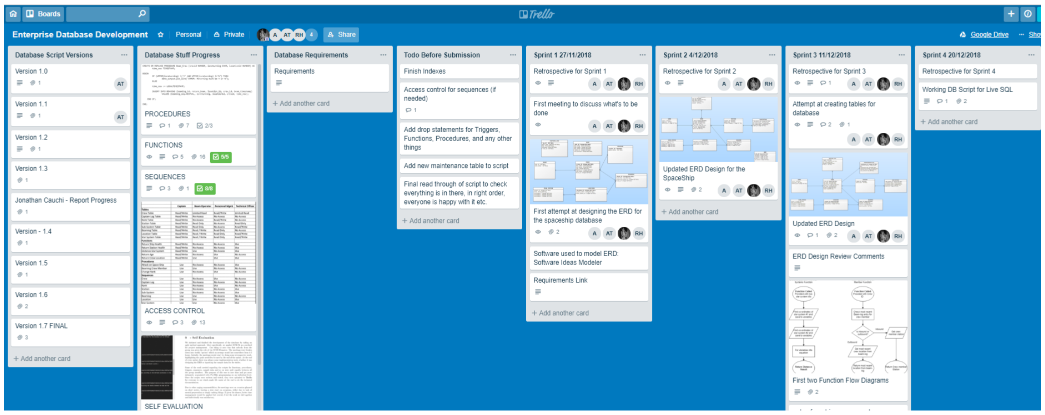
Figure 28: The indexes implemented.

9 - Self Evaluation

We initiated and finished the development of the database by taking an agile method approach. More specifically, we applied SCRUM as a method for project management. One thing to note was that nobody from the group was given the role of the SCRUM master. The meetings were broken down into weekly 'sprints' which on average would last somewhere from 4-5 hours. Initially, the meetings would start by doing some retrospective work, highlighting the goals needed to be met by the end of the sprint. At the end of every sprint, there was always some implementation work, whether it was designing the ERD or inputting the sample data for the tables.

Some of the work needed regarding the scripts for functions, procedures, triggers, sequences, sample data and so on were split equally between all the group members. The purpose of this was to save time and get more intimately acquainted with PL/SQL programming on an individual level. Once the scripts were written and tested, they were uploaded to **Trello** for everyone to see which made life easier at the end to do the technical documentation.

Due to other urging responsabilities, the meetings were on occasion planned on short notice, leaving a slow start on occasions, either due to lack of mental preperation or simply rushing things. If given the chance, better time management would be applied but overall, I feel the work we did together and individually was satisfactory.



Database Script Versions
- Version 1.0
- Version 1.1
- Version 1.2
- Version 1.3
- Jonathan Cauchi - Report Progress
- Version - 1.4
- Version 1.5
- Version 1.6
- Version 1.7 FINAL

Database Stuff Progress
- PROCEDURES
- FUNCTIONS
- SEQUENCES
- ACCESS CONTROL
- SELF EVALUATION

Database Requirements
- Requirements

Todo Before Submission
- Finish Indexes
- Access control for sequences (if needed)
- Add drop statements for Triggers, Functions, Procedures, and any other things
- Add new maintenance table to script
- Final read through of script to check everything is in there, in right order, everyone is happy with it etc.

Sprint 1 27/11/2018
- Retrospective for Sprint 1
- First meeting to discuss what's to be done
- First attempt at designing the ERD for the spaceship database
- Software used to model ERD: Software Ideas Modeler
- Requirements Link

Sprint 2 4/12/2018
- Retrospective for Sprint 2
- Updated ERD Design for the SpaceShip

Sprint 3 11/12/2018
- Retrospective for Sprint 3
- Attempt at creating tables for database
- Updated ERD Design
- ERD Design Review Comments
- First two Function Flow Diagrams

Sprint 4 20/12/2018
- Retrospective for Sprint 4
- Working DB Script for Live SQL

Figure 29: Trello notice board.