**Solent University**

**School of Media Arts and Technology**

BSc (Hons) **Software Engineering**

**Academic Year 2019-2020**

**Afonso Cabeçadas**

**Report Resit**

**Enterprise Database Development**

**Unit leader: Prins Butt September 2020**

**Table of Content**

Introduction……………………………………………………………………………………………………1

Elicit requirements and Analysis…………………………………………………………………..1

Logical Database Design………………………………………………………………………………..2

PL/SQL……………………………………………………………………………………………………………3

Sequences and Trigger.……………………………………………………………………….3

Procedure…………………………………………………………………………………………….5

Functions……………………………………………………………………………………………..5

Conclusion……………………………………………………………………………………………………….6

Appendix………………………………………………………………………………………………………….7

**Introduction**

The assessment for this unit is to create a database to fully/partly manage a spaceship (USS Enterprise) from Star Trek. At the start of the resit I had to consider what kind of database I was pursuing, a broader database or a more compact and concise database. Instead of pursuing a broader database trying to implement several different parts of management, I decided to create a Database more focused on crew member and their role and department inside the Enterprise.

Following this decision an ERD (Entity Relationship Diagram) was created as the skeleton of the database. What followed the ERD was the functional part of the software which includes several procedures, triggers, functions and some assistive technology to enhance data retrieval and insertion.

**Elicit requirements and Analysis**

During this phase of the resit I though about narrowing the scope of the database to avoid overcomplicating the project with a hight number of tables to work with. I decided to focus on the personnel management type of database. The database consists of:

**Crew members:** Maintaining record ranging from their name and role to what department they work in.

**Logs:** Created this be an interface where I could have both crew member and department information and facilitate retrieving information from the database.

**Species:** This table was created to complement and give crew members more personality and to add extra functionality to the database.

**Department:** Contains information of where the crew members work and the number of crew working is each department.

**Quarters:** Follows the same example of the table species and is used to complement table departments to make the database seem more realistic and to add extra functionality.

**Species audit:** Species audit was solely created to demonstrate a more in depth understanding of how databases work in general.

**Schema audit:** Similar to table species audit, this table was also created to demonstrate a broader understand of how to audit a database and to show extra functionality.

**Logical Database Design**

This task was the first technical task of the project, to create an ERD (Entity Relationship Diagram) I was a web tool called dbdiagram.io.

This tool allowed me to create an ERD, with a clean visual, using programming instead of just using visual tools to create the diagram. When creating a table, I had to specify which row was the primary key and connect said primary keys to foreign keys in different tables, if they exist.

This was a quick process, the ERD consists of seven tables, two of which are auditing tables.

The tables I created for this database are not extremely complex as they consist of 5 rows maximum. The data types I used in this ERD consisted mostly of Int (int is a numeric data type and maximum parameter size of 255), varchar (varchar is a string data type that contains numbers, letters and special characters) and date (this data type is used specifically to display date).

The schema table uses all these data types as well but uses a special type of statements that will be discussed further down on the report.

The crew table has id which is the primary key, name, speciesid as a foreign key of species table, access level and role.

The species table has an id that is the primary key, species name and species planet. The id from this table is connected to the table crew with a one to one relationship.

Tables logs has two foreign keys and they connect to both the crew table and department table on a 1 to many relationship. This table does not hold any unique rows or data because its purpose is to facilitate retrieving and aggregating data from tables to maintain a third form normalisation.

Table department contains an id, department name, number of employees and a foreign key that is connected to quarters.

Table quarters is serving the same purpose of table species and as an id and a name.

Species\_audit is an auditing table with the purpose of logging information on DML (Data Manipulation Language) operations done to the species table.

DML operations are the operations that we use to retrieve, insert, delete and update tables. The purpose of this table is to keep information on overtime operation done to the table to avoid any deleting, inserting or a updating on the data inside without any knowledge of it happening. This can also be used to backroll data if it got deleted or changed by mistake. When any data is manipulated on table species a log is also insert into the species audit that later can be displayed is new name, old name, the date and the type of operation.

Schema\_audit on the contrary logs information on what is done the schema of the database. When a table is created or dropped, this is considered a DDL (Data Definition Language) operation. This table was created with the purpose of preventing any changes done to the database without knowledge or consent.

This table is especially useful to log information on the creation of objects that complements other objects, such as triggers and other operations that are “concealed” inside other objects. Whenever any DDL operation is made a log is inserted into schema\_audit table containing the date, what kind of object was created, the name and operation.

**PL/SQL**

**Sequences and Triggers**

After creating the tables I created five sequences for the tables I wanted the ID to auto-increment every time a record was inserted into the table. To create a sequence I started the statement with CREATE SEQUENCE, to let the platform I was creating an object, followed by the name of the sequence.

I.e. CREATE SEQUENCE species\_sequence;

I created sequences for the following tables: species, crew, department, audit and schema. I only created sequences for these tables because they seemed the more volatile ones when it comes to data manipulation in them, this was I can make sure that every single entry has a unique ID and it is one less row that the user needs to insert into the table.

For the sequences to work I had to create triggers for each one of the sequences. To create a trigger, I started the statement with CREATE OR REPLACE TRIGGER followed by the name of the trigger.

I.e. CREATE OR REPLACE TRIGGER species\_insert;

I had to specify that before any data is inserted into the table species for each row the trigger would fire and select the next free ID value from dual, which is a special table that belongs to the system and allows to call functions from it, and assign to the new row of data inserted into the table. A trigger starts after the word BEGIN and finishes with the word END. Every piece of code in between those two words is the actual function that allows the interaction with the database.

As said above there are five triggers similar to each other due to having five sequences similar to each other as well.

Following the sequence triggers I created the trigger responsible for auditing the species table. This trigger is a bit more complex as we have to use 2 different tables that are not connected to each other and I also used an IF statement to cover all three of the operations responsible for data manipulation (insert, delete and update).

This trigger is still created the same way but now I had to specify that before every insert, delete or update to the tables I would also insert and update my auditing table.

This was possible by using an if statement with a few changes to my insert DML operation. Before an insert was done the trigger would fire and insert into de auditing table the new name, the date and the type of operation in this case insert. The same process is repeated for the delete DML operation but instead of insert a new name it would insert the old name or the name just deleted, the date and the operation that it would be delete. The same applies for the updating DML operation but now it would insert both new and old name, date and update operation.

This is followed by a END IF to let the trigger know that it is the end of the condition statement and follow by another END to let the platform know that the trigger is finished.

The next trigger is responsible to audit the schema in case a DDL operation is done. By creating any object including this trigger a log would be inserted into the schema\_audit table containing the date of the DDL operation, what kind of object, the name of that object and the operation. The only thing that differs this trigger from the other triggers is the specification of after a DDL operation happens on schema.

**Procedures**

After the creation of the triggers and sequences I inserted the values into the tables and populate the database including the auditing tables.

The creation of a procedure start CREATE OR REPLACE PROCEDURE followed by the name of the procedure.

The first procedure I created was a simple one and I made it to output species name, crew member name, the role and access level. To achieve this I had to select from the species table and the crew table by joining them together by their keys. When the data is selected, I stored it into a variable called **record** created a loop with a DBMS\_OUTPUT.PUT\_LINE to display all the data in the variable.

To display a procedure, I wrote EXEC or EXECUTE and the procedure name.

The most complex procedure I created was **display\_full\_crew\_details.** This procedure had as a purpose to display data from all the five main tables of the database. To achieve I create da FOR loop requesting species name, crew member name, role, access level, department and quarters. Every time I need to select one specific column, I need to specify form which table it is. The loop had four different conditions. The conditions were met by joining the tables together by using their primary and foreign keys. Once again I executed the procedure by using EXEC and the name after.

**Functions**

The difference between a function and a procedure is the function returns a value and procedure just executes commands.

Knowing this I created a function to return a value from the database, more specifically from species table.

The purpose of the first function is to return the specie of the selected crew member.

To create a function, I started by stating CREATE OR REPLACE FUNCTION followed by the name of the function and instead of commencing the main code block I need to specify from where do I want to select the data from, this case is the crew table.

At the beginning of the main code block I had to declare a variable and which table am I storing the data from. I, then, had to select the species name into the variable I declared from both species and crew tables by joining with their constraints and returned the variable I declared.

I have a version of the same function but one has a error handling exception that displays an error message if the data selected is not found.

To see if the function was correct, I wrote a block of code with 4 outputs with one of them having the name of the function and which ID I wanted to select from the table crew.

**Conclusion**

PL/SQL is a powerful tool to automate databases. The basic concept of each statement is not very hard to grasp but it does get tricky whenever a database is complex enough for a person to start joining massive amounts of tables. The ERD used in this project was simple as well which did not pose many issues in its creation but, again, when creating more complex databases the designer needs to have a good mental visualisation of what he wants for the ERD and the database.

**Appendix**

Appendix A:

