**Introduction**

This code is for a library management system that keeps track of the books, users, issues, returns and fines in a library. It uses SQL to create five tables: BOOKS, USERS, ISSUES, RETURNS and FINE. Each table has a primary key and some foreign keys to establish relationships with other tables. The code also uses SQL triggers to perform some actions before inserting a new row into the ISSUES, RETURNS or FINE tables. The triggers help to fill in some columns with values from other tables or to calculate some values based on some logic. The code also uses SQL sequences to generate unique IDs for the ISSUES and FINE tables. The code also inserts some sample data into the BOOKS and USERS tables for testing purposes.

**Question 1**

**Create tables**

* **BOOKS:** This table stores the information about the books in the library. It has three columns: BOOK\_ID, BOOK\_TITLE and BOOK\_AUTHOR. The BOOK\_ID column is the primary key of the table and it holds a unique identifier for each book. The BOOK\_TITLE column holds the name of the book. The BOOK\_AUTHOR column holds the name of the author of the book.
* **USERS:** This table stores the information about the users of the library. It has three columns: USER\_ID, NAME and DESIGNATION. The USER\_ID column is the primary key of the table and it holds a unique identifier for each user. The NAME column holds the name of the user. The DESIGNATION column holds the role of the user, either ‘STUDENT’ or ‘STAFF’.
* **ISSUES:** This table stores the information about the books that are issued to the users. It has eight columns: ISSUE\_ID, BOOK\_ID, BOOK\_TITLE, BORROWER\_ID, BORROWER\_NAME, BORROWER\_DESIGNATION, ISSUE\_DATE and DATE\_EXPIRY. The ISSUE\_ID column is the primary key of the table and it holds a unique identifier for each issue. The BOOK\_ID column is a foreign key that references the BOOK\_ID column in the BOOKS table and it holds the identifier of the book that is issued. The BOOK\_TITLE column holds the name of the book that is issued. The BORROWER\_ID column is a foreign key that references the USER\_ID column in the USERS table and it holds the identifier of the user who borrowed the book. The BORROWER\_NAME column holds the name of the user who borrowed the book. The BORROWER\_DESIGNATION column holds the role of the user who borrowed the book. The ISSUE\_DATE column holds the date when the book was issued. The DATE\_EXPIRY column holds the date when the book is due to be returned.
* **RETURNS:** This table stores the information about the books that are returned by the users. It has eight columns: RETURN\_ID, BOOK\_ID, BORROWER\_ID, BORROWER\_NAME, ISSUE\_DATE, DEADLINE\_DATE, RETURN\_DATE and DELAY. The RETURN\_ID column is both a primary key and a foreign key of this table. It is a primary key because it holds a unique identifier for each return. It is also a foreign key because it references the ISSUE\_ID column in the ISSUES table and it holds the identifier of the issue that corresponds to this return. The BOOK\_ID column is a foreign key that references the BOOK\_ID column in the BOOKS table and it holds the identifier of the book that is returned. The BORROWER\_ID column is a foreign key that references the USER\_ID column in the USERS table and it holds the identifier of the user who returned the book. The BORROWER\_NAME column holds the name of the user who returned the book. The ISSUE\_DATE column holds the date when the book was issued. The DEADLINE\_DATE column holds the date when the book was due to be returned. The RETURN\_DATE column holds the date when the book was actually returned. The DELAY column holds the number of days by which the book was returned late.
* **FINE:** This table stores the information about the fines that are imposed on the users who return the books late. It has seven columns: FINE\_ID, BOOK\_ID, BOOK\_TITLE, BORROWER\_ID, BORROWER\_NAME, DELAY and FINE\_AMOUNT. The FINE\_ID column is both a primary key and a foreign key of this table. It is a primary key because it holds a unique identifier for each fine. It is also a foreign key because it references the RETURN\_ID column in the RETURNS table and it holds the identifier of the return that corresponds to this fine. The BOOK\_ID column is a foreign key that references the BOOK\_ID column in the BOOKS table and it holds the identifier of the book that was returned late. The BOOK\_TITLE column holds the name of the book that was returned late. The BORROWER\_ID column is a foreign key that references the USER\_ID column in the USERS table and it holds the identifier of the user who returned the book late. The BORROWER\_NAME column holds the name of the user who returned the book late. The DELAY column holds the number of days by which the book was returned late. The FINE\_AMOUNT column holds the amount of money that the user has to pay as a fine.

**Sequence and Trigger Code**

**-ISSUE\_ID\_SEQ:** The sequence starts with a minimum value of 1 and a maximum value of 100000. It starts with the value 1 and increments by 1 each time a new value is generated.

**- ISSUE\_ID\_TRG:** This trigger generates a unique issue ID for each new row in the ISSUES table by using a sequence object called ISSUE\_ID\_SEQ. It assigns the next value of the sequence to the ISSUE\_ID column if it is null.

**- ISSUE\_NAME\_TRG**: This trigger fills in the borrower name for each new row in the ISSUES table by using a subquery that selects the name from the USERS table based on the borrower ID. It assigns the name to the BORROWER\_NAME column if it is null.

**- ISSUE\_DESIGN\_TRG:** This trigger fills in the borrower designation for each new row in the ISSUES table by using a subquery that selects the designation from the USERS table based on the borrower ID. It assigns the designation to the BORROWER\_DESIGNATION column if it is null.

**- ISSUE\_BOOK\_TRG:** This trigger fills in the book title for each new row in the ISSUES table by using a subquery that selects the book title from the BOOKS table based on the book ID. It assigns the title to the BOOK\_TITLE column if it is null.

**- ISSUE\_EXPIRY:** This trigger calculates the expiry date for each new row in the ISSUES table by adding a certain number of days to the issue date depending on the borrower designation. It uses a variable called ISSUER\_DESIGNATION to store the designation from the USERS table and then adds either 1 or 2 days to the issue date based on whether it is 'STUDENT' or 'STAFF'. It assigns the expiry date to the DATE\_EXPIRY column if it is null.

**- RETURN\_ID\_TRG:** This trigger fills in several columns for each new row in the RETURNS table by using subqueries that select the values from the ISSUES table based on the book ID. It assigns the values to the RETURN\_ID, BORROWER\_ID, BORROWER\_NAME, ISSUE\_DATE and DEADLINE\_DATE columns if they are null. The RETURN\_ID column is assigned with the maximum issue ID for that book ID, which means it corresponds to the most recent issue of that book. The other columns are assigned with the values from that issue ID.

**- RETURN\_DELAY\_TRG:** This trigger calculates the delay for each new row in the RETURNS table by subtracting the deadline date from the return date. It uses two variables called date1 and date2 to store the dates in a standard format and then computes their difference. If the difference is negative or zero, it means there is no delay and assigns zero to the DELAY column. If the difference is positive, it means there is a delay and assigns that value to the DELAY column.

**FINE\_TRG:** This trigger fills in several columns for each new row in the FINE table by using subqueries that select the values from the RETURNS and ISSUES tables based on the book ID. It assigns the values to the FINE\_ID, BOOK\_TITLE, BORROWER\_ID, BORROWER\_NAME and DELAY columns if they are null. The FINE\_ID column is assigned with the maximum return ID for that book ID, which means it corresponds to the most recent return of that book. The BOOK\_TITLE column is assigned with the value from the ISSUES table based on the maximum issue ID for that book ID. The other columns are assigned with the values from the RETURNS table based on the maximum return ID for that book ID.

**FINE\_AMT\_TRG**: This trigger calculates the fine amount for each new row in the FINE table by multiplying the delay by 10. It uses a subquery that selects the delay from the RETURNS table based on the maximum return ID for that book ID. It assigns the fine amount to the FINE\_AMOUNT column if it is null.

**Question 2**

Each insert statement adds a new row into the BOOKS or USERS table with the values specified in the parentheses. For example,

**Insert into Books (Book\_ID, Book\_Author, Book\_Title) values (‘B001’,‘Sewell’,‘The adventures of black beauty’);**

adds a new row into the BOOKS table with the values ‘B001’, ‘Sewell’ and ‘The adventures of black beauty’ for the columns BOOK\_ID, BOOK\_AUTHOR and BOOK\_TITLE respectively. This means that there is a book in the library with the ID ‘B001’, the author ‘Sewell’ and the title ‘The adventures of black beauty’. Similarly, the other insert statements add more books and users into their respective tables with different values for their columns. The commit statement at the end saves the changes made by the insert statements to the database.

**Insert into Books (Book\_ID, Book\_Author, Book\_Title) values (‘B001’,‘Sewell’,‘The adventures of black beauty’);**

adds a new row into the BOOKS table with the values ‘B001’, ‘Sewell’ and ‘The adventures of black beauty’ for the columns BOOK\_ID, BOOK\_AUTHOR and BOOK\_TITLE respectively. This means that there is a book in the library with the ID ‘B001’, the author ‘Sewell’ and the title ‘The adventures of black beauty’. Similarly, the other insert statements add more books and users into their respective tables with different values for their columns. The commit statement at the end saves the changes made by the insert statements to the database.

**Question 3**

1. Using `WHERE`:

**SELECT \* FROM books WHERE book\_author = 'Alcott';**

This query selects all columns from the `books` table where the `book\_author` is 'Alcott'.

2. Using `GROUP BY` and `HAVING`:

**SELECT book\_author, COUNT(\*) FROM books GROUP BY book\_author HAVING COUNT(\*) > 1;**

This query selects the `book\_author` and the count of books for each author from the `books` table, grouped by the `book\_author`. The `HAVING` clause filters the results to only include authors with more than one book.

3. Using `ORDER BY`:

**SELECT \* FROM books ORDER BY book\_title ASC;**

This query selects all columns from the `books` table and orders the results by the `book\_title` in ascending order.

**Question 4**

1. Single-row subquery:

**SELECT \* FROM books WHERE book\_id = (SELECT book\_id FROM issues WHERE borrower\_id = 'STUD\_005');**

This query selects all columns from the `books` table where the `book\_id` is equal to the `book\_id` returned by the subquery. The subquery selects the `book\_id` from the `issues` table where the `borrower\_id` is 'STUD\_005'. Since the subquery returns a single value, it can be used in a comparison with the `=` operator.

2. Multiple-row subquery:

**SELECT \* FROM books WHERE book\_id IN (SELECT book\_id FROM issues WHERE borrower\_id = 'STAF\_001');**

This query selects all columns from the `books` table where the `book\_id` is in the list of `book\_id`s returned by the subquery. The subquery selects the `book\_id` from the `issues` table where the `borrower\_id` is 'STAF\_001'. Since the subquery can return multiple values, it must be used with an operator that can handle multiple values, such as `IN`.

**Question 5**

1. `LEFT OUTER JOIN`:

**SELECT books.book\_id, books.book\_title, issues.issue\_id**

**FROM books**

**LEFT OUTER JOIN issues ON books.book\_id = issues.book\_id;**

This query selects the `book\_id` and `book\_title` from the `books` table and the `issue\_id` from the `issues` table. The `LEFT OUTER JOIN` returns all records from the `books` table and the matching records from the `issues` table. If there is no matching record in the `issues` table, the result will still include the record from the `books` table with null values for the columns from the `issues` table.

2. `RIGHT OUTER JOIN`:

**SELECT books.book\_id, books.book\_title, issues.issue\_id**

**FROM books**

**RIGHT OUTER JOIN issues ON books.book\_id = issues.book\_id;**

This query is similar to the previous one, but it uses a `RIGHT OUTER JOIN` instead of a `LEFT OUTER JOIN`. The `RIGHT OUTER JOIN` returns all records from the `issues` table and the matching records from the `books` table. If there is no matching record in the `books` table, the result will still include the record from the `issues` table with null values for the columns from the `books` table.

3. `FULL OUTER JOIN`:

**SELECT books.book\_id, books.book\_title, issues.issue\_id**

**FROM books**

**FULL OUTER JOIN issues ON books.book\_id = issues.book\_id;**

This query selects the same columns as the previous two queries, but it uses a `FULL OUTER JOIN`. The `FULL OUTER JOIN` returns all records from both tables and matches records where possible. If there is no matching record in either table, the result will still include the record from one of the tables with null values for the columns from the other table.

Note that these queries assume that suitable records have been inserted into both tables to get meaningful results.

**Question 6**

**CREATE VIEW books\_view AS**

**SELECT book\_id, book\_title, book\_author**

**FROM books;**

This query creates a view named `books\_view` that selects the `book\_id`, `book\_title`, and `book\_author` columns from the `books` table. Once the view is created, you can query it just like you would query a table. For example:

**SELECT \* FROM books\_view WHERE book\_author = 'Alcott';**

This query selects all columns from the `books\_view` view where the `book\_author` is 'Alcott'.

**Question 7**

**DECLARE**

**v\_book\_id books.book\_id%TYPE := 'B005';**

**v\_book\_title books.book\_title%TYPE;**

**BEGIN**

**SELECT book\_title INTO v\_book\_title FROM books WHERE book\_id = v\_book\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Book Title: ' || v\_book\_title);**

**END;**

This PL/SQL block declares two variables: `v\_book\_id` and `v\_book\_title`. The `v\_book\_id` variable is assigned the value 'B005', which is the specific input for this example. The block then selects the `book\_title` from the `books` table where the `book\_id` is equal to the value of the `v\_book\_id` variable and stores the result in the `v\_book\_title` variable. Finally, the block outputs the value of the `v\_book\_title` variable using the `DBMS\_OUTPUT.PUT\_LINE` procedure.

You can modify this PL/SQL block to retrieve records for different specific inputs by changing the value of the `v\_book\_id` variable and the query used to select data from the `books` table.

**Question 8**

**DECLARE**

**v\_book\_id books.book\_id%TYPE := 'B006';**

**v\_new\_title books.book\_title%TYPE := 'New Title';**

**BEGIN**

**UPDATE books SET book\_title = v\_new\_title WHERE book\_id = v\_book\_id;**

**COMMIT;**

**END;**

This PL/SQL block declares two variables: `v\_book\_id` and `v\_new\_title`. The `v\_book\_id` variable is assigned the value 'B006' and the `v\_new\_title` variable is assigned the value 'New Title', which are the specific inputs for this example. The block then updates the `book\_title` in the `books` table to the value of the `v\_new\_title` variable where the `book\_id` is equal to the value of the `v\_book\_id` variable. Finally, the block commits the changes to the database.

You can modify this PL/SQL block to update records for different specific inputs by changing the values of the `v\_book\_id` and `v\_new\_title` variables and the query used to update data in the `books` table.

**Question 9**

**DECLARE**

**v\_book\_id books.book\_id%TYPE := 'B001';**

**BEGIN**

**DELETE FROM books WHERE book\_id = v\_book\_id;**

**COMMIT;**

**END;**

This PL/SQL block declares a variable named `v\_book\_id` and assigns it the value 'B001', which is the specific input for this example. The block then deletes the record from the `books` table where the `book\_id` is equal to the value of the `v\_book\_id` variable. Finally, the block commits the changes to the database.

You can modify this PL/SQL block to delete records for different specific inputs by changing the value of the `v\_book\_id` variable and the query used to delete data from the `books` table.

**Question 10**

**DECLARE**

**v\_book\_id books.book\_id%TYPE := 'B003';**

**v\_rows\_deleted INTEGER;**

**BEGIN**

**DELETE FROM books WHERE book\_id = v\_book\_id;**

**v\_rows\_deleted := SQL%ROWCOUNT;**

**DBMS\_OUTPUT.PUT\_LINE('Rows deleted: ' || v\_rows\_deleted);**

**COMMIT;**

**END;**

This PL/SQL block is similar to the previous example, but it adds a new variable named `v\_rows\_deleted` and assigns it the value of `SQL%ROWCOUNT` after the `DELETE` statement. The `SQL%ROWCOUNT` attribute returns the number of rows affected by the last SQL statement, which in this case is the number of rows deleted. The block then outputs the value of the `v\_rows\_deleted` variable using the `DBMS\_OUTPUT.PUT\_LINE` procedure. Finally, the block commits the changes to the database.

You can modify this PL/SQL block to delete records for different specific inputs and display the number of rows deleted by changing the value of the `v\_book\_id` variable and the query used to delete data from the `books` table.