It seems like you are asking for help with SQL. I can help you with that. However, I would like to clarify that I am not a database management system and I cannot execute SQL commands directly. But I can provide you with the SQL queries that you can execute on your database management system.

Here are the SQL queries that you have requested:

1. Create tables with primary key and foreign key constraints having auto-increment sequence for one of the tables.

CREATE TABLE table1 (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

age INT NOT NULL

);

CREATE TABLE table2 (

id INT PRIMARY KEY AUTO\_INCREMENT,

table1\_id INT NOT NULL,

FOREIGN KEY (table1\_id) REFERENCES table1(id),

name VARCHAR(50) NOT NULL,

age INT NOT NULL

);

1. Insert a set of matching records for the above tables.

INSERT INTO table1 (name, age) VALUES ('John', 25), ('Jane', 30), ('Bob', 35);

INSERT INTO table2 (table1\_id, name, age) VALUES (1, 'John', 25), (2, 'Jane', 30), (3, 'Bob', 35);

1. Write any select queries each using where, group by, having, and order by.

-- Select query using WHERE clause

SELECT \* FROM table1 WHERE age > 25;

-- Select query using GROUP BY clause

SELECT age, COUNT(\*) FROM table1 GROUP BY age;

-- Select query using HAVING clause

SELECT age, COUNT(\*) FROM table1 GROUP BY age HAVING COUNT(\*) > 1;

-- Select query using ORDER BY clause

SELECT \* FROM table1 ORDER BY age DESC;

1. Write a single-row and multiple-row subquery using the above tables.

-- Single-row subquery

SELECT name FROM table2 WHERE table1\_id = (SELECT id FROM table1 WHERE name = 'John');

-- Multiple-row subquery

SELECT \* FROM table2 WHERE table1\_id IN (SELECT id FROM table1 WHERE age > 25);

1. Write queries using left, right, and full outer joins. (There should be suitable records inserted to get the required outputs)

-- Left join

SELECT \* FROM table1 LEFT JOIN table2 ON table1.id = table2.table1\_id;

-- Right join

SELECT \* FROM table1 RIGHT JOIN table2 ON table1.id = table2.table1\_id;

-- Full outer join

SELECT \* FROM table1 FULL OUTER JOIN table2 ON table1.id = table2.table1\_id;

1. Create a view using one of the tables created.

CREATE VIEW view\_table AS SELECT \* FROM table1 WHERE age > 25;

1. Write a PL/ SQL block to retrieve a record for specific input.

DECLARE

v\_name VARCHAR(50);

v\_age INT;

BEGIN

SELECT name, age INTO v\_name, v\_age FROM table1 WHERE id = 2;

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name || ', Age: ' || v\_age);

END;

1. Write a PL/ SQL block to update a record for specific input.

DECLARE

v\_name VARCHAR(50);

BEGIN

SELECT name INTO v\_name FROM table1 WHERE id = 3;

UPDATE table1 SET name = 'Robert' WHERE id = 3;

DBMS\_OUTPUT.PUT\_LINE('Record updated from ' || v\_name || ' to Robert');

END;

1. Write a PL/ SQL block to delete a record for specific input.

DECLARE

v\_name VARCHAR(50);

BEGIN

SELECT name INTO v\_name FROM table1 WHERE id = 3;

DELETE FROM table1 WHERE id = 3;

DBMS\_OUTPUT.PUT\_LINE('Record deleted: ' || v\_name);

END;

1. Modify the above query to display the number of rows deleted.

DECLARE

v\_count INT;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM table1 WHERE age > 25;

DELETE FROM table1 WHERE age > 25;

DBMS\_OUTPUT.PUT\_LINE(v\_count || ' rows deleted.');

END;

Finals

**Question 3**

1. Using `WHERE`:

```SQL

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`:

```SQL

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`:

```SQL

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:

```SQL

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

```

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\_001'. Since the subquery returns a single value, it can be used in a comparison with the `=` operator.

2. Multiple-row subquery:

```SQL

SELECT \* FROM books WHERE book\_id IN (SELECT book\_id FROM issues WHERE borrower\_id = 'STUD\_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 'STUD\_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`:

```SQL

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`:

```SQL

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`:

```SQL

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**

```SQL

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:

```SQL

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**

```SQL

DECLARE

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

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 'B001', 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**

```SQL

DECLARE

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

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 'B001' 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**

```SQL

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**

```SQL

DECLARE

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

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.