GENERIC COVER SHEET FOR ASSESSMENTS

Work submitted:

- Must be written in your own words (i.e. not plagiarised, not copied and pasted from other sources such textbooks or internet sources).
    - **Exception:** *You may use the course materials from this course that belong to the University of Aberdeen (slides, lecture notes, tutorial sheets) without attribution of source. You may quote other sources if the quoted material is clearly indicated and with appropriate attribution. However, any errors arising from your use of another person's incorrect material are your own, and will be marked as incorrect.*
- Must not have been prepared in collaboration with others (collusion)
- Must be prepared by you and not by a third party (contract cheating).

Your attention is drawn to the University's Code of Practice on Student Discipline (Academic) which outlines the standard penalties that will be applied to any student who is found to have committed academic misconduct. These penalties could include expulsion from the University.

'IT problems' will also not be regarded as an acceptable excuse for late or non-submission.

Course/assignment-specific information

COURSE: QC2002
Assessment: assessment 2 SQL
Course Coordinator: Rami Hamdi
Submission date: 17/10/2022

This document contains a report which reflects on each SQL questions in detail.
References at the end.

Student ID: 50080758 **Signature:** Niha Salman

(To make your signature, either

- type your name in the box, or
- sign by hand, photograph, and convert to pdf.)

I confirm that the work I am submitting is my own work, written in my own words by me without help from other people.

*** This cover sheet **must** be attached/uploaded along with your assessment. ***

SQL ASSESSMENT REPORT

This report critically reflects on what process was followed to implement a solution for each SQL question, how successful was the process and what could be improved.

Question 0 display all data for all users

```
SQLQ[0] = "SELECT *
          FROM Employees;"
```

SELECT statement is used to select the column needed for the question and in this case all the columns are selected. FROM is used to define which table is used where the columns are which in this case is the Employees. In writer's opinion, this is the best possible solution as the question is relatively straight forward and require basic SQL knowledge.

Question 1 display last name and first name data for all employees

```
SQLQ[1] = "SELECT LastName, FirstName
          FROM Employees;"
```

SELECT statement is used to select the columns LastName and FirstName. FROM is used to define the Employees table. In writer's opinion, this is the best possible solution as the question is relatively straight forward and require basic SQL knowledge.

Question 2 display employee id, last name and first name data for all employees and rename columns (EmployeeId renamed Staff ID, FirstName renamed First Name and LastName renamed Last Name)

```
SQLQ[2] = "SELECT EmployeeId AS 'Staff ID', FirstName AS 'First Name', LastName AS 'Last Name'
          FROM Employees;"
```

SELECT statement is used to select the EmployeeId, FirstName, LastName. AS is used to rename FirstName to "First Name" and LastName to "Last Name". FROM is used to define the table employees. One attempt was taken to solve this question. In writer's opinion, this is the best possible solution as the question is relatively straight forward and require basic SQL knowledge.

Question 3 display LastName and FirstName data for employee with id = 2

```
SQLQ[3] = "SELECT LastName, FirstName
          FROM Employees
          WHERE EmployeeId=2;"
```

SELECT statement is used to select the LastName, FirstName and FROM is used to define the table employees. WHERE is used to specify EmployeeId to be equal to 2 for the result to show the first and last name of the person with the employee Id of 2. One attempt was taken to solve this question as the writer believes that the question is straight forward. Moreover, in writer's opinion this is the best solution to the question as it requires basic SQL knowledge.

Question 4 display all data for employees whose last name begins with 'P'

```
SQLQ[4] = "SELECT *
        FROM employees
        WHERE employees.LastName LIKE'P%';"
```

SELECT statement is used to select all the columns and FROM is used to the define the table employees. WHERE is used to specify the requirement and LIKE is used to search for a specific pattern which in this case is last name starting with P. The percentage signs (%) represents the one character 'P'. It only took one attempt to solve this question as it is relatively straight forward therefore is a best possible solution as the question is simply asking for an employee with the last name beginning with P so there is no need for any type of JOINS to be used in the solution.

Question 5 display all invoice and customer details for invoice id 1

```
SQLQ[5] = "SELECT *
        FROM invoices INNER JOIN customers ON invoices.CustomerId = customers.CustomerId
        WHERE invoices.InvoiceId=1;"
```

SELECT statement is used to select all the columns from invoices and customers table. INNER JOIN is used to create a new table which combines invoices and customers table based on the join-predicate which in this case is the CustomerId. As the two tables are joined by the CustomerId, WHERE statement filters out the invoice id from invoices table to be equal to 1. This solution works and in writer's opinion is the best solution as the question is simply asking for customer with invoice id 1 meaning that no JOINS are required.

Question 6 display invoice id, customer first name and last name and total details for invoice id 1 and not using the JOIN keyword

```
SQLQ[6] = "SELECT invoices.InvoiceId, customers.FirstName, customers.LastName, invoices.Total
        FROM invoices, customers
        WHERE invoices.CustomerId = customers.CustomerId AND invoices.InvoiceId=1;"
```

SELECT statement selects the invoice id and total from the invoices table and first and last name from the customers table. FROM defines the table customers and invoices. WHERE filters out the data and specifies that CustomerId in the invoices table must equal to the CustomerId in the customers table. Moreover, it specifies the InvoiceId in the invoices table to be equal to 1. The solution worked however in writer's opinion, this is not the best possible solution. Using INNER JOIN would make the solution better, as it will reduce the number of connections. The writer didn't use JOIN as the question specifies to not use it. Furthermore, as it is tedious to write the table name multiple times to select the data or to specify requirements in the WHERE statement, the use of table alias could make the solution better as It makes it easier to write and read statements.

Question 7 display invoice id, customer first name and last name, and support representative first name and last name for invoice id 1, and not using the JOIN keyword

```
SQLQ[7] = "SELECT invoices.InvoiceId, customers.FirstName, customers.LastName, employees.FirstName, employees.LastName
          FROM invoices, customers, employees
          WHERE invoices.CustomerId=customers.CustomerID AND customers.SupportRepId=employees.EmployeeId AND invoices.InvoiceId=1;"
```

SELECT statement selects the InvoiceId from the invoices table, last and first name from the customers table and first and last name from the employees table. WHERE filters data and specifies the CustomerId in the invoices table to be equal to the CustomerId in the customers table. Moreover, it specifies the SupportRepId in the customers table to be equal to the EmployeeId in the employees table and it specifies the InvoiceId in the invoices table to be 1. The solution works however in writer's opinion, this is not the best possible solution. Using INNER JOIN would make the solution better, as it will reduce the number of connections. The writer didn't use JOIN as the question specifies to not use it. Furthermore, as it is tedious to write the table name multiple times to select the data or to specify requirements in the WHERE statement, the use of table alias could make the solution better as It makes it easier to write and read statements.

Question 8 display invoice id, customer first name and last name and merge the name of the support representative as one column named "Support Contact" for invoice id 1 and not using the JOIN keyword

```
SQLQ[8] = "SELECT invoices.InvoiceId, customers.FirstName, customers.LastName, employees.FirstName||' '||employees.LastName AS 'Support Contact'
          FROM invoices, customers, employees
          WHERE invoices.CustomerId=customers.CustomerId AND customers.SupportRepId=employees.EmployeeId AND invoices.InvoiceId=1;"
```

SELECT statement selects InvoiceId from the invoices table first and last name from the customers table and first and last name from the employees table. As the first and last name of the employee are two separate columns, ||''|| merges the employee's first and last name to one column and names the column "Support Contact". WHERE statement specifies the CustomerId in the invoice table to be equal to the CustomerId in the customers table. Moreover, it specifies the SupportRepId in the customers table to be equal to the EmployeeId in the employees table and it specifies the InvoiceId in the invoices table to be equal to 1. The solution works however in writer's opinion, this is not the best possible solution. Using

INNER JOIN would make the solution better, as it will reduce the number of connections. The writer didn't use JOIN as the question specifies to not use it. Furthermore, as it is tedious to write the table name multiple times to select the data or to specify requirements in the WHERE statement, the use of table alias could make the solution better as It makes it easier to write and read statements.

As a note the writer wanted to use the "employees.FirstName+' '+employees.LastName" method to merge the columns into one but the ruby didn't recognize this method. So, the ||''|| method is used (Aabaab et al., 2014)

Question 9 display German customers first name, last name and country along with the merged name as one column named "Support Contact" who are supported by employee "Johnson" not using the JOIN keyword

```
SQLQ[9] = "SELECT customers.FirstName, customers.LastName, customers.Country, employees.FirstName||' '||employees.LastName AS 'Support Contact'
        FROM customers, employees
        WHERE customers.Country='Germany' AND customers.SupportRepId=employees.EmployeeId AND employees.LastName='Johnson';"
```

SELECT statement selects first name, last name and country from the customers table and first and last name from the employees table. As the first and last name of the employee are two separate columns, ||''|| merges the employee's first and last name to one column and names the column "Support Contact". WHERE statement specifies the country in the customers table to be "Germany". Moreover, it specifies the SupportRepId in the customers table to be equal to the EmployeeId in the employees table and it specifies the employee's last name to be Johnson. The solution works however in writer's opinion, this is not the best possible solution. Using INNER JOIN would make the solution better. The writer didn't use JOIN as the question specifies to not use it. Furthermore, as it is tedious to write the table name multiple times to select the data or to specify requirements in the WHERE statement, the use of table alias could make the solution better as It makes it easier to write and read statements.

As a note the writer wanted to use the "employees.FirstName+' '+employees.LastName" method to merge the columns into one but the ruby didn't recognize this method. So, the ||''|| method is used (Aabaab et al., 2014)

Question 10 display employee first and last name along with the number of customers they support for employee id 5, not using the JOIN keyword, and using the table alias e for employees and c for customers

```
SQLQ[10] = "SELECT e.FirstName, e.LastName, COUNT(*)
                FROM customers c, employees e
                WHERE c.SupportRepId=e.EmployeeId AND e.EmployeeId=5;"
```

SELECT statement selects first and last name from the employees table and counts all the customers using the COUNT function. WHERE statement is used to specify that SupportRepId in the customers table must equal to the EmployeeId in the employees table. Moreover, it specifies the EmployeeId to be 5. This

solution produces the name of the employee with id 5 and the number of customers the employee supports. In this solution table alias is used which means that table are renamed temporarily. The solution works however in writer's opinion, this is not the best possible solution. Using INNER JOIN would make the solution better. The writer didn't use JOIN as the question specifies to not use it.

Question 11 display German customers last name and first name in descending order

```
SQLQ[11] = "SELECT customers.LastName, customers.FirstName
            FROM customers
           WHERE customers.Country='Germany'
           ORDER BY customers.LastName DESC; "
```

SELECT statement selects the last and first name from the customer's table. WHERE statement specifies the Country in the customers table to be "Germany". ORDER BY and DESC is used to order the customer's last name in descending order. The solution works and in writer's opinion is the best possible solution as there is no need for integrating any type of JOINS as the question is simply asking to filter out customer and show only the ones from Germany.

Question 12 display name of country and number of customers who are either German or Canadian in ascending order

```
SQLQ[12] = "SELECT customers.Country, count (*)
            FROM customers
            WHERE customers.country='Germany' OR customers.country='Canada'
            GROUP BY Country
            ORDER BY customers.Country;"
```

SELECT statement selects the country from the table customers. COUNT function is used to count the number of customers. WHERE statement specifies the country from the customers table to be either Germany or Canada. GROUP BY is used to arrange the two countries into groups. ORDER BY is used to order the two countries in ascending order. This displays the number of customers from Germany and from Canada. The solution works and in writer's opinion is the best possible solution as the question is simply asking for two sets of customers from either Germany or Canada.

Question 13 display customers ids along with their total spend for tracks in ascending order spent for the first 9 invoices

```
SQLQ[13] = "SELECT customers.CustomerId, invoices.Total AS 'Total Spent'
            FROM customers INNER JOIN invoices ON invoices.CustomerId=customers.CustomerId
            WHERE invoices.InvoiceId<10
```

```
                ORDER BY invoices.Total;"
```

SELECT statement selects CustomerId from the customers table and the total from the invoices table naming the total as “Total Spent”. INNER JOIN is used to create a new table which combines invoices and customers table based on the join-predicate which in this case is the CustomerId. WHERE statement specifies the InvoiceId from invoice table to be less than 10 and ORDER BY is used to order the total from the invoices table in ascending order. The solution works however in writer’s opinion, the solution could be made better if table alias is used to give tables a temporary name. It makes it easier to write and read statements.

Question 14 display customers ids along with their total spend for tracks in descending order spent for the invoices greater or equal to £45.

```
SQLQ[14] = "SELECT customers.CustomerId, SUM(invoices.Total) AS 'Total_Spent'
                FROM customers INNER JOIN invoices ON invoices.CustomerId=customers.CustomerId
               GROUP BY customers.CustomerId
               HAVING Total_Spent>=45
               ORDER BY Total_Spent DESC;"
```

SELECT statement selects the CustomerId from the customers table and Total from the invoices table. SUM function is used to sum up the total from the invoices table, naming the total as “Total Spent”. INNER JOIN is used to create a new table which combines invoices and customers table based on the join-predicate which in this case is the CustomerId. GROUP BY is used to group the CustomerId. As GROUP BY is used, HAVING is used to specify the “Total Spent” to be more than or equal to 45. Lastly, ORDER BY is used to order the Total Spent in descending order. The solution works and in writer’s opinion is the best possible solution. As a note, HAVING is used in this solution instead of WHERE, as, WHERE statement can only specify a condition before a grouping is made whereas HAVING statement can specify a condition from the groups made.

As multiple attempts were made, another method could be used for the solution. Instead of using INNER JOIN, WHERE statement can be used to specify the customerId in the invoices and the customers table to be equal. This solution works as well.

```
SQLQ[14] = "SELECT customers.CustomerId, SUM(invoices.Total) AS 'Total_Spent'
                FROM customers, invoices
                WHERE invoices.CustomerId=customers.CustomerId
               GROUP BY customers.CustomerId
               HAVING Total_Spent>=45
               ORDER BY Total_Spent DESC;"
```

Question 15 display last and first names, job title and date hired of employees hired after 2003

```
SQLQ[15] = "SELECT employees.LastName, employees.FirstName, employees.Title, employees.HireDate AS 'Date'
           FROM employees
           WHERE SUBSTRING(HireDate,-4)>='2004';"
```

SELECT statement selects last name, first name, title and HireDate from the employees table. The HireDate is named as "Date". WHERE statement specifies the HireDate to be more than or equal to 2004 using the function SUBSTRING which retrieves characters from a string (w3schools, *SQL Server SUBSTRING() Function*). SUBSTRING first specifies the column which is the HireDate then specifies the length which is -4. The number -4 is used to exclude the month and the day from the column HireDate as the condition is only specifying for the year. The solution works and as it took a couple of attempts to get the solution to work, it is the best possible solution.

Question 16 display last and first names, job title and date hired of employees hired during 2002;

```
SQLQ[16] = "SELECT employees.LastName, employees.FirstName, employees.Title, employees.HireDate AS 'Date'
           FROM employees
           WHERE employees.HireDate LIKE '%2002'
           GROUP BY employees.HireDate
           ORDER BY employees.LastName;"
```

SELECT statement selects the last name, first name, title and hire date from the employees table. Hire date named as "Date" (as specified in the expected result provided). WHERE specifies the requirement and using LIKE to search for a specific pattern which in this case is the year to be 2002. % sign is used to represent the multiple characters in the number '2002'. GROUP BY is used to arrange the same values from HireDate into groups. The question doesn't ask for the last name to be in the ascending, but the expected result displayed the last names in ascending order hence why the ORDER BY is used. The solution works and in writer's opinion is the best possible solution however the use of table alias could make the solution better as It makes it easier to write and read statements.

Question 17 display employee first, last names and job title who have other employees reporting to them.

```
SQLQ[17] = "SELECT E.LastName, E.FirstName, E.Title
           FROM employees E, employees R
           WHERE E.EmployeeId=R.ReportsTo
           GROUP BY E.EmployeeId;"
```

SELECT statement selects last name, first name and title from the employees table. WHERE statement specifies that the EmployeeId from employees table to be equal to the ReportsTo column in the employees table. As seen in the code above, two table alias is used to assign different names to the same table. This is done to create a SELF JOIN that allows the table to join to itself. SELF JOIN is used as both columns 'ReportsTo' and 'EmployeeId' are from the employees table. Lastly, GROUP BY statement groups

the EmployeeId. The solution works and in writer's opinion is the best possible solution as it is simply asking to filter employees who have others reporting to them.

Question 18 display a column called ''Customers Reporting to Managers' containing the number of employees who have other employees reporting to them and who also have customers to support.

```
SQLQ[18] = "SELECT COUNT(employees.EmployeeId) AS 'Customers Reporting to Managers'
            FROM employees INNER JOIN customers ON employees.EmployeeId=customers.SupportRepId
            WHERE employees.EmployeeId=employees.ReportsTo;"
```

SELECT statement selects the EmployeeId from the table employees. COUNT function is used to count the EmployeeId which is named as "Customers Reporting to Managers". INNER JOIN is used to create a new table which combines employees and customers table based on the join-predicate which in this case is the EmployeeId (EmployeeId is named as SupportRepId in the customers table). WHERE statement specify that the EmployeeId from the employees table to be equal to the ReportsTo column in the employees table. The solution works and in writer's opinion is the best possible solution as it is simply asking to filter out employees who support customers and who have others reporting to them.

Question 19 display employees last name, first name and job title who have employees who report to them who themselves have other employees who report to them.

```
SQLQ[19] = "SELECT employees.LastName, employees.FirstName, employees.Title

            FROM employees

            WHERE employees.ReportsTo IS NULL AND employees.Title LIKE ('%Manager');"
```

SELECT statement selects Last name, first name and title from the employees table. WHERE statement specifies that the ReportsTo column in the employees table to be null. LIKE statement is used to specify the Title in the employees table to have the word "Manager". % is used to find any title value to be ending with "Manager". The solution works and, in writer's opinion is the best possible solution as the question don't require them to be reporting to anyone but others reporting to them.

Question 20 display the last name, first name and title of employees who do not report to another employee

```
SQLQ[20] = "SELECT employees.LastName, employees.FirstName, employees.Title

            FROM employees

            WHERE employees.ReportsTo IS NULL;"
```

SELECT statement selects the last name, first name and the title from the employees table. WHERE statement specifies the ReportTo column in the employees table to be equal to null. The solution works and in writer's opinion, is the best possible solution as there is no need for JOINS and table alias as the question is not looking for a link between tables or self joins. The question is simply asking for employees that are not reporting to anyone meaning that column should be empty.

Question 21 display artist id, their name and their number of albums if produced more than 10.

```
SQLQ[21] = " SELECT artists.ArtistId, artists.Name, COUNT(albums.AlbumId) AS 'Number_of_Albums'
                FROM artists INNER JOIN albums ON albums.ArtistId=artists.ArtistId
               GROUP BY artists.ArtistId
               HAVING Number_of_Albums>10;"
```

SELECT statement selects ArtistId and name from the artists table and AlbumId from the albums which is counted using the COUNT function which is then named as "'Number_of_Albums". INNER JOIN is used to create a new table which combines albums and artists table based on the join predicate ArtistId. GROUP BY then groups the ArtistId. HAVING is used to specify the "Number_of_Albums" to be more than 10. The solution works and in writer's opinion is the best solution. It took a couple of attempts to get the solution to work and during the process, another possible solution was found, where INNER JOIN is replaced with WHERE but is specifying the same condition as done by INNER JOIN.

```
SQLQ[21] = "SELECT artists.ArtistId, artists.Name, COUNT(albums.AlbumId) AS 'Number_of_Albums'
FROM artists, albums WHERE albums.ArtistId=artists.ArtistId GROUP BY artists.ArtistId HAVING
Number_of_Albums>10;"
```

Question 22 display album id and title produced by AC/DC using a nested query.

```
SQLQ[22] = "SELECT albums.AlbumId, albums.Title FROM albums WHERE albums.ArtistId = (SELECT
artists.ArtistId FROM artists WHERE artists.Name='AC/DC');"
```

SELECT statement selects the AlbumId and title from the albums table. WHERE statement specifies the ArtistId in the albums table to be equal to the nested SELECT statement which selects ArtistId from the table artists. WHERE statement specifies the artists name to be "AC/DC". The solution works and in writer's opinion is the best possible solution as the question is simply asking to filter out the "AC/DC" title.

Question 23 display all invoice id, customer id whose total invoice is greater than the average price, and show by how much their price is greater than average where prices differ by more than £15.

```
SQLQ[23] = "SELECT invoices.InvoiceId, invoices.CustomerId, invoices.Total-(SELECT AVG(invoices.Total) FROM invoices) AS 'priceDiff'
            FROM invoices
            GROUP BY invoices.InvoiceId
            HAVING priceDiff>15;"
```

SELECT statement selects InvoiceId, CustomerId, and total from the invoices table. Another SELECT statement is nested into the query to calculate the average of the total from the invoices table using the AVG function. Total is then subtracted from the average total, and this is named as "priceDiff" in the query. GROUP BY statement is used to arrange the same values from InvoiceId into groups. As HAVING can specify conditions from the groups made, it is used to specify the "priceDiff" to be more than 15.

The solution works and in writer's opinion is the best possible solution. It took a couple of attempts to get the solution to work and during the process, another possible solution was found, where rather than using a nested query method, self-join method can be used.

```
SQLQ[23] = "SELECT I.InvoiceId, I.CustomerId, (I.Total-AVG(T.Total)) AS 'priceDiff'
            FROM invoices I, invoices T
            GROUP BY I.InvoiceId
            HAVING priceDiff>15;"
```

Question 24 display employees id first name, last name and title who support companies sorted by lastname and using the reserve EXIST word

```
SQLQ[24] = "SELECT employees.EmployeeId, employees.FirstName, employees.LastName, employees.Title
           FROM employees
           WHERE EXISTS (SELECT employees.EmployeeId FROM employees WHERE employees.EmployeeId<>employees.ReportsTo)
           AND employees.Title LIKE '%Support%'
           ORDER BY employees.LastName;"
```

SELECT statement selects EmployeeId, first name, last name and the title from the employees table. EXIST is used in the WHERE statement to see the existence of record where EmployeeId is not equal to ReportsTo column in the employees table. Furthermore, LIKE is used to search for a specific pattern which in this case is the title in the employees table to have the word "Support" in any position. The two percentage signs are used to find the word "Support" in any position (w3schools, *SQL LIKE Operator*). ORDER BY is used to order last name in ascending order. This solution works and in writer's opinion is the best possible solution as the question is simply asking to filter out employees who support companies.

Question 25 display tracks which exist on a Grunge playlist but no invoice using reserve EXCEPT word

```
SQLQ[25] = "SELECT playlist_track.TrackId
           FROM playlist_track
           INNER JOIN playlists ON playlist_track.PlaylistId=playlists.PlaylistId AND playlists.Name='Grunge'
           EXCEPT
           SELECT invoice_items.TrackId
           FROM invoice_items, playlist_track
           WHERE invoice_items.TrackId<>playlist_track.TrackId;"
```

SELECT statement selects TrackId from the playlist_track table. INNER JOIN is used to create a new table which combines playlist_track and playlists table based on the join-predicate which is the PlaylistId and the name in the playlist should be "Grunge". EXCEPT statement is used to combine two SELECT statements and return the first statement which is not returned by the second one (TutorialsPoint, *SQL - except clause*). In this second statement, SELECT statement selects TrackId from the invoice_items and the playlist_track table. WHERE statement specifies the TrackId in the invoice_track to be not equal to the TrackId in the playlist_track table. The solution works and in writer's opinion is the best possible solution. It took a couple of attempts to get the solution to work and during the process, another possible solution was found, where rather than using an INNER JOIN in the first SELECT statement, WHERE statement could be used which specifies the PlaylistId in the playlist_track table to be equal to 16.

```
SELECT playlist_track.TrackId

FROM playlist_track

WHERE playlist_track.PlaylistId=16

EXCEPT

SELECT invoice_items.TrackId FROM invoice_items, playlist_track WHERE
invoice_items.TrackId<>playlist_track.TrackId;
```

Question 26 display playlist id, track id and track name of playlists which contain only one track

```
SQLQ[26] = "SELECT playlists.PlaylistId, playlist_track.TrackId, tracks.Name

            FROM playlists

            INNER JOIN playlist_track ON playlist_track.PlaylistId=playlists.PlaylistId

            INNER JOIN tracks ON tracks.TrackId=playlist_track.TrackId

            GROUP BY playlists.PlaylistId

            HAVING COUNT(playlist_track.TrackId)=1;"
```

SELECT statement selects PlaylistId from the playlists table, TrackId from the playlist_track table and Name of the track from the tracks table. INNER join is then used to to create a new table which combines playlist_track and playlists table based on the join-predicate PlaylistId and combines the playlist_track and tracks table based on the join-predicate TrackId. As GROUP BY is used, HAVING is used to specify the TrackId to be equal to 1. The solution works and is the best possible solution however another solution could be used which replaces INNER JOIN with WHERE statement but works the same way as the INNER JOIN.

```
SELECT playlists.PlaylistId, playlist_track.TrackId, tracks.Name

FROM playlists, playlist_track, tracks

WHERE playlist_track.PlaylistId=playlists.PlaylistId AND playlist_track.TrackId=tracks.TrackId

GROUP BY playlists.PlaylistId

HAVING COUNT(playlist_track.TrackId)=1;
```

Question 27 display all artists id and their names, the number of albums produced containing the genre 'Soundtrack' using left joins only

```
SQLQ[27] = "SELECT albums.ArtistId, artists.Name, COUNT(albums.ArtistId), genres.Name

            FROM ((albums LEFT JOIN artists ON albums.ArtistId=artists.ArtistId)

            LEFT JOIN tracks ON tracks.AlbumId=albums.AlbumId)
```

```
 LEFT JOIN genres ON tracks.genreId=genres.GenreId

GROUP BY albums.ArtistId

HAVING genres.Name='Soundtrack';"
```

SELECT statement selects ArtistId from the albums table, artist Name from the artists table, genre name from the genre table. COUNT function is used to count the artist ids. LEFT JOIN is used as it return values from the left tables and the matched values from the right table (w3schools, *SQL left join keyword*). LEFT JOIN in this case, joins the artists and albums table using the join predicate ArtistId. Albums table is then joined with the tracks table using the join predicate AlbumId. Tracks table is joined with genres table using the join predicate GenreId. GROUP BY is used to group ArtistId. HAVING is used to specify the genre name in the genres table to be equal to “Soundtrack”. The solution works and in writer’s opinion is the best possible solution.

References:

- Aabaab 1 *et al.* (2014) *MySQL combine two columns into one column, Stack Overflow*. Available at: https://stackoverflow.com/questions/22739841/mysql-combine-two-columns-into-one-column#:~:text=try%20to%20use%20coalesce(),columns%20in%20the%20SQL%20query.&text=Eg%3A%201%2C%20vishnu%2C%209961907453,to%20get%20the%20above%20result (Accessed: October 13, 2022).

- TutorialsPoint (no date) *SQL - except clause, Tutorials Point*. Available at: https://www.tutorialspoint.com/sql/sql-except-clause.htm (Accessed: October 14, 2022).

- w3schools (no date) *SQL left join keyword*. Available at: https://www.w3schools.com/sql/sql_join_left.asp (Accessed: October 15, 2022).

- w3schools (no date) *SQL LIKE Operator, SQL like operator*. w3schools. Available at: https://www.w3schools.com/SQL/sql_like.asp (Accessed: October 12, 2022).

- w3schools (no date) *SQL Server SUBSTRING() Function, SQL Server substring() function*. Available at: https://www.w3schools.com/SQL/func_sqlserver_substring.asp (Accessed: October 13, 2022).