
# SQL as a Second Language

John HR Schuster

Version 2.1k, 01/09/2019

# Table of Contents

SQL as a Second Language (SASL) was a course that I taught many moons ago. It is being revisited now to help answer some SQL questions brought up by those whose are taking their SQL Skills to the next level.

A PDF version of this web site is available at this Link

# 1. Introduction

This SQL As a Second Language version will use the Teradata SQL syntax.

*Examples*

> A teacher, a really good teacher, is never a giver of truth; he is a guide, a pointer to truth
>
> — Bruce Lee

# 1.1. Chinook Database

The training database will be the Chinook database on music record sales.

Reference: https://github.com/lerocha/chinook-database

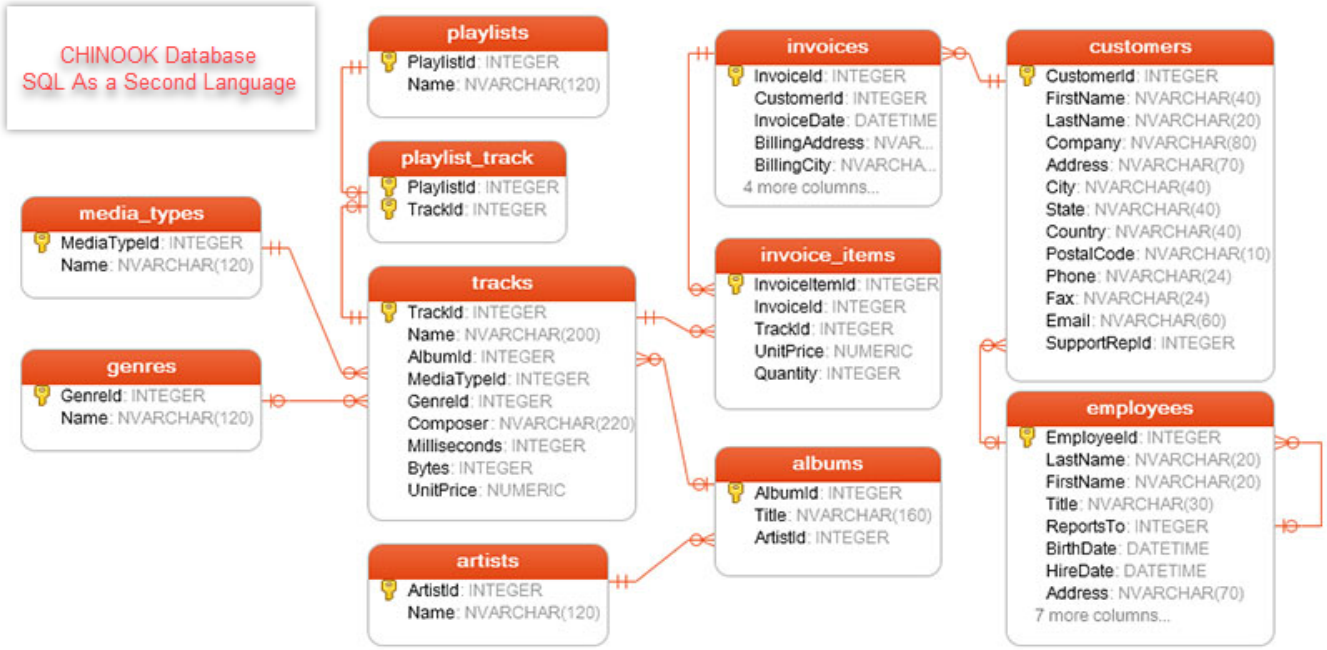


*Figure 1. Chinook Database Diagram*

## 1.1.1. Chinook Business Rules

1. Each **Artist** has 1 or more **Albums**
2. Each **Album** has just 1 **Artist**
3. Each **Album** has 1 or more **Tracks**
4. Each **Track** has just 1 **Genre**
5. Each **Track** has just 1 **Media Type**
6. **Tracks** have playtime measured in milliseconds
7. **Tracks** have disk space measured in bytes
8. Each **Employee** has just 1 Manager, who is also an **Employee**
9. Each **Customer** has 1 or more **Invoices**
10. Each **Customer** has just 1 support rep who is an **Employee**
11. Each **Invoice** has 1 or more **Invoice Lines**
12. Each **Invoice Line** is related to just 1 **Track**
13. **Invoices** span 2016 - 2018
14. Each **Playlist** has 1 or more **Tracks**
15. Not all **Genre** have **Tracks**

## 1.2. Database Objects

In this document the term `Database Object` is used to indicate one of the following Teradata database elements.

- Table (All types)
- View
- Stored Procedure
- Macro

The reason for the term `Database Object` is to eliminate the confusion that can occur when objects change. (IE: A view you created is materialized into a table, a query you wrote is changed into a store procedure)

# 2. Questions

To ask a **New question** to be added to this list, please email it to john.schuster@PhoenixWorkgroup.com.

**How can I get a count of the number of rows in some of the tables of a database?**

View this query tableCounts

**How can a get a list of the objects (Tables, View, Procedures) for a specific database?**

View This query dbObjects

**How can I find out the columns and metadata about a specific object (Table, View)?**

View this query tableColumns

**How can I find where a specific column exist in a database?**

View this query findColumn

**How do I Join two tables together?**

View this query joinQuery Part B

**How do I restrict the results by a column that `contains` some string?**

View this query joinQuery Part C

**How do I sort the results by multiple columns?**

View this query joinQuery Part C

**How do I use a sub-query to collect information?**

View this query derivedTable 1 - Part A

**How do I use a sub-query to reduce or filter information?**

View this query derivedTable 2 - Part A

**When do I use the `having` clause?**

View this query derivedTable 2 - Part D

**When is it a good idea to use a `view`?**

Visit this query Create View

**When do I use a `inner` join versus an `outer` join?**

Visit this query Join Types

**How can I best use "Prompts" in SQL Assistant?**

unlinked

# 3. SQL Overview

# 4. Queries

# 4.1. tableCounts

*SQL Example*

```
/*
=== tableCounts - Count rows in all Chinook tables

==== TOPICS

* Aggragate function count.
* UNION multiple queries.
* Sorting aka order by.

==== TIPS

* Use single quote (') to denote text.
* Use doble-quotes for renaming objects.
* Indent and one column per row.
* Put commas at the begining of the 2nd column.
* UNION allows joining of multiple queries, same number and data type of column.
* First query in union determine sizes and names of columns.
* Start small and build on to query.
* Order by can be by the ordinal number of the column
  (Column 1 is tableName, Column 2 is Rows)
*/

Select
    'Album          ' as tableName   ①
    , count(*) as "Rows" from Album  ②

UNION
Select
    'Artist' as tableName
    , count(*) from Artist

UNION
Select
    'Customer' as tableName
    , count(*) from Customer

UNION
Select
    'Employee' as tableName
    , count(*) from Employee

UNION
Select
    'Genre' as tableName
    , count(*) from Genre

UNION
```

```
Select
    'Invoice' as tableName
    , count(*) from Invoice

UNION
Select
    'Invoice Line' as tableName
    , count(*) from InvoiceLine

UNION
Select
    'Media Type' as tableName
    , count(*) from MediaType

UNION
Select
    'Playlist' as tableName
    , count(*) from Playlist

UNION
Select
    'Playlist Track' as tableName
    , count(*) from PlaylistTrack

UNION
Select
    'Track' as tableName
    , count(*) from Track

Order by 2 desc  ③
```

① First query in union determine sizes and names of columns.

② count(*) is an aggregate function

③ Order by can be by the ordinal number of the column (Column 1 is tableName, Column 2 is Rows)

SQL download link click here

*Table 1. Results*

| tableName | Rows |
|---|---|
| Playlist Track | 8715 |
| Track | 3503 |
| Invoice Line | 2240 |
| Invoice | 412 |
| Album | 347 |
| Artist | 275 |
| Customer | 59 |

| tableName | Rows |
| --- | --- |
| Genre | 25 |
| Playlist | 18 |
| Employee | 8 |
| Media Type | 5 |

Your table counts may vary from this example as the Chinook database is updated to help with lesson content.

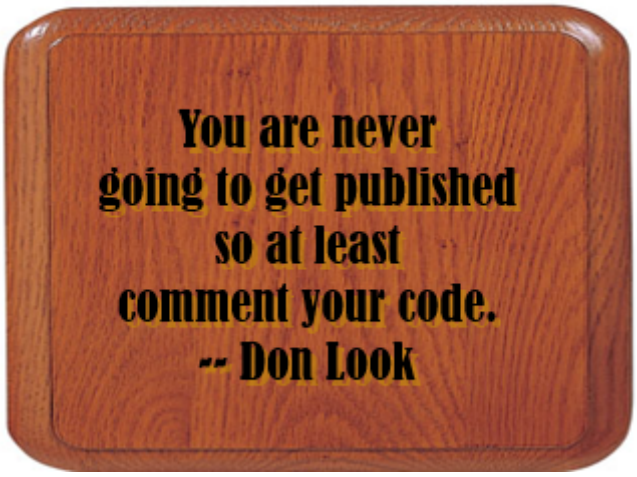


*Figure 2. Commenting your SQL*

Back to Introduction.

# 4.2. dbObjects

*SQL Example*

```
/*
=== dbObjects - Get a list of all the objects in a specific database

==== TOPICS

* This is a Teradata specific command

==== TIPS

* Change the database name from `chinook` to the database you are interested in
*/

help database chinook
```

SQL download link click here

*Table 2. Results*

| Table/View/Macro name | Kind | Comment | Protection | Creator Name | Commit Option | Transaction Log | Table/View/Macro Dictionary Name | Table/View/Macro SQL Name | Table/View/Macro Name UEscape | Creator Dictionary Name | Creator SQL Name | Creator Name UEscape |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Album | T | NULL | N | WINDUSER | N | Y | Album | Album | | WINDUSER | WINDUSER | |
| Artist | T | NULL | N | WINDUSER | N | Y | Artist | Artist | | WINDUSER | WINDUSER | |
| Customer | T | NULL | N | WINDUSER | N | Y | Customer | Customer | | WINDUSER | WINDUSER | |
| Employee | T | NULL | N | WINDUSER | N | Y | Employee | Employee | | WINDUSER | WINDUSER | |
| Genre | T | NULL | N | WINDUSER | N | Y | Genre | Genre | | WINDUSER | WINDUSER | |
| Invoice | T | NULL | N | WINDUSER | N | Y | Invoice | Invoice | | WINDUSER | WINDUSER | |
| InvoiceLine | T | NULL | N | WINDUSER | N | Y | InvoiceLine | InvoiceLine | | WINDUSER | WINDUSER | |
| mComedyList | M | NULL | F | DBC | N | Y | mComedyList | mComedyList | | DBC | DBC | |

| Table/View/Macro name | Kind | Comment | Protection | Creator Name | Commit Option | Transaction Log | Table/View/Macro Dictionary Name | Table/View/Macro SQL Name | Table/View/Macro Name UEscape | Creator Dictionary Name | Creator SQL Name | Creator Name UEscape |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MediaType | T | NULL | N | WINDUSER | N | Y | MediaType | MediaType | | WINDUSER | WINDUSER | |
| Playlist | T | NULL | N | WINDUSER | N | Y | Playlist | Playlist | | WINDUSER | WINDUSER | |
| PlaylistTrack | T | NULL | N | WINDUSER | N | Y | PlaylistTrack | PlaylistTrack | | WINDUSER | WINDUSER | |
| Track | T | NULL | N | WINDUSER | N | Y | Track | Track | | WINDUSER | WINDUSER | |
| vComedyTrack | V | NULL | F | DBC | N | Y | vComedyTrack | vComedyTrack | | DBC | DBC | |

Your database objects may vary from this example as the Chinook database is updated to help with lesson content.

Back to Introduction.

# 4.3. tableColumns

*SQL Example*

```
/*
=== tableColumns - Get a list of all the columns for a specific table or object in a
known database

==== TOPICS

* This is a Teradata specific command

==== TIPS

* Change the database name from `chinook` to the database you are interested in
*/

help table chinook.invoice
```

SQL Download link click here

*Table 3. Results*

| ColumnName | Type | Comment | Nullable | Format | Title | MaxLength | DecimalTotalDigits | DecimalFractionalDigits | RangeLow | RangeHigh | UpperCase | Table/View? | DefaultValue | CharType | IdColType | UDTName | Temporal | ColumnDictionaryName | ColumnSQLName | ColumnNameUEscape | DictionaryTitle | SQLTitle | TitleUEscape | UDTDictionaryName | UDTSQLName | UDTNameUEscape |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| InvoiceId | I | NULL | N | -(10)9 | NULL | 4 | NULL | NULL | NULL | NULL | N | T | NULL | NULL | GD | NULL | N | InvoiceId | InvoiceId | | NULL | NULL | NULL | NULL | NULL | NULL |
| CustomerId | I | NULL | N | -(10)9 | NULL | 4 | NULL | NULL | NULL | NULL | N | T | NULL | NULL | NULL | NULL | N | CustomerId | CustomerId | | NULL | NULL | NULL | NULL | NULL | NULL |

| Column Name | Type | Comment | Nullable | Format | Title | MaxLength | DecimalTotalDigits | DecimalFractionalDigits | RangeLow | RangeHigh | UpperCase | Table/View? | DefaultValue | CharType | IdColType | UDTName | Temporal | ColumnDictionaryName | ColumnSQLName | ColumnNameUEscape | DictionaryTitle | SQLTitle | TitleUEscape | UDTDictionaryName | UDTSQLName | UDTNameUEscape |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| InvoiceDate | DA | NULL | N | yyyy-mm-dd | NULL | 4 | NULL | NULL | NULL | NULL | N | T | NULL | NULL | NULL | NULL | N | InvoiceDate | InvoiceDate | | NULL | NULL | NULL | NULL | NULL | NULL |
| BillingAddress | CV | NULL | Y | X(70) | NULL | 70 | NULL | NULL | NULL | NULL | N | T | NULL | 1 | NULL | NULL | N | BillingAddress | BillingAddress | | NULL | NULL | NULL | NULL | NULL | NULL |
| BillingCity | CV | NULL | Y | X(40) | NULL | 40 | NULL | NULL | NULL | NULL | N | T | NULL | 1 | NULL | NULL | N | BillingCity | BillingCity | | NULL | NULL | NULL | NULL | NULL | NULL |
| BillingState | CV | NULL | Y | X(40) | NULL | 40 | NULL | NULL | NULL | NULL | N | T | NULL | 1 | NULL | NULL | N | BillingState | BillingState | | NULL | NULL | NULL | NULL | NULL | NULL |
| BillingCountry | CV | NULL | Y | X(40) | NULL | 40 | NULL | NULL | NULL | NULL | N | T | NULL | 1 | NULL | NULL | N | BillingCountry | BillingCountry | | NULL | NULL | NULL | NULL | NULL | NULL |
| BillingPostalCode | CV | NULL | Y | X(10) | NULL | 10 | NULL | NULL | NULL | NULL | N | T | NULL | 1 | NULL | NULL | N | BillingPostalCode | BillingPostalCode | | NULL | NULL | NULL | NULL | NULL | NULL |

| Column Name | Type | Comment | Nullable | Format | Title | MaxLength | Decimal Total Digits | Decimal Fractional Digits | RangeLow | RangeHigh | UpperCase | Table/View? | Default value | CharType | IdColType | UDTName | Temporal | Column Dictionary Name | Column SQLName | Column Name UEscape | Dictionary Title | SQLTitle | Title UEscape | UDT Dictionary Name | UDTSQLName | UDTName UEscape |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total | D | NULL | N | ----------.99 | NULL | 8 | 10 | 2 | NULL | NULL | N | T | NULL | NULL | NULL | NULL | N | Total | Total | | NULL | NULL | NULL | NULL | NULL | NULL |

ColumnType defined the data type of a column. Reference: [ColumnType]

Back to Introduction.

# 4.4. findColumn

*SQL Example*

```
/*
=== findColumn - Find out what objects (Table, View) where a specific named column
exist

==== TOPICS

* This is a Teradata specific command
* The list of columns are the ones that are the most important,
there are many other columns available. Use a single column name (*) to see them all.

==== TIPS

* CoumnTypes Reference:
http://developer.teradata.com/doc/connectivity/tdnetdp/14.00/webhelp/DataTypeMappings.
html
* Replace `CustomerID` with the column you are interested in

*/

select
    ColumnName
    ,DatabaseName
    ,TableName
    ,ColumnFormat
    ,ColumnType
    ,ColumnLength
from dbc.columnsX
where ColumnName = 'CustomerID'
```

SQL Download link click here

*Table 4. Results*

| ColumnName | DatabaseName | TableName | ColumnFormat | ColumnType | ColumnLength |
|---|---|---|---|---|---|
| CustomerId | Chinook | Invoice | -(10)9 | I | 4 |
| CustomerId | Chinook | Customer | -(10)9 | I | 4 |

The ColumnType identifies the data type of the column. Reference: Column Type

Back to Introduction.

# 4.5. joinQuery Part A

**Objective:** Get a list of tracks from the **Artist** `Aerosmith` where the **Composer** is `Joe Perry`.

**Approach:** Begin with a simple query to get one element of the objective. In this query we want to get the **ArtistID** for `Aerosmith`

*SQL Example*

```
/*
=== joinQuery - Multiple Table join with result restriction

==== TOPICS

* This query joins multiple table together

==== TIPS

* Table alias shorten the typing needed to complete the query
1-2 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
* Results being restricted with a a `where` clause
*/

Select
   AR.ArtistID
   ,AR.name as artistName  ①
From Artist AR  ②
where artistName = 'Aerosmith'
```

① Column alias, notice name is `artistName`

② Table alias, use alias on every instance of columns from that table

SQL Download link click here

*Table 5. Results*

| ArtistId | artistName |
|---|---|
| 3 | Aerosmith |

Back to Introduction.

# 4.6. joinQuery Part B

**Objective:** Get a list of tracks from the **Artist** `Aerosmith` where the **Composer** is `Joe Perry`.

**Approach:** Add on to the initial query to get a list of all the **Albums** for the **Artist** `Aerosmith`

*SQL Example*

```
/*
=== joinQuery - Multiple Table join with result restriction

Part B - Add the second table for the Join

==== TOPICS

* This query joins multiple table together

==== TIPS

* Table alias shorten the typing needed to complete the query
1-2 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Primary Key and Foreign Key matching columns should usew the same name

*/

Select
   AR.ArtistID
   ,AL.AlbumID
   ,AR.name as artistName  ①
   ,AL.AlbumTitle
From Artist AR  ②

inner Join Album AL ②
on AR.ArtistID = AL.ArtistID  ③

where artistName = 'Aerosmith'
```

① Column alias, notice name is `artistName`

② Table alias, use alias on every instance of columns from that table

③ The common column used to join the two tables together

SQL Download link click here

*Table 6. Results*

| ArtistId | AlbumId | artistName | AlbumTitle |
|---|---|---|---|
| 3 | 5 | Aerosmith | Big Ones |

Back to Introduction.

# 4.7. joinQuery Part C

**Objective:** Get a list of tracks from the **Artist** `Aerosmith` where the **Composer** is `Joe Perry`.

**Approach:** Complete the request by joining the `Title` table and a row restriction using the `like` with the wildcard `%` character.

*SQL Example*

```
/*
=== joinQuery - Multiple Table join with result restriction

Part C - Add the third table and composer restriction

==== TOPICS

* List of all the `Aerosmith` tracks that `Joe Perry` was one of the cpmposers
* This query joins multiple tables together
* Primary and Foreign keys are system generated and always exist

==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Primary Key and Foreign Key matching columns should usew the same name
* `like` used to find a column that contains some string`, note the % indicating any
number of characters before/after
* Order by can use either 1. Actual column names, 2. Alias column names, 3. Ordinal
position is results list
* Alias column names use lowerUpper naming to identify them as alias
only works because original names are camelback naming.

*/

Select
   AR.ArtistID
   ,AL.AlbumID
   ,T.TrackID
   ,AR.name as artistName  ①
   ,AL.AlbumTitle
   ,T.Name as trackName    ①
   ,T.Composer
From Artist AR  ②

inner Join Album AL ②
on AR.ArtistID = AL.ArtistID  ③
```

```
inner join Track T
on AL.AlbumID = T.AlbumID

where AR.Name = 'Aerosmith'
  and T.Composer like '%Joe Perry%'  ④

order by AL.AlbumTitle
         ,trackName  ⑤
-- order by 5,6 ⑥
```

① Column alias, notice name is `artistName`

② Table alias, use alias on every instance of columns from that table

③ The common column used to join the two tables together

④ The `like` with wildcard character `%` used on `Composer` column

⑤ The order by has both actual column name `AL.AlbumTitle` and an alias column `trackName`

⑥ Shows alternative method of `order by` using ordinal column numbers

SQL Download link click here

*Table 7. Results*

| ArtistId | AlbumId | TrackId | artistName | AlbumTitle | trackName | Composer |
|---|---|---|---|---|---|---|
| 3 | 5 | 31 | Aerosmith | Big Ones | Blind Man | Steven Tyler, Joe Perry, Taylor Rhodes |
| 3 | 5 | 34 | Aerosmith | Big Ones | Crazy | Steven Tyler, Joe Perry, Desmond Child |
| 3 | 5 | 29 | Aerosmith | Big Ones | Cryin' | Steven Tyler, Joe Perry, Taylor Rhodes |
| 3 | 5 | 27 | Aerosmith | Big Ones | Dude (Looks Like A Lady) | Steven Tyler, Joe Perry, Desmond Child |
| 3 | 5 | 35 | Aerosmith | Big Ones | Eat The Rich | Steven Tyler, Joe Perry, Jim Vallance |
| 3 | 5 | 37 | Aerosmith | Big Ones | Livin' On The Edge | Steven Tyler, Joe Perry, Mark Hudson |
| 3 | 5 | 24 | Aerosmith | Big Ones | Love In An Elevator | Steven Tyler, Joe Perry |
| 3 | 5 | 25 | Aerosmith | Big Ones | Rag Doll | Steven Tyler, Joe Perry, Jim Vallance, Holly Knight |
| 3 | 5 | 23 | Aerosmith | Big Ones | Walk On Water | Steven Tyler, Joe Perry, Jack Blades, Tommy Shaw |
| 3 | 5 | 26 | Aerosmith | Big Ones | What It Takes | Steven Tyler, Joe Perry, Desmond Child |

Back to Introduction.

# 4.8. Join Types

The relationships between database objects are defined using `join` statements.

Two database objects are joined using one or more common columns between them.

## 4.8.1. Inner Join

An `Inner Join` can be used when you know there is at least one row in each of the two tables being joined. The business rules or the data model can help define when this relationship occurs.

An `Inner Join` can be used to restrict/filter a large amount of rows from one table by a limited number of rows in the second table.

An `Inner Join` can be used to replace a `where` clause with a list of values or a sub-query. The `Inner Join` can be much more performant than a `where` clause when there are a large number of restriction/filter rows.

## 4.8.2. Outer Join

- An `Outer Join` can be used when you know that there may no or is not one row in each of the tables being joined.
- An `Outer Join` returns **ALL** of the rows from one table and any matching rows from the second table.
- All `Outer Joins` has a direction, left outer join or right outer join. Think of it as one object on the left and the second object on the right.
- The first `Select` object in the query is the `Left` object by default.
- The outer joined object is the `right` object.
- The direction of the outer join determines which object is the **All** object
- A `Right Outer Join` returns **ALL** the rows from the object on the **RIGHT** and any matching rows from the left object.
- A `Left Outer Join` returns **ALL** the rows from the object on the **LEFT** and any matching rows from the right object.
- The `on` clause of the join does **NOT** indicate the direction of the join. `on G.GenreID = T.GenreID` and `on T.GenreID = G.GenreID`


*Figure 3. Right Outer Join*

The same query above could be written reversing the order in which the objects are referenced to show a Left Outer join.


*Figure 4. Left Outer Join*

*SQL Example*

```
/*
=== joinTypes - Inner Join and Outer Join


==== TOPICS

* List of Track count for *ALL* Genres
* Multiple queries in one statment
* Inner Join
* Outer Join


==== TIPS


*/

-- List of *all* Genres

select
    G.GenreID
```

```
    ,G.Name as genreName  ①
from Genre G
order by genreName;
-- Order by 2     ②


-- Track Count by Genre (Inner Join)
Select
    G.GenreID
    ,G.Name as genreName  ①
    ,count(T.TrackID) as numberTracks ③
from Track T

inner Join Genre G   ④
on G.GenreID = T.GenreID

Group by
    G.GenreID
    ,genreName   ⑤
order by
    genreName;  ⑤


-- Track Count by Genre (Inner Join)  Left
Select
    G.GenreID
    ,G.Name as genreName  ①
    ,count(T.TrackID) as numberTracks ③
from Track T

left outer Join Genre G   ⑥
on G.GenreID = T.GenreID

Group by
    G.GenreID
    ,genreName   ⑤
order by
    genreName;  ⑤
```

SQL Download link click here

*Table 8. Results*

| GenreId | genreName |
|---|---|
| 23 | Alternative |
| 4 | Alternative & Punk |
| 6 | Blues |
| 11 | Bossa Nova |
| 24 | Classical |

| GenreId | genreName |
|---|---|
| 32 | Club |
| 22 | Comedy |
| 21 | Drama |
| 30 | Dub Step |
| 12 | Easy Listening |
| 15 | Electronica/Dance |
| 13 | Heavy Metal |
| 17 | Hip Hop/Rap |
| 2 | Jazz |
| 7 | Latin |
| 3 | Metal |
| 25 | Opera |
| 9 | Pop |
| 14 | R&B/Soul |
| 8 | Reggae |
| 1 | Rock |
| 5 | Rock And Roll |
| 20 | Sci Fi & Fantasy |
| 18 | Science Fiction |
| 10 | Soundtrack |
| 31 | Trance |
| 19 | TV Shows |
| 16 | World |

Back to Introduction.

## 4.9. derivedTable 1 - Part A

**Objective:** Get a list by Artist that includes number of albums, number of tracks, total artist minutes and average minutes per track.

**Approach:** Use a set of derived queries (aka sub-queries) to get the parts of the request and assemble the parts together in the main query.

This collection approach is typically used when the parts being collected come from varied databases or putting them in a single query makes it difficult to compose the query.

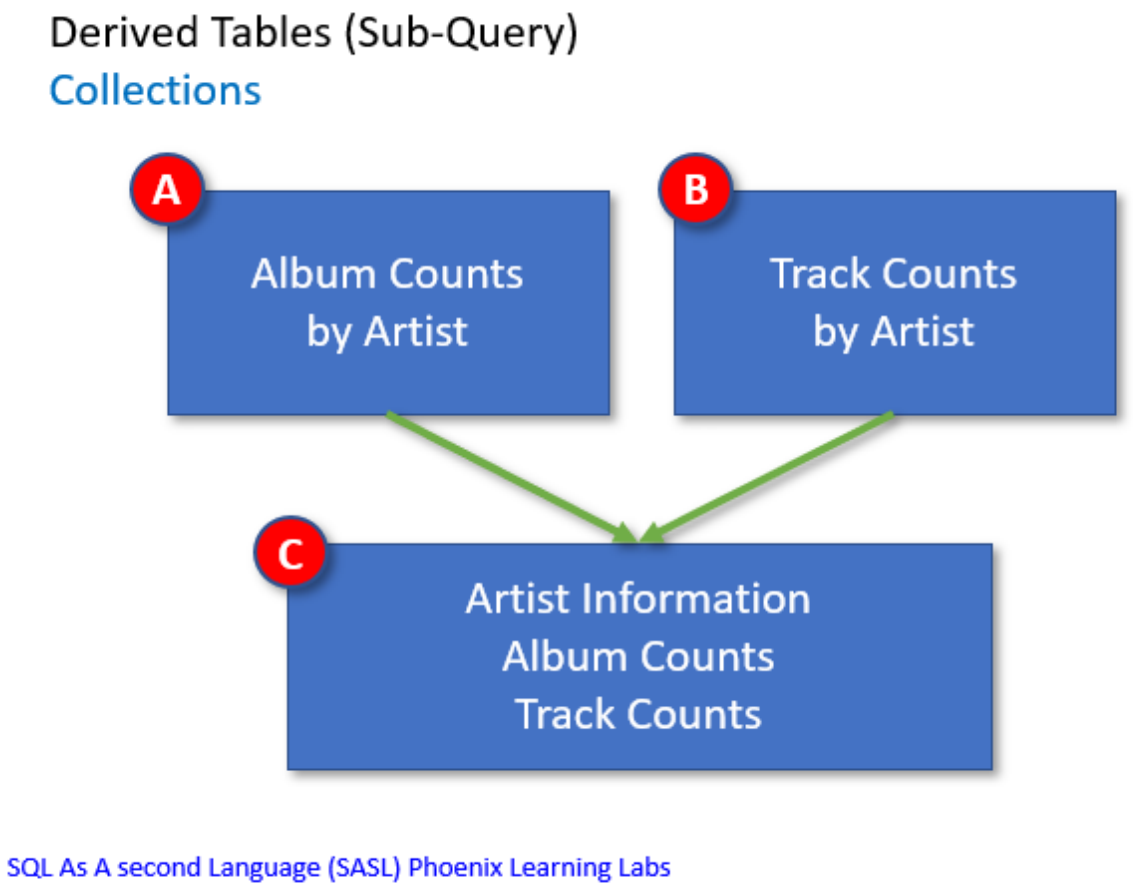


*Figure 5. derivedTable Collection*

Part A gets the number of albums by artist. It is the (Album Part) we need for the main query.

This request could be done using a single query. The derived tables are being used here, to show how they can be used to build parts of the results, that are assembled in a main query later.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*
=== derivedTable - Gets the list of artists and album counts
```

```
Part A - Gets the list of artists with album count
this will become one of the derived tables in the final query

==== TOPICS

* Table alias
* Column Alias
* Join objects (Table, View, Derived Table) together
* Derived Table (Sub-Query)
* Count aggregate


==== TIPS

.Table Alias
* Table alias shorten the typing needed to complete the query,
1-3 character alias recommended.
* Good practice is to have table alias used everywhere in the query.

.Column Alias
* Column alias help to give meaningful, non conflicting names.
* Good practice, alias column names use lowerUpper naming
to identify them as alias (albumCount)
* Good practic, Use the same alias across all queries.

.Naming practice
* Good practice, System Generated Primary Index column name be
include with `ID` suffix.
* Good practice, Primary Key and Foreign Key join columns
should use the same name.

.Join requirements
* Coulmns for join must match in data type and size.

.Derived Tables (Sub-Queries)
* Start with a regular query until you get the results you need.
* Enclose the regular query in `(...)` and assign a table alias name.
* Join the derived table to another.

.Aggregate Functions
* Functions used to compress or limit the number of rows into a single row.
* `Count(?)` Counts the number of rows, duplicates included in the count.

*/


select
    AR.ArtistID
    ,count(AL.AlbumID) as albumCount  ① ②
from Album AL ③
```

```
INNER JOIN Artist AR  ③
on AR.ArtistID = AL.ArtistID ④

Group by AR.ArtistID  ⑤
```

① Column Alias

② Aggregate `Count` function

③ Table alias

④ Common column to join two tables

⑤ Any non-aggregate column must be included in `group by`

SQL Download link click here

*Table 9. Results*

| ArtistId | albumCount |
|---|---|
| 223 | 1 |
| 265 | 1 |
| 19 | 2 |
| 122 | 1 |
| 80 | 2 |
| 244 | 1 |
| 202 | 1 |
| 101 | 2 |
| 59 | 3 |
| 141 | 1 |
| 242 | 1 |
| 263 | 1 |
| 221 | 1 |
| 120 | 1 |
| 78 | 1 |
| 99 | 2 |
| 200 | 1 |
| 17 | 1 |
| 57 | 1 |

Only 20 rows of the result being shown. The result set has 204 rows, one for each artist.

Back to Introduction.

# 4.10. derived Table 1 -Part B

**Objective:** Get a list by Artist that includes number of albums, number of tracks, total artist minutes and average minutes per track.

**Approach:** Use a set of derived queries (aka sub-queries) to get the parts of the request and assemble the parts together in the main query.

This collection approach is typically used when the parts being collected come from varied databases or putting them in a single query makes it difficult to compose the query.

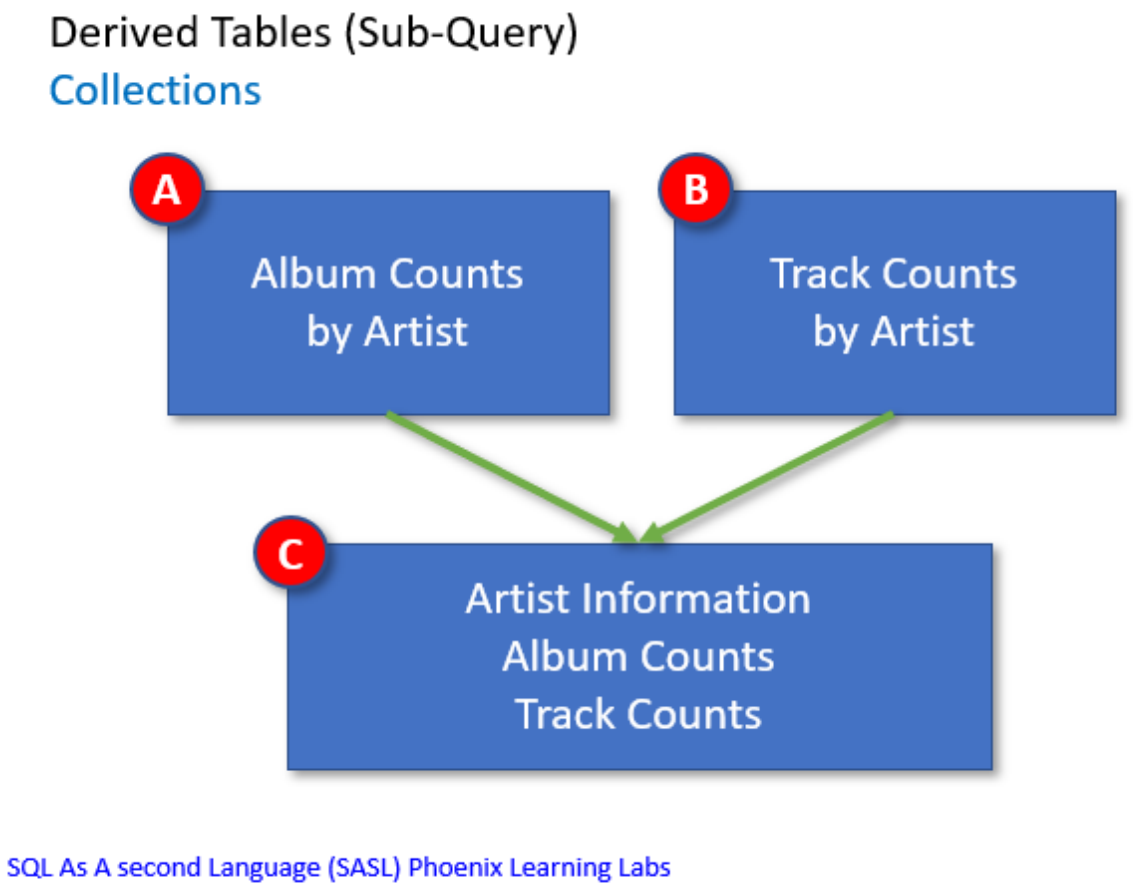


*Figure 6. derivedTable Collection*

This part gets the number of tracks by artist and calculates the number of minutes total and average minutes per track from the milliseconds. It is the second part (Tracks Part) we need for the main query.

This request could be done using a single query. The derived tables are being used here, to show how they can be used to build parts of the results, that are assembled in a main query later.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*
=== derivedTable - The derived table collects data on one topic (Tracks)

Part B -Build the query that will become one of the derived tables

==== TOPICS

* Count the number of tracks, total minutes, and average minutes per track
* Aggregation functions to be used for count and avg


==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Primary Key and Foreign Key matching columns should usew the same name
* Alias column names use lowerUpper naming to identify them as alias (trackCount,
totalMinutes, avgMinutes)

*/


Select
    AL.ArtistID
    ,count(T.TrackID) as trackCount
    ,sum(T.Milliseconds) / 60000.00 as totalMinutes  ①
    ,avg(T.Milliseconds) / 60000.00 as avgMinutes    ②
from Album AL

inner Join Track T
on T.AlbumID = AL.AlbumID

group by AL.ArtistID
```

SQL Download link click here

*Table 10. Results*

| ArtistId | trackCount | totalMinutes | avgMinutes |
|---|---|---|---|
| 72 | 15 | 54.25 | 3.61635666666667 |
| 179 | 12 | 57.47 | 4.78950277777778 |
| 96 | 12 | 50.27 | 4.18938194444444 |

| ArtistId | trackCount | totalMinutes | avgMinutes |
|---|---|---|---|
| 227 | 1 | 2.95 | 2.94851666666667 |
| 144 | 20 | 76.33 | 3.81672 |
| 88 | 42 | 205.93 | 4.9029876984127 |
| 203 | 1 | 5.94 | 5.94043333333333 |
| 99 | 31 | 136.11 | 4.39078279569893 |
| 275 | 1 | 3.43 | 3.43341666666667 |
| 230 | 1 | 3.30 | 3.30106666666667 |
| 21 | 56 | 233.26 | 4.16541994047619 |
| 139 | 30 | 128.59 | 4.28646833333333 |
| 238 | 1 | 6.52 | 6.51666666666667 |
| 246 | 1 | 2.22 | 2.21553333333333 |
| 56 | 14 | 48.92 | 3.49458571428571 |
| 83 | 14 | 47.37 | 3.38387619047619 |
| 91 | 20 | 70.19 | 3.50936833333333 |
| 155 | 19 | 67.75 | 3.56591228070175 |
| 115 | 12 | 60.82 | 5.06811111111111 |

① This calculated or derived field is the sum of time (Milliseconds) for all tracks for this artist. Dividing by 60,000.00 translates this into minutes.

② This calculated or derived field is the average of the time (Milliseconds) for all tracks for this artist. Dividing by 60,000.00 translate this into minutes.

Notice that the division was done using a fractional number 60000.00 which returns the results with two decimal points of precision. If we had used an integer number 60000 we would have gotten an whole number back and lost the two decimal points of precision.

Only 20 rows of the result being shown. The result set has 204 rows, one for each artist.

Back to Introduction.

## 4.11. derivedTable 1 - Part C

**Objective:** Get a list by Artist that includes number of albums, number of tracks, total artist minutes and average minutes per track.

**Approach:** Use a set of derived queries (aka sub-queries) to get the parts of the request and assemble the parts together in the main query.

This collection approach is typically used when the parts being collected come from varied databases or putting them in a single query makes it difficult to compose the query.

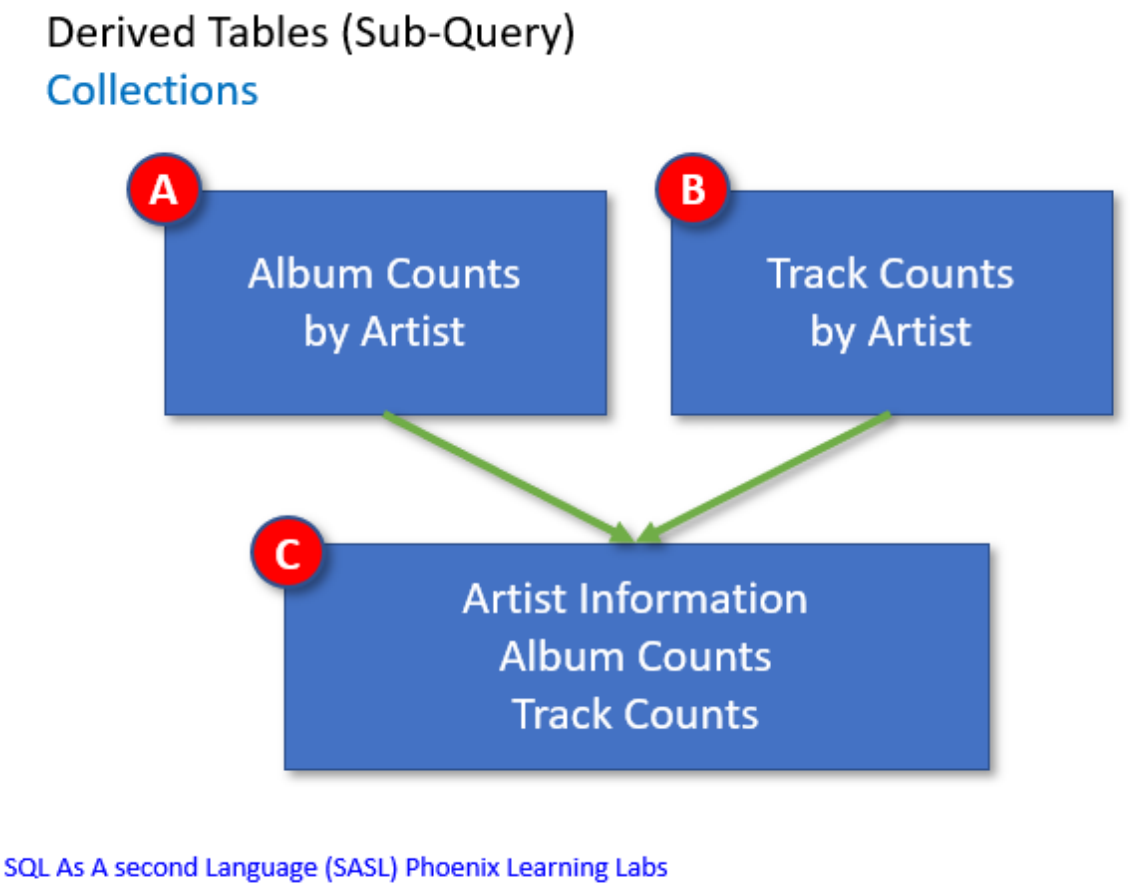


*Figure 7. derivedTable Collection*

This part assembles the album part (Part A) and tracks part (Part B) with artist information for the main or final query.

This request could be done using a single query. The derived tables are being used here, to show how they can be used to build parts of the results, that are assembled in a main query later.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*
```

```
=== derivedTable - The derived table collects data on one topic (Albums)

Part C - Build the main query and include the two derived tables

==== TOPICS

* Join multiple tables
* Aggregation functions to be used for count
* Table and Column Alias

=== ASSUMPTIONS

* Every Artist has at least one album

==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used.
Use the same alias across all queries.
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Good practice, alias column names use lowerUpper naming to identify them as alias
(albumCount)
* Primary Key and Foreign Key matching columns should usew the same name

*/

Select
    AR.ArtistID
    ,AR.Name as artistName
    ,AC.albumCount   ①
    ,TC.totalMinutes
    ,cast(TC.avgMinutes as decimal(5,2))  as averageMinutes  ②
from Artist AR

inner join
    (                   ③
    select
        AR.ArtistID
        ,count(AL.AlbumID) as albumCount  ①
    from Album AL

    INNER JOIN Artist AR
    on AR.ArtistID = AL.ArtistID

    Group by AR.ArtistID
    ) as AC               ④
on AC.ArtistID = AR.ArtistID
```

```
inner Join
    (                       ③
    Select
        AL.ArtistID
        ,count(T.TrackID) as trackCount
        ,sum(T.Milliseconds) / 60000.00 as totalMinutes  ⑤
        ,avg(T.Milliseconds) / 60000.00 as avgMinutes    ⑥
    from Album AL

    inner Join Track T
    on T.AlbumID = AL.AlbumID

    group by AL.ArtistID
    ) as TC                 ④
on TC.ArtistID = AR.ArtistID

Order by artistName
```

① A column from the derived table (sub-query) being used in the result

② CAST being used to reformat a decimal number to 5,2. Five digits total, two digits precision

③ The beginning of a derived table (sub-query) starts with a (

④ The end of the derived table (sub-query) is marked with a ) and a name for the derived table. The derived table name is used to qualify the use of the columns in the main query.

⑤ This calculated or derived field is the sum of time (Milliseconds) for all tracks for this artist. Dividing by 60,000.00 translates this into minutes.

⑥ This calculated or derived field is the average of the time (Milliseconds) for all tracks for this artist. Dividing by 60,000.00 translate this into minutes.

SQL Download link click here

*Table 11. Results*

| ArtistId | artistName | albumCount | totalMinutes | averageMinutes |
|---|---|---|---|---|
| 230 | Aaron Copland & London Symphony Orchestra | 1 | 3.30 | 3.30 |
| 202 | Aaron Goldberg | 1 | 4.45 | 4.45 |
| 1 | AC/DC | 2 | 80.89 | 4.49 |
| 214 | Academy of St. Martin in the Fields & Sir Neville Marriner | 1 | 7.75 | 3.88 |

| ArtistId | artistName | albumCount | totalMinutes | averageMinutes |
|---|---|---|---|---|
| 215 | Academy of St. Martin in the Fields Chamber Ensemble & Sir Neville Marriner | 1 | 5.82 | 5.82 |
| 222 | Academy of St. Martin in the Fields, John Birch, Sir Neville Marriner & Sylvia McNair | 1 | 4.32 | 4.32 |
| 257 | Academy of St. Martin in the Fields, Sir Neville Marriner & Thurston Dart | 1 | 3.77 | 3.77 |
| 2 | Accept | 2 | 20.01 | 5.00 |
| 260 | Adrian Leaper & Doreen de Feis | 1 | 9.46 | 9.46 |
| 3 | Aerosmith | 1 | 73.53 | 4.90 |
| 197 | Aisha Duo | 1 | 9.23 | 4.62 |
| 4 | Alanis Morissette | 1 | 57.52 | 4.42 |
| 206 | Alberto Turco & Nova Schola Gregoriana | 1 | 4.09 | 4.09 |
| 5 | Alice In Chains | 1 | 54.16 | 4.51 |
| 252 | Amy Winehouse | 2 | 93.00 | 4.04 |
| 209 | Anne-Sophie Mutter, Herbert Von Karajan & Wiener Philharmoniker | 1 | 3.32 | 3.32 |
| 243 | Antal Doráti & London Symphony Orchestra | 1 | 6.87 | 6.87 |
| 6 | Antônio Carlos Jobim | 2 | 118.81 | 3.83 |
| 7 | Apocalyptica | 1 | 44.52 | 5.57 |

Only 20 rows of the result being shown. The result set has 204 rows, one for each artist.

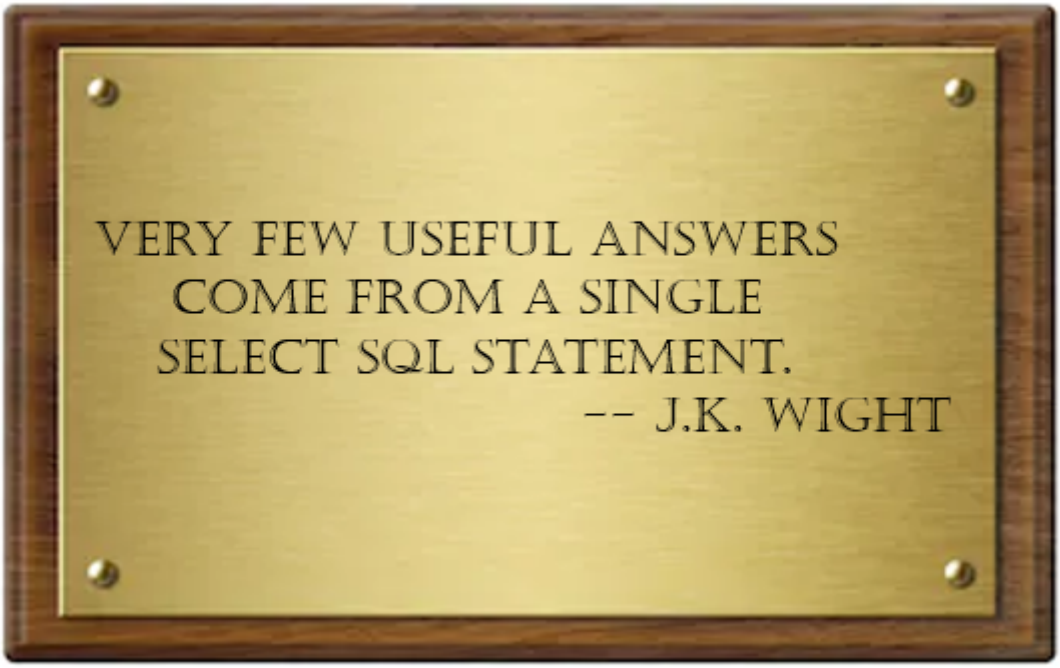


*Figure 8. Multiple SQL Selects*

Back to Introduction.

# 4.12. derivedTable 2 - Part A

**Objective:** The **Top Order Tracks** is a list of tracks from orders in a year that are $5 or more. What other tracks across all orders share a track from the **Top Order Tracks** list?

**Approach:** Use a set of derived tables (sub-queries) to create progressive filters to get the result

- Part A - List of invoices for one year that are $5 or more
- Part B - Get the list of tracks for the invoices in the Part A, this is the **Top Order Tracks** list
- Part C - Find all invoices that have a track from Part B the **Top order Tacks** list.
- Part D - Find all tracks, across all invoices, with a match in the **Top Order List** and count them

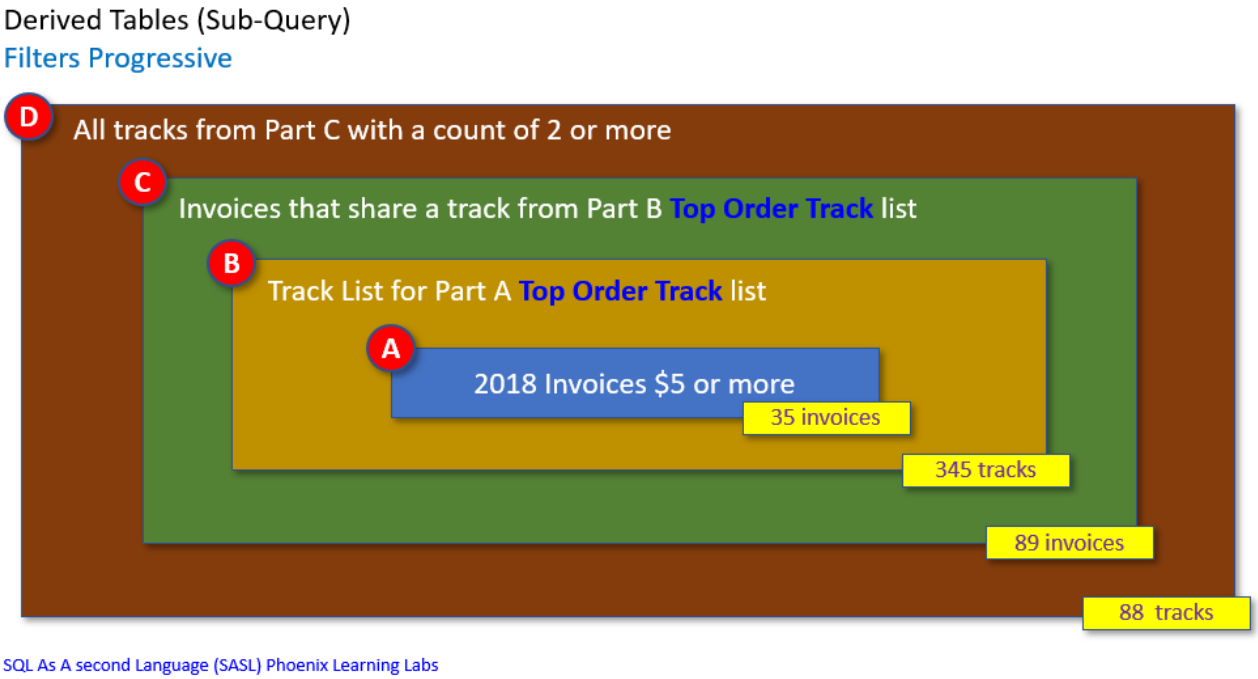


*Figure 9. derivedTable Filter*

This request could be done using a single query. The derived tables are being used here, to show how they can be used as a filter or reduce the number of rows in the result.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*
=== derivedTable  List of invoice numbers for orders $5 or more

Part A - Which invoices have a totoal of $5 or more.

==== TOPICS

* Join multiple tables
* Aggregation functions to be used for count
* Table Alias
* Between function
* Date representation


==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used.
Use the same alias across all queries.
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Good practice, alias column names use lowerUpper naming to identify them as alias
(albumCount)
* Primary Key and Foreign Key matching columns should usew the same name

Invoice numbers for 2018 at or over $5
*/

select
--  i.customerid  ①
  I.InvoiceID
--  ,i.InvoiceDate  ①
--  ,i.total  ①
from Invoice I

where I.InvoiceDate between date '2018-01-01' and date '2018-12-31' ② ③
and I.total >= 5.00
```

① Fields used during the testing of the derived table (sub-query) two hypens `--` used to comment out a line

② `between` function requires starting value `and` ending value

③ `date` indicates a date field, Teradata standard format is `yyyy-mm-dd`

SQL Download link click here

*Table 12. Results*

| InvoiceId |
|---|
| 383 |
| 404 |
| 362 |
| 339 |
| 341 |
| 375 |
| 402 |
| 354 |
| 381 |
| 333 |
| 360 |
| 411 |
| 396 |
| 390 |
| 348 |
| 369 |
| 367 |
| 409 |
| 346 |

Only 20 rows of the result being shown. The result set has 35 rows, one for each invoice of $5 or more.

Back to Introduction.

# 4.13. derivedTable 2 - Part B

**Objective:**

**Approach:**

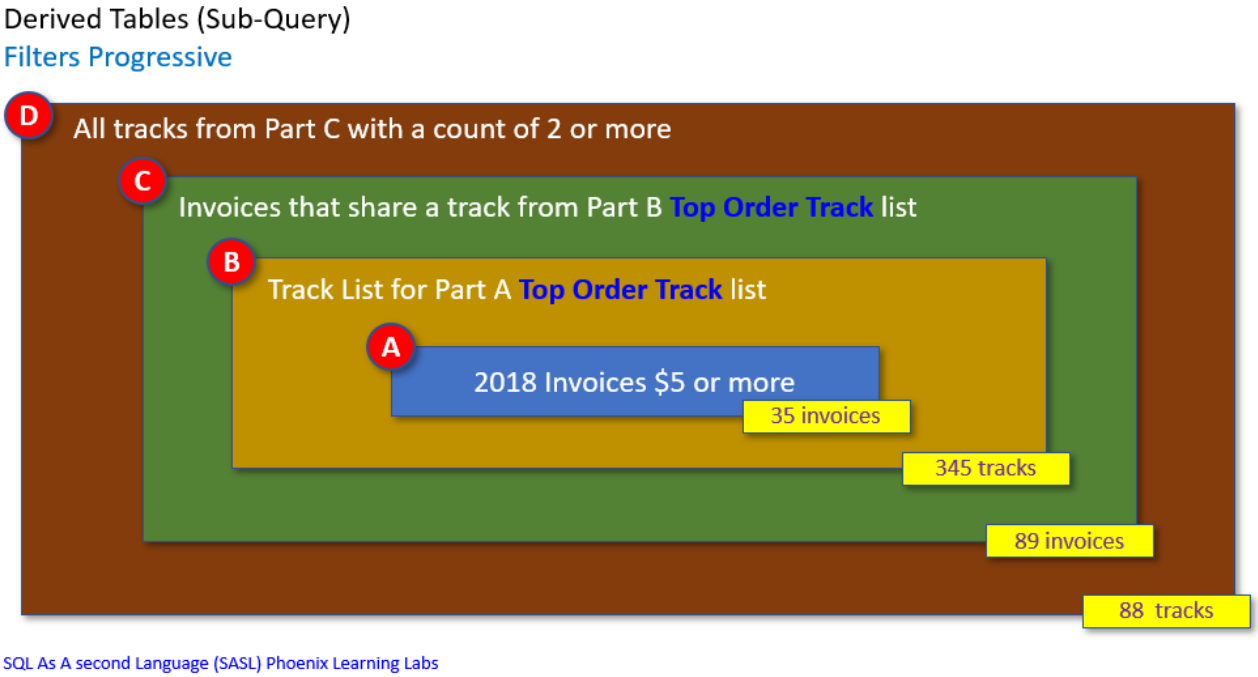


*Figure 10. derivedTable Filter*

This request could be done using a single query. The derived tables are being used here, to show how they can be used as a filter or reduce the number of rows in the result.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*
=== derivedTable  List of tracks on the invoice numbers for orders $5 or more for year
2018

Part B - List of tracks on the 2018 orders $5 or larger.

==== TOPICS

* Distinct to elminate duplicates
* Join multiple tables
* Derived table or sub-query
* Aggregation functions to be used for count
* Table Alias
* Between function
* Date representation


==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used.
Use the same alias across all queries.
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Good practice, alias column names use lowerUpper naming to identify them as alias
* Primary Key and Foreign Key matching columns should usew the same name


\
*/

Select Distinct     ⑤
    IL.TrackID
from invoiceline IL

inner join
    ( -- 2018 Invoices $5 or more    ①
    select
        I.InvoiceID
    from Invoice I
    where I.InvoiceDate between date '2018-01-01' and date '2018-12-31' ② ③
      and I.total >= 5.00
    ) as RI  ④
on RI.InvoiceID = IL.InvoiceID
```

SQL Download link click here

*Table 13. Results*

| TrackId |
|---|
| 1422 |
| 711 |
| 2085 |
| 735 |
| 2609 |
| 807 |
| 2681 |
| 2109 |
| 3109 |
| 1235 |
| 449 |
| 1307 |
| 473 |
| 1807 |
| 521 |
| 1823 |
| 2299 |
| 949 |
| 2323 |

① Derived Table (Sub-query) begins with a `(` then the sub-query SQL

② `between` function requires starting value `and` ending value

③ `date` indicates a date field, Teradata standard format is `yyyy-mm-dd`

④ The end of the derived table (sub-query) is marked with a `)` and a name for the derived table `RI` Reduced Invoices. The derived table name is used to qualify the use of the columns in the main query.

⑤ `DISTINCT` used to elminate duplicates, applied to the entire result set

Only 20 rows of the result being shown. The result set has 345 rows, one for each track of a 2018 morder of $5 or more.

Back to Introduction.

# 4.14. derivedTable 2 - Part C

**Objective:**

**Approach:**

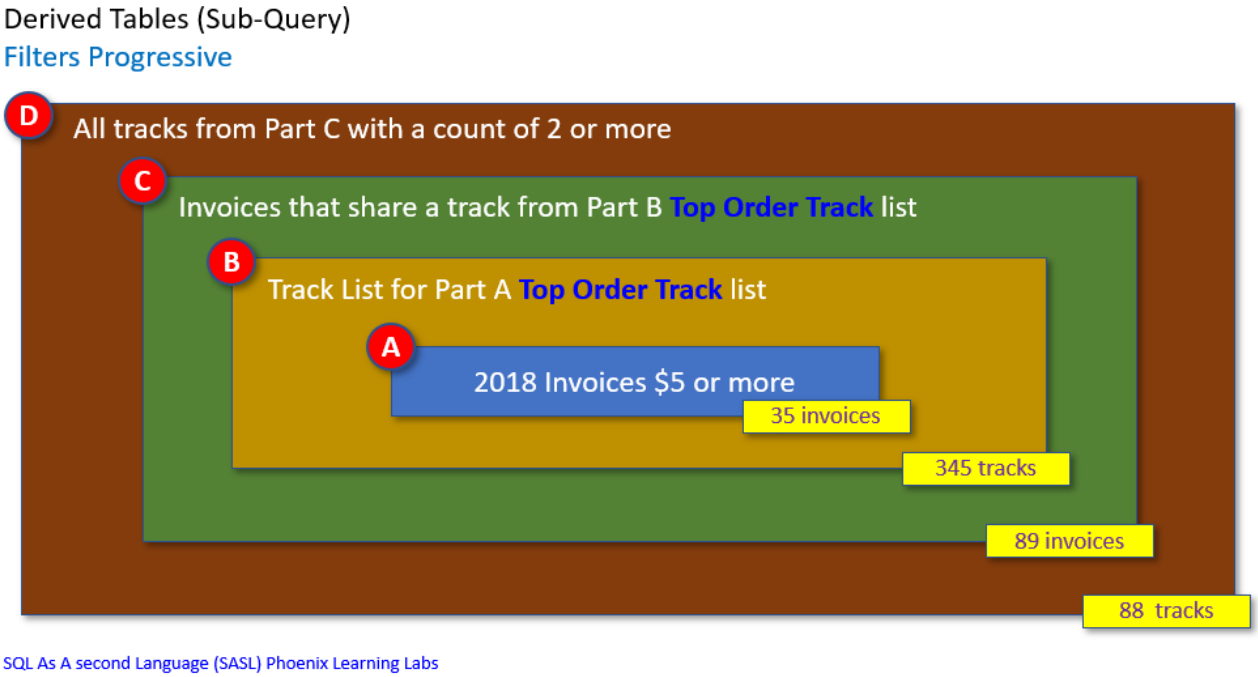


*Figure 11. derivedTable Filter*

This request could be done using a single query. The derived tables are being used here, to show how they can be used as a filter or reduce the number of rows in the result.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*
=== derivedTable  List of invoices that share a track with another list

Part C - List of invoices that share a track from the 2018 orders $5 or larger (*Top
Order Tracks* list*)

==== TOPICS

* Distinct to elminate duplicates
* Join multiple tables
* Derived table or sub-query
* Aggregation functions to be used for count
* Table Alias
* Between function
* Date representation
```

```
==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used.
Use the same alias across all queries.
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Good practice, alias column names use lowerUpper naming to identify them as alias
* Primary Key and Foreign Key matching columns should usew the same name
*/

Select distinct  ①
    IL.InvoiceID
from InvoiceLine IL

inner join
    ( -- Track list for 2018 invoice $5 or more  ②
    Select DISTINCT  ①
        IL.TrackID
    from invoiceline IL

    inner join
        ( -- 2018 Invoices $5 or more   ③
        select
            I.InvoiceID
        from Invoice I
        where I.InvoiceDate between date '2018-01-01' and date '2018-12-31'  ⑦ ⑧
          and I.total >= 5.00
        ) as RI  ④
    on RI.InvoiceID = IL.InvoiceID

    ) TL  ⑤
on TL.TrackID = IL.TrackID  ⑥
```

① DISTINCT used to eliminate duplicates, applied to the entire result set

② Derived Table (Sub-query) begins with a ( then the sub-query SQL

③ Derived Table (Sub-query) begins with a ( then the sub-query SQL

④ The end of the derived table (sub-query) is marked with a ) and a name for the derived table RI Reduced Invoices. The derived table name is used to qualify the use of the columns in the main query.

⑤ The end of the derived table (sub-query) is marked with a ) and a name for the derived table TL Track List. The derived table name is used to qualify the use of the columns in the main query.

⑥ Filter for only tracks that match tracks in the TL trackList **Top OrderTrack** list

⑦ `between` function requires starting value `and` ending value

⑧ `date` indicates a date field, Teradata standard format is `yyyy-mm-dd`

SQL Download link click here

*Table 14. Results*

| InvoiceId |
|---|
| 163 |
| 88 |
| 361 |
| 334 |
| 198 |
| 171 |
| 150 |
| 32 |
| 348 |
| 369 |
| 396 |
| 409 |
| 134 |
| 155 |
| 169 |
| 353 |
| 185 |
| 190 |
| 383 |

Only 20 rows of the result being shown. The result set has ??? rows, one for each track of a 2018 morder of $5 or more.

Back to Introduction.

# 4.15. derivedTable 2 - Part D

**Objective:**

**Approach:**

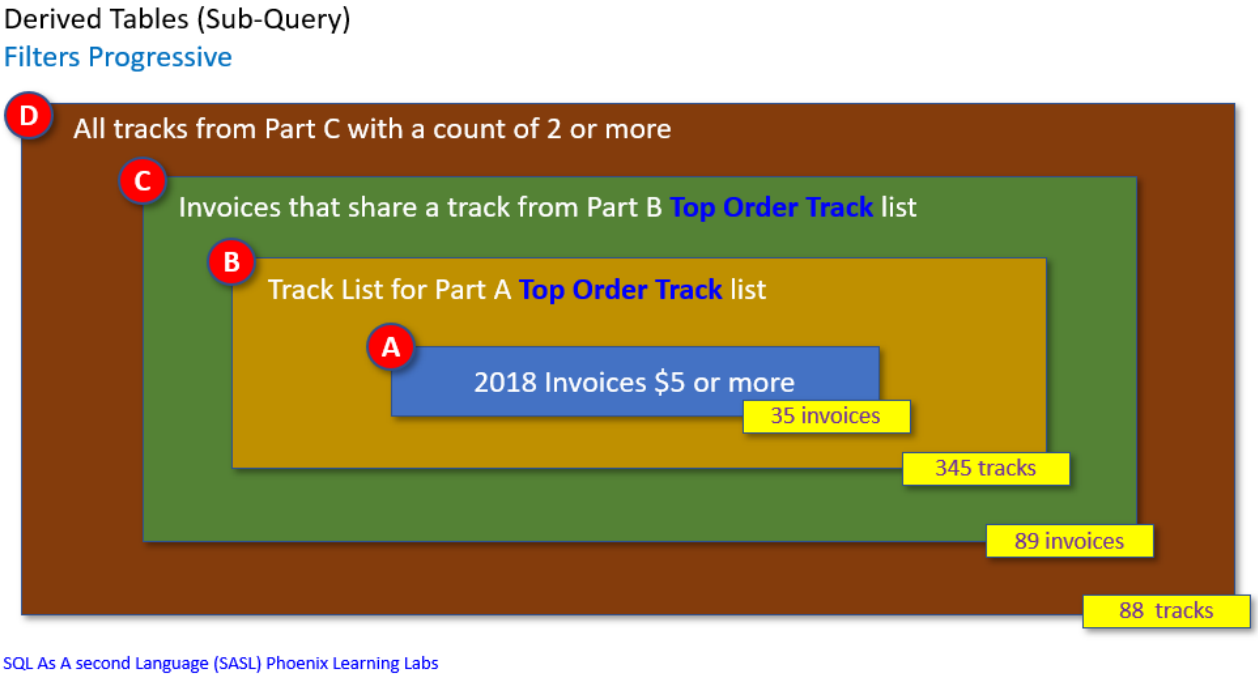


*Figure 12. derivedTable Filter*

This request could be done using a single query. The derived tables are being used here, to show how they can be used as a filter or reduce the number of rows in the result.

Derived Tables are also know as sub-queries.

*SQL Example*

```
/*

derivedTable  List of album/track info for other tracks that match 2018 order $5 or
more track list

Part D - List of tracks and album information

==== TOPICS

* Distinct to elminate duplicates
* Join multiple tables
* Derived table or sub-query
* Aggregation functions to be used for count
* Table Alias
* Between function
```

```
* Date representation
* Group by columns
* Order By (Sort) columns
* Having (Aggregate filter)


==== TIPS

* Table alias shorten the typing needed to complete the query
1-3 character alias recommended.
* Column alias help to give meaningful, non conflicting names.
Good practice is to have table alias used everywhere if used.
Use the same alias across all queries.
* Coulmns for join must match in data type and size
* Good practice is that System Generated Primary Key column name be named with `ID` at
the end
* Good practice, alias column names use lowerUpper naming to identify them as alias
* Primary Key and Foreign Key matching columns should usew the same name

*/

Select
    IL.TrackID
    ,T.Name as trackName   ①
    ,A.AlbumID
    ,A.AlbumTitle
    ,count(T.TrackID) as trackCount  ②
from InvoiceLine IL

inner Join Track T
on T.TrackID = IL.TrackID

inner join Album A
on A.AlbumID = T.AlbumID

inner join
    ( -- Any invoice with tracks from the 2018 invoice $5 or more track list  ③
    Select distinct   ④
        IL.InvoiceID
    from InvoiceLine IL

    inner join
        ( -- Track list for 2018 invoice $5 or more  ⑤
        Select
            IL.TrackID
        from invoiceline IL

        inner join
            ( -- 2018 Invoices $5 or more  ⑥
            select DISTINCT  ④
                I.InvoiceID
```

```
            from Invoice I
            where I.InvoiceDate between date '2018-01-01' and date '2018-12-31' ⑦ ⑧
              and I.total >= 5.00
            ) as RI  ⑨
        on RI.InvoiceID = IL.InvoiceID

        ) TL ⑩
    on TL.TrackID = IL.TrackID

    ) as SI ⑪
on SI.InvoiceID = IL.InvoiceID

-- Syntax below is related to the top most query

group by ⑫
    IL.TrackID
    ,T.Name
    ,A.AlbumID
    ,A.AlbumTitle

Order by
    trackCount Desc  ⑬
    ,trackname       ⑭

Having trackCount >= 2  ⑮
```

① column alias

② derived column with column alias

③ The beginning of a derived table (sub-query) starts with a `(`

④ `DISTINCT` used to eliminate duplicates, applied to the entire result set

⑤ The beginning of a derived table (sub-query) starts with a `(`

⑥ The beginning of a derived table (sub-query) starts with a `(`

⑦ `between` function requires starting value `and` ending value

⑧ `date` indicates a date field, Teradata standard format is `yyyy-mm-dd`

⑨ The end of the derived table (sub-query) is marked with a `)` and a name for the derived table `RI` Reduced Invoices. The derived table name is used to qualify the use of the columns in the main query.

⑩ The end of the derived table (sub-query) is marked with a `)` and a name for the derived table `TL` Track List. The derived table name is used to qualify the use of the columns in the main query.

⑪ The end of the derived table (sub-query) is marked with a `)` and a name for the derived table `SI` Special Invoices. The derived table name is used to qualify the use of the columns in the main query.

⑫ `Group by` required for any column that is not an aggregate column

⑬ `trackCount` is a column alias of a derived column (Count) used here for sorting

⑭ `trackName` is a column alias

⑮ `having` is a filter for a aggregate column

SQL Download link click here

*Table 15. Results*

| TrackId | trackName | AlbumId | AlbumTitle | trackCount |
|---|---|---|---|---|
| 2108 | Children Of The Grave | 174 | Tribute | 5 |
| 698 | Good Golly Miss Molly | 55 | Chronicle, Vol. 2 | 3 |
| 2250 | Nega Do Cabelo Duro | 184 | Os Cães Ladram Mas A Caravana Não Pára | 3 |
| 530 | Ando Meio Desligado | 42 | Minha História | 2 |
| 1853 | Battery | 152 | Master Of Puppets | 2 |
| 1865 | Better Than You | 153 | ReLoad | 2 |
| 2172 | Big Wave | 179 | Pearl Jam | 2 |
| 2305 | Binky The Doormat | 189 | New Adventures In Hi-Fi | 2 |
| 2793 | Cabeça Dinossauro | 224 | Acústico | 2 |
| 449 | Calling Dr. Love | 37 | Greatest Kiss | 2 |
| 1226 | Can I Play With Madness | 96 | A Real Live One | 2 |
| 2781 | Comida | 224 | Acústico | 2 |
| 681 | Commotion | 54 | Chronicle, Vol. 1 | 2 |
| 2763 | Compadre | 222 | Serie Sem Limite (Disc 1) | 2 |
| 512 | Comportamento Geral | 41 | Meus Momentos | 2 |
| 2317 | Country Feedback | 187 | Out Of Time | 2 |
| 1621 | Dazed and Confused | 132 | Led Zeppelin I | 2 |
| 2531 | End Of Romanticism | 204 | Morning Dance | 2 |
| 3064 | Eruption | 243 | The Best Of Van Halen, Vol. I | 2 |

Only 20 rows of the result being shown. The result set has 88 rows, one for each track of an order with a matching track in the **Top Order Track** list.

Back to Introduction.

# 4.16. Create View

A view is a database object is built from other database objects like tables and views.

Views can help give you a simpler viewable version of a complex query.

Views can help you reuse a standard copy of a database object across queries.

Unlike a table which takes up permanent space for its data, a view does not take up permanent space.

## 4.16.1. Example

In the Chinook database many of the queries require comprehensive Album information.

To get the comprehensive information we need data from five different tables.

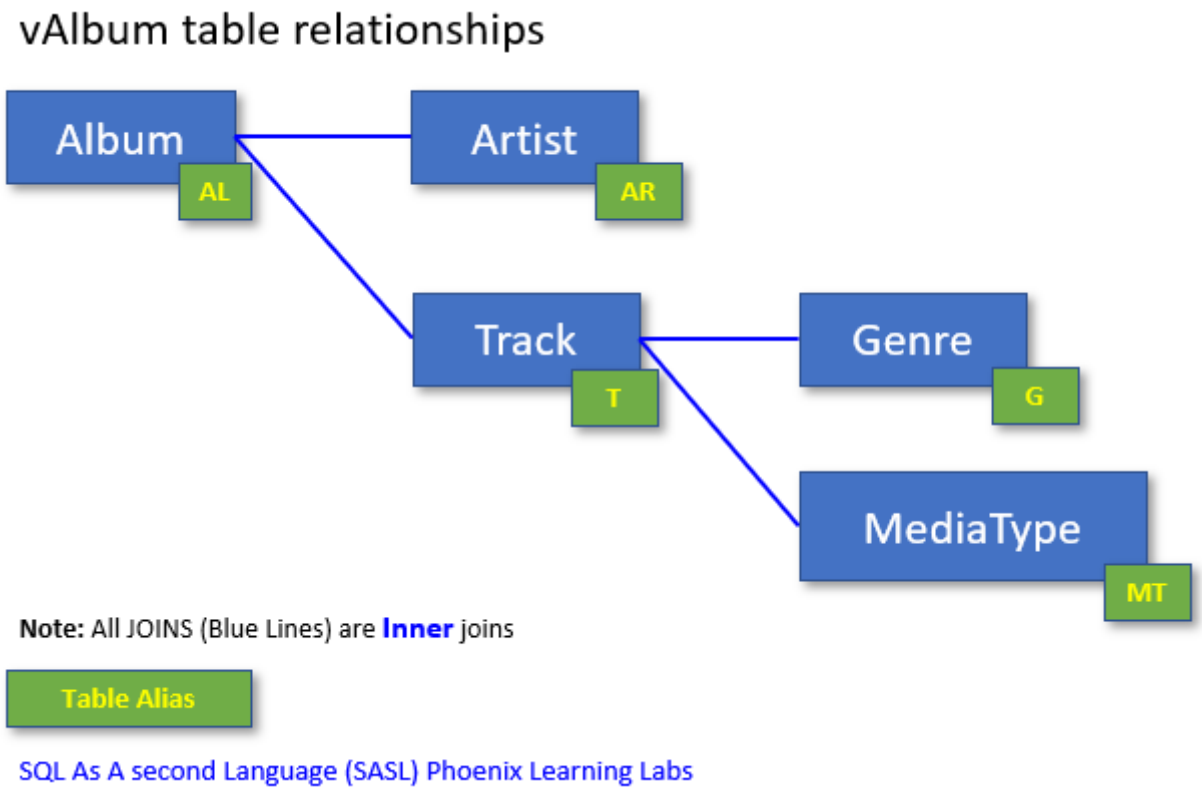


*Figure 13. vAlbum Relationships*

*SQL Example*

```
/*   Create vAlbums with all required parts

=== Create View - Create a view for comprehensive album information

NOTE: Your account must have `perm space` assigned to it to create a view

* Need to join the following tables
** Album - Artist
** Album - Track
```

```
** Track - Genre
** Track - Media Type

==== TOPICS

* Table alias
* Column Alias
* Join Table together


==== TIPS

.Table Alias
* Table alias shorten the typing needed to complete the query,
1-3 character alias recommended.
* Good practice is to have table alias used everywhere in the query.

.Column Alias
* Column alias help to give meaningful, non conflicting names.
* Good practice, alias column names use lowerUpper naming
to identify them as alias (albumCount)
* Good practic, Use the same alias across all queries.

.Naming practice
* Good practice, System Generated Primary Index column name be
include with `ID` suffix.
* Good practice, Primary Key and Foreign Key join columns
should use the same name.

.Join requirements
* Coulmns for join must match in data type and size.
* Good practice, bring keys from join tables into the view when possible
Helps to reduce debugging efforts later.

*/


Create view vAlbum  ①
-- Update View vAlbum   ②

as

Select
    AL.AlbumId
    ,AL.AlbumTitle
    ,AR.ArtistID
    ,AR.Name as artistName  ③
    ,T.TrackId
    ,T.Name as titleName  ③
    ,T.Composer
    ,T.Milliseconds
```

```
    ,T.Milliseconds / 60000.00 as minutes -- Divide milliseconds by 60,000 for minutes
④
    ,T.SizeBytes
    ,T.SizeBytes / 1048576.00 as mbSize  -- Megabyte 2^20 1024 x 1024 or 1,048,567 ⑤
    ,G.GenreId
    ,G.Name as genreName
    ,MT.MediaTypeID
    ,MT.Name as mediaTypeName
from Album AL    ⑥

INNER JOIN Artist AR  -- Artist name ⑥
on AR.ArtistID = AL.ArtistID

INNER JOIN Track T -- All tacks info  ⑥
on T.AlbumID = AL.AlbumId

INNER JOIN Genre G -- Genre for each track
on G.GenreId = T.GenreId  ⑥

INNER JOIN MediaType MT -- MediaType for each track
on MT.MediaTypeId = T.MediaTypeId ⑥
```

① The first time a view is built use the `Create` statement

② To change the view use the `update` statement

③ Column Alias in order avoid confusion with the field `name`

④ Derived Column (aka Calculated Column) notice the 60000.00 has two zeros of precision in order to return a decimal number

⑤ Derived Column (aka Calculated Column) notice the 1048576.00 has two zeros of precision in order to return a decimal number

⑥ Table alias, each joined table has it own unique table alias. NOTE: This example the `Album` table was chosen as the main table of the view.

SQL Download link click here

If you didn't create the view then you may not know the column names of the view.

To display a list of the `vAlbum` view columns use

```
help view vAlbum
```

The column list of the view will be returned.

| | Column Name Char(30) | Type Char(2) | Comment VarChar(255) | Nullable Char(1) | Format Char(30) | Title VarChar(60) | Max Length Integer | Decimal Total Digits SmallInt |
|---|---|---|---|---|---|---|---|---|
| 1 | AlbumId ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2 | AlbumTitle ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 3 | ArtistId | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 4 | artistName ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 5 | TrackId | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 6 | titleName ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 7 | Composer ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 8 | Milliseconds ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 9 | minutes ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 10 | SizeBytes ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 11 | mbSize ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 12 | GenreId ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 13 | genreName ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 14 | MediaTypeId ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 15 | mediaTypeName ... | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

*Figure 14. Display View Columns*

This column list may be sufficient enough for you to use the view in your work.

If you need more knowledge about how the vAlbum view was built use

```
show view vAlbum
```

The SQL statement used to create the vAlbum view will be returned.

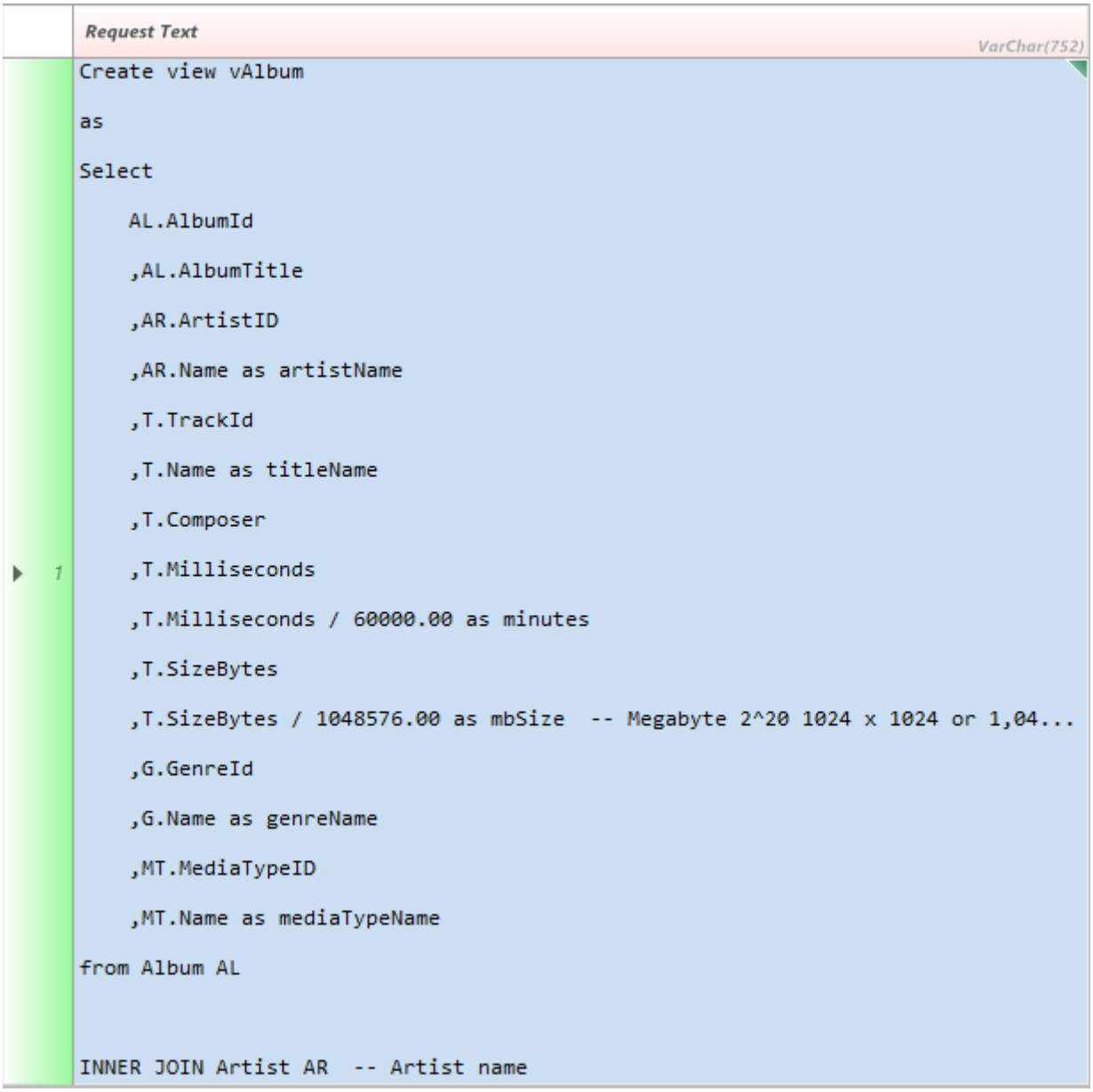


*Figure 15. Display View SQL*

Any bulk comments `/* comments \*/` will have been stripped out by Teradata when the view is created or updated.

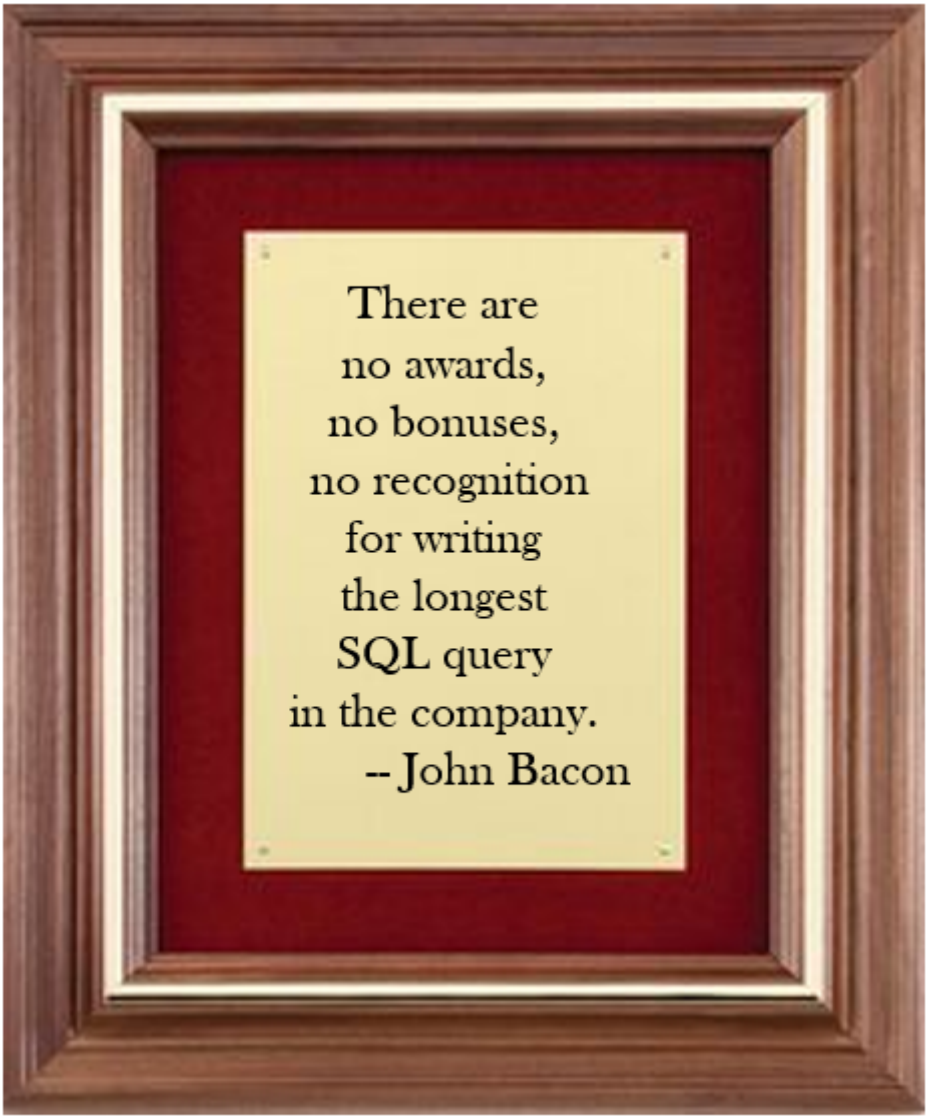


*Figure 16. Longest Query Rewards*

Back to Introduction.

# 5. Contact Information

Author contact Information

**Email:** schusterj@Michigan.gov

**Email:** john.schuster@PhoenixWorkgroup.com

**Phone:** 517-927-7400

Please provide any feedback, corrections or suggestions to the author. The folks providing the most anount of feedback will win prizes.

# 6. Reference

## 6.1. Teradata links

Documentation: https://docs.teradata.com/landing-page/

Forum: https://community.teradata.com/

SQL Assistant: https://bit.ly/2LYpXxB

SQL Introduction: https://bit.ly/2Fhxp5C

Stack Overflow (Teradata Forum): https://stackoverflow.com/questions/tagged/teradata

## 6.2. Column Type

ColumnType reference table.

*Table 16. ColumnType*

| ColumnType Abbrev | ColumnType Description |
|---|---|
| A1 | ARRAY |
| AN | MULTI-DIMENSIONAL ARRAY |
| AT | TIME |
| BF | BYTE |
| BO | BLOB |
| BV | VARBYTE |
| CF | CHARACTER |
| CO | CLOB |
| CV | VARCHAR |
| D | DECIMAL |
| DA | DATE |
| DH | INTERVAL DAY TO HOUR |
| DM | INTERVAL DAY TO MINUTE |
| DS | INTERVAL DAY TO SECOND |
| DY | INTERVAL DAY |
| F | FLOAT |
| HM | INTERVAL HOUR TO MINUTE |
| HS | INTERVAL HOUR TO SECOND |
| HR | INTERVAL HOUR |
| I | INTEGER |
| I1 | BYTEINT |

| ColumnType Abbrev | ColumnType Description |
|---|---|
| I2 | SMALLINT |
| I8 | BIGINT |
| JN | JSON |
| MI | INTERVAL MINUTE |
| MO | INTERVAL MONTH |
| MS | INTERVAL MINUTE TO SECOND |
| N | NUMBER |
| PD | PERIOD(DATE) |
| PM | PERIOD(TIMESTAMP WITH TIME ZONE) |
| PS | PERIOD(TIMESTAMP) |
| PT | PERIOD(TIME) |
| PZ | PERIOD(TIME WITH TIME ZONE) |
| SC | INTERVAL SECOND |
| SZ | TIMESTAMP WITH TIME ZONE |
| TS | TIMESTAMP |
| TZ | TIME WITH TIME ZONE |
| UT | UDT Type |
| XM | XML |
| YM | INTERVAL YEAR TO MONTH |
| YR | INTERVAL YEAR |
| ++ | TD_ANYTYP |

# 7. Document History

*Table 17. Document History*

| Date | Version | Author | Description |
|---|---|---|---|
| 01/09/2019 | v2.1k | JHRS | Added View and join types, plaques |
| 01/08/2018 | V2.1j | JHRS | Updated Chinook introduction |
| 01/03/2019 | V2.1h | JHRS | Added derived Table for Filters Added diagrams for derived tables |
| 01/02/2019 | V2.1g | JHRS | Updated derived query for collections |
| 12/28/2018 | V2.1f | JHRS | Added derivedQuery from archived |
| 12/27/2018 | V2.1e | JHRS | Added joinQuery set from archive |
| 12/21/2018 | V2.1d | JHRS | Added vsCode snippet for quick query insert added Reference section |
| 12/20/2018 | V2.1c | JHRS | Attempting standard document template |
| 12/17/2018 | V2.1b | JHRS | Initial version |