SQL Basics

Lecture 7b

Outline

- SQL
- Getting Data Out
  - **`SELECT`**, **`FROM`**, **`WHERE`**
  - **`ORDER BY`**, **`DISTINCT/ALL`**, **`UNION/INTERSECT`**
  - Joins (**`INNER`**, **`OUTER`**, **`LEFT`**, **`RIGHT`**, **`NATURAL`**)
  - Aggregation (**`GROUP BY`**, **`MIN/MAX/SUM/AVG/COUNT`**, **`HAVING`**)
  - Nesting (**`IN`**, **`ALL`**, **`EXISTS`**)
- Changing Data
  - **`INSERT`**
  - **`UPDATE`**
  - **`DELETE`**

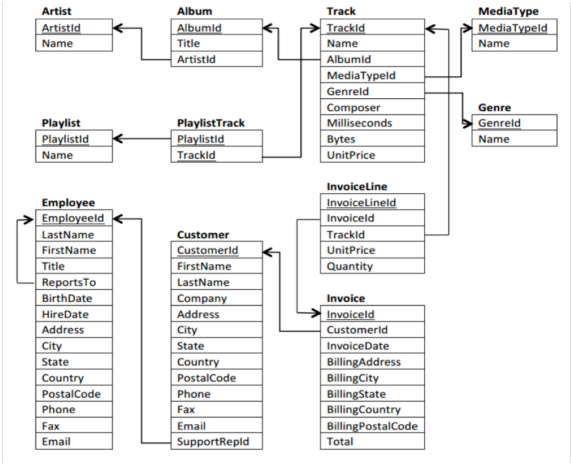
SQL: Structured Query Language

- Declarative: says what, not how
  - For the most part
- Originally based on relational model/calculus
  - Now industry standards: SQL-86, SQL-92, SQL:1999 (-2011)
  - Various degrees of adoption
- Capabilities
  - Data Definition (DDL): table/index structure
  - Data Manipulation (DML): add/update/delete
  - Transaction Management: begin/commit/rollback
  - Data Control: grant/revoke
  - Query
- Good reference: http://www.w3schools.com/sql

Simplest Query Form

```
SELECT <attribute list>
FROM <table list>
[WHERE <condition list>];
```

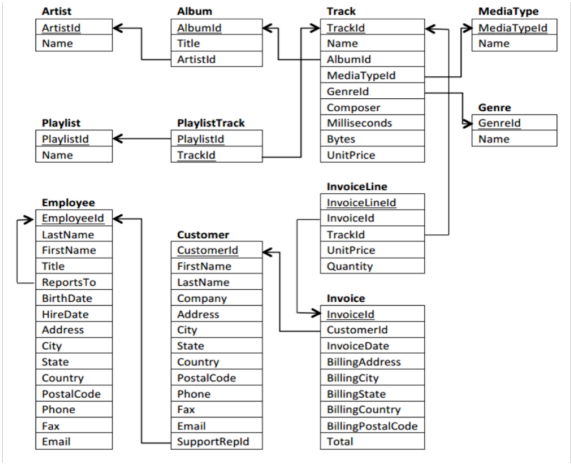
Basic Queries (1)



All artist names

```
SELECT Name
FROM artist;
```

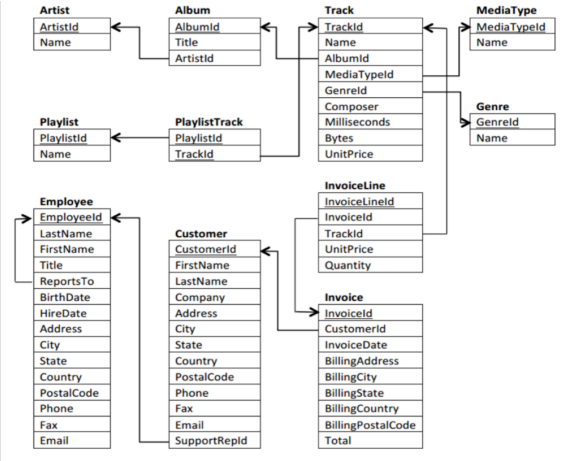
Basic Queries (2)



All employee names, full address info

```
SELECT FirstName, LastName, Address, City, State, PostalCode, Country
FROM employee;
```

Basic Queries (3)



All invoice line(s) with invoice, unit price, quantity

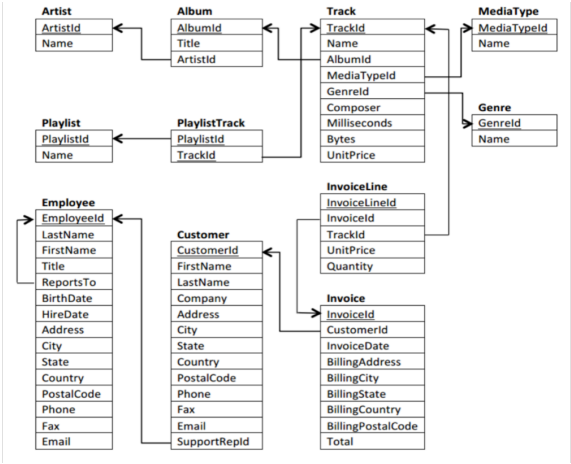
```
SELECT InvoiceId, UnitPrice, Quantity
FROM invoiceline;
```

Condition List ~ Boolean Expression

Clauses separated by **AND**/**OR** and **( )**

Operator	Meaning	Example
=	Equal to	`InvoiceId = 2`
<>	Not equal to	`Name <> 'U2'`
< or >	Less/Greater than	`UnitPrice < 5`
<= or >=	Less/Greater than or equal to	`UnitPrice >= 0.99`
LIKE	Matches pattern	`PostalCode LIKE 'T2%'`
IN	Within a set	`City IN ('Calgary', 'Edmonton')`
IS or IS NOT	Compare to NULL	`ReportsTo IS NULL`
BETWEEN	Inclusive range (esp. dates)	`UnitPrice BETWEEN 0.99 AND 1.99`

Conditional Query



All non-boss employee names in Calgary

```
SELECT FirstName, LastName
FROM employee
WHERE ( ReportsTo IS NOT NULL ) AND ( City = 'Calgary' );
```

Attribute List

- To get all fields, use *

```
SELECT * FROM employee;
```

- To rename a field in the result, use AS

```
SELECT FirstName AS fname, LastName AS lname
FROM employee;
```

- Field can be the result of an expression on one/more fields (available functions depend upon DBMS), usually rename

```
SELECT InvoiceId, UnitPrice, Quantity,
InvoiceLineId, (UnitPrice*Quantity) AS cost
FROM invoiceline WHERE UnitPrice >= 1;
```

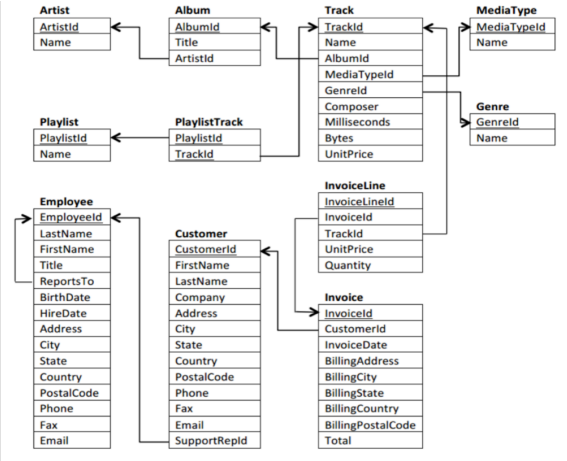
Non-Standard SQL

- SQLite
  - http://sqlite.org/lang.html
- MySQL
  - http://dev.mysql.com/doc/refman/5.0/en/func-op-summary-ref.html

Example: Concatenate fields

- SQLite
  - `SELECT (field1 || field2) AS field3`
- MySQL
  - `SELECT CONCAT(field1, field2) AS field3`

Complex Output Query (SQLite)

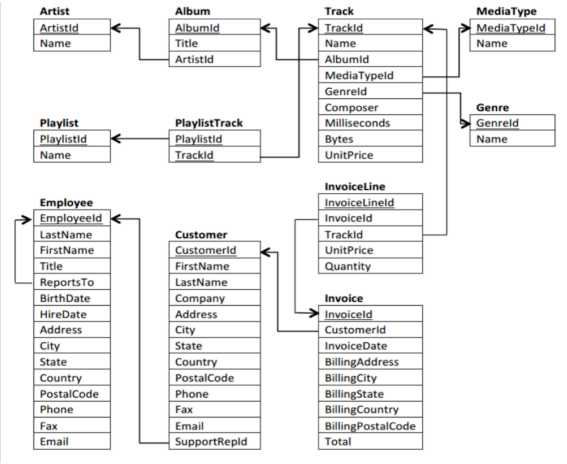


	german_city	total
1	Stuttgart	$1.98
2	Berlin	$1.98
3	Stuttgart	$13.86
4	Berlin	$1.98
5	Berlin	$3.96
6	Berlin	$13.86
7	Berlin	$5.94
8	Stuttgart	$8.91
9	Berlin	$8.91
10	Frankfurt	$1.98
11	Frankfurt	$13.86
12	Frankfurt	$14.91
13	Stuttgart	$1.98
14	Stuttgart	$3.96
15	Berlin	$1.98
16	Berlin	$1.98
17	Berlin	$13.86
18	Stuttgart	$5.94
19	Berlin	$3.96
20	Berlin	$5.94
21	Berlin	$8.91
22	Frankfurt	$1.98
23	Frankfurt	$3.96
24	Frankfurt	$5.94

Find all German invoices greater than $1, output city using the column header “german_city” and “total” prepending $ to the total

```
SELECT BillingCity AS german_city, ( '$' || Total ) AS total
FROM invoice
WHERE ( BillingCountry = 'Germany' ) AND ( Total > 1 );
```

Complex Output Query (MySQL)



german_city	total
Stuttgart	$1.98
Berlin	$1.98
Stuttgart	$13.86
Berlin	$1.98
Berlin	$3.96
Berlin	$13.86
Berlin	$5.94
Stuttgart	$8.91
Berlin	$8.91
Frankfurt	$1.98
Frankfurt	$13.86
Frankfurt	$14.91
Stuttgart	$1.98
Stuttgart	$3.96
Berlin	$1.98
Berlin	$1.98
Berlin	$13.86
Stuttgart	$5.94
Berlin	$3.96
Berlin	$5.94
Berlin	$8.91
Frankfurt	$1.98
Frankfurt	$3.96
Frankfurt	$5.94

Find all German invoices greater than $1, output city using the column header “german_city” and “total” prepending $ to the total

```
SELECT BillingCity AS german_city, CONCAT( '$', Total ) AS total
FROM invoice
WHERE ( BillingCountry = 'Germany' ) AND ( Total > 1 );
```

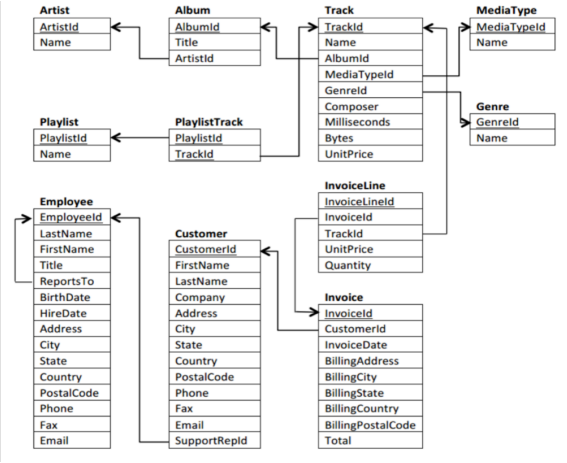
Ordering Output

SELECT <attribute list>
FROM <table list>
[**WHERE** <condition list>]
[**ORDER BY** <attribute-order list>];

Attribute-Order List (comma separated):

- <Attribute Name> [Order]
  - Order = ASC/DESC, ASC by default
  - Ties are processed in order of fields

Ordering Query



	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
1	299	26	2012-08-05 00:00:00	2211 W Berry Street	Fort Worth	TX	USA	76110	23.86
2	201	25	2011-05-29 00:00:00	319 N. Frances Street	Madison	WI	USA	53703	18.86
3	103	24	2010-03-21 00:00:00	162 E Superior Street	Chicago	IL	USA	60611	15.86
4	397	27	2013-10-13 00:00:00	1033 N Park Ave	Tucson	AZ	USA	85719	13.86
5	26	19	2009-04-14 00:00:00	1 Infinite Loop	Cupertino	CA	USA	95014	13.86
6	145	16	2010-09-23 00:00:00	1600 Amphitheatre Parkway	Mountain View	CA	USA	94043-1351	13.86
7	124	20	2010-06-22 00:00:00	541 Del Medio Avenue	Mountain View	CA	USA	94040-111	13.86
8	320	22	2012-11-06 00:00:00	120 S Orange Ave	Orlando	FL	USA	32801	13.86
9	5	23	2009-01-11 00:00:00	69 Salem Street	Boston	MA	USA	2113	13.86
10	222	21	2011-08-30 00:00:00	801 W 4th Street	Reno	NV	USA	89503	13.86
11	341	18	2013-02-07 00:00:00	627 Broadway	New York	NY	USA	10012-2612	13.86
12	82	28	2009-12-18 00:00:00	302 S 700 E	Salt Lake City	UT	USA	84102	13.86
13	243	17	2011-12-01 00:00:00	1 Microsoft Way	Redmond	WA	USA	98052-8300	13.86
14	311	28	2012-09-28 00:00:00	302 S 700 E	Salt Lake City	UT	USA	84102	11.94
15	298	17	2012-07-31 00:00:00	1 Microsoft Way	Redmond	WA	USA	98052-8300	10.91

All invoice info from the USA with greater than or equal to $10 total, ordered by the total (highest first), and then by state (alphabetical), then by city (alphabetical)

```
SELECT *
FROM invoice
WHERE ( BillingCountry = 'USA' ) AND ( Total >= 10 )
ORDER BY Total DESC, BillingState ASC, BillingCity;
```

Set vs. Bag/Multiset

- By default, Relational DBMSs treat results like bags/multisets (i.e. duplicates allowed)

- Use **DISTINCT** to remove duplicates

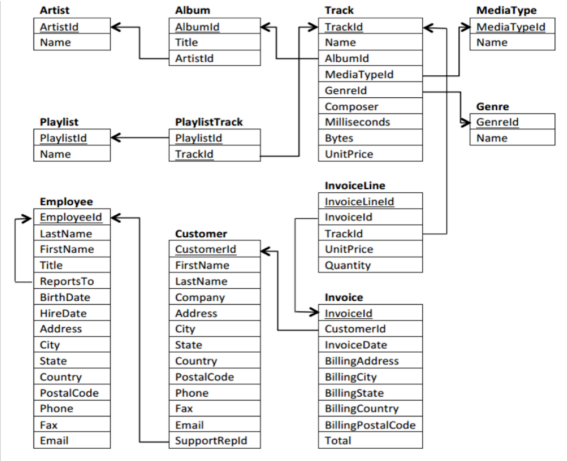
```
SELECT [DISTINCT] BillingState FROM
invoice WHERE BillingCountry='USA'
ORDER BY BillingState;
```

Set Operations

Use UNION, INTERSECT, EXCEPT/MINUS to combine results from queries

- Fields must match exactly in both results
- By default, set handling
  - Use ALL after to provide multiset
- Support is spotty here

Combining Queries (1)

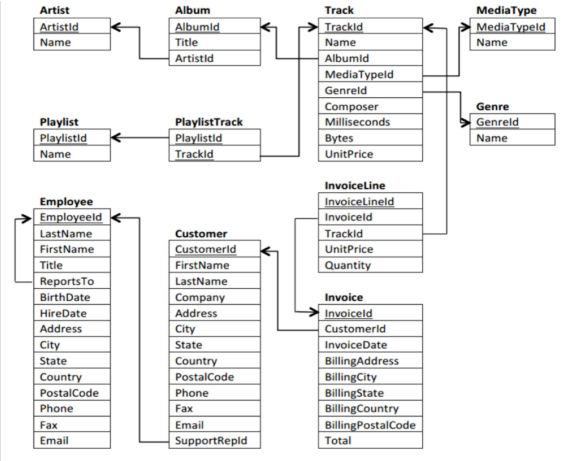


	city
1	Montréal
2	Edmonton
3	Vancouver
4	Toronto
5	Ottawa
6	Halifax
7	Winnipeg
8	Yellowknife

All Canadian cities in which customers live (call result “city”, i.e. lowercase)

```
SELECT City AS city
FROM customer
WHERE Country = 'Canada';
```

Combining Queries (2)

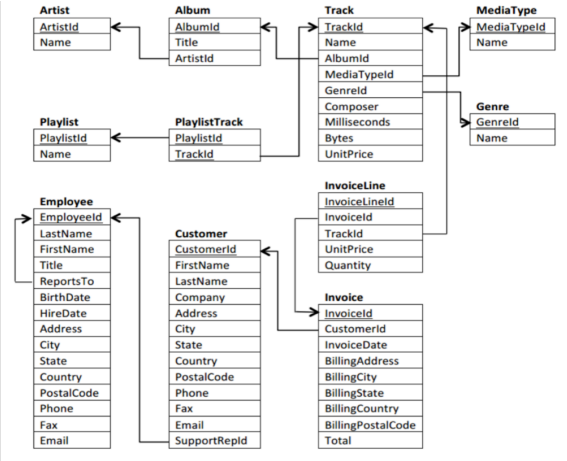


	city
1	Edmonton
2	Calgary
3	Calgary
4	Calgary
5	Calgary
6	Calgary
7	Lethbridge
8	Lethbridge

All Canadian cities in which employees live (call result “city”, i.e. lowercase)

```
SELECT City AS city
FROM employee
WHERE Country = 'Canada';
```

Combining Queries (3)

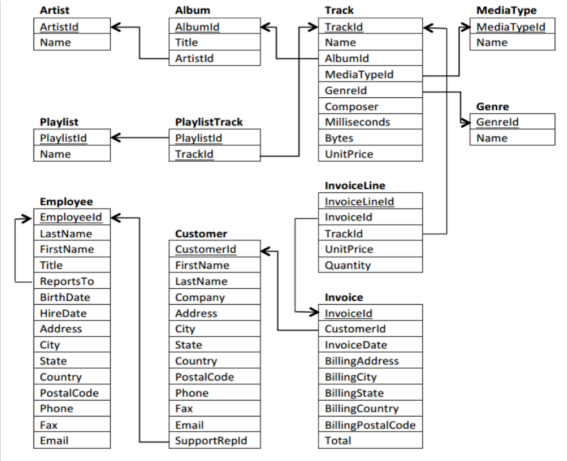


	city
1	Montréal
2	Edmonton
3	Vancouver
4	Toronto
5	Ottawa
6	Halifax
7	Winnipeg
8	Yellowknife
9	Edmonton
10	Calgary
11	Calgary
12	Calgary
13	Calgary
14	Calgary
15	Lethbridge
16	Lethbridge

All Canadian cities in which employees OR customers live (including duplicates)

```
SELECT City AS city FROM customer WHERE Country = 'Canada'
UNION ALL
SELECT City AS city FROM employee WHERE Country = 'Canada';
```

Combining Queries (4)

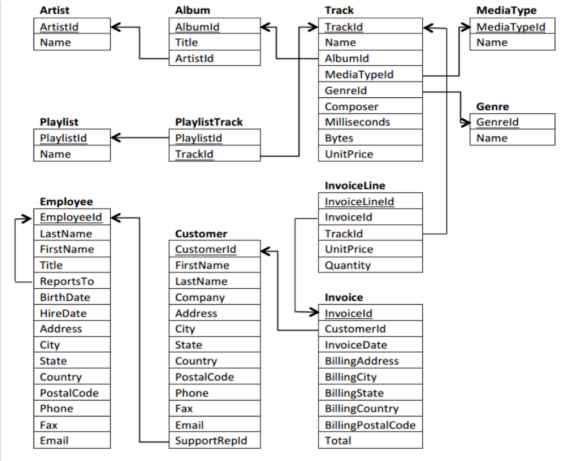


	city
1	Calgary
2	Edmonton
3	Halifax
4	Lethbridge
5	Montréal
6	Ottawa
7	Toronto
8	Vancouver
9	Winnipeg
10	Yellowknife

All Canadian cities in which employees OR customers live (excluding duplicates)

```
SELECT City AS city FROM customer WHERE Country = 'Canada'
UNION
SELECT City AS city FROM employee WHERE Country = 'Canada';
```

Combining Queries (5)

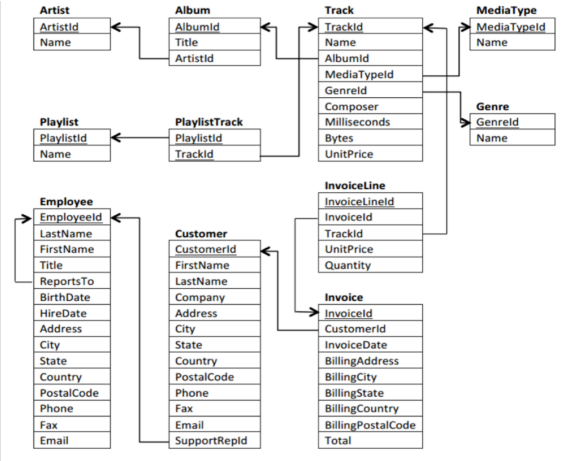


	city
1	Edmonton

All Canadian cities in which employees AND customers live (excluding duplicates)

```
SELECT City AS city FROM customer WHERE Country = 'Canada'
INTERSECT
SELECT City AS city FROM employee WHERE Country = 'Canada';
```

Combining Queries (6)



	city
1	Halifax
2	Montréal
3	Ottawa
4	Toronto
5	Vancouver
6	Winnipeg
7	Yellowknife

All Canadian cities in which customers live BUT employees do not (excluding duplicates)

```
SELECT City AS city FROM customer WHERE Country = 'Canada'
EXCEPT
SELECT City AS city FROM employee WHERE Country = 'Canada';
```

Joining Tables

To "join" a table means to combine row(s) from one or more tables

Basic syntax

```
SELECT <attribute list>
FROM
    (T1 <join type> T2 [ON <condition list>])
    <join type> T3 [ON <condition list>]…
[WHERE <condition list>];
```

"Left"

"Right"

"Right"

Common Join Types

`INNER JOIN`	Row must exist in both tables
`LEFT OUTER JOIN`	Row must exist in the table to the left of the type (padded with **`NULL`**)
`RIGHT OUTER JOIN`	Row must exist in the table to the right of the type (padded with **`NULL`**)
`FULL OUTER JOIN`	Row must exist in either table (padded with **`NULL`**)

Notes on Joins

- When dealing with multiple tables, advised to use full attribute addressing (table.attribute)
  - Tip: when listing the table name, give it a shortcut

```
SELECT * FROM table1 t1
```

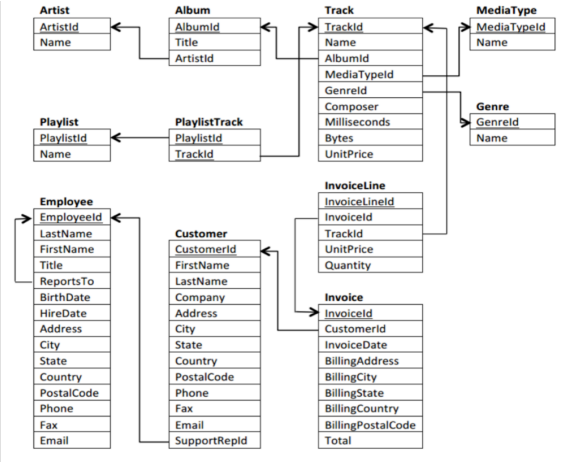
- NATURAL
  - Optional shortcut if joining attribute(s) have same name(s) in both tables

- Support/syntax can be spotty
  - Particularly full outer, natural

- Older style syntax (“soft join”)

```
SELECT * FROM t1, t2 WHERE t1.a1=t2.a1
```

Joins (1)

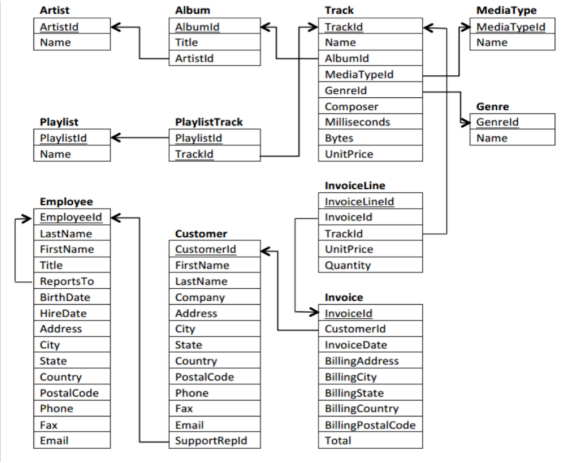


	ArtistId	Name
1	169	Black Eyed Peas
2	11	Black Label Society
3	12	Black Sabbath

Get all artist information for those whose name begins with ‘Black’, sort by name (alphabetically)

```
SELECT *
FROM artist
WHERE Name LIKE 'Black%'
ORDER BY Name ASC;
```

Joins (2)

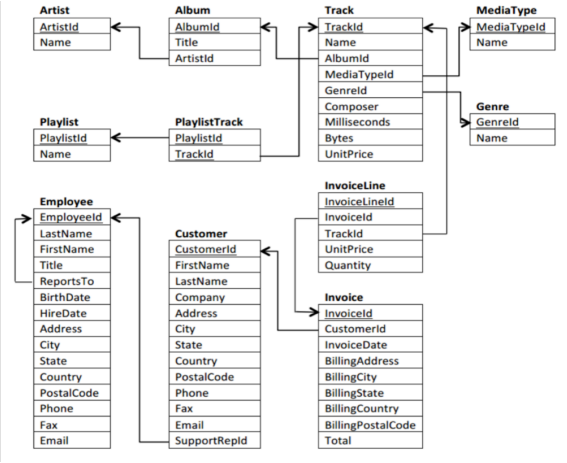


	ArtistId	Name	AlbumId	Title	ArtistId
1	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]	11
2	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]	11
3	12	Black Sabbath	16	Black Sabbath	12
4	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)	12

Get all artist AND album information for those artists whose name begins with ‘Black’ (don’t include those without albums), sort by artist name, then album name

```
SELECT *
FROM artist art INNER JOIN album alb ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name ASC, alb.Title ASC;
```

Joins (3)

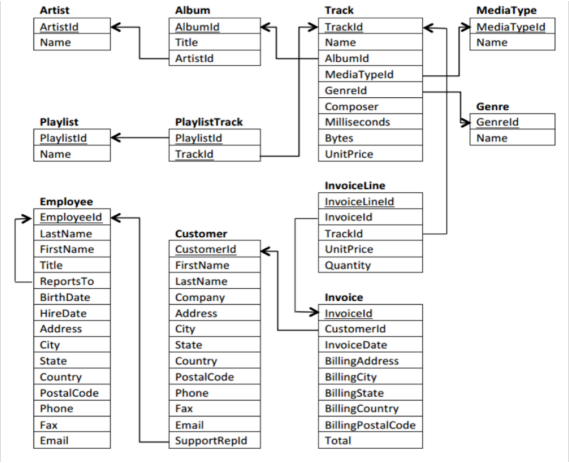


	ArtistId	Name	AlbumId	Title	ArtistId
1	169	Black Eyed Peas	{null}	{null}	{null}
2	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]	11
3	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]	11
4	12	Black Sabbath	16	Black Sabbath	12
5	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)	12

Get all artist AND album information for those artists whose name begins with 'Black' (do include those without albums!), sort by artist name, then album title

```
SELECT *
FROM artist art LEFT OUTER JOIN album alb ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name, alb.Title;
```

Joins (4)

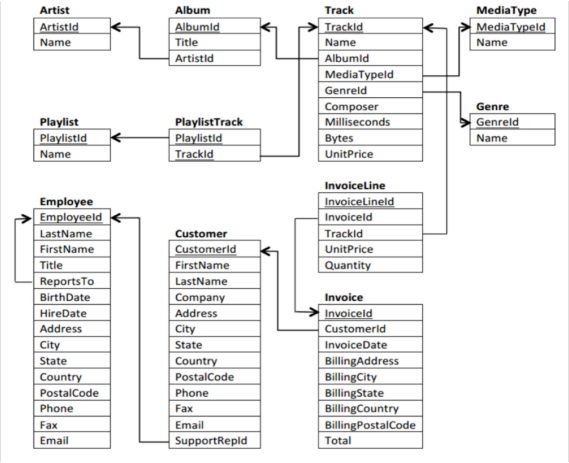


	ArtistId	Name	AlbumId	Title
1	169	Black Eyed Peas	{null}	{null}
2	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]
3	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]
4	12	Black Sabbath	16	Black Sabbath
5	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)

Get all artist AND album information for those artists whose name begins with 'Black' (do include those without albums!), provide only a single correct ArtistId, sort by artist name, then album title

```
SELECT art.ArtistId, art.Name, alb.AlbumId, alb.Title
FROM artist art LEFT OUTER JOIN album alb ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name, alb.Title;
```

Joins (5)

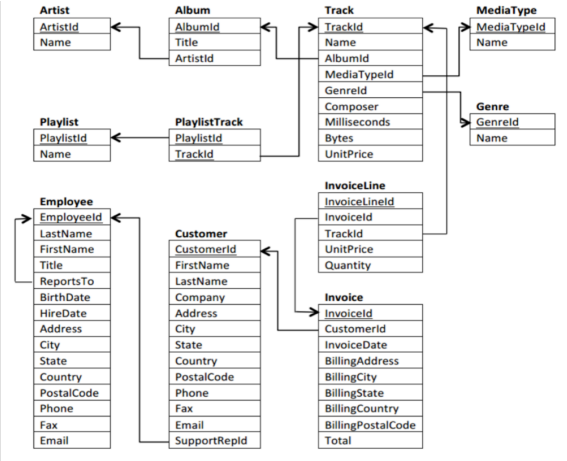


	ArtistId	Name	AlbumId	Title
1	169	Black Eyed Peas	{null}	{null}
2	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]
3	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]
4	12	Black Sabbath	16	Black Sabbath
5	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)

Get all artist AND album information for those artists whose name begins with 'Black' (do include those without albums!), provide only a single correct ArtistId – use NATURAL – sort by artist name, then album title

```
SELECT art.ArtistId, art.Name, alb.AlbumId, alb.Title
FROM artist art NATURAL LEFT OUTER JOIN album alb
WHERE Name LIKE 'Black%'
ORDER BY art.Name, alb.Title;
```

Joins (6)



	TrackId	tName	Composer	UnitPrice	Title	mName	gName
1	1139	Give Me Novacaine	Green Day	0.99	American Idiot	MPEG audio file	Alternative & Punk

Get track id, track name, composer, unit price, album title, media type name, and genre for the track titled “Give Me Novacaine”

```
SELECT t.TrackId, t.Name AS tName, t.Composer, t.UnitPrice,
       a.Title, m.Name AS mName, g.Name AS gName
FROM ((track t INNER JOIN album a ON t.AlbumId=a.AlbumId)
INNER JOIN mediatype m ON t.MediaTypeId=m.MediaTypeId)
INNER JOIN genre g ON t.GenreId=g.GenreId
WHERE t.Name='Give Me Novacaine';
```

Aggregation

- Aggregate functions take the value of a field (or an expression over multiple fields) for a set of rows and outputs a single value
  - Common: **MAX**, **MIN**, **SUM**, **AVG**, **COUNT**

- Example: number of tracks for an album

```
SELECT COUNT(*) FROM track WHERE AlbumId=1;
```

  - **COUNT(*)** = number of rows
  - **COUNT(field)** = number of non-NULL values
  - **COUNT(DISTINCT field)** = number of distinct values of a field

- Example: compute the total cost of an album

```
SELECT SUM(UnitPrice) FROM track WHERE AlbumId=1;
```

- Example: min/max/average track unit price overall

```
SELECT MIN(UnitPrice) AS min_price FROM track;
SELECT MAX(UnitPrice) AS max_price FROM track;
SELECT AVG(UnitPrice) AS avg_price FROM track;

SELECT MIN(UnitPrice) AS min_price, MAX(UnitPrice) AS max_price,
AVG(UnitPrice) AS avg_price FROM track;
```

Grouping

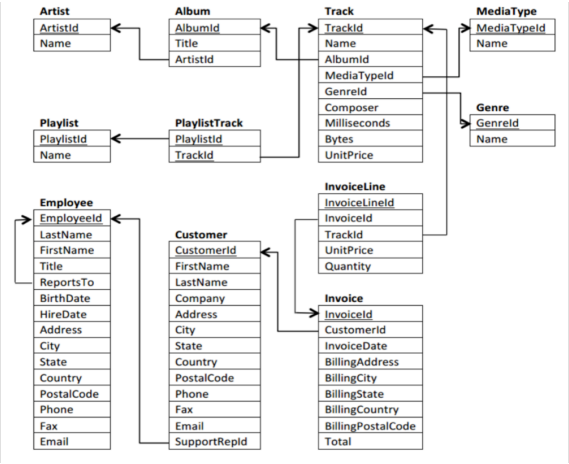
The **GROUP BY** statement allows you to define subgroups for aggregation functions. **GROUP BY** list should be a subset of **SELECT** list.

```
SELECT <attribute list>
FROM <table list>
[WHERE <condition list>]
[GROUP BY <attribute list>]
[ORDER BY <attribute-order list>];
```

Example: track price stats by genre

```
SELECT MediaTypeId, MIN(UnitPrice) AS min_price,
       MAX(UnitPrice) AS max_price, AVG(UnitPrice) AS avg_price
FROM track
GROUP BY MediaTypeId
ORDER BY avg_price DESC, MediaTypeId ASC;
```

Aggregation (1)

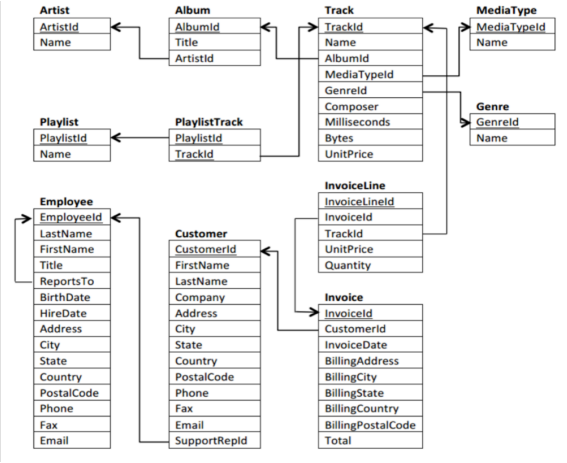


	BillingCity	BillingState	avg_total	sum_total	ct
1	Fort Worth	TX	6.80285714285714	47.62	7
2	Chicago	IL	6.23142857142857	43.62	7
3	Salt Lake City	UT	6.23142857142857	43.62	7
4	Madison	WI	6.08857142857143	42.62	7
5	Orlando	FL	5.66	39.62	7
6	Redmond	WA	5.66	39.62	7
7	Cupertino	CA	5.51714285714286	38.62	7
8	Mountain View	CA	5.51714285714286	77.24	14
9	Tucson	AZ	5.37428571428571	37.62	7
10	Boston	MA	5.37428571428571	37.62	7
11	Reno	NV	5.37428571428571	37.62	7
12	New York	NY	5.37428571428571	37.62	7

Get the average, sum, and number of all US invoices, grouped by city and state. Order by average cost (greatest first), then state, then city.

```
SELECT BillingCity, BillingState,
       AVG(Total) AS avg_total, SUM(Total) AS sum_total, COUNT(*) AS ct
FROM invoice
WHERE BillingCountry='USA'
GROUP BY BillingCity, BillingState
ORDER BY avg_total DESC, BillingState ASC, BillingCity ASC;
```

Aggregation (2)

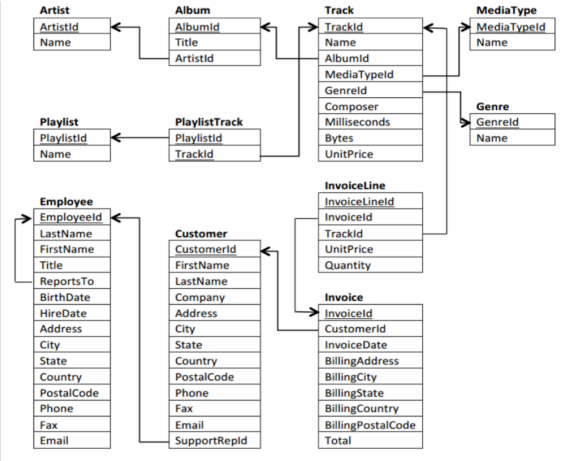


	InvoiceId	total
1	404	25.86
2	299	23.86
3	96	21.86
4	194	21.86
5	201	18.86
6	89	18.86
7	88	17.91
8	306	16.86
9	313	16.86
10	103	15.86
11	208	15.86
12	193	14.91
13	5	13.86
14	12	13.86
15	19	13.86

Using only the invoiceline table, compute the total cost of each order, sorted by total (greatest first), then invoice id (smallest first).

```
SELECT InvoiceId, SUM(UnitPrice*Quantity) AS total
FROM invoiceline
GROUP BY InvoiceId
ORDER BY total DESC, InvoiceId ASC;
```

Aggregation (3)



	TrackId	Name	Title	num_sold
1	430	I'm Going Slightly Mad	Greatest Hits II	2
2	2263	Somebody To Love	Greatest Hits I	2
3	2272	We Are The Champions	News Of The World	2
4	2259	You're My Best Friend	Greatest Hits I	2
5	419	A Kind Of Magic	Greatest Hits II	1
6	2274	All Dead, All Dead	News Of The World	1
7	2255	Another One Bites The Dust	Greatest Hits I	1
8	2258	Bicycle Race	Greatest Hits I	1
9	2254	Bohemian Rhapsody	Greatest Hits I	1
10	426	Breakthru	Greatest Hits II	1
11	2257	Fat Bottomed Girls	Greatest Hits I	1
12	2276	Fight From The Inside	News Of The World	1
13	2267	Flash	Greatest Hits I	1
14	2277	Get Down, Make Love	News Of The World	1
15	428	Headlong	Greatest Hits II	1

Generate a ranked list of Queen's best selling tracks. Display the track id, track name, and album name, along with number of tracks sold, sorted by tracks sold (greatest first), then by track name (alphabetical).

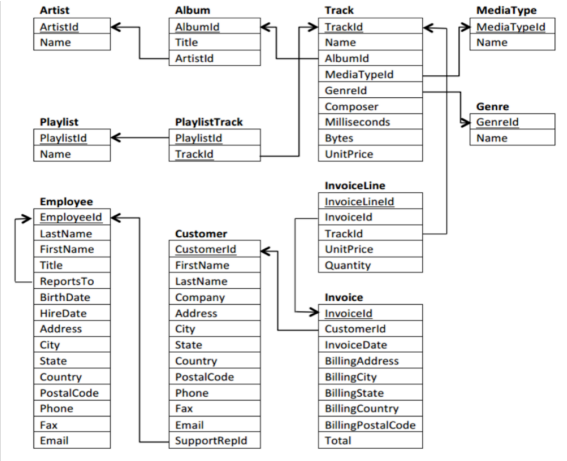
```
SELECT invoiceline.TrackId, track.Name, album.Title,
       SUM(invoiceline.Quantity) AS num_sold
FROM ((invoiceline INNER JOIN track ON invoiceline.TrackId=track.TrackId)
INNER JOIN album ON track.AlbumId=album.AlbumId)
INNER JOIN artist ON album.ArtistId=artist.ArtistId
WHERE artist.Name='Queen'
GROUP BY invoiceline.TrackId
ORDER BY num_sold DESC, track.Name ASC;
```

HAVING

The **HAVING** statement allows you to place constraint(s), similar to **GROUP BY**, that use aggregate functions (separate by **AND**/**OR**)

```
SELECT <attribute list>
FROM <table list>
[WHERE <condition list>]
[GROUP BY <attribute list>]
[HAVING <condition list>]
[ORDER BY <attribute-order list>];
```

Aggregation (4)



	TrackId	Name	Title	num_sold
1	430	I'm Going Slightly Mad	Greatest Hits II	2
2	2263	Somebody To Love	Greatest Hits I	2
3	2272	We Are The Champions	News Of The World	2
4	2259	You're My Best Friend	Greatest Hits I	2

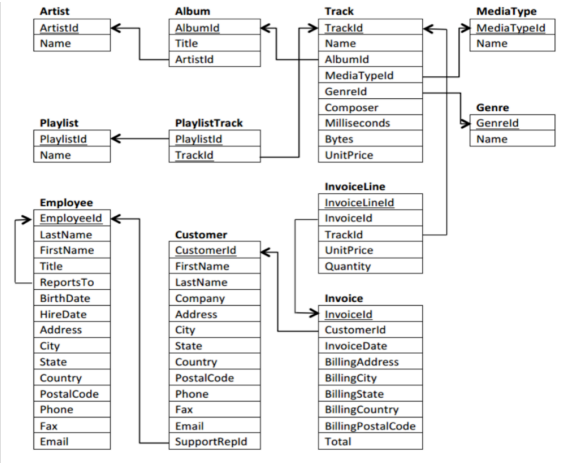
Generate a ranked list of Queen's best selling tracks. Display the track id, track name, and album name, along with number of tracks sold, sorted by tracks sold (greatest first), then by track name (alphabetical). Only show those tracks that have sold at least twice.

```
SELECT invoiceline.TrackId, track.Name, album.Title,
       SUM(invoiceline.Quantity) AS num_sold
FROM ((invoiceline INNER JOIN track ON invoiceline.TrackId=track.TrackId)
INNER JOIN album ON track.AlbumId=album.AlbumId)
INNER JOIN artist ON album.ArtistId=artist.ArtistId
WHERE artist.Name='Queen'
GROUP BY invoiceline.TrackId
HAVING SUM(invoiceline.Quantity)>=2
ORDER BY num_sold DESC, track.Name ASC;
```

Query in a Query

- Termed inner query, nested query, or subquery
- Most common locations
  - **`SELECT`** (fills in the value for a field)
  - **`FROM`** (becomes a view of the data)
  - **`WHERE`** (serves as part of a constraint)
    - [**`NOT`**] **`IN`** = query returns a single column of options
    - [**`NOT`**] **`EXISTS`** = checks if query returns at least a single field
    - <op> **`ALL`** = true if <op> returns true for all results (single field)
    - <op> **`ANY`**/**`SOME`** = true if <op> returns true for any result (single field)
- Correlated
  - Inner query references a value from an outer query
  - Inner query must be run *once for every tuple* of the outer query (i.e. performance issue)

Subquery: `WHERE` (1)

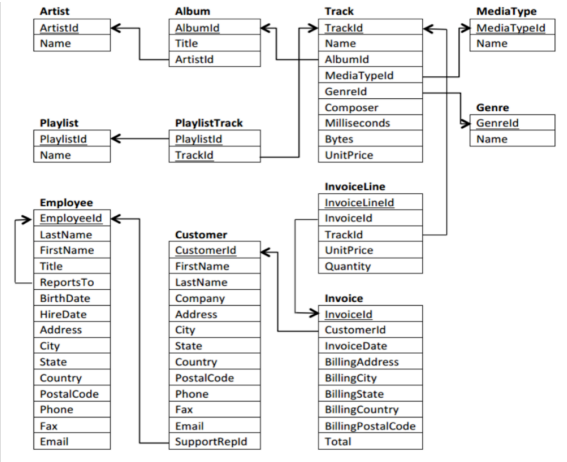


	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	38	All I Really Want	6	1	1	Alanis Morissette & Glenn Ballard	284891	9375567	0.99
2	39	You Oughta Know	6	1	1	Alanis Morissette & Glenn Ballard	249234	8196916	0.99
3	40	Perfect	6	1	1	Alanis Morissette & Glenn Ballard	188133	6145404	0.99
4	41	Hand In My Pocket	6	1	1	Alanis Morissette & Glenn Ballard	221570	7224246	0.99
5	42	Right Through You	6	1	1	Alanis Morissette & Glenn Ballard	176117	5793082	0.99
6	43	Forgiven	6	1	1	Alanis Morissette & Glenn Ballard	300355	9753256	0.99
7	44	You Learn	6	1	1	Alanis Morissette & Glenn Ballard	239699	7824837	0.99
8	45	Head Over Feet	6	1	1	Alanis Morissette & Glenn Ballard	267493	8758008	0.99
9	46	Mary Jane	6	1	1	Alanis Morissette & Glenn Ballard	280607	9163588	0.99
10	47	Ironic	6	1	1	Alanis Morissette & Glenn Ballard	229825	7598866	0.99
11	48	Not The Doctor	6	1	1	Alanis Morissette & Glenn Ballard	227631	7604601	0.99
12	49	Wake Up	6	1	1	Alanis Morissette & Glenn Ballard	293485	9703359	0.99
13	50	You Oughta Know (Alternate)	6	1	1	Alanis Morissette & Glenn Ballard	491885	16008629	0.99

Get all track information for the album Jagged Little Pill (do not use a join)

```
SELECT t.*
FROM track t
WHERE t.AlbumId = (
   SELECT a.AlbumId
   FROM album a
   WHERE a.Title='Jagged Little Pill'
);
```

Subquery: `WHERE` (2)

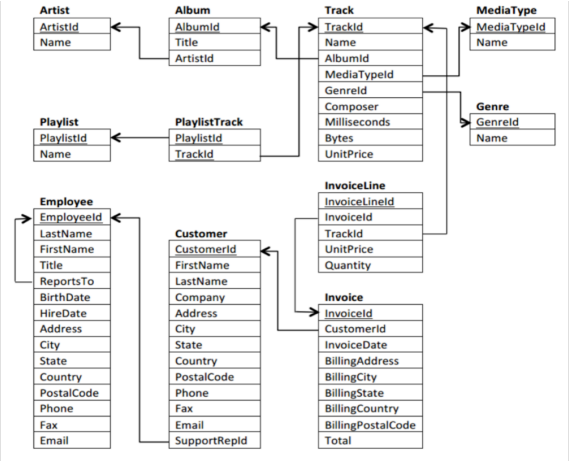


	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	419	A Kind Of Magic	36	1	1	Roger Taylor	262608	8689618	0.99
2	420	Under Pressure	36	1	1	Queen & David Bowie	236617	7739042	0.99
3	421	Radio GA GA	36	1	1	Roger Taylor	343745	11358573	0.99
4	422	I Want It All	36	1	1	Queen	241684	7876564	0.99
5	423	I Want To Break Free	36	1	1	John Deacon	259108	8552861	0.99
6	424	Innuendo	36	1	1	Queen	387761	12664591	0.99
7	425	It's A Hard Life	36	1	1	Freddie Mercury	249417	8112242	0.99
8	426	Breakthru	36	1	1	Queen	249234	8150479	0.99
9	427	Who Wants To Live Forever	36	1	1	Brian May	297691	9577577	0.99
10	428	Headlong	36	1	1	Queen	273057	8921404	0.99
11	429	The Miracle	36	1	1	Queen	294974	9671923	0.99
12	430	I'm Going Slightly Mad	36	1	1	Queen	248032	8192339	0.99
13	431	The Invisible Man	36	1	1	Queen	238994	7920353	0.99
14	432	Hammer To Fall	36	1	1	Brian May	220316	7255404	0.99
15	433	Friends Will Be Friends	36	1	1	Freddie Mercury & John Deacon	248920	8114582	0.99
16	434	The Show Must Go On	36	1	1	Queen	263784	8526760	0.99
17	435	One Vision	36	1	1	Queen	242599	7936928	0.99
18	2254	Bohemian Rhapsody	185	1	1	Mercury, Freddie	358948	11619868	0.99
19	2255	Another One Bites The Dust	185	1	1	Deacon, John	216946	7172355	0.99

Get all track information for the artist Queen (do not use a join)

```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (
   SELECT alb.AlbumId
   FROM album alb
   WHERE alb.ArtistId = (
        SELECT art.ArtistId
        FROM artist art
        WHERE art.Name='Queen'
   )
);
```

Subquery: FROM (1)

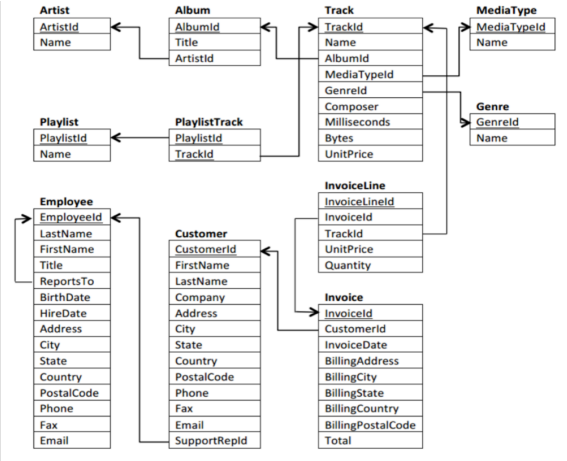


	min_q	max_q	avg_q	num_customers
1	36	38	37.966101694915	59

Find the minimum, maximum, and average number of tracks ordered per customer (across all invoices). Also include the number of customers.

```
SELECT MIN(q2.sum_q) AS min_q, MAX(q2.sum_q) AS max_q, AVG(q2.sum_q) AS avg_q,
       COUNT(*) AS num_customers
FROM
   (SELECT q1.CustomerId, SUM(Quantity) AS sum_q
    FROM
        (SELECT i.CustomerId, il.Quantity
         FROM invoice i NATURAL JOIN invoiceline il
        ) q1
    GROUP BY q1.CustomerId
   ) q2;
```

Subquery: FROM (2)

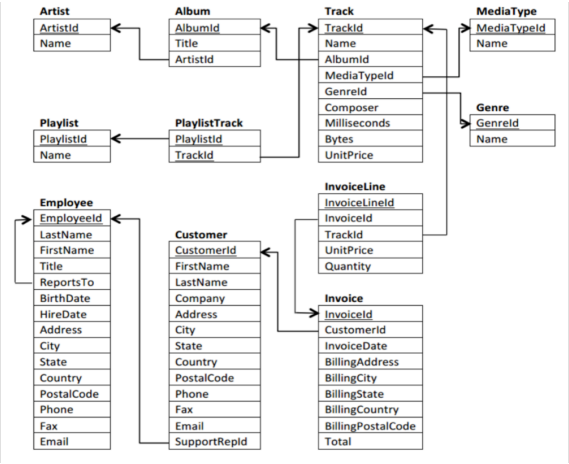


	CustomerId	FirstName	LastName	distinct_artists
1	41	Marc	Dubois	22
2	33	Ellie	Sullivan	21
3	3	François	Tremblay	21
4	34	João	Fernandes	20
5	13	Fernanda	Ramos	20
6	39	Camille	Bernard	19
7	23	John	Gordon	19
8	54	Steve	Murray	19
9	14	Mark	Philips	19
10	24	Frank	Ralston	19
11	59	Puja	Srivastava	19
12	4	Bjørn	Hansen	18
13	45	Ladislav	Kovács	18
14	43	Isabelle	Mercier	18
15	31	Martha	Silk	18

Produce a ranked list of customers: for each customer, the number of distinct artists in all the purchased tracks. Sort by number of distinct artists (greatest first), last name then first name (alphabetical).

```
SELECT CustomerId, FirstName, LastName,
       COUNT(DISTINCT ArtistId) AS distinct_artists
FROM
(
   SELECT i.CustomerId, c.FirstName, c.LastName, a.ArtistId
   FROM (((invoice i NATURAL JOIN invoiceline il)
           NATURAL JOIN customer c)
           NATURAL JOIN track t)
           NATURAL JOIN album a
) q1
GROUP BY CustomerId
ORDER BY distinct_artists DESC, LastName, FirstName;
```

Subquery: SELECT (1)

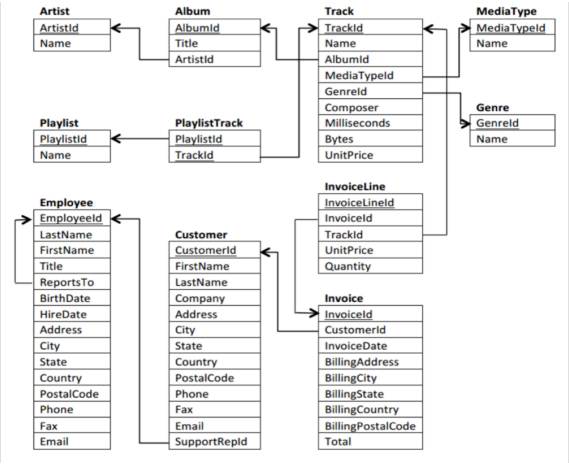


	Name	album_ct
1	Santana	3
2	Santana Feat. Dave Matthews	0
3	Santana Feat. Everlast	0
4	Santana Feat. Rob Thomas	0
5	Santana Feat. Lauryn Hill & Cee-Lo	0
6	Santana Feat. The Project G&B	0
7	Santana Feat. Maná	0
8	Santana Feat. Eagle-Eye Cherry	0
9	Santana Feat. Eric Clapton	0

For each artist starting with Santana, get the number of albums, sorted by count (greatest first)

```
SELECT art.Name,
       (
            SELECT COUNT(*)
            FROM album alb
            WHERE alb.ArtistId=art.ArtistId
        ) AS album_ct
FROM artist art
WHERE art.Name LIKE 'Santana%'
ORDER BY album_ct DESC;
```

Subquery: SELECT (2)



	FirstName	LastName	total_spent
1	Helena	Holý	49.62
2	Richard	Cunningham	47.62
3	Luis	Rojas	46.62
4	Ladislav	Kovács	45.62
5	Hugh	O'Reilly	45.62
6	Julia	Barnett	43.62
7	Frank	Ralston	43.62
8	Fynn	Zimmermann	43.62
9	Astrid	Gruber	42.62
10	Victor	Stevens	42.62
11	Terhi	Hämäläinen	41.62
12	Isabelle	Mercier	40.62
13	František	Wichterlová	40.62
14	Johannes	Van der Berg	40.62

Find the highest spending customers: get a ranked list of customers (first name, last name) who have spent at least $40, sorted by amount spent (greatest first), then last name, then first name

```
SELECT * FROM (
   SELECT c.FirstName, c.LastName, (
        SELECT SUM(i.Total)
        FROM invoice i
        WHERE c.CustomerId=i.CustomerId
   ) AS total_spent
   FROM customer c) q1
WHERE q1.total_spent>=40
ORDER BY q1.total_spent DESC, q1.LastName ASC, q1.FirstName ASC;
```

Inserting Rows

- Insert all attributes, in same order as table

```
INSERT INTO table
VALUES (a, b, … n);
```

- Insert a subset of attributes (not assigned = **NULL**)

```
INSERT INTO table (a1, a2, … an)
VALUES (a,b, … n)
```

- Insert via query

```
INSERT INTO table (a1, a2, … an)
SELECT a1, a2, … an FROM …
```

Updating Rows

General syntax

```
UPDATE table
SET <attribute=value list>
[WHERE <condition list>];
```

- Attribute=value is comma-separated
- Condition list may result in more than one rows being updated via a single statement

Deleting Rows

General syntax

```
DELETE FROM table
[WHERE <condition list>];
```

- Condition list may result in more than one rows being deleted via a single statement
- No condition = clear table