SQL: Part 1

Lecture 3

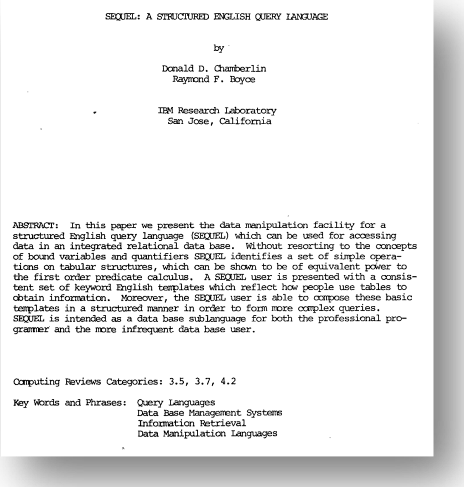
Outline

1. Context
2. Getting Data Out: **SELECT**
3. Changing Data: **INSERT**, **UPDATE**, **DELETE**

In the Beginning…

Chamberlin, Donald D., and Raymond F. Boyce. "SEQUEL: A structured English query language." *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*. ACM, **1974**.

"In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. ***A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user****."*



SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

Computing Reviews Categories: 3.5, 3.7, 4.2

Key Words and Phrases: Query Languages
Data Base Management Systems
Information Retrieval
Data Manipulation Languages

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): schema structure
  - Data Manipulation (DML): add/update/delete
  - Transaction Management: begin/commit/rollback
  - Data Control: grant/revoke
  - Query
  - Configuration

  …

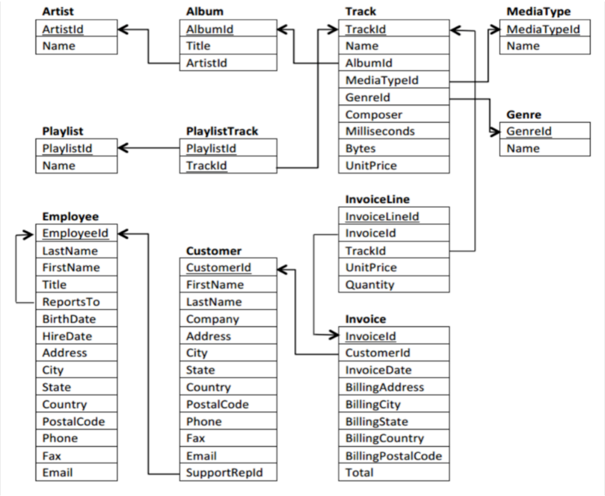
Good reference: http://www.w3schools.com/sql

Simplest Query Form

```
SELECT *
FROM <table name>;
```

Gets all the attributes for all the rows in the specified table. Result set order is arbitrary.

Your First Query!



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Get all information about all artists

```
SELECT *
FROM artist;
```

Attribute Control

```
SELECT <attribute list>
FROM <table name>;
```

Defines the columns of the result set. All rows are returned. Result set order is arbitrary.

Attribute List

- Comma separated

- As we saw, to get all fields in the table, use *

```
SELECT * FROM employee;
```

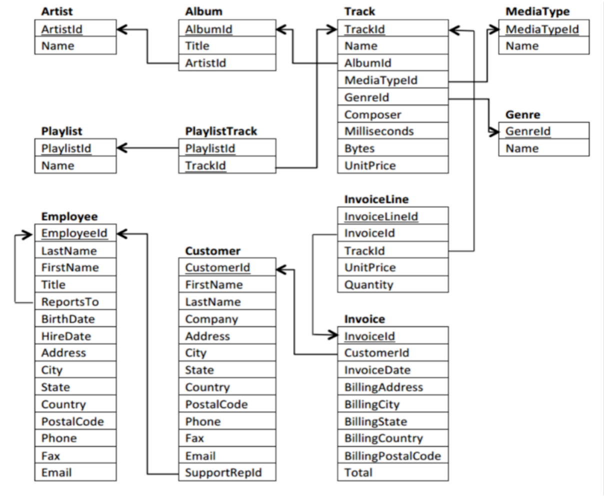
- To rename a field in the result, use AS

```
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 *, (UnitPrice*Quantity) AS cost
FROM invoiceline;
```

Basic Queries (1)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Playlist
PlaylistId
Name

PlaylistTrack
PlaylistId
TrackId

Genre
GenreId
Name

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

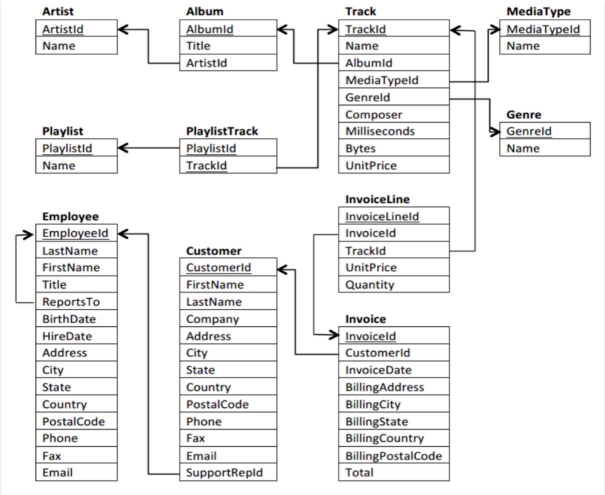
Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Get all artist names

```
SELECT Name
FROM artist;
```

Basic Queries (2)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

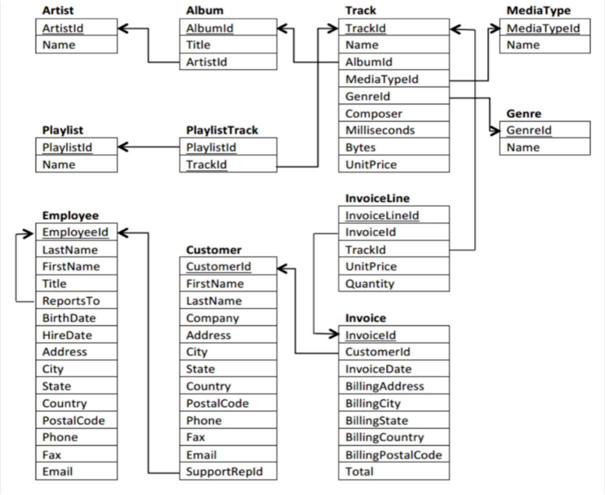
Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Get all employee names (first & last), with their full address info (address, city, state, zip, country)

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

Basic Queries (3)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Get all invoice line(s) with invoice, unit price, quantity

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

Choosing Rows to Include

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

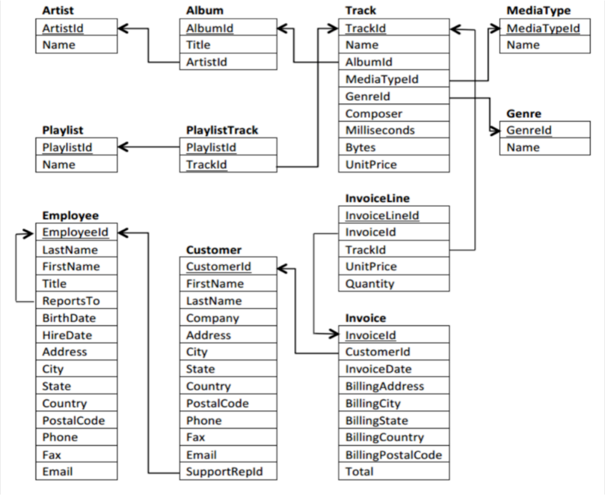
Defines the columns of the result set. Only those rows that satisfy the conditions are returned. Result set order is arbitrary.

Condition List ~ Boolean Expression

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

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`

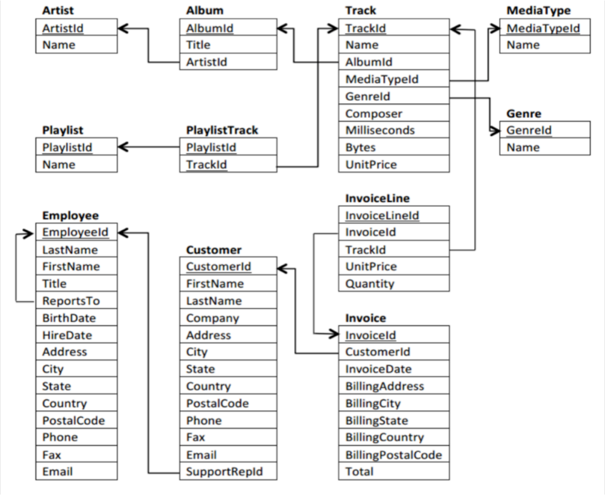
Conditional Query (1)



Get the billing country of all invoices totaling more than $10

```
SELECT BillingCountry
FROM invoice
WHERE Total>10;
```

Conditional Query (2)



Artist
- ArtistId
- Name

Album
- AlbumId
- Title
- ArtistId

Track
- TrackId
- Name
- AlbumId
- MediaTypeId
- GenreId
- Composer
- Milliseconds
- Bytes
- UnitPrice

MediaType
- MediaTypeId
- Name

Genre
- GenreId
- Name

Playlist
- PlaylistId
- Name

PlaylistTrack
- PlaylistId
- TrackId

InvoiceLine
- InvoiceLineId
- InvoiceId
- TrackId
- UnitPrice
- Quantity

Employee
- EmployeeId
- LastName
- FirstName
- Title
- ReportsTo
- BirthDate
- HireDate
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email

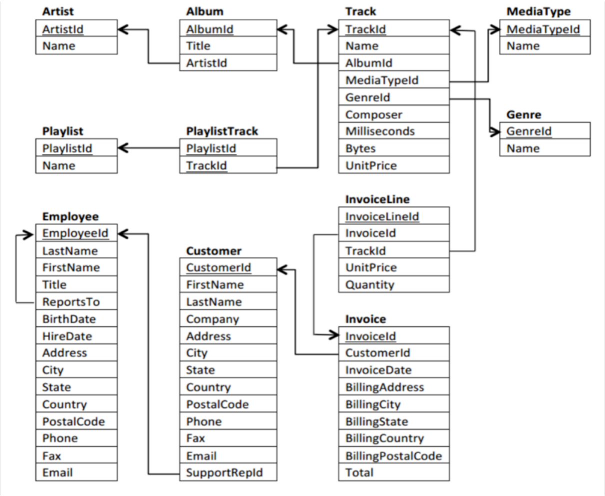
Customer
- CustomerId
- FirstName
- LastName
- Company
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email
- SupportRepId

Invoice
- InvoiceId
- CustomerId
- InvoiceDate
- BillingAddress
- BillingCity
- BillingState
- BillingCountry
- BillingPostalCode
- Total

Get all information about tracks whose name contains the word “Rock”

```
SELECT *
FROM track
WHERE Name LIKE '%Rock%';
```

Conditional Query (3)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Get the name (first, last) of all non-boss employees in Calgary (ReportsTo is NULL for the boss).

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

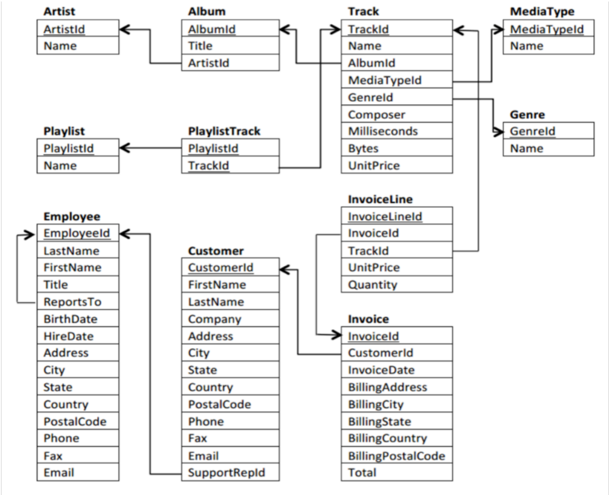
Non-Standard Functions

- 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)



Artist: ArtistId, Name

Album: AlbumId, Title, ArtistId

Track: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice

MediaType: MediaTypeId, Name

Genre: GenreId, Name

Playlist: PlaylistId, Name

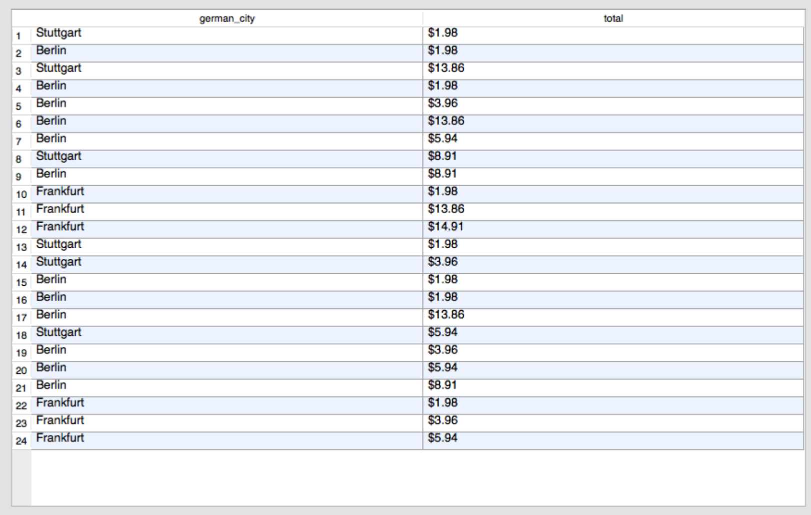
PlaylistTrack: PlaylistId, TrackId

InvoiceLine: InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity

Employee: EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, Country, PostalCode, Phone, Fax, Email

Customer: CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId

Invoice: InvoiceId, CustomerId, InvoiceDate, BillingAddress, BillingCity, BillingState, BillingCountry, BillingPostalCode, Total

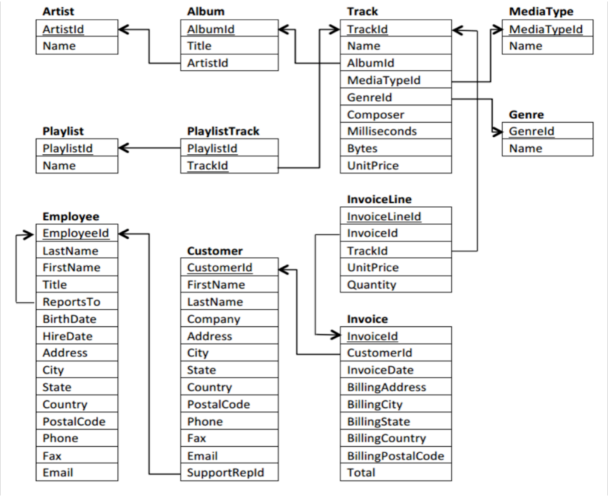


	german_city	total
1	Stuttgart	$1.98
2	Berlin	$1.98
3	Stuttgart	$13.86
4	Berlin	$1.98
5	Berlin	$3.96
6	Berlin	$13.86
7	Berlin	$5.94
8	Stuttgart	$8.91
9	Berlin	$8.91
10	Frankfurt	$1.98
11	Frankfurt	$13.86
12	Frankfurt	$14.91
13	Stuttgart	$1.98
14	Stuttgart	$3.96
15	Berlin	$1.98
16	Berlin	$1.98
17	Berlin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

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

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

Complex Output Query (MySQL)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

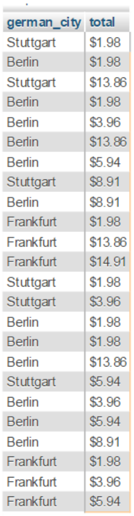
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



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

Get all German invoices greater than $1, output the 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 name>
[WHERE <condition list>]
[ORDER BY <attribute-order list>];
```

Defines the columns of the result set. Only those rows that satisfy the conditions are returned. Result set order is optionally defined.

Attribute Order List

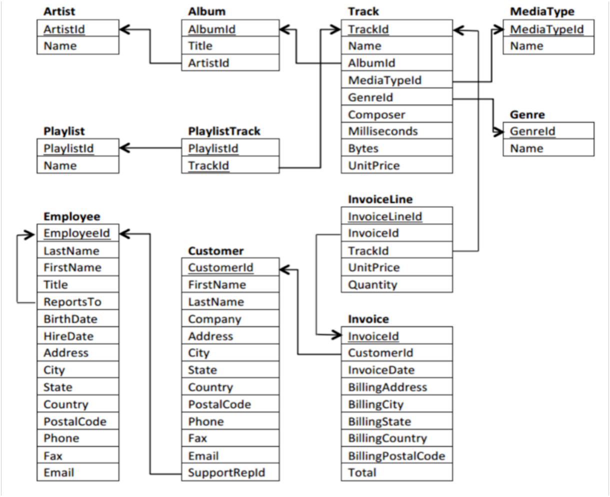
- Comma separated list

- Format: <attribute name> [Order]
  - Order can be **ASC** or **DESC**
  - Default is **ASC**

Example: order all employee information by last name (alphabetical), then first name (alphabetical), then birthdate (youngest first)

```
SELECT *
FROM employee
ORDER BY LastName, FirstName ASC, BirthDate DESC;
```

Ordering Query



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

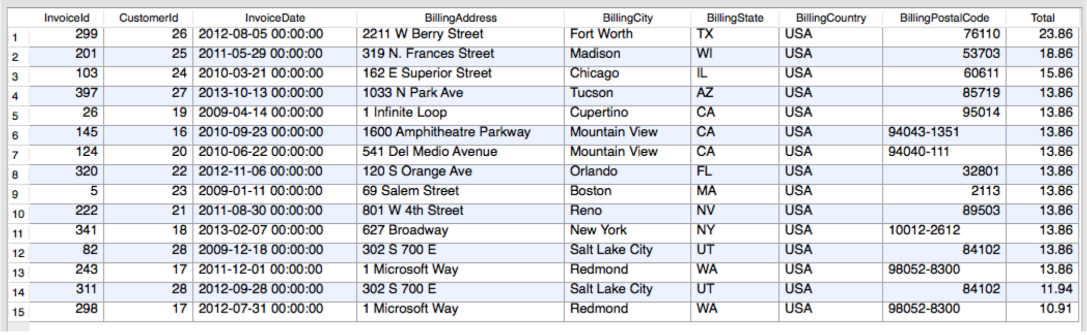
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



	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

Get 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, RDBMSs treat results like bags/multisets (i.e. duplicates allowed)

- Use **DISTINCT** to remove duplicates

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

Example

```sql
SELECT BillingState
FROM invoice
WHERE BillingCountry='USA'
ORDER BY BillingState;
```

vs.

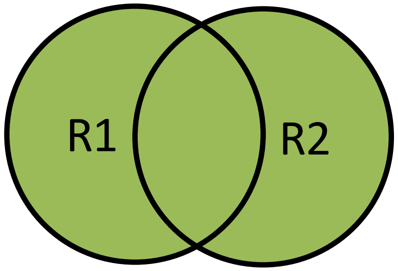
```sql
SELECT DISTINCT BillingState
FROM invoice
WHERE BillingCountry='USA'
ORDER BY BillingState;
```

Set Operations

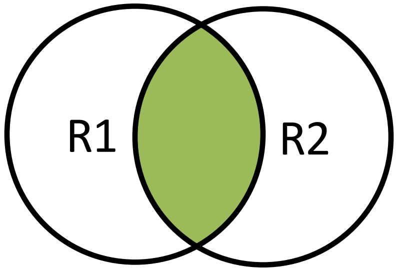
Use **UNION**, **INTERSECT**, **EXCEPT**/**MINUS** to combine results from queries

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

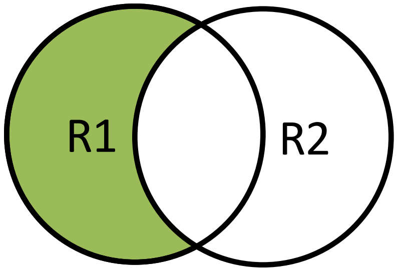
R1 UNION R2



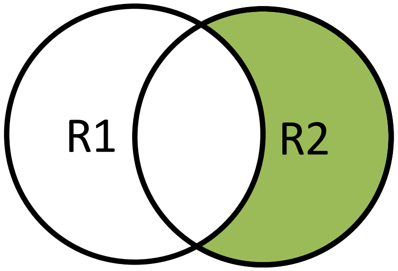
R1 INTERSECT R2



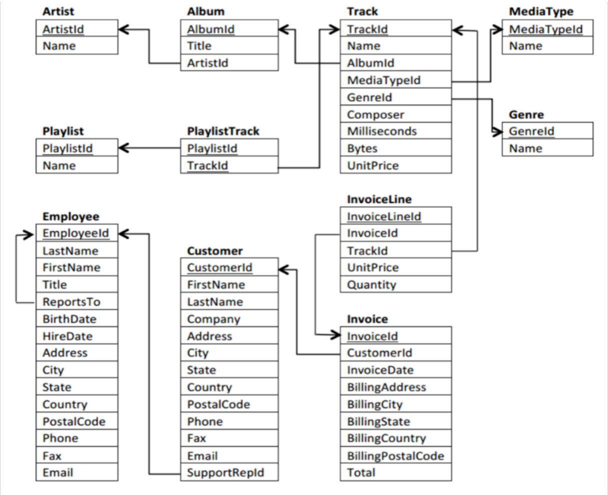
R1 MINUS R2



R2 MINUS R1



Combining Queries (1)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

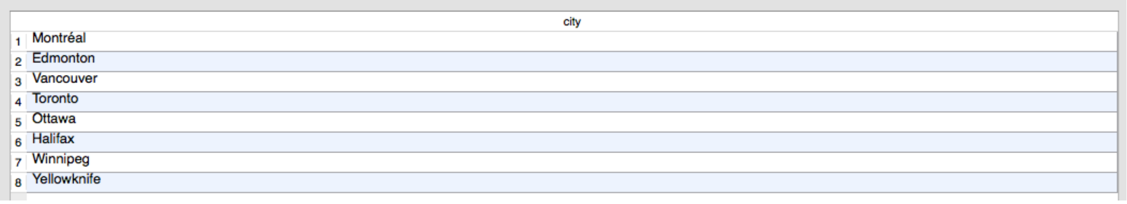
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

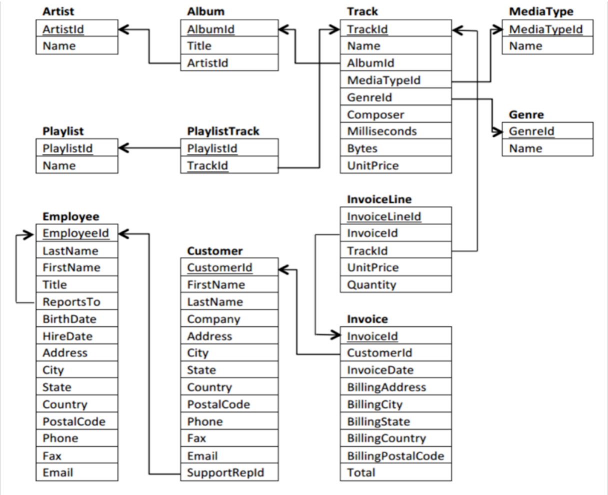


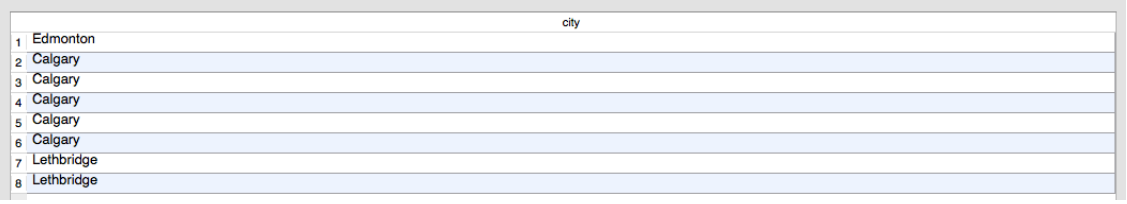
	city
1	Montréal
2	Edmonton
3	Vancouver
4	Toronto
5	Ottawa
6	Halifax
7	Winnipeg
8	Yellowknife

Get 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)



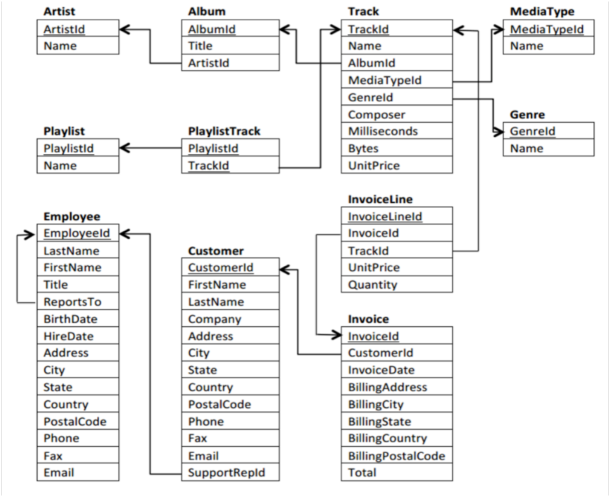


	city
1	Edmonton
2	Calgary
3	Calgary
4	Calgary
5	Calgary
6	Calgary
7	Lethbridge
8	Lethbridge

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

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

Combining Queries (3)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

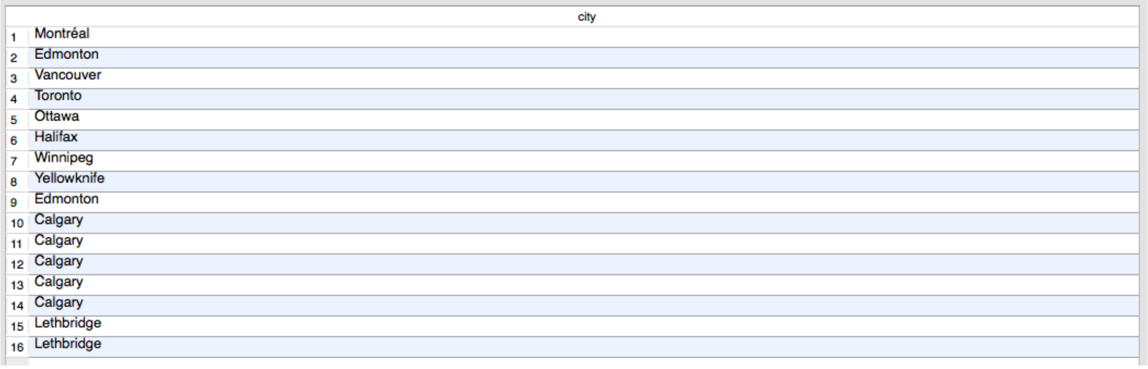
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

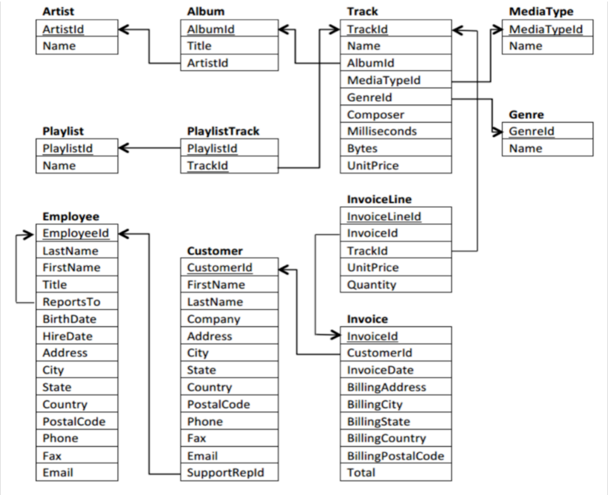


	city
1	Montréal
2	Edmonton
3	Vancouver
4	Toronto
5	Ottawa
6	Halifax
7	Winnipeg
8	Yellowknife
9	Edmonton
10	Calgary
11	Calgary
12	Calgary
13	Calgary
14	Calgary
15	Lethbridge
16	Lethbridge

Get 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)



Artist: ArtistId, Name

Album: AlbumId, Title, ArtistId

Track: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice

MediaType: MediaTypeId, Name

Genre: GenreId, Name

Playlist: PlaylistId, Name

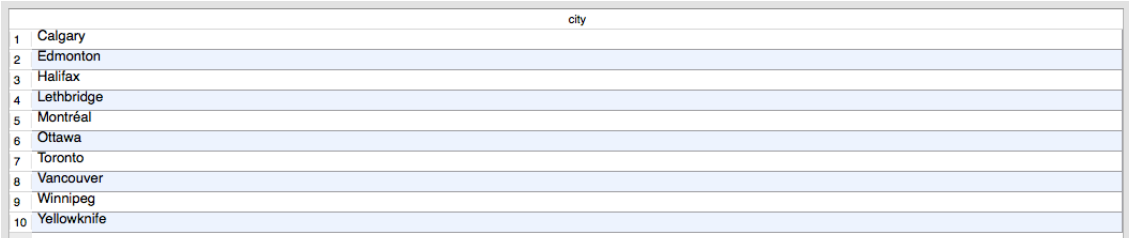
PlaylistTrack: PlaylistId, TrackId

InvoiceLine: InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity

Employee: EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, Country, PostalCode, Phone, Fax, Email

Customer: CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId

Invoice: InvoiceId, CustomerId, InvoiceDate, BillingAddress, BillingCity, BillingState, BillingCountry, BillingPostalCode, Total

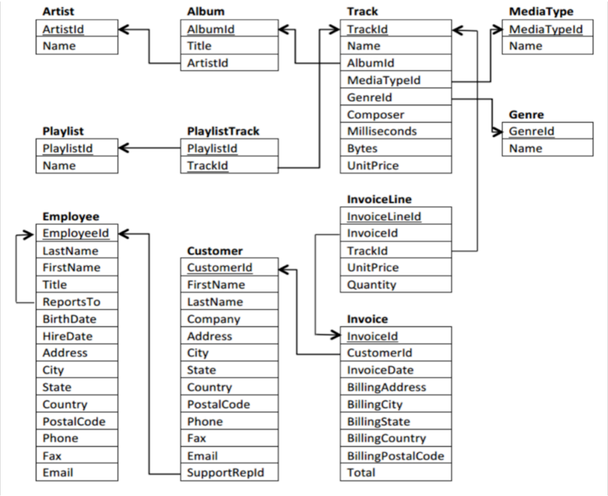


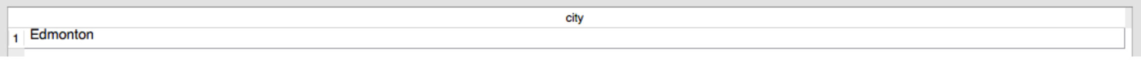
	city
1	Calgary
2	Edmonton
3	Halifax
4	Lethbridge
5	Montréal
6	Ottawa
7	Toronto
8	Vancouver
9	Winnipeg
10	Yellowknife

Get 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)





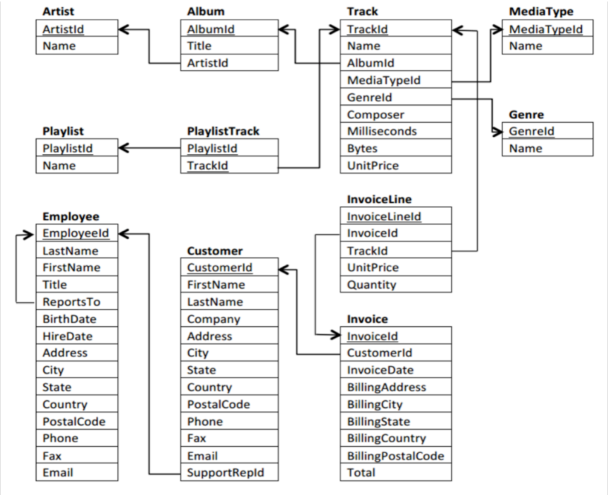
```
  city
1 Edmonton
```

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

[no MySQL support]

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

Combining Queries (6)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

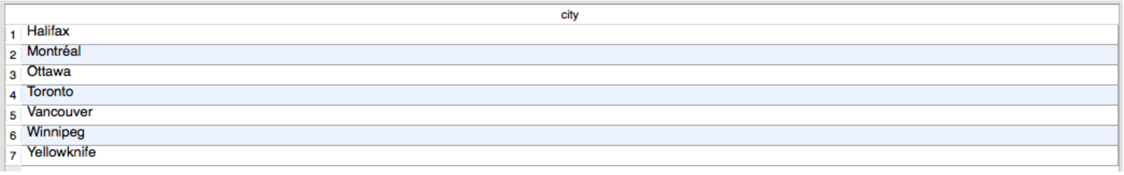
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



	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)

[no MySQL support]

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

Joining Multiple Tables

- SQL supports two methods of **joining** tables, both of which expand the `FROM` clause
  - – Basic idea: take Cartesian product of rows, filter

- The first is called a "soft join" and is older and less expressive
  - – Not recommended
  - – Not covered in detail

- The second uses the `JOIN` keyword and supports more functionality

Intuition: Cartesian Product, Filter (1)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
x	i
y	ii

ALPHA X BETA

Alpha.a	Alpha.b	Beta.c	Beta.d
x	1	x	i
x	1	y	ii
y	2	x	i
y	2	y	ii
z	3	x	i
z	3	y	ii

Intuition: Cartesian Product, Filter (2)

Alpha

a	b
x	1
y	2
z	3

Beta

c	d
x	i
y	ii

Alpha X Beta | Alpha.a = Beta.c

Alpha.a	Alpha.b	Beta.c	Beta.d
x	1	x	i
y	2	y	ii
y	2	x	i
y	2	y	ii
z	3	x	i
z	3	y	ii

Simple Join

STUDENT

Name	SSN	Phone	Address	Age	GPA
Ben Bayer	305-61-2435	555-1234	1 Foo Lane	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2 Bar Court	25	3.53
Barbara Benson	533-69-1238	555-6758	3 Baz Blvd	19	3.25

GPA
3.21
3.25

Goal: find the GPA of students in MATH650

1. Find all SSN in table Class where Class=MATH650
2. Find all GPA in table Student where SSN=#1

Approach: cross all rows in STUDENT with all rows in CLASS and keep the Student(GPA) of those where STUDENT(SSN)=CLASS(SSN) *and* CLASS(Class)=MATH650

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

Simple Join - JOIN

STUDENT

Name	SSN	Phone	Address	Age	GPA
Ben Bayer	305-61-2435	555-1234	1 Foo Lane	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2 Bar Court	25	3.53
Barbara Benson	533-69-1238	555-6758	3 Baz Blvd	19	3.25

GPA
3.21
3.25

Goal: find the GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650

```
SELECT STUDENT.GPA
FROM STUDENT INNER JOIN CLASS
ON STUDENT.SSN=CLASS.SSN
WHERE CLASS.Class='MATH650';
```

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

Simple Join - Soft

STUDENT

Name	SSN	Phone	Address	Age	GPA
Ben Bayer	305-61-2435	555-1234	1 Foo Lane	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2 Bar Court	25	3.53
Barbara Benson	533-69-1238	555-6758	3 Baz Blvd	19	3.25

GPA
3.21
3.25

Goal: find the GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650

```
SELECT  STUDENT.GPA
FROM STUDENT, CLASS
WHERE STUDENT.SSN=CLASS.SSN AND
CLASS.Class='MATH650';
```

Soft Joins (older style) intermix row filtration with table join conditions

General Syntax

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

Table List

```
(T1 <join type> T2 [ON <condition list>])
     <join type> T3 [ON <condition list>]…
```

Join Types

[INNER] JOIN	Row must exist in both tables
LEFT [OUTER] JOIN	Row must *at least* exist in the table to the left (padded with **NULL**)
RIGHT [OUTER] JOIN	Row must exist *at least* in the table to the right (padded with **NULL**)
FULL OUTER JOIN	Row exists in either table (padded with **NULL**)

Join Type Example (1)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha INNER JOIN Beta ON
Alpha.a=Beta.c
```

Alpha.a	Alpha.b	Beta.c	Beta.d
y	2	y	ii

Join Type Example (2)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha LEFT OUTER JOIN Beta ON
Alpha.a=Beta.c
```

Alpha.a	Alpha.b	Beta.c	Beta.d
x	1	NULL	NULL
y	2	y	ii
z	3	NULL	NULL

Join Type Example (3)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha RIGHT OUTER JOIN Beta ON
Alpha.a=Beta.c
```

Alpha.a	Alpha.b	Beta.c	Beta.d
y	2	y	ii
NULL	NULL	w	-

Join Type Example (4)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha FULL OUTER JOIN Beta ON
Alpha.a=Beta.c
```

Alpha.a	Alpha.b	Beta.c	Beta.d
x	1	NULL	NULL
y	2	y	ii
z	3	NULL	NULL
NULL	NULL	w	-

Notes on Joins

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

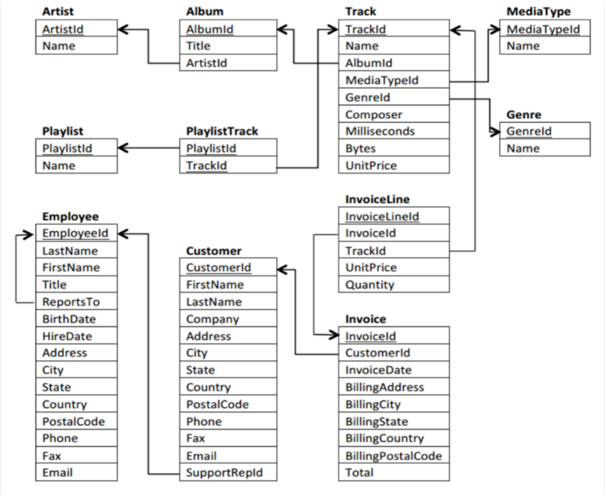
```
SELECT * FROM table1 t1
```

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

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

- When joining, the new set of available attributes (*****) is the <u>concatenation</u> of the attributes from *both* tables

Exploring Joins (1)



Artist
- ArtistId
- Name

Album
- AlbumId
- Title
- ArtistId

Track
- TrackId
- Name
- AlbumId
- MediaTypeId
- GenreId
- Composer
- Milliseconds
- Bytes
- UnitPrice

MediaType
- MediaTypeId
- Name

Genre
- GenreId
- Name

Playlist
- PlaylistId
- Name

PlaylistTrack
- PlaylistId
- TrackId

InvoiceLine
- InvoiceLineId
- InvoiceId
- TrackId
- UnitPrice
- Quantity

Employee
- EmployeeId
- LastName
- FirstName
- Title
- ReportsTo
- BirthDate
- HireDate
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email

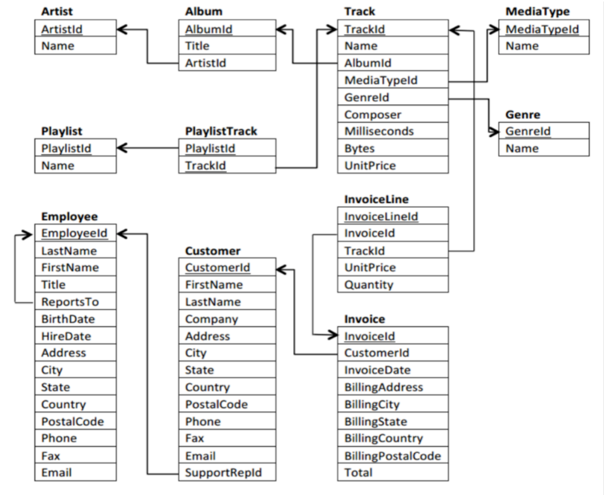
Customer
- CustomerId
- FirstName
- LastName
- Company
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email
- SupportRepId

Invoice
- InvoiceId
- CustomerId
- InvoiceDate
- BillingAddress
- BillingCity
- BillingState
- BillingCountry
- BillingPostalCode
- Total

Get the cross product of genres and media types

```
SELECT *
FROM genre INNER JOIN mediatype;
```

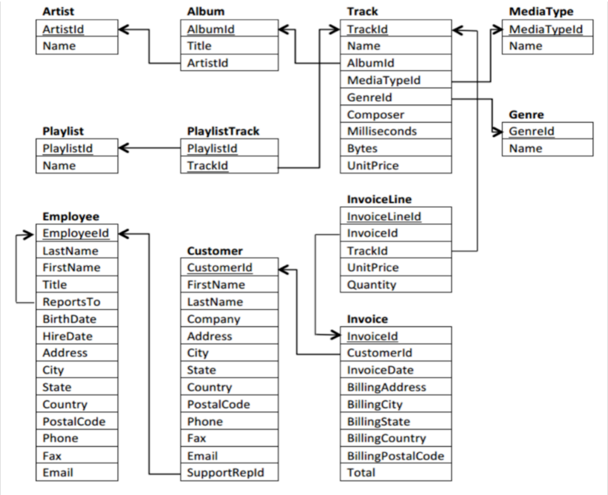
Exploring Joins (2)



Get all track information, with the appropriate genre name and media type name, for all jazz tracks where Miles Davis helped compose

```
SELECT *
FROM (track t INNER JOIN mediatype mt ON t.MediaTypeId=mt.MediaTypeId)
INNER JOIN genre g ON t.GenreId=g.GenreId
WHERE g.Name='Jazz' AND t.Composer LIKE '%Miles Davis%';
```

Advanced Joins (1)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

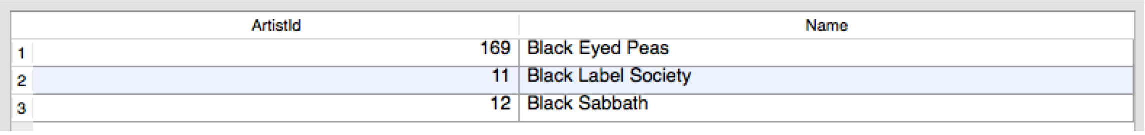
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

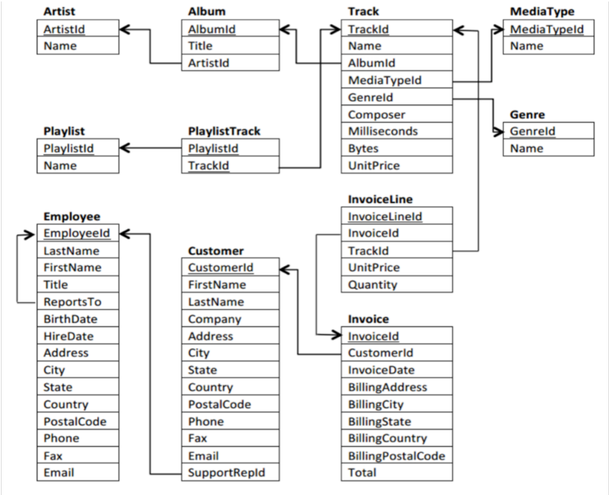


	ArtistId	Name
1	169	Black Eyed Peas
2	11	Black Label Society
3	12	Black Sabbath

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;
```

Advanced Joins (2)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

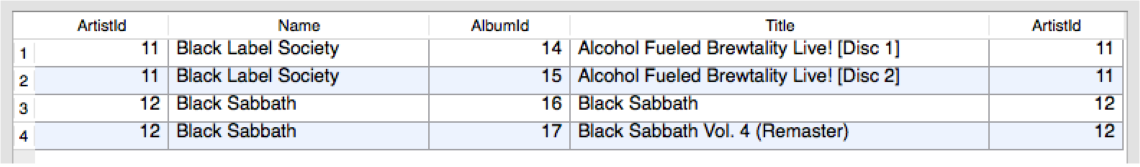
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

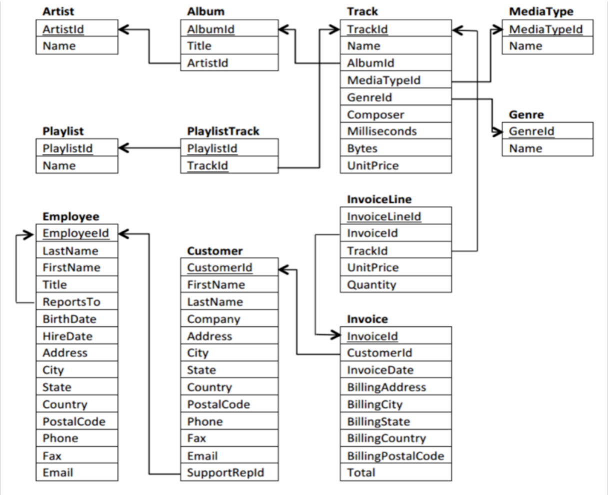


	ArtistId	Name	AlbumId	Title	ArtistId
1	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]	11
2	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]	11
3	12	Black Sabbath	16	Black Sabbath	12
4	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)	12

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;
```

Advanced Joins (3)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

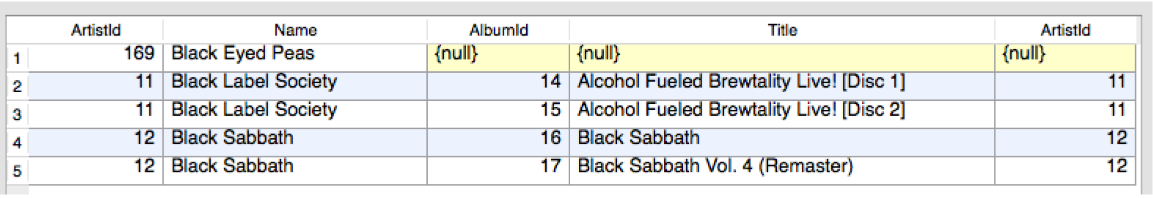
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

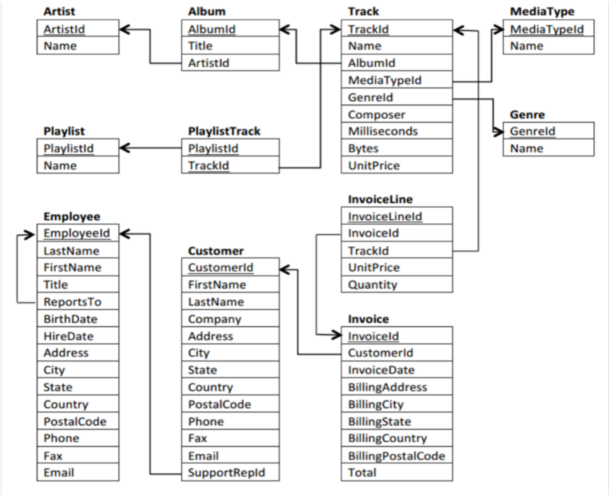


	ArtistId	Name	AlbumId	Title	ArtistId
1	169	Black Eyed Peas	{null}	{null}	{null}
2	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]	11
3	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]	11
4	12	Black Sabbath	16	Black Sabbath	12
5	12	Black Sabbath	17	Black Sabbath Vol.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

Advanced Joins (4)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

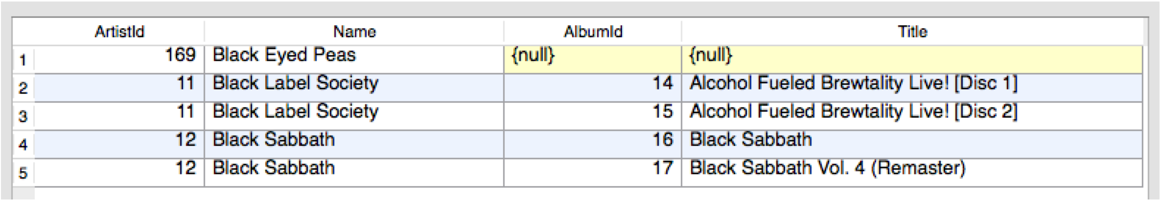
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

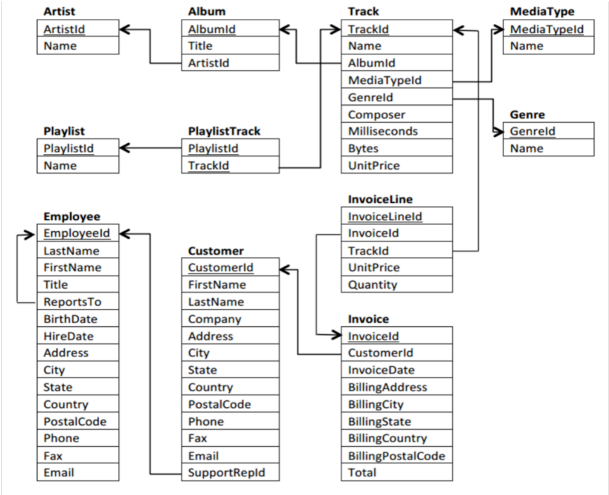


	ArtistId	Name	AlbumId	Title
1	169	Black Eyed Peas	{null}	{null}
2	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]
3	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]
4	12	Black Sabbath	16	Black Sabbath
5	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)

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;
```

Advanced Joins (5)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

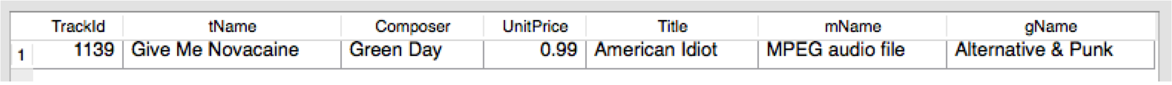
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



	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';
```

Aggregate Function

- An aggregate function takes the value of a field (or an expression over multiple fields) for a set of rows and outputs a single value

- When used alone, an aggregate function reduces a set of rows to a single row

- Common aggregate functions include **MAX**, **MIN**, **SUM**, **AVG**, **COUNT**

Continuing Our Example

STUDENT

Name	SSN	Phone	Address	Age	GPA
Ben Bayer	305-61-2435	555-1234	1 Foo Lane	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2 Bar Court	25	3.53
Barbara Benson	533-69-1238	555-6758	3 Baz Blvd	19	3.25

GPA
3.21
3.25

Goal: find the GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650

```
SELECT STUDENT.GPA
FROM STUDENT INNER JOIN CLASS
ON STUDENT.SSN=CLASS.SSN
WHERE CLASS.Class='MATH650';
```

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

Now Take the Average!

STUDENT

Name	SSN	Phone	Address	Age	GPA
Ben Bayer	305-61-2435	555-1234	1 Foo Lane	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2 Bar Court	25	3.53
Barbara Benson	533-69-1238	555-6758	3 Baz Blvd	19	3.25

aGPA
3.23

Goal: find the average GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650, average result set

```
SELECT AVG(STUDENT.GPA) AS aGPA
FROM STUDENT INNER JOIN CLASS
ON STUDENT.SSN=CLASS.SSN
WHERE CLASS.Class='MATH650';
```

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

Other Examples

- Get the number of tracks for an album

```
SELECT COUNT(*) AS num_tracks 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

- Compute the total cost of an album

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

- Get the 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 aggregate functions. The **GROUP BY** attribute list should be a subset of **SELECT** list.

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

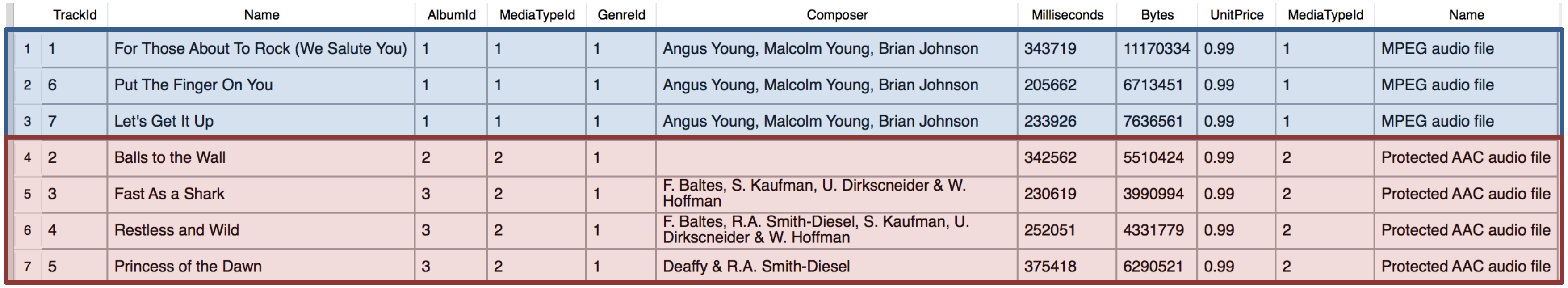
Example: track price stats by media type

```
SELECT mt.Name AS media_type, MIN(t.UnitPrice) AS min_price,
       MAX(t.UnitPrice) AS max_price, AVG(t.UnitPrice) AS avg_price
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
GROUP BY mt.Name
ORDER BY avg_price DESC, mt.Name ASC;
```

Conceptually

```
SELECT mt.Name AS media_type, MIN(t.UnitPrice) AS min_price,
       MAX(t.UnitPrice) AS max_price, AVG(t.UnitPrice) AS avg_price
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
GROUP BY mt.Name
ORDER BY avg_price DESC, mt.Name ASC;
```

```
SELECT *
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
ORDER BY mt.Name ASC;
```

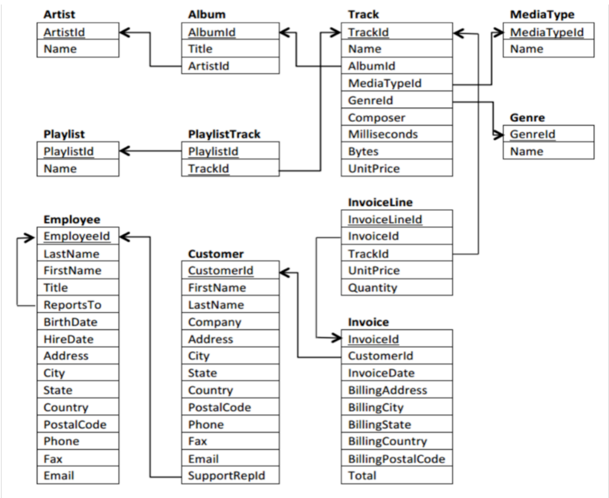


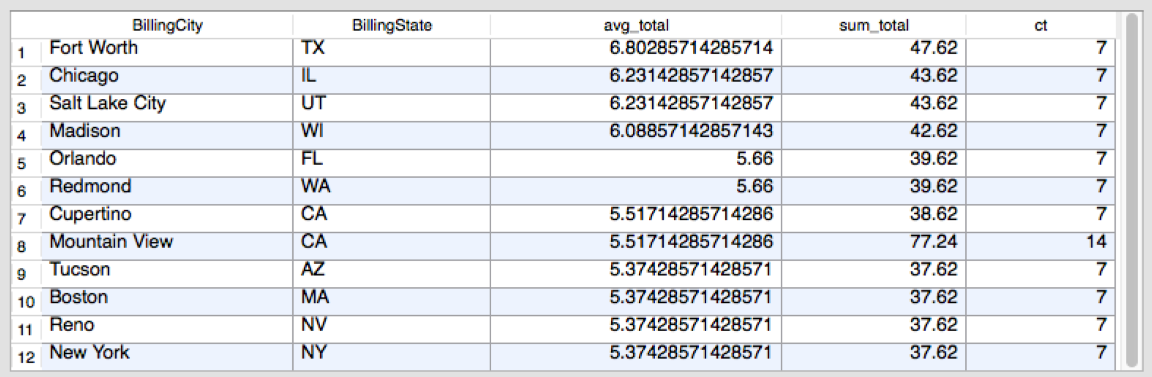
	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice	MediaTypeId	Name
1	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99	1	MPEG audio file
2	6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99	1	MPEG audio file
3	7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99	1	MPEG audio file
4	2	Balls to the Wall	2	2	1		342562	5510424	0.99	2	Protected AAC audio file
5	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	230619	3990994	0.99	2	Protected AAC audio file
6	4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	252051	4331779	0.99	2	Protected AAC audio file
7	5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99	2	Protected AAC audio file

…

GROUP BY

Grouped Aggregation (1)



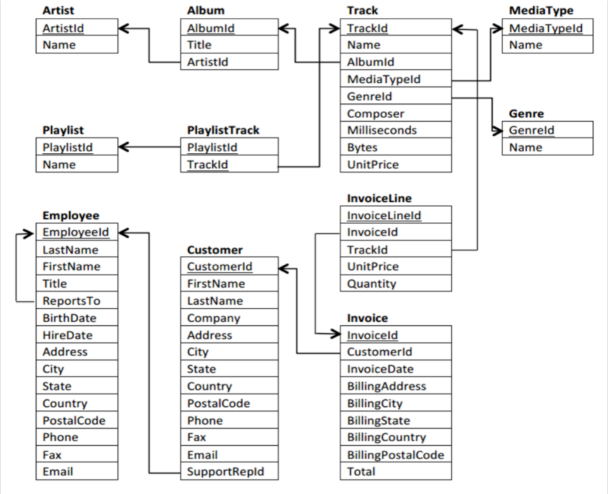


	BillingCity	BillingState	avg_total	sum_total	ct
1	Fort Worth	TX	6.80285714285714	47.62	7
2	Chicago	IL	6.23142857142857	43.62	7
3	Salt Lake City	UT	6.23142857142857	43.62	7
4	Madison	WI	6.08857142857143	42.62	7
5	Orlando	FL	5.66	39.62	7
6	Redmond	WA	5.66	39.62	7
7	Cupertino	CA	5.51714285714286	38.62	7
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;
```

Grouped Aggregation (2)



Artist: ArtistId, Name

Album: AlbumId, Title, ArtistId

Track: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice

MediaType: MediaTypeId, Name

Genre: GenreId, Name

Playlist: PlaylistId, Name

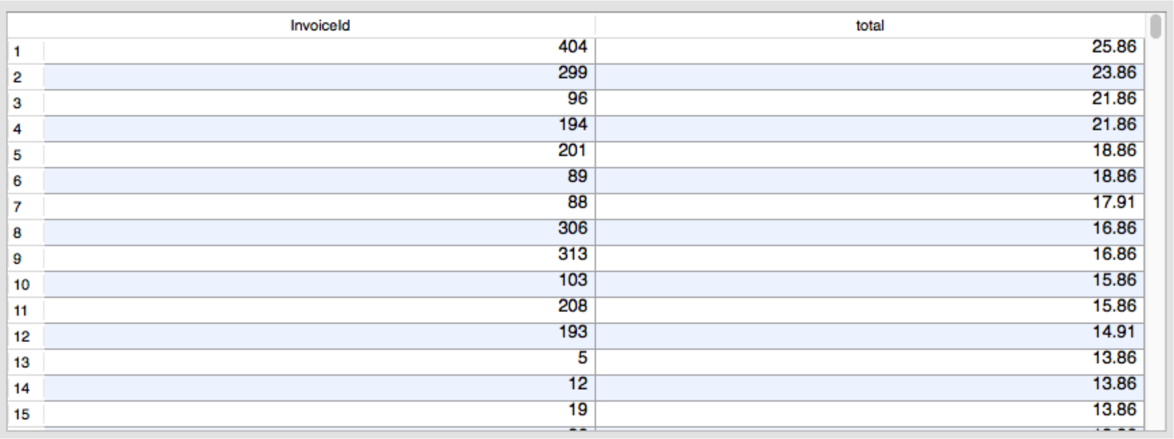
PlaylistTrack: PlaylistId, TrackId

InvoiceLine: InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity

Employee: EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, Country, PostalCode, Phone, Fax, Email

Customer: CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId

Invoice: InvoiceId, CustomerId, InvoiceDate, BillingAddress, BillingCity, BillingState, BillingCountry, BillingPostalCode, Total

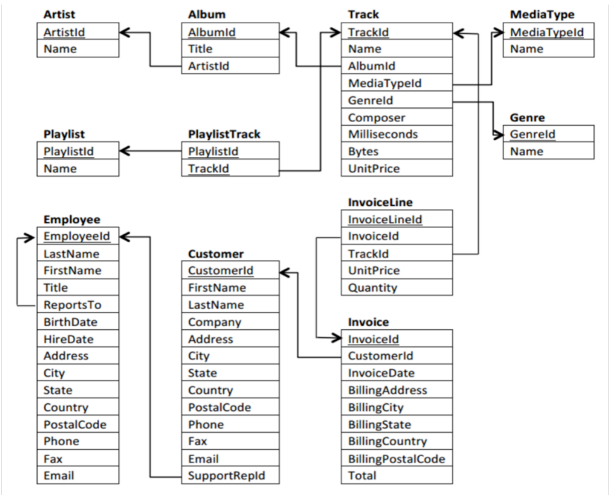


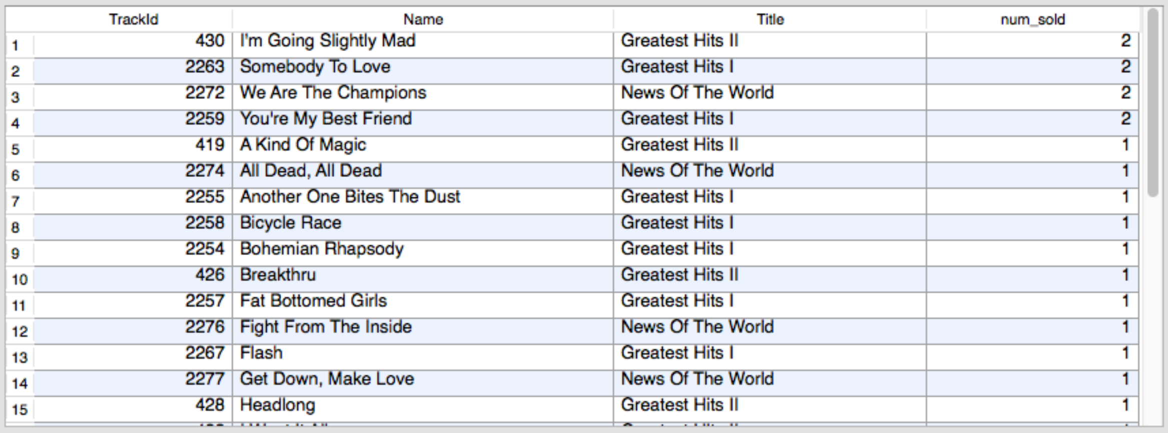
	InvoiceId	total
1	404	25.86
2	299	23.86
3	96	21.86
4	194	21.86
5	201	18.86
6	89	18.86
7	88	17.91
8	306	16.86
9	313	16.86
10	103	15.86
11	208	15.86
12	193	14.91
13	5	13.86
14	12	13.86
15	19	13.86

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;
```

Grouped 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).

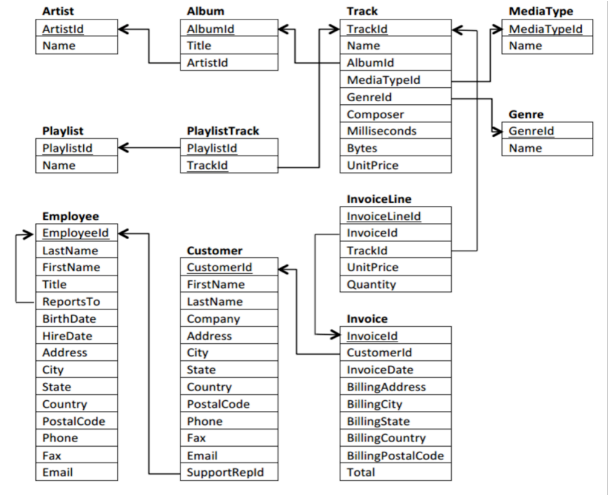
```
SELECT invoiceline.TrackId, track.Name, album.Title,
       SUM(invoiceline.Quantity) AS num_sold
FROM ((invoiceline INNER JOIN track ON invoiceline.TrackId=track.TrackId)
INNER JOIN album ON track.AlbumId=album.AlbumId)
INNER JOIN artist ON album.ArtistId=artist.ArtistId
WHERE artist.Name='Queen'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

HAVING

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

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

Aggregation (4)



Artist: ArtistId, Name

Album: AlbumId, Title, ArtistId

Track: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice

MediaType: MediaTypeId, Name

Genre: GenreId, Name

Playlist: PlaylistId, Name

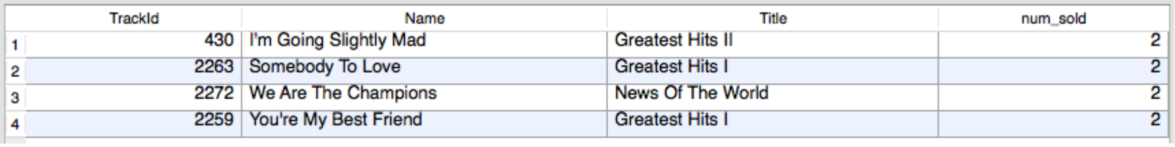
PlaylistTrack: PlaylistId, TrackId

InvoiceLine: InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity

Employee: EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, Country, PostalCode, Phone, Fax, Email

Customer: CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId

Invoice: InvoiceId, CustomerId, InvoiceDate, BillingAddress, BillingCity, BillingState, BillingCountry, BillingPostalCode, Total



	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

A feature of SQL is its *composability* – the result(s) of one query, which is a set of rows/columns, can be used by another

- Termed inner/nested query or subquery

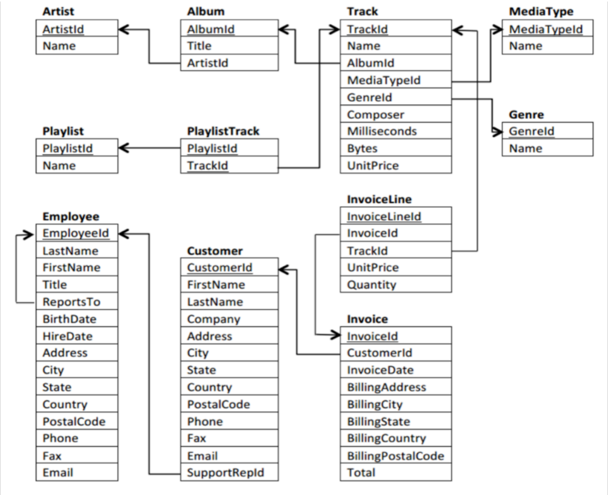
Most common locations

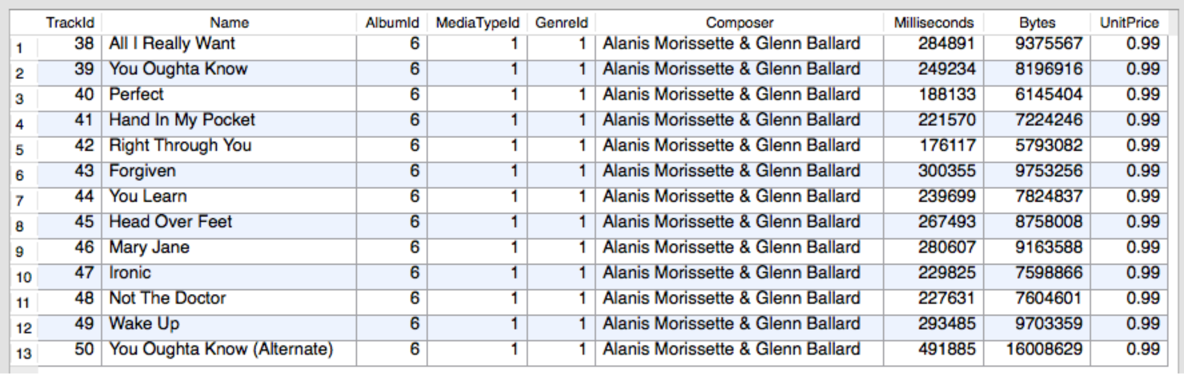
- **SELECT** (returns a value for an attribute)
- **FROM** (becomes a "table" to query/join)
- **WHERE** (serves as part of a constraint)

Notes about Subqueries

- Tip: when designing subqueries, work inside out – come up with each query separately, then piece them together
  - Helps with debugging
- A **correlated** subquery is an *inner* query that references a value from an *outer* query
  - The inner query will be run once for *every* tuple of the outer query (i.e. slow!)
- Don't use `ORDER BY` in inner queries (some DBMSs don't allow, typically wasteful anyhow)

Example: WHERE





	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'
);
```

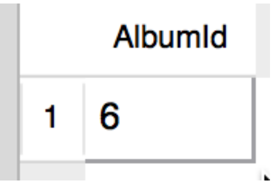
Notes

1. **The subquery needs to return a *single* value for the = to make sense**
2. **Not correlated!**

How the Query Works Conceptually

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

Inner Query



	AlbumId
1	6

```
SELECT t.*
FROM track t
WHERE t.AlbumId = 6;
```

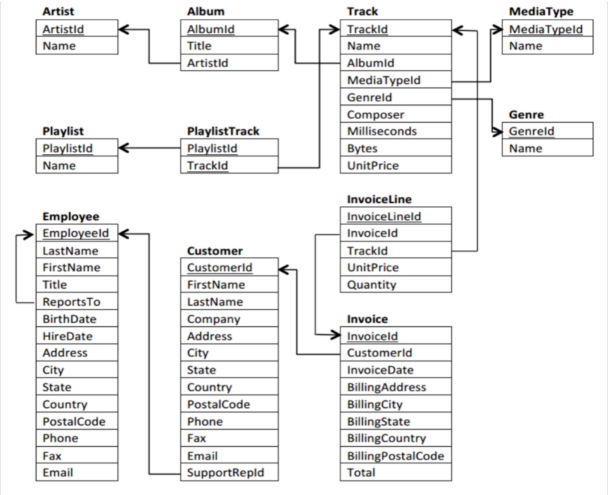
Notes about Subqueries and `WHERE`

For most operators, the subquery will need to return a single value

Other operators:

- [`NOT`] `IN` = query returns a single column of options
- [`NOT`] `EXISTS` = checks if query returns at least a single row
- <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)

Nesting Example: WHERE



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

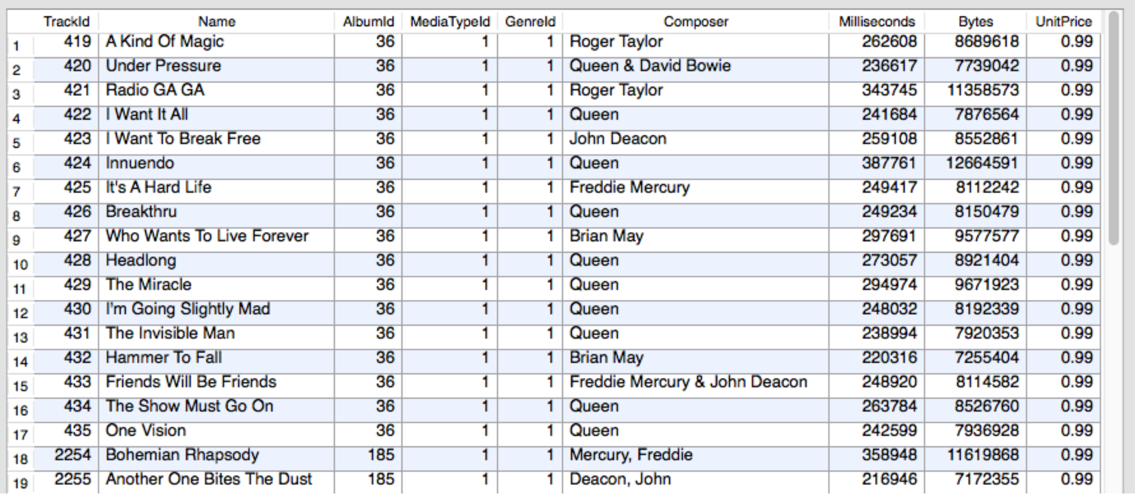
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



	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)

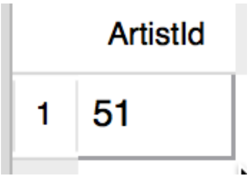
```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (
   SELECT alb.AlbumId
   FROM album alb
   WHERE alb.ArtistId = (
        SELECT art.ArtistId
        FROM artist art
        WHERE art.Name='Queen'
   )
);
```

Notes

1. **Not correlated!**

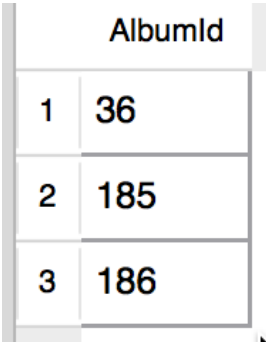
How the Query Works Conceptually

```
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'
   )
);
```



	ArtistId
1	51

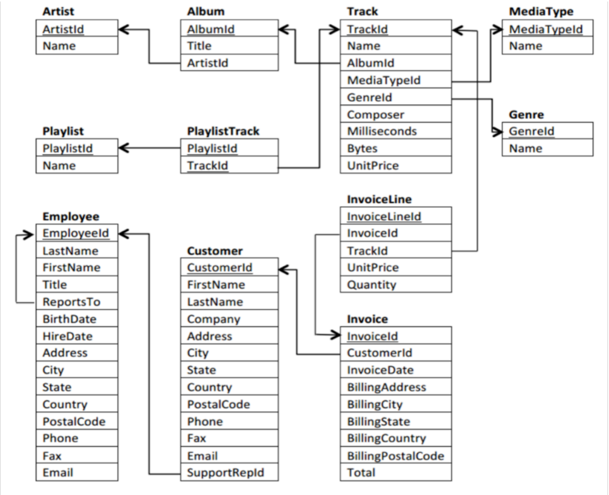
```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (
   SELECT alb.AlbumId
   FROM album alb
   WHERE alb.ArtistId = 51
);
```



	AlbumId
1	36
2	185
3	186

```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (36, 185, 186);
```

Example: SELECT



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

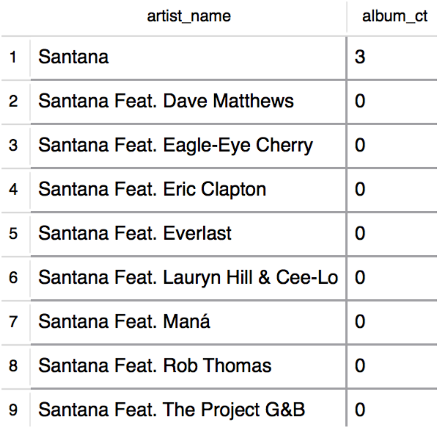
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



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

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

```
SELECT art.Name AS artist_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, art.Name;
```

Notes

1. The subquery needs to return a *single* value for each tuple generated
2. Correlated subquery!

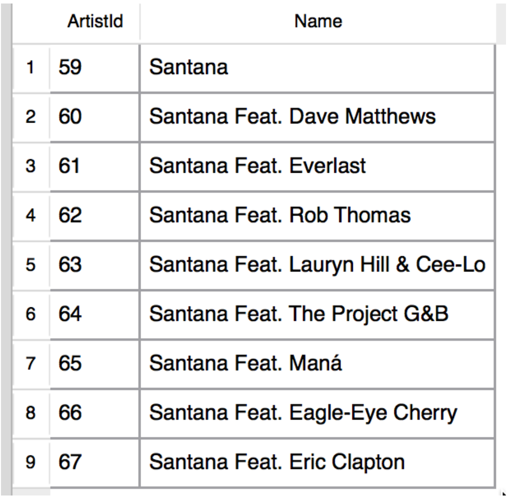
How the Query Works Conceptually

```
SELECT art.Name AS artist_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, art.Name;
```

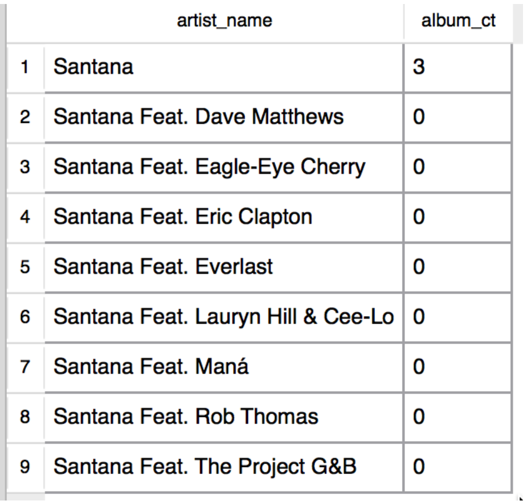
Correlated - one query per row to fill in album_ct column!

```
SELECT COUNT(*)
FROM album alb
WHERE alb.ArtistId=59;
                  =60;
                  …
```

```
SELECT *
FROM artist art
WHERE art.Name LIKE 'Santana%';
```

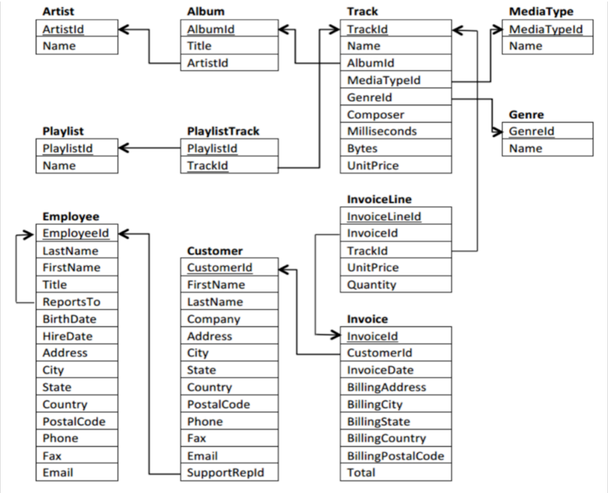


	ArtistId	Name
1	59	Santana
2	60	Santana Feat. Dave Matthews
3	61	Santana Feat. Everlast
4	62	Santana Feat. Rob Thomas
5	63	Santana Feat. Lauryn Hill & Cee-Lo
6	64	Santana Feat. The Project G&B
7	65	Santana Feat. Maná
8	66	Santana Feat. Eagle-Eye Cherry
9	67	Santana Feat. Eric Clapton



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

[Better] Example: FROM



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

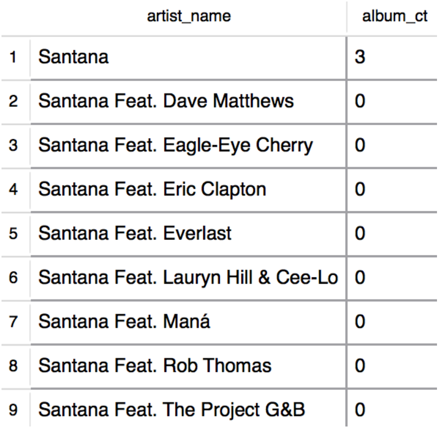
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



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

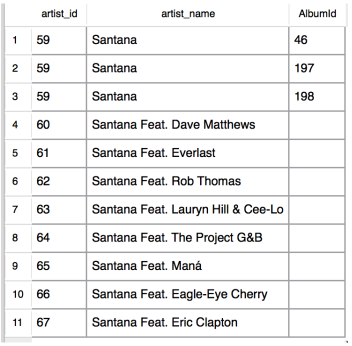
For each artist starting with Santana, get the number of albums, sorted by count (greatest first), then artist (alphabetical)

```
SELECT artist_name, COUNT(q1.AlbumId) AS album_ct
FROM
(
   SELECT art.ArtistId AS artist_id, art.Name AS artist_name, alb.AlbumId
   FROM artist art LEFT JOIN album alb ON art.ArtistId=alb.ArtistId
   WHERE art.Name LIKE 'Santana%'
) q1
GROUP BY artist_id
ORDER BY album_ct DESC, artist_name;
```

How the Query Works Conceptually

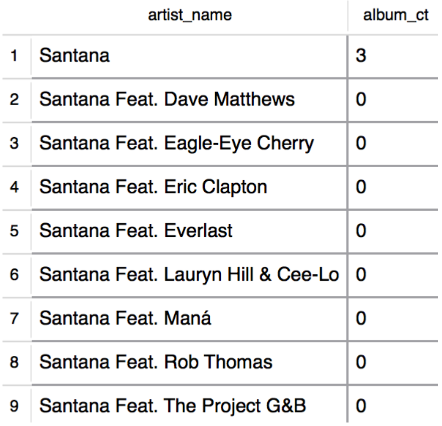
```
SELECT artist_name, COUNT(q1.AlbumId) AS album_ct
FROM
(
   SELECT art.ArtistId AS artist_id, art.Name AS artist_name, alb.AlbumId
   FROM artist art LEFT JOIN album alb ON art.ArtistId=alb.ArtistId
   WHERE art.Name LIKE 'Santana%'
) q1
GROUP BY artist_id
ORDER BY album_ct DESC, artist_name;
```

q1



	artist_id	artist_name	AlbumId
1	59	Santana	46
2	59	Santana	197
3	59	Santana	198
4	60	Santana Feat. Dave Matthews	
5	61	Santana Feat. Everlast	
6	62	Santana Feat. Rob Thomas	
7	63	Santana Feat. Lauryn Hill & Cee-Lo	
8	64	Santana Feat. The Project G&B	
9	65	Santana Feat. Maná	
10	66	Santana Feat. Eagle-Eye Cherry	
11	67	Santana Feat. Eric Clapton	

```
SELECT artist_name, COUNT(q1.AlbumId) AS album_ct
FROM q1
GROUP BY artist_id
ORDER BY album_ct DESC, artist_name;
```



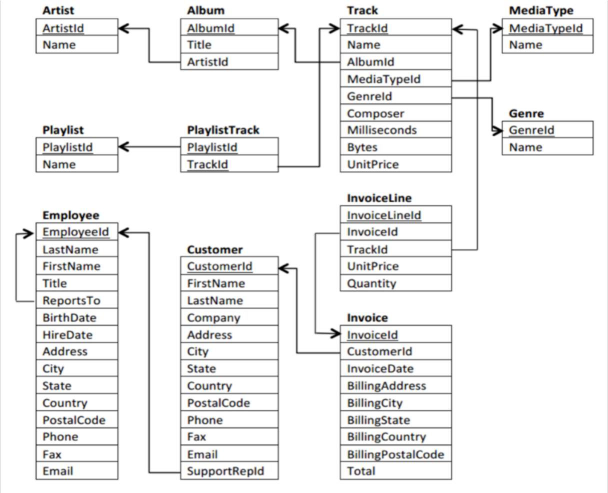
	artist_name	album_ct
1	Santana	3
2	Santana Feat. Dave Matthews	0
3	Santana Feat. Eagle-Eye Cherry	0
4	Santana Feat. Eric Clapton	0
5	Santana Feat. Everlast	0
6	Santana Feat. Lauryn Hill & Cee-Lo	0
7	Santana Feat. Maná	0
8	Santana Feat. Rob Thomas	0
9	Santana Feat. The Project G&B	0

Notes about Subqueries and `FROM`

- When using one or more subqueries in the `FROM` clause, remember two important items
  - The subquery must be enclosed within parentheses
  - The subquery must have a name (e.g. **`q1`** in the previous example), which is indicated just after the close parenthesis

- The name can be used to refer to columns in the subquery via the dot notation (e.g. subqueryname.columnname) – this is required if the column name is not unique

Nesting Example: FROM



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

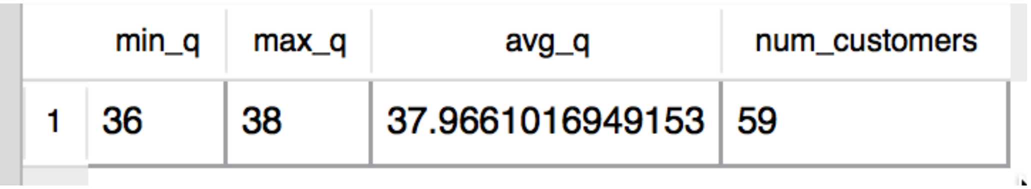
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total



	min_q	max_q	avg_q	num_customers
1	36	38	37.9661016949153	59

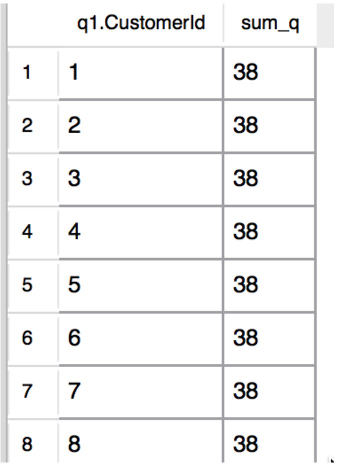
Find the minimum, maximum, and average number of tracks ordered per customer (across all invoices). Also include the total number of customers.

```
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;
```

How the Query Works Conceptually

```
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;
```

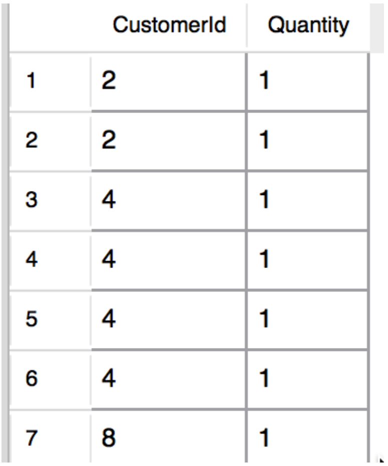
q2



	q1.CustomerId	sum_q
1	1	38
2	2	38
3	3	38
4	4	38
5	5	38
6	6	38
7	7	38
8	8	38

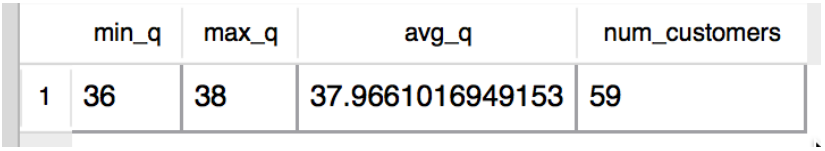
…

q1



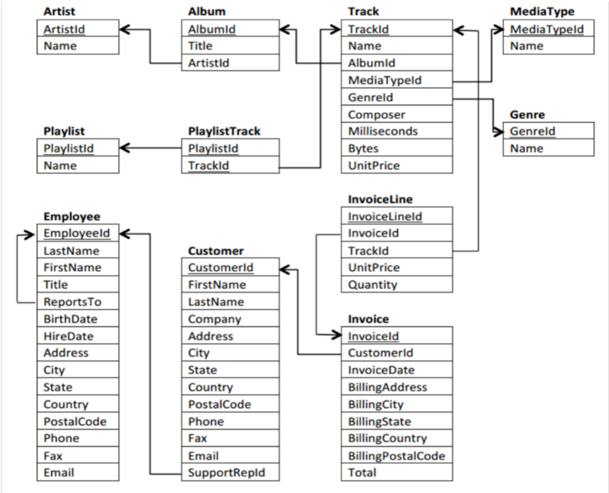
	CustomerId	Quantity
1	2	1
2	2	1
3	4	1
4	4	1
5	4	1
6	4	1
7	8	1

…



	min_q	max_q	avg_q	num_customers
1	36	38	37.9661016949153	59

Subquery (1)



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

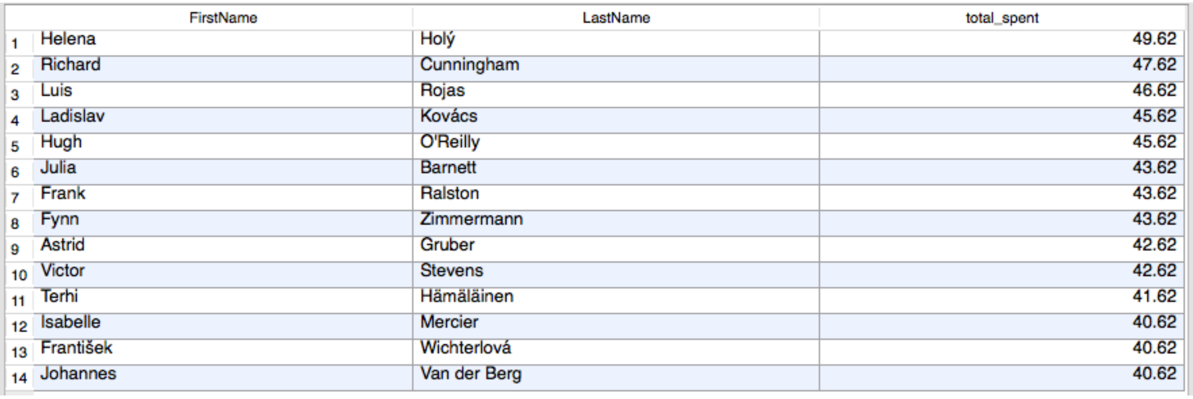
PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

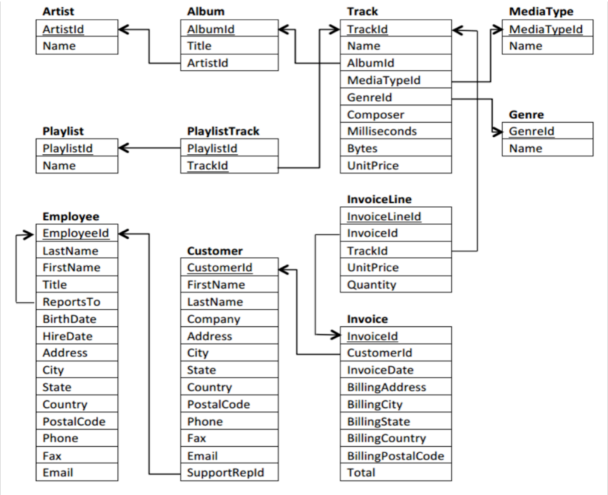


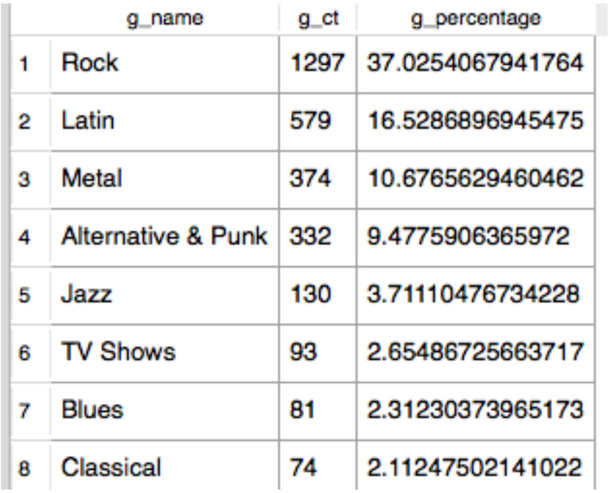
	FirstName	LastName	total_spent
1	Helena	Holý	49.62
2	Richard	Cunningham	47.62
3	Luis	Rojas	46.62
4	Ladislav	Kovács	45.62
5	Hugh	O'Reilly	45.62
6	Julia	Barnett	43.62
7	Frank	Ralston	43.62
8	Fynn	Zimmermann	43.62
9	Astrid	Gruber	42.62
10	Victor	Stevens	42.62
11	Terhi	Hämäläinen	41.62
12	Isabelle	Mercier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

Subquery (2)





	g_name	g_ct	g_percentage
1	Rock	1297	37.0254067941764
2	Latin	579	16.5286896945475
3	Metal	374	10.6765629460462
4	Alternative & Punk	332	9.4775906365972
5	Jazz	130	3.71110476734228
6	TV Shows	93	2.65486725663717
7	Blues	81	2.31230373965173
8	Classical	74	2.11247502141022

Create a report of the distribution of tracks into genres. The result set should list each genre by name, the number of tracks of that genre, and the percentage of overall tracks for that genre. The rows should be sorted by the percentage (greatest first), then genre name (alphabetically).

```
SELECT x.Name AS g_name, x.g_ct AS g_ct, (100.0 * g_ct / ct) AS g_percentage
FROM (SELECT *, (SELECT COUNT(*) FROM track t1 WHERE t1.GenreId=g.GenreId) AS g_ct,
                (SELECT COUNT(*) FROM track t2) AS ct
      FROM genre g) x
ORDER BY g_percentage DESC, g_name ASC;
```

Inserting Rows

- Insert all attributes, in same order as table

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

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

```
INSERT INTO table_name (a1, a2, … an)
VALUES (a, b, … n)[, (a2, b2, … n2), …];
```

- Insert via query

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

Updating Rows

General syntax

```
UPDATE table_name
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_name
[WHERE <condition list>];
```

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

Summary

- You have now learned most of the DML components of SQL
  - **SELECT**: get stuff out
  - **INSERT**: add row(s)
  - **UPDATE**: change existing row(s)
  - **DELETE**: remove row(s)

- While using **SELECT** you learned about attribute ordering/renaming (**AS**), row filtering (**WHERE**) and sorting (**ORDER BY**), table joining (**FROM** + **JOIN**/**ON**), grouped aggregation (**GROUP BY** + **FN** + **HAVING**), set operations on multiple queries (e.g. **UNION**), and subqueries (**SELECT** within **SELECT**)