SQL: Part 1
DML, Relational Algebra

Lecture 3

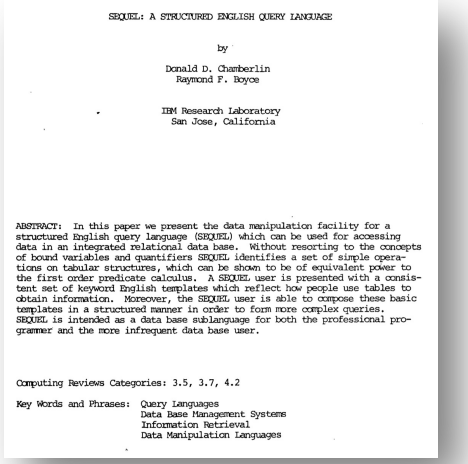
Relational Algebra

- The basic set of operations for the relational model
  - Note that the relational model assumes sets, so some of the database operations will not map
- Allows the user to formally express a retrieval over one or more relations, as a **relational algebra expression**
  - Results in a new relation, which could itself be queried (i.e. **composable**)
- Why is RA important?
  - Formal basis for SQL
  - Used in query optimization
  - Common vocabulary in data querying technology
  - Sometimes easier to understand the flow of complex SQL

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 (-2023)
  - 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
  - …

Selection

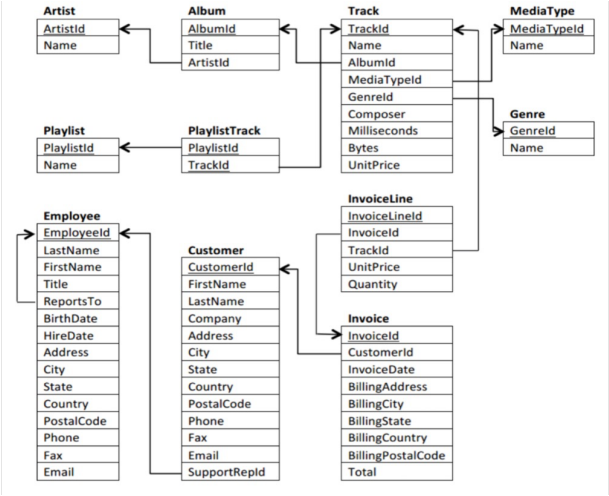
- Our first operation will be to select some tuples from a relation
- This corresponds to the SELECT relational algebra operator (σ; sigma)
  - General form: $\sigma_{<condition>}(Relation)$
- In SQL this corresponds to the **`SELECT`** statement

SQL: Simplest Selection

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

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

Your First Query!



Get all information about all artists

```
SELECT *
FROM artist;
```

$$\sigma_{true}(artist)$$

Projection/Renaming

- The ability to select a subset of columns from a relation, discarding the rest, is achieved via the PROJECT operator (π; pi)
  - General form: $\pi_{\text{<attribute list>}}(\text{Relation})$
  - The "attribute list" can include function(s) on existing attributes
- The ability to rename a relation and/or list of attributes is achieved via the RENAME operator (ρ; rho)
  - General form: $\rho_{\text{<new relation name>(new attribute names)}}(\text{Relation})$
- In SQL these get mapped to the attribute list of the **`SELECT`** statement (+ the **`AS`** modifier)

SQL: 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 (1)

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

```
SELECT *
FROM employee;
```

$\sigma_{true}(employee)$

- For a subset, simply list them (comma separated)

```
SELECT FirstName, LastName
FROM employee;
```

$\pi_{FirstName,LastName}(\sigma_{true}(employee))$

- To rename (or *alias*) an attribute in the result, use **AS**

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

$\rho_{(fname,\ lname)}(\pi_{FirstName,LastName}(\sigma_{true}(employee)))$

Attribute List (2)

- In relational algebra, you can optionally show a sequence of steps, giving a name to intermediate relations

$$\rho_{(fname,\ lname)}(\pi_{FirstName,LastName}(\sigma_{true}(employee)))$$

vs

$$ALL\_E \leftarrow \sigma_{true}(employee)$$
$$NAME\_E \leftarrow \pi_{FirstName,LastName}(ALL\_E)$$
$$RESULT \leftarrow \rho_{(fname,\ lname)}(NAME\_E)$$

Attribute List (3)

- In projection, an attribute can be the result of an expression relating existing attributes
  - Available functions depend upon DBMS
  - It is good form to RENAME the result (and makes it easier to access contents via code)

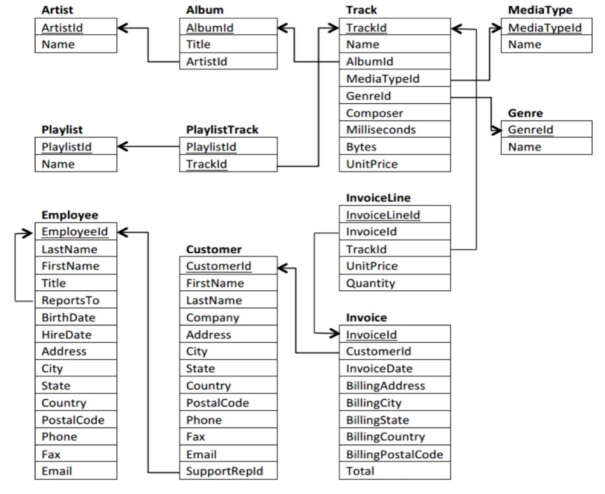
```
SELECT
    InvoiceId, InvoiceLineId,
    (UnitPrice*Quantity) AS cost
FROM invoiceline;
```

$ALL\_ILINES \leftarrow \sigma_{true}(invoiceline)$

$ILINE\_INFO \leftarrow \pi_{InvoiceId,InvoiceLineId,UnitPrice*Quantity}(ALL\_ILINES)$

$RESULT \leftarrow \rho_{(InvoiceId,InvoiceLineId,cost)}(ILINE\_INFO)$

Basic Queries (1)

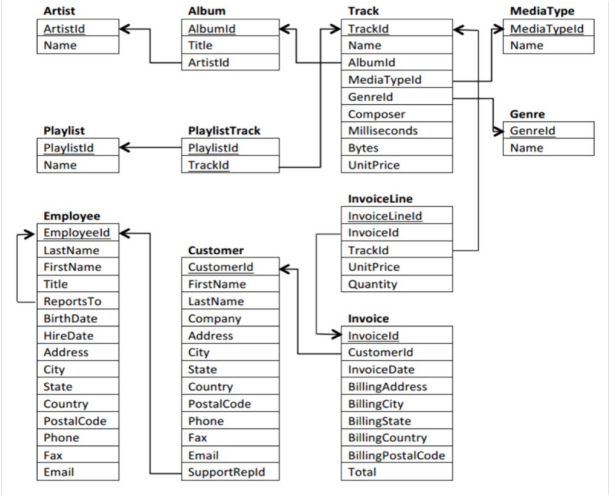


Get all artist names

```
SELECT Name
FROM artist;
```

$$\pi_{Name}(\sigma_{true}(artist))$$

Basic Queries (2)



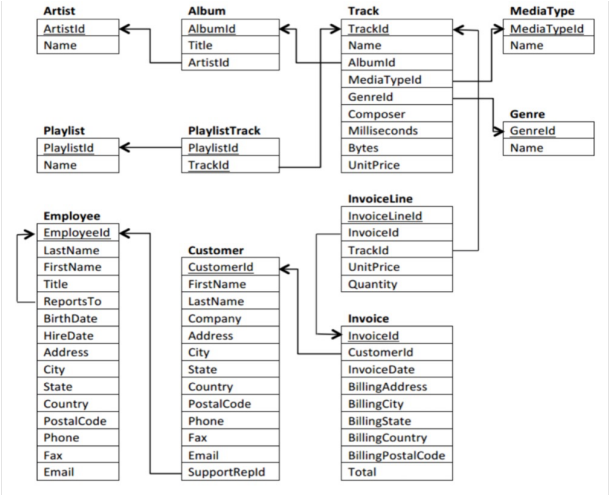
Get all employee names (first & last), with their full address info (address, city, state, zip, country)

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

$$ALL\_E \leftarrow \sigma_{true}(employee)$$

$$RESULT \leftarrow \pi_{FirstName,LastName,Address,City,State,PostalCode,Country}(ALL\_E)$$

Basic Queries (3)



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

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

$$\pi_{InvoiceId,UnitPrice,Quantity}(\sigma_{true}(invoiceline))$$

Conditional Selection

- Thus far we have included all tuples in a relation
- However, the condition clause of the SELECT operator permits Boolean expressions to restrict included rows
- This corresponds to the **`WHERE`** clause of the SQL SELECT statement

SQL: 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 condition(s) are returned. Result set order is arbitrary.

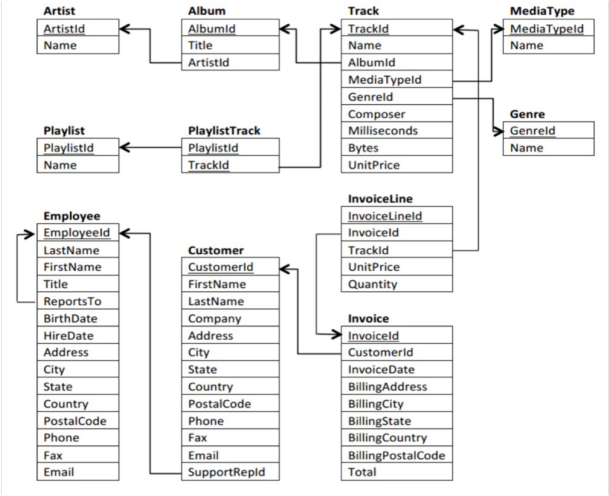
Condition List ~ Boolean Expression

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`

*There are actually is no concept of NULL in relational algebra

Conditional Query (1)

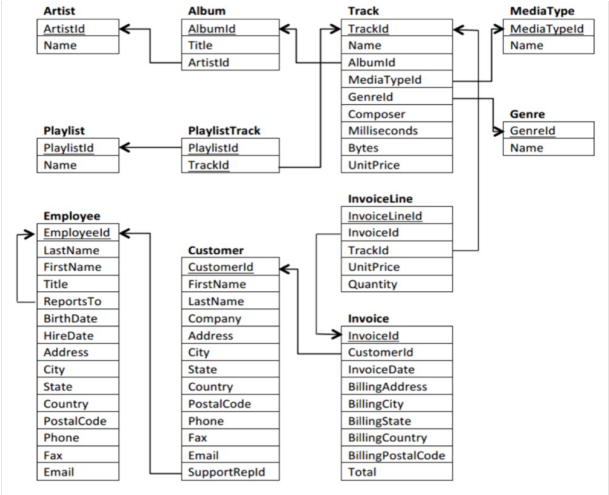


Get the billing country of all invoices totaling more than $10

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

$$\pi_{BillingCountry}(\sigma_{Total>10}(invoice))$$

Conditional Query (2)

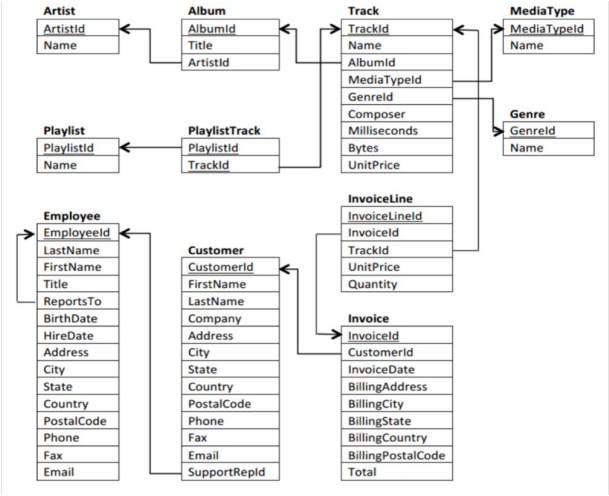


Get all information about tracks whose name contains the word "Rock"

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

$$\sigma_{Name\ LIKE\ '\%Rock\%'}(track)$$

Conditional Query (3)



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' );
```

$$\pi_{FirstName,LastName}(\sigma_{ReportsTo \neq EmployeeId \ AND \ City='Calgary'}(employee))$$

Since RA doesn't have NULL, we could imagine having the Boss report to only themself

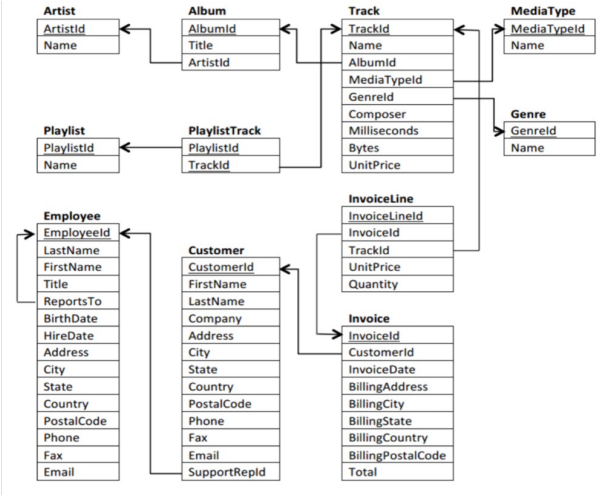
Non-Standard Functions

- SQLite
  - http://sqlite.org/lang.html
- MariaDB
  - https://mariadb.com/kb/en/sql-statements/

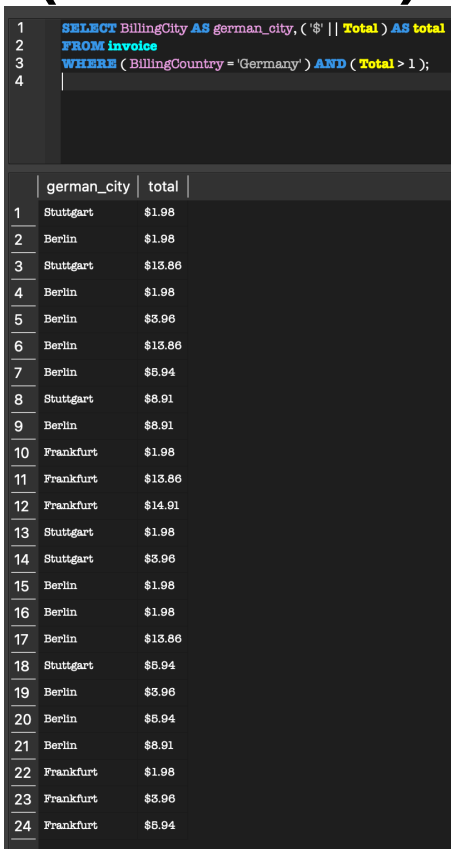
Example: Concatenate fields

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

Complex Output Query (SQLite)



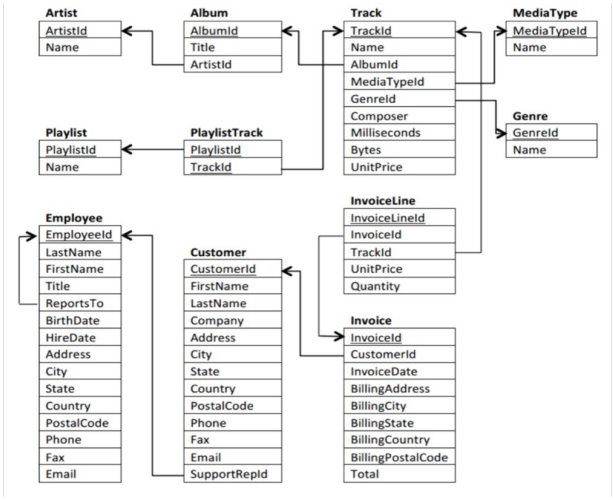
Get all German invoices greater than $1, output the city using the column header “german_city” and “total” prepending $ to the total

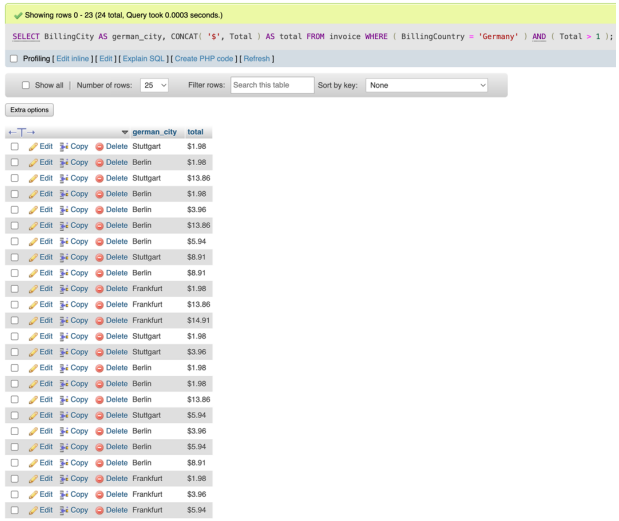


```
SELECT BillingCity AS german_city, ( '$' || Total ) AS total
FROM invoice
WHERE ( BillingCountry = 'Germany' ) AND ( Total > 1 );
```

	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

Complex Output Query (MariaDB)





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

$$G\_INV \leftarrow \sigma_{BillingCountry='Germany' \ AND \ Total>1}(invoice)$$

$$DATA \leftarrow \pi_{BillingCity,CONCAT('\$',Total)}(G\_INV)$$

$$RES \leftarrow \rho_{(german\_city,total)}(DATA)$$

CONCAT is totally non-standard for relational algebra

SQL: 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.

Relational Algebra Note

- Since the relational model considers relations to be sets (whereas SQL=bags), there is no concept of order

- Some extensions to relational algebra consider that the τ (tau) operator converts the input relation to a bag and outputs an ordered list of tuples
  - General form: $\tau_{\text{<attribute list>}}$(Relation)

SQL: Attribute Order List

- Comma separated list

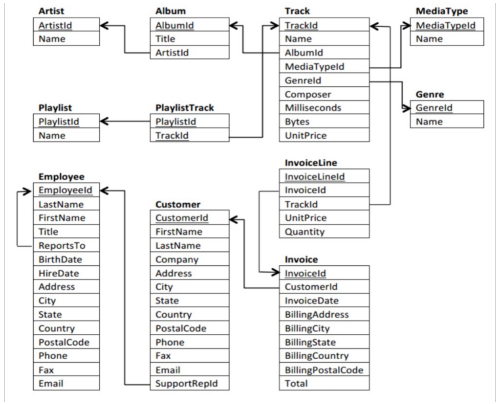
- Format: <attribute name> [Order]
  - Order can be **`ASC`** or **`DESC`**
  - Default is **`ASC`**

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;
```

$$\tau_{LastName,FirstName,BirthDate\ DESC}(\sigma_{true}(employee))$$

Ordering Query




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

	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
1	299	26	2012-08-06 00:00:00	2211 W Berry Street	Fort Worth	TX	USA	76110	23.86
2	201	28	2011-08-29 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	124	20	2010-06-22 00:00:00	541 Del Medio Avenue	Mountain View	CA	USA	94040-111	13.86
7	145	16	2010-09-23 00:00:00	1600 Amphitheatre Parkway	Mountain View	CA	USA	94043-1351	13.86
8	320	22	2012-11-06 00:00:00	120 S Orange Ave	Orlando	FL	USA	32801	13.86
9	6	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;
```

$$\tau_{Total\ DESC, BillingState, BillingCity}(\sigma_{(BillingCountry='USA')\wedge(Total\geq 10)}(invoice))$$

SQL: Set vs. Bag/Multiset

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

- Use **`DISTINCT`** to remove duplicates
- For relational algebra, delta: δ(Relation)

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

Example

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

$$\pi_{BillingState}(\sigma_{BillingCountry='USA'}(\tau_{BillingState}(invoice)))$$

vs.

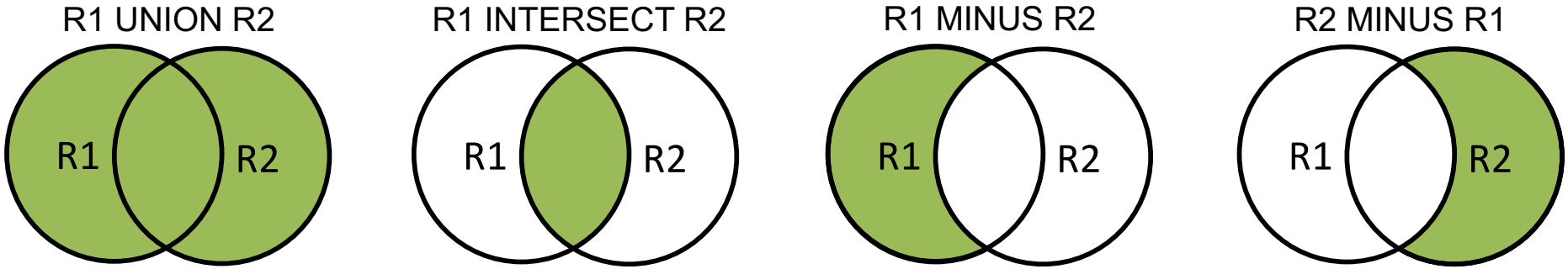
```
SELECT DISTINCT BillingState
FROM invoice
WHERE BillingCountry='USA'
ORDER BY BillingState;
```

$$\delta(\pi_{BillingState}(\sigma_{BillingCountry='USA'}(\tau_{BillingState}(invoice))))$$

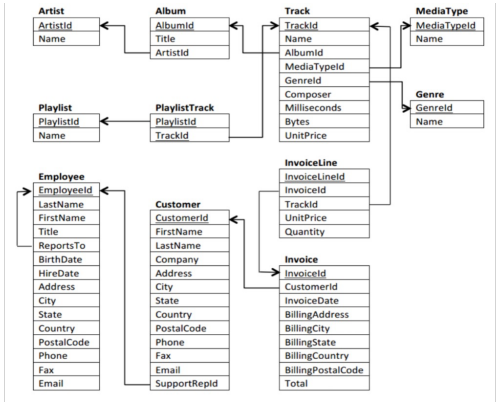
Set Operations

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)



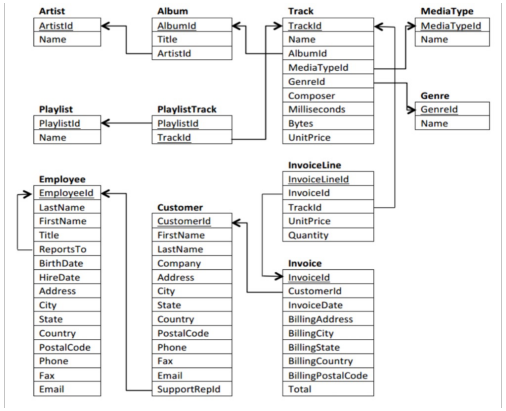
	city
1	Montréal
2	Edmonton
3	Vancouver
4	Toronto
5	Ottawa
6	Halifax
7	Winnipeg
8	Yellowknife

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

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

$$\rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(customer)))$$

Combining Queries (2)



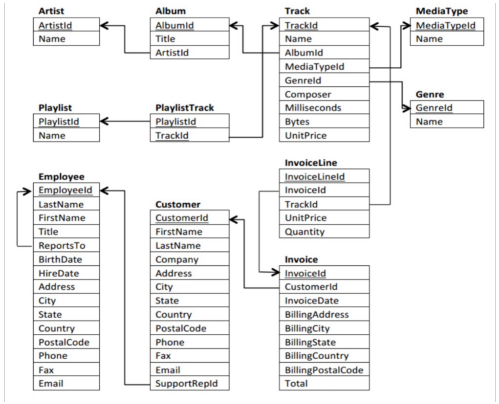
	city
1	Edmonton
2	Calgary
3	Calgary
4	Calgary
5	Calgary
6	Calgary
7	Lethbridge
8	Lethbridge

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

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

$$\rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(employee)))$$

Combining Queries (3)



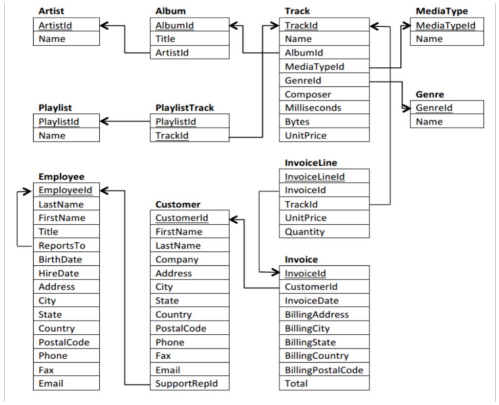
	city
1	Montréal
2	Edmonton
3	Vancouver
4	Toronto
5	Ottawa
6	Halifax
7	Winnipeg
8	Yellowknife
9	Edmonton
10	Calgary
11	Calgary
12	Calgary
13	Calgary
14	Calgary
15	Lethbridge
16	Lethbridge

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';
```

$$R1 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(\tau(customer))))$$
$$R2 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(\tau(employee))))$$
$$RESULT \leftarrow R1 \cup R2$$

Combining Queries (4)



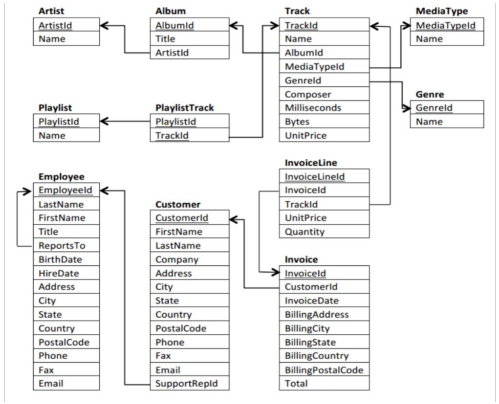
	city
1	Calgary
2	Edmonton
3	Halifax
4	Lethbridge
5	Montréal
6	Ottawa
7	Toronto
8	Vancouver
9	Winnipeg
10	Yellowknife

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

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

$$R1 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(customer)))$$
$$R2 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(employee)))$$
$$RESULT \leftarrow R1 \cup R2$$

Combining Queries (5)



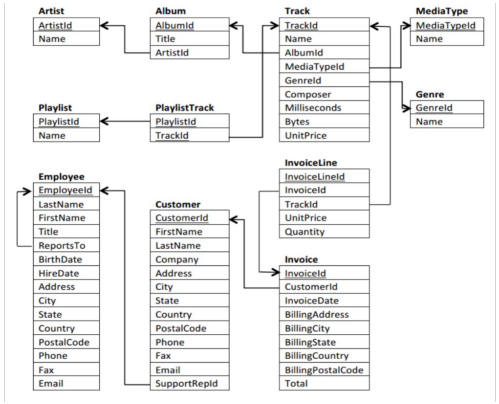
	city
1	Edmonton

Get 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';
```

$$R1 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(customer)))$$
$$R2 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(employee)))$$
$$RESULT \leftarrow R1 \cap R2$$

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)

[recent MariaDB support]

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

$$R1 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(customer)))$$
$$R2 \leftarrow \rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(employee)))$$
$$RESULT \leftarrow R1 - R2$$

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
- Relational algebra: $R_1 \bowtie_{\text{<join condition>}} R_2$

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	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	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	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	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	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	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

```
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

Simple Join – Relational Algebra

STUDENT

Name	SSN	Phone	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	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

$$JOIN \leftarrow STUDENT \bowtie_{STUDENT.SSN=CLASS.SSN} CLASS$$
$$M650 \leftarrow \sigma_{CLASS.Class='MATH650'}(JOIN)$$
$$RES \leftarrow \pi_{STUDENT.GPA}(M650)$$

CLASS

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

SQL: Join 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` $A \bowtie B$	Row must exist in <u>both</u> tables
`LEFT [OUTER] JOIN` $A \bowtie B$	Row must *at least* exist in the table to the left (padded with **`NULL`**)
`RIGHT [OUTER] JOIN` $A \bowtie B$	Row must exist *at least* in the table to the right (padded with **`NULL`**)
`FULL OUTER JOIN` $A \bowtie B$	Row exists in <u>either</u> 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
```

$$Alpha \bowtie_{Alpha.a=Beta.c} Beta$$

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
```

$$Alpha \ltimes_{Alpha.a=Beta.c} Beta$$

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
```

$$Alpha \bowtie\!\!\!\!\!\sqsubset_{Alpha.a=Beta.c} Beta$$

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
```

$$Alpha \bowtie_{Alpha.a=Beta.c} Beta$$

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
```

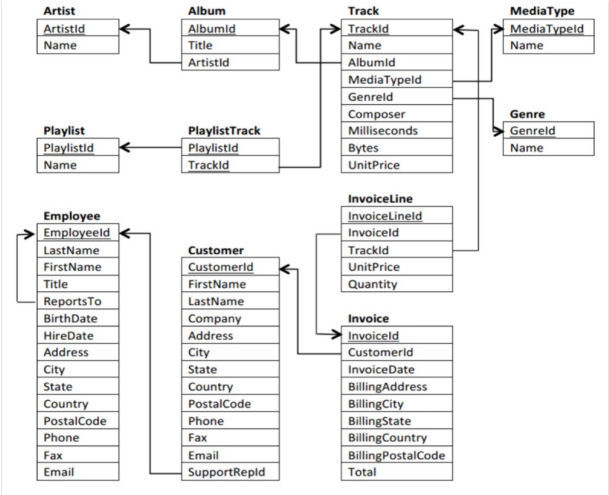
$\sigma_{true}(\rho_{t1}(table1))$

- NATURAL (R_1 * R_2)
  - 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 concatenation of the attributes from *both* tables

Exploring Joins (1)

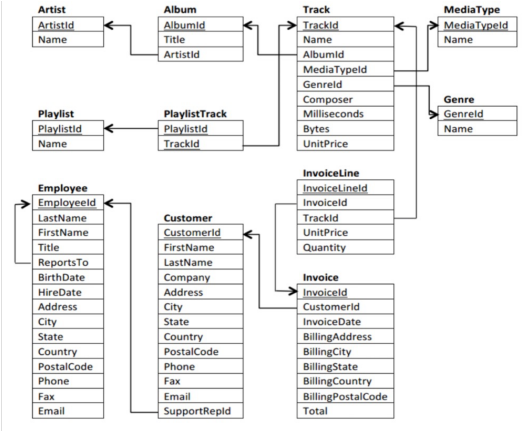


Get the cross product of genres and media types

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

$$\sigma_{true}(genre \bowtie mediatype)$$

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

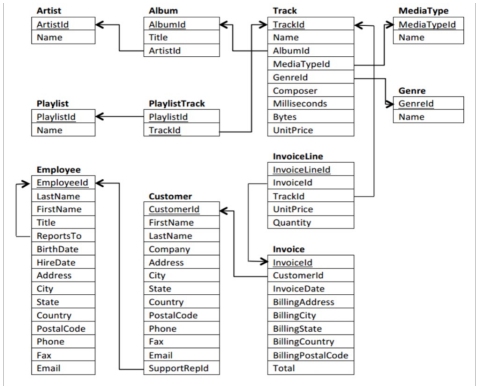
```
SELECT *
FROM (track t INNER JOIN mediatype mt ON t.MediaTypeId=mt.MediaTypeId)
INNER JOIN genre g ON t.GenreId=g.GenreId
WHERE g.Name='Jazz' AND t.Composer LIKE '%Miles Davis%';
```

$$J1 \leftarrow \rho_t(track) \bowtie_{t.MediaTypeId=mt.MediaTypeId} \rho_{mt}(mediatype)$$

$$J2 \leftarrow J1 \bowtie_{t.GenreId=g.GenreId} \rho_g(genre)$$

$$RES \leftarrow \sigma_{g.Name='Jazz' \ AND \ t.Composer \ LIKE \ '\%MilesDavis\%'}(J2)$$

Advanced Joins (1)



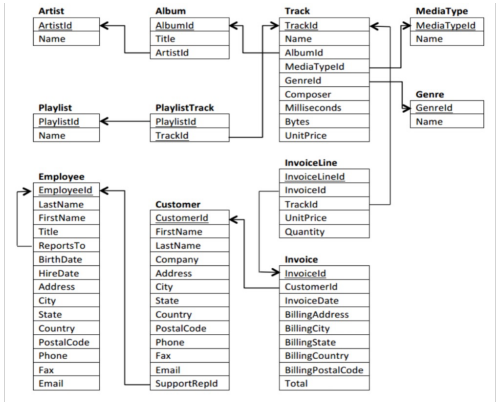
	ArtistId	Name
1	169	Black Eyed Peas
2	11	Black Label Society
3	12	Black Sabbath

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

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

$$\tau_{Name}(\sigma_{Name\ LIKE\ 'Black\%'}(artist))$$

Advanced 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

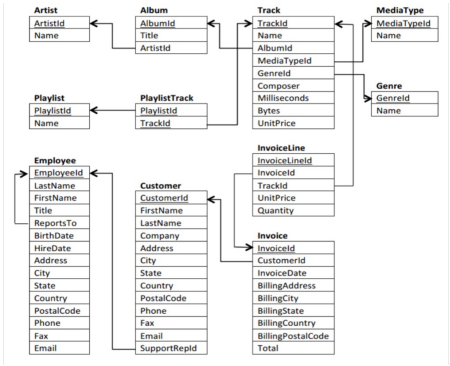
```
SELECT *
FROM artist art INNER JOIN album alb ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name ASC, alb.Title ASC;
```

$$J \leftarrow \rho_{art}(artist) \bowtie_{art.ArtistId=alb.ArtistId} \rho_{alb}(album)$$

$$S \leftarrow \sigma_{Name\ LIKE\ 'Black\%'}(J)$$

$$RES \leftarrow \tau_{art.Name,alb.Title}(S)$$

Advanced 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

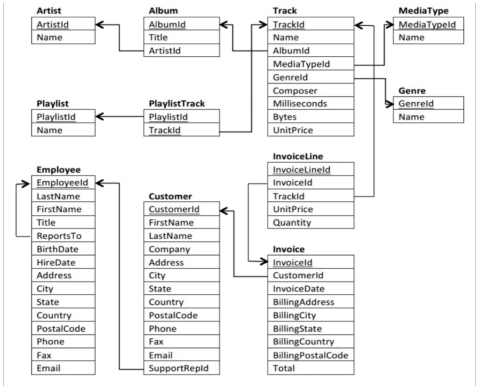
```
SELECT *
FROM artist art LEFT OUTER JOIN album alb ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name, alb.Title;
```

$$J \leftarrow \rho_{art}(artist) \bowtie_{art.ArtistId=alb.ArtistId} \rho_{alb}(album)$$

$$S \leftarrow \sigma_{Name\ LIKE\ 'Black\%'}(J)$$

$$RES \leftarrow \tau_{art.Name,alb.Title}(S)$$

Advanced Joins (4)



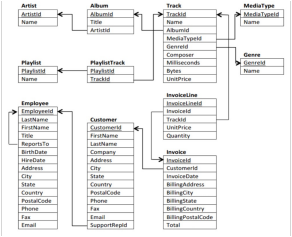
	ArtistId	Name	AlbumId	Title
1	169	Black Eyed Peas	*NULL*	*NULL*
2	11	Black Label Society	14	Alcohol Fueled Brewtality Live! [Disc 1]
3	11	Black Label Society	15	Alcohol Fueled Brewtality Live! [Disc 2]
4	12	Black Sabbath	16	Black Sabbath
5	12	Black Sabbath	17	Black Sabbath Vol. 4 (Remaster)

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;
```

$$J \leftarrow \rho_{art}(artist) \bowtie_{art.ArtistId=alb.ArtistId} \rho_{alb}(album)$$
$$S \leftarrow \sigma_{Name\ LIKE\ 'Black\%'}(J)$$
$$P \leftarrow \pi_{art.ArtistId,art.Name,alb.AlbumId,alb.Title}(S)$$
$$RES \leftarrow \tau_{art.Name,alb.Title}(P)$$

Advanced Joins (5)



	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';
```

$$TA \leftarrow \rho_t(track) \bowtie_{t.AlbumId=a.AlbumId} \rho_a(album)$$
$$M \leftarrow TA \bowtie_{t.MediaTypeId=m.MediaTypeId} \rho_m(mediatype)$$
$$G \leftarrow M \bowtie_{t.GenreId=g.GenreId} \rho_g(genre)$$
$$S \leftarrow \sigma_{t.Name='Give\ Me\ Novacaine'}(G)$$
$$P \leftarrow \pi_{t.TrackId,t.Name,t.Composer,t.UnitPrice,a.Title,m.Name,g.Name}(S)$$
$$RES \leftarrow \rho_{(TrackId,tName,Composer,UnitPrice,Title,mName,gName)}(P)$$