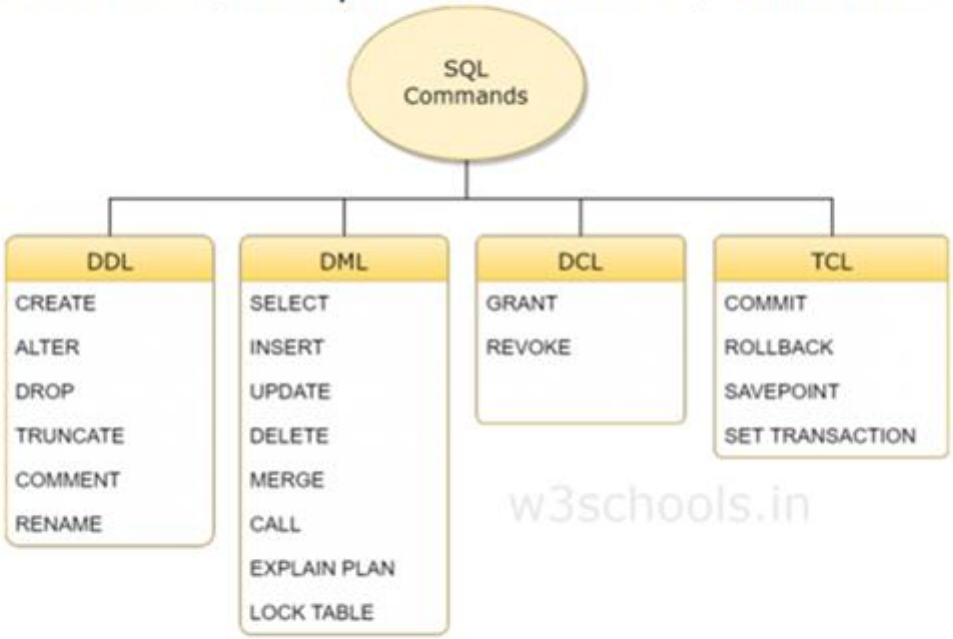
El lenguaje SQL

Diversos tipos de sentencias

SQL = Declaración + Manipulación + Control + Control transaccional



Hoy vamos a trabajar con DML.

Dataset para la clase de hoy

Dataset

▨ Utilizaremos the Chinook database

About the Chinook database

The Chinook data model represents a digital media store, including tables for artists, albums, media tracks, invoices, and customers.

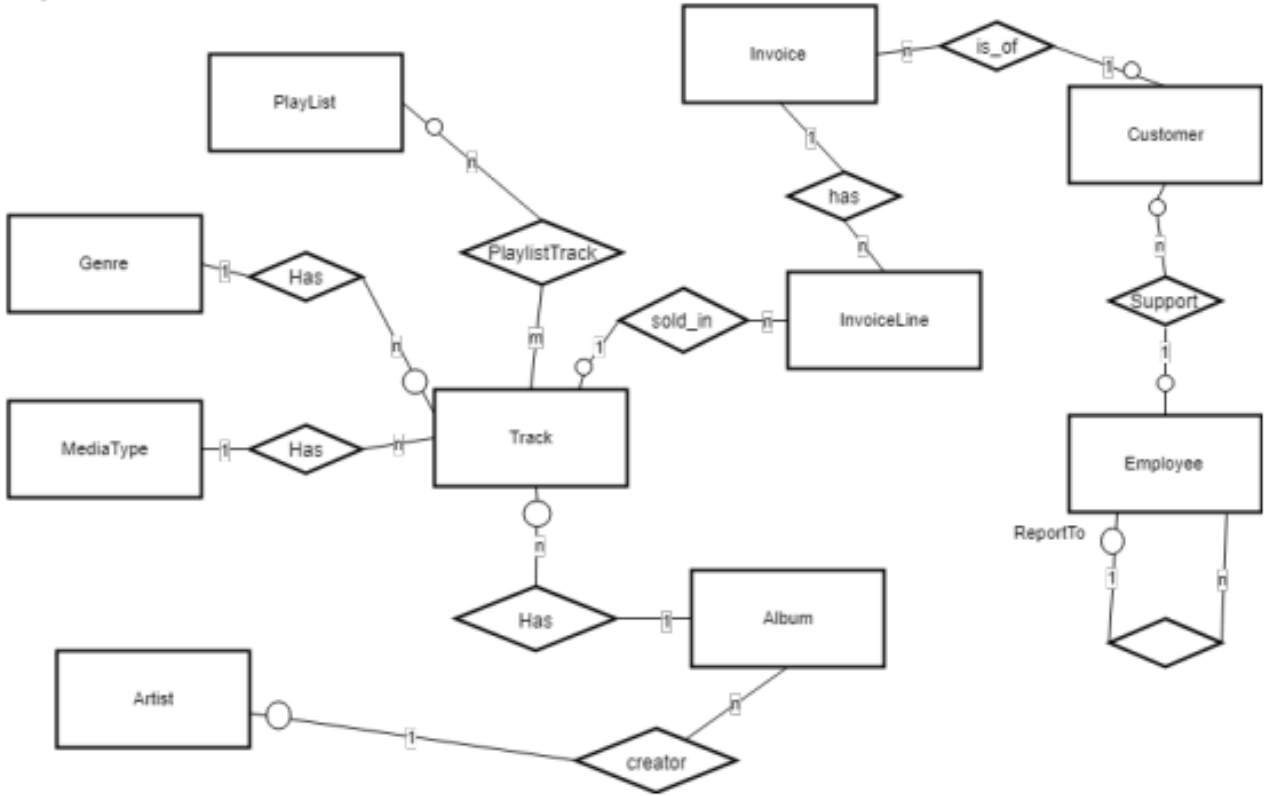
- Media-related data was created using real data from an Apple iTunes library.
- Customer and employee information was created using fictitious names and addresses that can be located on Google maps, and other well formatted data (phone, fax, email, etc.)
- Sales information was auto generated using random data for a four year period.

The Chinook sample database includes:

- 11 tables
- A variety of indexes, primary and foreign key constraints
- Over 15,000 rows of data

Base de Datos 'Chinook'

DER simplificado de Chinook:



Esquema básico de consulta

Esquema básico de consulta en SQL

- El esquema básico de una consulta en SQL es:

```
SELECT A1, A2, …, An
    FROM T1, T2, …, Tn
    [ WHERE condicion ];
```

Se puede formar condiciones más complejas combinando simples con AND, OR y NOT.

- En donde (A_1, A_2, …, A_n) son nombres de columnas, (T_1, T_2, …, T_n) son nombres de tablas, y la condición es una expresión que devuelve un valor booleano.

Esquema básico de consulta en SQL

```
SELECT A1, A2, …, An
    FROM T1, T2, …, Tn
    [ WHERE condicion ];
```

- Esta consulta en SQL es análoga a la siguiente consulta en álgebra relacional:

$$\pi_{A1,\ A2,\ \ldots,\ An}\ (\ \sigma_{(cond)}(T_1\ x\ T_2\ x\ \ldots\ x\ T_n)\ )$$

Con la diferencia de que en SQL el SELECT no elimina por defecto las filas repetidas del resultado

Si ponemos más de una tabla en el FROM, se hace el producto cartesiano entre ellas

Esquema básico de consulta: ambigüedades en nombres

```
SELECT Alumnos.nombre, c.nombre_materia
  FROM Alumnos, Cursa c
 WHERE Alumnos.legajo = c.legajo;
```

alias de la tabla

- Se pueden usar **alias** para los nombres de tabla
- Para indicar sin ambigüedades a que tabla nos referimos, podemos escribir el nombre de la tabla o su alias antes del atributo.

Esquema básico de consulta en SQL: ejercicio

1. Devolver los títulos de los álbumes.
¿Cuántos hay?

Esquema básico de consulta en SQL: ejercicio

Devolver los títulos de los álbumes.

```
SELECT
    Title
FROM album
;
```

(Hay 347 filas)

Agregación en el SELECT

- ¿Es posible directamente obtener la cantidad de filas del resultado? Sí, usando funciones de agregación en el select:

SELECT **agg**(A_1), **agg**(A_2), …, **agg**(A_n)

FROM T_1, T_2, …, T_n

[WHERE condicion **]**;

- Las funciones de agregación pueden ser: COUNT(A_i), SUM(A_i), AVG(A_i), MIN(A_i), MAX(A_i), …
- Al usar una función de agregación en el SELECT de una consulta básica, el resultado sólo tendrá una fila.
- Por eso, si usamos una función de agregación en una de las columnas, debemos usar funciones de agregación en todas.

Agregación en el SELECT: ejercicio

2. Devolver la cantidad de facturas que tienen un total mayor o igual a 20.

Agregación en el SELECT: ejercicio

Devolver la cantidad de facturas que tienen un total mayor o igual a 20.

```
SELECT COUNT(invoice_id)
FROM invoice
WHERE total >= 20;
```

Agregación en el SELECT: ejercicio

3. Devolver la cantidad de facturas que existen en la base de datos y su valor total promedio.

Agregación en el SELECT: ejercicio

Devolver la cantidad de facturas que existen en la base de datos y su valor total promedio.

```
SELECT COUNT(invoice_id), AVG(total)
FROM invoice
;
```

Consultas básicas con más de una tabla en el FROM

- Recordemos que cuando escribimos más de una tabla en la cláusula FROM, el resultado es equivalente a realizar el producto cartesiano entre ellas.
- Por lo tanto, si usamos la condición correcta en el WHERE, podemos hacer que SQL calcule una **junta**.

4. Encontrar todos los tracks de la playlist cuyo id es 5, mostrando el nombre del track, su compositor y su precio.

Consultas básicas con más de una tabla en el FROM: ejercicio

Encontrar todos los tracks de la playlist cuyo id es 5, mostrando el nombre del track, su compositor y su precio.

```
SELECT track.name, track.composer,track.unit_price
    FROM playlist_track, track
    WHERE playlist_track.track_id = track.track_id
    AND playlist_id = 5
```

Junta: INNER JOIN

- La consulta anterior también puede resolverse usando el operador INNER JOIN en el FROM, y poniendo la condición de junta ahí mismo:

Encontrar todos los tracks de la playlist cuyo id es 5, mostrando el nombre del track, su compositor y su precio.

```
SELECT track.name, track.composer,track.unit_price
FROM playlist_track INNER JOIN track
ON (playlist_track.track_id = track.track_id)
WHERE playlist_id = 5
```

Ordenamiento y paginación

- Para ordenar el resultado y paginar los datos, extendemos el esquema básico de consulta con las clausulas ORDER BY y FETCH FIRST:

```
SELECT A1, A2, …, An
       FROM T1, T2, …, Tn
       [ WHERE condicion ]
[ ORDER BY B1, B2, …, Bm ];
[ OFFSET k ROWS FETCH FIRST n ROWS ONLY ];
```

Ordenamos el resultado primero por B_1; si hay empate por B_1 desempatamos por B_2, etc.

El orden por cada atributo es ascendente, salvo que pongamos la palabra DESC después del atributo

Una vez ordenado, descartamos las primeras k, y luego mostramos las siguientes n

Ordenamiento y paginación: ejercicio

Devolver las 5 facturas con mayor importe, mostrando el identificador de la factura, su importe, y el nombre del usuario a quien se emitió cada factura. Ordenar los resultados en forma descendente por importe.

Ordenamiento y paginación: ejercicio

Devolver las 5 facturas con mayor importe, mostrando el identificador de la factura, su importe, y el nombre del usuario a quien se emitió cada factura. Ordenar los resultados en forma descendente por importe.

```
SELECT invoice_id, total, first_Name, last_name
FROM invoice, customer
WHERE invoice.customer_id = customer .customer_id
ORDER BY total DESC
OFFSET 0 ROWS FETCH FIRST 5 ROWS ONLY;
```

Ejercicio

6. Encontrar el nombre y la duración de los 5 tracks con mayor duración de los que se conozca el compositor y que están en la playlist ‘Classical’

Ejercicio integrador - repaso

7. Obtener los nombres de los tracks, sus compositores, el álbum y el artista al que pertenecen de todos aquellos tracks que están en facturas con fecha mayor a cuatro años.

Agregación en SQL

Agregación en SQL

- Analicemos la siguiente tabla **Campeón(nombre_tenista, nombre_torneo, premio)** que indica los torneos ganados por distintos tenistas:

CAMPEÓN

nombre_tenista	nombre_torneo	premio
Novak Djokovic	Abierto de Australia	8.000.000
Rafael Nadal	Abierto de Barcelona	1.500.000
Novak Djokovic	Abierto de Madrid	2.500.000
Novak Djokovic	Roland Garros	5.000.000
Andy Murray	Master de Shanghai	4.000.000
Juan Martín del Potro	Abierto de Estocolmo	300.000
Andy Murray	Master BNP Paribas	2.000.000
Andy Murray	ATP Tour Finals de Londres	4.000.000

Agregación en SQL

- Nos gustaría resumir en una tabla la cantidad de títulos ganados por cada tenista y su premio total.
- Para ello necesitamos **agrupar** los datos por cada tenista:

CAMPEÓN

nombre_tenista	nombre_torneo	premio
Novak Djokovic	Abierto de Australia	8.000.000
Novak Djokovic	Abierto de Madrid	2.500.000
Novak Djokovic	Roland Garros	5.000.000
Rafael Nadal	Abierto de Barcelona	1.500.000
Juan Martín del Potro	Abierto de Estocolmo	300.000
Andy Murray	Master de Shanghai	4.000.000
Andy Murray	Master BNP Paribas	2.000.000
Andy Murray	ATP Tour Finals de Londres	4.000.000

Agregación en SQL

- Una vez agrupados, queremos reemplazar cada grupo de datos (que corresponde a un tenista) por una única fila, que muestre el nombre del tenista, pero resumiendo a los torneos ganados indicando su cantidad, y a los premios ganados indicando su suma.

- El resultado sería:

nombre_tenista	cantidad_torneos	premio_total
Novak Djokovic	3	15.500.000
Rafael Nadal	1	1.500.000
Juan Martín del Potro	1	300.000
Andy Murray	3	10.000.000

Agregación en SQL

- La **agregación** en SQL colapsa todas las filas que coinciden en el valor de uno o más atributos (en este caso, en la columna “nombre_tenista”) en una única fila que las representa a todas ellas.

- Decimos que “agrupamos las filas de Campeones por el nombre del tenista, resumiendo los torneos con su cantidad, y los premios con su suma”.

nombre_tenista	cantidad_torneos	premio_total
Novak Djokovic	3	15.500.000
Rafael Nadal	1	1.500.000
Juan Martín del Potro	1	300.000
Andy Murray	3	10.000.000

Agregación en SQL

nombre_tenista	cantidad_torneos	premio_total
Novak Djokovic	3	15.500.000
Rafael Nadal	1	1.500.000
Juan Martín del Potro	1	300.000
Andy Murray	3	10.000.000

- En SQL, esto se logra con la cláusula **GROUP BY**:

Funciones de agregación sobre alguno/s de los atributos restantes

```
SELECT nombre_tenista, COUNT(nombre_torneo), SUM(premio)
FROM Campeon
GROUP BY nombre_tenista
```

Atributo de agrupamiento

Esquema de consulta con agregación en SQL

- El esquema de consulta con agregación en SQL es:

Algunos de los atributos por los que agrupamos a las filas (A_1, A_2, …, A_n) podemos mostrarlos en la "fila resumen" si lo deseamos

```
SELECT Ak1, Ak2, …, Akp, agg(B1), agg(B2), …, agg(Bm)
    FROM T1, T2, …, Tn
    [ WHERE condicion ]
GROUP BY A1, A2, …, An
[ HAVING condicion ];
```

En cambio, los atributos que no están en el GROUP BY sólo pueden aparecer en el resultado si pasan por una función de agregación

`COUNT(), SUM(), AVG(), MIN(), MAX()`

Esquema de consulta con agregación en SQL: ejercicio

1. Encontrar para cada compositor la cantidad de tracks compuestos.

Agregación en el SELECT: ejercicio

Encontrar para cada compositor la cantidad de tracks compuestos.

```
SELECT composer, COUNT(trackid)
FROM Track
GROUP BY composer;
```

Esquema de consulta con agregación en SQL

- La cláusula **HAVING** permite filtrar alguno de los grupos, en función del valor de los atributos agregados.

```
SELECT Ak1, Ak2, …, Akp, agg(B1), agg(B2), …, agg(Bm)
       FROM T1, T2, …, Tn
       [ WHERE condicion ]
GROUP BY A1, A2, …, An
[ HAVING condicion ];
```

Aquí podemos poner condiciones sobre los valores agregados de los atributos dentro del grupo.

Esquema de consulta con agregación en SQL: ejercicio

2.Encontrar para cada compositor la cantidad de tracks compuestos **mostrando únicamente aquellos compositores con al menos 20 tracks compuestos.**

Esquema de consulta con agregación en SQL: ejercicio

Encontrar para cada compositor la cantidad de tracks compuestos **mostrando únicamente aquellos compositores con al menos 20 tracks compuestos.**

```
SELECT composer, COUNT(trackid)
FROM Track
GROUP BY composer
HAVING COUNT(trackid)>=20;
```

Consultas de agregación con más de una tabla en el FROM

- Recordemos que poniendo más de una tabla en el FROM, obtenemos el producto cartesiano entre esas tablas.
- Esto podemos transformarlo en una junta usando la condición correcta en el WHERE, o bien usando el INNER JOIN en el FROM.
- Si además de esto utilizamos un GROUP BY, podremos agrupar y resumir el resultado de esa junta.

Esquema de consulta con agregación en SQL: ejercicio

3.Encontrar quienes son los clientes que tienen facturas con importes promedio más alto. Mostrar a los 10 clientes con mayor importe promedio, mostrando el apellido del cliente, la cantidad de facturas a su nombre y el promedio de esas facturas.

Esquema de consulta con agregación en SQL: ejercicio

Encontrar quienes son los clientes que tienen facturas con importes promedio más alto. Mostrar a los 10 clientes con mayor importe promedio, mostrando el apellido del cliente, la cantidad de facturas a su nombre y el promedio de esas facturas.

```
SELECT last_name, COUNT(invoice_id), AVG(total)
    FROM customer c INNER JOIN invoice i
    ON c.customer_id = i.customer_id
GROUP BY last_name
ORDER BY AVG(total) DESC
OFFSET 0 ROWS FETCH FIRST 10 ROWS ONLY;
```

Esquema de consulta con agregación en SQL: ejercicio

4.¿Y si quisiéramos saber en cuántas facturas aparece cada track, pero sólo para aquellos que aparecen en al menos dos facturas?

Esquema de consulta con agregación en SQL: ejercicio

¿Y si quisiéramos saber en cuántas facturas aparece cada track, pero sólo para aquellos que aparecen en al menos dos facturas?

```
SELECT t.name, COUNT(il.invoice_line_id) cantidad
    FROM Track t INNER JOIN invoice_line il
    ON t.track_id = il.track_id
GROUP BY t.name
HAVING COUNT(il.track_id) >= 2
ORDER BY cantidad DESC
```

Ejercicio

5. Encontrar al máximo fan de Iron Maiden, al que definiremos como aquel que tenga más líneas de factura vinculadas con tracks de la banda. Mostrar el apellido de dicho usuario y la cantidad de líneas de factura vinculadas con tracks de la banda.

Ejercicio

5. Encontrar al máximo fan de Iron Maiden, al que definiremos como aquel que tenga más líneas de factura vinculadas con tracks de la banda. Mostrar el apellido de dicho usuario y la cantidad de líneas de factura vinculadas con tracks de la banda.

```
SELECT c.last_name, count(il.invoice_line_id) cantidad
from invoice_line il   INNER JOIN invoice i on il.invoice_id = i.invoice_id
                       INNER JOIN customer c on i.customer_id = c.customer_id
                       INNER JOIN track t on il.track_id = t.track_id
                       INNER JOIN album a on a.album_id = t.album_id
                       INNER JOIN artist ar on ar.artist_id = a.artist_id
where ar.name = 'Iron Maiden’
group by c.last_name
order by cantidad desc
OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY;
```

Consultas anidadas

- El resultado de una consulta SQL siempre es una tabla…
- Pero cuando esa tabla tiene solo una fila y una columna, entonces puede ser pensada como un valor…
- y entonces puede ser incluida en cualquier parte en la que SQL espera un valor.

SELECT A_{k1}, A_{k2}, …, A_{kp}, agg(B_1), agg(B_2), …, agg(B_m)
FROM T_1, T_2, …, T_n
[**WHERE** (**SELECT** …
FROM …
WHERE) = **valor** …];

Subconsulta anidada

¡Cuidado! Si el resultado tiene más de una fila, devolverá error

Consultas anidadas: ejercicio

6. Encuentre los nombres de los tracks pertenecientes a facturas del cliente que tiene la factura de mayor importe.

Sugerencia:

1. Escriba una consulta simple (sin subconsultas anidadas) que encuentre el Id del usuario asociado a la factura de mayor importe.
2. Escriba ahora una consulta que utilice el Id obtenido en el punto 1 para encontrar todos los tracks vinculados a facturas de ese usuario.

Consultas anidadas: ejercicio

Encuentre los nombres de los tracks pertenecientes a facturas del cliente que tiene la factura de mayor importe.

```
select t.name
from invoice i   INNER JOIN invoice_line il on i.invoice_id = il.invoice_id
        INNER JOIN track t on il.track_id = t.track_id
where customer_id =
        (Select  c.customer_id
        from customer c INNER JOIN invoice i on c.customer_id = i.customer_id
                order by total desc
                OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY)
```

Cláusula WITH

- La cláusula WITH nos permite definir una tabla temporal para ser utilizada en una consulta.
- Esta tabla nunca se almacenará. Sólo existirá durante el cálculo de la consulta.

Sintaxis:

```
WITH nombre_tabla_temp AS (SELECT …)
SELECT A1, A2, …, An
        FROM T1, T2, …, Tn
        [ WHERE condición ];
```

Subconsulta auxiliar

Consulta principal

Cláusula WITH: ejercicio

7. Utilizando una cláusula WITH, listar nombres de playlists que tienen tracks que aparecen en las facturas de menor importe.

Cláusula WITH: ejercicio

Utilizando una cláusula WITH, listar nombres de playlists que tienen tracks que aparecen en las facturas de menor importe.

```
WITH facturasConMenorImporte AS
     (Select invoice_id
             FROM invoice
     Where total = (Select Min(total) from invoice))

Select DISTINCT p.name
     From playlist as p
     INNER JOIN playlist_track as pt ON pt. playlist_id = p.playlist_id
     INNER JOIN track as t ON pt.track_id = t.track_id
     INNER JOIN invoice_line as il ON il.track_id = t.track_id
     INNER JOIN facturasConMenorImporte as f ON f.invoice_id = il.invoice_id
```

Consultas anidadas, menú completo

- Existen varias formas de anidar una consulta en el WHERE:
  - WHERE valor = (SELECT…) → solo funciona si la subconsulta devuelve una columna con un valor
  - WHERE valor **IN|NOT IN** (SELECT…) → funciona si la subconsulta devuelve sólo 1 columna o si “valor” es una tupla y la consulta devuelve un tupla similar.
  - WHERE valor **>= ALL|SOME** (SELECT…) → funciona si la subconsulta devuelve sólo 1 columna
  - WHERE **EXISTS|NOT EXISTS** (SELECT…) → funciona siempre
- Cuando la consulta anidada utiliza columnas de una fila de alguna tabla de la consulta principal, se dice que las consultas de adentro y la de afuera están **correlacionadas.**
  - Esto implica que la consulta de adentro deberá ejecutarse una vez por cada fila de la consulta de afuera.

Consultas anidadas, menú completo: ejercicio

1. Usando consultas anidadas y la cláusula NOT EXIST, seleccione nombre del track y composer de los tracks que nunca fueron facturados.

Consultas anidadas, menú completo: ejercicio

Usando consultas anidadas y la cláusula NOT EXIST, seleccione nombre y composer de los tracks que nunca fueron facturados.

```
SELECT  track.name, track.composer
     FROM track
     WHERE NOT EXISTS (
               SELECT *
               FROM invoice_line
               WHERE invoice_line.track_id = track.track_id )
```

Operaciones de conjuntos en SQL

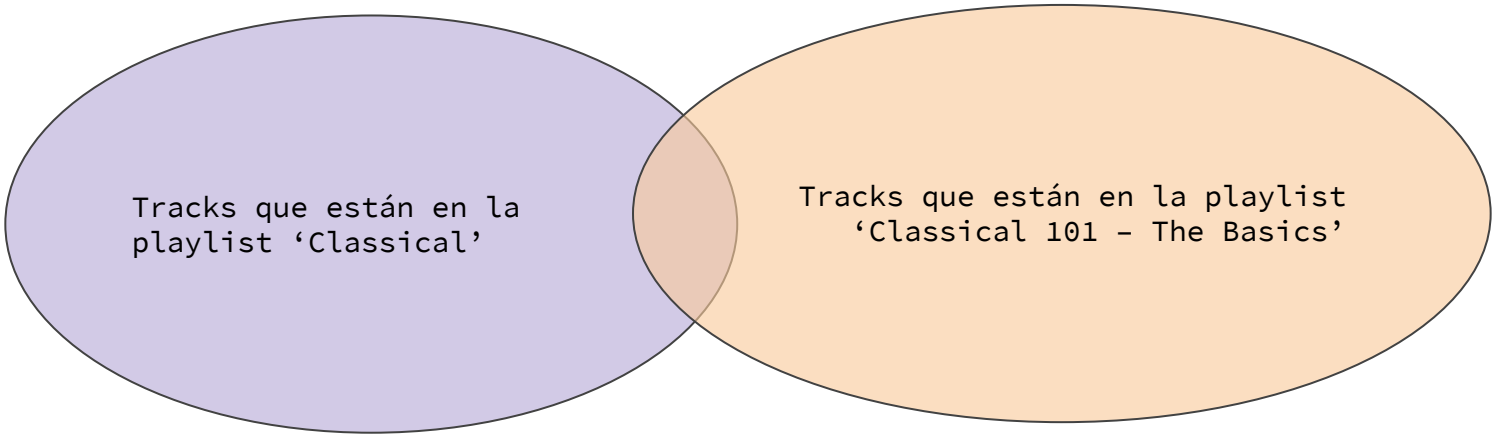
Ejemplo

Supongamos que queremos encontrar los nombres de los tracks que están tanto en la playlist de nombre ‘Classical’ como en la de nombre ‘Classical 101 - The Basics’. La siguiente podría ser una forma de resolverlo:

```
Select t.name, p1.name, p2.name
from track t, playlist p1, playlist_track pt1, playlist_track pt2,playlist p2
        where pt1.track_id = pt2.track_id
                and t.track_id = pt1.track_id
                and pt1.playlist_id = p1.playlist_id
                and pt2.playlist_id = p2.playlist_id
                and p1.name = 'Classical’
                and p2.name = 'Classical 101 - The Basics'
```

Ejemplo

- Sin embargo, también podríamos resolverlo de la siguiente forma:



- Es decir, con una **intersección** de conjuntos.

Operaciones de conjuntos

- SQL permite calcular la unión, intersección o diferencia del resultado de dos consultas SQL independientes.
- El resultado es análogo a lo que ocurría en el álgebra relacional con U, ∩, -.
- Al igual que en álgebra, se requiere **compatibilidad de tipos** entre las relaciones.
- La sintaxis es:

(SELECT … consulta SQL 1 **)**

UNION | INTERSECT | EXCEPT

(SELECT … consulta SQL 2 **)**

Operaciones de conjuntos: ejercicio

2.Encontrar los nombres de los tracks que están tanto en la playlist de nombre ‘Classical’ como en la de nombre ‘Classical 101 - The Basics’ utilizando operaciones de conjuntos en SQL.

Operaciones de conjuntos: ejercicio

Encontrar los nombres de los tracks que están tanto en la playlist de nombre ‘Classical’ como en la de nombre ‘Classical 101 - The Basics’ utilizando operaciones de conjuntos en SQL.

```
( SELECT t.trackid, t.name FROM Track t, Playlist p, Playlisttrack pt
     WHERE p.playlistid = pt.playlistid
     AND pt.trackid = t.trackid AND p.name = 'Classical' )
INTERSECT
( SELECT t.trackid, t.name FROM Track t, Playlist p, Playlisttrack pt
     WHERE p.playlistid = pt.playlistid
     AND pt.trackid = t.trackid AND p.name = 'Classical 101 - The Basics')
```

Operaciones de conjuntos: ejercicio 2

3.Encontrar los nombres de los tracks que están o bien en la playlist de nombre ‘Music Videos’ o bien en la de nombre ‘On-The-Go 1’ utilizando operaciones de conjuntos en SQL.

Operaciones de conjuntos: ejercicio 2

Encontrar los nombres de los tracks que están en o bien en la playlist de nombre ‘Music Videos’ o bien en la de nombre ‘On-The-Go 1’ utilizando operaciones de conjuntos en SQL.

```
( SELECT t.trackid, t.name
      FROM Track t, Playlist p, Playlisttrack pt
              WHERE p.playlistid = pt.playlistid
              AND pt.trackid = t.trackid AND p.name = 'Music Videos' )
UNION
( SELECT t.trackid, t.name
      FROM Track t, Playlist p, Playlisttrack pt
              WHERE p.playlistid = pt.playlistid
              AND pt.trackid = t.trackid AND p.name = 'On-The-Go 1' )
```

Junta externa (OUTER JOIN)

Ejemplo

- Supongamos ahora que queremos mostrar para cada artista cuántos álbumes tiene de su autoría en nuestra base de datos, pero asegurándonos de no perder a aquellos de los cuáles no tenemos registrado un álbum. ¿Qué ocurriría con la siguiente solución?

```
SELECT a.name, COUNT(*)
        FROM Artist a INNER JOIN Album al
                ON (a.artist_id = al.artist_id)
                GROUP BY a.artist_id, a.name;
```

Ejemplo

- Para asegurarnos de que todos los artistas ‘a’ que aparecen del lado izquierdo de la consulta sobrevivan en el resultado, se puede usar un **LEFT JOIN**:

```
SELECT a.name, COUNT(album_id)
        FROM Artist a LEFT JOIN Album al
                ON (a.artist_id = al.artist_id)
                GROUP BY a.artist_id, a.name;
```

- De esta manera, los artistas que no se vinculan con ningún álbum quedan presentes en el resultado de la junta.

Outer join

- El ***outer join*** o **junta externa** permite juntar dos tablas asegurándonos de que los valores en las filas de ellas siempre aparezcan en el resultado, aún cuando no hayan logrado combinarse con ninguna fila de la otra tabla.
- Viene en 3 variantes:
  - **LEFT OUTER JOIN:** Asegura que las filas de la tabla izquierda siempre estén en el resultado.
  - **RIGHT OUTER JOIN:** Asegura que las filas de la tabla derecha siempre estén en el resultado.
  - **FULL OUTER JOIN:** Asegura que las filas de ambas tablas siempre estén en el resultado.
- Para lograr esto, el OUTER JOIN completa en el resultado las filas que no logró conectar con valores nulos (NULL).

Outer join: sintaxis

```
SELECT A1, A2, …, An
    FROM T1 LEFT|RIGHT|FULL [OUTER] JOIN T2 ON (cond_junta)
    [ WHERE condicion ];
```

Outer join: ejercicio

4.Para cada track de género “Rock And Roll” muestre el nombre del track y la cantidad de invoice_lines en las que aparece, asegurándose de que si un track nunca estuvo en una factura se devuelva cero.

Outer join: ejercicio

Para cada track de género “Rock And Roll” muestre el nombre del track y la cantidad de invoice_lines en las que aparece, asegurándose de que si un track nunca estuvo en una factura se devuelva cero.

¿Por qué no puede ser COUNT(*)?

```
SELECT t.name, COUNT(i.invoice_line_id)
    FROM Track t INNER JOIN genre g on (t.genre_id = g.genre_id)
LEFT JOIN Invoice_line i ON (t.track_id = i.track_id)
    WHERE g.name = 'Rock And Roll'
    GROUP BY t.name
```