

Vistas, indexación y optimización de consultas SQL.

Alex Di Genova

23/06/2022

Contenidos

- Notas y revisión control II
- Vistas
- Indexación
- Optimización de consultas SQL

Notas y revisión control II

Revisión control II

Respuestas

1. **Base de datos Chinook:** Escriba una consulta que  muestre los distintos tipos de empleados que trabajan en Canadá. Hint: Title. (2 puntos)

```
select distinct (Title) from Employee where country == "Canada";
```

2. **Base de datos Chinook:** Explique que tabla produce el siguiente código SQL (3 punto)

```
select i.billingcountry, sum(total) as 'TotalSales'  
from invoice as i  
group by billingcountry  
order by TotalSales desc
```

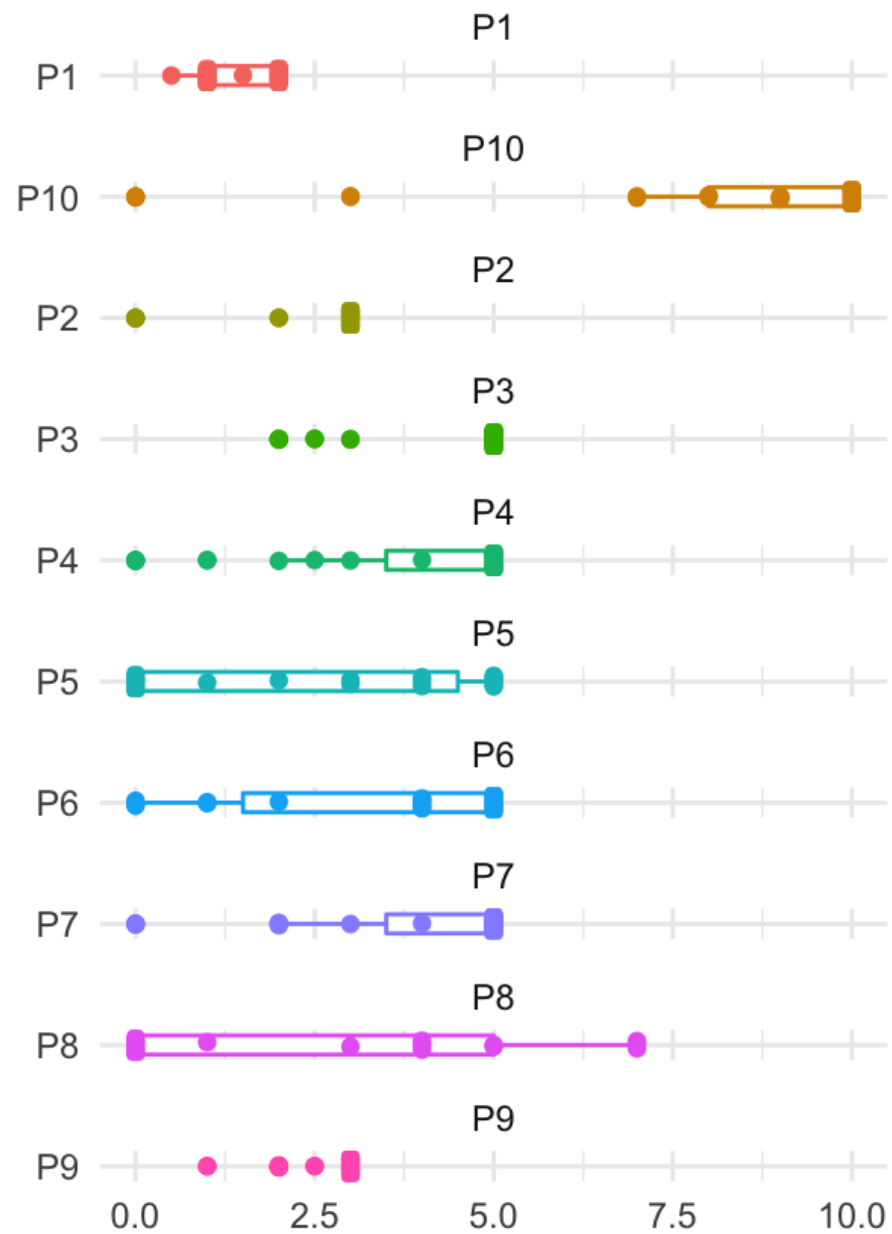
Muestra las ventas totales de canciones por país.

BillingCountry	TotalSales
USA	523.06000000000003
Canada	303.95999999999999
France	195.09999999999994
Brazil	190.09999999999997
Germany	156.48
United Kingdom	112.85999999999999
Czech Republic	90.24000000000001
Portugal	77.23999999999998
India	75.25999999999999
Chile	46.62

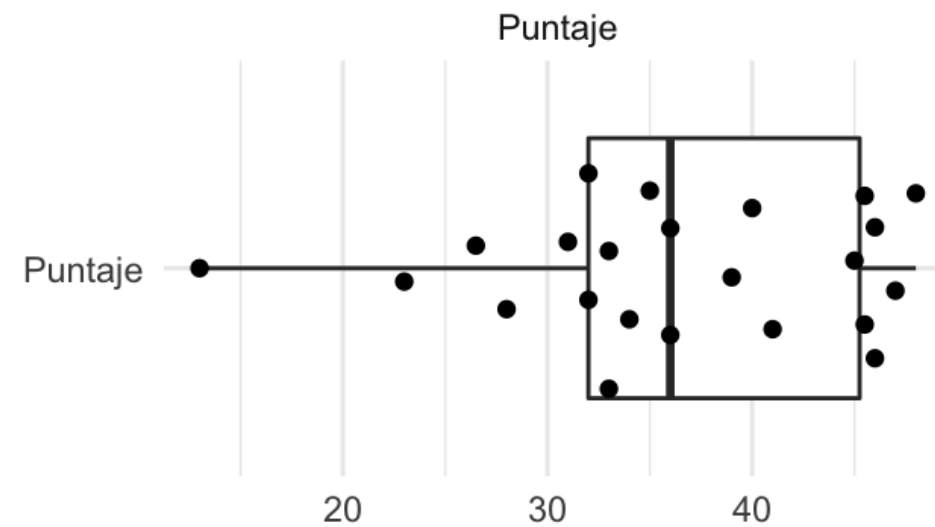
Notas

Control II

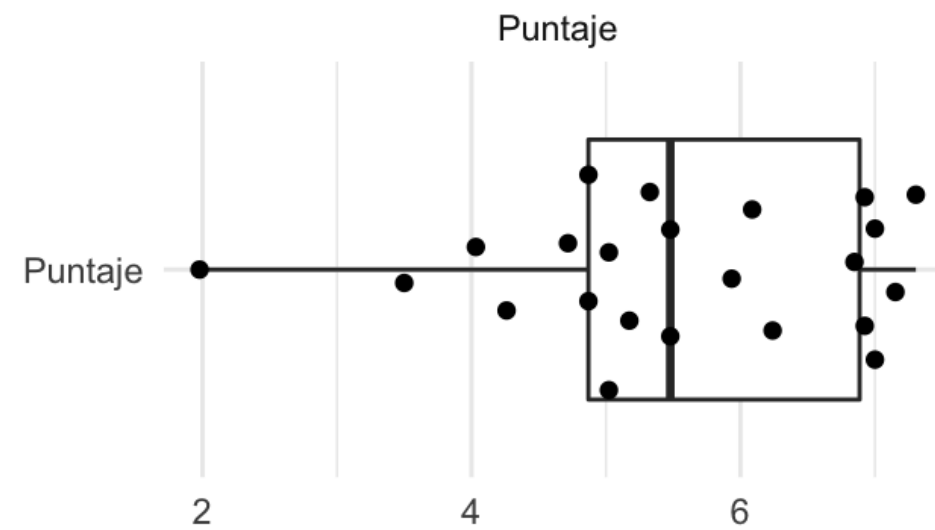
Preguntas



Puntaje



Notas



- $\text{Nota} = \text{Puntaje} / 46 * 7$
- Promedio = 5,5
- En promedio incremento de 0.6 Decimas.

Vistas

Vistas

Consultas con nombre

- Las vistas proporcionan una forma de empaquetar consultas en un objeto predefinido.
- Una vez creadas son similares a tablas de solo lectura.
 - `CREATE VIEW nombre_vista AS SELECT consulta`
- Las vistas son dinamicas, cada vez que son invocadas se ejecuta el `SELECT` otra vez
- Alternativamente son llamadas consultas con nombre.
- Usos:
 - Para almacenar consultas frecuentes o complejas.
 - Para crear versiones de tablas mas amigables al usuario (tiempo y fechas).
- `DROP VIEW nombre_vista;` # que pasa con los datos?

Vista de un Join anidado

Ejemplo

- Crear una vista de un join anidado

```
create view join_album_artista_track as select
* from artist natural join album join track
using (albumid)
```

- Como desplegamos los elementos de la lista?

```
select * from join_album_artista_track limit
10;
```

- Como eliminamos la vista?

```
drop view join_album_artista_track;
```

```
[7] 1 %%sql
2 create view join_album_artista_track as select * from artist natural join album join track using (albumid)

* sqlite:///content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.
[]
```

```
[8] 1 %%sql
2 select * from join_album_artista_track limit 10;
```

ArtistId	Name	AlbumId	Title	TrackId	Name:1	MediaTypeId	GenreId	Composer
1	AC/DC	1	For Those About To Rock We Salute You	1	For Those About To Rock (We Salute You)	1	1	Angus Young, Malcolm Young, I
2	Accept	2	Balls to the Wall	2	Balls to the Wall	2	1	None
2	Accept	3	Restless and Wild	3	Fast As a Shark	2	1	F. Baltes, S. Kaufman, U. Dirks Hoffman
2	Accept	3	Restless and Wild	4	Restless and Wild	2	1	F. Baltes, R.A. Smith-Diesel, S. Dirkschneider & W. Hoffman
2	Accept	3	Restless and Wild	5	Princess of the Dawn	2	1	Deaffy & R.A. Smith-Diesel
1	AC/DC	1	For Those About To Rock We Salute You	6	Put The Finger On You	1	1	Angus Young, Malcolm Young, I
1	AC/DC	1	For Those About To Rock We Salute You	7	Let's Get It Up	1	1	Angus Young, Malcolm Young, I
1	AC/DC	1	For Those About To Rock We Salute You	8	Inject The Venom	1	1	Angus Young, Malcolm Young, I
1	AC/DC	1	For Those About To Rock We Salute You	9	Snowballed	1	1	Angus Young, Malcolm Young, I
1	AC/DC	1	For Those About To Rock We Salute You	10	Evil Walks	1	1	Angus Young, Malcolm Young, I

```
1 #drop view
2 %%sql
3 drop view join_album_artista_track;
```

```
* sqlite:///content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.
[]
```


Indices

Indices

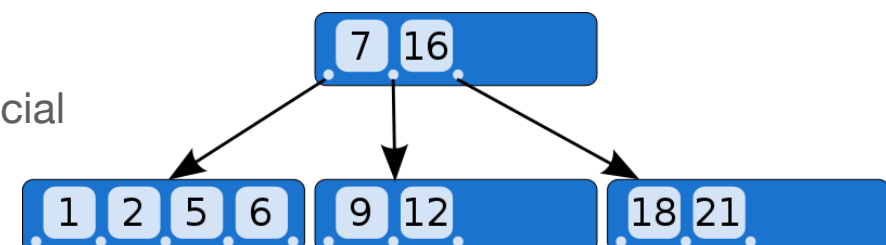
- Los indices sirven para optimizar consultas mediante el ordenamiento o indexación de uno o más columnas de una tabla.
 - La idea es encontrar filas sin la necesidad de revisar toda la tabla.
- Los indices requieren ser actualizados una vez que las tablas son modificadas.
 - `CREATE [UNIQUE] INDEX nombre_index ON nombre_tabla (atributo1, atributo2...)`
 - Los indices permiten valores duplicados pero la sentencia `UNIQUE` prohíbe repeticiones.
- Existen algunos estandar para nombrar indices:
 - `CREATE INDEX idx_alumno_nombre ON Alumno (nombre);`
- `DROP INDEX idx_alumno_nombre;` # que pasa con la tabla alumno?
- Las claves primarias?

Indices

Optimización de consultas

- Sin índices, el motor de base de datos está obligado de revisar las tablas completas en cada consulta.
 - Los JOINS son costosos computacionalmente.
 - La idea es encontrar filas sin la necesidad de revisar toda la tabla.
- Cada índice agrega una carga adicional a INSERT, UPDATE y DELETE.
 - Debemos evaluar donde colocar los índices.
- Los índices no se pueden crear en Vistas.
- Como almacena los datos el motor SQL?
- Por defecto, cada tabla es almacenada utilizando una estructura indexada (B-Tree SQLite).
 - A medida que se insertan filas en el B-Tree, las filas se ordenan, organizan y optimizan, de modo que una fila con un ROWID específico y conocido se puede recuperar de manera relativamente directa y rápida.

Permite: búsquedas, inserciones, deletaciones y acceso secuencial
 $\text{Time}(n) = O(\log n)$



- Cuando creamos un índice, el sistema de base de datos crea otro B-Tree para almacenar los datos del índice.
- El nuevo B-Tree se ordena y organiza usando la columna o columnas que se especifican en la definición del índice (! ROWID).

Indices

Ejemplo

- Creamos una tabla con 4 x 1000 numeros.

```
create table tbl (a,b,c,d);
INSERT INTO tbl (a, b,c,d)
VALUES
(84,39,78,79),
(182,39,67,153),
(83,166,143,188),
(145,205,380,366),
(317,358,70,303), ...
```

```
1 %%time
2 %%sql
3 select * from tbl where a=29238;
```

```
* sqlite:////content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.
CPU times: user 5.2 ms, sys: 0 ns, total: 5.2 ms
Wall time: 7.23 ms
  a      b      c      d
29238 3171  23996 48794
29238 297097 765679 213680
```

```
[12] 1 %%time
      2 %%sql
      3 create index idx_tbl_a_b ON tbl(a,b)
```

```
* sqlite:////content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.
CPU times: user 9.09 ms, sys: 32 µs, total: 9.13 ms
Wall time: 22.4 ms
[]
```

```
1 %%time
2 %%sql
3 select * from tbl where a=29238;
```

```
[>] * sqlite:////content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.
CPU times: user 3.88 ms, sys: 0 ns, total: 3.88 ms
Wall time: 3.76 ms
  a      b      c      d
29238 3171  23996 48794
29238 297097 765679 213680
```

Practicar en GoogleColab!!!

The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'The SQLite Query Optimizer', 'SQL_IV_chinook_db.ipynb', 'Home / Twitter', 'join sqlite - Google Search', and 'SQLite: Joins'. The notebook's address bar shows the URL: `colab.research.google.com/drive/1hyhjRD2swHrT4y3acApBX6xXxCorLZ5b#scrollTo=uBTPUc-qyoCm`. The notebook title is 'SQL_IV_chinook_db.ipynb' with a star icon. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', and 'All changes saved'. The left sidebar has icons for 'Code', 'Text', 'Find', 'Run', and 'Table'. The main area shows a table with 18 rows and 1 column, titled 'Name', generated by SchemaSpy. Below this, there is a paragraph explaining primary and foreign key constraints. The notebook is divided into sections: 'SQL con Chinook' and 'Joins'. Under 'Joins', there are two code blocks. The first block contains a CROSS JOIN query, and the second block contains an INNER JOIN query. Both queries are executed, and their results are displayed as tables. The first table shows the result of a CROSS JOIN between the 'album' and 'Artist' tables, limited to 5 rows. The second table shows the result of an INNER JOIN between the 'album' and 'Artist' tables, limited to 5 rows.

Generated by SchemaSpy

In the above ER diagram, the tiny vertical key icon indicates a column is a primary key. A primary key is a column (or set of columns) whose values uniquely identify every row in a table. For example, `Employeeid` is the primary key in the `Employee` table.

The relationship icon indicates a foreign key constraint and a one-to-many relationship.

SQL con Chinook

Joins

```
1 # CROSS Join example
2 %%sql
3 select * from album CROSS JOIN Artist limit 5;
```

* sqlite:///content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.

AlbumId	Title	ArtistId	ArtistId_1	Name
1	For Those About To Rock We Salute You	1	1	AC/DC
1	For Those About To Rock We Salute You	2	2	Accept
1	For Those About To Rock We Salute You	3	3	Aerosmith
1	For Those About To Rock We Salute You	4	4	Alanis Morissette
1	For Those About To Rock We Salute You	5	5	Alice In Chains

```
[ ] 1 #INNER Join
2 %%sql
3 select * from album JOIN Artist ON album.artistId=Artist.artistId limit 5;
```

* sqlite:///content/chinook-database/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite
Done.

AlbumId	Title	ArtistId	ArtistId_1	Name
1	For Those About To Rock We Salute You	1	1	AC/DC
2	Balls to the Wall	2	2	Accept
3	Restless and Wild	2	2	Accept
4	Let There Be Rock	1	1	AC/DC
5	Big Ones	3	3	Aerosmith

```
[ ] 1 #SELECT ... FROM alumno JOIN curso USING (aid,...)
2 %%sql
```

0s completed at 9:43 AM

Consultas?

Consultas o comentarios?

Muchas gracias