


Open · Opened 2 days ago by  **Rubén Montero**

Hagamos algo con esa conexión

Resumen

- Expondremos la estructura de `movies.db`
- Emplearemos `.createStatement` y `.executeQuery` para lanzar un `SELECT` que consulte los títulos de todas las películas
- Asignaremos el resultado de la consulta a un objeto `ResultSet`
- Usaremos un bucle `while` para imprimir, uno a uno, los valores obtenidos

Descripción

Hemos abierto una conexión a una base de datos local SQLite, y luego la hemos cerrado.

Estaría bien **hacer algo** en el medio. Por ejemplo, **consultar** datos.

¿Qué datos?

Nuestra base de datos `movies.db` contiene información sobre películas. Es muy **sencilla**. Contiene **sólo una tabla** llamada `TMovies` con los siguientes **campos** (o *columnas*)

- **id** (INTEGER)
- **title** (VARCHAR(200))
- **year** (SMALLINT)
- **duration** (SMALLINT)
- **countryIso3166** (CHAR(2))
- **genre** (VARCHAR(30))
- **synopsis** (VARCHAR(1000))

¿Columnas?

Efectivamente. Recordemos que la idea del [modelo relacional](#) es que los datos se almacenan en forma de tablas:

id	title	year	duration	countryIso3166	genre	synopsis
1	The Shawshank Redemption	1994	142	US	drama	Acusado del asesinato de su mujer, Andrew Dufresne (Tim Robbins), tras ser condenado a cadena perpetua, es enviado a la cárcel de Shawshank. Con el paso de los años...
2	Secondhand Lions	2003	109	US	comedy	Cuando al introvertido Walter (Haley Joel Osment) lo obliga su irresponsable madre (Kyra Sedgwick) a pasar el verano en un rancho de Texas con sus viejos y excéntricos tíos (Michael Caine y Robert Duvall), a los que apenas conoce, la idea no le hace ninguna gracia. Pero tampoco a ellos les agrada mucho la idea de cuidar del chaval. Sin embargo...
...

A pesar de que la **gracia de SQL** está en que diferentes tablas se **relacionan** entre sí, en nuestro primer caso de uso, sólo lanzaremos `SELECT` sencillos a **una** tabla `TMovies`

SELECT SQL

Una sentencia `SELECT`, según el estándar SQL, tiene **2** partes obligatorias:

- `SELECT`
- `FROM`

En `SELECT` ha de especificarse el campo o campos que se quieren *consultar*.

En `FROM` ha de especificarse el nombre de la tabla.

Un **SELECT** sencillo

Vamos a consultar sólo el campo **title** de las filas de la solitaria tabla **TMovies**. Así:

```
SELECT title
FROM TMovies;
```

Un momento, eso es SQL

Yo estoy escribiendo Java

Correcto.

Para **preparar** una consulta, debemos *invocar* el método `.createStatement()` sobre la *instancia* de nuestra conexión. Ello devuelve un objeto de tipo **Statement**

```
Statement statement = conn.createStatement();
```

Sobre dicho objeto podemos invocar `.executeQuery`, y pasar como parámetro **el código SQL** en formato **String**. Devolverá el **resultado** de la consulta en un objeto de tipo **ResultSet**:

```
ResultSet result = statement.executeQuery("SELECT title FROM TMovies");
```

Leyendo el resultado

Nuestro objeto **ResultSet** contiene los datos resultantes, **y** un cursor para acceder a ellos uno por uno

¿Cursor?

Es como un puntero que almacena internamente *qué* fila estamos leyendo.

Por ejemplo, si una hipotética consulta `"SELECT name FROM developers;"` devuelve un resultado con 3 filas:

name
Alice
Bob
Charlie

...invocar `.next()` *avanzará* el cursor al siguiente resultado (el primero):

```
result.next();
```

name
Alice
Bob
Charlie

Una vez ahí, podemos recuperar los datos *de esa fila* con las sentencias `.getXXX`. Debemos saber **qué tipo de dato** queremos recuperar (va en el nombre del método), y de **qué columna** (se pasa por parámetro).

Por ejemplo, para acceder a una **cadena de texto** (e.g.: VARCHAR) de la **primera** columna (¡sólo hay **una**!), invocamos:

```
// Devuelve "ALice"
String primerNombre = result.getString(1);
```

¡Ojo! Se empieza en **1**, **no** en **0**.

Si te resulta más sencillo, puedes invocar `.getXXX` pasando el *nombre del campo*:

```
// Esto es equivalente
// Devuelve "ALice"
```

```
String primerNombre = result.getString("name");
```

Seguimos *iterando*

Invocar `.next()` de nuevo, *avanzará el cursor*:

```
result.next();
```

name
Alice
Bob
Charlie

Así que ahora, la misma sentencia de antes, devuelve un resultado distinto.

Esta vez, asociado a la segunda **fila**:

```
// Devuelve "Bob"
String segundoNombre = result.getString(1);
```

Otra vez más

```
result.next();
```

name
Alice
Bob
Charlie

```
// Devuelve "Charlie"
String tercerNombre = result.getString(1);
```

Y... ¿otra?

Cuando **no hay más filas** que recorrer, la invocación `.next()` devuelve un valor *booleano* **false** :

```
// Devuelve false
boolean hayMasFilas = result.next();
```

Se me ocurre algo...

¡Usar `.next()` como condición de salida de un bucle **while** !

¡Estupenda idea!

Así podremos escribir *dentro del cuerpo* de un [bucle while](#) el código que procesa *cada fila*. Cuando no haya más filas, `.next()` devolverá **false** y el bucle **no** se repetirá:

```
while (result.next()) {
    // Hacer cosas con la fila...
}
```

La tarea

Se pide que, en el constructor de `MoviesDataProvider` , después de abrir la conexión y antes de cerrarla...

- Lances una consulta SQL `"SELECT title FROM TMovies"`
- Almacenes el resultado en un `ResultSet result`
- Escribas un bucle **while** que se ejecute mientras haya filas disponibles

- En dicho bucle, harás un `System.out.println` de la primera (y única) columna tipo `String` de cada fila

Por último

Verifica que el `test` funciona correctamente.

Haz `commit` y `push` para subir los cambios al repositorio.



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 2](#) 2 days ago