El futuro digital es de todos

MinTIC

# Unidad 5

Universidad Tecnológica de Pereira

MisiónTIC 2022

El futuro digital es de todos

MinTIC

# Tema 4: Almacenamiento de datos - SQLite en Android: Consultar Datos

Universidad Tecnológica de Pereira

MisionTIC2022

# SQLite: Consultar Datos

Para consultar los datos almacenados en una base de datos, podremos usar dos métodos:

rawQuery(): el cual recibe dos parámetros, la sentencia SELECT propia de SQL y otro que indicará los argumentos de la cláusula WHERE si esta existiera.

query(): el cual recibe siete parámetros.

1. String: nombre de la tabla a consultar.
2. Array de strings: las columnas que queremos recuperar. Se usa null para todas las columnas.
3. String: la cláusula WHERE si es necesaria, si no ponemos null.
4. Array de strings: los argumentos de la cláusula WHERE anterior si existiera.
5. String: la cláusula GROUP BY si es necesario, si no será null.
6. String: la cláusula HAVING si es necesario, si no será null.
7. String: la cláusula ORDER BY si es necesario, si no será null.

Ambos métodos devolverán un objeto de tipo Cursor que contendrá los datos de la consulta.

# Veamos un ejemplo usando ambos métodos

Empezando por **a. rawQuery()**

```
> Cursor c = db.rawQuery("SELECT _id, user, comment FROM comments", null);
```

Y para el caso del método **b. query()**

```
> Cursor c = db.query("comments", new String[]{"_id", "user", "comment"}, null, null, null, null. null):
```

Ambos cursores tendrán los mismos datos, que serán todos los de la tabla *comments* devolviendo los campos *user* y *comment*

# Veamos cómo acceder a los datos a través del cursor devuelto

**Para iterar sobre las filas devueltas por un cursor** usaremos principalmente dos métodos:

### a. moveToFirst()

Para posicionarnos al principio del cursor

### b. moveToNext()

Para movernos a la siguiente fila.

Para obtener los **valores de cada fila**, usaremos los métodos `getString()` , `getInt()` , `getFloat()` , etc., todas ellas recibiendo como único parámetro un entero que identifica la posición de la columna que queramos obtener.

En nuestro caso, cada fila de la base de datos estaba compuesta por tres columnas:

- columna 0: _id
- columna 1: user
- columna 2: comment

**Si no sabemos** con exactitud qué posición ocupa la **columna** deseada, podemos recurrir al método `getColumnIndex()` , que recibe como parámetro una cadena de texto con el nombre de la columna de la cual queramos obtener el dato y devolviendo la posición que ocupa en la fila, y así asegurarnos que estamos obteniendo el valor deseado.

Un detalle importante a tener en cuenta es que una vez finalicemos de manejar el cursor, es conveniente llamar al método close() para cerrarlo y así liberar todos sus recursos no pudiendo hacer uso de él más.

Veamos un **ejemplo** teniendo en cuenta nuestro cursor creado anteriormente:

Descargar código

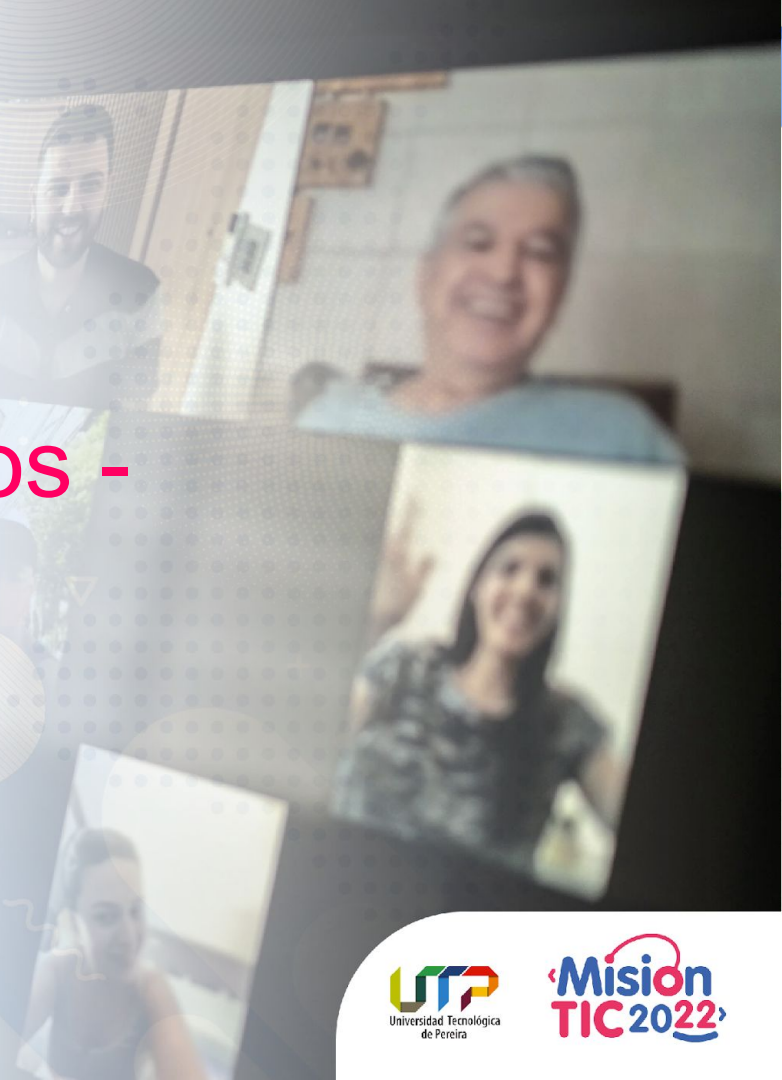
Añadimos este código a nuestra aplicación quedando de la siguiente forma:

Descargar código

El futuro digital es de todos

MinTIC

# Tema 5: Almacenamiento de datos - SQLite en Android: Inserción de Datos



Universidad Tecnológica de Pereira

MisionTIC2022

# SQLite Inserción de Datos

Para insertar datos en una base de datos tendremos dos posibilidades:

## a. execSQL()

Usando este método, pasando como parámetro un String con la sentencia SQL necesaria para insertar datos.

## b. insert()

*Usando este método,* pasando tres parámetros:

- El nombre de la tabla.
- Nombre de la columna, en el caso de que queramos insertar un registro completamente a nulo (normalmente este parámetro lo pondremos a null en la mayoría de los casos)
- Un objeto de tipo *ContentValues* que contendrá los datos a insertar en forma de clave-valor, siendo la clave el nombre de la columna y el valor el dato a insertar en dicha columna.

# Vamos a insertar un dato a nuestra base de datos usando ambas posibilidades.

En el caso de usar el método `execSQL()` quedaría así:

```
db.execSQL("INSERT INTO comments (user, comment) VALUES ('Digital Learning', 'Esto es un comentario insertado usando el método execSQL()')");
```

Y para el caso de usar el método `insert()` quedaría tal que así:

```
ContentValues cv = new ContentValues();
cv.put("user", "Academia Android");
cv.put("comment", "Esto es un comentario insertado usando el método insert()");
db.insert("comments", null, cv);usando el método execSQL()')");
```

Vemos que no es necesario insertar un valor al campo _id ya que lo hemos declarado como autoincrementable, cada vez que añadamos un nuevo comentario se le asignará un id automáticamente.

Añadimos este código en el método onCreate de MainActivity: https://bit.ly/3jYnjuH

El futuro digital es de todos

MinTIC

# Tema 6: Almacenamiento de datos - SQLite en Android: Eliminación de Datos

Universidad Tecnológica de Pereira

MisionTIC2022

# SQLite Eliminación de Datos

Para la eliminación de datos tendríamos de nuevo dos posibilidades:

## a. execSQL()

**Usar el método** execSQL() , pasando como parámetro la sentencia SQL para la eliminación de datos.

## b. delete()

**Usar el método** delete() que recibirá tres parámetros:

- Nombre de la tabla
- La cláusula WHERE
- Y el tercero, al igual que ocurría con el método *update()*, serían los argumentos de la cláusula WHERE, si así lo requiere, si no, tomaría el valor null.

# SQLite Eliminación de Datos

Veamos un ejemplo usando ambos métodos, primero con *execSQL()*:

```
> db.execSQL("DELETE FROM comments WHERE user='Digital Learning'");
```

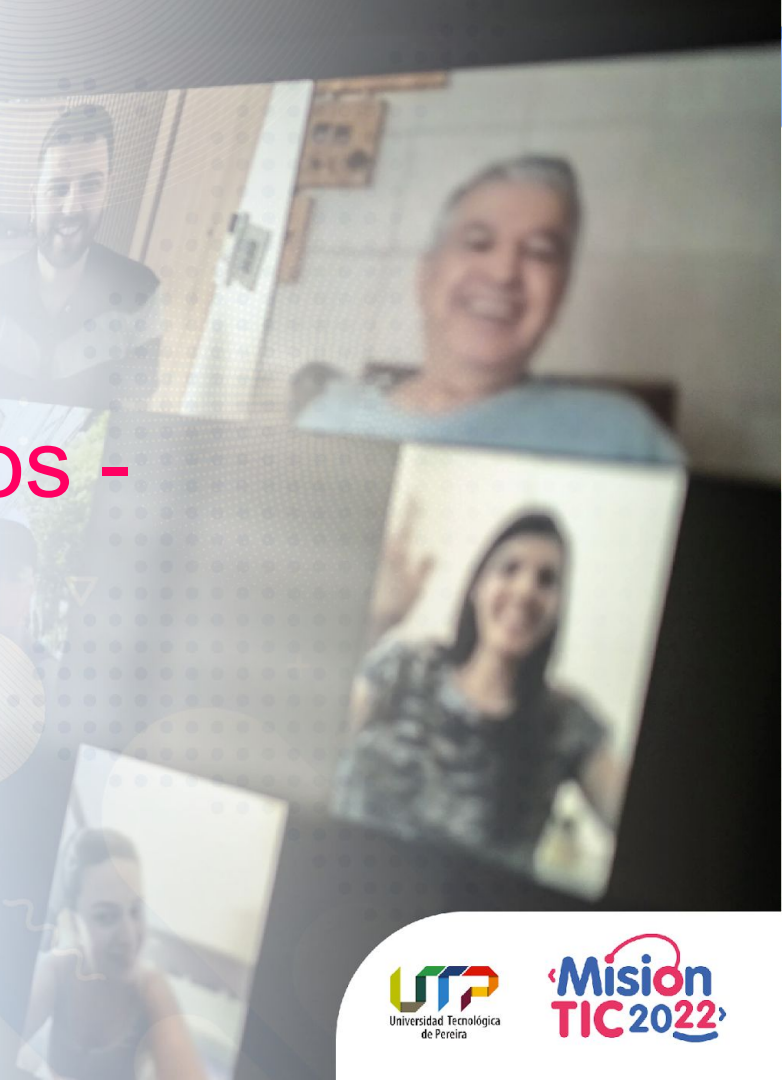
Y ahora con *update()* usando el tercer parámetro:

```
> String[] args = new String[]{"Academia Android"};
> db.delete("comments", "user=?", args);
```

El futuro digital es de todos

MinTIC

# Tema 7: Almacenamiento de datos - SQLite en Android: Edición de Registros



Universidad Tecnológica de Pereira

MisionTIC2022

# SQLite Edición de Registros

Para actualizar un registro ya insertado previamente en nuestra base de datos, tendremos las siguientes dos posibilidades:

## a. execSQL()

**Usando de nuevo el método** execSQL() pero esta vez pasando como parámetro la sentencia SQL propia para la actualización de un campo.

Con él tendríamos lo siguiente:

## b. update()

**Usando el método** update() que recibe cuatro parámetros:

- El nombre de la tabla
- El objeto *ContentValues*, que incluye los nuevos datos a modificar.
- La condición WHERE de la sentencia SQL.
- El cuarto es un caso especial que veremos a continuación en un ejemplo para que se entienda mejor.

# SQLite Edición de Registros

Usando el método **a. execSQL()** tendríamos lo siguiente:

```
db.execSQL("UPDATE comments SET comment='Esto es un comentario
actualizado por el método execSQL()' WHERE user='Digital Learning'");
```

Con el método **b. update()** tendremos dos formas de hacerlo, y aunque las dos son totalmente viables, **la segunda es más recomendable por seguridad**. Vamos a verlas:

En primer lugar crearíamos un ContentValues con el valor actualizado como sigue:

```
ContentValues cv = new ContentValues();
cv.put("comment", "Esto es un comentario actualizado por el método update()");
```

# SQLite Edición de Registros

Ambas formas producen el mismo resultado.

y a continuación, tendremos las dos alternativas posibles:

a) Poniendo el cuarto parámetro a null:

```
db.update("comments", cv, "user='Academia Android'", null);
```

b) O bien usando el cuarto parámetro. En lugar de pasar directamente el argumento de la cláusula WHERE, le indicamos el valor '?' y usamos el cuarto parámentro para indicarle estos argumentos, que en este caso sólo es uno:

```
String[] args = new String []{ "Academia Android"};
```

El **segundo método es más recomendable por seguridad**. Separar los valores de la comparación hace nuestro código menos vulnerable a posibles fallos con la sintaxis de la consulta, e incluso más robusto ante posibles ataques en caso de que construyamos la consulta de forma dinámica a partir de datos introducidos por el usuario manualmente.

Añadimos el código de la actualización de datos a nuestra aplicación: https://cutt.ly/bmQSmFU