Logotipo, nombre de la empresa

El contenido generado por IA puede ser incorrecto.

**INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO**

**INSTITUTO TECNOLÓGICO DE CIUDAD MADERO**

**PERSISTENCIA DE DATOS**

**LUIS FERNANDO HERNANDEZ MERCADO 20070531**

**HUGO ANDRE MARTINEZ GALVAN 20070545**

**Unidad 6: Persistencia de datos**

**Ruta de aprendizaje 1**

[**Introducción a SQL**](https://developer.android.com/courses/pathways/android-basics-compose-unit-6-pathway-1?hl=es-419)

1. En este video aprendremos los aspectos básicos de las bases de datos relacionales y a usar SQL para interactuar con ellas.
2. Cómo usar SQL para leer y escribir en una base de datos

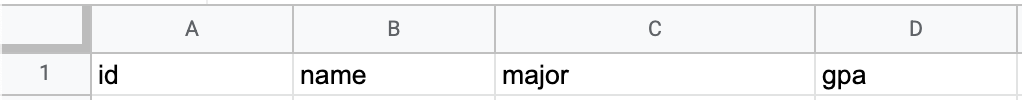
¿Qué es una base de datos?

Si usas un programa de hojas de cálculo, como Hojas de cálculo de Google, ya conoces una analogía básica de una base de datos.

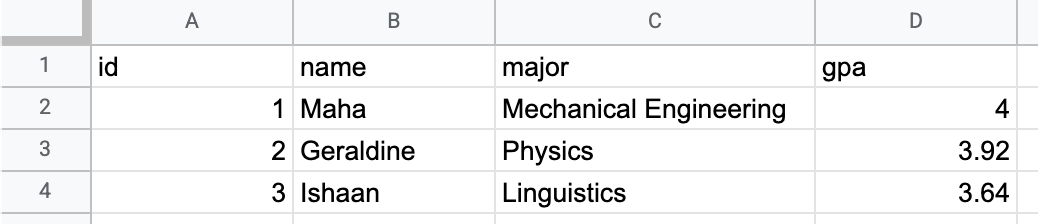
Una hoja de cálculo consta de tablas de datos separadas, o bien hojas individuales de la misma hoja de cálculo.



Cada tabla consta de columnas que definen lo que representan los datos y filas que representan elementos individuales con valores para cada columna. Por ejemplo, puedes definir columnas para el ID, el nombre, la especialización y la calificación de un alumno.



Cada fila contiene datos de un solo alumno, con valores de cada una de las columnas.



Una base de datos relacional funciona de la misma manera.

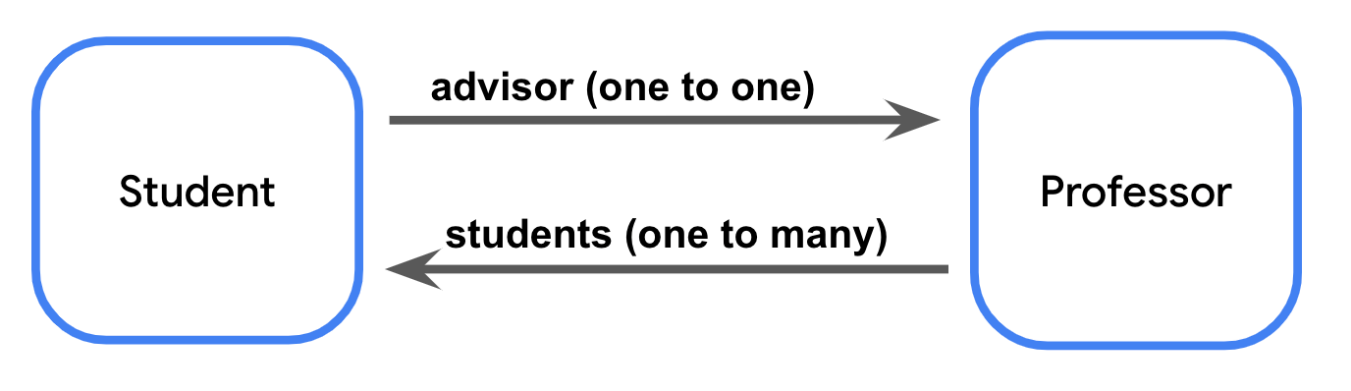
* Las tablas definen agrupaciones de datos de alto nivel que quieres representar, como alumnos y profesores.
* Las columnas definen los datos que contiene cada fila de la tabla.
* Las filas contienen los datos en sí que constan de valores para cada columna de la tabla.

La estructura de una base de datos relacional también refleja lo que ya sabes sobre clases y objetos en Kotlin.

data class Student(  
    id: Int,  
    name: String,  
    major: String,  
    gpa: Double  
)

* Las clases, como las tablas, modelan los datos que deseas representar en tu app.
* Como las columnas, las propiedades definen los datos específicos que debe contener cada instancia de la clase.
* Los objetos, como las filas, son los datos reales. Los objetos contienen valores para cada propiedad definida en la clase, al igual que las filas contienen valores para cada columna definida en la tabla de datos.

Así como una hoja de cálculo puede contener varias hojas y una app puede contener varias clases, una base de datos puede contener varias tablas. Una base de datos se denomina base de datos relacional cuando puede modelar relaciones entre tablas. Por ejemplo, un estudiante de posgrado puede tener un solo profesor como asesor de doctorado, mientras que ese profesor es el asesor de doctorado de varios alumnos.



Cada tabla de una base de datos relacional contiene un identificador único para las filas, como una columna en el que el valor de cada fila es un número entero que se incrementa automáticamente. Este identificador se conoce como la clave primaria.

Cuando una tabla hace referencia a la clave primaria de otra tabla, se conoce como clave externa. La presencia de una clave externa significa que existe una relación entre las tablas.

**Nota:** Al igual que con las clases de Kotlin, la convención es usar la forma singular para el nombre de las tablas de la base de datos. En el ejemplo anterior, eso significa que nombras las tablas teacher, student y course, no las formas plurales teachers, students y courses.

¿Qué es SQLite?

SQLite es una base de datos relacional de uso general. Específicamente, SQLite hace referencia a una biblioteca C liviana para la administración de bases de datos relacionales con el lenguaje de consulta estructurado, conocida como SQL, que a veces se pronuncia "sícuel".

No tendrás que aprender lenguaje C ni ningún otro lenguaje de programación completamente nuevo para trabajar con una base de datos relacional. SQL es solo una forma de agregar y recuperar datos de una base de datos relacional con algunas líneas de código.

**Nota:** No todas las bases de datos se organizan en tablas, columnas y filas. Otros tipos de bases de datos, conocidos como NoSQL, se estructuran de manera similar a un objeto JSON con pares de claves y valores anidados. Algunos ejemplos de bases de datos NoSQL incluyen Redis o [Cloud Firestore](https://firebase.google.com/docs/firestore?hl=es-419).

Cómo representar datos con SQLite

En Kotlin, conoces los tipos de datos, como Int y Boolean. Las bases de datos SQLite también usan tipos de datos. Las columnas de la tabla de datos deben tener un tipo de datos específico. En la siguiente tabla, se asignan tipos de datos comunes de Kotlin a sus equivalentes de SQLite.

|  |  |
| --- | --- |
| **Tipo de datos de Kotlin** | **Tipo de datos de SQLite** |
| Int | INTEGER |
| String | VARCHAR o TEXT |
| Boolean | BOOLEAN |
| Float, Double | REAL |

Las tablas en una base de datos y las columnas en cada tabla se conocen en conjunto como el *esquema*. En la siguiente sección, descargarás el conjunto de datos inicial y obtendrás más información sobre su esquema.

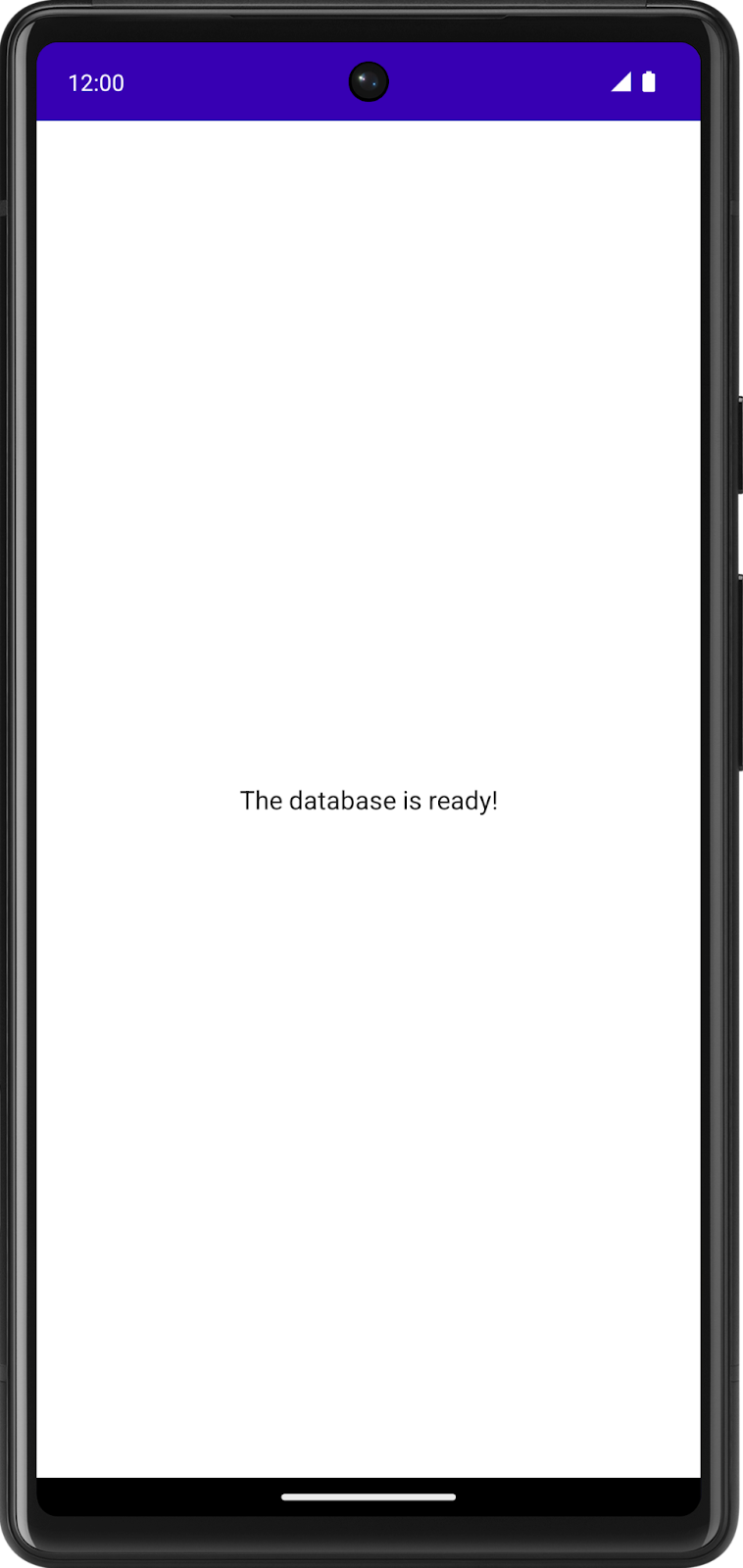
La base de datos de este codelab es para una app hipotética de correo electrónico. En este codelab, se usan ejemplos conocidos, como ordenar y filtrar correos electrónicos, o buscar por asunto o remitente, para demostrar todo lo que puedes hacer con SQL. Este ejemplo también garantiza que tengas experiencia con los tipos de situaciones que podrías encontrar en una app antes de trabajar con Room en la próxima ruta de aprendizaje.

Descarga el proyecto inicial de la rama compose del repositorio de GitHub de aspectos básicos de SQL [aquí](https://github.com/google-developer-training/android-basics-kotlin-sql-basics-app/tree/compose).

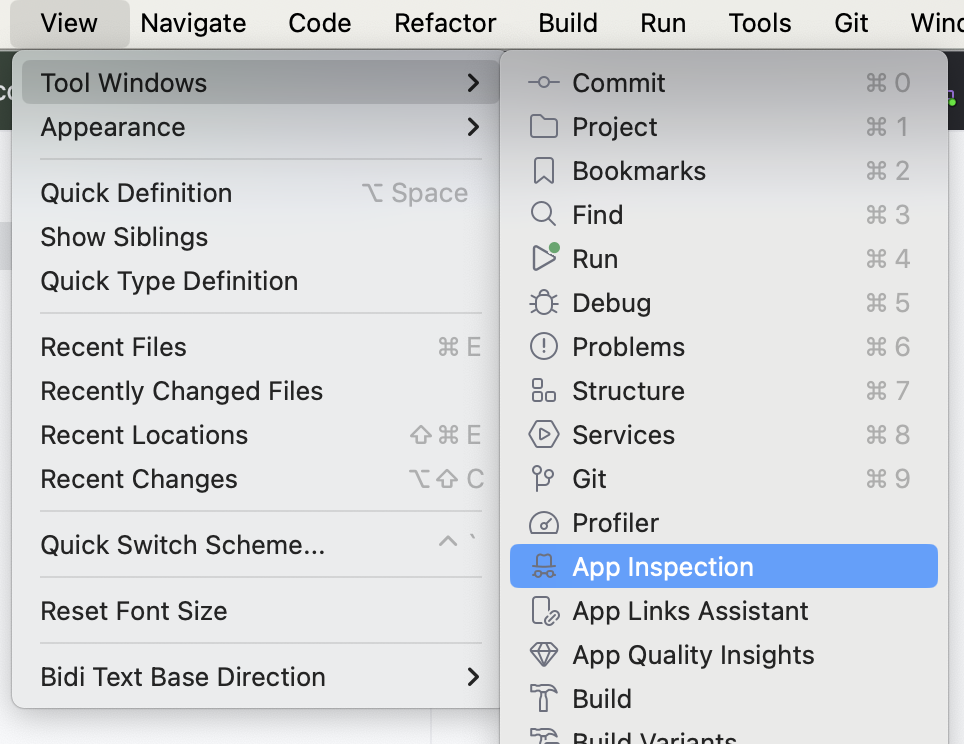
Cómo usar el Inspector de bases de datos

Para usar el Inspector de bases de datos, sigue estos pasos:

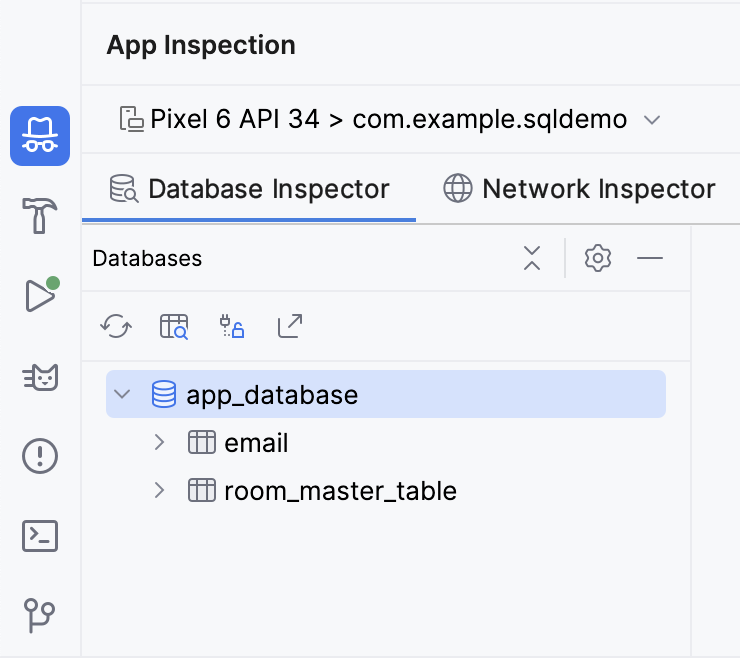
1. Ejecuta la app de SQL Basics en Android Studio. Cuando se inicie la app, verás la siguiente pantalla.



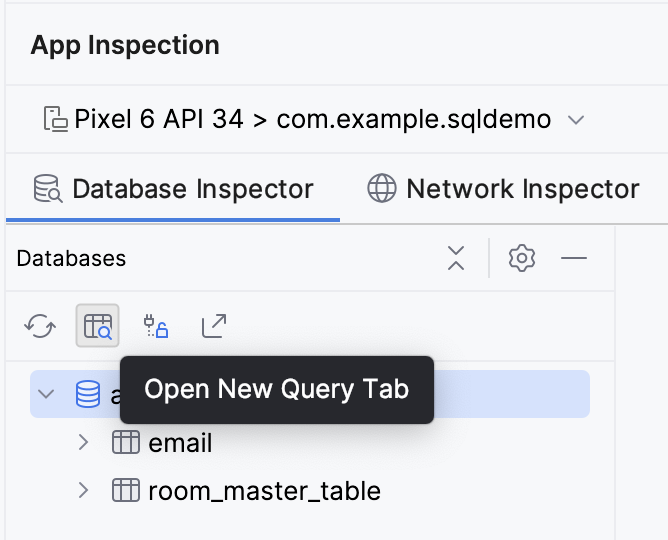
1. En Android Studio, haz clic en **View** > **Tool Windows** > **App Inspection**.



Ahora, verás una pestaña nueva en la parte inferior con la etiqueta **App Inspection** con la pestaña **Database Inspector** seleccionada. Hay dos pestañas adicionales, pero no es necesario que las utilices. Es posible que demore unos segundos en cargarse, pero deberías ver una lista a la izquierda con las tablas, que puedes seleccionar si deseas ejecutar búsquedas.



1. Haz clic en el botón **Open New Query Tab** para abrir un panel y ejecutar una consulta en la base de datos.



La tabla email tiene 7 columnas:

* id: Es la clave primaria.
* subject: Es el asunto del correo electrónico.
* sender: Es la dirección desde la que se originó el correo electrónico.
* folder: Es la carpeta en la que se puede encontrar el mensaje, como Recibidos o Spam.
* starred: Indica si el usuario destacó el correo electrónico.
* read: Indica si el usuario leyó el correo electrónico.
* received: Es la marca de tiempo en la que se recibió el correo electrónico.

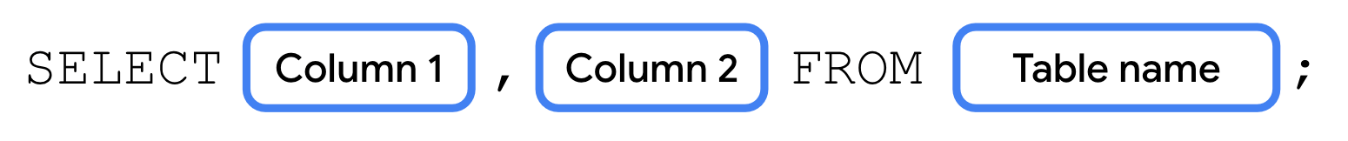
Instrucción SELECT de SQL

Una instrucción de SQL, a veces llamada consulta, se usa para leer o manipular una base de datos.

Leerás datos de una base de datos SQLite con una instrucción SELECT. Una instrucción SELECT simple consta de la palabra clave SELECT, seguida del nombre de la columna, seguida de la palabra clave FROM y, luego, el nombre de la tabla. Cada instrucción de SQL finaliza con un punto y coma (;).



Una instrucción SELECT también puede mostrar datos de varias columnas. Debes separar los nombres de las columnas con una coma.



Si deseas seleccionar todas las columnas de la tabla, usa el carácter comodín (\*) en lugar de los nombres de columna.



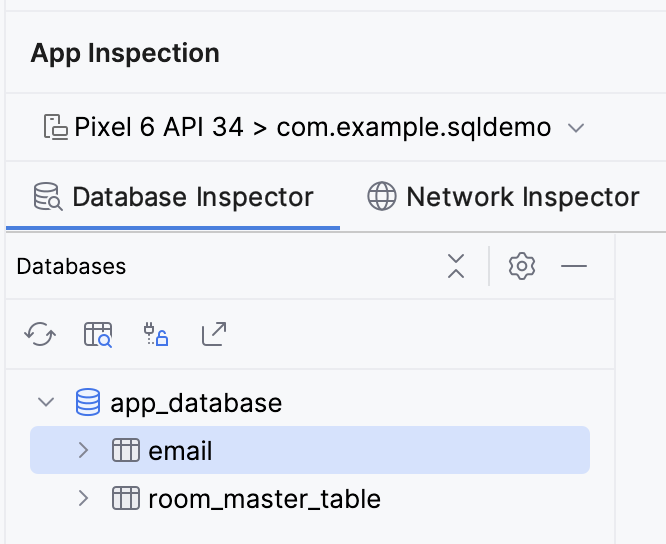
En cualquier caso, una instrucción SELECT simple como esta muestra todas las filas de la tabla. Solo debes especificar los nombres de columna que deseas que se muestren.

**Nota:** Si bien es común finalizar cada instrucción de SQL con un punto y coma (;), algunos editores, como el Inspector de base de datos de Android Studio, podrían permitirte omitir el punto y coma. En los diagramas de este codelab, se muestra un punto y coma al final de cada consulta en SQL completa.

Lee datos de correo electrónico con una instrucción SELECT

Una de las tareas principales que debe hacer una app de correo electrónico es mostrar una lista de mensajes. Con una base de datos SQL, puedes obtener esta información con una instrucción SELECT.

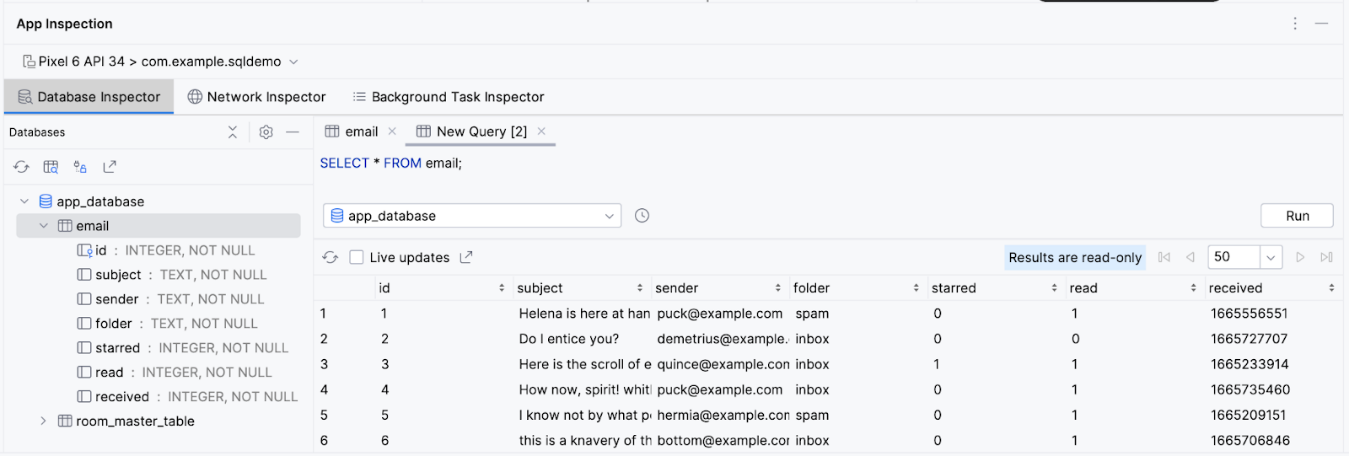
1. Asegúrate de que la tabla **email** esté seleccionada en el **Inspector de base de datos**.



1. Primero, intenta seleccionar todas las columnas de cada fila de la tabla email.

SELECT \* FROM email;

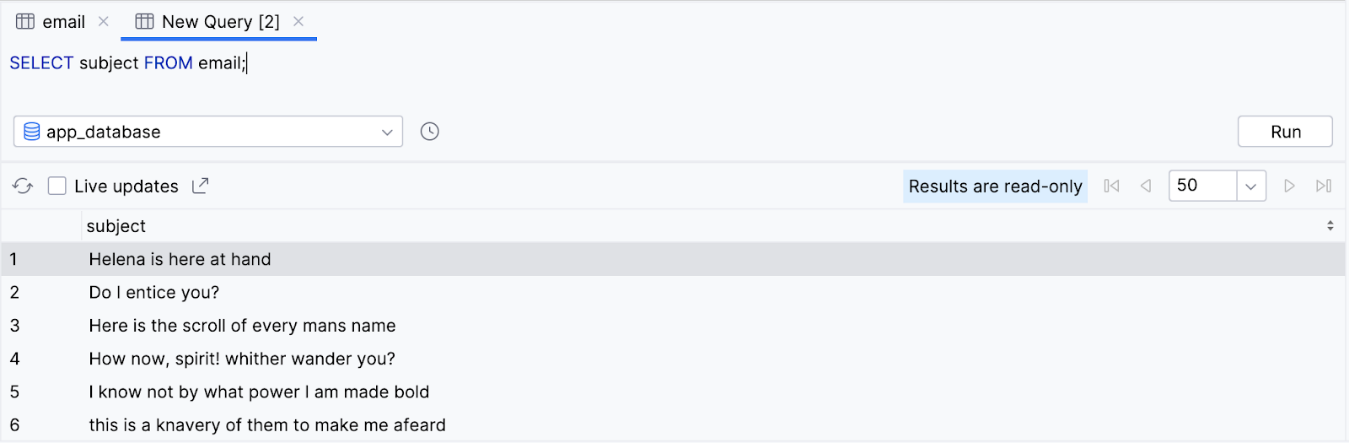
1. Haz clic en el botón **Run** de la esquina inferior derecha del cuadro de texto. Observa que se muestra toda la tabla email.



1. Ahora, intenta seleccionar solo el asunto para cada fila.

SELECT subject FROM email;

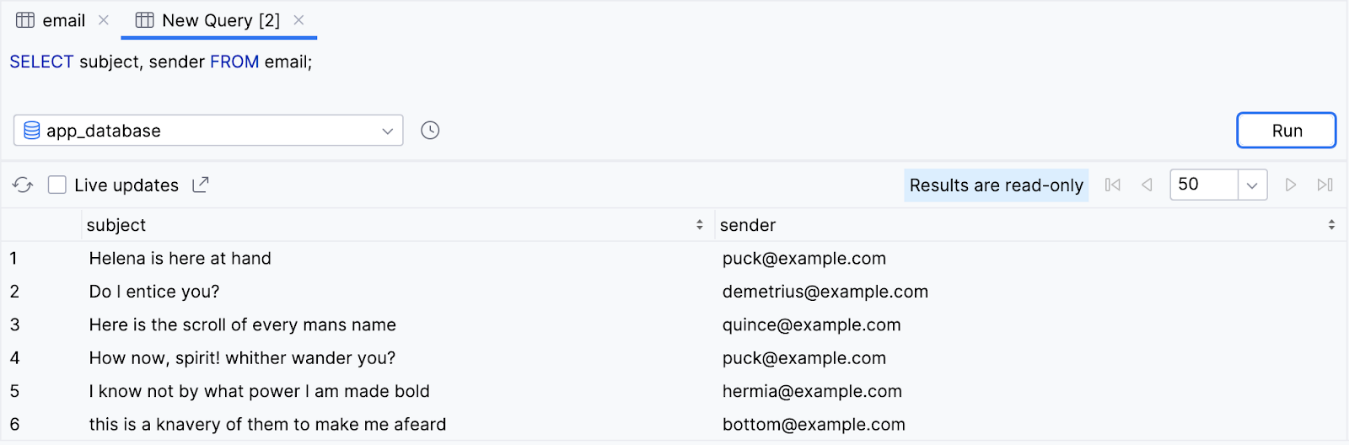
1. Observa que, una vez más, la consulta muestra cada fila, pero solo para esa columna.



1. También puedes seleccionar varias columnas. Intenta seleccionar el asunto y el remitente.

SELECT subject, sender FROM email;

1. Observa que la consulta muestra todas las filas de la tabla email, pero solo los valores de la columna del asunto y del remitente.



Puedes hacer que las instrucciones SELECT sean más específicas si agregas cláusulas para especificar un subconjunto de los datos y hasta cambiar el formato del resultado. En las siguientes secciones, aprenderás sobre las cláusulas de uso frecuente de las sentencias SELECT y cómo darles formato a los datos.

Reduce las columnas con funciones agregadas

Las instrucciones de SQL no se limitan a mostrar filas. SQL ofrece una variedad de funciones que pueden realizar una operación o un cálculo en una columna específica, como encontrar el valor máximo o contar la cantidad de valores posibles únicos para una columna en particular. Se las conoce como **funciones de agregación**. En lugar de mostrar todos los datos de una columna específica, puedes mostrar un solo valor de una columna específica.

Algunos ejemplos de funciones de agregación de SQL son los siguientes:

* COUNT(): Muestra la cantidad total de filas que coinciden con la consulta.
* SUM(): Muestra la suma de los valores de todas las filas de la columna seleccionada.
* AVG(): Muestra el valor medio (promedio) de todos los valores de la columna seleccionada.
* MIN(): Muestra el valor más pequeño en la columna seleccionada.
* MAX(): Muestra el valor más grande de la columna seleccionada.

En lugar de un nombre de columna, puedes llamar a una función de agregación y pasar el nombre de una columna como un argumento entre paréntesis.



En lugar de mostrar el valor de esa columna para cada fila de la tabla, se muestra un solo valor luego de la llamada a la función de agregación.

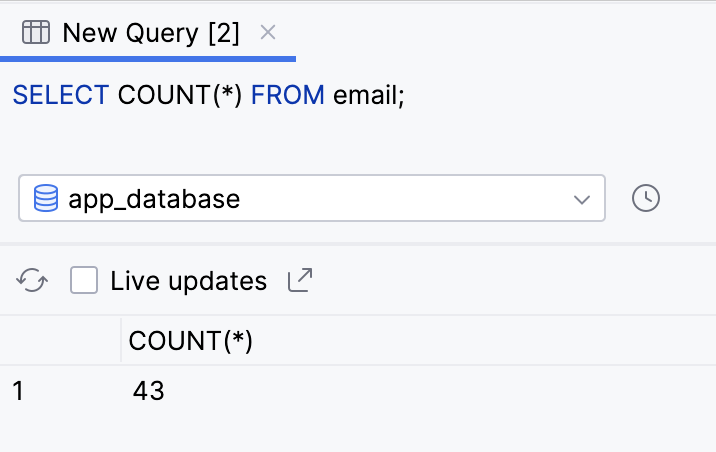
Las funciones de agregación pueden ser una forma eficiente de realizar cálculos en un valor cuando no necesitas leer todos los datos de una base de datos. Por ejemplo, es posible que desees encontrar el promedio de los valores de una columna sin cargar toda tu base de datos en una lista y hacerlo de forma manual.

Veamos en acción algunas de las funciones de agregación con la tabla email:

1. Una app podría querer obtener la cantidad total de correos electrónicos recibidos. Para ello, usa la función COUNT() y el carácter comodín (\*).

SELECT COUNT(\*) FROM email;

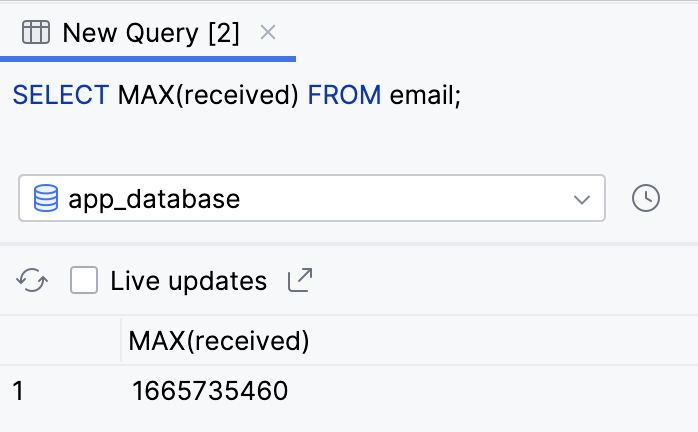
1. La consulta muestra un solo valor. Puedes hacerlo por completo con una consulta en SQL, sin código de Kotlin para contar las filas de forma manual.



1. Para obtener la hora del mensaje más reciente, puedes usar la función MAX() en la columna de recibidos porque la marca de tiempo de Unix más reciente es el número más alto.

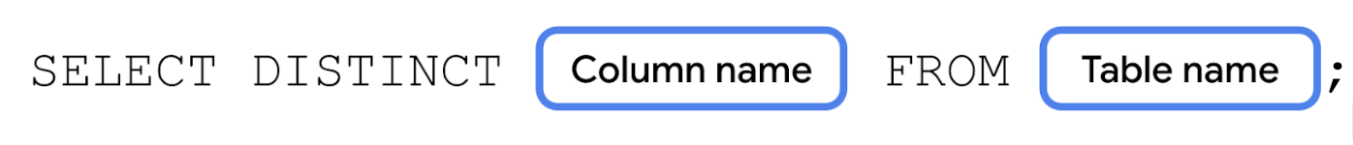
SELECT MAX(received) FROM email;

1. La consulta muestra un solo resultado, la marca de tiempo más alta (más reciente) en la columna de recibidos.



Filtra los resultados duplicados con DISTINCT

Cuando seleccionas una columna, puedes anteponer la palabra clave DISTINCT. Este enfoque puede ser útil si deseas quitar duplicados del resultado de la consulta.

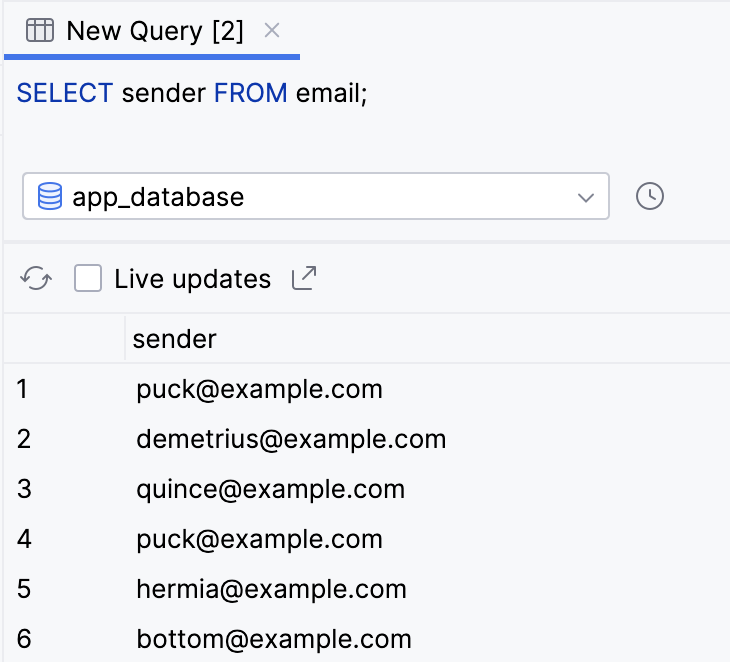


Por ejemplo, muchas apps de correo electrónico tienen una función de autocompletar para las direcciones. Es posible que desees incluir todas las direcciones de las que recibas un correo electrónico y mostrarlas en una lista.

1. Ejecuta la siguiente consulta para mostrar la columna sender para cada fila.

SELECT sender FROM email;

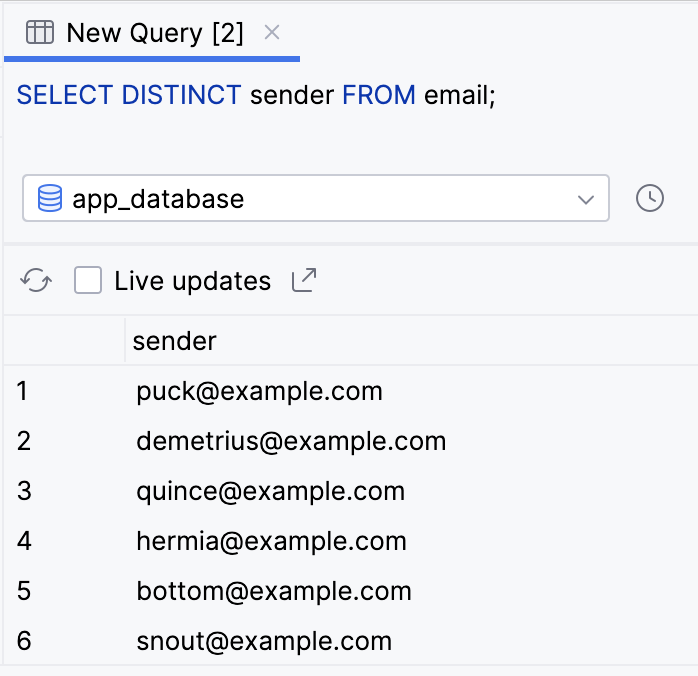
1. Observa que el resultado contiene muchos duplicados. Definitivamente, no es una experiencia ideal.



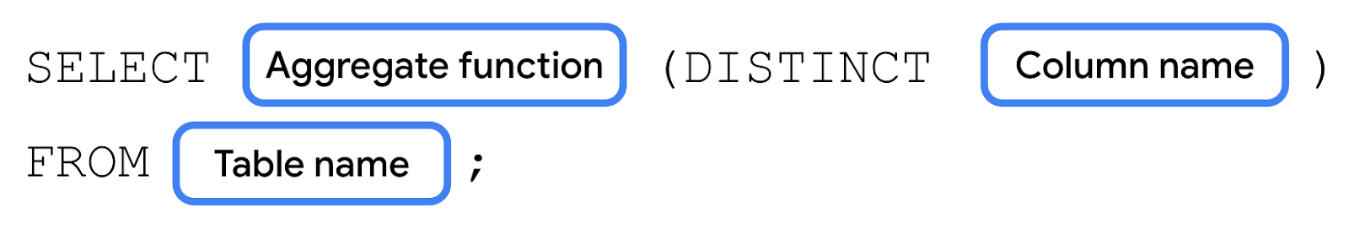
1. Agrega la palabra clave DISTINCT antes de la columna del remitente y vuelve a ejecutar la consulta.

SELECT DISTINCT sender FROM email;

1. Observa que el resultado ahora es mucho más pequeño y cada valor es único.



También puedes anteponer el nombre de la columna en una función de agregación con la palabra clave DISTINCT.

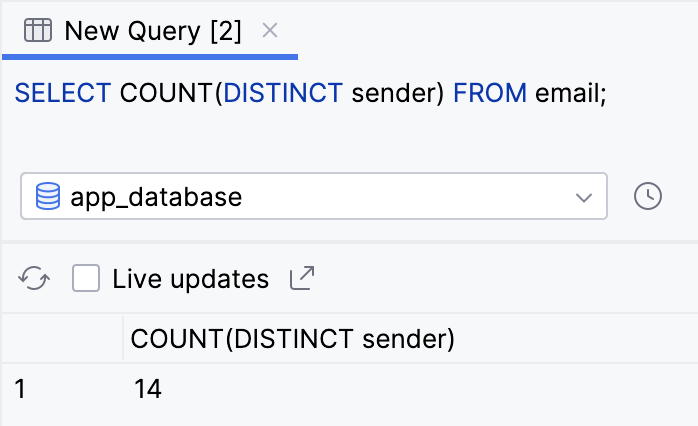


Supongamos que deseas conocer la cantidad de remitentes únicos en la base de datos. Puedes contar la cantidad de remitentes únicos con la función de agregación COUNT() y con la palabra clave DISTINCT en la columna sender.

1. Ejecuta una sentencia SELECT y pasa DISTINCT sender a la función COUNT().

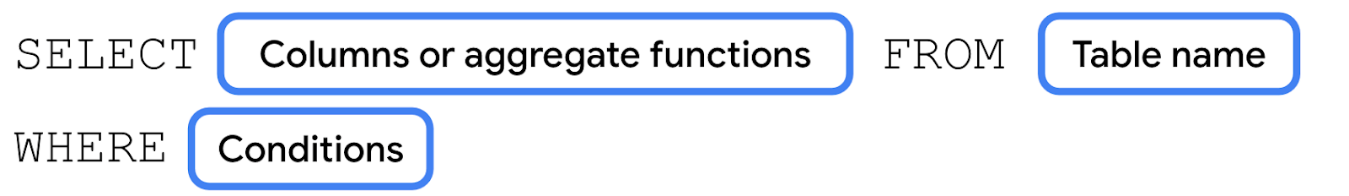
SELECT COUNT(DISTINCT sender) FROM email;

1. Observa que la consulta nos indica que hay 14 remitentes únicos.



Muchas apps de correo electrónico ofrecen la función para filtrar los mensajes que se muestran según ciertos criterios, como datos, término de búsqueda, carpeta, remitente, etc. Para estos tipos de casos de uso, puedes agregar una cláusula WHERE a la consulta SELECT.

Después del nombre de la tabla, en una línea nueva, puedes agregar la palabra clave WHERE seguida de una expresión. Cuando se escriben consultas en SQL más complejas, es común colocar cada cláusula en una línea nueva para facilitar la lectura.



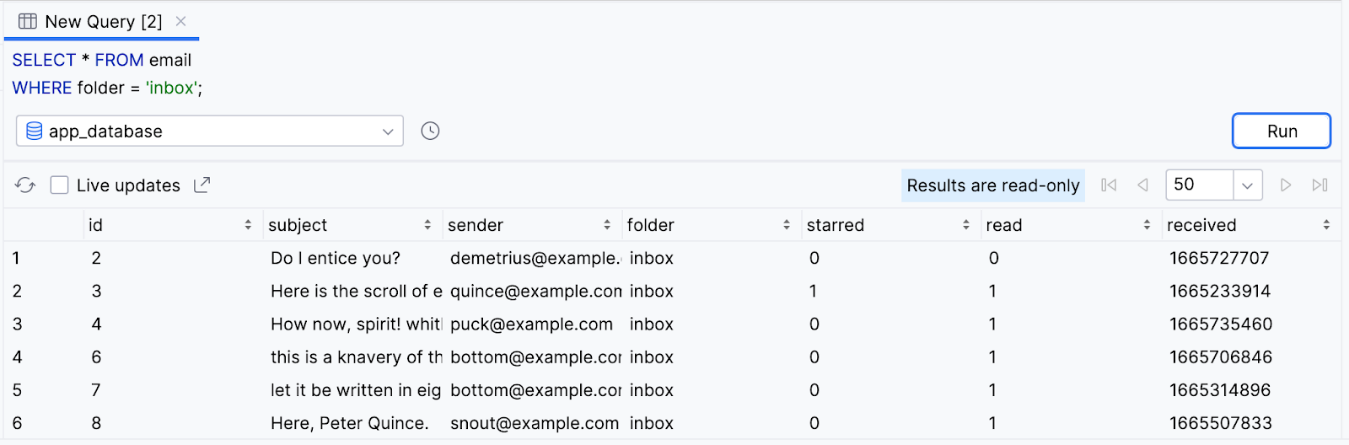
Esta consulta realiza una verificación booleana para cada fila seleccionada. Si el resultado es "true", incluye la fila en el resultado de la consulta. Las filas para las que la consulta muestra "false" no se incluyen en el resultado.

Por ejemplo, una app de correo electrónico puede tener filtros de spam, papelera, borradores o filtros creados por el usuario. En las siguientes sentencias, se hace esto con una cláusula WHERE:

1. Ejecuta una sentencia SELECT para mostrar todas las columnas (\*) de la tabla email, incluida una cláusula WHERE para verificar la condición folder = 'inbox'. No, no es un error tipográfico: usas un solo signo igual para verificar la igualdad en SQL y comillas simples en lugar de dobles para representar un valor de cadena.

SELECT \* FROM email  
WHERE folder = 'inbox';

1. El resultado solo muestra filas para los mensajes en la bandeja Recibidos del usuario.



**Nota:** Presta especial atención a los operadores de comparación de SQL.

A diferencia de Kotlin, el operador de comparación en SQL es un signo igual (=), en lugar de un signo igual doble (==).

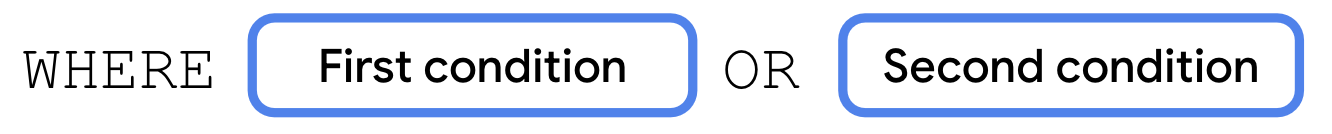
El operador de desigualdad (!=) es el mismo que el de Kotlin. SQL también proporciona los operadores de comparación <, <=, > y >=.

Operadores lógicos con cláusulas WHERE

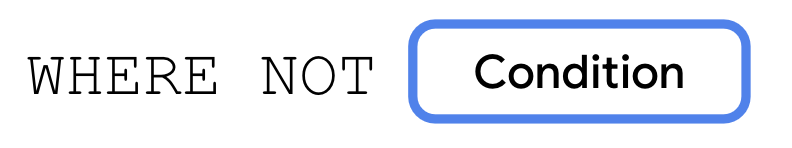
Las cláusulas WHERE de SQL no se limitan a una sola expresión. Puedes usar la palabra clave AND, que equivale al operador **and** de Kotlin (&&), para incluir solo los resultados que cumplan con ambas condiciones.



Como alternativa, puedes usar la palabra clave OR, que equivale al operador **or** de Kotlin (||), para incluir filas en los resultados que cumplan con cualquier condición.



Para facilitar la lectura, también puedes anular una expresión con la palabra clave NOT.



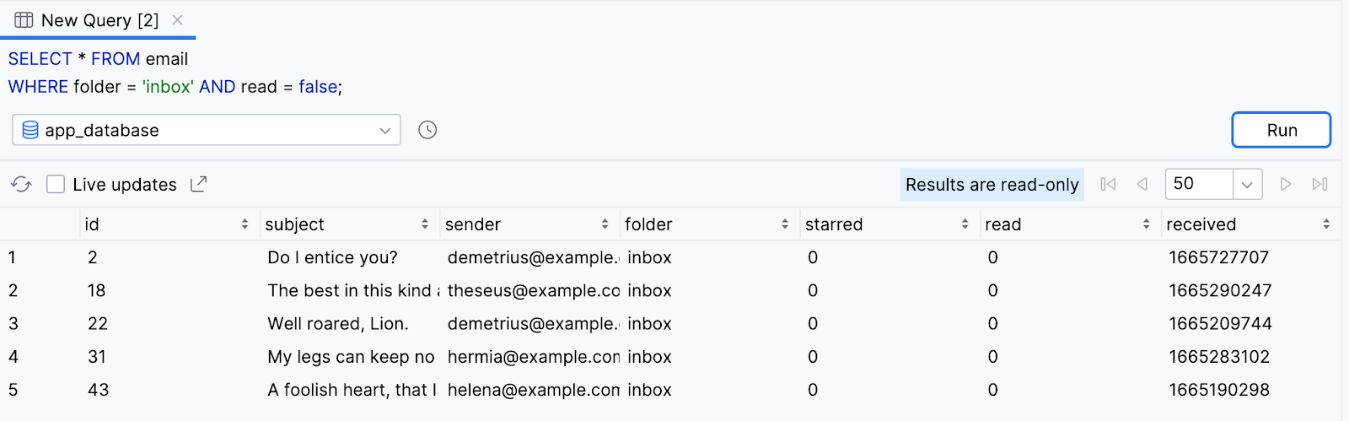
Muchas apps de correo electrónico admiten varios filtros; por ejemplo, mostrar solo los correos electrónicos no leídos.

Prueba las siguientes cláusulas WHERE más complicadas en la tabla email:

1. Además de mostrar solo los mensajes en la bandeja Recibidos del usuario, prueba limitar también los resultados a los mensajes no leídos, en los que el valor de la columna de leídos sea false.

SELECT \* FROM email  
WHERE folder = 'inbox' AND read = false;

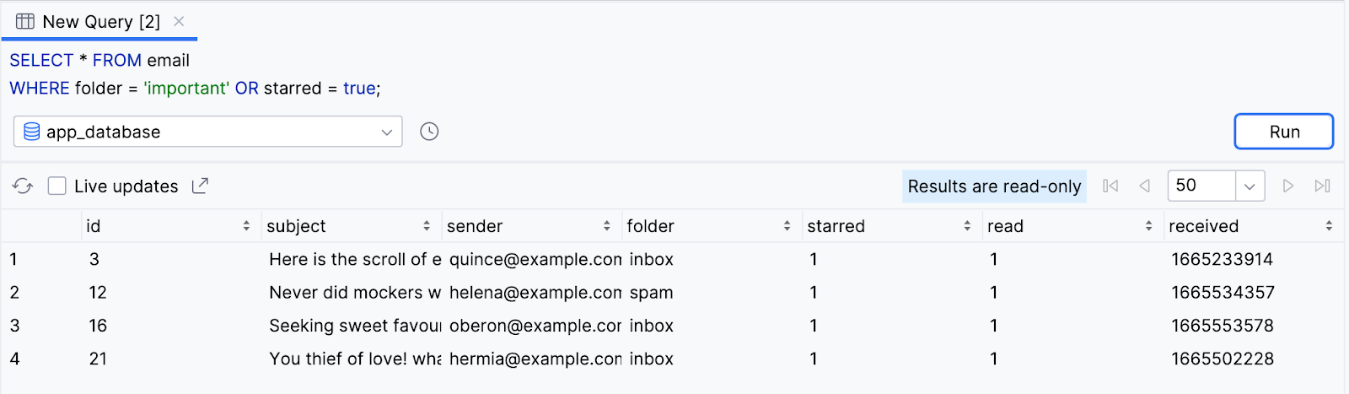
1. Observa que, después de ejecutar la consulta, los resultados solo contienen correos electrónicos no leídos en la bandeja Recibidos del usuario.



1. Devuelve todos los correos electrónicos incluidos en la carpeta **important** OR que tienen el valor **starred** (starred = true). Eso significa que los resultados incluirán correos electrónicos en diferentes carpetas, siempre que estén destacados.

SELECT \* FROM email  
WHERE folder = 'important' OR starred = true;

1. Observa el resultado.



**Nota:** También puedes escribir la condición de SQL NOT folder = 'spam' como folder != 'spam'.

Busca texto con LIKE

Una acción muy útil que puedes hacer con una cláusula WHERE es buscar texto en una columna específica. Puedes obtener este resultado cuando especificas un nombre de columna, seguido de la palabra clave LIKE y de una string de búsqueda.



La string de búsqueda comienza con el símbolo de porcentaje (%), seguido del texto que se desea buscar (Término de búsqueda) y, luego, el símbolo de porcentaje (%) de nuevo.



Si buscas un prefijo (los resultados que comienzan con el texto especificado), omite el primer símbolo de porcentaje (%).



De manera alternativa, si estás buscando un sufijo, omite el último símbolo de porcentaje (%).



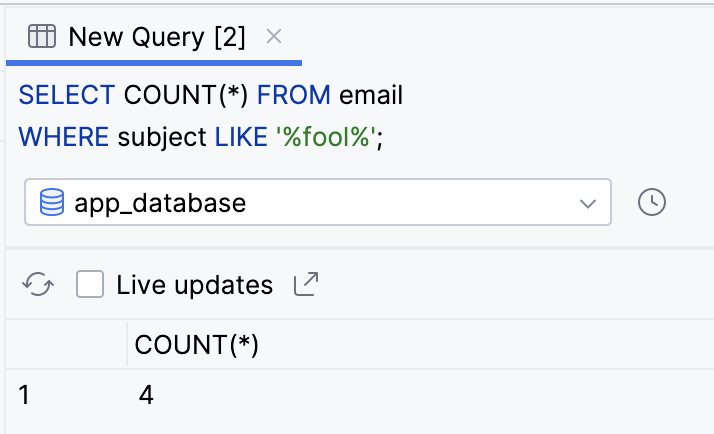
Hay muchos casos de uso en los que una app puede utilizar la búsqueda de texto, como buscar correos electrónicos con un texto en particular en el asunto o actualizar las sugerencias de autocompletar mientras el usuario escribe.

Las siguientes instrucciones te permiten usar la búsqueda de texto cuando consultas la tabla email.

1. A los personajes de Shakespeare, como los de nuestra base de datos, les encantaba hablar sobre los "fools" (bufones). Ejecuta la siguiente consulta para obtener la cantidad total de correos electrónicos con el texto "fool" en el asunto.

SELECT COUNT(\*) FROM email  
WHERE subject LIKE '%fool%';

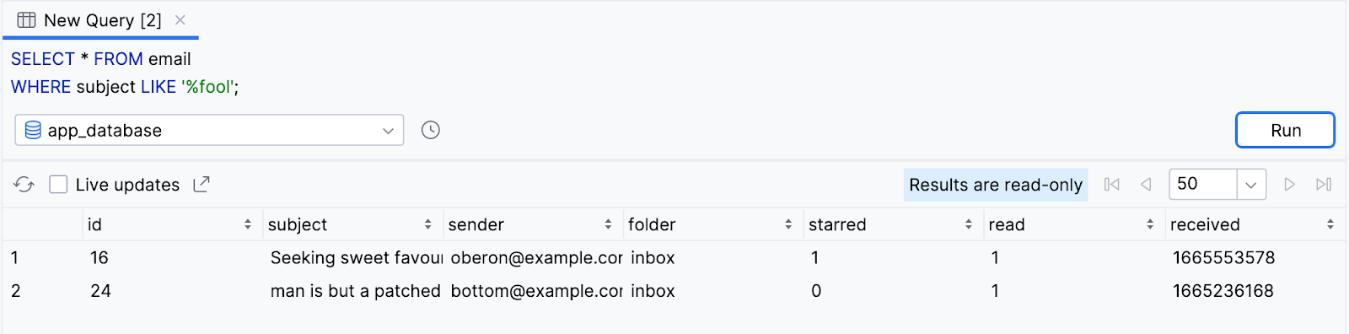
1. Observa el resultado.



1. Ejecuta la siguiente consulta para devolver todas las columnas de todas las filas en las que el asunto termina con la palabra "fool".

SELECT \* FROM email  
WHERE subject LIKE '%fool';

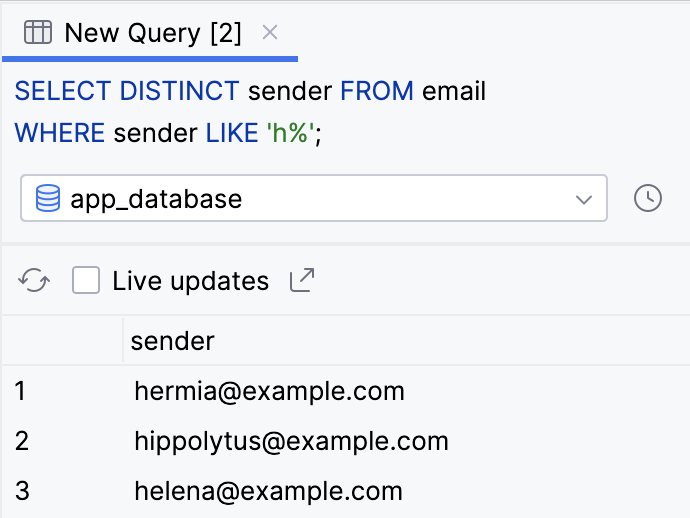
1. Observa que se muestran dos filas.



1. Ejecuta la siguiente consulta para devolver valores distintos de la columna sender que comienzan con la letra h.

SELECT DISTINCT sender FROM email  
WHERE sender LIKE 'h%';

1. Observa que la consulta muestra tres valores: helena@example.com, hyppolytus@example.com y hermia@example.com.



Agrupa los resultados con GROUP BY

Acabas de ver cómo usar las funciones de agregación y la cláusula WHERE para filtrar y reducir los resultados. SQL ofrece muchas otras cláusulas que pueden ayudarte a dar formato a los resultados de tu consulta. Entre estas cláusulas, se encuentran la agrupación, el ordenamiento y la limitación de resultados.

Puedes usar una cláusula GROUP BY para agrupar los resultados de modo que todas las filas que tengan el mismo valor para una columna determinada se agrupen una junto a la otra en los resultados. Esta cláusula no cambia los resultados, solo el orden en que se muestran.

Para agregar una cláusula GROUP BY a una instrucción SELECT, agrega la palabra clave GROUP BY seguida del nombre de la columna por la que deseas agrupar los resultados.



Un caso de uso común es combinar una cláusula GROUP BY con una función de agregación para particionar el resultado de esta función en diferentes buckets, como los valores de una columna. Por ejemplo: supongamos que deseas obtener la cantidad de correos electrónicos de cada carpeta: 'inbox', 'spam', etc. Puedes seleccionar la columna folder y la función de agregación COUNT(), y especificar la columna folder en la cláusula GROUP BY.

1. Realiza la siguiente consulta para seleccionar la columna de carpeta y el resultado de la función de agregación COUNT(). Usa una cláusula GROUP BY para agrupar los resultados por el valor de la columna folder.

SELECT folder, COUNT(\*) FROM email  
GROUP BY folder;

1. Observa los resultados. La consulta muestra la cantidad total de correos electrónicos de cada carpeta.



**Nota:** Puedes especificar varias columnas, separadas por una coma en la cláusula GROUP BY, si deseas separar cada grupo en subgrupos adicionales basados en una columna diferente.

Ordena los resultados con ORDER BY

También puedes cambiar el orden de los resultados de la consulta cuando los ordenas con la cláusula ORDER BY. Agrega la palabra clave ORDER BY seguida de un nombre de columna y de la dirección del ordenamiento.



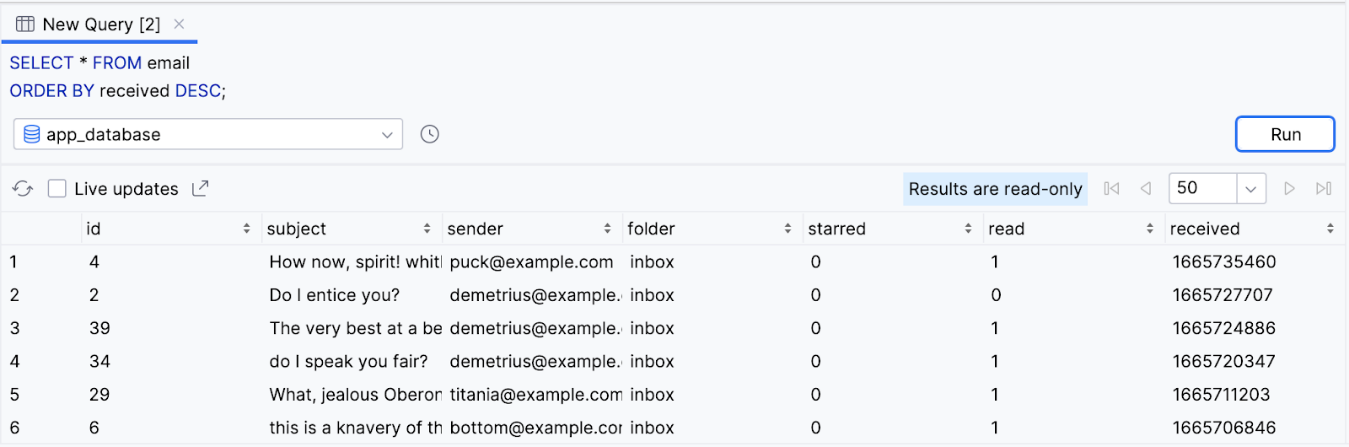
De forma predeterminada, la dirección de ordenamiento es ascendente, que puedes omitir en la cláusula ORDER BY. Si quieres ordenar los resultados en orden descendente, agrega DESC después del nombre de la columna.

Es probable que esperes que una app de correo electrónico muestre primero los correos electrónicos más recientes. Las siguientes instrucciones te permiten hacerlo con una cláusula ORDER BY.

1. Agrega una cláusula ORDER BY para ordenar los correos electrónicos no leídos, según la columna received. Como el orden ascendente (el más bajo o el más antiguo primero) es el predeterminado, debes usar la palabra clave DESC.

SELECT \* FROM email  
ORDER BY received DESC;

1. Observa el resultado.

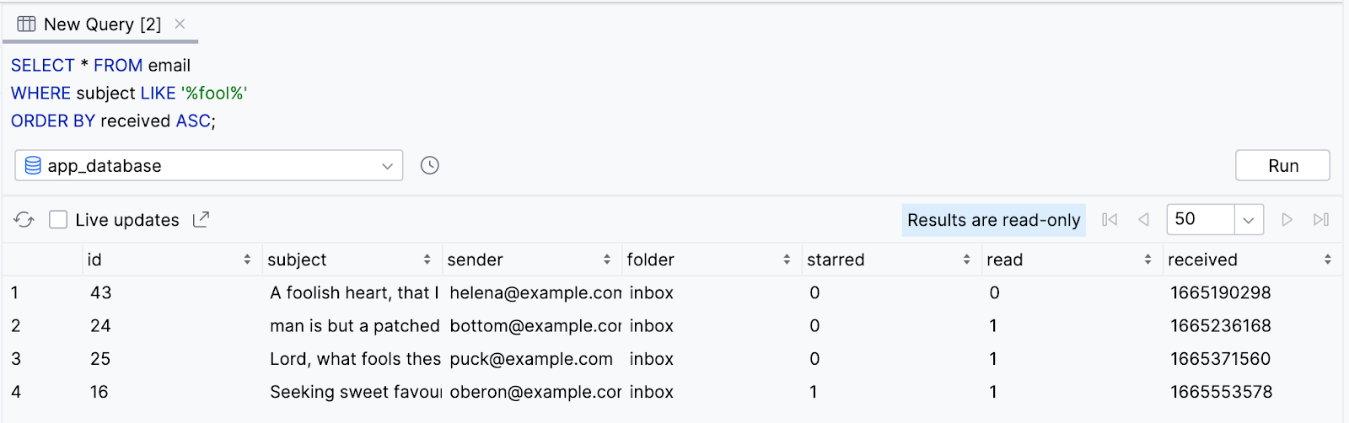


Puedes usar una cláusula ORDER BY con una cláusula WHERE. Supongamos que un usuario desea buscar correos electrónicos antiguos que contienen el texto *fool*. Puede ordenar los resultados para mostrar primero los correos electrónicos más antiguos, en orden ascendente.

1. Selecciona todos los correos electrónicos en los que el asunto incluya el texto "fool" y ordena los resultados de forma ascendente. Como el orden es ascendente, que es el predeterminado cuando no se especifica ninguno, usar la palabra clave ASC con la cláusula ORDER BY es opcional.

SELECT \* FROM email  
WHERE subject LIKE '%fool%'  
ORDER BY received ASC;

1. Observa que los resultados filtrados se devuelven primero con el valor más antiguo (el valor más bajo en la columna de recibidos).



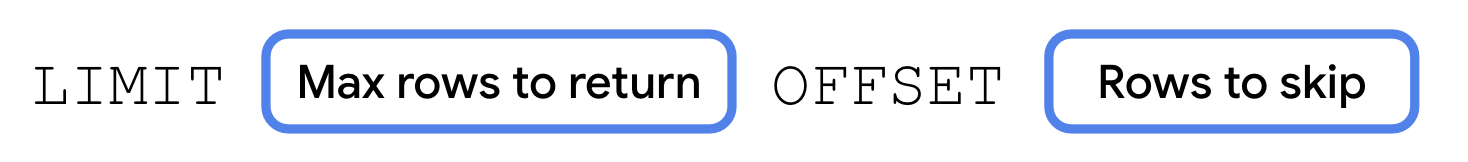
**Nota:** Si ambos se usan en la misma consulta, la cláusula GROUP BY antecede a la cláusula ORDER BY.

Restringe el número de resultados con LIMIT

Hasta ahora, todos los ejemplos muestran cada resultado único de la base de datos que coincide con la consulta. En muchos casos, solo necesitas mostrar una cantidad limitada de filas de tu base de datos. Puedes agregar una cláusula LIMIT a tu consulta para mostrar solo una cantidad específica de resultados. Agrega la palabra clave LIMIT seguida de la cantidad máxima de filas que deseas mostrar. Si corresponde, la cláusula LIMIT viene después de la cláusula ORDER BY.



De forma opcional, puedes incluir la palabra clave OFFSET seguida de otro número para el número de las filas que se "omitirán". Por ejemplo, si quieres los diez resultados siguientes, después de los primeros diez, pero no quieres que se muestren los veinte resultados, puedes usar LIMIT 10 OFFSET 10.

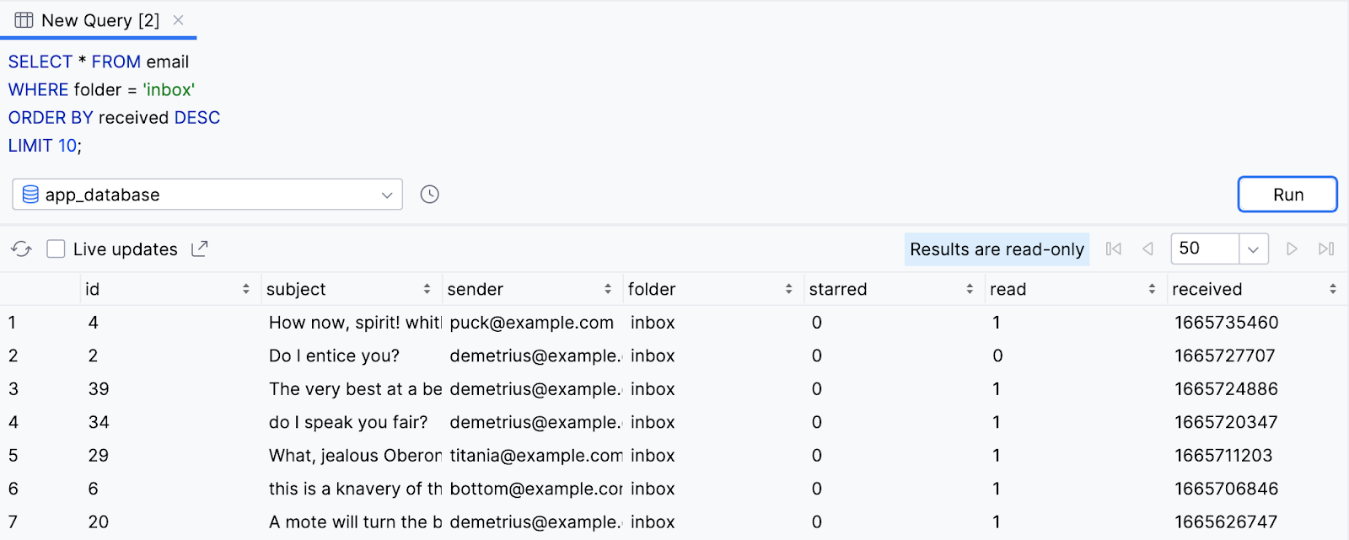


En una app, es posible que desees cargar los correos electrónicos más rápido mostrando solo los primeros diez en Recibidos del usuario. Luego, los usuarios pueden desplazarse para ver las siguientes páginas de correos electrónicos. En las siguientes instrucciones, usa una cláusula LIMIT para lograr este comportamiento.

1. Realiza la siguiente instrucción SELECT para obtener todos los correos electrónicos de la carpeta Recibidos del usuario en orden descendente y limitados a los primeros diez resultados.

SELECT \* FROM email  
WHERE folder = 'inbox'  
ORDER BY received DESC  
LIMIT 10;

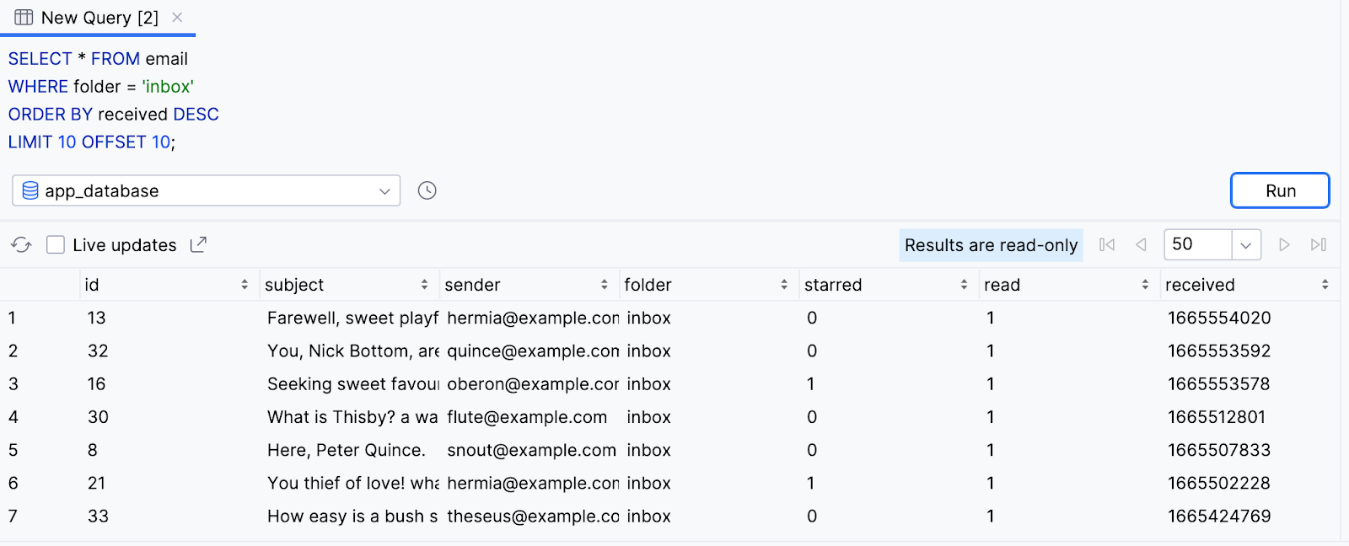
1. Observa que solo se devuelven diez resultados.



1. Modifica y vuelve a ejecutar la consulta para incluir la palabra clave OFFSET con un valor de 10.

SELECT \* FROM email  
WHERE folder = 'inbox'  
ORDER BY received DESC  
LIMIT 10 OFFSET 10;

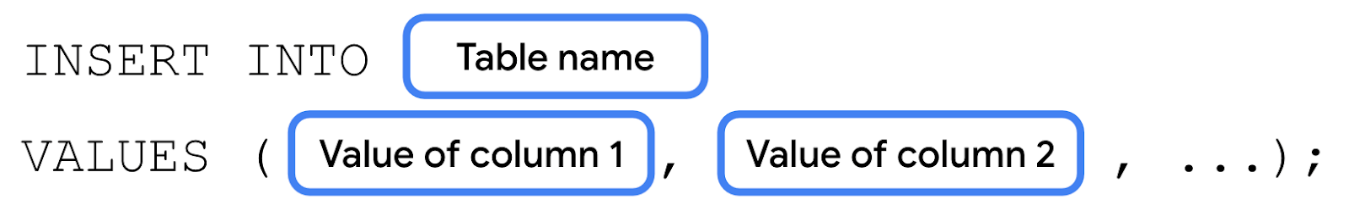
1. La consulta muestra diez resultados en orden descendente. Sin embargo, la consulta omite el primer conjunto de diez resultados.



Inserta datos en una base de datos

Además de leer desde una base de datos, hay diferentes instrucciones de SQL para escribir en una base de datos. Los datos necesitan una forma de llegar allí, ¿no?

Puedes agregar una fila nueva a una base de datos con una instrucción INSERT. Una instrucción INSERT comienza con INSERT INTO seguido del nombre de la tabla en la que deseas insertar una fila nueva. La palabra clave VALUES aparece en una línea nueva seguida de un conjunto de paréntesis que contienen una lista de valores separados por comas. Debes enumerar los valores en el mismo orden que las columnas de la base de datos.



Supongamos que el usuario recibe un correo electrónico nuevo y que debemos almacenarlo en la base de datos de nuestra app. Podemos usar una instrucción INSERT para agregar una fila nueva a la tabla email.

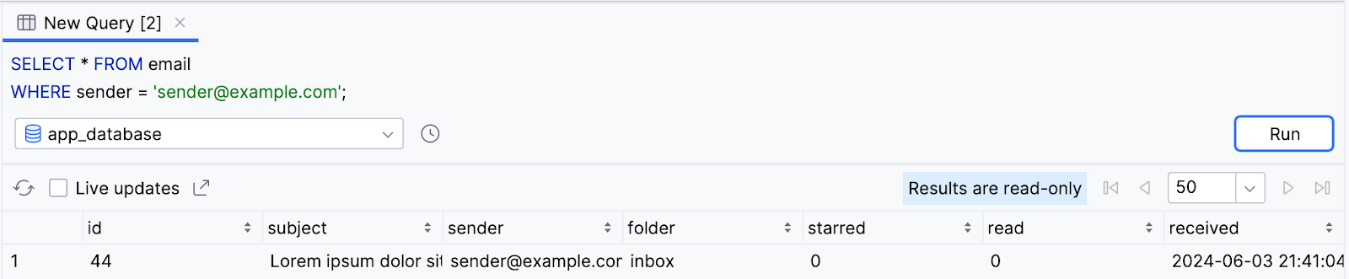
1. Realiza una instrucción INSERT con los siguientes datos para un correo electrónico nuevo. Como el correo electrónico es nuevo, no está leído y en un principio aparece en folder de la bandeja de entrada. Se proporciona un valor de NULL para la columna id, lo que significa que id se generará automáticamente con el siguiente número entero disponible que se incremente automáticamente.

INSERT INTO email  
VALUES (  
    NULL, 'Lorem ipsum dolor sit amet', 'sender@example.com', 'inbox', false, false, CURRENT\_TIMESTAMP  
);

**Nota:** CURRENT\_TIMESTAMP es una variable especial que se reemplaza por la hora actual en [UTC](https://en.wikipedia.org/wiki/Coordinated_Universal_Time) cuando se ejecuta la consulta, lo cual es conveniente para insertar filas nuevas.

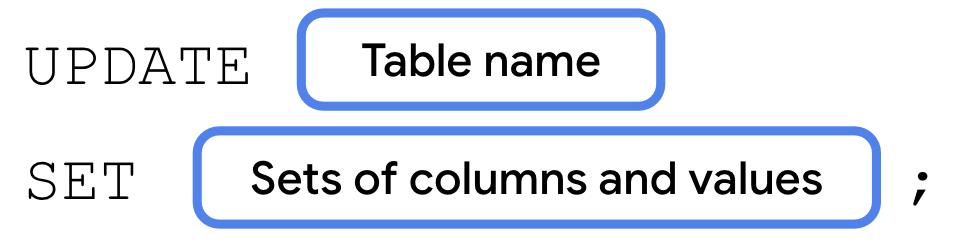
1. Observa que el resultado se inserta en la base de datos con un id de 44.

SELECT \* FROM email  
WHERE sender = 'sender@example.com';

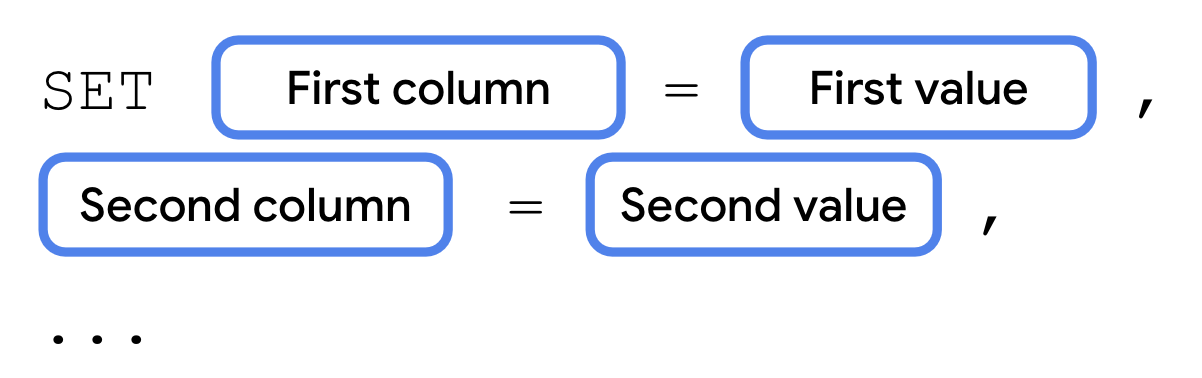


Actualiza los datos existentes en una base de datos

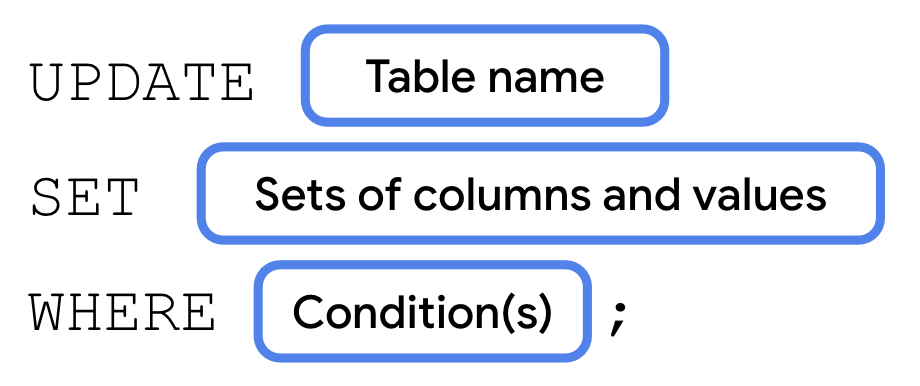
Después de insertar datos en una tabla, podrás cambiarlos más adelante. Puedes actualizar el valor de una o más columnas con una instrucción UPDATE. Una instrucción UPDATE comienza con la palabra clave UPDATE, seguida del nombre de la tabla y una cláusula SET.



Una cláusula SET consta de la palabra clave SET, seguida del nombre de la columna que deseas actualizar.



Una instrucción UPDATE a menudo incluye una cláusula WHERE para especificar la fila o varias filas que deseas actualizar con el par columna-valor especificado.



Si el usuario desea marcar un correo electrónico como leído, por ejemplo, usa una instrucción UPDATE para actualizar la base de datos. Las siguientes instrucciones te permiten marcar el correo electrónico insertado en el paso anterior como leído.

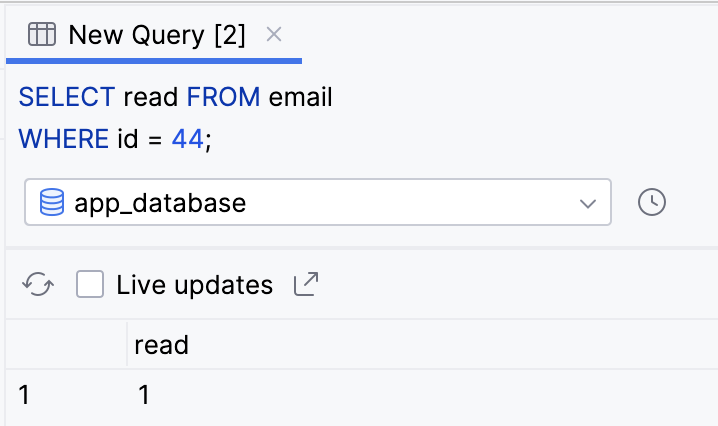
1. Realiza la siguiente instrucción UPDATE para establecer la fila con un id de 44 para que el valor de la columna read sea true.

UPDATE email  
SET read = true  
WHERE id = 44;

1. Ejecuta una sentencia SELECT para esa fila específica para validar el resultado.

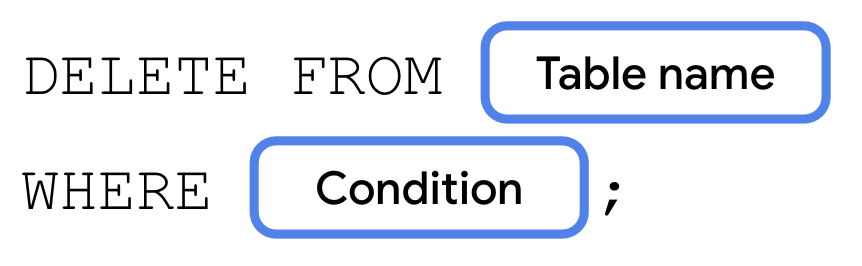
SELECT read FROM email  
WHERE id = 44;

1. Observa que el valor de la columna de leídos ahora es 1 para un valor "true" y 0 para "false".



Borra una fila de una base de datos

Por último, puedes usar una instrucción DELETE de SQL para borrar una o más filas de una tabla. Una instrucción DELETE comienza con la palabra clave DELETE, seguida de la palabra clave FROM, seguida del nombre de la tabla y, luego, una cláusula WHERE para especificar qué fila o filas deseas borrar.



En las siguientes instrucciones, se usa una sentencia DELETE para borrar de la base de datos la fila que se insertó y actualizó posteriormente.

1. Realiza la siguiente instrucción DELETE para borrar la fila con un id de 44 de la base de datos.

DELETE FROM email  
WHERE id = 44;

1. Valida tus cambios con una instrucción SELECT.

SELECT \* FROM email  
WHERE id = 44;

1. Observa que una fila con un id de 44 ya no existe.

