

## DOCUMENTACIÓN

Descripción general

Guías

Referencia

Ejemplos

Diseño y calidad

Cómo guardar contenido en el almacenamiento específico de la app

Cómo guardar contenido en el almacenamiento compartido

Cómo administrar todos los archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

Cómo guardar contenido en una base de datos local

Descripción general

Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre objetos

Escribe consultas DAO asíncronas

Cómo implementar vistas en una base de datos

Cómo autocompletar el contenido de tu base de datos

Cómo migrar tu base de datos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

Cómo guardar contenido en una base de datos local

Descripción general

Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre objetos

Escribe consultas DAO asíncronas

Cómo implementar vistas en una base de datos

Cómo autocompletar el contenido de tu base de datos

Cómo migrar tu base de datos

Cómo probar y depurar tu base de datos

Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

Cómo guardar contenido en una base de datos local

Descripción general

Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre objetos

Escribe consultas DAO asíncronas

Cómo implementar vistas en una base de datos

Cómo autocompletar el contenido de tu base de datos

Cómo migrar tu base de datos

Cómo probar y depurar tu base de datos

Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

Cómo guardar contenido en una base de datos local

Descripción general

Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre

Desarrolladores de Android &gt; Documentos &gt; Guías

¿Te resultó útil?

## Cómo definir datos con entidades Room

Cuando usas la biblioteca de persistencias de Room para almacenar los datos de tu app, defines entidades para representar los objetos que deseas almacenar. Cada entidad corresponde a una tabla en la base de datos de Room asociada y cada instancia de una entidad representa una fila de datos en la tabla correspondiente.

Eso significa que puedes usar las entidades de Room para definir tu [esquema de base de datos](#) sin escribir ningún código de SQL.

### Anatomía de una entidad

Define cada entidad de Room como una clase con `@Entity` como anotación. Una entidad de Room incluye campos para cada columna de la tabla correspondiente en la base de datos, incluidas una o más columnas que conforman la [clave primaria](#).

El siguiente código es un ejemplo de una entidad simple que define una tabla `User` con columnas para el ID, el nombre y el apellido:

Kotlin

```
@Entity
data class User(
    @PrimaryKey val id: Int,
    val firstName: String?,
    val lastName: String?
)
```



**Nota:** Para conservar un campo, Room debe tener acceso a él. Para asegurarte de que Room tenga acceso a un campo, hazlo público o proporciona métodos get y set para él.

De forma predeterminada, Room usa el nombre de la clase como el nombre de la tabla de la base de datos. Si quieres que la tabla tenga un nombre diferente, configura la propiedad `tableName` de la anotación `@Entity`. De manera similar, Room usa los nombres de campos como nombres de columna en la base de datos de forma predeterminada. Si quieres que una columna tenga un nombre diferente, agrega la anotación `@ColumnInfo` al campo y establece la propiedad `name`. En el siguiente ejemplo, se muestran los nombres personalizados para tablas y columnas:

Kotlin

```
@Entity(tableName = "users")
data class User (
    @PrimaryKey val id: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```



**Nota:** Los nombres de tablas y columnas en SQLite no distinguen mayúsculas de minúsculas.

### Cómo definir una clave primaria

Cada entidad de Room debe definir una [clave primaria](#) que identifique de manera única cada fila en la tabla de base de datos correspondiente. La manera más sencilla de hacerlo es anotar una sola columna con `@PrimaryKey`:

Kotlin

```
@PrimaryKey val id: Int
```



**Nota:** Si necesitas que Room asigne ID automáticos a instancias de entidades, configura la propiedad `autoGenerate` de `@PrimaryKey` en `true`.

### Cómo definir una clave primaria compuesta

Si necesitas que las instancias de una entidad se identifiquen de forma única mediante una combinación de varias columnas, puedes definir una [clave primaria compuesta](#) si enumeras esas columnas en la propiedad `primaryKey` de `@Entity`:

Kotlin

```
@Entity(primaryKeys = ["firstName", "lastName"])
data class User(
    val firstName: String?,
    val lastName: String?
)
```



### Cómo ignorar campos

De forma predeterminada, Room crea una columna para cada campo que se define en la entidad. Si una entidad tiene

#### En esta página

[Anatomía de una entidad](#)[Cómo definir una clave primaria](#)[Cómo definir una clave primaria compuesta](#)[Cómo ignorar campos](#)[Cómo proporcionar compatibilidad con la búsqueda de tablas](#)[Cómo admitir la búsqueda en el texto completo](#)[Columnas específicas del índice](#)[Cómo incluir objetos basados en AutoValue](#)

#### Recommendations

[Cómo implementar vistas en una base de datos](#)Updated 25 ene 2022[Cómo propagar la base de datos de Room](#)Updated 25 ene 2022[Cómo acceder a los datos con DAO de Room](#)Updated 25 ene 2022

objetos  
Escribe consultas DAO  
asíncronas  
Cómo implementar vistas en una base de datos  
Cómo autocompletar el contenido de tu base de datos  
Cómo migrar tu base de datos  
Cómo probar y depurar tu base de datos  
Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

• Cómo guardar contenido en una base de datos local

Descripción general

#### Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre objetos

Escribe consultas DAO asíncronas

Cómo implementar vistas en una base de datos

Cómo autocompletar el contenido de tu base de datos

Cómo migrar tu base de datos

Cómo probar y depurar tu base de datos

Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

• Cómo guardar contenido en una base de datos local

Descripción general

#### Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre objetos

Escribe consultas DAO asíncronas

Cómo implementar vistas en una base de datos

Cómo autocompletar el contenido de tu base de datos

Cómo migrar tu base de datos

Cómo probar y depurar tu base de datos

Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

• Cómo guardar contenido en una base de datos local

Descripción general

#### Cómo definir datos mediante entidades

Cómo acceder a datos mediante DAO

Cómo definir relaciones entre objetos

Escribe consultas DAO asíncronas

Cómo implementar vistas en una base de datos

Cómo autocompletar el contenido de tu base de datos

Cómo migrar tu base de datos

Cómo probar y depurar tu base de datos

Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

• Cómo guardar contenido en una base de datos local

campos no deseados, puedes usar la anotación `@Ignore` en ellos, como se muestra en el siguiente fragmento de código:

```
Kotlin Java
@Entity
data class User(
    @PrimaryKey val id: Int,
    val firstName: String?,
    val lastName: String?,
    @Ignore val picture: Bitmap?
)
```

Cuando una entidad hereda campos de una entidad principal, suele ser más fácil usar la propiedad `ignoredColumns` del atributo `@Entity`:

```
Kotlin Java
open class User (
    var picture: Bitmap? = null
)

@Entity(ignoredColumns = ["picture"])
data class RemoteUser(
    @PrimaryKey val id: Int,
    val hasVpn: Boolean
) : User()
```

## Cómo proporcionar compatibilidad con la búsqueda de tablas

Room admite varios tipos de anotaciones que facilitan la búsqueda de detalles en las tablas de la base de datos. Usa la búsqueda en el texto completo, a menos que la `minSdkVersion` de tu app sea inferior a 16.

### Cómo admitir la búsqueda en el texto completo

Si la app requiere un acceso muy rápido a la información de la base de datos mediante la búsqueda en el texto completo (FTS), respalda tus entidades con una tabla virtual que use el [módulo de extensión SQLite](#) FTS3 o FTS4. Para usar esta función, disponible en Room 2.1.0 y versiones posteriores, agrega la anotación `@Fts3` o `@Fts4` a una entidad dada, como se muestra en el siguiente fragmento de código:

```
Kotlin Java
// Use '@Fts3' only if your app has strict disk space requirements or if you
// require compatibility with an older SQLite version.
@Fts4
@Entity(tableName = "users")
data class User(
    /* Specifying a primary key for an FTS-table-backed entity is optional, but
     * if you include one, it must use this type and column name. */
    @PrimaryKey @ColumnInfo(name = "rowid") val id: Int,
    @ColumnInfo(name = "first_name") val firstName: String?
)
```

★ Nota: Las tablas habilitadas para la FTS siempre usan una clave primaria de tipo `INTEGER` y el nombre de columna "rowid". Si la entidad respaldada por tablas de FTS define una clave primaria, **debe** usar ese tipo y nombre de columna.

Cuando una tabla admite contenido en varios idiomas, usa la opción `languageId` para especificar la columna que almacena la información de idioma de cada fila:

```
Kotlin Java
@Fts4(languageId = "lid")
@Entity(tableName = "users")
data class User(
    ...
    @ColumnInfo(name = "lid") val languageId: Int
)
```

Room ofrece varias opciones para definir entidades respaldadas por FTS, entre ellas el orden de los resultados, los tipos de tokenizadores y las tablas que se administran como contenido externo. Para obtener más información sobre estas opciones, consulta la referencia de [FtsOptions](#).

### Columnas específicas del índice

Si la app debe admitir versiones de SDK que no permiten el uso de entidades respaldadas por tablas FTS3 o FTS4, igual puedes indexar algunas columnas de la base de datos para agilizar las búsquedas. Para agregar índices a una entidad, incluye la propiedad `indices` dentro de la anotación `@Entity` y enumera los nombres de las columnas que quieras incluir en el índice o en el índice compuesto. En el siguiente fragmento de código, se muestra este proceso de anotación:

```
Kotlin Java
@Entity(indices = [Index(value = ["last_name", "address"])])
data class User(
    @PrimaryKey val id: Int,
    val firstName: String?,
    val address: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @Ignore val picture: Bitmap?
)
```

A veces, ciertos campos o grupos de campos de una base de datos deben ser únicos. Puedes aplicar esta propiedad de exclusividad con la propiedad `unique` de una anotación `@Index` como `true`. En la siguiente muestra de código, se evita que una tabla tenga dos filas con el mismo conjunto de valores para las columnas `firstName` y `lastName`:

## Descripción general

### Cómo definir datos mediante entidades

- Cómo acceder a datos mediante DAO
- Cómo definir relaciones entre objetos
- Escribe consultas DAO asíncronas
- Cómo implementar vistas en una base de datos
- Cómo autocompletar el contenido de tu base de datos
- Cómo migrar tu base de datos
- Cómo probar y depurar tu base de datos
- Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

- Como guardar contenido en una base de datos local

Descripción general

### Cómo definir datos mediante entidades

- Cómo acceder a datos mediante DAO
- Cómo definir relaciones entre objetos
- Escribe consultas DAO asíncronas
- Cómo implementar vistas en una base de datos
- Cómo autocompletar el contenido de tu base de datos
- Cómo migrar tu base de datos
- Cómo probar y depurar tu base de datos
- Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

- Como guardar contenido en una base de datos local

Descripción general

### Cómo definir datos mediante entidades

- Cómo acceder a datos mediante DAO
- Cómo definir relaciones entre objetos
- Escribe consultas DAO asíncronas
- Cómo implementar vistas en una base de datos
- Cómo autocompletar el contenido de tu base de datos
- Cómo migrar tu base de datos
- Cómo probar y depurar tu base de datos
- Cómo hacer referencia a datos complejos



archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

- Como guardar contenido en una base de datos local

Accesos directos generales  
archivos de un dispositivo de almacenamiento

Cómo guardar datos de pares clave-valor

- Como guardar contenido en una base de datos local

Descripción general

### Cómo definir datos mediante entidades

- Cómo acceder a datos mediante DAO
- Cómo definir relaciones entre objetos
- Escribe consultas DAO asíncronas
- Cómo implementar vistas en una base de datos
- Cómo autocompletar el contenido de tu base de datos

## Kotlin Java

```
@Entity(indices = [Index(value = ["first_name", "last_name"], unique = true)])
data class User(
    @PrimaryKey val id: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @Ignore var picture: Bitmap?
)
```

## Cómo incluir objetos basados en AutoValue

★ Nota: La función está diseñada para usarse solo en entidades basadas en Java. Para lograr la misma funcionalidad en las entidades basadas en Kotlin, es mejor usar [clases de datos](#).

En Room 2.1.0 y versiones posteriores, puedes usar [clases de valores inmutables](#) basadas en Java, que puedes anotar con `@AutoValue`, como entidades en la base de datos de la app. La compatibilidad es particularmente útil cuando dos instancias de una entidad se consideran iguales si sus columnas tienen valores idénticos.

Cuando usas clases anotadas con `@AutoValue` como entidades, puedes anotar los métodos abstractos de la clase con `@PrimaryKey`, `@ColumnInfo`, `@Embedded` y `@Relation`. Sin embargo, si las usas, debes incluir la anotación `@CopyAnnotations` cada vez, para que Room pueda interpretar correctamente las implementaciones autogeneradas de los métodos.

En el siguiente fragmento de código, se muestra un ejemplo de una clase anotada con `@AutoValue` que Room reconoce como una entidad:

User.java

```
@AutoValue
@Entity
public abstract class User {
    // Supported annotations must include `@CopyAnnotations`.
    @CopyAnnotations
    @PrimaryKey
    public abstract long getId();

    public abstract String getFirstName();
    public abstract String getLastName();

    // Room uses this factory method to create User objects.
    public static User create(long id, String firstName, String lastName) {
        return new AutoValue_User(id, firstName, lastName);
    }
}
```



Cómo migrar tu base de datos

Cómo probar y depurar tu  
base de datos

Cómo hacer referencia a  
datos complejos

Google Developers

Android

Chrome

Firebase

Google Cloud Platform

Todos los productos

Privacidad | Licencia | Lineamientos de marca

Recibe noticias y sugerencias por correo electrónico

Suscribirse

Español - A...