The Spring Data Cloud Spanner module helps you use Cloud Spanner in any Java application that's built with the Spring Framework (https://spring.io/projects/spring-framework).

Like all Spring Data (https://spring.io/projects/spring-data) modules, Spring Data Cloud Spanner provides a Spring-based programming model that retains the consistency guarantees and scalability of Cloud Spanner. Its features are similar to Spring Data JPA (https://spring.io/projects/spring-data-jpa) and Hibernate ORM (https://hibernate.org/orm/), with annotations designed for Cloud Spanner.

If you're already familiar with Spring, then Spring Data Cloud Spanner can make it easier to work with Cloud Spanner in your application and reduce the amount of code that you need to write.

This page explains how to add Spring Data Cloud Spanner to a Java application. For detailed information about the module, see the Spring Data Cloud Spanner reference (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#spring-data-cloud-spanner)
.

If you use Maven, add the Spring Cloud GCP Bill of Materials (BOM) (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#bill-of-materials)
and Spring Data Cloud Spanner to your `pom.xml` file. These dependencies provide the Spring Data Cloud Spanner components to your Spring `ApplicationContext` (https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/ApplicationContext.html)
:

spanner/spring-data/pom.xml (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/pom.xml)

(https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/pom.xml)

You must also create a service account (/docs/authentication/getting-started) and use the service account key to authenticate with Google Cloud.

For more information, see the instructions for setting up a Java development environment (/java/docs/setup). You do not need to install the Google Cloud Client Library for Java; the Spring Boot starter installs the client library automatically.

This section describes some of the most commonly used configuration settings for Spring Data Cloud Spanner. For a complete list of settings, see the reference documentation (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#cloud-spanner-settings)
.

You must specify the Cloud Spanner instance and database that your application connects to.

To specify the default instance and database, set the following configuration properties for your application:

Property	Description
`spring.cloud.gcp.spanner.project-id`	Optional. The Google Cloud project ID. Overrides the value of `spring.cloud.gcp.config.project-id`.
`spring.cloud.gcp.spanner.instance-id`	The Cloud Spanner instance ID.
`spring.cloud.gcp.spanner.database`	The database to connect to.

With Spring Data Cloud Spanner, you can use plain old Java objects (POJOs) to model the data you store in your Cloud Spanner tables.

For each table in your database, declare an entity that represents a record in that table. Use annotations to map the entity and its properties to a table and its columns.

You can use the following annotations to model simple relationships between entities and tables:

Entity annotations	
`@Column(name = "columnName")`	Optional. Maps the property to a specific column in the Cloud Spanner table, overriding the naming strategy that automatically maps the names. When you omit this property, the default naming strategy for Spring Data Cloud Spanner maps Java `camelCase` property names to `PascalCase` column names. For example, the property `singerId` maps to the column name `SingerId`.

Entity annotations	
`@Embedded`	Indicates that the property is an embedded object that can hold components of a primary key. If the property is actually used in the primary key, you must also include the `@PrimaryKey` annotation.

`@Interleaved`
 	
`@Interleaved(lazy = true)`
	Indicates that a property contains a list of rows that are interleaved (/spanner/docs/schema-and-data-model#creating-interleaved-tables) with the current row.
	
	By default, Spring Data Cloud Spanner fetches the interleaved rows at instance creation. To fetch the rows lazily, when you access the property, use `@Interleaved(lazy = true)`.
	
	Example: If a `Singer` entity can have interleaved `Album` entries as children, add a `List<Album>` property to the `Singer` entity. Also, add an `@Interleaved` annotation to the property.

`@NotMapped`	Indicates that a property is not stored in the database and should be ignored.

`@PrimaryKey`
 	
`@PrimaryKey(keyOrder = N)`
	Indicates that the property is a component of the primary key, and identifies the position of the property within the primary key, starting at 1. The default `keyOrder` is `1`.
	
	Example: `@PrimaryKey(keyOrder = 3)`

`@Table(name = "TABLE_NAME")`	The table that the entity models. Each instance of the entity represents a record in the table. Replace ***TABLE_NAME*** with the name of your table. Example: `@Table(name = "Singers")`

If you need to model more complex relationships, see the Spring Data Cloud Spanner reference (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#spring-data-cloud-spanner) for details about other annotations that the module supports.

The following examples show one way to model the `Singers` and `Albums` tables for Spring Data Cloud Spanner:

- For `Singer` entities, the example includes an `albums` property, with an `@Interleaved` annotation. This property contains a list of albums that are interleaved with the `Singer` entity. Spring Data Cloud Spanner populates this property automatically.
- For `Album` entities, the example includes a `relatedAlbums` property that is not stored in Cloud Spanner.

spanner/spring-data/src/main/java/com/example/spanner/Singer.java (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/Singer.java)

n/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/Singer.java)

spanner/spring-data/src/main/java/com/example/spanner/Album.java (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/Album.java)

n/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/Album.java)

To query and modify data with Spring Data Cloud Spanner, you can acquire a `SpannerTemplate` (https://github.com/spring-cloud/spring-cloud-gcp/blob/master/spring-cloud-gcp-data-spanner/src/main/java/org/springframework/cloud/gcp/data/spanner/core/SpannerTemplate.java)

bean, which implements `SpannerOperations` (https://github.com/spring-cloud/spring-cloud-gcp/blob/master/spring-cloud-gcp-data-spanner/src/main/java/org/springframework/cloud/gcp/data/spanner/core/SpannerOperations.java). `SpannerTemplate` provides methods for performing SQL queries (/spanner/docs/query-syntax) and modifying data with Data Manipulation Language (DML) statements (/spanner/docs/dml-tasks). You can also use this bean to access the read API (/spanner/docs/reads) and mutation API (/spanner/docs/modify-mutation-api) for Cloud Spanner.

In addition, you can extend the `SpannerRepository` (https://github.com/spring-cloud/spring-cloud-gcp/blob/master/spring-cloud-gcp-data-spanner/src/main/java/org/springframework/cloud/gcp/data/spanner/repository/SpannerRepository.java) interface to encapsulate all of the application logic that queries and modifies data in Cloud Spanner.

The following sections explain how to work with `SpannerTemplate` and `SpannerRepository`.

Use the `@Autowired` annotation to acquire a `SpannerTemplate` bean automatically. You can then use the `SpannerTemplate` throughout your class.

The following example shows a class that acquires and uses the bean:

spanner/spring-data/src/main/java/com/example/spanner/SpannerTemplateSample.java (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SpannerTemplateSample.java)

es/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SpannerTemplateSample.java)

You can use the `SpannerTemplate` bean to execute read-only transactions (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#read-write-transaction) and read-write transactions (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#read-only-transaction). In addition, you can use the `@Transactional` (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#declarative-transactions-with-transactional-annotation) annotation to create declarative transactions.

If you use a `SpannerRepository`, you can use the `@Autowired` annotation to acquire a bean that implements your repository's interface. A repository includes methods for running Java functions as read-only transactions (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#read-write-transaction) and read-write transactions

(https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#read-only-transaction)
. For lower-level operations, you can get the template bean that the repository uses.

The following examples show the interface for a repository and a class that acquires and uses the bean:

spanner/spring-data/src/main/java/com/example/spanner/SingerRepository.java
(https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SingerRepository.java)

s-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SingerRepository.java)

spanner/spring-data/src/main/java/com/example/spanner/SpannerRepositorySample.java
(https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SpannerRepositorySample.java)

s/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SpannerRepositorySample.java)

To get information about your Cloud Spanner databases, update a schema with a Data Definition Language (DDL) statement, or complete other administrative tasks, you can acquire a `SpannerDatabaseAdminTemplate` (https://github.com/spring-cloud/spring-cloud-gcp/blob/master/spring-cloud-gcp-data-spanner/src/main/java/org/springframework/cloud/gcp/data/spanner/core/admin/SpannerDatabaseAdminTemplate.java) bean.

Use the `@Autowired` annotation to acquire the bean automatically. You can then use the `SpannerDatabaseAdminTemplate` throughout your class.

The following example shows a class that acquires and uses the bean:

spanner/spring-data/src/main/java/com/example/spanner/SpannerSchemaToolsSample.java (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/spanner/spring-data/src/main/java/com/example/spanner/SpannerSchemaToolsSample.java)

```
lob/master/spanner/spring-data/src/main/java/com/example/spanner/SpannerSchemaToolsSample.java)
```

- Get started with Spring Cloud GCP (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#getting-started).
- Learn more about using Spring Data Cloud Spanner in your applications (https://cloud.spring.io/spring-cloud-static/spring-cloud-gcp/1.2.0.RELEASE/reference/html/#spring-data-cloud-spanner).
- File a GitHub issue (https://github.com/spring-cloud/spring-cloud-gcp/issues) to report a bug or ask a question about the module.
- Get more information about Spring Framework support on Google Cloud (/java/docs/reference/spring).
- Try a codelab to deploy and run an application that uses Spring Cloud GCP (https://codelabs.developers.google.com/spring/).