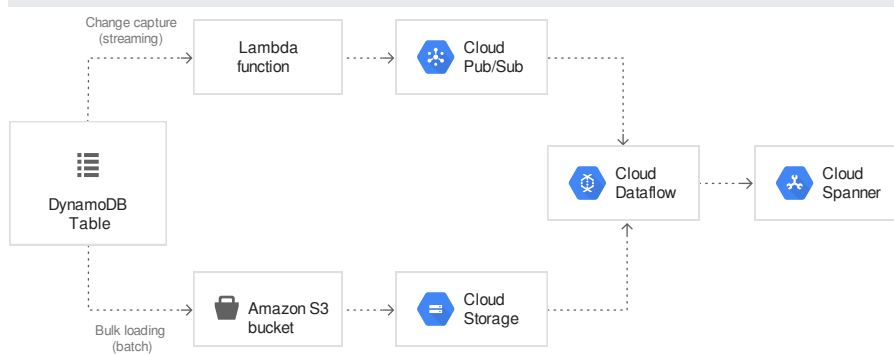


This tutorial describes how to migrate from Amazon DynamoDB to [Spanner](/spanner/). It is primarily intended for app owners who want to move from a NoSQL system to Spanner, a fully relational, fault-tolerant, highly scalable SQL database system that supports transactions. If you have consistent Amazon DynamoDB table usage, in terms of types and layout, mapping to Spanner is straightforward. If your Amazon DynamoDB tables contain arbitrary data types and values, it might be simpler to move to other NoSQL services, such as [Datastore](/datastore/) or [Firebase](https://firebase.google.com/).

This tutorial assumes that you are familiar with database schemas, data types, the fundamentals of NoSQL, and relational database systems. The tutorial relies on running predefined tasks to perform an example migration. After the tutorial, you can modify the provided code and steps to [match your environment](#) (#adjusting_for_your_data_model).

The following architectural diagram outlines the components used in the tutorial to migrate data:



- Migrate data from Amazon DynamoDB to Spanner.
- Create a Spanner database and migration table.
- Map a NoSQL schema to a relational schema.
- Create and export a sample dataset that uses Amazon DynamoDB.
- Transfer data between Amazon S3 and Cloud Storage.
- Use Dataflow to load data into Spanner.

This tutorial uses the following billable components of Google Cloud:

- GKE
- Pub/Sub
- Cloud Storage
- Dataflow

Spanner charges are based on the number of node-hours and the amount of data stored during the monthly billing cycle. During the tutorial, you use a minimal configuration of these resources, which are [cleaned up at the end](#) (#clean-up). For real-world scenarios, estimate your throughput and storage requirements, and then use the [Spanner instances documentation](/spanner/docs/instances/) to determine the number of nodes that you need.

In addition to Google Cloud resources, this tutorial uses the following Amazon Web Services (AWS) resources:

- Amazon EMR
- AWS Lambda
- Amazon S3
- Amazon DynamoDB

These services are only needed during the migration process. At the end of the tutorial, follow the instructions to clean up all resources to prevent unnecessary charges. Use the [AWS pricing calculator](https://calculator.s3.amazonaws.com/index.html) to estimate these costs.

To generate a cost estimate based on your projected usage, use the [pricing calculator](/products/calculator). New Google Cloud users might be eligible for a [free trial](/free-trial).

1. [Sign in](https://accounts.google.com/Login) to your Google Account.

If you don't already have one, [sign up for a new account](https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note:** If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

[Go to the project selector page](https://console.cloud.google.com/projectselector2/home/dashboard)

3. Make sure that billing is enabled for your Google Cloud project. [Learn how to confirm billing is enabled for your project](/billing/docs/how-to/modify-project)

4. Enable the Spanner, Pub/Sub, Compute Engine, and Dataflow APIs.

[Enable the APIs](https://console.cloud.google.com/flows/enableapi?apiid=spanner.googleapis.com,compute-component.googleapis.com,dataflow.googlea)

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see [Cleaning up](#) (#clean-up).

In this tutorial, you run commands in Cloud Shell. Cloud Shell gives you access to the command line in Google Cloud, and includes the Cloud SDK and other tools that you need for Google Cloud development. Cloud Shell can take several minutes to initialize.

1. Activate Cloud Shell.

[ACTIVATE Cloud Shell!](https://console.cloud.google.com/?cloudshell=true)

2. Set the default Compute Engine zone. For example, `us-central1-b`.

3. Clone the GitHub repository containing the sample code.

4. Go to the cloned directory.

5. Create a Python [virtual environment](https://virtualenv.pypa.io/en/stable/) (<https://virtualenv.pypa.io/en/stable/>).

6. Activate the virtual environment.

7. Install the required Python modules.

In this tutorial, you create and delete Amazon DynamoDB tables, Amazon S3 buckets, and other resources. To access these resources, you first need to create the required AWS Identity and Access Management (IAM) permissions. You can use a test or sandbox AWS account to avoid affecting production resources in the same account.

In this section, you create an AWS IAM role that AWS Lambda uses at a later step in the tutorial.

1. In the AWS console, go to the **IAM** section, click **Roles**, and then select **Create role**.
2. Under **Choose the service that will use this role**, click **Lambda**, and then select **Next:Permissions**.
3. In the **Policy Type** box, enter `AWSLambdaDynamoDBExecutionRole`.
4. Select the `AWSLambdaDynamoDBExecutionRole` checkbox, and then click **Next:Review**.
5. In the **Role name** box, enter `dynamodb-spanner-lambda-role`, and then click **Create role**.

Follow these steps to create an AWS IAM user with programmatic access to AWS resources, which are used throughout the tutorial.

1. While you are still in the **IAM** section of the AWS console, click **Users**, and then select **Add User**.
2. In the **User name** box, enter `dynamodb-spanner-migration`.
3. Under **Access type**, click **Programmatic access**.

★ **Note:** Don't enable management console access, because you won't use this IAM user to sign in to the console.

4. Click **Next: Permissions**.
5. Click **Attach existing policies directly** and select the following two policies:
 - `AmazonDynamoDBFullAccesswithDataPipeline`
 - `AmazonS3FullAccess`

6. Click **Next: Review**, and then click **Create user**.

7. Click **Show** to view the credentials. The access key ID and secret access key are displayed for the newly created user. Leave this window open for now because the credentials are needed in the following section. Safely store these credentials because with them, you can make changes to your account and affect your environment. At the end of this tutorial, you can [delete the IAM user](#) (#delete_aws_resources).

1. In Cloud Shell, configure the AWS Command Line Interface (CLI).

The following output appears:

- Enter the **ACCESS KEY ID** and **SECRET ACCESS KEY** from the AWS IAM account that you created.
- In the **Default region name** field, enter `us-west-2`. Leave other fields at their default values.

2. Close the AWS IAM console window.

The following section outlines the similarities and differences between data types, keys, and indexes for Amazon DynamoDB and Spanner.

Spanner uses standard SQL data types. The following table describes how Amazon DynamoDB [data types](#) (<http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes>) map to Spanner [data types](#) (/spanner/docs/data-types).

Amazon DynamoDB	Spanner
Number	Depending on precision or intended usage, might be mapped to INT64, FLOAT64, TIMESTAMP, or DATE.
String	String
Boolean	BOOL
Null	No explicit type. Columns can contain null values.
Binary	Bytes
Sets	Array
Map and List	Struct if the structure is consistent and can be described by using table DDL syntax.

An Amazon DynamoDB primary key establishes uniqueness and can be either a hash key or a combination of a hash key plus a range key. This tutorial starts by modeling the migration of a Amazon DynamoDB table whose primary key is a hash key. This hash key becomes the primary

key of your Spanner table. Later, in the [section on interleaved tables](#) (#interleaved_indexes), you model a situation where an Amazon DynamoDB table uses a primary key composed of a hash key and a range key.

Both Amazon DynamoDB and Spanner support creating an index on a non-primary key attribute. Take note of any secondary indexes in your Amazon DynamoDB table so that you can create them on your Spanner table, which is covered in a [later section of this tutorial](#) (#apply_secondary_indexes).

To facilitate this tutorial, you migrate the following sample table from Amazon DynamoDB to Spanner:

	Amazon DynamoDB	Spanner
Table name	Migration	Migration
Primary key	"Username" : String	"Username" : STRING(1024)
Key type	Hash	n/a
Other fields	Zipcode: Number Subscribed: Boolean ReminderDate: String PointsEarned: Number	Zipcode: INT64 Subscribed: BOOL ReminderDate: DATE PointsEarned: INT64

In the following section, you create an Amazon DynamoDB source table and populate it with data.

1. In Cloud Shell, create an Amazon DynamoDB table that uses the [sample table](#) (#sample_table) attributes.

2. Verify that the table status is ACTIVE.

3. Populate the table with sample data.

★ **Note:** If slightly fewer than the number of requested records are created, you can ignore the discrepancy. Fewer records might be created if duplicate sample usernames were generated.

You create a single-node instance, which is appropriate for testing and the scope of this tutorial. For a production deployment, refer to the documentation for [Spanner instances](/spanner/docs/instances#node_count) (/spanner/docs/instances#node_count) to determine the appropriate node count to meet your database performance requirements.

In this example, you create a table schema at the same time as the database. It is also possible, and common, to carry out [schema updates](/spanner/docs/gcloud-spanner#update_databases) (/spanner/docs/gcloud-spanner#update_databases) after you create the database.

Note: Both Amazon DynamoDB and Spanner support the use of secondary indexes. The best practices for [bulk loading](/spanner/docs/bulk-loading) (/spanner/docs/bulk-loading) Spanner recommend that you create secondary indexes *after* you load your initial data. You [set up a secondary index](#) (#apply_secondary_indexes) later in this tutorial.

1. Create a Spanner instance in the same region where you set the default Compute Engine zone. For example, us-central1.
2. Create a database in the Spanner instance along with the sample table.

The next sections show you how to export the Amazon DynamoDB source table and set Pub/Sub replication to capture any changes to the database that occur while you export it. If changes to your database aren't [idempotent](https://wikipedia.org/wiki/Idempotence) (https://wikipedia.org/wiki/Idempotence) and it isn't safe to write the same data more than once, it is best to carry out the following steps during a maintenance period when you can pause app changes to the table.

You use an AWS Lambda function to stream database changes to Pub/Sub.

1. In Cloud Shell, enable Amazon DynamoDB streams on your source table.
2. Set up a Pub/Sub topic to receive the changes.

The following output appears:

3. Create a Cloud IAM service account to push table updates to the Pub/Sub topic.

The following output appears:

4. Create a Cloud IAM policy binding so that the service account has permission to publish to Pub/Sub. Replace `GOOGLE_CLOUD_PROJECT` with the name of your Google Cloud project.

The following output appears:

5. Create credentials for the service account.

The following output appears:

6. Prepare and package the AWS Lambda function to push Amazon DynamoDB table changes to the Pub/Sub topic.

7. Create a variable to capture the Amazon Resource Name (ARN) of the Lambda execution role that you created earlier.

8. Use the `pubsub-lambda.zip` package to create the AWS Lambda function.

The following output appears:

9. Create a variable to capture the ARN of the Amazon DynamoDB stream for your table.

10. Attach the Lambda function to the Amazon DynamoDB table.

11. To optimize responsiveness during testing, add `--batch-size 1` to the end of the previous command, which triggers the function every time you create, update, or delete an item.

The following output appears:

1. In Cloud Shell, create a variable for a bucket name that you use in several of the following sections.

2. Create an Amazon S3 bucket to receive the DynamoDB export.

3. In the AWS Management Console, click **Data Pipeline**.
4. Click **Create new pipeline** to define the export job.
5. In the **Name** field, enter **Export to Amazon S3**.
6. For the **Source**, select the following:
 - **Build using a template.**
 - **Export DynamoDB table to Amazon S3.**
7. In the **Parameters** section, define the following:
 - a. In the **Source DynamoDB table name** field, enter **Migration**.
 - b. In the **Output S3 folder** field, click the **Folder** icon and select the [Your-Project-ID]-dynamodb-spanner-export Amazon S3 bucket that you just created where [YOUR-PROJECT-ID] represents your Google Cloud project ID.
 - c. To consume all available read-capacity during the export, in the **DynamoDB read throughput ratio** field, enter **1**. In a production environment, you adjust this value so that it doesn't hinder live operations.
 - d. In the **Region of your DynamoDB table** field, enter the name of the region, for example, **us-west-2**.
8. To start the backup jobs immediately, in the **Schedule** section for **Run**, click **On pipeline activation**.
9. Under **Pipeline Configuration**, in the **Logging** field, enter **Disabled**. If you are following this guide to migrate a production table, leave this option enabled and pointed at a separate Amazon S3 bucket for logs to help you troubleshoot errors. Leave other default parameters.
10. To begin the backup process, click **Activate**.
11. If you are prompted to address validation warnings, click **Activate**. In a production situation, you set a maximum duration for the job and enable logging.
12. Click **Refresh** to update the status of the backup process. The job takes several minutes to create the resources and finish exporting. In a production environment, you can speed this process up by modifying the **Data Pipeline** jobs to use more EMR resources.

When the process finishes, look at the output bucket.

The export job is done when there is a file named `_SUCCESS`.

If you paused writes to the database before exporting, it's time to reactivate writes to the database. Now that the Pub/Sub delivery is in place, you can push forward any table changes that occurred after the export.

1. In Cloud Shell, create a Cloud Storage bucket to receive the exported files from Amazon S3.
2. [Sync](#) (/storage/docs/gsuitil/commands/rsync) the files from Amazon S3 into Cloud Storage. For most copy operations, the `rsync` command is effective. If your export files are large (several GBs or more), use the [Cloud Storage transfer service](#) (/storage-transfer/docs/overview) to manage the transfer in the background.

The following output appears:

1. To write the data from the exported files into the Spanner table, run a Dataflow job with sample Apache Beam code.
 - a. To watch the progress of the import job, in the Cloud Console, go to Dataflow.
[GO TO Dataflow](https://console.cloud.google.com/dataflow) (https://console.cloud.google.com/dataflow)
 - b. While the job is running, you can watch the execution graph to examine the logs. Click the job that shows the **Status of Running**.

Name	Type	End time	Elapsed time	Start time	Status	ID	Region
spannerbulkwrite-0704011438-a8da563	Batch	—	29 sec	Jul 3, 2018, 6:14:45 PM	Running	2018-07-03_18_14_45-17424791151959584588	us-central1

2. Click each stage to see how many elements have been processed. The import is complete when all stages say **Succeeded**. The same number of elements that were created in your Amazon DynamoDB table display as processed at each stage.

← spanner...a36d47
LOGS
× Step

Step summary

Step name	CreateItemMutations
Wall time	1 sec
Transform Function	com.example.spanner_migration.SpannerBulkWrite\$CreateItemMutations

Input collections

ParseItems.out0

Elements added	25,000
Estimated size	22.48 MB

Output collections

CreateItemMutations.out0

Elements added	25,000
Estimated size	36.55 MB

3. Verify that the number of records in the destination Spanner table matches the number of items in the Amazon DynamoDB table.

★ **Note:** The Amazon DynamoDB item count is based on consistent table metadata and might not immediately reflect the total number of items in the table. The Spanner item count is based on a table query and accurately reflects the number of rows in the database when you run the query.

The following output appears:

4. Sample random entries in each table to make sure the data is consistent.

The following output appears:

5. Query the Amazon DynamoDB table with the same `Username` that was returned from the Spanner query in the previous step. For example, `aa11en2538`. Your value is specific to your database.

The values of the other fields should match those from the Spanner output. The following output appears:

When the batch import job is complete, you set up a streaming job to write ongoing updates from the source table into Spanner. You [subscribe \(/pubsub/docs/subscriber\)](#) to the events from Pub/Sub and write them to Spanner

The Lambda function you created is configured to capture changes to the source Amazon DynamoDB table and publish them to Pub/Sub.

1. Create a subscription to the Pub/Sub topic that AWS Lambda sends events to.

The following output appears:

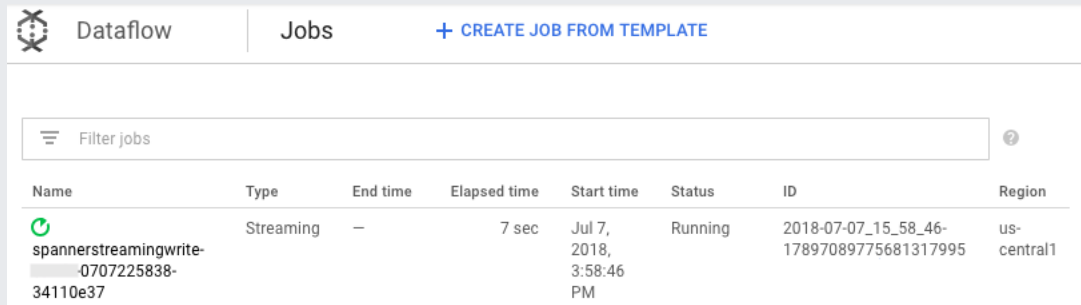
2. To stream the changes coming into Pub/Sub to write to the Spanner table, run the Dataflow job from Cloud Shell.



★ **Note:** To improve responsiveness during testing, you can set the [Windowing](https://beam.apache.org/documentation/programming-guide/#windowing) (https://beam.apache.org/documentation/programming-guide/#windowing) to 5 seconds by adding `--window 5` to the end of this command. The default is set to 60 seconds.

a. Similar to the [batch load](#) (#batch_import_the_data) step, to watch the progress of the job, in the Cloud Console, go to Dataflow.

[GO TO Dataflow](https://console.cloud.google.com/dataflow) (https://console.cloud.google.com/dataflow)

b. Click the job that has the **Status** of **Running**.



Name	Type	End time	Elapsed time	Start time	Status	ID	Region
 Dataflow	Jobs	+ CREATE JOB FROM TEMPLATE					
<input type="text" value="Filter jobs"/>							
 spannerstreamingwrite-0707225838-34110e37	Streaming	–	7 sec	Jul 7, 2018, 3:58:46 PM	Running	2018-07-07_15_58_46-17897089775681317995	us-central1

The processing graph shows a similar output as before, but each processed item is counted in the status window. The system lag time is a rough estimate of how much delay to expect before changes appear in the Spanner table.

spanner...110e37
LOGS
Step

```

graph TD
    A[Reading from PubSub  
Running  
0 sec] --> B[Create-or-Update?  
Running  
0 sec]
    A --> C[Delete?  
Running  
0 sec]
    B --> D[CU->Mutations  
Running  
0 sec]
    C --> E[D->Mutations  
Running  
0 sec]
    D --> F[Merging Mutations  
Running  
0 sec]
    E --> F
    F --> G[Creating Windows  
Running  
0 sec]
    G --> H[Commit->Spanner  
Part running  
5 sec]
  
```

Step summary

Step name	Commit->Spanner
System lag	35 sec
Data watermark	2018-07-07 (16:16:59)
Wall time	5 sec

Input collections

Creating Windows/Window.Assign.out0

Elements added	2
Estimated size	2.4 KB

The Dataflow job that you ran in the batch loading phase was a finite set of input, also known as a *bounded* dataset. This Dataflow job uses Pub/Sub as a streaming source and is considered *unbounded*. For more information about these two types of sources, review the section on PCollections in the [Apache Beam programming guide](https://beam.apache.org/documentation/programming-guide/#pcollections) (<https://beam.apache.org/documentation/programming-guide/#pcollections>). The Dataflow job in this step is meant to stay active, so it doesn't terminate when finished. The streaming Dataflow job remains in the **Running** status, instead of the **Succeeded** status.

You make some changes to the source table to verify that the changes are replicated to the Spanner table.

Note: The changes are asynchronous and might take several moments to appear in Spanner. Wait a few minutes before each step.

1. Query a nonexistent row in Spanner.

2. Create a record in Amazon DynamoDB with the same key that you used in the Spanner query. If the command runs successfully, there is no output.

3. Run the original query again to verify that the row is now in Spanner.

The output shows the inserted row:

4. Change some attributes in the original item and update the Amazon DynamoDB table.

The output appears as follows:

5. Verify that the changes are propagated to the Spanner table.

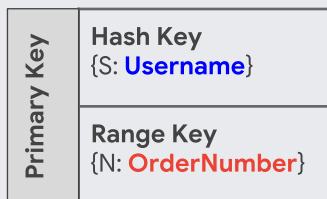
The output appears as follows:

6. Delete the test item from the Amazon DynamoDB source table.

7. Verify that the corresponding row is deleted from the Spanner table. When the change is propagated, the following command returns zero rows:

Spanner supports the concept of [interleaving tables](#) (/spanner/docs/schema-and-data-model#creating_a_hierarchy_of_interleaved_tables). This is a design model where a top-level item has several nested items that relate to that top-level item, such as a customer and their orders, or a player and their game scores. If your Amazon DynamoDB source table uses a primary key composed of a hash key and a range key, you can model an interleaved table schema as shown in the following diagram. This structure lets you efficiently query the interleaved table while joining fields in the parent table.

DynamoDB



Cloud Spanner

```
CREATE TABLE Users (
  Username STRING(1024) NOT NULL,
  PointsEarned INT64,
  ReminderDate DATE,
  Subscribed BOOL,
  Zipcode INT64,
) PRIMARY KEY (Username)

CREATE TABLE Orders (
  Username STRING(1024) NOT NULL,
  OrderNumber INT64 NOT NULL,
  OrderAmount INT64,
  OrderDate DATE,
  OrderFulfilled BOOL,
) PRIMARY KEY (Username, OrderNumber),
INTERLEAVE IN PARENT Users ON DELETE CASCADE
```

It is a [best practice](#) (/spanner/docs/bulk-loading#secondary-indexes) to apply secondary indexes to Spanner tables *after* you load the data. Now that replication is working, you set up a [secondary index](#) (/spanner/docs/secondary-indexes) to speed up queries. Like Spanner tables, Spanner secondary indexes are fully consistent. They are *not eventually consistent* (https://wikipedia.org/wiki/Eventual_consistency), which is common in many NoSQL databases. This feature can help simplify your app design

Run a query that doesn't use any indexes. You are looking for the top *N* occurrences, given a particular column value. This is a common query in Amazon DynamoDB for database efficiency.

1. Go to Spanner.

[GO TO Spanner](https://console.cloud.google.com/spanner/instances/spanner-migration/databases/migrationdb/schema/Migration) (https://console.cloud.google.com/spanner/instances/spanner-migration/databases/migrationdb/schema/Migration)

2. Click **QUERY**.

The screenshot shows the 'Table details' page for the 'Migration' table. The left sidebar shows the navigation path: Instance > Migration Demo > migrationdb > Migration. The main content area shows the 'Migration' table schema with columns: Username (Type: STRING(1024), Nullable: No).

3. In the **Query** field, enter the following query, and then click **Run query**.

After the query runs, click **Explanation** and take note of the **Rows scanned** versus **Rows returned**. Without an index, Spanner scans the entire table to return a small subset of data that matches the query.

The screenshot shows the 'Query database: migrationdb' page. The query is: `SELECT Username, PointsEarned FROM Migration WHERE Subscribed=true AND ReminderDate > DATE_SUB(DATE(current_timestamp()), INTERVAL 3 DAY)`. The 'Explanation' tab is selected, showing a table with columns: Total elapsed time (171.4 msec), CPU time (168.25 msec), Rows returned (853), and Rows scanned (25000). Below the table is an 'Operator reference' table showing the execution plan.

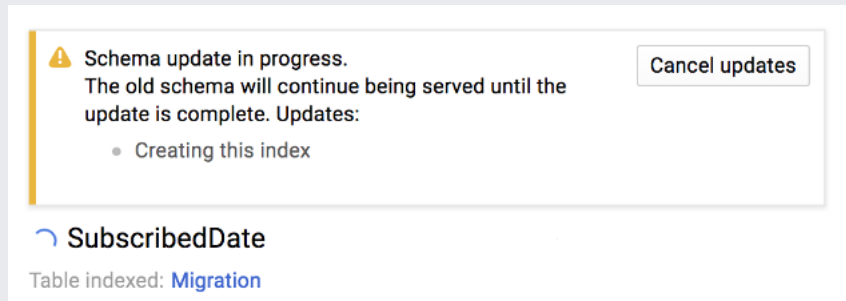
Operator	Rows returned	Executions	Latency
■ Distributed union	853	1	166 ms
↑ Local distributed union	853	1	166 ms
↑ Serialize Result	853	1	166 ms
↑ FilterScan	853	1	165 ms
↑ Table Scan: Migration	25000	1	163 ms

4. If this represents a commonly occurring query, create a composite index on the Subscribed and ReminderDate columns. In the Spanner console, click **Create Index**.

5. Click to turn on **Edit as Text**.

6. In the **DDL statements**, enter the index definition.

7. To begin building the database in the background, click **Create**.



A notification box with a yellow warning icon on the left. The text inside reads: "Schema update in progress. The old schema will continue being served until the update is complete. Updates:" followed by a bulleted list item "Creating this index". In the top right corner of the box is a button labeled "Cancel updates". Below the notification box, the text "SubscribedDate" is displayed with a refresh icon to its left. Underneath that, it says "Table indexed: Migration" where "Migration" is a blue link.

8. After the index is created, run the query again and add the index.

Examine the query explanation again. Notice that the number of **Rows scanned** has decreased. The **Rows returned** at each step matches the number returned by the query.

Schema Results table Explanation

Total elapsed time ? 39.76 msec	CPU time ? 36.45 msec	Rows returned 853	Rows scanned 1706
---	---------------------------------	-----------------------------	-----------------------------

[Operator reference](#) | [Guided tour](#)

Operator ?	Rows returned	Executions ?	Latency ?
▪ Distributed union	853	1	27 ms
↗ Distributed cross apply: Input : Map	853	1	27 ms
↗ Input Create Batch ▾	-	-	-
↑ Local distributed union	853	1	1 ms
↑ Compute Struct ▾	853	1	1 ms
↑ FilterScan	853	1	1 ms
↑ Index Scan: SubscribedDate ▾	853	1	1 ms
↗ Map Local distributed union	853	1	9 ms
↑ Serialize Result	853	1	9 ms
↗ Cross Apply: Input : Map	853	1	8 ms
↗ Input Batch Scan: \$v2 ▾	853	1	0 ms
↗ Map FilterScan	853	853	~0 ms
↑ Table Scan: Migration ▾	853	853	~0 ms

You can set up interleaved indexes in Spanner. The secondary indexes discussed in the [previous section](#) (#apply_secondary_indexes) are at the root of the database hierarchy, and they use indexes the same way as a conventional database. An interleaved index is within the context of its interleaved row. See [index options](#) (/spanner/docs/whitepapers/optimizing-schema-design#index_options) for more details about where to apply interleaved indexes.

In order to adapt the migration portion of this tutorial to your own situation, modify your Apache Beam source files. It is important that you don't change the source schema during the actual migration window, otherwise you can lose data.

1. To parse incoming JSON and build mutations, use [GSON](https://github.com/google/gson/blob/master/UserGuide.md) (https://github.com/google/gson/blob/master/UserGuide.md). Adjust the JSON definition to match your data.

[dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java](https://github.com/GoogleCloudPlatform/dynamodb-spanner-migration/blob/master/dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java)
(https://github.com/GoogleCloudPlatform/dynamodb-spanner-migration/blob/master/dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java)

github.com/GoogleCloudPlatform/dynamodb-spanner-migration/blob/master/dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java)

2. Adjust the corresponding JSON mapping.

[dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java](https://github.com/GoogleCloudPlatform/dynamodb-spanner-migration/blob/master/dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java)
(https://github.com/GoogleCloudPlatform/dynamodb-spanner-migration/blob/master/dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java)

github.com/GoogleCloudPlatform/dynamodb-spanner-migration/blob/master/dataflow/src/main/java/com/example/spanner_migration/SpannerBulkWrite.java)

In the previous steps, you modified the Apache Beam source code for bulk import. Modify the source code for the streaming part of the pipeline in a similar manner. Finally, adjust the table creation scripts, schemas, and indexes of your Spanner target database.


To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

[Go to the Manage resources page \(https://console.cloud.google.com/iam-admin/projects\)](https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project you want to delete and click **Delete** .

3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

If your AWS account is used outside of this tutorial, use caution when you delete the following resources:

1. Delete the [DynamoDB table](https://console.aws.amazon.com/dynamodb/home?tables:#tables:) (https://console.aws.amazon.com/dynamodb/home?tables:#tables:) called **Migration**.
2. Delete the [Amazon S3 bucket](https://s3.console.aws.amazon.com/s3/home?region=us-west-2) (https://s3.console.aws.amazon.com/s3/home?region=us-west-2) and [Lambda function](https://console.aws.amazon.com/lambda/home) (https://console.aws.amazon.com/lambda/home) that you created during the migration steps.
3. Finally, delete the [AWS IAM](https://console.aws.amazon.com/iam/home#/users) (https://console.aws.amazon.com/iam/home#/users) user that you created during this tutorial.

- Read about how to [optimize your Spanner schema](/spanner/docs/whitepapers/optimizing-schema-design) (/spanner/docs/whitepapers/optimizing-schema-design).
- Learn how to use [Dataflow](/dataflow/docs/how-to) (/dataflow/docs/how-to) for more complex situations.
- Review other [Spanner how-to guides](/spanner/docs/how-to) (/spanner/docs/how-to).
- Try out other Google Cloud features for yourself. Have a look at our [tutorials](/docs/tutorials) (/docs/tutorials).