This article explains how to migrate your database from Oracle® Online Transaction Processing (OLTP) systems to Cloud Spanner (/spanner).

For Online Analytical Processing (OLAP) databases, consider using BigQuery (/bigquery) as an alternative.

Spanner uses certain concepts differently from other enterprise database management tools, so you might need to adjust your application to take full advantage of its capabilities. You might also need to supplement Spanner with other services from Google Cloud to meet your needs.

When you migrate your application to Spanner, you must take into account the different features available. You probably need to redesign your application architecture to fit with Spanner's feature set and to integrate with additional Google Cloud services.

Spanner does not support running user code in the database level, so as part of the migration, you must move business logic implemented by database-level stored procedures and triggers into the application.

Spanner does not implement a sequence generator, and as explained below, using monotonically increasing numbers as primary keys is an anti-pattern in Spanner. An alternative way to generate a unique primary key is to use a random UUID (https://wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)).

If sequences are required for external reasons, then you must implement them in the application layer.

Spanner supports only database-level access controls using Cloud IAM access permissions and roles. Predefined roles can give read-write or read-only access to the database.

If you require finer grained permissions, you must implement them at the application layer. In a normal scenario, only the application should be allowed to read and write to the database.

If you need to expose your database to users for reporting, and want to use fine-grained security permissions (such as table- and view-level permissions), you should export your database to BigQuery (/bigquery/docs/loading-data-cloud-storage-avro).

Spanner can support a limited set of data validation constraints in the database layer.

If you need more complex data constraints, implement them in the application layer.

The following table discusses the types of constraints commonly found in Oracle® databases, and how to implement them with Spanner.

Constraint	Implementation with Spanner
Not null	`NOT NULL` column constraint
Unique	Secondary index with `UNIQUE` constraint
Foreign key (for normal tables)	Implemented in the application layer
Foreign key `ON DELETE/ON UPDATE` actions	Only possible for interleaved tables, otherwise implemented in the application layer
Value checks and validation via `CHECK` constraints or triggers	Implemented in the application layer

Oracle® databases and Spanner support different sets of data types. The following table lists the Oracle data types and their equivalent in Spanner. For detailed definitions of each Spanner data type, see Data Types (/spanner/docs/data-types).

You might also have to perform additional transformations on your data as described in the Notes column to make Oracle data fit in your Spanner database.

For example, you can store a large `BLOB` as an object in a Cloud Storage bucket rather than in the database, and then store the URI reference to the Cloud Storage object in the database as a `STRING`.

Oracle data type	Spanner equivalent	Notes
Character types (`CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`)	`STRING`	Note: Spanner uses Unicode strings throughout. Oracle supports a maximum length of 32,000 bytes or characters (depending on type), while Spanner supports up to 2,621,440 characters.
`BLOB`, `LONG RAW`, `BFILE`	`BYTES` or `STRING` containing URI to the object.	Small objects (less than 10 MiB) can be stored as `BYTES`. Consider using alternative Google Cloud offerings such as Cloud Storage to store larger objects.
`CLOB`, `NCLOB`, `LONG`	`STRING` (either containing data or URI to external object)	Small objects (less than 2,621,440 characters) can be stored as `STRING`. Consider using alternative Google Cloud offerings such as Cloud Storage to store larger objects.
`NUMBER`, `NUMERIC`, `DECIMAL`	`STRING`, `FLOAT64`, `INT64`	The `NUMBER` Oracle data type supports up to 38 digits of precision, while the `FLOAT64` Spanner data type supports up to 16 digits of precision. See Storing arbitrary precision numeric data (/spanner/docs/storing-numeric-data) for alternative mechanisms.
`INT`, `INTEGER`, `SMALLINT`	`INT64`	
`BINARY_FLOAT`, `BINARY_DOUBLE`	`FLOAT64`	
`DATE`	`DATE`	The default `STRING` representation of the Spanner `DATE` type is `yyyy-mm-dd`, which is different from Oracle's, so use caution when automatically converting to and from `STRING` representations of dates. SQL functions are provided to convert dates to a formatted string.
`DATETIME`	`TIMESTAMP`	Spanner stores time independent of timezone. If you need to store a timezone, you need to use a separate `STRING` column. SQL functions are provided to convert timestamps to a formatted string using timezones.

Oracle data type	Spanner equivalent	Notes
`XML`	`STRING` (either containing data or URI to external object)	Small XML objects (less than 2,621,440 characters) can be stored as `STRING`. Consider using alternative Google Cloud offerings such as Cloud Storage to store larger objects.
`URI`, `DBURI`, `XDBURI`, `HTTPURI`	`STRING`	
`ROWID`	`PRIMARY KEY`	Spanner uses the table's primary key to sort and reference rows internally, so in Spanner it is effectively the same as the `ROWID` data type.
`SDO_GEOMETRY`, `SDO_TOPO_GEOMETRY_SDO_GEORASTER`		Spanner does not support geospatial data types. You will have to store this data using standard data types, and implement any searching and filtering logic in the application layer.
`ORDAudio`, `ORDDicom`, `ORDDoc`, `ORDImage`, `ORDVideo`, `ORDImageSignature`		Spanner does not support media data types. Consider using Cloud Storage to store media data.

An overall timeline of your migration process would be:

- Convert your schema and data model.
- Translate any SQL queries.
- Migrate your application to use Spanner in addition to Oracle.
- Bulk export your data from Oracle and import your data into Spanner using Dataflow.
- Maintain consistency between both databases during your migration.
- Migrate your application away from Oracle.

You convert your existing schema to a Spanner schema (/spanner/docs/schema-and-data-model) to store your data. This should match the existing Oracle schema as closely as possible to make application

modifications simpler. However, due to the differences in features, some changes will be necessary.

Using best practices in schema design (/spanner/docs/schema-design) can help you increase throughput and reduce hot spots in your Spanner database.

Every table that needs to store more than one row must have a primary key consisting of one or more columns of the table. Your table's primary key uniquely identifies each row in a table, and because the table rows are sorted by primary key, the table itself will act as a primary index.

You should avoid designating columns that monotonically increase or decrease as the first part of the primary key (examples include sequences or timestamps), because this can lead to hot spots caused by inserts occurring at the end of your keyspace. A hot spot is a concentration of operations on a single node, which lowers the write throughput to the node's capacity instead of benefiting from load-balancing all writes among the Cloud Spanner nodes.

Use the following techniques to generate unique primary key values and reduce the risk of hot spots:

- Swap the order of keys (/spanner/docs/schema-design#fix_swap_key_order) so that the column that contains the monotonically increasing or decreasing value is not the first key part.
- Hash the unique key and spread the writes across logical shards (/spanner/docs/schema-design#fix_hash_the_key) by creating a column that contains the hash of the actual unique key, and then use the hash column (or the hash column and the unique key columns together) as the primary key. This helps avoid hot spots because new rows are spread more evenly across the keyspace.
- Use a universally unique identifier (UUID) (/spanner/docs/schema-design#uuid_primary_key) as defined by RFC 41122 (https://tools.ietf.org/html/rfc4122) as the primary key. You should use version 4 UUID because it uses random values in the bit sequence.
- Bit-reverse sequential values (/spanner/docs/schema-design#bit_reverse_primary_key) to distribute high order bits of subsequent numbers roughly equally over the entire number space.

After you designate your primary key for your table, you cannot change it later without deleting and recreating the table. For more information on how to designate your primary key, see Schema and data model-primary keys (/spanner/docs/schema-and-data-model#primary_keys).

Here is an example DDL statement creating a table for a database of music tracks:

Spanner has a feature where you can define two tables as having a one-to-many, parent-child relationship (/spanner/docs/schema-and-data-model#parent-child_table_relationships). This interleaves child data rows with their parent row in storage, effectively pre-joining the table and improving data retrieval efficiency when the parent and children are queried together.

The child table's primary key must start with the primary key column(s) of the parent table. From the child row's perspective, the parent row primary key is referred to as a foreign key. You can define up to 6 levels of parent-child relationships.

You can define on-delete actions (/spanner/docs/schema-and-data-model#creating-interleaved-tables) for child tables to determine what happens when the parent row is deleted: either all child rows are deleted, or the parent row deletion is blocked while child rows exist.

Here is an example of creating an Albums table interleaved in the parent Singers table defined earlier:

You can also create secondary indexes (/spanner/docs/secondary-indexes) to index data within the table outside of the primary key.

Spanner implements secondary indexes in the same way as tables, so the column values to be used as index keys have the same constraints (/spanner/docs/schema-and-data-model#primary_keys) as the primary keys of tables. This also means that indexes have the same consistency guarantees as Spanner tables.

Value lookups using secondary indexes are effectively the same as a query with a table join. You can improve the performance of queries using indexes by storing copies the original table's column values in the secondary index using the `STORING` clause, making it a covering index (https://wikipedia.org/wiki/Database_index#Covering_index).

Spanner's query optimizer will only automatically use secondary indexes when the index itself stores all the columns being queried (a covered query). To force the use of an index when querying columns in the original table, you must use a `FORCE_INDEX` directive (/spanner/docs/secondary-indexes#index_directive) in the SQL statement, for example:

Indexes can be used to enforce unique values within a table column, by defining a `UNIQUE` index (/spanner/docs/secondary-indexes#unique_indexes) on that column. Adding duplicate values will be prevented by the index.

Here is an example DDL statement creating a secondary index for the Albums table:

Note that if you create additional indexes after your data is loaded, populating the index may take some time. You should limit the rate at which you add them to an average of three per day. For more guidance on creating secondary indexes, see Secondary indexes (/spanner/docs/secondary-indexes). For more information on the limitations on index creation, see Schema updates (/spanner/docs/schema-updates#large-updates).

Spanner uses the ANSI 2011 dialect of SQL with extensions (/spanner/docs/query-syntax), and has many functions and operators (/spanner/docs/functions-and-operators) to help translate and aggregate your data. You must convert any SQL queries that use Oracle-specific syntax, functions, and types to be compatible with Spanner.

While Spanner does not support structured data as column definitions, structured data can be used in SQL queries using `ARRAY` and `STRUCT` types.

For example, a query could be written to return all Albums for an artist using an `ARRAY` of `STRUCT`s in a single query (taking advantage of the pre-joined data). For more information see the Notes about subqueries (/spanner/docs/query-syntax#notes-about-subqueries) section of the documentation.

SQL queries can be profiled using the Spanner Query interface in the Google Cloud Console to execute the query. In general, queries that perform full table scans on large tables are very expensive, and should be used sparingly.

See the SQL best practices (/spanner/docs/sql-best-practices) documentation for more information on optimising SQL queries.

Spanner provides a set of Client libraries (/spanner/docs/reference/libraries) for various languages, and the ability to read and write data using Spanner-specific API calls, as well as by using SQL queries (/spanner/docs/query-syntax) and Data modification language (DML) (/spanner/docs/dml-syntax) statements. Using API calls may be faster for some queries, such as direct row reads by key, because the SQL statement does not have to be translated.

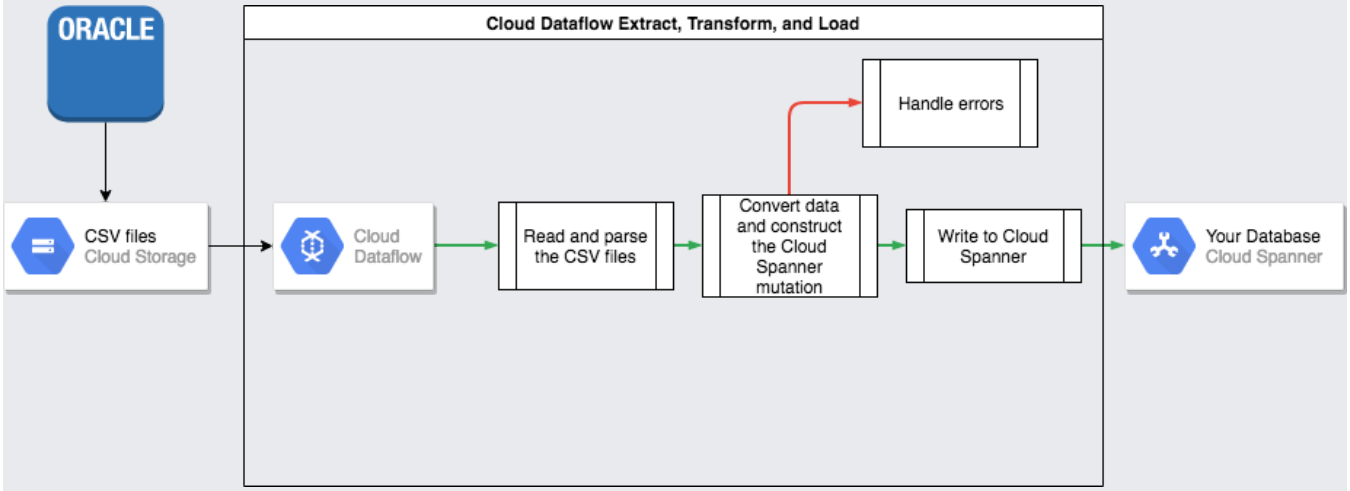
You can also use the Java Database Connectivity (JDBC) driver (/spanner/docs/partners/drivers) to connect to Spanner, leveraging existing tooling and infrastructure that does not have native integration.

As part of the migration process, features not available in Spanner must be implemented in the application. For example, a trigger to verify data values and update a related table would need to be implemented in the application using a read/write transaction to read the existing row, verify the constraint, then write the updated rows to both tables.

Spanner offers read-write and read-only transactions (/spanner/docs/transactions), which ensure external consistency of your data. Additionally, read transactions can have timestamp bounds (/spanner/docs/timestamp-bounds) applied, where you are reading a consistent version of the data specified in these ways:

- At an exact time in the past (up to 1 hour ago).
- In the future (where the read will block until that time arrives).
- With an acceptable amount of bounded staleness, which will return a consistent view up to some time in the past without needing to check that later data is available on another replica. This can give performance benefits at the expense of possibly stale data.

To transfer your data from Oracle to Spanner, you will need to export your Oracle database to a portable file format, for example CSV, then import that data into Spanner using Dataflow.



Oracle does not provide any built-in utilities for exporting or unloading your entire database into a portable file format.

Some options for performing an export are listed in the Oracle FAQ (http://www.orafaq.com/wiki/SQL*Loader_FAQ#Is_there_a_SQL.2AUnloader_to_download_data_to_a_flat_file.3F) .

These include:

- Using SQL*plus or SQLcl to spool a query to a text file.
- Writing a PL/SQL function using UTL_FILE (https://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:9536328100346697722) to unload a table in parallel to text files.
- Using features within Oracle APEX (https://docs.oracle.com/cd/E18283_01/appdev.112/e12511/sql_utl.htm#insertedID2) or Oracle SQL Developer (https://www.oracle.com/database/technologies/appdev/sql-developer.html) to unload a table to a CSV or XML file.

Each of these has the disadvantage that only one table can be exported at a time, which means that you must pause your application or quiesce your database (http://www.orafaq.com/node/2943) so that the database remains in a consistent state for export.

Other options include third-party tools as listed in the Oracle FAQ (http://www.orafaq.com/wiki/SQL*Loader_FAQ#Is_there_a_SQL.2AUnloader_to_download_data_to_a_flat_file.3F) page, some of which can unload a consistent view of the entire database.

After they're unloaded, you should upload these datafiles to a Cloud Storage (/storage) bucket so that they are accessible for import.

Because the database schemas probably differ between Oracle and Spanner, you might need to make some data conversions as part of the import process.

The easiest way to perform these data conversions and import the data into Spanner is by using Dataflow (/dataflow/).

Dataflow is the Google Cloud distributed Extract Transform and Load (ETL) service. It provides a platform for running data pipelines written using the Apache Beam SDK (https://beam.apache.org/get-started/beam-overview/) in order to read and process large amounts of data in parallel over multiple machines.

The Apache Beam SDK requires you to write a simple Java program to set read, transform and write the data. Beam connectors exist for Cloud Storage and Spanner, so the only code that needs to be written is the data transform itself.

See an example of a simple pipeline that reads from CSV files and writes to Spanner in the sample code repository (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/dataflow/spanner-io/src/main/java/com/example/dataflow/SpannerWrite.java) that accompanies this article.

If parent-child interleaved tables are used in your Spanner schema, then care must be taken in the import process so that the parent row is created before the child row. The Spanner Import pipeline code (https://github.com/GoogleCloudPlatform/DataflowTemplates/blob/master/src/main/java/com/google/cloud/teport/spanner/ImportTransform.java) handles this by importing all data for root level tables first, then all the level 1 child tables, then all the level 2 child tables, and so on.

The Spanner import pipeline can be used directly to bulk import your data, (/spanner/docs/import-non-spanner) but this requires that your data exist in Avro files using the correct schema.

Many applications have availability requirements that make it impossible to keep the application offline for the time required to export and import your data. While you are transferring your data to Spanner, your application continues modifying the existing database. You must duplicate updates to the Spanner database while the application is running.

There are various methods of keeping your two databases in sync, including Change Data Capture, and implementing simultaneous updates in the application.

Oracle GoldenGate (https://www.oracle.com/middleware/technologies/goldengate.html) can provide a change data capture (https://wikipedia.org/wiki/Change_data_capture) (CDC) stream for your Oracle database. Oracle LogMiner (https://docs.oracle.com/database/121/SUTIL/GUID-3417B738-374C-4EE3-B15C-3A66E01AE2B5.htm) or Oracle XStream Out (https://docs.oracle.com/database/121/XSTRM/xstrm_pt_xout.htm) are alternative interfaces for the Oracle database to obtain a CDC stream that does not involve Oracle GoldenGate.

You can write an application that subscribes to one of these streams and that applies the same modifications (after data conversion, of course) to your Spanner database. Such a stream processing application has to implement several features:

- Connecting to the Oracle database (source database).
- Connecting to Cloud Spanner (target database).
- Repeatedly performing the following:
    - Receiveing the data produced by one of the Oracle database CDC streams.
    - Interpreting the data produced by the CDC stream.
    - Converting the data into Spanner `INSERT` statements.
    - Executing the Spanner `INSERT` statements.

Database migration technology is middleware technology that has implemented the required features as part of its functionality. The database migration platform is installed as a separate component either at the source location or the target location, in accordance with customer requirements. The database migration platform only requires connectivity configuration of the databases involved in order to specify and start continuous data transfer from the source to the target database.

Striim (https://www.striim.com/) is a database migration technology platform that's available on Google Cloud. It provides connectivity to CDC streams from Oracle GoldenGate as well as from Oracle LogMiner and Oracle XStream Out. Striim provides a graphical tool that lets you configure database connectivity and any transformation rules that are required in order to transfer data from Oracle to Spanner.

You can install Striim from the Google Cloud Marketplace connect to the source and target databases, implement any transformation rules, and start transferring data without having to build a stream processing application yourself.

An alternative method is to modify your application to perform writes to both databases. One database (initially Oracle) would be considered the source of truth, and after each database write, the entire row is read, converted, and written to the Spanner database.

In this way, the application constantly overwrites the Spanner rows with the latest data.

After you're confident that all your data has been transferred correctly, you can switch the source of truth to the Spanner database.

This mechanism provides a rollback path if issues are found when switching to Spanner.

As data streams into your Spanner database, you can periodically run a comparison between your Spanner data and your Oracle data to make sure that the data is consistent.

You can validate consistency by querying both data sources and comparing the results.

You can use Dataflow to perform a detailed comparison over large data sets by using the Join transform (https://beam.apache.org/documentation/pipelines/design-your-pipeline/#multiple-sources). This transform takes 2 keyed data sets, and matches the values by key. The matched values can then be compared for equality.

You can regularly run this verification until the level of consistency matches your business requirements.

When you have confidence in the data migration, you can switch your application to using Spanner as the source of truth. Continue writing back changes to the Oracle database to keep the Oracle database up to date, giving you a rollback path should issues arise.

Finally, you can disable and remove the Oracle database update code and shut down the Oracle database.

You can optionally export your tables from Spanner to a Cloud Storage bucket using a Dataflow template to perform the export. The resulting folder contains a set of Avro files and JSON manifest files containing your exported tables. These files can serve various purposes, including:

- Backing up your database for data retention policy compliance or disaster recovery.
- Importing the Avro file into other Google Cloud offerings such as BigQuery.

For more information on the export and import process, see Exporting Databases (/spanner/docs/export) and Importing Databases (/spanner/docs/import).

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