Using SAP HANA Cloud for Production

Table of Contents

- SAP HANA Cloud System Limits
- Native Associations

SAP HANA Cloud is supported as the CAP standard database and recommended for productive use with full support for schema evolution and multitenancy.

> **Validation strategy**
>
> CAP isn't validated with other variants of SAP HANA, like "SAP HANA Database as a Service" or "SAP HANA (on premise)".
>
> The database services are validated against the latest maintained QRC version of SAP HANA Cloud. It's not guaranteed that outdated versions are fully functional with the latest database services.
>
> ↳ *See the official SAP HANA Cloud documentation for their maintenance strategy.*

Setup & Configuration

To use SAP HANA Cloud for production, add a dependency to the *package.json* for Node.js or to the *pom.xml* for a CAP Java application:

Shell/Bash pom.xml

```sh
npm add @cap-js/hana
```

▶ *Using other SAP HANA drivers...*

▶ *In CAP Java ...*

> **Prefer `cds add`**
>
> ... as documented in the deployment guide, which also does the equivalent of `npm add @cap-js/hana` but in addition cares for updating `mta.yaml` and other deployment resources.

Running *cds build*

Deployment to SAP HANA is done via the SAP HANA Deployment Infrastructure (HDI) which in turn requires running *cds build* to generate all the deployable HDI artifacts. For example, run this in capire/bookshop :

```sh
cds build --for hana
```

Which should display this log output:

```log
[cds] - done > wrote output to:
   gen/db/init.js
   gen/db/package.json
   gen/db/src/gen/.hdiconfig
   gen/db/src/gen/.hdinamespace
   gen/db/src/gen/AdminService.Authors.hdbview
   gen/db/src/gen/AdminService.Books.hdbview
   gen/db/src/gen/AdminService.Books_texts.hdbview
   gen/db/src/gen/AdminService.Currencies.hdbview
   gen/db/src/gen/AdminService.Currencies_texts.hdbview
   gen/db/src/gen/AdminService.Genres.hdbview
   gen/db/src/gen/AdminService.Genres_texts.hdbview
   gen/db/src/gen/CatalogService.Books.hdbview
   gen/db/src/gen/CatalogService.Books_texts.hdbview
   gen/db/src/gen/CatalogService.Currencies.hdbview
   gen/db/src/gen/CatalogService.Currencies_texts.hdbview
   gen/db/src/gen/CatalogService.Genres.hdbview
   gen/db/src/gen/CatalogService.Genres_texts.hdbview
   gen/db/src/gen/CatalogService.ListOfBooks.hdbview
   gen/db/src/gen/data/sap.capire.bookshop-Authors.csv
   gen/db/src/gen/data/sap.capire.bookshop-Authors.hdbtabledata
   gen/db/src/gen/data/sap.capire.bookshop-Books.csv
   gen/db/src/gen/data/sap.capire.bookshop-Books.hdbtabledata
   gen/db/src/gen/data/sap.capire.bookshop-Books.texts.csv
   gen/db/src/gen/data/sap.capire.bookshop-Books.texts.hdbtabledata
   gen/db/src/gen/data/sap.capire.bookshop-Genres.csv
   gen/db/src/gen/data/sap.capire.bookshop-Genres.hdbtabledata
   gen/db/src/gen/localized.AdminService.Authors.hdbview
   gen/db/src/gen/localized.AdminService.Books.hdbview
   gen/db/src/gen/localized.AdminService.Currencies.hdbview
   gen/db/src/gen/localized.AdminService.Genres.hdbview
```

```
gen/db/src/gen/localized.CatalogService.Books.hdbview
gen/db/src/gen/localized.CatalogService.Currencies.hdbview
gen/db/src/gen/localized.CatalogService.Genres.hdbview
gen/db/src/gen/localized.CatalogService.ListOfBooks.hdbview
gen/db/src/gen/localized.sap.capire.bookshop.Authors.hdbview
gen/db/src/gen/localized.sap.capire.bookshop.Books.hdbview
gen/db/src/gen/localized.sap.capire.bookshop.Genres.hdbview
gen/db/src/gen/localized.sap.common.Currencies.hdbview
gen/db/src/gen/sap.capire.bookshop.Authors.hdbtable
gen/db/src/gen/sap.capire.bookshop.Books.hdbtable
gen/db/src/gen/sap.capire.bookshop.Books_author.hdbconstraint
gen/db/src/gen/sap.capire.bookshop.Books_currency.hdbconstraint
gen/db/src/gen/sap.capire.bookshop.Books_foo.hdbconstraint
gen/db/src/gen/sap.capire.bookshop.Books_genre.hdbconstraint
gen/db/src/gen/sap.capire.bookshop.Books_texts.hdbtable
gen/db/src/gen/sap.capire.bookshop.Genres.hdbtable
gen/db/src/gen/sap.capire.bookshop.Genres_parent.hdbconstraint
gen/db/src/gen/sap.capire.bookshop.Genres_texts.hdbtable
gen/db/src/gen/sap.common.Currencies.hdbtable
gen/db/src/gen/sap.common.Currencies_texts.hdbtable
```

Generated HDI Artifacts

As we see from the log output `cds build` generates these deployment artifacts as expected by HDI , based on CDS models and .csv files provided in your projects:

- `.hdbtable` files for entities
- `.hdbview` files for views / projections
- `.hdbconstraint` files for database constraints
- `.hdbtabledata` files for CSV content
- a few technical files required by HDI, such as `.hdinamespace` and `.hdiconfig`

Custom HDI Artifacts

In addition to the generated HDI artifacts, you can add custom ones by adding according files to folder `db/src` . For example, let's add an index for Books titles...

1. Add a file *db/src/sap.capire.bookshop.Books.hdbindex* and fill it with this content:

db/src/sap.capire.bookshop.Books.hdbindex

```sql
INDEX sap_capire_bookshop_Books_title_index
ON sap_capire_bookshop_Books (title)
```

2. Run cds build again → this time you should see this additional line in the log output:

```log
[cds] - done > wrote output to:
[...]
gen/db/src/sap.capire.bookshop.Books.hdbindex
```

↳ *Learn more about HDI Design-Time Resources and Build Plug-ins*

Deploying to SAP HANA

There are two ways to include SAP HANA in your setup: Use SAP HANA in a hybrid mode, meaning running your services locally and connecting to your database in the cloud, or running your whole application on SAP Business Technology Platform. This is possible either in trial accounts or in productive accounts.

To make the following configuration steps work, we assume that you've provisioned, set up, and started, for example, your SAP HANA Cloud instance in the trial environment . If you need to prepare your SAP HANA first, see How to Get an SAP HANA Cloud Instance for SAP Business Technology Platform, Cloud Foundry environment to learn about your options.

Prepare for Production

To prepare the project, execute:

```sh
cds add hana --for hybrid
```

This configures deployment for SAP HANA to use the *hdbtable* and *hdbview* formats. The configuration is added to a `[hybrid]` profile in your *package.json*.

> **The profile `hybrid` relates to the hybrid testing scenario**
>
> If you want to prepare your project for production and use the profile `production`, read the Deploy to Cloud Foundry guide.

No further configuration is necessary for Node.js. For Java, see the Use SAP HANA as the Database for a CAP Java Application tutorial for the rest of the configuration.

Using `cds deploy` for Ad-Hoc Deployments

`cds deploy` lets you deploy *just the database parts* of the project to an SAP HANA instance. The server application (the Node.js or Java part) still runs locally and connects to the remote database instance, allowing for fast development roundtrips.

Make sure that you're logged in to Cloud Foundry with the correct target, that is, org and space. Then in the project root folder, just execute:

```sh
cds deploy --to hana
```

> To connect to your SAP HANA Cloud instance use `cds watch --profile hybrid` in Node.js or `mvn cds:watch` in Java projects.

Behind the scenes, `cds deploy` does the following:

- Compiles the CDS model to SAP HANA files (usually in *gen/db*, or *db/src/gen*)
- Generates *.hdbtabledata* files for the CSV files in the project. If a *.hdbtabledata* file is already present next to the CSV files, no new file is generated.
- Creates a Cloud Foundry service of type `hdi-shared`, which creates an HDI container. Also, you can explicitly specify the name like so: `cds deploy --to hana:<myService>`.
- Starts `@sap/hdi-deploy` locally. If you need a tunnel to access the database, you can specify its address with `--tunnel-address <host:port>`.
- Stores the binding information with profile `hybrid` in the *.cdsrc-private.json* file of your project. You can use a different profile with parameter `--for`. With this information, `cds watch` / `run` can fetch the SAP HANA credentials at runtime, so that the server can connect to it.

Specify `--profile` when running `cds deploy` as follows:

```sh
cds deploy --to hana --profile hybrid
```

Based on these profile settings, `cds deploy` executes `cds build` and also resolves additionally binding information. If a corresponding binding exists, its service name and service key are used. The development profile is used by default.

↳ *Learn more about the deployment using HDI.*

↳ *Learn more about hybrid testing using service bindings to Cloud services.*

If you run into issues, see the Troubleshooting guide.

Deploy Parameters

When using the option `--to hana` , you can specify the service name and logon information in several ways.

`cds deploy --to hana`

In this case the service name and service key either come from the environment variable `VCAP_SERVICES` or are defaulted from the project name, for example, `myproject-db` with `myproject-db-key` . Service instances and key either exist and will be used, or otherwise they're created.

`cds deploy --to hana:myservice`

This overwrites any information coming from environment variables. The service name `myservice` is used and the current Cloud Foundry client logon information is taken to connect to the system.

`cds deploy --vcap-file someEnvFile.json`

This takes the logon information and the service name from the `someEnvFile.json` file and overwrite any environment variable that is already set.

`cds deploy --to hana:myservice --vcap-file someEnvFile.json`

This is equivalent to `cds deploy --to hana:myservice` and ignores information coming from `--vcap-file` . A warning is printed after deploying.

Using *cf deploy* or *cf push*

See the Deploying to Cloud Foundry guide for information about how to deploy the complete application to SAP Business Technology Platform, including a dedicated deployer application for the SAP HANA database.

Native SAP HANA Features

The HANA Service provides dedicated support for native SAP HANA features as follows.

Vector Embeddings

Vector embeddings let you add semantic search, recommendations, and generative AI features to your CAP application. Embeddings are numeric arrays that represent the meaning of unstructured data (text, images, etc.), making it possible to compare and search for items that are semantically related to each other or a user query.

Choose an Embedding Model

Choose an embedding model that fits your use case and data (for example english or multilingual text). The model determines the number of dimensions of the resulting output vector. Check the documentation of the respective embedding model for details.

Use the SAP Generative AI Hub for unified consumption of embedding models and LLMs across different vendors and open source models. Check for available models on the SAP AI Launchpad .

Add Embeddings to Your CDS Model

Use the *cds.Vector* type in your CDS model to store embeddings on SAP HANA Cloud. Set the dimension to match your embedding model (for example, 1536 embedding dimensions for OpenAI *text-embedding-3-small*).

```cds
entity Books : cuid {
  title       : String(111);
  description : LargeString;
```

```
  embedding    : Vector(1536); // adjust dimensions to embedding model
}
```

Generate Embeddings

Use an embedding model to convert your data (for example, book descriptions) into vectors. The SAP Cloud SDK for AI makes it easy to call SAP AI Core services to generate these embeddings.

▶ *Example using SAP Cloud SDK for AI*

Query for Similarity

At runtime, use SAP HANA's built-in vector functions to search for similar items. For example, find books with embeddings similar to a user question:

Java Node.js

Java

```
// Compute embedding for user question
var request = new OpenAiEmbeddingRequest(List.of("How to use vector embedding
CdsVector userQuestion = CdsVector.of(
 aiClient.embedding(request).getEmbeddingVectors().get(0));
// Compute similarity between user question and book embeddings
var similarity = CQL.cosineSimilarity( // computed on SAP HANA
  CQL.get(Books.EMBEDDING), userQuestion);
// Find Books related to user question ordered by similarity
hana.run(Select.from(BOOKS).limit(10)
.columns(b -> b.ID(), b -> b.title(),
         b -> similarity.as("similarity"))
.orderBy(b -> b.get("similarity").desc()));
```

TIP

Store embeddings when you create or update your data. Regenerate embeddings if you change your embedding model.

SAP Cloud SDK for AI

Use the SAP Cloud SDK for AI for unified access to embedding models and large language models (LLMs) from SAP AI Core.

↳ *Learn more about the SAP Cloud SDK for AI (Java)*

↳ *Learn more about Vector Embeddings in CAP Java*

↳ *Learn more about the SAP Cloud SDK for AI (JavaScript)*

Geospatial Functions

CDS supports the special syntax for SAP HANA geospatial functions:

```cds
entity Geo as select from Foo {
  geoColumn.ST_Area() as area : Decimal,
  new ST_Point(2.25, 3.41).ST_X() as x : Decimal
};
```

↳ *Learn more in the SAP HANA Spatial Reference .*

Spatial Grid Generators

SAP HANA Spatial has some built-in grid generator table functions . To use them in a CDS model, first define corresponding facade entities in CDS.

Example for function `ST_SquareGrid` :

```cds
@cds.persistence.exists
entity ST_SquareGrid(size: Double, geometry: hana.ST_GEOMETRY) {
  geom: hana.ST_GEOMETRY;
  i: Integer;
  j: Integer;
}
```

Then the function can be called, parameters have to be passed by name:

```cds
entity V as select
  from ST_SquareGrid(size: 1.0, geometry: ST_GeomFromWkt('Point(1.5 -2.5)'))
{ geom, i, j };
```

Functions Without Arguments

SAP HANA allows to omit the parentheses for functions that don't expect arguments. For example:

```cds
entity Foo { key ID : UUID; }
entity Bar as select from Foo {
  ID, current_timestamp
};
```

Some of which are well-known standard functions like `current_timestamp` in the previous example, which can be written without parentheses in CDS models. However, there are many unknown ones, that aren't known to the compiler, for example:

- `current_connection`
- `current_schema`
- `current_transaction_isolation_level`
- `current_utcdate`
- `current_utctime`
- `current_utctimestamp`
- `sysuuid`

To use these in CDS models, you have to add the parentheses so that CDS generic support for using native features can kick in:

```cds
entity Foo { key ID : UUID; }
entity Bar as select from Foo {
  ID, current_timestamp,
  sysuuid() as sysid
};
```

Regex Functions

CDS supports SAP HANA Regex functions (`locate_regexpr` , `occurrences_regexpr` , `replace_regexpr` , and `substring_regexpr`), and SAP HANA aggregate functions with an additional `order by` clause in the argument list. Example:

```sql
locate_regexpr(pattern in name from 5)
first_value(name order by price desc)
```

Restriction: `COLLATE` isn't supported.

For other functions, where the syntax isn't supported by the compiler (for example, `xmltable(...)`), a native *.hdbview* can be used. See Using Native SAP HANA Artifacts for more details.

HDI Schema Evolution

CAP supports database schema updates by detecting changes to the CDS model when executing the CDS build. If the underlying database offers built-in schema migration techniques, compatible changes can be applied to the database without any data loss or the need for additional migration logic. Incompatible changes like deletions are also detected, but require manual resolution, as they would lead to data loss.

Change	Detected Automatically	Applied Automatically
Adding fields	**Yes**	**Yes**
Deleting fields	**Yes**	No
Renaming fields	n/a [1]	No
Changing datatype of fields	**Yes**	No
Changing type parameters	**Yes**	**Yes**
Changing associations/compositions	**Yes**	No [2]
Renaming associations/compositions	n/a [1]	No
Renaming entities	n/a	No

> [1] Rename field or association operations aren't detected as such. Instead, corresponding ADD and DROP statements are rendered requiring manual resolution activities.
> [2] Changing targets may lead to renamed foreign keys. Possibly hard to detect data integrity issues due to non-matching foreign key values if target key names remain the same (for example "ID").

> **No support for incompatible schema changes**
>
> Currently there's no framework support for incompatible schema changes that require scripted data migration steps (like changing field constraints NULL > NOT NULL).

> However, the CDS build does detect those changes and renders them as non-executable statements, requesting the user to take manual resolution steps. We recommend avoiding those changes in productive environments.

Schema Evolution and Multitenancy/Extensibility

There's full support for schema evolution when the *cds-mtxs* library is used for multitenancy handling. It ensures that all schema changes during base-model upgrades are rolled out to the tenant databases.

> **WARNING**
>
> Tenant-specific extensibility using the *cds-mtxs* library isn't supported yet Right now, you can't activate extensions on entities annotated with `@cds.persistence.journal` .

Schema Updates with SAP HANA

All schema updates in SAP HANA are applied using SAP HANA Deployment Infrastructure (HDI) design-time artifacts, which are auto-generated during CDS build execution.

Schema updates using *.hdbtable* deployments are a challenge for tables with large data volume. Schema changes with *.hdbtable* are applied using temporary table generation to preserve the data. As this could lead to long deployment times, the support for *.hdbmigrationtable* artifact generation has been added. The Migration Table artifact type uses explicit versioning and migration tasks. Modifications of the database table are explicitly specified in the design-time file and carried out on the database table exactly as specified. This saves the cost of an internal table-copy operation. When a new version of an already existing table is deployed, HDI performs the migration steps that haven't been applied.

Deploy Artifact Transitions as Supported by HDI

Current format	hdbcds	hdbtable	hdbmigrationtable
hdbcds		yes	n/a
hdbtable	n/a		yes
hdbmigrationtable	n/a	Yes	

> **WARNING**
>
> Direct migration from *.hdbcds* to *.hdbmigrationtable* isn't supported by HDI. A deployment using *.hdbtable* is required up front.
>
> ↳ *Learn more in the **Enhance Project Configuration for SAP HANA Cloud** section.*
>
> During the transition from *.hdbtable* to *.hdbmigrationtable* you have to deploy version=1 of the *.hdbmigrationtable* artifact, which must not include any migration steps.

HDI supports the *hdbcds → hdbtable → hdbmigrationtable* migration flow without data loss. Even going back from *.hdbmigrationtable* to *.hdbtable* is possible. Keep in mind that you lose the migration history in this case. For all transitions you want to execute in HDI, you need to specify an undeploy allowlist as described in HDI Delta Deployment and Undeploy Allow List in the SAP HANA documentation.

> **Moving From *.hdbcds* To *.hdbtable***
>
> There a migration guide providing you step-by-step instructions for making the switch.
>
> ↳ *Learn more about Moving From .hdbcds To .hdbtable*

Enabling hdbmigrationtable Generation for Selected Entities During CDS Build

If you're migrating your already deployed scenario to *.hdbmigrationtable* deployment, you've to consider the remarks in Deploy Artifact Transitions as Supported by HDI.

By default, all entities are still compiled to *.hdbtable* and you only selectively choose the entities for which you want to build *.hdbmigrationtable* by annotating them with `@cds.persistence.journal`.

Example:

cds

```
namespace data.model;

  @cds.persistence.journal
  entity LargeBook {
    key id : Integer;
    title : String(100);
    content : LargeString;
  }
```

CDS build generates *.hdbmigrationtable* source files for annotated entities as well as a *last-dev/csn.json* source file representing the CDS model state of the last build.

> These source files have to be checked into the version control system.

Subsequent model changes are applied automatically as respective migration versions including the required schema update statements to accomplish the new target state. There are cases where you have to resolve or refactor the generated statements, like for reducing field lengths. As they can't be executed without data loss (for example, `String(100)` → `String(50)`), the required migration steps are only added as comments for you to process explicitly.

Example:

```txt
>>>> Manual resolution required - DROP statements causing data loss are disab
>>>> by default.
>>>> You may either:
>>>>   uncomment statements to allow incompatible changes, or
>>>>   refactor statements, e.g. replace DROP/ADD by single RENAME statement
>>>> After manual resolution delete all lines starting with >>>>>
-- ALTER TABLE my_bookshop_Books DROP (title);
-- ALTER TABLE my_bookshop_Books ADD (title NVARCHAR(50));
```

Changing the type of a field causes CDS build to create a corresponding ALTER TABLE statement. Data type conversion rules are applied by the SAP HANA database as part of the deployment step. This may cause the deployment to fail if the column contents can't be converted to the new format.

Examples:

1. Changing the type of a field from String to Integer may cause tenant updates to fail if existing content can't be converted.
2. Changing the type of a field from Decimal to Integer can succeed, but decimal places are truncated. Conversion fails if the content exceeds the maximum Integer length.

We recommend keeping *.hdbtable* deployment for entities where you expect low data volume. Every *.hdbmigrationtable* artifact becomes part of your versioned source code, creating a new migration version on every model change/build cycle. In turn, each such migration can require manual resolution. You can switch large-volume tables to *.hdbmigrationtable* at any time, keeping in mind that the existing *.hdbtable* design-time artifact needs to be undeployed.

When choosing to use *.hdbmigrationtable* for an entity with localized elements or compositions of aspects, the generated `.texts` and composition child entities are

automatically handled via *.hdbmigrationtable*, too. If this is not desired, annotate these generated entities with `@cds.persistence.journal: false` .

> **TIP**
>
> Sticking to *.hdbtable* for the actual application development phase avoids lots of initial migration versions that would need to be applied to the database schema.

CDS build performs rudimentary checks on generated *.hdmigrationtable* files:

- CDS build fails if inconsistencies are encountered between the generated *.hdbmigrationtable* files and the *last-dev/csn.json* model state. For example, the last migration version not matching the table version is such an inconsistency.
- CDS build fails if manual resolution comments starting with `>>>>>` exist in one of the generated *.hdbmigrationtable* files. This ensures that manual resolution is performed before deployment.

Native Database Clauses

Not all clauses supported by SQL can directly be written in CDL syntax. To use native database clauses also in a CAP CDS model, you can provide arbitrary SQL snippets with the annotations `@sql.prepend` and `@sql.append` . In this section, we're focusing on schema evolution specific details.

Schema evolution requires that any changes are applied by corresponding ALTER statements. See ALTER TABLE statement reference for more information. A new migration version is generated whenever an `@sql.append` or `@sql.prepend` annotation is added, changed, or removed. ALTER statements define the individual changes that create the final database schema. This schema has to match the schema defined by the TABLE statement in the *.hdbmigrationtable* artifact. Please note that the compiler doesn't evaluate or process these SQL snippets. Any snippet is taken as is and inserted into the TABLE statement and the corresponding ALTER statement. The deployment fails in case of syntax errors.

CDS Model:

```cds
@cds.persistence.journal
@sql.append: 'PERSISTENT MEMORY ON'
entity E {
  ...,
  @sql.append: 'FUZZY SEARCH INDEX ON'
```

```
  text: String(100);
}
```

Result in hdbmigrationtable file:

```sql
== version=2
COLUMN TABLE E (
  ...,
  text NVARCHAR(100) FUZZY SEARCH INDEX ON
) PERSISTENT MEMORY ON

== migration=2
ALTER TABLE E PERSISTENT MEMORY ON;
ALTER TABLE E ALTER (text NVARCHAR(100) FUZZY SEARCH INDEX ON);
```

It's important to understand that during deployment new migration versions will be applied on the existing database schema. If the resulting schema doesn't match the schema as defined by the TABLE statement, deployment fails and any changes are rolled-back. In consequence, when removing or replacing an existing `@sql.append` annotation, the original ALTER statements need to be undone. As the required statements can't automatically be determined, manual resolution is required. The CDS build generates comments starting with `>>>>` in order to provide some guidance and enforce manual resolution.

Generated file with comments:

```txt
== migration=3
>>>>> Manual resolution required - insert ALTER statement(s) as described bel
>>>>> After manual resolution delete all lines starting with >>>>>
>>>>> Insert ALTER statement for: annotation @sql.append of artifact E has be
>>>>> Insert ALTER statement for: annotation @sql.append of element E:text ha
```

Manually resolved file:

```sql
== migration=3
ALTER TABLE E PERSISTENT MEMORY DEFAULT;
ALTER TABLE E ALTER (text NVARCHAR(100) FUZZY SEARCH INDEX OFF);
```

Appending text to an existing annotation is possible without manual resolution. A valid ALTER statement will be generated in this case. For example, appending the `NOT NULL`

column constraint to an existing `FUZZY SEARCH INDEX ON` annotation generates the following statement:

```sql
ALTER TABLE E ALTER (text NVARCHAR(100) FUZZY SEARCH INDEX ON NOT NULL);
```

> **WARNING**
>
> You can use `@sql.append` to partition your table initially, but you can't subsequently change the partitions using schema evolution techniques as altering partitions isn't supported yet.

Advanced Options

The following CDS configuration options are supported to manage *.hdbmigrationtable* generation.

> **WARNING**
>
> This hasn't been finalized yet.

```js
{
  "hana" : {
    "journal": {
      "enable-drop": false,
      "change-mode": "alter" // "drop"
    },
    // ...
  }
}
```

The `"enable-drop"` option determines whether incompatible model changes are rendered as is (`true`) or manual resolution is required (`false`). The default value is `false` .

The `change-mode` option determines whether `ALTER TABLE ... ALTER` (`"alter"`) or `ALTER TABLE ... DROP` (`"drop"`) statements are rendered for data type related changes. To ensure that any kind of model change can be successfully deployed to the database, you can switch the `"change-mode"` to `"drop"` , keeping in mind that any

existing data will be deleted for the corresponding column. See hdbmigrationtable Generation for more details. The default value is `"alter"`.

Caveats

CSV Data Gets Overridden

HDI deploys CSV data as *.hdbtabledata* and assumes exclusive ownership of the data. It's overridden with the next application deployment; hence:

> **TIP**
>
> Only use CSV files for *configuration data* that can't be changed by application users.

Yet, if you need to support initial data with user changes, you can use the `include_filter` option that *.hdbtabledata* offers.

Undeploying Artifacts

As documented in the HDI Deployer docs, an HDI deployment by default never deletes artifacts. So, if you remove an entity or CSV files, the respective tables, and content remain in the database.

By default, `cds add hana` creates an `undeploy.json` like this:

db/undeploy.json

```json
[
  "src/gen/**/*.hdbview",
  "src/gen/**/*.hdbindex",
  "src/gen/**/*.hdbconstraint",
  "src/gen/**/*_drafts.hdbtable",
  "src/gen/**/*.hdbcalculationview"
]
```

If you need to remove deployed CSV files, also add this entry:

db/undeploy.json

```json
[
  [...]
  "src/gen/**/*.hdbtabledata"
]
```

↳ *See this troubleshooting entry for more information.*

SAP HANA Cloud System Limits

All limitations for the SAP HANA Cloud database can be found in the SAP Help Portal .

Native Associations

In previous CAP releases, CDS associations were by default reflected in SAP HANA database tables and views by *Native HANA Associations* (HANA SQL clause `WITH ASSOCIATIONS`). But the presence of such native associations significantly increases (re-)deploy times: They need to be validated in the HDI deployment, and they can introduce indirect dependencies between other objects, which can trigger other unnecessary revalidations or even unnecessary drop/create of indexes.

As CAP doesn't need these native associations, by default no native HANA associations are created anymore starting with CAP 9.

In the unlikely case that you need native HANA associations because you explicitly use them in other native HANA objects or in custom code, you can switch them back on with `cds.sql.native_hana_associations = true` ☼.

> **Initial full table migration**
>
> Be aware that the first deployment after this **configuration change may take longer**.
>
> For each entity with associations, the respective database object is touched (DROP/CREATE for views, full table migration via shadow table and data copy for tables).

Previous page
PostgreSQL

Next page
SAP HANA Native

Was this page helpful?