Search documentation Search

Appian

Database Schema Best Practices

Appian 7.7

NOTE: This site does not include documentation on the latest release of Appian. Please upgrade to benefit from our newest features. For more information on the latest release of Appian, please see the Appian 7.8 documentation

Toggle Na

Schema Design

Schema Management

Schema Design

This section describes recommended approaches to database schema design when creating them for use with Appian.

Annotations

If you use the JoinColumn annotation in your XSD, when you insert/update data in a data store, the corresponding field does not update in the StoredValue output of the Write to Data Store node. Creating a query rule to retrieve the updated value, as a custom output on the Write to Data Sto node or as part of a subsequent script task, ensures the StoredValue output contains the correct data.

CDT and Element Names

Keep the names assigned to your custom data types and CDT elements shorter than 27 characters.The names of database tables, columns, and other SQL objects created from an XSD are truncated if they exceed 27 characters in length. This is done to maintain compatibility with all supported databases.

Consider using JPA annotations in your XSD to avoid truncation or a change in letter casing. Name attributes for the @Table and @Column annotation are used exactly when referencing the data type for a data store.

See also: Using Annotations in Your XSD

Your server administrator can adjust this limit, which is only advisable if you are using an RDBMS other than Oracle.

See also: Data Store SQL Name Length

The truncation of long SQL names (longer than 27 characters) assigned to your custom data types and CDT elements occurs in the following manner when deriving database table and column names:

- Vowels are removed from the name, in the order of u, o, a, e, then i. Names that begin with a vowel retain that vowel only.
- The name is split using underscores.
- The longest segment of the name is then trimmed, one character at a time, until the name is shortened to 27 characters.

If you have two similar long names for columns, it is possible for the two column names to be truncated to the same value.

- Use `@Table` and `@Column` annotations to define table names within the character limit that should be used if the type or element names excee the target database's limits.

Column Types

The column type is inferred from the XSD type according to it's associated data type. When mapping to existing tables, if the inferred column type doesn't match an existing table's column type, you must explicitly define the column type using the `@Column` annotation with the `columnDefinition` attribute.

For example, if you have an `xsd:string` typed element that maps to a column defined as `CHAR(40)`, use the following annotation within the XSD: @Column(columnDefinition="CHAR(40)")

See also: Primitive Data Types

Note: When mapping to a column defined with `CHAR(N)` where `N > 1`, the database may return a value padded with blanks if the length of the value less than **N**. When used in conjunction with a data type annotated with the `@Version` annotation, this behavior may cause the system to issue an update statement to the database, triggering the version value to increment, even if the significant part of the value remains unchanged. This is becaus the trailing spaces will be trimmed from the value to be stored.

If this behavior is not desired, consider using `VARCHAR` or the equivalent for the target database. Of the supported databases, MySQL is the only database that trims the padding from the returned value and therefore will not trigger the version update.

For integer values, use `xsd:int` as the element type instead of `xsd:integer`. `xsd:int` has the same maximum value as a CDT field of type Number(Integer).

See also: Integers.

Composite Keys

Tables with composite keys are not supported. The table definition must be altered to only have one primary key for query rules and the Write to Data Store node to fully work with them.

Constraints

Databases can enforce constraints on column values that are not enforced by Appian on CDT field values.

For example, a database may require a column value to be `"not null"`, but the field in Appian can be empty. Be aware of these differences when designi your data stores.

Use the attribute nillable="true" to prevent columns in your data store from enforcing a `"non-null"` constraint.

DDL Editing

There are some classes of differences between an existing table schema and a CDT that are detected, but cannot be automatically resolved by opting t have the system update the tables for you. In such cases, you must alter the database tables by downloading and executing the DDL script.

The system does not alter column types if there is a mismatch between an element type in the XSD and the corresponding column type. Either add a JPA annotation to do so, or change the column type directly within the RDBMS.

If the XSD has a type name or `@Table` annotation with a name that is intended to map to a synonym or a view, the system cannot validate that the table exists.

It may be necessary to drop the existing tables and recreate them using the generated DDL. The generated DDL includes code comments that can be used for this purpose.

DDL Version Control

When verifying a data store against the schema in your data source, download the generated DDL and store it in a version control system.

Each time the DDL is generated it contains the commands needed to update an existing schema as well as the DDL commands needed to drop and recreate the schema (in a commented out section of the DDL document).

You can use the DDL to load the schema onto a fresh system, such as when you're importing an application that uses data stores onto a new system.

Null Values

Empty text process variables are handled by the system as empty strings, not as `"null"` values. This allows empty strings to be inserted into the data store. If the corresponding column in the RDBMS is configured as `"not null"`, the database does not prevent the empty string from being inserted.

Optimistic Locking

Consider using the `@Version` annotation on a CDT field to enable optimistic locking on a data object.

Without using the `@Version` annotation, if a secondary process has updated the object in the data store since the current process last read it, the valu stored by the secondary process is overwritten.

With the `@Version` annotation, the database detects that the value being stored by the current process is stale and prevents the update. The `@Version` annotation can be used on only one field per CDT.

For information on using `CHAR` columns in the target database table when using the `@Version` annotation, see above: Column Types

Primary Keys

Records stored in a data store are keyed by a primary key. The `@Id` annotation can be used to indicate which field of the CDT you want to designate

your primary key.

If your XSD doesn't define a field as its primary key using the @Id annotation, a primary key is created for it automatically.

If the primary key for a data store entity is created by the system, you won't be able identify which record you want to update, because that primary key is hidden from your view. This behavior inserts new records in the data store, even when the data in two records are the same.

Generating Primary Keys

When using a sequence to generate ids (such as `@GeneratedValue(strategy = SEQUENCE)`), we recommend allowing the system to name the sequenc instead of providing a name with the `"generator"` attribute.

If you provide a name using the `"generator"` attribute and multiple entities are defined in the same data store, you may receive the following error message during validation: *A type mapping annotation is invalid: could not instantiate id generator (APNX-2-4055-000)*. This is a known issue that shou be resolved in a future release.

Query Rules

Creating a query rule as a custom output on a Write to Data Store node or as part of a subsequent script task after inserting or updating data in a data store ensures the StoredValue output contains the correct data.

Triggers

If you use triggers to insert/update data in a data store, the values set by your triggers do not populate in the `StoredValue` output of the Write to Data Store node.

Creating a query rule to retrieve the new/updated data, as a custom output on the Write to Data Store Smart Service or as part of a subsequent script task, ensures the `StoredValue` output contains the correct data.

Validation

The following differences between an existing table schema and a CDT cannot be detected during table validation:

- Conflicting null value definitions, such as when a CDT field was defined with an XSD element using the attribute `nillable = "true"`, while the existin table column is set to `NOT NULL`. The opposite is also true
- If the primary key was annotated with a `@GenereratedValue` annotation, but the existing table column is not set to auto increment or use a sequence
- If a sequence is specified by the `@GeneratedValue` annotation, but that sequence does not already exist
- If the CDT field defines a length using an `@Column` annotation in the XSD that doesn't match an existing column's length constraint
- If the existing table contains extra columns that do not map to fields in the CDT. This is only an issue if the columns are defined as non nullable ar no default value is defined.

Schema Management

This section describes recommended approaches to managing database schemas used with Appian.

User Privileges

When creating a tablespace, create a user with privileges only to that tablespace. Granting a user privileges to multiple schemas (or DBA privileges on ar Oracle database) may effect queries to other schemas.

Affected Data Stores

After you modify a database schema, always make sure to re-verify and re-publish the affected data stores. This ensures any changes you made will tal effect and any issues raised because of the change are addressed.

If your application is restarted, and you begin receiving errors when query rules and the Write to Data Store Smart Service nodes execute, it may be because a database schema was modified and its data store was not re-verified and re-published. You can resolve the errors by republishing the affecte data stores.

User Guides by Role

- Designer
- Developer
- Web Admin
- Server Admin (On-Premise Only)

Tutorials

- Records
- Interfaces