21

Migrating From JSON To Duality

The **JSON-To-Duality Migrator** can migrate one or more *existing* sets of JSON documents to JSON-relational duality views. Its PL/SQL subprograms generate the views based on inferred *implicit* document-content relations. By default, document parts that *can* be shared *are* shared, and the views are defined for *maximum updatability*.

- JSON Columns in Duality Views
  With JSON Relational Duality, developers can leverage JSON documents for data access while using the highly efficient relational data storage model, without compromising simplicity or efficiency.
- About Migrations From JSON To Duality
  Learn about the **JSON-To-Duality Migrator** use cases for existing and new applications.
- JSON To Duality Migrator Components: Converter and Importer
  The JSON To Duality Migrator has two components: the *converter* and the *importer*. Their PL/SQL subprograms are described.
- JSON Configuration Fields Specifying Migrator Parameters
  You configure JSON-to-duality migration by passing a migrator *configuration object* as argument to PL/SQL `DBMS_JSON_DUALITY` subprograms `infer_schema`, `infer_and_generate_schema`, and `import_all`. The supported *fields* of such an object are described.
- School Administration Example, Migrator Input Documents
  Existing student, teacher, and course document sets comprise the JSON-to-duality migrator input for the school-administration example. In a typical migration scenario each might be received in the form of a JSON dump file from another database.
- Before Using the Converter (1): Create Database Document Sets
  Before using the JSON-to-duality converter you need to create `JSON`-type document sets in Oracle Database from the original external document sets. The input to the converter for each set of documents is an Oracle Database table with a single column of `JSON` data type.
- Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas
  A data-guide JSON schema provides frequency information about the fields in a document set, in addition to structure and type information. You can use such schemas to get an idea how migration might proceed, and you can compare them with other JSON schemas as a shortcut for comparing document sets.
- JSON-To-Duality Converter: What It Does
  The converter infers the inherent structure and typing of one or more sets of stored documents, as a JSON schema. Using the schema, the converter generates DDL code to create the database objects needed to support the document sets: duality views and their underlying tables and indexes.
- Migrating To Duality, Simplified Recipe
  By ignoring whether an input field occurs rarely, or with a rarely used type, it's easier to migrate to JSON-relational duality. Handling such outlier cases can complicate the migration process.
- Using the Converter, Default Behavior
  Use of the JSON-to-duality converter with its default configuration-field values (except for `minFieldFrequency` and `minTypeFrequency`) is illustrated. In particular, configuration field

`useFlexFields` is `true`. The database objects needed to support the document sets are inferred, and the SQL DDL code to construct them is generated.

- Import After Default Conversion
  After default conversion (except for `minFieldFrequency` and `minTypeFrequency`), in particular with `useFlexFields:true`), almost all documents from the student, teacher, and course input document sets are successfully imported, but some fields are not exactly as they were in the original, input documents.
- Using the Converter with useFlexFields=false
  Use of the JSON-to-duality converter with `useFlexFields` = `false` is illustrated. Otherwise the configuration is default (except for `minFieldFrequency` and `minTypeFrequency`). The database objects needed to support the document sets are inferred, and the SQL DDL code to construct them is generated.
- Import After Conversion with useFlexFields=false
  After trying to import, error-log tables are created and queried to show import errors and imported documents.
- Errors That Migrator Configuration Alone Can't Fix
  Even if you configure the migrator to not consider any fields or their values to be outliers, the migrator can detect other kinds of problems. A simple example shows detection of data contradiction between different document sets.

21.1 JSON Columns in Duality Views

With JSON Relational Duality, developers can leverage JSON documents for data access while using the highly efficient relational data storage model, without compromising simplicity or efficiency.

Including columns of JSON data type in tables that underlie a duality view lets applications add and delete fields, and change the types of field values, in the documents supported by the view. The stored JSON data can be schemaless or JSON Schema-based (to enforce particular types of values).

When you define a duality view, you can declaratively choose the kind and degree of schema flexibility you want, for particular document parts or whole documents.

A duality-view flex column stores (in an underlying table) JSON objects whose fields aren't predefined: they're not mapped individually to specific underlying columns. Unrecognized fields of an object in a document you insert or update are automatically added to the flex column for that object's underlying table.

You can thus add fields to the document object produced by a duality view with a flex column underlying that object, without redefining the duality view. This provides another kind of schema flexibility to a duality view, and to the documents it supports. If a given underlying table has no column identified in the view as flex, then new fields are not automatically added to the object produced by that table. Add flex columns where you want this particular kind of flexibility.

Note:

To understand more about JSON-relational duality views, and details about duality-view flex columns, see *JSON-Relational Duality Developer's Guide*.

Related Topics

- Using JSON-Relational Duality Views in *JSON-Relational Duality Developer's Guide*

21.2 About Migrations From JSON To Duality

Learn about the **JSON-To-Duality Migrator** use cases for existing and new applications.

Including columns of JSON data type in tables that underlie a duality view lets applications add and delete fields, and change the types of field values, in the documents supported by the view. The stored JSON data can be schemaless or JSON Schema-based (to enforce particular types of values).

Migration requires no supervision, but you should of course check the resulting duality views and their supported documents to verify their adequacy to your needs. You can modify the migration behavior to change the result.

There are *two main use cases*[1] for the JSON-to-duality migrator:

- Migrate an *existing application* and its sets of JSON documents from a document database to Oracle Database.
- Create a *new application*, based on knowledge of the different kinds of JSON documents it will use (their structure and typing). The migrator can simplify this job, by automatically creating the necessary duality views.

Migration of existing stored document sets to document collections supported by duality views consists of the following operations. You use the **converter** for steps 3-8, and the **importer** for steps 9-13. Steps 1-2 are preliminary. Steps are 2, 4, 7, 10, and 13 are optional.

1. *Create `JSON`-type document sets* in Oracle Database from the original external document sets.

   The input to the converter for each set of documents is an Oracle Database table with a single column of `JSON` data type. You can export JSON document sets from a document database and import them into `JSON`-type columns.

2. Optionally *create JSON data guides* that are *JSON schemas* that describe the input document sets.

   These can be useful for later comparison with parts of the JSON schema inferred by the converter or with the completed document collections resulting from migration.

3. *Infer database objects needed* to support the input documents: relational tables, indexes, and duality views.

   This step first validates the existing input data, checking whether the document sets can in fact be converted to duality-view support.

   The tables constitute a *normalized relational schema*. Normalization is both across and within document sets: equivalent data in different document sets is *shared*, by storing it in the same table.

4. Optionally *modify/edit the inferred JSON schema*.
5. *Create database objects* needed to support the input data:
   a. *Generate SQL data definition language (DDL) scripts* that create the database objects (duality views and their underlying tables and indexes).
   b. Optionally *modify/edit the DDL scripts*.
   c. *Run the scripts* to create the database objects.

1 The migrator doesn't help with the third main use case of duality views: Reusing *existing relational* data (tables) for use in JSON documents.

6. *Validate created database objects* (the duality views and their underlying relational schema). That is, validate the input documents against the database objects, to see which, if any, documents aren't supported by the duality views, and why.
7. Optionally *refine/fix* (modify/edit) input documents or DDL scripts.
   a. Optionally *modify/edit some input documents* that are erroneous (outliers you don't want to keep as is, or conflicting data between different documents), so they fit the created database objects.
   b. Optionally *modify/edit the DDL scripts*, to change the conversion behavior or the names of the views, tables, or indexes to be created.
8. Repeat steps 3-9, until the document sets to be imported and the (unpopulated) duality views fit together as desired.
9. **Import** the document sets into the duality views. Check for any errors logged.
10. Optionally *refine/fix* (modify/edit) input documents or DDL scripts, to fix import errors.
11. Repeat steps 9-10 (or 3-10), as needed, to fix logged import errors, until all documents are successfully imported.
12. *Validate import*: Check for any problems, with the successfully imported documents.
13. Optionally *refine/fix* (modify/edit) input documents or DDL scripts to resolve import validation problems.
14. Repeat steps 9-13 (or 3-13), as needed, to fix import problems.

To illustrate the use of the JSON-to-duality migrator we employ three small sets of documents that could be used by a school-administration application: *student*, *teacher*, and *course* documents. (A real application would of course likely have many more documents in its document sets, and the documents might be complex.) The pre-existing input document sets are shown in the student documentation set, teacher documentation set, and course documentation set in School Administration Example, Migrator Input Documents.

Each of the document sets is loaded into a `JSON`-type column, **`data`**, of a temporary **transfer table** from a document-database dump file of documents of a given kind (for example, student documents). The transfer-table names have suffix **`_tab`** (e.g., `student_tab` for student documents). Column `data` is the only column in a transfer table.

The migrator creates the corresponding duality views (e.g. view `student` for student documents) and populates them with the data from the transfer tables of stored documents. Once this is done, and you've *verified* the adequacy of the duality views, the transfer tables are no longer needed; you can drop them. The document sets are then no longer stored as such; their now-normalized data is stored in the tables underlying the duality views.

Note:

There's *no guarantee* that migration to duality views preserves all pre-existing application data completely. In the process of normalization some data may be transformed, cast to different data types, or truncated to respect maximum size limits. Input data that doesn't conform to the destination relational schema might then be rejected during import.

You need to check that all data has been successfully imported, by running migrator verification tests and examining error logs.

You can ensure that your imported data is valid by comparing the documents in an input document set with those supported by the corresponding duality view, checking that the duality-view documents contain only the expected fields, and that no fields are missing or modified in unacceptable ways.

Related Topics

- Schema Flexibility with JSON Columns in Duality Views
- Flex Columns, Beyond the Basics

21.3 JSON To Duality Migrator Components: Converter and Importer

The JSON To Duality Migrator has two components: the *converter* and the *importer*. Their PL/SQL subprograms are described.

- **Converter**: Create the database objects needed to support the original JSON documents: duality views and their underlying tables and indexes.
- **Importer**: Import Oracle Database `JSON`-type document sets that correspond to the original external documents into the duality views created by the converter.

The **converter** is composed of these PL/SQL functions in package **`DBMS_JSON_DUALITY`**:

- **`infer_schema`** infers a JSON schema that represents all of the input document sets.
  - *Input:* A JSON object whose members specify configuration parameters for the inference operation — see JSON Configuration Fields Specifying Migrator Parameters.
  - *Output:* a *JSON Schema* document that specifies the inferred relational schema. If no such schema can be found then an error is raised saying that the converter can't create duality views that correspond to the input document sets.
- **`generate_schema`** produces the DDL code to create the required database objects for each duality view.
  - *Input:* the JSON schema returned by function `infer_schema`.
  - *Output:* DDL scripts to create the needed database objects.
- **`infer_and_generate_schema`** performs both operations.
  - *Input:* same as `infer_schema`.
  - *Output:* same as `generate_schema`.

- **`validate_schema_report`** checks the adequacy of the database objects to be created by the DDL code generated by function `generate_schema`. It reports on the validity of the input JSON documents according to the duality views to be created, identifying documents that can't be supported, with reasons why not. These are the documents that fail validation against a JSON schema for the duality views.
  - *Input:*
    - `table_owner_name`: The name of the database schema (user) that owns table `table_name`.
    - `table_name`: The name of an input table of JSON documents.
    - `view_owner_name`: The name of the database schema (user) that owns view `view_name`.
    - `view_name`: The name of the corresponding duality view to be created by the DDL code generated by `generate_schema`.
  - *Output:* A table of validation failures for input JSON documents, one row per failed document (no rows means that all documents are valid). A row has `CLOB` columns `DATA` (the invalid document) and `ERRORS` (a JSON array of errors, with the same format as field `errors` of function `DBMS_JSON_SCHEMA.validate_report`).

The **importer** is composed of these PL/SQL subprograms in package **`DBMS_JSON_DUALITY`**:

- Procedure **`import_all`** populates *all* duality views created by the converter with the documents from the corresponding input document sets (more precisely, with the relational data needed to support such documents). In an error-log table it reports an error for each document that couldn't be imported. (Only the first such error encountered per document is reported.)
  - *Input:* A JSON object whose members specify configuration parameters for the import operation — see JSON Configuration Fields Specifying Migrator Parameters .
  - *Output:* (1) *Duality views* with their underlying tables *filled* with the relational data that supports the same documents. (2) *Error-log* tables that report any documents that could not be imported.
- Procedure **`import`** populates a *single* duality view with the documents from the corresponding input document set. Its error logging is the same as that of procedure `import_all`.
  - *Input:* (1) An Oracle Database JSON document set, that is, a table with a single `JSON`-type column containing documents of a given kind. (2) The name of a duality view to populate. (3, optional) Name of the table owner. (4, optional) Name of the view owner. (5, optional) Name of the owner of the error log table. (6, optional) Name of the error log table. (7, optional) Reject limit value, whose meaning is the same as configuration field `rejectLimit`: the maximum number of errors that can be logged (importing is ended when this limit is exceeded).
  - *Output:* (1) A *duality view* with its underlying tables *filled* with the relational data that supports the same documents. (2) An *error-log* table that reports any documents that could not be imported.

Tip:

In general, use procedure `import_all`, *not* procedure `import`. Perform a single-view `import` only when it's unlikely to interfere with the data in other duality views.

Using `import` to import multiple single views separately can be problematic because of view interdependencies. For example:

- You can't use `import` to populate *view* `student` *before view* `teacher`, because root table `student_root` has foreign key column `advisor_id`, which *requires the corresponding teacher data to already exist*.
- On the other hand, you can't use `import` to populate *view* `teacher` *before view* `student`, because teacher documents have a `dormId` field in their embedded student objects, and the corresponding column, `dorm_id`, is a foreign key from table `student_root` to table `student_dormitory`. This *requires that table* `student_dormitory` *be populated before view* `teacher`. And that table can only be populated by importing existing student documents.

- Function **`validate_import_report`** reports successfully imported JSON documents that are invalid.

  Each row in the output table corresponds to a validation failure for a JSON document that was imported (no rows means all documents are valid). A row has `CLOB` columns `DATA` (the invalid document) and `ERRORS` (a JSON array of errors, each having the format of a *JSON Patch* document that compares an input document and the corresponding imported document in the duality view). See JavaScript Object Notation (JSON) Patch, IETF RFC6902 for the error format.

  - `table_owner_name`: The name of the database schema (user) that owns table `table_name`.
  - `table_name`: The name of an input table of JSON documents.
  - `view_owner_name`: The name of the database schema (user) that owns view `view_name`.
  - `view_name`: The name of the corresponding duality view to be populated with the documents in table `table_name`.

Note that *import error logging* reports only on a document that couldn't be imported, and *import validation* reports only on documents that were successfully imported (but that are problematic in some way).

Related Topics

- Flex Columns, Beyond the Basics

See Also:

- DBMS_JSON_DUALITY in *Oracle Database PL/SQL Packages and Types Reference* for information about subprograms `generate_schema`, `infer_schema`, , `import`, `import_all`, `infer_and_generate_schema`, `validate_import_report`, and `validate_schema_report`
- VALIDATE_REPORT Function in *Oracle Database PL/SQL Packages and Types Reference* for information about function `DBMS_JSON_SCHEMA.validate_report`.

21.4 JSON Configuration Fields Specifying Migrator Parameters

You configure JSON-to-duality migration by passing a migrator *configuration object* as argument to PL/SQL `DBMS_JSON_DUALITY` subprograms `infer_schema`, `infer_and_generate_schema`, and `import_all`. The supported *fields* of such an object are described.

Note:

You might want to skim this topic on a first reading, and refer back to it later. The information is presented here to give you an idea of what's available.

Procedure `import_all` is the only subprogram that uses configuration fields `errorLog`, `errorLogSchema`, and `rejectLimit`. Functions `infer_schema` and `infer_and_generate_schema` are the only subprograms that use configuration fields `hints`, `ingestLimit`, `minFieldFrequency`, `minTypeFrequency`, `normalize`, `outputFormat`, `softnessThreshold`, `tablespace`, `updatability`, and `useFlexFields`.

The configuration fields actually used by the various `DBMS_JSON_DUALITY` migrator subprograms are thus different, but there is some overlap. You can pass any of the configuration fields to any of these subprograms; fields that aren't used are ignored. In particular, this means that you can pass a common configuration document to any of these subprograms.

Instead of accepting a JSON configuration object, PL/SQL subprograms `import`, `validate_schema_report`, and `validate_import_report` accept specific non-JSON *arguments* that act the same as, or similarly to, the use of some of the configuration fields. The parameter names are similar to the field names, and the field descriptions here generally apply to the corresponding parameters as well. For example, parameter `table_name` of function `validate_import_report` corresponds to configuration field `tableName`.

These are the migrator configuration fields. All of them *except* `tableNames` are *optional*. The use of any fields other than those listed raises an error.

- **`errorLog`** (Optional) — A *string* that names the single error log to use, or *an array of strings* that name the error logs to use, one for each duality view.

  Field `errorLog` is used only for procedure `import_all`.

- **`errorLogSchema`** (Optional) — A string that names the database schema (user) that owns the error log(s). If you don't specify an error-log owner in `errorLogSchema`, then the name of the currently connected user is used.

  Field `errorLogSchema` is used only for procedure `import_all`.

- **`hints`** (Optional) — A JSON array with elements that are JSON objects whose fields specify *overrides* for the behavior of the converter in generating a relational schema. The name "hint" is a bit of a misnomer, as these are imperatives, not mere suggestions: if a hint can't be respected for some reason then an error is raised; a hint is never ignored. An error is also raised if a hint is specified incorrectly.

  A hint object must have these fields (otherwise, an error is raised):

  – **`type`** — The value is one of these strings, specifying the type of migrator-behavior override:

    * **`"datatype"`** — Mandates the SQL data type to use for a column in a table underlying a duality view definition. The column corresponds to the document field targeted by `path`. Field `value` is a string naming a scalar SQL data type (including `"json"` and `"vector"`, for types `JSON` and `VECTOR`). The data-type name is interpreted case-insensitively, and it can be any column type accepted by `CREATE TABLE`.

    * **`"key"`** — Mandates the identifying column or columns for the table underlying the document object (or array of objects) targeted by field `path`. Field `value` is either a *string* naming a scalar JSON field whose column is an identifying column, or it is an *array* of such field-name strings, each of whose column is an identifying column for the table (that is, together these columns uniquely identify a row; for an example, see: Car-Racing Example, Tables in *JSON-Relational Duality Developer's Guide*).

    * **`"normalize"`** — Mandates (if field `value` is `false`) that the JSON data targeted in documents by field `path` is *not* to be *shared*.

      This applies to all documents in the input document sets acted on by `infer_schema` and `infer_and_generate_schema` (field `hints` is used only by those functions). The targeting of document data is not specific to any particular kind of document. Any document, of any kind, with data that comes from columns in `table` and is targeted by the same `path` value, is affected.

      Field `path` must target an *object*, which can be at any level (including the root object of the document, which is targeted by path `$`). An error is raised if `path` targets a scalar or array value.

      In effect, the given table is locked/dedicated to data at the specified path, and vice versa. The table is not mapped to any *other data than that targeted by `path`*, anywhere in the input document sets. The data targeted by the given path (in any document) is thus *not shared* anywhere at a *different* path, either within the same document or in different documents. It *is* shared in all documents at the locations specified by path.

      Only the table underlying the targeted object is locked; sharing of data underlying a subobject within the targeted object is controlled by its own underlying table.

      If field `value` is `true` then this hint has no effect, since trying to normalize input JSON data is the default behavior.

  – **`table`** — A string naming an input table whose document-set data is used to define a duality view.

  - **`path`** — A SQL/JSON path expression string that targets data in input JSON documents. An error is raised if the path targets no data.
  - **`value`** — Information specific to the particular `type`, providing detail that defines the hint. (This is the only hint field whose value is not necessarily a string. For `normalize` it is a Boolean.)

  Field `hints` is used only for functions `infer_schema` and `infer_schema_and_generate`.

- **`ingestLimit`** (Optional) — The maximum number of documents to be analyzed in each document set. No error is raised if the limit is exceeded; the additional documents are simply not examined.

  The default value is 100,000.

  Field `ingestLimit` is used only for functions `infer_schema` and `infer_schema_and_generate`.

- **`minFieldFrequency`** (Optional) — The minimum frequency for a field *not* to be considered an outlier (high-entropy).

  A field is an **occurrence outlier** for a given document set if it occurs in less than `minFieldFrequency` percent of the documents. A value of zero (`0`) percent means that *no* fields are considered as outliers.

  For example, in the input course documents, if a value of `25` is used for `minFieldFrequency` then field `Notes` is an occurrence outlier because it occurs in less than 25% of the documents in the course document set.

  The *default* `minFieldFrequency` value is `5`, meaning that a field that occurs in less than 5% of an input document-set's documents is considered high-entropy.

  The converter does not map an occurrence-outlier field to any underlying column. When there are flex columns, the importer puts all fields (such as occurrence-outlier fields) that are not mapped to columns into the flex columns corresponding to the field locations.

  > **Note:**
  >
  > In the student-teacher-course examples presented in this documentation, which involve very few documents in each document set, we use `25` as the `minFieldFrequency` value, in order to demonstrate the determination and handling of occurrence outliers.

- **`minTypeFrequency`** (Optional) — The minimum frequency for the type of a field's value *not* to be considered an outlier (high-entropy).

  A field is an **type-occurrence outlier**, or **type outlier**, for a given document set if any of its values occurs with a given type in less than `minTypeFrequency` percent of the documents. A value of zero (0) percent means that *no* fields are considered as outliers.

  For example, in the input course documents, if a value of `15` is used for `minTypeFrequency` then student field `age` is a type outlier because it has a *string* value in 10% (less than 15%) of the documents. (It has a *number* value in the other documents.)

  The *default* `minTypeFrequency` value is `5`, meaning that a field has a given type in less than 5% of an input document-set's documents is considered an outlier.

  The importer tries to convert a value of rare type to the common type for the field. For example, if the common type for a `length` field is `number` then a `length` occurrence with a value of `"42"` is converted to the number `42`. If the conversion attempt fails then an error is logged for that occurrence.

Note:

In the examples presented here, which involve very few documents in each document set, we use `15` as the `minTypeFrequency` value, in order to demonstrate the determination and handling of type outliers.

- **`normalize`** (Optional) — A Boolean value (`true`/`false`) that indicates whether the converter should try to normalize (share) the relational tables it infers. A `false` value means that each object in a document supported by a duality view has its own underlying table, that is, a table that's *not shared* with any other duality view.

  The default value is `true`.

  Field `normalize` is used only for functions `infer_schema` and `infer_schema_and_generate`.

  Note that this *top-level* configuration field `normalize` applies to the general converter behavior, for *all* tables and duality views being generated. On the other hand, field `normalize` in a `hints` field's object provides a more fine-grained prevention of sharing, the sharing of a table that underlies a particular document object.

- **`outputFormat`** (Optional) — A string whose value defines the format of the output data definition language (DDL) script.

  The default value is **`"executable"`**, which means you can execute the DDL script directly: it uses PL/SQL `EXECUTE IMMEDIATE`. The other possible value is **`"standalone"`**, which means you can use the DDL script in a SQL script that you run separately.

  Field `outputFormat` is used only for functions `infer_schema` and `infer_schema_and_generate`.

  If the generated DDL is larger than 32K bytes then you *must* use `"standalone`; otherwise, an error is raised when `EXECUTE IMMEDIATE` is invoked. An `"executable"` DDL script can be too large if the input data sets are themselves very large or if they have many levels of nested values.

- **`rejectLimit`** (Optional) — The maximum number of errors that can be logged. If this limit is exceeded then the import operation is canceled (fails) and is rolled back, so no error logs are available. By default there is no limit.

  Field `rejectLimit` is used only for procedure `import_all`.

- **`softnessThreshold`** (Optional) — The minimum cleanliness level allowed for input data. The default value is `99`, meaning that at least 99% of the input documents must not have missing or incorrect information.

  Field `softnessThreshold` is used only for functions `infer_schema` and `infer_schema_and_generate`.

- **`sourceSchema`** (Optional) — A string whose value is the name of the database schema (user) that owns the input tables (`tableNames`).

  If not provided then the database schema used to identify the input tables is the one that's current when the DDL is *generated* (not when it is executed).

- **`tableNames`** (*Required*) — An array of strings naming the Oracle Database transfer tables that correspond to the original external document sets. Each table must have a `JSON`-type column (it need not be named `data`), which stores the documents of a given document set.

If field `viewNames` is provided then its array length must be the same as that of field `tableNames`; otherwise, an error is raised (not logged).

- **`tablespace`** (Optional) — A string whose value is the name of the tablespace to use for all of the tables underlying the duality views.

  If not provided then no tablespace is specified in the output DDL. This means that the tablespace used is the one that's current at the time the DDL code is *executed* (not when it is generated).

  Field `tablespace` is used only for functions `infer_schema` and `infer_schema_and_generate`.

- **`targetSchema`** (Optional) — A string whose value is the name of the database schema (user) that will own the output database views (`viewNames`).

  If not provided then no database schema is specified in the output DDL; the names of the database objects to be created are unqualified. This means that the schema used is the one that's current at the time the DDL code is *executed* (not when it is generated).

- **`updatability`** (Optional) — A Boolean value determining whether the duality views to be generated are to be updatable (`true`) or not (`false`). When `true`, annotations (for an example, see Annotations (NO)UPDATE, (NO)INSERT, (NO)DELETE, To Allow/Disallow Updating Operations) are set for maximum updatability of each view. When `false` all of the views created are read-only.

  The default value is `true`.

  Field `updatability` is used only for functions `infer_schema` and `infer_schema_and_generate`.

- **`useFlexFields`** (Optional) — A Boolean value determining whether flex columns are to be added to the tables underlying the duality views. Flex columns are used at application runtime to store unrecognized fields in an incoming document to be inserted or updated.

  If `useFlexFields` is `true`, then for each duality view *`<view-name>`*, a flex column named **`ora$`***`<view-name>`***`_flex`** is added to *each* table that directly underlies the top-level fields of an object in the supported documents. (The fields stored in a given flex column are unnested to that object.)

  The default value is `true`.

  Field `useFlexFields` is used only by converter functions `infer_schema` and `infer_schema_and_generate`.

  The importer doesn't use field `useFlexFields`. But when flex columns have been created by the converter, the *importer puts all fields that are not mapped to columns into flex columns* corresponding to the field locations. For example, occurrence-outlier fields are handled this way. If there are no flex columns then the importer reports an error for an unmapped field.

- **`viewNames`** (Optional) — An array of strings naming the duality views to be created, one for each document set.

  If not provided then the `tableNames` with `_duality` appended are used as the view names. For example the name of the view corresponding to the documents in table `foo` defaults to `foo`**`_duality`**.

  If field `viewNames` is provided then its array length must be the same as that of field `tableNames`; otherwise, an error is raised (not logged).

21.5 School Administration Example, Migrator Input Documents

Existing student, teacher, and course document sets comprise the JSON-to-duality migrator input for the school-administration example. In a typical migration scenario each might be received in the form of a JSON dump file from another database.

Note:

The document sets in the examples here are *very small*. In order to demonstrate the handling of outlier (high-entropy) fields, in examples here we use large values for migrator configuration fields `minFieldFrequency` (value `25`) and `minTypeFrequency` (value `15`), instead of the default value of `5`.

A field is an **occurrence outlier** for a given document set if it occurs in less than `minFieldFrequency` percent of the documents.

A field is a **type outlier** for a given document set if any of its values occurs with a given type in less than `minTypeFrequency` percent of the documents.

- An **occurrence-outlier** field (a field that occurs rarely) is not mapped by the *converter* to any underlying column. If the converter produces flex columns (configuration field `useFlexFields` = `true`, the default value), then the *importer* places an unmapped field in a flex column of a table underlying the duality view. If there are no flex columns then the importer reports an unmapped field in an import error log, and the field is not supported in the duality view.
- A **type-outlier** field (a field whose value is rarely of a different type than usual) is handled differently. Import tries to convert any values of a rare type to the expected type for the field. Unsuccessful conversion is reported in an import error log, and the field is not used in the duality view.

Example 21-1 Student Document Set (Migrator Input)

These are the student documents that we assume comprise an existing external document set that serves as input to the JSON-to-duality migrator.

```
{"studentId" : 1,
 "name"      : "Donald P.",
 "age"       : 20,
 "advisorId" : 102,
 "courses"   : [ {"courseNumber" : "CS101",
                  "name"         : "Algorithms",
                  "avgGrade"     : 75},
                 {"courseNumber" : "CS102",
                  "name"         : "Data Structures",
                  "avgGrade"     : "TBD"},
                 {"courseNumber" : "MATH101",
                  "name"         : "Algebra",
                  "avgGrade"     : 90} ],
 "dormitory" : {"dormId" : 201, "dormName" : "ABC"}}

{"studentId" : 2,
 "name"      : "Elena H.",
 "age"       : 21,
 "advisorId" : 103,
 "courses"   : [ {"courseNumber" : "CS101",
                  "name"         : "Algorithms",
                  "avgGrade"     : 75},
                 {"courseNumber" : "CS102",
                  "name"         : "Data Structures",
                  "avgGrade"     : "TBD"},
                 {"courseNumber" : "MATH102",
                  "name"         : "Calculus",
                  "avgGrade"     : 95} ],
 "dormitory" : {"dormId" : 202, "dormName" : "XYZ"}}

{"studentId" : 3,
 "name"      : "Francis K.",
 "age"       : 20,
 "advisorId" : 103,
 "courses"   : [ {"courseNumber" : "MATH103",
                  "name"         : "Advanced Algebra",
                  "avgGrade"     : 82} ],
 "dormitory" : {"dormId" : 204, "dormName" : "QWE"}}

{"studentId" : 4,
 "name"      : "Georgia D.",
 "age"       : 19,
 "advisorId" : 101,
 "courses"   : [ {"courseNumber" : "CS101",
                  "name"         : "Algorithms",
                  "avgGrade"     : 75},
                 {"courseNumber" : "MATH102",
                  "name"         : "Calculus",
                  "avgGrade"     : 95},
                 {"courseNumber" : "MATH103",
                  "name"         : "Advanced Algebra",
                  "avgGrade"     : 82} ],
 "dormitory" : {"dormId" : 203, "dormName" : "LMN"}}

{"studentId" : 5,
 "name"      : "Hye E.",
 "age"       : 21,
 "advisorId" : 103,
 "courses"   : [ {"courseNumber" : "CS102",
                  "name"         : "Data Structures",
                  "avgGrade"     : "TBD"},
                 {"courseNumber" : "MATH101",
                  "name"         : "Algebra",
                  "avgGrade"     : 90} ],
 "dormitory" : {"dormId" : 201, "dormName" : "ABC"}}

{"studentId" : 6,
 "name"      : "Ileana D.",
 "age"       : 21,
 "advisorId" : 102,
 "courses"   : [ {"courseNumber" : "MATH103",
                  "name"         : "Advanced Algebra",
                  "avgGrade"     : 82} ],
 "dormitory" : {"dormId" : 205, "dormName" : "GHI"}}

{"studentId" : 7,
 "name"      : "Jatin S.",
 "age"       : 20,
 "advisorId" : 101,
 "courses"   : [ {"courseNumber" : "CS101",
                  "name"         : "Algorithms",
                  "avgGrade"     : 75},
                 {"courseNumber" : "CS102",
                  "name"         : "Data Structures",
                  "avgGrade"     : "TBD"} ],
 "dormitory" : {"dormId" : 204, "dormName" : "QWE"}}

{"studentId" : 8,
 "name"      : "Katie H.",
 "age"       : 21,
 "advisorId" : 102,
 "courses"   : [ {"courseNumber" : "CS102",
                  "name"         : "Data Structures",
                  "avgGrade"     : "TBD"},
                 {"courseNumber" : "MATH103",
                  "name"         : "Advanced Algebra",
                  "avgGrade"     : 82} ],
 "dormitory" : {"dormId" : 205, "dormName" : "GHI"}}

{"studentId" : 9,
 "name"      : "Luis F.",
 "age"       : "Nineteen",
 "advisorId" : 101,
 "courses"   : [ {"courseNumber" : "CS101",
                  "name"         : "Algorithms",
                  "avgGrade"     : 75},
                 {"courseNumber" : "MATH102",
                  "name"         : "Calculus",
                  "avgGrade"     : 95},
                 {"courseNumber" : "MATH103",
                  "name"         : "Advanced Algebra",
                  "avgGrade"     : 82} ],
 "dormitory" : {"dormId" : 201, "dormName" : "ABC"}}

{"studentId" : 10,
 "name"      : "Ming L.",
 "age"       : 20,
 "advisorId" : 101,
 "courses"   : [ {"courseNumber" : "MATH102",
                  "name"         : "Calculus",
                  "avgGrade"     : 95} ],
 "dormitory" : {"dormId" : 202, "dormName" : "XYZ"}}
```

Notice these two fields, in particular:

- Field `age` is of a mixed type: number and string. In one of the ten documents (10%) its value is a string (`"Nineteen"`); in the others (90%) the value is a number.

- Field `avgGrade` is of a mixed type: number and string. In all ten documents (100%) at least one of its occurrences has a number value. In five of the ten documents (50%) at least one of its occurrences has a string value (`"TBD"`).

You might want to consider field `age` to be a **type outlier**, because you consider that you normally expect its value to be a number but the field occurs rarely with a string value. The migrator lets you decide the occurrence frequencies to consider "rare", and thus handle such fields specially (with configuration fields `minFieldFrequency` and `minTypeFrequency`).

Example 21-2 Teacher Document Set (Migrator Input)

These are the teacher documents that we assume comprise an existing external document set that serves as input to the JSON-to-duality migrator.

```
{"_id"              : 101,
 "name"             : "Abdul J.",
 "phoneNumber"      : [ "222-555-011", "222-555-012" ],
 "salary"           : 200000,
 "department"       : "Mathematics",
 "coursesTaught"    : [ {"courseId"  : "MATH101",
                         "name"      : "Algebra",
                         "classType" : "Online"},
                        {"courseId"  : "MATH102",
                         "name"      : "Calculus",
                         "classType" : "In-person"} ],
 "studentsAdvised" : [ {"studentId" : 4,  "name"  : "Georgia D.", "dormId" : 203},
                       {"studentId" : 7,  "name"  : "Jatin S.",   "dormId" : 204},
                       {"studentId" : 9,  "name"  : "Luis F.",    "dormId" : 201},
                       {"studentId" : 10, "name"  : "Ming L.",    "dormId" : 202} ]}

{"_id"             : 102,
 "name"            : "Betty Z.",
 "phoneNumber"     : "222-555-022",
 "salary"          : 300000,
 "department"      : "Computer Science",
 "coursesTaught"   : [ {"courseId"  : "CS101",
                        "name"      : "Algorithms",
                        "classType" : "Online"},
                       {"courseId"  : "CS102",
                        "name"      : "Data Structures",
                        "classType" : "In-person"} ],
 "studentsAdvised" : [ {"studentId" : 1, "name" : "Donald P.", "dormId" : 201},
                       {"studentId" : 6, "name" : "Ileana D.", "dormId" : 205},
                       {"studentId" : 8, "name" : "Katie H.",  "dormId" : 205} ]}

{"_id"              : 103,
 "name"             : "Colin J.",
 "phoneNumber"      : [ "222-555-023" ],
 "salary"           : 220000,
 "department"       : "Mathematics",
 "coursesTaught"    : [ {"courseId"  : "MATH103",
                         "name"      : "Advanced Algebra",
                         "classType" : "Online"} ],
 "studentsAdvised" : [ {"studentId" : 2, "name" : "Elena H.",   "dormId" : 202},
                       {"studentId" : 3, "name" : "Francis K.", "dormId" : 204},
                       {"studentId" : 5, "name" : "Hye E.",     "dormId" : 201} ]}
```

```
{"_id"              : 104,
 "name"             : "Natalie C.",
 "phoneNumber"      : "222-555-044",
 "salary"           : 180000,
 "department"       : "Computer Science",
 "coursesTaught"    : [],
 "studentsAdvised"  : []}
```

Field phoneNumber is of a mixed type: string and array (array of strings). In two of the four documents (50%) its value is a string; in the other two the value is an array of strings.

(Fields coursesTaught and studentsAdvised each have one occurrence whose value is the *empty* array.)

Example 21-3   Course Document Set (Migrator Input)

These are the course documents that we assume comprise an existing external document set that serves as input to the JSON-to-duality migrator.

```
{"courseId"         : "MATH101",
 "name"             : "Algebra",
 "creditHours"      : 3,
 "students"         : [ {"studentId" : 1, "name" : "Donald P."},
                        {"studentId" : 5, "name" : "Hye E."} ],
 "teacher"          : {"teacherId" : 104, "name" : "Abdul J."},
 "Notes"            : "Prerequisite for Advanced Algebra"}
{"courseId"         : "MATH102",
 "name"             : "Calculus",
 "creditHours"      : 4,
 "students"         : [ {"studentId" : 2,  "name" : "Elena H."},
                        {"studentId" : 4,  "name" : "Georgia D."},
                        {"studentId" : 9,  "name" : "Luis F."},
                        {"studentId" : 10, "name" : "Ming L."} ],
 "teacher"          : {"teacherId" : 101,  "name" : "Abdul J."}}

{"courseId"         : "CS101",
 "name"             : "Algorithms",
 "creditHours"      : 5,
 "students"         : [ {"studentId" : 1, "name" : "Donald P."},
                        {"studentId" : 2, "name" : "Elena H."},
                        {"studentId" : 4, "name" : "Georgia D."},
                        {"studentId" : 7, "name" : "Jatin S."},
                        {"studentId" : 9, "name" : "Luis F."} ],
 "teacher"          : {"teacherId" : 102, "name" : "Betty Z."}}

{"courseId"         : "CS102",
 "name"             : "Data Structures",
 "creditHours"      : 3,
 "students"         : [ {"studentId" : 1, "name" : "Donald P."},
                        {"studentId" : 2, "name" : "Elena H."},
                        {"studentId" : 5, "name" : "Hye E."},
                        {"studentId" : 7, "name" : "Jatin S."},
                        {"studentId" : 8, "name" : "Katie H."} ],
 "teacher"          : {"teacherId" : 102, "name" : "Betty Z."}}

{"courseId"         : "MATH103",
"name"              : "Advanced Algebra",
"creditHours"       : "3",
"students"          : [ {"studentId" : 3, "name" : "Francis K."},
                        {"studentId" : 4, "name" : "Georgia D."},
                        {"studentId" : 6, "name" : "Ileana D."},
                        {"studentId" : 8, "name" : "Katie H."},
                        {"studentId" : 9, "name" : "Luis F."} ],
"teacher"           : {"teacherId" : 103, "name" : "Colin J."}}
```

Notice these two fields, in particular:

- Field `Notes` occurs in only one course document (one out of five, 20%).
- Field `creditHours` is of a mixed type: number and string. In one of the five documents (20%) its value is a *string*; in the others (80%) the value is a *number*.

You might want to consider field `Notes` to be an **occurrence outlier**, because you consider 20% occurrence to be rare, and you might want to consider field `creditHours` to be a **type outlier**, because it occurs rarely (20%) with a string value. The migrator lets you decide the occurrence frequencies to consider "rare", and thus handle such fields specially (with configuration fields `minFieldFrequency` and `minTypeFrequency`).

21.6 Before Using the Converter (1): Create Database Document Sets

Before using the JSON-to-duality converter you need to create `JSON`-type document sets in Oracle Database from the original external document sets. The input to the converter for each set of documents is an Oracle Database table with a single column of `JSON` data type.

You can export JSON document sets from a document database and import them into `JSON`-type columns using various tools provided by Oracle and document databases. (MongoDB command-line tools `mongoexport` and `mongoimport` provide one way to do this.)

We assume that each of the student, teacher, and course document sets has been thus loaded into a `JSON`-type column, `data`, of a temporary **transfer table** (e.g. `course_tab` for course documents) from a document-database dump file of documents of the given kind (e.g. course documents). This is shown in the following example:

Example 21-4 Create an Oracle Document Set (Course) From a JSON Dump File.

This example creates an Oracle Database external table, `dataset_course`, from a JSON dump file of a set of course documents, `course.json`. It then creates temporary transfer table `course_tab` with `JSON`-type column data. Finally, it imports the course documents into temporary transfer table `course_tab`, which can be used as input to the JSON-relational converter.

The documents in `course_tab.data` are those shown in the example "Course Document Set (Migrator Input)" in School Administration Example, Migrator Input Documents.

(Similarly student and teacher document sets are loaded into transfer tables `student_tab` and `teacher_tab` from external tables `dataset_student` and `dataset_teacher` created from dump files `student.json` and `teacher.json`, respectively.)

```
CREATE TABLE dataset_course (data JSON)
  ORGANIZATION EXTERNAL
```

```
    (TYPE ORACLE_BIGDATA
     ACCESS PARAMETERS (com.oracle.bigdata.fileformat = jsondoc)
     LOCATION (data_dir:'course.json'))
  PARALLEL
  REJECT LIMIT UNLIMITED;


CREATE TABLE course_tab AS SELECT * FROM dataset_course;
SELECT json_serialize(data PRETTY) FROM course_tab;


JSON_SERIALIZE(DATAPRETTY)
-----------------------------------
{
  "courseId" : "MATH101",
  "name" : "Algebra",
  "creditHours" : 3,
  "students" :
  [
    {
      "studentId" : 1,
      "name" : "Donald P."
    },
    {
      "studentId" : 5,
      "name" : "Hye E."
    }
  ],
  "teacher" :
  {
    "teacherId" : 101,
    "name" : "Abdul J."
  },
  "Notes" : "Prerequisite for Advanced Algebra"
}

{
  "courseId" : "MATH102",
  "name" : "Calculus",
  "creditHours" : 4,
  "students" :
  [
    {
      "studentId" : 2,
      "name" : "Elena H."
    },
    {
      "studentId" : 4,
      "name" : "Georgia D."
    },
    {
      "studentId" : 9,
      "name" : "Luis F."
    },
    {
      "studentId" : 10,
        "name" : "Ming L."
      }
    ],
    "teacher" :
    {
      "teacherId" : 101,
      "name" : "Abdul J."
    }
  }

  {
    "courseId" : "CS101",
    "name" : "Algorithms",
    "creditHours" : 5,
    "students" :
    [
      {
        "studentId" : 1,
        "name" : "Donald P."
      },
      {
        "studentId" : 2,
        "name" : "Elena H."
      },
      {
        "studentId" : 4,
        "name" : "Georgia D."
      },
      {
        "studentId" : 7,
        "name" : "Jatin S."
      },
      {
        "studentId" : 9,
        "name" : "Luis F."
      }
    ],
    "teacher" :
    {
      "teacherId" : 102,
      "name" : "Betty Z."
    }
  }

  {
    "courseId" : "CS102",
    "name" : "Data Structures",
    "creditHours" : 3,
    "students" :
    [
      {
        "studentId" : 1,
        "name" : "Donald P."
      },
      {
        "studentId" : 2,
        "name" : "Elena H."
      },
      {
        "studentId" : 5,
        "name" : "Hye E."
      },
      {
        "studentId" : 7,
        "name" : "Jatin S."
      },
      {
        "studentId" : 8,
        "name" : "Katie H."
      }
    ],
    "teacher" :
    {
      "teacherId" : 102,
      "name" : "Betty Z."
    }
  }

  {
    "courseId" : "MATH103",
    "name" : "Advanced Algebra",
    "creditHours" : "3",
    "students" :
    [
      {
        "studentId" : 3,
        "name" : "Francis K."
      },
      {
        "studentId" : 4,
        "name" : "Georgia D."
      },
      {
        "studentId" : 6,
        "name" : "Ileana D."
      },
      {
        "studentId" : 8,
        "name" : "Katie H."
      },
      {
        "studentId" : 9,
        "name" : "Luis F."
      }
    ],
    "teacher" :
    {
      "teacherId" : 103,
      "name" : "Colin J."
    }
  }
```

Note:

Oracle Database supports the use of textual JSON **objects** that represent nonstandard-type scalar JSON values. For example, the **extended object** `{"$numberDecimal" : 31}` represents a JSON scalar value of the nonstandard type **decimal number**, and when interpreted as such it is replaced by a decimal number in Oracle's native binary JSON format, OSON.

Some non-Oracle databases also use such extended objects. If such an external extended object is a format recognized by Oracle then, when the JSON data is loaded (ingested), the extended object is replaced by the corresponding Oracle scalar JSON value. If the format isn't supported by Oracle then the extended object is retained as such, that is, as an object.

See Textual JSON Objects That Represent Extended Scalar Values in *Oracle Database JSON Developer's Guide* for information about Oracle support for extended objects.

See Also:

- Migrate Application Data from MongoDB to Oracle Database in *Oracle Database API for MongoDB* for information about using commands `mongoexport` and `mongoimport` to migrate
- Loading External JSON Data in *Oracle Database JSON Developer's Guide* for loading data from a document-database dumpfile into Oracle Database

21.7 Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas

A data-guide JSON schema provides frequency information about the fields in a document set, in addition to structure and type information. You can use such schemas to get an idea how migration might proceed, and you can compare them with other JSON schemas as a shortcut for comparing document sets.

A **JSON data guide** created using keyword `FORMAT_SCHEMA` is a special kind of JSON schema that includes not only the usual document structure and type information but also statistical information about the specific content; in particular, for each field, in what *percentage of documents it occurs*, in what *percentage of documents it has values of which types*, and the *range of values for each type*.

Creating data-guide JSON schemas for your input document sets is *optional*[2], and you can create them at any time (as long as you still have the transfer tables of input documents). But it's a good idea to create them before starting to convert your input document sets, in particular because they can help guide how you configure the converter.

[2] Transfer tables for your input document sets are all you need, to use the JSON-To-Duality converter.

Example 21-5 Create JSON Data Guides For Input Document Sets

This example uses Oracle SQL function `json_dataguide` to create data guides for the input student, teacher, and course document sets. These are JSON schemas that can be used to validate their documents.

Parameter `DBMS_JSON.`**`FORMAT_SCHEMA`** ensures that the data guide is usable for validating. Parameter `DBMS_JSON.`**`PRETTY`** pretty-prints the result. Parameter `DBMS_JSON.`**`GATHER_STATS`** provides the data guide with statistical fields such as **`o:frequency`**, which specifies the percentage of documents in which a given field occurs or has a given type of value.

```
SELECT json_dataguide(data,
                      DBMS_JSON.FORMAT_SCHEMA,
                      DBMS_JSON.PRETTY+DBMS_JSON.GATHER_STATS)
  FROM student_tab;

SELECT json_dataguide(data,
                      DBMS_JSON.FORMAT_SCHEMA,
                      DBMS_JSON.PRETTY+DBMS_JSON.GATHER_STATS)
  FROM teacher_tab;

SELECT json_dataguide(data,
                      DBMS_JSON.FORMAT_SCHEMA,
                      DBMS_JSON.PRETTY+DBMS_JSON.GATHER_STATS)
  FROM course_tab;
```

See Also:

- JSON_DATAGUIDE in *Oracle Database SQL Language Reference*
- DBMS_JSON Constants in *Oracle Database PL/SQL Packages and Types Reference* for information about constants `DBMS_JSON.FORMAT_SCHEMA`, `DBMS_JSON.GATHER_STATS`, and `DBMS_JSON.PRETTY`

The resulting data guides for input student document set, input teacher document set, and input course documentation set are shown in the JSON data guide examples that follow in this topic, "JSON Data Guide for Input Student Document Set", "JSON Data Guide for Input Teacher Document Set", and "JSON Data Guide for Input Course Document Set", which describe the documents in `JSON`-type column `data` of tables `student_tab`, `teacher_tab`, and `course_tab`, respectively.

Comparing JSON schemas can serve as a useful *proxy for comparing entire document sets* — in particular, migration input document sets versus output document collections supported by duality views (proposed during migration or fully created).

- As the first conversion step, PL/SQL function `DBMS_JSON_DUALITY.infer_schema` produces a JSON schema that describes the entire proposed relational schema for a migration, that is, the proposed duality views plus their underlying tables.

  The JSON schema produced by function `infer_schema` is *not* a data-guide JSON schema — there are no duality views yet, so there are no supported document collections from which statistical information can be gathered. But it does specify the structure and typing of the document sets that could result from a migration.

You can use the view parts of this JSON schema to compare against JSON schemas for input document sets (in a transfer table).

For example, instead of comparing the individual input documents in table `course_tab` with the individual documents to be supported by the (inferred) `course` duality view, you can compare the data-guide JSON schema from "JSON Data Guide for Input Course Document Set" in this topic with the `COURSE` duality-view part of the JSON schema inferred by `infer_schema` — see the section "JSON Schema from INFER_SCHEMA for Duality Vies with No Outliers" in Migrating To Duality, Simplified Recipe. When you do that, you can ignore fields that are relevant to only one or the other kind of JSON schema--for example, fields named with prefix "o:" (for Oracle) and fields named with prefix "db" (for database).

- Similarly, comparing a JSON schema for an input document set against a JSON schema for the created and populated duality view that replaces it after migration can highlight differences. For example, you can compare the data-guide JSON schema for the course input table in the example "JSON Data Guide For Import Course Document Set" against a data-guide JSON schema for the course duality view. A data-guide schema serves as a shortcut (proxy) for comparing the documents supported by a duality view with the corresponding input documents.

Example 21-6   JSON Data Guide For Input Student Document Set

This data guide summarizes the input set of student documents stored in transfer table `student_tab`.

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2024-12-30T18:12:41",
  "o:sample_size" : 10,
  "required" : true,
  "properties" :
  {
    "age" :
    {
      "oneOf" :
      [
        {
          "type" : "number",
          "o:preferred_column_name" : "age",
          "o:frequency" : 90,
          "o:low_value" : 19,
          "o:high_value" : 21,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2024-12-30T18:12:41",
          "o:sample_size" : 10,
          "maximum" : 21,
          "minimum" : 19
        },
        {
          "type" : "string",
          "o:length" : 8,
          "o:preferred_column_name" : "age",
          "o:frequency" : 10,
          "o:low_value" : "Nineteen",
          "o:high_value" : "Nineteen",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2024-12-30T18:12:41",
          "o:sample_size" : 10,
          "maxLength" : 8,
          "minLength" : 8
        }
      ]
    },
    "name" :
    {
      "type" : "string",
      "o:length" : 16,
      "o:preferred_column_name" : "name",
      "o:frequency" : 100,
      "o:low_value" : "Donald P.",
      "o:high_value" : "Ming L.",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2024-12-30T18:12:41",
      "o:sample_size" : 10,
      "required" : true,
      "maxLength" : 10,
      "minLength" : 6
    },
    "courses" :
    {
      "type" : "array",
      "o:preferred_column_name" : "courses",
      "o:frequency" : 100,
      "o:last_analyzed" : "2024-12-30T18:12:41",
      "o:sample_size" : 10,
      "required" : true,
      "items" :
      {
        "properties" :
        {
          "name" :
          {
            "type" : "string",
            "o:length" : 16,
            "o:preferred_column_name" : "name",
            "o:frequency" : 100,
            "o:low_value" : "Advanced Algebra",
            "o:high_value" : "Data Structures",
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2024-12-30T18:12:41",
            "o:sample_size" : 10,
            "required" : true,
            "maxLength" : 16,
            "minLength" : 7
          },
          "avgGrade" :
          {
            "oneOf" :
            [
              {
                "type" : "number",
                "o:preferred_column_name" : "avgGrade",
                "o:frequency" : 100,
                "o:low_value" : 75,
                "o:high_value" : 95,
                "o:num_nulls" : 0,
                "o:last_analyzed" : "2024-12-30T18:12:41",
                "o:sample_size" : 10,
                "required" : true,
                "maximum" : 95,
                "minimum" : 75
              },
              {
                "type" : "string",
                "o:length" : 4,
                "o:preferred_column_name" : "avgGrade",
                "o:frequency" : 50,
                "o:low_value" : "TBD",
                "o:high_value" : "TBD",
                "o:num_nulls" : 0,
                "o:last_analyzed" : "2024-12-30T18:12:41",
                "o:sample_size" : 10,
                "maxLength" : 3,
                "minLength" : 3
              }
            ]
          },
          "courseNumber" :
          {
            "type" : "string",
            "o:length" : 8,
            "o:preferred_column_name" : "courseNumber",
            "o:frequency" : 100,
            "o:low_value" : "CS101",
            "o:high_value" : "MATH103",
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2024-12-30T18:12:41",
            "o:sample_size" : 10,
            "required" : true,
            "maxLength" : 7,
            "minLength" : 5
          }
        }
      }
    },
    "advisorId" :
    {
      "type" : "number",
      "o:preferred_column_name" : "advisorId",
      "o:frequency" : 100,
      "o:low_value" : 101,
      "o:high_value" : 103,
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2024-12-30T18:12:41",
      "o:sample_size" : 10,
      "required" : true,
      "maximum" : 103,
              "minimum" : 101
            },
            "dormitory" :
            {
              "type" : "object",
              "o:preferred_column_name" : "dormitory",
              "o:frequency" : 100,
              "o:last_analyzed" : "2024-12-30T18:12:41",
              "o:sample_size" : 10,
              "required" : true,
              "properties" :
              {
                "dormId" :
                {
                  "type" : "number",
                  "o:preferred_column_name" : "dormId",
                  "o:frequency" : 100,
                  "o:low_value" : 201,
                  "o:high_value" : 205,
                  "o:num_nulls" : 0,
                  "o:last_analyzed" : "2024-12-30T18:12:41",
                  "o:sample_size" : 10,
                  "required" : true,
                  "maximum" : 205,
                  "minimum" : 201
                },
                "dormName" :
                {
                  "type" : "string",
                  "o:length" : 4,
                  "o:preferred_column_name" : "dormName",
                  "o:frequency" : 100,
                  "o:low_value" : "ABC",
                  "o:high_value" : "XYZ",
                  "o:num_nulls" : 0,
                  "o:last_analyzed" : "2024-12-30T18:12:41",
                  "o:sample_size" : 10,
                  "required" : true,
                  "maxLength" : 3,
                  "minLength" : 3
                }
              }
            },
            "studentId" :
            {
              "type" : "number",
              "o:preferred_column_name" : "studentId",
              "o:frequency" : 100,
              "o:low_value" : 1,
              "o:high_value" : 10,
              "o:num_nulls" : 0,
              "o:last_analyzed" : "2024-12-30T18:12:41",
              "o:sample_size" : 10,
              "required" : true,
              "maximum" : 10,
              "minimum" : 1
```

```
      }
    }
  }
```

Field `age` has a `type` that is *either* (1) a `number`, with `o:frequency` **`90`**, or (2) a `string`, with `o:frequency` **`10`**. This means that a *numeric* age appears in 90% of the documents, and a *string* grade appears in 10% of the documents. Field `age` is thus a *mixed-type* field.

Similarly, field `avgGrade` is a *mixed-type* field, with a *numeric* grade in 100% of the documents, and a *string* grade in 50% of the documents.

Converter configuration fields `min`*`Field`*`Frequency` and `min`*`Type`*`Frequency` test the percentage of documents where a *field*, or a field of a given *type*, respectively, is present *across the document set*.

All of the fields in the student documents are present in 100% of the documents, so none of them can be occurrence outliers, regardless of the value of `min`*`Field`*`Frequency`.

If the converter is used with a value of `15` for configuration field `min`*`Type`*`Frequency` then field `age` will be considered a *type-occurence* outlier, because it occurs with a string value in only 10% of the student documents (10 < 15). Field `avgGrade` will not be considered a type-occurrence outlier, because neither of its types is used in less than 15% of the student documents.

As a type-occurrence outlier, field `age` would be mapped by the converter to a column with SQL type `NUMBER` (JSON number type being dominant for the field). Then the importer would try, and fail, to convert the string value `"Nineteen"` to a number, and would log an error for the document where `age` is `"Nineteen"`.

A field that doesn't occur rarely but has a type that occurs rarely is not removed from the data.

Because there is no SQL data type of number-or-string, non-outlier mixed-type field `avgGrade` will be mapped by the converter to a *`JSON`-type* column, and it will apply a JSON schema to that column as a *validating check constraint*, to *require* the value to always be either a string or a number.

Example 21-7   JSON Data Guide For Input Teacher Document Set

This data guide summarizes the input set of teacher documents stored in transfer table `teacher_tab`.

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2024-12-30T18:12:41",
  "o:sample_size" : 4,
  "required" : true,
  "properties" :
  {
    "_id" :
    {
      "type" : "number",
      "o:preferred_column_name" : "_id",
      "o:frequency" : 100,
      "o:low_value" : 101,
      "o:high_value" : 104,
      "o:num_nulls" : 0,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 4,
    "required" : true,
    "maximum" : 104,
    "minimum" : 101
  },
  "name" :
  {
    "type" : "string",
    "o:length" : 16,
    "o:preferred_column_name" : "name",
    "o:frequency" : 100,
    "o:low_value" : "Abdul J.",
    "o:high_value" : "Natalie C.",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 4,
    "required" : true,
    "maxLength" : 10,
    "minLength" : 8
  },
  "salary" :
  {
    "type" : "number",
    "o:preferred_column_name" : "salary",
    "o:frequency" : 100,
    "o:low_value" : 180000,
    "o:high_value" : 300000,
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 4,
    "required" : true,
    "maximum" : 300000,
    "minimum" : 180000
  },
  "department" :
  {
    "type" : "string",
    "o:length" : 16,
    "o:preferred_column_name" : "department",
    "o:frequency" : 100,
    "o:low_value" : "Computer Science",
    "o:high_value" : "Mathematics",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 4,
    "required" : true,
    "maxLength" : 16,
    "minLength" : 11
  },
  "phoneNumber" :
  {
    "oneOf" :
    [
      {
        "type" : "string",
      "o:length" : 16,
      "o:preferred_column_name" : "phoneNumber",
      "o:frequency" : 50,
      "o:low_value" : "222-555-022",
      "o:high_value" : "222-555-044",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2024-12-30T18:12:41",
      "o:sample_size" : 4,
      "maxLength" : 11,
      "minLength" : 11
    },
    {
      "type" : "array",
      "o:preferred_column_name" : "phoneNumber",
      "o:frequency" : 50,
      "o:last_analyzed" : "2024-12-30T18:12:41",
      "o:sample_size" : 4,
      "items" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "scalar_string",
        "o:frequency" : 50,
        "o:low_value" : "222-555-011",
        "o:high_value" : "222-555-023",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2024-12-30T18:12:41",
        "o:sample_size" : 4,
        "maxLength" : 11,
        "minLength" : 11
      }
    }
  ]
},
"coursesTaught" :
{
  "type" : "array",
  "o:preferred_column_name" : "coursesTaught",
  "o:frequency" : 100,
  "o:last_analyzed" : "2024-12-30T18:12:41",
  "o:sample_size" : 4,
  "required" : true,
  "items" :
  {
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "name",
        "o:frequency" : 75,
        "o:low_value" : "Advanced Algebra",
        "o:high_value" : "Data Structures",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2024-12-30T18:12:41",
        "o:sample_size" : 4,
        "maxLength" : 16,
        "minLength" : 7
      },
      "courseId" :
      {
        "type" : "string",
        "o:length" : 8,
        "o:preferred_column_name" : "courseId",
        "o:frequency" : 75,
        "o:low_value" : "CS101",
        "o:high_value" : "MATH103",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2024-12-30T18:12:41",
        "o:sample_size" : 4,
        "maxLength" : 7,
        "minLength" : 5
      },
      "classType" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "classType",
        "o:frequency" : 75,
        "o:low_value" : "In-person",
        "o:high_value" : "Online",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2024-12-30T18:12:41",
        "o:sample_size" : 4,
        "maxLength" : 9,
        "minLength" : 6
      }
    }
  }
},
"studentsAdvised" :
{
  "type" : "array",
  "o:preferred_column_name" : "studentsAdvised",
  "o:frequency" : 100,
  "o:last_analyzed" : "2024-12-30T18:12:41",
  "o:sample_size" : 4,
  "required" : true,
  "items" :
  {
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "name",
        "o:frequency" : 75,
        "o:low_value" : "Donald P.",
        "o:high_value" : "Ming L.",
        "o:num_nulls" : 0,
                "o:last_analyzed" : "2024-12-30T18:12:41",
                "o:sample_size" : 4,
                "maxLength" : 10,
                "minLength" : 6
              },
              "dormId" :
              {
                "type" : "number",
                "o:preferred_column_name" : "dormId",
                "o:frequency" : 75,
                "o:low_value" : 201,
                "o:high_value" : 205,
                "o:num_nulls" : 0,
                "o:last_analyzed" : "2024-12-30T18:12:41",
                "o:sample_size" : 4,
                "maximum" : 205,
                "minimum" : 201
              },
              "studentId" :
              {
                "type" : "number",
                "o:preferred_column_name" : "studentId",
                "o:frequency" : 75,
                "o:low_value" : 1,
                "o:high_value" : 10,
                "o:num_nulls" : 0,
                "o:last_analyzed" : "2024-12-30T18:12:41",
                "o:sample_size" : 4,
                "maximum" : 10,
                "minimum" : 1
              }
            }
          }
        }
      }
    }
```

Field `phoneNumber` has a `type` that is either (1) a `string` or (2) an `array` of `string`s. Each of those types occurs in 50% (`o:frequency` = `50`) of the teacher documents. It is thus a mixed-type field. If the converter is used with a `minTypeFrequency` value of `15` then it will not be considered a type-occurrence outlier.

Because there is no SQL data type of string-or-array-of-strings, (non-outlier) mixed-type field `phoneNumber` will be mapped by the converter to a *`JSON`-type* column, and it will apply a JSON schema to that column as a *validating check constraint*, to *require* the value to always be either a string or an array of strings.

Example 21-8   JSON Data Guide For Input Course Document Set

This data guide summarizes the input set of course documents stored in transfer table `course_tab`.

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2024-12-30T18:12:41",
"o:sample_size" : 5,
"required" : true,
"properties" :
{
  "name" :
  {
    "type" : "string",
    "o:length" : 16,
    "o:preferred_column_name" : "name",
    "o:frequency" : 100,
    "o:low_value" : "Advanced Algebra",
    "o:high_value" : "Data Structures",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 5,
    "required" : true,
    "maxLength" : 16,
    "minLength" : 7
  },
  "Notes" :
  {
    "type" : "string",
    "o:length" : 64,
    "o:preferred_column_name" : "Notes",
    "o:frequency" : 20,
    "o:low_value" : "Prerequisite for Advanced Algebra",
    "o:high_value" : "Prerequisite for Advanced Algebra",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 5,
    "maxLength" : 33,
    "minLength" : 33
  },
  "teacher" :
  {
    "type" : "object",
    "o:preferred_column_name" : "teacher",
    "o:frequency" : 100,
    "o:last_analyzed" : "2024-12-30T18:12:41",
    "o:sample_size" : 5,
    "required" : true,
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 8,
        "o:preferred_column_name" : "name",
        "o:frequency" : 100,
        "o:low_value" : "Abdul J.",
        "o:high_value" : "Colin J.",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2024-12-30T18:12:41",
        "o:sample_size" : 5,
        "required" : true,
        "maxLength" : 8,
      "minLength" : 8
    },
    "teacherId" :
    {
      "type" : "number",
      "o:preferred_column_name" : "teacherId",
      "o:frequency" : 100,
      "o:low_value" : 101,
      "o:high_value" : 103,
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2024-12-30T18:12:41",
      "o:sample_size" : 5,
      "required" : true,
      "maximum" : 103,
      "minimum" : 101
    }
  }
},
"courseId" :
{
  "type" : "string",
  "o:length" : 8,
  "o:preferred_column_name" : "courseId",
  "o:frequency" : 100,
  "o:low_value" : "CS101",
  "o:high_value" : "MATH103",
  "o:num_nulls" : 0,
  "o:last_analyzed" : "2024-12-30T18:12:41",
  "o:sample_size" : 5,
  "required" : true,
  "maxLength" : 7,
  "minLength" : 5
},
"students" :
{
  "type" : "array",
  "o:preferred_column_name" : "students",
  "o:frequency" : 100,
  "o:last_analyzed" : "2024-12-30T18:12:41",
  "o:sample_size" : 5,
  "required" : true,
  "items" :
  {
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "name",
        "o:frequency" : 100,
        "o:low_value" : "Donald P.",
        "o:high_value" : "Ming L.",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2024-12-30T18:12:41",
        "o:sample_size" : 5,
            "required" : true,
            "maxLength" : 10,
            "minLength" : 6
          },
          "studentId" :
          {
            "type" : "number",
            "o:preferred_column_name" : "studentId",
            "o:frequency" : 100,
            "o:low_value" : 1,
            "o:high_value" : 10,
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2024-12-30T18:12:41",
            "o:sample_size" : 5,
            "required" : true,
            "maximum" : 10,
            "minimum" : 1
          }
        }
      }
    },
    "creditHours" :
    {
      "oneOf" :
      [
        {
          "type" : "number",
          "o:preferred_column_name" : "creditHours",
          "o:frequency" : 80,
          "o:low_value" : 3,
          "o:high_value" : 5,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2024-12-30T18:12:41",
          "o:sample_size" : 5,
          "maximum" : 5,
          "minimum" : 3
        },
        {
          "type" : "string",
          "o:length" : 1,
          "o:preferred_column_name" : "creditHours",
          "o:frequency" : 20,
          "o:low_value" : "3",
          "o:high_value" : "3",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2024-12-30T18:12:41",
          "o:sample_size" : 5,
          "maxLength" : 1,
          "minLength" : 1
        }
      ]
    }
  }
}
```

Field `Notes` occurs in only 20% of the documents (field `o:frequency` is `20`). If the converter is used with a value of `25` for configuration field `minFieldFrequency` then field `Notes` is considered an *occurrence outlier*, and the *converter will not map it to any column*.

However, if configuration field `useFlexFields` is `true` (the default) then the converter creates flex columns, and when flex columns exist the *importer places all unmapped fields into flex columns*. In that default (`useFlexFields` = `true`) case, field `Notes` will therefore be supported (by a flex column) after migration. If the converter is used with `useFlexFields` = `false` then the importer will log an error for rare field `Notes`.

Field `creditHours` has a `type` that is either (1) a `number`, with `o:frequency` = `80`, or (2) a `string`, with `o:frequency` = `20`. It is thus a mixed-type field. If the converter is used with a `minTypeFrequency` value of `15` then it will *not* be considered a type-occurrence outlier.

Because there is no SQL data type of number-or-string, (non-outlier) mixed-type field `creditHours` will be mapped by the converter to a *`JSON`-type* column, and it will apply a JSON schema to that column as a *validating check constraint*, to *require* the value to always be either a string or a number.

See Also:

- Validating JSON Documents with a JSON Schema in *Oracle Database JSON Developer's Guide* for information about using JSON schemas to constrain or validate JSON data
- JSON Data Guide in *Oracle Database JSON Developer's Guide*
- JSON_DATAGUIDE in *Oracle Database SQL Language Reference*
- json-schema.org for information about JSON Schema

21.8 JSON-To-Duality Converter: What It Does

The converter infers the inherent structure and typing of one or more sets of stored documents, as a JSON schema. Using the schema, the converter generates DDL code to create the database objects needed to support the document sets: duality views and their underlying tables and indexes.

The JSON schema inferred from the input document sets includes a *relational schema* that represents the relations (tables, columns, and key constraints) that implicitly underlie the data in the JSON documents.

The generated DDL code creates the appropriate duality views; their underlying tables; primary, unique, and foreign key constraints; indexes; and default values — everything needed to support the original document sets.

In some cases the converter creates fields and columns for a duality view definition that are not in the original document set.

- Document-identifer field **`_id`** is generated for each document, if it is not already present in the input documents.

  A duality view must have a top-level `_id` field (the document identifier), which corresponds to the identifying column(s) of the view's root table (primary-key columns, identity columns, or columns with a unique constraint or unique index). If a document input to the converter *already* has a top-level `_id` field, then its associated columns are in the root table and are chosen as the table's identifying columns.

- Document-handling field **`_metadata`** is generated and maintained for each document, to record its content-hash version (ETAG) and its latest system change number (SCN). This field is not part of the document content per se (payload) .
- Other generated field and column names always have the prefix **`ora$`**.

A duality view definition needs explicit fields for the identifying columns of each of its underlying tables, and this is another case where new fields are sometimes added.

This is the case for views `course` and `student`, which use an underlying *mapping table*, `map_table_course_root_to_student_root`, which has two identifying columns, `map_course_id` and `map_student_id`. These have foreign-key references to the identifying columns, `course_id` and `student_id`, of the course and student tables, `course_root` and `student_root`.

At the place where the mapping table is used in the view definitions, each of its identifying columns (`map_course_id` and `map_student_id`) must be present, with a field assigned to it. These fields are present in the documents supported by the view. The converter uses prefix `ora$` for their names, with the remainder taken from the column names (converted to camelCase, without underscore separators): `ora$mapCourseId` and `ora$mapStudentId`.

When configuration field `useFlexFields` is `true`, the converter adds flex columns to the tables underlying the duality views it creates. Each flex column is named **`ora$`***`<view-name>`***`_flex`**, where *`<view-name>`* is the name of the duality view where it is defined — see "DDL Code from GENERATE_SCHEMA with useFlexFields=true" in Using the Converter, Default Behavior. (You might mistake this for a field name in the *view* definition, but it's a column name; the name does not appear in the documents supported by the view.)

For descriptions of the PL/SQL subprograms comprising the converter, see:

About Migrations from JSON to Duality.

21.9 Migrating To Duality, Simplified Recipe

By ignoring whether an input field occurs rarely, or with a rarely used type, it's easier to migrate to JSON-relational duality. Handling such outlier cases can complicate the migration process.

But handling such cases can allow finer-degree normalization, and it can help find anomalies in your data that could represent bugs. This topic is about the simpler approach. Subsequent topics go into details that help you better understand and configure the migrator.

By default, the converter treats fields that occur in less than 5% of the documents of a document set as **occurrence outliers**, and field occurrences that have a given type in less than 5% of the documents as **type-occurrence outliers**, or **type outliers**. You can change the values of these thresholds using configuration fields `minFieldFrequency` and `minTypeFrequency`, respectively.

To show the reporting and handling of outliers, in the student-teacher-course example we generally use `25`% for `minFieldFrequency` and `15`% for `minTypeFrequency`, because the document sets are small. But for the simplified recipe used in this topic we set both of them to *zero* percent, so *no fields are considered outliers*.

Besides setting these two thresholds to zero, the *default* behavior of the migrator is what's illustrated here. This simplified recipe — default behavior except no outliers — isn't a bad way to begin whenever you migrate an application. And in many cases it will also be just what you need in the end.

By setting the minimum frequency thresholds to zero percent we configure the converter to accept as much as possible of the input data to be migrated, as is, at the possible cost of sacrificing maximal normalization. Our input student-teacher-course data contains some fields that we might ultimately want to treat as *outliers*, but there's no attempt in this topic to deal with them specially.

The fields that we normally treat as outliers in the rest of the migrator documentation are handled in these ways in this topic:

- The `Notes` occurrence in the course document for `MATH101` (`Algebra`) isn't removed, even though it occurs in only one (20%) of the documents.
- The `age` occurrence with string value `"Nineteen"`, in the student document for `Luis F.`, isn't converted to the number `19` so that its type agrees with the `age` occurrences (numbers) in the other nine documents (90%). Nor is a schema-inference validation error reported for this occurrence.

Instead, such input data, which could otherwise be considered problematic, is simply kept *as is*.

It's important to point out that outlier fields are not the only problems that migration might uncover. Even using the simplified recipe presented here it's possible that the importer can raise errors. A good example of that is two document sets that contradict each other, making it impossible to reconcile them without fixing the input data — for example, a *course* document says that `Natalie C.` teaches course `MATH101` and a *teacher* document says that `Abdul J.` teaches it. See Errors That Migrator Configuration Alone Can't Fix . (The migrator can help you discover some data coherency problems such as this, even if you're *not* migrating any data!)

Only *you* know, for your application, whether any particular data is an anomaly, according to your use of it. For example, only you know whether a rare type for a field, such as the single occurrence of string `"Nineteen"` for a student `age` field (whose value is usually a number), is normal or abnormal. Wanting *maximum respect of your input data* is the use case explored here with the simplified recipe. This is also an approach you might want to use generally, as a first step in migrating document sets, because it can quickly show you most of what's what.

The starting point for the migration is the three input document sets stored in Oracle Database transfer tables, as covered in Before Using the Converter (1): Create Database Document Sets . The input documents are shown both there and (more compactly) in School Administration Example, Migrator Input Documents .

We first use PL/SQL function `DBMS_JSON_DUALITY.`**`infer_schema`**, followed by function `DBMS_JSON_DUALITY.`**`generate_schema`**, to produce the SQL data-definition (DDL) code that creates (1) the duality views, (2) their underlying tables, (3) foreign-key constraints and indexes on the tables, and (4) triggers to create document-identifier fields `_id` for duality views where it doesn't already exist for the document set. The DDL code also adds top-level document-identifier field `_id`, because the input data doesn't already have it.

We then run that generated DDL code, creating those database objects.

Example 21-9 INFER_SCHEMA and GENERATE_SCHEMA with Zero Frequency Thresholds: No Outliers

In this example, PL/SQL function `DBMS_JSON_DUALITY.infer_schema` returns the JSON schema representing the inferred duality views and their underlying tables and indexes in JSON-type variable `er_schema`, which is passed to PL/SQL function `DBMS_JSON_DUALITY.generate_schema`. The output from `generate_schema`, SQL DDL code to create those database objects, is invoked using `EXECUTE IMMEDIATE`.

Configuration fields `minFieldFrequency` and `minTypeFrequency` are both set to zero for the schema inference by function `infer_schema`. This means that no fields in the input JSON data are to be considered outliers.

```
DECLARE
  er_schema    JSON;
  schema_sql  CLOB;
BEGIN
  er_schema :=
   DBMS_JSON_DUALITY.infer_schema(
     JSON('{"tableNames"        : [ "STUDENT_TAB",
                                    "TEACHER_TAB",
                                    "COURSE_TAB"],
            "viewNames"         : [ "STUDENT",
                                    "TEACHER",
                                    "COURSE" ],
            "minFieldFrequency" : 0,
            "minTypeFrequency"  : 0}'));
  schema_sql := DBMS_JSON_DUALITY.generate_schema(er_schema);
  EXECUTE IMMEDIATE schema_sql;
END;
/
```

Function `infer_schema` produces a JSON schema that describes the duality views and their tables. In this case, the schema shows that all of the input-data fields will be supported by the duality views.

Example 21-10 JSON Schema from INFER_SCHEMA for Duality Views with No Outliers

```
{"tables"         :
  [ {"title"         : "map_course_root_to_student_root",
     "dbObject"      : "map_course_root_to_student_root",
     "type"          : "object",
     "dbObjectType"  : "table",
     "dbMapTable"    : true,
     "properties"    : {"map_course_id"   : {"sqlType"    : "varchar2",
                                             "maxLength" : 64,
                                             "nullable"   : false},
                         "map_student_id" : {"sqlType" : "number",
                                             "nullable" : false}},
                        "required"     : [ "map_course_id", "map_student_id" ],
                        "dbPrimaryKey" : [ "map_course_id",
                                           "map_student_id"],
                        "dbForeignKey" : [ {"map_course_id"  : {"dbObject" : "course_root",
                                                                 "dbColumn" : "course_id"}},
                                           {"map_student_id" : {"dbObject" : "student_root",
                                                                 "dbColumn" : "student_id"}} ]},
    {"title"         : "teacher_root",
     "dbObject"      : "teacher_root",
     "type"          : "object",
     "dbObjectType"  : "table",
     "properties"    : {"_id"          : {"sqlType"   : "number", "nullable" : false},
                         "name"         : {"sqlType"   : "varchar2",
                                           "maxLength" : 64,
                                           "nullable"  : true,
                                         "unique"    : false},
                  "salary"       : {"sqlType"  : "number",
                                    "nullable" : true,
                                    "unique"   : false},
                  "department"   : {"sqlType"   : "varchar2",
                                    "maxLength" : 64,
                                    "nullable"  : true,
                                    "unique"    : false},
                  "phone_number" : {"sqlType"  : "json",
                                    "nullable" : true,
                                    "unique"   : false}},
  "required"     : [ "_id" ],
  "dbPrimaryKey" : [ "_id" ]},
 {"title"        : "course_root",
  "dbObject"     : "course_root",
  "type"         : "object",
  "dbObjectType" : "table",
  "properties"   : {"name"             : {"sqlType"   : "varchar2",
                                          "maxLength" : 64,
                                          "nullable"  : true,
                                          "unique"    : false},
                    "notes"            : {"sqlType"   : "varchar2",
                                          "maxLength" : 64,
                                          "nullable"  : true,
                                          "unique"    : false},
                    "course_id"        : {"sqlType"   : "varchar2",
                                          "maxLength" : 64,
                                          "nullable"  : false},
                    "credit_hours"     : {"sqlType"  : "json",
                                          "nullable" : true,
                                          "unique"   : false},
                    "class_type"       : {"sqlType"   : "varchar2",
                                          "maxLength" : 64,
                                          "nullable"  : true,
                                          "unique"    : false},
                    "avg_grade"        : {"sqlType"  : "json",
                                          "nullable" : true,
                                          "unique"   : false},
                    "_id_teacher_root" : {"sqlType"  : "number",
                                          "nullable" : true,
                                          "unique"   : false}},
  "required"     : [ "course_id" ],
  "dbPrimaryKey" : [ "course_id" ],
  "dbForeignKey" : [ {"_id_teacher_root" : {"dbObject" : "teacher_root",
                                            "dbColumn" : "_id"}} ]},
 {"title"        : "student_root",
  "dbObject"     : "student_root",
  "type"         : "object",
  "dbObjectType" : "table",
  "properties"   : {"age"        : {"sqlType" : "json",
                                    "nullable" : true,
                                    "unique" : false},
                    "name"       : {"sqlType" : "varchar2",
                                    "maxLength" : 64,
                                    "nullable" : true,
                                    "unique" : false},
                        "advisor_id" : {"sqlType" : "number",
                                        "nullable" : true,
                                        "unique" : false},
                        "student_id" : {"sqlType" : "number",
                                        "nullable" : false},
                        "dorm_id"    : {"sqlType" : "number",
                                        "nullable" : true,
                                        "unique" : false}},
      "required"     : [ "student_id" ],
      "dbPrimaryKey" : [ "student_id" ],
      "dbForeignKey" : [ {"advisor_id" : {"dbObject" : "teacher_root",
                                          "dbColumn" : "_id"}},
                         {"dorm_id"    : {"dbObject" : "student_dormitory",
                                          "dbColumn" : "dorm_id"}} ]},
     {"title"        : "student_dormitory",
      "dbObject"     : "student_dormitory",
      "type"         : "object",
      "dbObjectType" : "table",
      "properties"   : { "dorm_id"   : {"sqlType" : "number",
                                        "nullable" : false},
                         "dorm_name" : {"sqlType" : "varchar2",
                                        "maxLength" : 64,
                                        "nullable" : true,
                                        "unique" : false}},
      "required"     : [ "dorm_id" ],
      "dbPrimaryKey" : [ "dorm_id" ]} ],
  "views"         : [ {"title"                  : "STUDENT",
                       "dbObject"               : "STUDENT",
                       "dbObjectType"           : "dualityView",
                       "dbObjectProperties"     : [ "insert", "update", "delete", "check" ],
                       "dbMappedTableObject"    : "student_root",
                       "type"                   : "object",
                       "properties"             :
                         {"_id"          : {"type"                  : "number",
                                            "dbAssigned"            : true,
                                            "dbFieldProperties"     : [ "check" ],
                                            "dbObject"              : "student_root",
                                            "dbColumn"              : "student_id"},
                          "dbPrimaryKey" : [ "_id" ],
                          "age"          : {"type"                  : [ "number",
                                                                        "string",
                                                                        "null" ],
                                            "dbFieldProperties"     : [ "update", "check" ],
                                            "dbObject"              : "student_root",
                                            "dbColumn"              : "age"},
                          "name"         : {"type"                  : [ "string", "null" ],
                                            "maxLength"             : 64,
                                            "dbFieldProperties"     : [ "update", "check" ],
                                            "dbObject"              : "student_root",
                                            "dbColumn"              : "name"},
                          "courses"      :
                            {"type"   : "array",
                             "items"  : {"type"                  : "object",
                                         "dbMappedTableObject"   : "course_root",
                                         "properties" :
                                           {"dbPrimaryKey"       : [ "ora$mapCourseId",
                                              "ora$mapStudentId" ],
                        "ora$mapCourseId"   :
                          {"type"               : "string",
                           "maxLength"          : 64,
                           "dbAssigned"         : true,
                           "dbFieldProperties" : [ "check" ]},
                        "ora$mapStudentId" :
                           {"type" : "number",
                            "dbAssigned" : true,
                            "dbFieldProperties" : [ "check" ] },
                        "name"              :
                          {"type"               : [ "string",
                                                    "null" ],
                           "maxLength"          : 64,
                           "dbFieldProperties" : [ "update", "check" ],
                           "dbObject"           : "course_root",
                           "dbColumn"           : "name"},
                        "avgGrade"          :
                          {"type"               : [ "number",
                                                    "string",
                                                    "null" ],
                           "dbFieldProperties" : [ "update", "check" ],
                           "dbObject"           : "course_root",
                           "dbColumn"           : "avg_grade"},
                        "courseNumber"      :
                          {"type"               : "string",
                           "maxLength"          : 64,
                           "dbFieldProperties" : [ "check" ],
                           "dbObject"           : "course_root",
                           "dbColumn"           : "course_id"}},
                     "required"    : [ "ora$mapCourseId",
                                       "ora$mapStudentId",
                                       "courseNumber" ]}},
   "advisorId" : {"type"               : [ "number", "null" ],
                  "dbFieldProperties" : [ "update", "check" ],
                  "dbObject"           : "student_root",
                  "dbColumn"           : "advisor_id"},
   "dormitory" : {"type"                 : "object",
                  "dbMappedTableObject" : "student_dormitory",
                  "properties" :
                    {"dormId"   :
                       {"type"               : "number",
                        "dbFieldProperties" : [ "check" ],
                        "dbObject"           : "student_dormitory",
                        "dbColumn"           : "dorm_id"},
                     "dormName" :
                       {"type"               : [ "string", "null" ],
                        "maxLength"          : 64,
                        "dbFieldProperties" : [ "update", "check" ],
                       "dbObject"            : "student_dormitory",
                       "dbColumn"            : "dorm_name"}},
                  "required"   : [ "dormId" ]},
   "studentId" : {"dbFieldProperties" : [ "computed" ]}}},
{"title"                : "COURSE",
 "dbObject"             : "COURSE",
 "dbObjectType"         : "dualityView",
"dbObjectProperties"  : [ "insert", "update", "delete", "check" ],
"dbMappedTableObject" : "course_root",
"type"                : "object",
"properties" :
  {"_id"         : { "type"               : "string",
                     "maxLength"          : 64,
                     "dbAssigned"         : true,
                     "dbFieldProperties" : [ "check" ],
                     "dbObject"           : "course_root",
                     "dbColumn"           : "course_id"},
   "dbPrimaryKey" : [ "_id" ],
   "name"         : {"type"               : [ "string", "null" ],
                     "maxLength"          : 64,
                     "dbFieldProperties" : [ "update", "check" ],
                     "dbObject"           : "course_root",
                     "dbColumn"           : "name"},
   "Notes"        : {"type"               : [ "string", "null" ],
                     "maxLength"          : 64,
                     "dbFieldProperties" : ["update", "check" ],
                     "dbObject"           : "course_root",
                     "dbColumn"           : "notes"},
   "teacher"      :
     {"type"                : "object",
      "dbMappedTableObject" : "teacher_root",
      "properties" :
      {"name"      : {"type"               : [ "string", "null" ],
                      "maxLength" : 64,
                      "dbFieldProperties" : [ "update", "check" ],
                      "dbObject"           : "teacher_root",
                      "dbColumn"           : "name"},
       "teacherId" : {"type"               : "number",
                      "dbFieldProperties" : [ "check" ],
                      "dbObject"           : "teacher_root",
                      "dbColumn"           : "_id"}},
      "required"   : [ "teacherId" ]},
   "courseId"     : {"dbFieldProperties" : [ "computed" ]},
   "students"     :
     {"type" : "array",
      "items" :
        {"type" : "object",
         "dbMappedTableObject" : "student_root",
         "properties" :
           {"dbPrimaryKey"     : [ "ora$mapCourseId",
                                   "ora$mapStudentId" ],
            "ora$mapCourseId"  : {"type"               : "string",
                                  "maxLength"          : 64,
                                  "dbAssigned"         : true,
                                  "dbFieldProperties" : [ "check" ]},
            "ora$mapStudentId" : {"type"               : "number",
                                  "dbAssigned"         : true,
                                  "dbFieldProperties" : [ "check" ]},
            "name"             :
              {"type"               : [ "string", "null" ],
               "maxLength"          : 64,
               "dbFieldProperties" : [ "update", "check" ],
               "dbObject"           : "student_root",
                    "dbColumn"           : "name"},
                 "studentId"        : {"type"                : "number",
                                       "dbFieldProperties" : [ "check" ],
                                       "dbObject"          : "student_root",
                                       "dbColumn"          : "student_id"}},
              "required"   : [ "ora$mapCourseId",
                               "ora$mapStudentId",
                               "studentId" ]}},
        "creditHours"   :
          {"type"                : [ "number", "string", "null" ],
           "dbFieldProperties" : [ "update", "check" ],
           "dbObject"          : "course_root",
           "dbColumn"          : "credit_hours"}}},
  {"title"                : "TEACHER",
   "dbObject"             : "TEACHER",
   "dbObjectType"         : "dualityView",
   "dbObjectProperties"   : [ "insert", "update", "delete", "check" ],
   "dbMappedTableObject" : "teacher_root",
   "type"                 : "object",
   "properties" :
     {"_id"             : {"type"                 : "number",
                           "dbFieldProperties"  : [ "check" ],
                           "dbObject"           : "teacher_root",
                           "dbColumn"           : "_id"},
      "name"            : {"type"                : [ "string", "null" ],
                           "maxLength"          : 64,
                           "dbFieldProperties" : [ "update", "check" ],
                           "dbObject"          : "teacher_root",
                           "dbColumn"          : "name"},
      "salary"          : {"type"                : [ "number", "null" ],
                           "dbFieldProperties" : [ "update", "check" ],
                           "dbObject"          : "teacher_root",
                           "dbColumn"          : "salary"},
      "department"      : {"type"                : [ "string", "null" ],
                           "maxLength"          : 64,
                           "dbFieldProperties" : [ "update", "check" ],
                           "dbObject"          : "teacher_root",
                           "dbColumn"          : "department"},
      "phoneNumber"     :
        {"type"                : [ "string", "array", "null" ],
         "dbFieldProperties" : [ "update", "check" ],
         "dbObject"          : "teacher_root",
         "dbColumn"          : "phone_number"},
      "coursesTaught"   :
        {"type"   : "array",
         "items" :
           {"type"                 : "object",
            "dbMappedTableObject" : "course_root",
            "properties" :
              {"name"     :  {"type"                : [ "string", "null" ],
                              "maxLength"          : 64,
                              "dbFieldProperties" : [ "update", "check" ],
                              "dbObject"          : "course_root",
                              "dbColumn"          : "name"},
               "courseId"  : {"type"                : "string",
                              "maxLength"          : 64,
                                          "dbFieldProperties" : [ "check" ],
                                          "dbObject"          : "course_root",
                                          "dbColumn"          : "course_id"},
                           "classType" : {"type"              : [ "string", "null" ],
                                          "maxLength"         : 64,
                                          "dbFieldProperties" : [ "update", "check" ],
                                          "dbObject"          : "course_root",
                                          "dbColumn"          : "class_type"}},
                        "required"    : [ "courseId" ]}},
                  "studentsAdvised" :
                    {"type"   : "array",
                     "items" :
                       {"type"                : "object",
                        "dbMappedTableObject" : "student_root",
                        "properties" :
                          {"name"      : {"type"              : [ "string", "null" ],
                                          "maxLength"         : 64,
                                          "dbFieldProperties" : [ "update", "check" ],
                                          "dbObject"          : "student_root",
                                          "dbColumn"          : "name"},
                           "dormId"    : {"type"              : [ "number", "null" ],
                                          "dbFieldProperties" : [ "update", "check" ],
                                          "dbObject"          : "student_root",
                                          "dbColumn"          : "dorm_id"},
                           "studentId" : {"type"              : "number",
                                          "dbFieldProperties" : [ "check" ],
                                          "dbObject"          : "student_root",
                                          "dbColumn"          : "student_id"}},
                        "required"    : [ "studentId" ]}}}} ],
 "configOptions" : {"outputFormat"  : "executable",
                    "useFlexFields" : true}}
```

General observations:

- There are two parts to the schema: (1) a specification of the *tables* underlying the duality views (field `tables`) and (2) a specification of the duality *views* themselves (field `views`).
- The SQL data types of columns in the tables are specified by field **`sqlType`**. For example, column `department` has SQL type `VARCHAR2` because the value of document field `department` is always a JSON string.
- `JSON` data type is used for columns `phone_number`, `credit_hours`, `avg_grade`, and `age`, because the corresponding fields (`phoneNumber`, `creditHours`, `avgGrade`, and `age`) in the input documents have mixed type. There's no single SQL scalar type that can be used for such a field.
- Table `map_course_root_to_student_root` is a mapping table between tables `course_root` and `student_root`.
- All of the columns except[3] identifying columns (primary-key columns in the example data) are flagged with `nullable` = `true`, which means that their values can be (SQL) `NULL`. Fields corresponding to nullable columns (1) need not be present in a given document, and (2) when present, can have (JSON) `null` values.

There's a lot more information in that schema. Let's just point out some of it for now, using the schema of the *student* view and its documents as an example.

3 Primary-key column values cannot be `NULL`.

- For document-identifier field `_id`, schema field **`dbAssigned`** tells us that `_id` is added automatically to the documents for each duality view — it isn't present in the input data. And schema field **`dbColumn`** tells us that the `_id` value is stored in column **`student_id`**.
- Elsewhere, schema field **`dbAssigned`** tells us that fields `ora$mapCourseId` and `ora$mapStudentId` are also added automatically to the student documents.
- The (singleton array) value of schema field **`dbPrimaryKey`** tells us that document field `_id` corresponds to the only identifying (primary key) column of the duality view; field `_id` is the document identifier.
- In the *input* student documents, field `studentId` is the document identifier. That top-level field is supported by the student duality view, but its value is *generated* by the view, not stored. The string **`"computed"`** in the array value of schema field **`dbFieldProperties`** tells us this. (Its value is in fact taken from the `_id` value in column `student_id`.)
- Schema field **`dbMappedTableObject`** tells us that (1) table `student_root` is the root table underlying the student view, (2) table `course_root` underlies the array value of field `courses`, table `student_dormitory` underlies the fields in the object value of field `dormitory`.

Function `generate_schema` accepts as input a JSON schema such as the one produced by function `infer_schema`. In particular, this means that you can *edit the JSON schema that `infer_schema` produces*, to influence what `generate_schema` does. You can do this by hand-editing or by using SQL/JSON function `json_transform`. The following example illustrates the method of editing the JSON schema:

Example 21-11 Using JSON_TRANSFORM To Edit Inferred JSON Schema

As one example of modifying the JSON schema returned by `DBMS_JSON_DUALITY.infer_schema`, we change the `maxLength` value for a column from `64` to `100`.

We assume here that the value of PL/SQL variable `er_schema` is the schema returned by `DBMS_JSON_DUALITY.infer_schema`, as in the example "INFER_SCHEMA and GENERATE_SCHEMA with Zero Frequency Thresholds: No Outliers". This `json_transform` code changes that schema, to maximum length of the `name` field of a student document to `100`, saving the transformed value back into variable `er_schema`. The updated variable can then be passed to `DBMS_JSON_DUALITY.generate_schema`.

```
SELECT json_transform(
         er_schema,
         SET '$.tables[3].properties.name.maxLength' = 100)
  INTO er_schema FROM dual;
```

The left-hand side of this `SET` operation is a SQL/JSON path expression. The schema specifying table `student_root` is the fourth[4] entry in array `tables` of the overall JSON schema. Top-level field `name` for student documents is specified as a child of schema field `properties`, and field `maxLength` is a child of field `name`. (See Oracle SQL Function JSON_TRANSFORM in *Oracle Database JSON Developer's Guide*.)

In the topic Migrating To Duality, Simplified Recipe, the example in "DDL Code from GENERATE_SCHEMA for No-Outlier Use Case" shows the DDL code produced by function `generate_schema` in the example "INFER_SCHEMA and GENERATE_SCHEMA with Zero Frequency Thresholds: No Outliers", which indicates that all input-data fields are supported.

[4] JSON array indexing is zero-based.

Example 21-12   DDL Code from GENERATE_SCHEMA for No-Outlier Use Case

This example draws from the other examples in this topic. Function `DBMS_JSON_DUALITY.generate_schema`, produces the generated DDL code shown here if passed the JSON schema in the example "JSON Schema from INFER_SCHEMA for Duality Views with No Outliers", which is returned by function `infer_schema` (as shown in the example "INFER_SCHEMA and GENERATE_SCHEMA with Zero Frequency Thresholds: No Outliers", and using this as input.

(If instead it were passed the schema resulting from the modification in the example "Using JSON_TRANSFORM To Edit Inferred JSON Schema", then the only change here would be that column `student_root.name` would have a `maxLength` of `100` instead of `64`.)

Because the value of configuration field `outputFormat` is `"executable"` (by default), the generated DDL code uses `EXECUTE IMMEDIATE` for its statements.

The triggers created by the DDL code add top-level field `_id` to each document of a view, giving it the value of the corresponding primary-key field in each case. For example, for a student document it gives the added `_id` field the value of input field `studentId`.

```
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE student_dormitory(
   dorm_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   dorm_name  varchar2(64),
   ora$student_flex  JSON(Object),
   PRIMARY KEY(dorm_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE map_course_root_to_student_root(
   map_course_id  varchar2(64)  DEFAULT ON NULL SYS_GUID(),
   map_student_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   ora$student_flex  JSON(Object),
   ora$course_flex  JSON(Object),
   PRIMARY KEY(map_course_id,map_student_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE student_root(
   age  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
{ "type" :"string"}]}'',
   name  varchar2(64),
   dorm_id  number,
   advisor_id  number,
   student_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   ora$student_flex  JSON(Object),
   ora$teacher_flex  JSON(Object),
   PRIMARY KEY(student_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE teacher_root(
   "_id"  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   name  varchar2(64),
   salary  number,
   department  varchar2(64),
   phone_number  json VALIDATE ''{"oneOf" : [{ "type" :"string"},
{ "type" :"array"}]}'',
   ora$course_flex  JSON(Object),
   ora$teacher_flex  JSON(Object),
   PRIMARY KEY("_id")
)';

EXECUTE IMMEDIATE 'CREATE TABLE course_root(
   name  varchar2(64),
   notes  varchar2(64),
   avg_grade  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
{ "type" :"string"}]}'',
   course_id  varchar2(64)  DEFAULT ON NULL SYS_GUID(),
   class_type  varchar2(64),
   credit_hours  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
{ "type" :"string"}]}'',
   "_id_teacher_root"  number,
   ora$course_flex  JSON(Object),
   ora$teacher_flex  JSON(Object),
   PRIMARY KEY(course_id)
)';

EXECUTE IMMEDIATE 'ALTER TABLE map_course_root_to_student_root
ADD CONSTRAINT fk_map_course_root_to_student_root_to_course_root FOREIGN KEY
(map_course_id) REFERENCES course_root(course_id) DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE map_course_root_to_student_root
ADD CONSTRAINT fk_map_course_root_to_student_root_to_student_root FOREIGN KEY
(map_student_id) REFERENCES student_root(student_id) DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE student_root
ADD CONSTRAINT fk_student_root_to_teacher_root FOREIGN KEY (advisor_id)
REFERENCES teacher_root("_id") DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE student_root
ADD CONSTRAINT fk_student_root_to_student_dormitory FOREIGN KEY (dorm_id)
REFERENCES student_dormitory(dorm_id) DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE course_root
ADD CONSTRAINT fk_course_root_to_teacher_root FOREIGN KEY
("_id_teacher_root") REFERENCES teacher_root("_id") DEFERRABLE';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_map_course_root_to_student_root_to_course_root_index ON
map_course_root_to_student_root(map_course_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_map_course_root_to_student_root_to_student_root_index ON
map_course_root_to_student_root(map_student_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_student_root_to_teacher_root_index ON student_root(advisor_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_student_root_to_student_dormitory_index ON student_root(dorm_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_course_root_to_teacher_root_index ON course_root("_id_teacher_root")';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW STUDENT AS
student_root @insert @update @delete
{
  _id : student_id
  age
  name
  courses: map_course_root_to_student_root @insert @update @delete @array
  {
    ora$mapCourseId: map_course_id
    ora$mapStudentId: map_student_id
```

```
        ora$student_flex @flex
        course_root @unnest @insert @update @object
        {
          name
          avgGrade: avg_grade
          courseNumber: course_id
        }
      }
      advisorId:advisor_id
      dormitory: student_dormitory @insert @update @object
      {
        dormId: dorm_id
        dormName: dorm_name
        ora$student_flex @flex
      }
      studentId @generated (path: "$._id")
      ora$student_flex @flex
    }';

    EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW COURSE AS
    course_root @insert @update @delete
    {
      _id : course_id
      name
      Notes: notes
      teacher: teacher_root @insert @update @object
      {
        name
        teacherId: "_id"
        ora$course_flex @flex
      }
      courseId @generated (path: "$._id")
      students: map_course_root_to_student_root @insert @update @delete @array
      {
        ora$mapCourseId: map_course_id
        ora$mapStudentId: map_student_id
        ora$course_flex @flex
        student_root @unnest @insert @update @object
        {
          name
          studentId: student_id
        }
      }
      creditHours: credit_hours
      ora$course_flex @flex
    }';

    EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW TEACHER AS
    teacher_root @insert @update @delete
    {
      "_id"
      name
      salary
      department
      phoneNumber: phone_number
      coursesTaught: course_root @insert @update @delete @array
```

```
  {
    name
    courseId: course_id
    classType: class_type
    ora$teacher_flex @flex
  }
  studentsAdvised: student_root @insert @update @delete @array
  {
    name
    dormId:dorm_id
    studentId: student_id
    ora$teacher_flex @flex
  }
  ora$teacher_flex @flex
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER INSERT_TRIGGER_STUDENT
  BEFORE INSERT
  ON STUDENT
  FOR EACH ROW
DECLARE
  inp_jsonobj json_object_t;
BEGIN
  inp_jsonobj := json_object_t(:new.data);
  IF NOT inp_jsonobj.has(''_id'')
  THEN
    inp_jsonobj.put(''_id'', inp_jsonobj.get(''studentId''));
    :new.data := inp_jsonobj.to_json;
  END IF;
END;';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER INSERT_TRIGGER_COURSE
  BEFORE INSERT
  ON COURSE
  FOR EACH ROW
DECLARE
  inp_jsonobj json_object_t;
BEGIN
  inp_jsonobj := json_object_t(:new.data);
  IF NOT inp_jsonobj.has(''_id'')
  THEN
    inp_jsonobj.put(''_id'', inp_jsonobj.get(''courseId''));
    :new.data := inp_jsonobj.to_json;
  END IF;
END;';
END;
```

After executing the DDL code the conversion is complete, but we need to validate it, using PL/SQL function `DBMS_JSON_DUALITY.`**`validate_schema_report`**. That shows no errors (`no rows selected`) for each duality view, which means there are no validation failures — the duality views and the relational schema they represent are good.

Example 21-13   VALIDATE_SCHEMA_REPORT for No Outlier Use Case

```
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name  => 'TEACHER');


no rows selected


SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'COURSE_TAB',
                                  view_name  => 'COURSE');


no rows selected


SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'STUDENT_TAB',
                                  view_name  => 'STUDENT');


no rows selected
```

The duality views are still empty, not yet populated with the input data. We next (1) *create error logs* for the views and then (2) *import the data* from the temporary transfer tables, **`*_TAB`** into the views, using procedure `import_all`.

Example 21-14   Creating Error Logs for No Outlier Use Case

```
BEGIN
DBMS_ERRLOG.create_error_log(dml_table_name     => 'COURSE',
                             err_log_table_name => 'COURSE_ERR_LOG',
                             skip_unsupported   => TRUE);
DBMS_ERRLOG.create_error_log(dml_table_name     => 'TEACHER',
                             err_log_table_name => 'TEACHER_ERR_LOG',
                             skip_unsupported   => TRUE);
DBMS_ERRLOG.create_error_log(dml_table_name     => 'STUDENT',
                             err_log_table_name => 'STUDENT_ERR_LOG',
                             skip_unsupported   => TRUE);
END;
/
```

Error logging only reports *documents that can't be imported* (and only the first such error encountered in a given document is reported).

Example 21-15   Importing Document Sets, for No Outlier Use Case

```
BEGIN
DBMS_JSON_DUALITY.import_all(
                   JSON('{"tableNames" : [ "STUDENT_TAB",
                                           "TEACHER_TAB",
                                           "COURSE_TAB" ],
                          "viewNames"  : [ "STUDENT",
                                          "TEACHER",
                                          "COURSE" ],
                          "errorLog"    : [ "STUDENT_ERR_LOG",
                                          "TEACHER_ERR_LOG",
                                          "COURSE_ERR_LOG" ]}'));
END;
/
```

Import done; the duality views are populated.

Example 21-16 Checking Error Logs from Import, for No Outlier Use Case

The error logs are empty, showing that there are *no import errors* — there are no documents that didn't get imported.

```
SELECT ora_err_number$, ora_err_mesg$, ora_err_tag$ FROM student_err_log;


no rows selected


SELECT ora_err_number$, ora_err_mesg$, ora_err_tag$ FROM teacher_err_log;


no rows selected


SELECT ora_err_number$, ora_err_mesg$, ora_err_tag$ FROM course_err_log;


no rows selected
```

We next use `DBMS_JSON_DUALITY.`**`validate_import_report`** to report on any problems with *documents that have been imported* successfully. In this case, nothing is reported (no rows selected), which means that there are no such problems.

Example 21-17 VALIDATE_IMPORT_REPORT for No Outlier Use Case

```
SELECT * FROM DBMS_JSON_DUALITY.validate_import_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name  => 'TEACHER');


no rows selected


SELECT * FROM DBMS_JSON_DUALITY.validate_import_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name  => 'TEACHER');


no rows selected


SELECT * FROM DBMS_JSON_DUALITY.validate_import_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name => 'TEACHER');


no rows selected
```

Note:

An example of a problem that could be reported by `validate_import_report` is a contradiction between documents that were successfully imported. For example, if a *course* document says that the teacher of the `MATH101` course (`Algebra`) is `Natalie C.` and a *teacher* document says that the teacher of that course is `Abdul J.`, those documents are incompatible. The import validation report would provide a JSON Patch recipe that reconciles the problem by altering documents, for example by removing `MATH101` from the teacher document for `Abdul J.`, and adding it to the teacher document for `Natalie C.` That particular reconciliation might or might not be the one you want; a better data correction might be to instead change the course document for `MATH101` to show `Abdul J.` as the teacher. Only *you* know which content corrections are the most appropriate.

You can pretty-print the document collections supported by the resulting duality views using SQL function `json_serialize`, like this:

```
SELECT json_serialize(data PRETTY) FROM student;
```

Comparing the documents supported by the duality views with the original documents in School Administration Example, Migrator Input Documents or in the temporary transfer tables shows that the data is the same, *except* for the addition of the following:

- Document-identifier field `_id`, whose value corresponds to the identifying column(s) of the root table underlying the view. (The value is typically the same as an input-data identifier field; for example, for student documents field `_id` has the same value as field `studentId`.)

See Document-Identifier Field for Duality Views in *JSON-Relational Duality Developer's Guide*.

- Document-handling field `_metadata`. See Creating Duality Views in *JSON-Relational Duality Developer's Guide*.
- Fields, such as `ora$mapCourseId`, named for identifying columns of the mapping table. SeeJSON-To-Duality Converter: What It Does .

These differences are expected. See JSON-To-Duality Converter: What It Does.

Here's a document supported by the student duality view:

```
{"_id" : 1,
 "_metadata" : {"etag" : "39BA872C7E20186761BDD47B8AF40E3D",
                "asof" : "000000000043EF3F"},
 "age" : 20,
 "name" : "Donald P.",
 "courses"   : [ {"ora$mapCourseId"  : "CS101",
                  "ora$mapStudentId" : 1,
                  "name"             : "Algorithms",
                  "avgGrade"         : 75,
                  "courseNumber"     : "CS101"},
                 {"ora$mapCourseId"  : "CS102",
                  "ora$mapStudentId" : 1,
                  "name"             : "Data Structures",
                  "avgGrade"         : "TBD",
                  "courseNumber"     : "CS102"},
                 {"ora$mapCourseId"  : "MATH101",
                  "ora$mapStudentId" : 1,
                  "name"             : "Algebra",
                  "avgGrade"         : 90,
                  "courseNumber"     : "MATH101"} ],
 "advisorId" : 102,
 "dormitory" : {"dormId" : 201, "dormName" : "ABC"},
 "studentId" : 1}
```

```
SELECT json_serialize(data PRETTY) FROM student;
```

```
JSON_SERIALIZE(DATAPRETTY)
--------------------------
{
  "_id" : 1,
  "_metadata" :
  {
    "etag" : "39BA872C7E20186761BDD47B8AF40E3D",
    "asof" : "0000000000461E3D"
  },
  "age" : 20,
  "name" : "Donald P.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 1,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 1,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 1,
      "name" : "Algebra",
      "avgGrade" : 90,
      "courseNumber" : "MATH101"
    }
  ],
  "advisorId" : 102,
  "dormitory" :
  {
    "dormId" : 201,
    "dormName" : "ABC"
  },
  "studentId" : 1
}

{
  "_id" : 2,
  "_metadata" :
  {
    "etag" : "65B5DD1BE7B819306F2735F325E26400",
    "asof" : "0000000000461E3D"
  },
  "age" : 21,
  "name" : "Elena H.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 2,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 2,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 2,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    }
  ],
  "advisorId" : 103,
  "dormitory" :
  {
    "dormId" : 202,
    "dormName" : "XYZ"
  },
  "studentId" : 2
}

{
  "_id" : 3,
  "_metadata" :
  {
    "etag" : "E5AE58B21076D06FBA05010F0E1BEF21",
    "asof" : "0000000000461E3D"
  },
  "age" : 20,
  "name" : "Francis K.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 3,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 103,
  "dormitory" :
  {
    "dormId" : 204,
    "dormName" : "QWE"
  },
  "studentId" : 3
}

{
  "_id" : 4,
  "_metadata" :
  {
    "etag" : "D3B57FC478449FA24E123432C9D38673",
    "asof" : "0000000000461E3D"
  },
  "age" : 19,
  "name" : "Georgia D.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 4,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 4,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 4,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 101,
  "dormitory" :
  {
    "dormId" : 203,
    "dormName" : "LMN"
  },
  "studentId" : 4
}

{
  "_id" : 5,
  "_metadata" :
  {
    "etag" : "3FA71878EA5F02343CD62BC97F4C078E",
    "asof" : "0000000000461E3D"
  },
  "age" : 21,
  "name" : "Hye E.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 5,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 5,
      "name" : "Algebra",
      "avgGrade" : 90,
      "courseNumber" : "MATH101"
    }
  ],
  "advisorId" : 103,
  "dormitory" :
```

```
    {
      "dormId" : 201,
      "dormName" : "ABC"
    },
    "studentId" : 5
  }

  {
    "_id" : 6,
    "_metadata" :
    {
      "etag" : "6F06B3DFCAEB4CF71669FDA9263B3236",
      "asof" : "0000000000461E3D"
    },
    "age" : 21,
    "name" : "Ileana D.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 6,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 102,
    "dormitory" :
    {
      "dormId" : 205,
      "dormName" : "GHI"
    },
    "studentId" : 6
  }

  {
    "_id" : 7,
    "_metadata" :
    {
      "etag" : "6A44A0B63DEC99978D98813B9D7C1D07",
      "asof" : "0000000000461E3D"
    },
    "age" : 20,
    "name" : "Jatin S.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS101",
        "ora$mapStudentId" : 7,
        "name" : "Algorithms",
        "avgGrade" : 75,
        "courseNumber" : "CS101"
      },
      {
        "ora$mapCourseId" : "CS102",
        "ora$mapStudentId" : 7,
        "name" : "Data Structures",
        "avgGrade" : "TBD",
        "courseNumber" : "CS102"
      }
    ],
    "advisorId" : 101,
    "dormitory" :
    {
      "dormId" : 204,
      "dormName" : "QWE"
    },
    "studentId" : 7
  }

  {
    "_id" : 8,
    "_metadata" :
    {
      "etag" : "0B254C00DBCAA2E59DE30377138BD004",
      "asof" : "0000000000461E3D"
    },
    "age" : 21,
    "name" : "Katie H.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS102",
        "ora$mapStudentId" : 8,
        "name" : "Data Structures",
        "avgGrade" : "TBD",
        "courseNumber" : "CS102"
      },
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 8,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 102,
    "dormitory" :
    {
      "dormId" : 205,
      "dormName" : "GHI"
    },
    "studentId" : 8
  }

  {
    "_id" : 9,
    "_metadata" :
    {
      "etag" : "32D58F0278F226E26A5D4039A01D1288",
      "asof" : "0000000000461E3D"
    },
    "age" : "Nineteen",
    "name" : "Luis F.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS101",
        "ora$mapStudentId" : 9,
        "name" : "Algorithms",
        "avgGrade" : 75,
        "courseNumber" : "CS101"
      },
      {
        "ora$mapCourseId" : "MATH102",
        "ora$mapStudentId" : 9,
        "name" : "Calculus",
        "avgGrade" : 95,
        "courseNumber" : "MATH102"
      },
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 9,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 101,
    "dormitory" :
    {
      "dormId" : 201,
      "dormName" : "ABC"
    },
    "studentId" : 9
  }

  {
    "_id" : 10,
    "_metadata" :
    {
      "etag" : "979816C4FD15DC805007B9FF7D822168",
      "asof" : "0000000000461E3D"
    },
    "age" : 20,
    "name" : "Ming L.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "MATH102",
        "ora$mapStudentId" : 10,
        "name" : "Calculus",
        "avgGrade" : 95,
        "courseNumber" : "MATH102"
      }
    ],
    "advisorId" : 101,
    "dormitory" :
```

```
  {
    "dormId" : 202,
    "dormName" : "XYZ"
  },
  "studentId" : 10
}

10 rows selected.
```

You can also create data-guide JSON schemas that describe the document sets supported by the duality views, for comparison with those for the input document sets. Creating them is identical to creating the data guides for the input tables (see "Create JSON Data Guides for Input Document Sets" in Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas), except that the input data is selected from duality views `student`, `teacher`, and `course`, instead of from input transfer tables `student_tab`, `teacher_tab`, and `course_tab`.

The *schemas are identical* for each kind of documents (input and view-supported), *except* for the following:

- Document-identifier and document-handling fields `_id` and `_metadata` are added to the duality-view schemas.
- Fields such as `ora$mapCourseId` and `ora$mapStudentId` are added to the duality-view schemas. These identify columns of the mapping table.
- Dates in field `o:last_analyzed` differ. These just record when the data guide was created.

Except for field `o:last_analyzed`, these are the *same differences* noted above as existing between the (1) the original input documents and the transfer-table documents, on the one hand, and (2) the documents supported by the duality views, on the other hand. This points to a general tip:

Tip:

You can *compare JSON schemas* that model document sets as a *shortcut for comparing the document sets*. Of course, JSON schemas don't contain all of the information in the documents they describe, but they can highlight structure and typing differences, and thus serve as a proxy that gives you a good 50,000-foot view.

See Also:

- Oracle SQL Function JSON_TRANSFORM in *Oracle Database JSON Developer's Guide*
- DBMS_JSON_DUALITY in *Oracle Database PL/SQL Packages and Types Reference* for information about subprograms `generate_schema`, `infer_schema`, , , `import_all`, `validate_import_report`, and `validate_schema_report`
- VALIDATE_REPORT Function in *Oracle Database PL/SQL Packages and Types Reference* for information about function `DBMS_JSON_SCHEMA.validate_report`.
- DBMS_ERRLOG in *Oracle Database PL/SQL Packages and Types Reference* in for information about procedure `DBMS_ERRLOG.create_error_log`

21.10 Using the Converter, Default Behavior

Use of the JSON-to-duality converter with its default configuration-field values (except for `minFieldFrequency` and `minTypeFrequency`) is illustrated. In particular, configuration field `useFlexFields` is `true`. The database objects needed to support the document sets are inferred, and the SQL DDL code to construct them is generated.

Unlike the case in Migrating To Duality, Simplified Recipe, here we look at the effect of nonzero (and non-default) values of `minFieldFrequency` and `minTypeFrequency`, 25 and 15, respectively. The input document sets are the same (`student_tab`, `teacher_tab`, and `course_tab`), as are the names of the duality views generated (`student`, `teacher`, and `course`).

Here we again use the value of configuration field `useFlexFields`, `true`, which means the tables underlying duality views have flex columns. This allows the views to support some scalar fields whose values don't consistently correspond to single SQL scalar data types.

Note:

For more information about flex columns, see:

Flex Columns, Beyond the Basics in *JSON-Relational Duality Developer's Guide*

The document sets in the examples here are *very small*. In order to demonstrate the handling of outlier (high-entropy) fields, in examples here we use large values for migrator configuration fields `minFieldFrequency` (value `25`) and `minTypeFrequency` (value `15`), instead of the default value of `5`.

A field is an **occurrence outlier** for a given document set if it occurs in less than `minFieldFrequency` percent of the documents.

A field is a **type outlier** for a given document set if any of its values occurs with a given type in less than `minTypeFrequency` percent of the documents.

- An **occurrence-outlier** field (a field that occurs rarely) is not mapped by the *converter* to any underlying column. If the converter produces flex columns (configuration field `useFlexFields` = `true`, the default value), then the *importer* places an unmapped field in a flex column of a table underlying the duality view. If there are no flex columns then the importer reports an unmapped field in an import error log, and the field is not supported in the duality view.

- A **type-outlier** field (a field whose value is rarely of a different type than usual) is handled differently. Import tries to convert any values of a rare type to the expected type for the field. Unsuccessful conversion is reported in an import error log, and the field is not used in the duality view.

See JSON Configuration Fields Specifying Migrator Parameters in *JSON-Relational Duality Developer's Guide* for information about configuration fields `minFieldFrequency`, `minTypeFrequency`, and `useFlexFields`.

Example 21-18 INFER_SCHEMA and GENERATE_SCHEMA with useFlexFields = true

The code here to infer and generate the schema is the same as that in the example "INFER_SCHEMA and GENERATE_SCHEMA with Zero Frequency Thresholds: No Outliers" in Migrating To Duality, Simplified Recipe, except that (1) the configuration-fields input argument to `infer_schema` includes values for `minFieldFrequency` (`25`) and `minTypeFrequency` (`15`) that suit our small document sets. The value of `useFlexFields` is `true`, the default value, so this gives us a good idea of the default converter behavior.

See JSON Configuration Fields Specifying Migrator Parameters in *JSON-Relational Duality Developer's Guide* for the default behavior of other configuration fields.

```
DECLARE
  er_schema   JSON;
  schema_sql  CLOB;
BEGIN
  er_schema :=
   DBMS_JSON_DUALITY.infer_schema(
     JSON('{"tableNames"        : [ "STUDENT_TAB",
                                    "TEACHER_TAB",
                                    "COURSE_TAB"],
            "viewNames"         : [ "STUDENT",
                                    "TEACHER",
                                    "COURSE" ],
            "minFieldFrequency" : 25,
            "minTypeFrequency"  : 15}'));
  schema_sql := DBMS_JSON_DUALITY.generate_schema(er_schema);
  EXECUTE IMMEDIATE schema_sql;
END;
/
```

The following example shows the JSON schema returned by function `DBMS_JSON_DUALITY.infer_schema`:

Example 21-19 JSON Schema from INFER_SCHEMA for Duality Views: Default Behavior

```
{"tables"        :
  [ {"title"        : "map_course_root_to_student_root",
     "dbObject"     : "map_course_root_to_student_root",
     "type"         : "object",
     "dbObjectType" : "table",
     "dbMapTable"   : true,
     "properties"   : {"map_course_id"  : {"sqlType"   : "varchar2",
                                            "maxLength" : 64,
                                           "nullable"  : false},
                       "map_student_id" : {"sqlType" : "number",
                                           "nullable" : false}},
                      "required"     : [ "map_course_id", "map_student_id" ],
                      "dbPrimaryKey" : [ "map_course_id",
                                         "map_student_id"],
                      "dbForeignKey" : [ {"map_course_id"  : {"dbObject" : "course_root",
                                                              "dbColumn" : "course_id"}},
                                         {"map_student_id" : {"dbObject" : "student_root",
                                                              "dbColumn" : "student_id"}} ]},
    {"title"        : "teacher_root",
     "dbObject"     : "teacher_root",
     "type"         : "object",
     "dbObjectType" : "table",
     "properties"   : {"_id"           : {"sqlType"   : "number", "nullable" : false},
                       "name"          : {"sqlType"   : "varchar2",
                                         "maxLength" : 64,
                                         "nullable"  : true,
                                         "unique"    : false},
                       "salary"        : {"sqlType"  : "number",
                                         "nullable" : true,
                                         "unique"   : false},
                       "department"    : {"sqlType"   : "varchar2",
                                         "maxLength" : 64,
                                         "nullable"  : true,
                                         "unique"    : false},
                       "phone_number"  : {"sqlType"  : "json",
                                         "nullable" : true,
                                         "unique"   : false}},
     "required"     : [ "_id" ],
     "dbPrimaryKey" : [ "_id" ]},
    {"title"        : "course_root",
     "dbObject"     : "course_root",
     "type"         : "object",
     "dbObjectType" : "table",
     "properties"   : {"name"              : {"sqlType"   : "varchar2",
                                             "maxLength" : 64,
                                             "nullable"  : true,
                                             "unique"    : false},
                       "course_id"         : {"sqlType"   : "varchar2",
                                             "maxLength" : 64,
                                             "nullable"  : false},
                       "credit_hours"      : {"sqlType"  : "json",
                                             "nullable" : true,
                                             "unique"   : false},
                       "class_type"        : {"sqlType"   : "varchar2",
                                             "maxLength" : 64,
                                             "nullable"  : true,
                                             "unique"    : false},
                       "avg_grade"         : {"sqlType"  : "json",
                                             "nullable" : true,
                                             "unique"   : false},
                       "_id_teacher_root"  : {"sqlType"  : "number",
                                             "nullable" : true,
                                             "unique"   : false}},
     "required"     : [ "course_id" ],
         "dbPrimaryKey" : [ "course_id" ],
         "dbForeignKey" : [ {"_id_teacher_root" : {"dbObject" : "teacher_root",
                                                   "dbColumn" : "_id"}} ]},
        {"title"        : "student_root",
         "dbObject"     : "student_root",
         "type"         : "object",
         "dbObjectType" : "table",
         "properties"   : {"age"          : {"sqlType" : "number",
                                             "nullable" : true,
                                             "unique" : false},
                           "name"         : {"sqlType" : "varchar2",
                                             "maxLength" : 64,
                                             "nullable" : true,
                                             "unique" : false},
                           "advisor_id" : {"sqlType" : "number",
                                             "nullable" : true,
                                             "unique" : false},
                           "student_id" : {"sqlType" : "number",
                                             "nullable" : false},
                           "dorm_id"    : {"sqlType" : "number",
                                             "nullable" : true,
                                             "unique" : false}},
         "required"     : [ "student_id" ],
         "dbPrimaryKey" : [ "student_id" ],
         "dbForeignKey" : [ {"advisor_id" : {"dbObject" : "teacher_root",
                                             "dbColumn" : "_id"}},
                           {"dorm_id"    : {"dbObject" : "student_dormitory",
                                             "dbColumn" : "dorm_id"}} ]},
        {"title"        : "student_dormitory",
         "dbObject"     : "student_dormitory",
         "type"         : "object",
         "dbObjectType" : "table",
         "properties"   : { "dorm_id"   : {"sqlType" : "number",
                                            "nullable" : false},
                           "dorm_name" : {"sqlType" : "varchar2",
                                            "maxLength" : 64,
                                            "nullable" : true,
                                            "unique" : false}},
         "required"     : [ "dorm_id" ],
         "dbPrimaryKey" : [ "dorm_id" ]} ],
     "views"         : [ {"title"                : "STUDENT",
                          "dbObject"             : "STUDENT",
                          "dbObjectType"         : "dualityView",
                          "dbObjectProperties"   : [ "insert", "update", "delete", "check" ],
                          "dbMappedTableObject" : "student_root",
                          "type"                 : "object",
                          "properties"           :
                           {"_id"          : {"type"                : "number",
                                               "dbAssigned"          : true,
                                               "dbFieldProperties" : [ "check" ],
                                               "dbObject"            : "student_root",
                                               "dbColumn"            : "student_id"},
                            "dbPrimaryKey" : [ "_id" ],
                            "age"          : {"type"                : [ "number",
                                                                         "null" ],
                                               "dbFieldProperties" : [ "update", "check" ],
                      "dbObject"          : "student_root",
                      "dbColumn"          : "age"},
    "name"         : {"type"              : [ "string", "null" ],
                      "maxLength"         : 64,
                      "dbFieldProperties" : [ "update", "check" ],
                      "dbObject"          : "student_root",
                      "dbColumn"          : "name"},
    "courses"      :
      {"type"    : "array",
       "items"   : {"type"                  : "object",
                   "dbMappedTableObject" : "course_root",
                   "properties" :
                     {"dbPrimaryKey"      : [ "ora$mapCourseId",
                                             "ora$mapStudentId" ],
                      "ora$mapCourseId"   :
                        {"type"               : "string",
                         "maxLength"          : 64,
                         "dbAssigned"         : true,
                         "dbFieldProperties" : [ "check" ]},
                      "ora$mapStudentId" :
                         {"type" : "number",
                          "dbAssigned" : true,
                          "dbFieldProperties" : [ "check" ] },
                      "name"              :
                        {"type"               : [ "string",
                                                  "null" ],
                         "maxLength"          : 64,
                         "dbFieldProperties" : [ "update", "check" ],
                         "dbObject"           : "course_root",
                         "dbColumn"           : "name"},
                      "avgGrade"          :
                        {"type"               : [ "number",
                                                  "string",
                                                  "null" ],
                         "dbFieldProperties" : [ "update", "check" ],
                         "dbObject"           : "course_root",
                         "dbColumn"           : "avg_grade"},
                      "courseNumber"      :
                        {"type"               : "string",
                         "maxLength"          : 64,
                         "dbFieldProperties" : [ "check" ],
                         "dbObject"           : "course_root",
                         "dbColumn"           : "course_id"}},
                   "required"   : [ "ora$mapCourseId",
                                    "ora$mapStudentId",
                                    "courseNumber" ]}},
    "advisorId" : {"type"                : [ "number", "null" ],
                   "dbFieldProperties" : [ "update", "check" ],
                   "dbObject"            : "student_root",
                   "dbColumn"            : "advisor_id"},
    "dormitory" : {"type"                  : "object",
                   "dbMappedTableObject" : "student_dormitory",
                   "properties" :
                     {"dormId"    :
                        {"type"               : "number",
                         "dbFieldProperties" : [ "check" ],
                                    "dbObject"          : "student_dormitory",
                                    "dbColumn"          : "dorm_id"},
                                 "dormName" :
                                   {"type"              : [ "string", "null" ],
                                    "maxLength"         : 64,
                                    "dbFieldProperties" : [ "update", "check" ],
                                   "dbObject"           : "student_dormitory",
                                   "dbColumn"           : "dorm_name"}},
                              "required"  : [ "dormId" ]},
           "studentId" : {"dbFieldProperties" : [ "computed" ]}}},
       {"title"                 : "COURSE",
        "dbObject"              : "COURSE",
        "dbObjectType"          : "dualityView",
        "dbObjectProperties"    : [ "insert", "update", "delete", "check" ],
        "dbMappedTableObject"   : "course_root",
        "type"                  : "object",
        "properties" :
          {"_id"         : { "type"              : "string",
                            "maxLength"         : 64,
                            "dbAssigned"        : true,
                            "dbFieldProperties" : [ "check" ],
                            "dbObject"          : "course_root",
                            "dbColumn"          : "course_id"},
           "dbPrimaryKey" : [ "_id" ],
           "name"         : {"type"              : [ "string", "null" ],
                            "maxLength"         : 64,
                            "dbFieldProperties" : [ "update", "check" ],
                            "dbObject"          : "course_root",
                            "dbColumn"          : "name"},
           "teacher"      :
             {"type"                 : "object",
              "dbMappedTableObject" : "teacher_root",
              "properties" :
              {"name"      : {"type"              : [ "string", "null" ],
                              "maxLength" : 64,
                              "dbFieldProperties" : [ "update", "check" ],
                              "dbObject"          : "teacher_root",
                              "dbColumn"          : "name"},
               "teacherId" : {"type"              : "number",
                              "dbFieldProperties" : [ "check" ],
                              "dbObject"          : "teacher_root",
                              "dbColumn"          : "_id"}},
              "required"   : [ "teacherId" ]},
           "courseId"     : {"dbFieldProperties" : [ "computed" ]},
           "students"     :
             {"type" : "array",
              "items" :
                {"type" : "object",
                 "dbMappedTableObject" : "student_root",
                 "properties" :
                   {"dbPrimaryKey"     : [ "ora$mapCourseId",
                                           "ora$mapStudentId" ],
                    "ora$mapCourseId"  : {"type"              : "string",
                                         "maxLength"         : 64,
                                         "dbAssigned"        : true,
                                         "dbFieldProperties" : [ "check" ]},
                 "ora$mapStudentId" : {"type"              : "number",
                                       "dbAssigned"        : true,
                                       "dbFieldProperties" : [ "check" ]},
                 "name"             :
                   {"type"              : [ "string", "null" ],
                    "maxLength"         : 64,
                    "dbFieldProperties" : [ "update", "check" ],
                    "dbObject"          : "student_root",
                    "dbColumn"          : "name"},
                 "studentId"        : {"type"              : "number",
                                       "dbFieldProperties" : [ "check" ],
                                       "dbObject"          : "student_root",
                                       "dbColumn"          : "student_id"}},
              "required"   : [ "ora$mapCourseId",
                               "ora$mapStudentId",
                               "studentId" ]}},
        "creditHours"  :
          {"type"              : [ "number", "string", "null" ],
           "dbFieldProperties" : [ "update", "check" ],
           "dbObject"          : "course_root",
           "dbColumn"          : "credit_hours"}}},
    {"title"               : "TEACHER",
     "dbObject"            : "TEACHER",
     "dbObjectType"        : "dualityView",
     "dbObjectProperties"  : [ "insert", "update", "delete", "check" ],
     "dbMappedTableObject" : "teacher_root",
     "type"                : "object",
     "properties" :
       {"_id"             : {"type"               : "number",
                            "dbFieldProperties"  : [ "check" ],
                            "dbObject"           : "teacher_root",
                            "dbColumn"           : "_id"},
        "name"            : {"type"              : [ "string", "null" ],
                            "maxLength"         : 64,
                            "dbFieldProperties" : [ "update", "check" ],
                            "dbObject"          : "teacher_root",
                            "dbColumn"          : "name"},
        "salary"          : {"type"              : [ "number", "null" ],
                            "dbFieldProperties" : [ "update", "check" ],
                            "dbObject"          : "teacher_root",
                            "dbColumn"          : "salary"},
        "department"      : {"type"              : [ "string", "null" ],
                            "maxLength"         : 64,
                            "dbFieldProperties" : [ "update", "check" ],
                            "dbObject"          : "teacher_root",
                            "dbColumn"          : "department"},
        "phoneNumber"     :
          {"type"              : [ "string", "array", "null" ],
           "dbFieldProperties" : [ "update", "check" ],
           "dbObject"          : "teacher_root",
           "dbColumn"          : "phone_number"},
        "coursesTaught"  :
          {"type"  : "array",
           "items" :
             {"type"                : "object",
              "dbMappedTableObject" : "course_root",
                                        "properties" :
                                         {"name"     :  {"type"               : [ "string", "null" ],
                                                         "maxLength"          : 64,
                                                         "dbFieldProperties" : [ "update", "check" ],
                                                         "dbObject"           : "course_root",
                                                         "dbColumn"           : "name"},
                                           "courseId"  : {"type"               : "string",
                                                         "maxLength"          : 64,
                                                         "dbFieldProperties" : [ "check" ],
                                                         "dbObject"           : "course_root",
                                                         "dbColumn"           : "course_id"},
                                           "classType" : {"type"               : [ "string", "null" ],
                                                         "maxLength"          : 64,
                                                         "dbFieldProperties" : [ "update", "check" ],
                                                         "dbObject"           : "course_root",
                                                         "dbColumn"           : "class_type"}},
                                        "required"   : [ "courseId" ]}},
                                  "studentsAdvised" :
                                    {"type"  : "array",
                                     "items" :
                                       {"type"                 : "object",
                                        "dbMappedTableObject" : "student_root",
                                        "properties" :
                                         {"name"      : {"type"               : [ "string", "null" ],
                                                         "maxLength"          : 64,
                                                         "dbFieldProperties" : [ "update", "check" ],
                                                         "dbObject"           : "student_root",
                                                         "dbColumn"           : "name"},
                                           "dormId"    : {"type"               : [ "number", "null" ],
                                                         "dbFieldProperties" : [ "update", "check" ],
                                                         "dbObject"           : "student_root",
                                                         "dbColumn"           : "dorm_id"},
                                           "studentId" : {"type"               : "number",
                                                         "dbFieldProperties" : [ "check" ],
                                                         "dbObject"           : "student_root",
                                                         "dbColumn"           : "student_id"}},
                                        "required"   : [ "studentId" ]}}}} ],
     "configOptions" : {"outputFormat"   : "executable",
                        "useFlexFields" : true}}
```

The differences here from the schema inferred when `minFieldFrequency` and `minTypeFrequency` are zero (see the example "JSON Schema from INFER_SCHEMA for Duality Views with No Outliers" in Migrating To Duality, Simplified Recipe) are these:[5]

- For the student table and view, column and field `age` have type number.
- For the course table and view, column `notes` and field `Notes` are *absent*.

In the schema inferred when `minFieldFrequency` and `minTypeFrequency` are *zero*, the notes column and field are present, the `age` column has type json, and the `age` field has type number-or-string.

So even before generating DDL code to create the duality views and their tables, you can see from the output of `infer_schema` some of what to expect for those two outlier fields. If you

5 All of these fields also have type null, which is generally the case for fields that don't correspond to identifying columns.

recall that there is a student document with `age` = `"Nineteen"` then you already know that, on import, that document won't have an `age` field.

The following example shows the DDL code produced by `generate_schema`.

Example 21-20 DDL Code from GENERATE_SCHEMA with useFlexFields = true

Function `DBMS_JSON_DUALITY.generate_schema`, produces the generated DDL code shown here if passed the JSON schema in the example "JSON Schema from INFER_SCHEMA for Duality Views: Default Behavior" in Using the Converter, Default Behavior, which is returned by function `infer_schema` (example "INFER_SCHEMA and GENERATE_SCHEMA with useFlexFields = true" in Using the Converter, Default Behavior) as input.

Differences from the example "DDL Code from GENERATE_SCHEMA for No-Outlier Use Case" in Migrating To Duality, Simplified Recipe are as follows:

- Column `student_root.age` has type `number` here, not number-or-string.
- There is no column `course_root.notes` to support field `Notes`. (Instead, the *importer* will place field `Notes` in flex column `course_root.ora$course_flex`.)

The duality-view definitions here use GraphQL syntax. Equivalent SQL duality-view definitions are shown in the example "SQL DDL Code for Duality-View Creations with useFlexFields = true" in Using the Converter, Default Behavior.

```
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE student_dormitory(
   dorm_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   dorm_name  varchar2(64),
   ora$student_flex  JSON(Object),
   PRIMARY KEY(dorm_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE map_course_root_to_student_root(
   map_course_id  varchar2(64)  DEFAULT ON NULL SYS_GUID(),
   map_student_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   ora$student_flex  JSON(Object),
   ora$course_flex  JSON(Object),
   PRIMARY KEY(map_course_id,map_student_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE student_root(
   age  number,
   name  varchar2(64),
   dorm_id  number,
   advisor_id  number,
   student_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   ora$student_flex  JSON(Object),
   ora$teacher_flex  JSON(Object),
   PRIMARY KEY(student_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE teacher_root(
   "_id"  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   name  varchar2(64),
   salary  number,
   department  varchar2(64),
   phone_number  json VALIDATE ''{"oneOf" : [{ "type" :"string"},
{ "type" :"array"}]}'',
   ora$teacher_flex  JSON(Object),
   ora$course_flex  JSON(Object),
   PRIMARY KEY("_id")
)';

EXECUTE IMMEDIATE 'CREATE TABLE course_root(
   name  varchar2(64),
   avg_grade  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
{ "type" :"string"}]}'',
   course_id  varchar2(64)  DEFAULT ON NULL SYS_GUID(),
   class_type  varchar2(64),
   credit_hours  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
{ "type" :"string"}]}'',
   "_id_teacher_root"  number,
   ora$teacher_flex  JSON(Object),
   ora$course_flex  JSON(Object),
   PRIMARY KEY(course_id)
)';

EXECUTE IMMEDIATE 'ALTER TABLE map_course_root_to_student_root
ADD CONSTRAINT fk_map_course_root_to_student_root_to_course_root
  FOREIGN KEY (map_course_id) REFERENCES course_root(course_id) DEFERRABLE';

EXECUTE IMMEDIATE 'ALTER TABLE map_course_root_to_student_root
ADD CONSTRAINT fk_map_course_root_to_student_root_to_student_root
  FOREIGN KEY (map_student_id) REFERENCES student_root(student_id)
DEFERRABLE';

EXECUTE IMMEDIATE 'ALTER TABLE student_root
ADD CONSTRAINT fk_student_root_to_teacher_root
  FOREIGN KEY (advisor_id) REFERENCES teacher_root("_id") DEFERRABLE';

EXECUTE IMMEDIATE 'ALTER TABLE student_root
ADD CONSTRAINT fk_student_root_to_student_dormitory
  FOREIGN KEY (dorm_id) REFERENCES student_dormitory(dorm_id) DEFERRABLE';

EXECUTE IMMEDIATE 'ALTER TABLE course_root
ADD CONSTRAINT fk_course_root_to_teacher_root
  FOREIGN KEY ("_id_teacher_root") REFERENCES teacher_root("_id") DEFERRABLE';

EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
  fk_map_course_root_to_student_root_to_course_root_index
  ON map_course_root_to_student_root(map_course_id)';

EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
  fk_map_course_root_to_student_root_to_student_root_index
  ON map_course_root_to_student_root(map_student_id)';

EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
  fk_student_root_to_teacher_root_index
  ON student_root(advisor_id)';

EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
  fk_student_root_to_student_dormitory_index
  ON student_root(dorm_id)';
```

```
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
  fk_course_root_to_teacher_root_index
  ON course_root("_id_teacher_root")';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW STUDENT AS
student_root @insert @update @delete
{
  _id : student_id
  age
  name
  courses: map_course_root_to_student_root @insert @update @delete @array
  {
    ora$mapCourseId: map_course_id
    ora$mapStudentId: map_student_id
    ora$student_flex @flex
    course_root @unnest @insert @update @object
    {
      name
      avgGrade: avg_grade
      courseNumber: course_id
    }
  }
  advisorId:advisor_id
  dormitory: student_dormitory @insert @update @object
  {
    dormId: dorm_id
    dormName: dorm_name
    ora$student_flex @flex
  }
  studentId @generated (path: "$._id")
  ora$student_flex @flex
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW TEACHER AS
teacher_root @insert @update @delete
{
  "_id"
  name
  salary
  department
  phoneNumber: phone_number
  coursesTaught: course_root @insert @update @delete @array
  {
    name
    courseId: course_id
    classType: class_type
    ora$teacher_flex @flex
  }
  studentsAdvised: student_root @insert @update @delete @array
  {
    name
    dormId:dorm_id
    studentId: student_id
    ora$teacher_flex @flex
  }
  ora$teacher_flex @flex
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW COURSE AS
course_root @insert @update @delete
{
  _id : course_id
  name
  teacher: teacher_root @insert @update @object
  {
    name
    teacherId: "_id"
    ora$course_flex @flex
  }
  courseId @generated (path: "$._id")
  students: map_course_root_to_student_root @insert @update @delete @array
  {
    ora$mapCourseId: map_course_id
    ora$mapStudentId: map_student_id
    ora$course_flex @flex
    student_root @unnest @insert @update @object
    {
      name
      studentId: student_id
    }
  }
  creditHours: credit_hours
  ora$course_flex @flex
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER INSERT_TRIGGER_STUDENT
  BEFORE INSERT
  ON STUDENT
  FOR EACH ROW
DECLARE
  inp_jsonobj json_object_t;
BEGIN
  inp_jsonobj := json_object_t(:new.data);
  IF NOT inp_jsonobj.has(''_id'')
  THEN
    inp_jsonobj.put(''_id'', inp_jsonobj.get(''studentId''));
    :new.data := inp_jsonobj.to_json;
  END IF;
END;';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER INSERT_TRIGGER_COURSE
  BEFORE INSERT
  ON COURSE
  FOR EACH ROW
DECLARE
  inp_jsonobj json_object_t;
BEGIN
  inp_jsonobj := json_object_t(:new.data);
  IF NOT inp_jsonobj.has(''_id'')
  THEN
    inp_jsonobj.put(''_id'', inp_jsonobj.get(''courseId''));
```

```
      :new.data := inp_jsonobj.to_json;
    END IF;
  END;';
  END;
```

Besides creating the duality views and their underlying tables, the DDL code does the following as part of the default behavior:

- For each duality view *<view-name>*, each table that directly underlies the top-level fields of an object in the supported documents has a flex column named **ora$***<view-name>***_flex** (because useFlexFields was implicitly true for the DDL generation).
- Tables student_root and teacher_root have primary-key columns student_id and _id, respectively.
- Table course_root has primary-key column course_id. Its column _id_teacher_root is a foreign key to column _id of table teacher_root, which is the primary key of that table. Table course_root has an index on its foreign-key column, _id_teacher_root.
- Table map_course_root_to_student_root is a mapping table between tables course_root and student_root.
  - Its primary key is a composite of its columns map_course_id and map_student_id.
  - Its columns map_course_id and map_student_id are foreign keys to columns course_id and student_id in tables course_root and student_root, respectively, which are the primary-key columns of those tables.
  - It has indexes on its two foreign-key columns.
- Views course and student each have a field (courseId and studentId, respectively) whose value is not stored but is generated from the value of the view's field _id.

  This is because a duality view must have an _id field, which corresponds to the identifying columns of the root table that underlies it, but documents from the existing app instead have a courseId or studentId field. In views course and student those fields are always generated from field _id, so inserting a document stores their values in field _id instead. (See Document-Identifier Field for Duality Views in *JSON-Relational Duality Developer's Guide*.)
- Views course and student each have a before-insert trigger (insert_trigger_course and insert_trigger_student, respectively) that stores the value of an incoming courseId or studentId field, respectively, in field _id. If the incoming document has no field _id at its top level yet, then the trigger (1) adds it and (2) gives it the value of field courseId or studentId. (Importing uses INSERT operations, and these triggers fire just before such operations.)

Example 21-21 SQL DDL Code For Duality-View Creations with useFlexFields = true

For information, in case SQL is more familiar to you than GraphQL, this SQL DDL code is equivalent to the GraphQL duality-view creation code shown in the preceding example "DDL Code from GENERATE_SCHEMA with useFlexFields = true" .

```
CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW STUDENT AS
  SELECT JSON {'_id' : s.student_id,
               'age'        :  s.age,
               'name'       :  s.name,
               'courses'    :
                  [SELECT JSON {'ora$mapCourseId'   : m.map_course_id,
                                'ora$mapStudentId'  : m.map_student_id,
                                 m.ora$student_flex AS FLEX,
                                 UNNEST
                                 (SELECT JSON {'name'         : c.name,
                                               'avgGrade'     : c.avg_grade,
                                               'courseNumber' : c.course_id}
                                    FROM course_root c WITH INSERT UPDATE
                                    WHERE c.course_id = m.map_course_id)}
                    FROM map_course_root_to_student_root m WITH INSERT UPDATE DELETE
                    WHERE s.student_id = m.map_student_id],
               'advisorId' :  s.advisor_id,
               'dormitory' :
                  (SELECT JSON {'dormId'    : sd.dorm_id,
                                'dormName'  : sd.dorm_name,
                                 sd.ora$student_flex AS FLEX}
                      FROM student_dormitory sd  WITH INSERT UPDATE
                      WHERE s.dorm_id = sd.dorm_id),
               'studentId' IS GENERATED USING PATH '$._id',
               s.ora$student_flex AS FLEX
               RETURNING JSON}
    FROM student_root s WITH INSERT UPDATE DELETE;

CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW TEACHER AS
  SELECT JSON {'_id'            : t."_id",
               'name'            : t.name,
               'salary'          : t.salary,
               'department'      : t.department,
               'coursesTaught' :
                 [SELECT JSON {'name'      : c.name,
                               'courseId'  : c.course_id,
                               'classType' : c.class_type,
                               c.ora$teacher_flex AS FLEX}
                    FROM course_root c WITH INSERT UPDATE DELETE
                    WHERE c."_id_teacher_root" = t."_id"],
               'studentsAdvised' :
                 [SELECT JSON {'name'      : s.name,
                               'dormId'    : s.dorm_id,
                               'studentId' : s.student_id,
                               s.ora$teacher_flex AS FLEX}
                    FROM student_root s WITH INSERT UPDATE DELETE
                    WHERE s.advisor_id = t."_id"],
               t.ora$teacher_flex AS FLEX
               RETURNING JSON}
    FROM teacher_root t WITH INSERT UPDATE DELETE;

CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW COURSE AS
  SELECT JSON {'_id'         : c.course_id,
                'name'        : c.name,
                'teacher'     :
                  (SELECT JSON {'name'      : t.name,
                                'teacherId' : t."_id",
                                t.ora$course_flex AS FLEX}
                     FROM teacher_root t WITH INSERT UPDATE
                WHERE t."_id" = c."_id_teacher_root"),
                'courseId' IS GENERATED USING PATH '$._id',
                'students' :
                   [SELECT JSON {'ora$mapCourseId'  : m.map_course_id,
                                 'ora$mapStudentId' : m.map_student_id,
                                 m.ora$course_flex AS FLEX,
                                 UNNEST
                                 (SELECT JSON {'name'      : s.name,
                                               'studentId' : s.student_id}
                                    FROM student_root s WITH INSERT UPDATE
                                    WHERE s.student_id = m.map_student_id)}
                     FROM map_course_root_to_student_root m WITH INSERT UPDATE DELETE
                     WHERE c.course_id = m.map_course_id],
                'creditHours' : c.credit_hours,
                c.ora$course_flex AS FLEX
                RETURNING JSON}
    FROM course_root c WITH INSERT UPDATE DELETE;
```

When you generate the DDL code and then execute it, the duality views and their underlying tables are created.

After executing the DDL code, you run converter function `DBMS_JSON_DUALITY.validate_schema_report` for each kind (student, teacher, course) of input table and duality view, to validate the conversion.

Example 21-22 VALIDATE_SCHEMA_REPORT for Default Case (useFlexFields = true)

```
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'STUDENT_TAB',
                                  view_name  => 'STUDENT');
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name  => 'TEACHER');
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'COURSE_TAB',
                                  view_name  => 'COURSE');
```

For the student data, an error is reported for the string value ("`Nineteen`") of field `age` in the document for student `Luis F.` (`studentId` = `9`) — only a numeric or `null` value is allowed.

Function `validate_schema_report` places the anomalous input document in column `DATA` of its output, and it places an entry in column `ERRORS` of the same report row:

```
DATA
--------
{"studentId":9,"name":"Luis F.","age":"Nineteen","advisorId":101,"courses":[{"co
urseNumber":"CS101","name":"Algorithms","avgGrade":75},{"courseNumber":"MATH102"
,"name":"Calculus","avgGrade":95},{"courseNumber":"MATH103","name":"Advanced Alg
ebra","avgGrade":82}],"dormitory":{"dormId":201,"dormName":"ABC"}}


ERRORS
----------
[{"schemaPath":"$","instancePath":"$","code":"JZN-00501","error":"JSON schema va
lidation failed"},{"schemaPath":"$.properties","instancePath":"$","code":"JZN-00
514","error":"invalid properties: 'age'"},{"schemaPath":"$.properties.age.type",
"instancePath":"$.age","code":"JZN-00504","error":"invalid type found, actual: s
tring, expected one of: number, null"},{"schemaPath":"$.properties.age.extendedT
ype","instancePath":"$.age","code":"JZN-00504","error":"invalid type found, actu
al: string, expected one of: number, null"}]
```

For that document, the *importer* will try, and fail, to convert the string value of `"Nineteen"` in the input data to a number. It will log that type failure as an error. If the input string value were instead `"19"` then the importer would be able to convert the value to the number `19` and store it as such.

There are no errors reported for the teacher data.

No errors are reported for the course data either. In particular, there is no error for occurrence-outlier field `Notes` of the `Algebra` course (`MATH101`). This is because `useFlexFields` is `true` creates flex columns to the underlying tables. As it does with all input fields that aren't mapped to columns, the *importer* will place field `Notes` in a flex column, so it will be supported by the `course` duality view.

At this point, before importing you could choose to change the `age` field in the student input document for Luis J., to give a number value of `19` instead of the string value `"Nineteen"`. Besides fixing problematic data, before importing you might sometimes want to modify/edit the DDL scripts, to change the conversion behavior or the names of the views, tables, or indexes to be created.

- DBMS_JSON_DUALITY in *Oracle Database PL/SQL Packages and Types Reference* for information about subprograms `generate_schema`, `infer_schema`, and `validate_import_report`
- VALIDATE_REPORT Function in *Oracle Database PL/SQL Packages and Types Reference* for information about function `DBMS_JSON_SCHEMA.validate_report`

21.11 Import After Default Conversion

After default conversion (except for `minFieldFrequency` and `minTypeFrequency`), in particular with `useFlexFields:true`), almost all documents from the student, teacher, and course input document sets are successfully imported, but some fields are not exactly as they were in the original, input documents.

The process of creating error logs and importing the input document sets (in tables `student_tab`, `teacher_tab`, and `course_tab`) into the duality views created in Using the Converter, Default Behavior is exactly the same as in the simplified recipe case: see "Creating Error Logs for No Outlier Use Case" and "Importing Document Sets, for No Outlier Use Case" in Migrating To Duality, Simplified Recipe. Checking the error logs for the default case tells a different story.

Example 21-23 Checking Error Logs from Import, for Default Case

There are no errors logged for import into duality views `teacher` and `course`. But unlike the simplified recipe case ("Checking Error Logs from Import, for No Outlier Use Case" in Migrating To Duality, Simplified Recipe), import into the student view logs an error for the type-occurrence outlier for field `age` with value `"Nineteen"`.

```
SELECT ora_err_number$, ora_err_mesg$, ora_err_tag$
  FROM student_err_log;


ORA_ERR_NUMBER$
---------------
ORA_ERR_MESG$
```

```
-------------
ORA_ERR_TAG$
------------
42555

ORA-42555: Cannot insert into JSON Relational Duality View 'STUDENT': The
input JSON document is invalid.

JZN-00671: value of field 'age' can not be converted to a number

Import Error
```

Select the culprit student document from the input student table:

```
SELECT * FROM "JANUS".student_tab
  WHERE ROWID IN (SELECT ora_err_rowid$ FROM student_err_log);
```

```
DATA
----
{"studentId":9,"name":"Luis F.","age":"Nineteen","advisorId":101,"courses":[{"co
urseNumber":"CS101","name":"Algorithms","avgGrade":75},{"courseNumber":"MATH102"
,"name":"Calculus","avgGrade":95},{"courseNumber":"MATH103","name":"Advanced Alg
ebra","avgGrade":82}],"dormitory":{"dormId":201,"dormName":"ABC"}}
```

Unlike what happens in the simplified migration case (see "VALIDATE_IMPORT_REPORT for No Outlier Use Case" in Migrating To Duality, Simplified Recipe), validating the import using `DBMS_JSON_DUALITY.validate_import_report` reports an error for the documents that have been imported successfully: the student document for Luis F. has a `null` value for its `age` field, corresponding to the input string value `"Nineteen"`.

PL/SQL function `validate_import_report` compares input documents with documents imported into the duality views, ignoring any additional fields added by the converter (`_id`, `_metadata`, `ora$map*`, `ora$*_flex`). It uses the format of JSON Patch to identify the problematic fields and specify editing operations you can perform on the input data to resolve the problems (differences).

The examples that follow ("Student Duality View Document Collection (useFlexFields = true)", "Teacher Duality View Document Collection (useFlexFields = true)" and "Course Duality View Document Collection (useFlexFields = true)") show the document collections supported by the duality views, that is, the result of importing.

Example 21-24   Student Duality View Document Collection (useFlexFields = true)

Compare this with the input student document set, School Administration Example, Migrator Input Documents, which (with conversion using `minFieldFrequency` = `25` and `minTypeFrequency` = `15`) has only one outlier field: `age` (with a type-occurrence frequency of 10%).

These are the only differences (ignoring field order, which is irrelevant):

- Document identifier field `_id` and document-state field `_metadata` have been added. (Every document supported by a duality view has these fields.)
- Fields `ora$mapCourseId` and `ora$mapStudentId` have been added. These correspond to the identifying columns (primary-key columns in this case) for underlying mapping table

`map_table_course_root_to_student_root`. Their values are the same as the values of fields `courseNumber` and `studentId`, respectively.

- Even though the document for student Luis F. (`studentId` = `9`) *failed import* into the *student* duality view (because field `age` has the *string* value `"Nineteen"`, and its 10% occurrence is a type-occurrence outlier), that document is nevertheless *present* in the duality view. When we import documents into the *course* and *teacher* duality views, a row is added to table `student_root` that has `9` as the value for column `student_root.student_id`, because `studentId` with value `9` is present in both input tables `course_tab` and `teacher_tab`.

  The `age` field value for that student document for Luis F. is *`null`*, however (not `"Nineteen"` and not `19`). No `age` field exists in either of the course or teacher input document sets, so importing their student data for Luis F. into the course and teacher views stores *`NULL`* in the `age` column in table `student_root`. And that `NULL` column value maps to JSON `null` in the student documents.

There are no other differences. In particular, mixed-type field `avgGrade` is unchanged from the input data, as it is not an outlier: each of its types occurs in more than 15% of the documents.

```
{
  "_id" : 1,
  "_metadata" :
  {
    "etag" : "4F39C8B86F4295AD2958B18A77B0AACC",
    "asof" : "0000000000423804"
  },
  "age" : 20,
  "name" : "Donald P.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 1,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 1,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 1,
      "name" : "Algebra",
      "avgGrade" : 90,
      "courseNumber" : "MATH101"
    }
  ],
  "advisorId" : 102,
  "dormitory" :
  {
    "dormId" : 201,
    "dormName" : "ABC"
  },
  "studentId" : 1
}

{
  "_id" : 2,
  "_metadata" :
  {
    "etag" : "758A4F3E6EF3152A4FA0892AB38635D4",
    "asof" : "0000000000423804"
  },
  "age" : 21,
  "name" : "Elena H.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 2,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 2,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 2,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    }
  ],
  "advisorId" : 103,
  "dormitory" :
  {
    "dormId" : 202,
    "dormName" : "XYZ"
  },
  "studentId" : 2
}

{
  "_id" : 3,
  "_metadata" :
  {
    "etag" : "06905F120EF74124C5985354BBCE5CC1",
    "asof" : "0000000000423804"
  },
  "age" : 20,
  "name" : "Francis K.",
  "courses" :
```

```
  [
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 3,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 103,
  "dormitory" :
  {
    "dormId" : 204,
    "dormName" : "QWE"
  },
  "studentId" : 3
}

{
  "_id" : 4,
  "_metadata" :
  {
    "etag" : "50847D1AB63537118A6133A4CC1B8708",
    "asof" : "0000000000423804"
  },
  "age" : 19,
  "name" : "Georgia D.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 4,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 4,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 4,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 101,
  "dormitory" :
  {
    "dormId" : 203,
    "dormName" : "LMN"
  },
  "studentId" : 4
}

{
  "_id" : 5,
  "_metadata" :
  {
    "etag" : "FD6E27A868C56D1EF9C7AEB3F08C7F9B",
    "asof" : "0000000000423804"
  },
  "age" : 21,
  "name" : "Hye E.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 5,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 5,
      "name" : "Algebra",
      "avgGrade" : 90,
      "courseNumber" : "MATH101"
    }
  ],
  "advisorId" : 103,
  "dormitory" :
  {
    "dormId" : 201,
    "dormName" : "ABC"
  },
  "studentId" : 5
}

{
  "_id" : 6,
  "_metadata" :
  {
    "etag" : "2BDA7862330B0687F22F830F3E314E34",
    "asof" : "0000000000423804"
  },
  "age" : 21,
  "name" : "Ileana D.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 6,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 102,
  "dormitory" :
  {
    "dormId" : 205,
    "dormName" : "GHI"
  },
  "studentId" : 6
}

{
  "_id" : 7,
  "_metadata" :
  {
    "etag" : "F1EF0CCD54EDFA78D2263D7E742D6CE8",
    "asof" : "0000000000423804"
  },
  "age" : 20,
  "name" : "Jatin S.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 7,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 7,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    }
  ],
  "advisorId" : 101,
  "dormitory" :
  {
    "dormId" : 204,
    "dormName" : "QWE"
  },
  "studentId" : 7
}

{
  "_id" : 8,
  "_metadata" :
  {
    "etag" : "9A25A267BC08858E0F754E0C00B32F9E",
    "asof" : "0000000000423804"
  },
  "age" : 21,
  "name" : "Katie H.",
  "courses" :
```

```
  [
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 8,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 8,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 102,
  "dormitory" :
  {
    "dormId" : 205,
    "dormName" : "GHI"
  },
  "studentId" : 8
}

{
  "_id" : 10,
  "_metadata" :
  {
    "etag" : "94376DA05B92E47718AF70A31FBE56E7",
    "asof" : "0000000000423804"
  },
  "age" : 20,
  "name" : "Ming L.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 10,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    }
  ],
  "advisorId" : 101,
  "dormitory" :
  {
    "dormId" : 202,
    "dormName" : "XYZ"
  },
  "studentId" : 10
}

{
  "_id" : 9,
  "_metadata" :
```

```
  {
    "etag" : "579824C71904C46901BBA605E8539943",
    "asof" : "0000000000423804"
  },
  "age" : null,
  "name" : "Luis F.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 9,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 9,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 9,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 101,
  "dormitory" :
  {
    "dormId" : 201,
    "dormName" : "ABC"
  },
  "studentId" : 9
}
```

Example 21-25 Teacher Duality View Document Collection (useFlexFields = true)

Compare this with the input teacher document set, in School Administration Example, Migrator Input Documents, which had no outliers.

The only difference (ignoring field order, which is irrelevant) is that document identifier field `_id` and document-state field `_metadata` have been added. (Every document supported by a duality view has these fields.)

Field `phoneNumber` is of mixed type: 50% string and 50% array of strings. (Because it's of mixed type and is not a type-occurrence outlier, it's stored in its own `JSON`-type column.)

```
{
  "_id" : 101,
  "_metadata" :
  {
    "etag" : "F919587CCFAD69F2208B0CDDC80BFAB8",
    "asof" : "000000000042348C"
  },
  "name" : "Abdul J.",
  "salary" : 200000,
  "department" : "Mathematics",
  "phoneNumber" :
  [
    "222-555-011",
    "222-555-012"
  ],
  "coursesTaught" :
  [
    {
      "name" : "Algebra",
      "courseId" : "MATH101",
      "classType" : "Online"
    },
    {
      "name" : "Calculus",
      "courseId" : "MATH102",
      "classType" : "In-person"
    }
  ],
  "studentsAdvised" :
  [
    {
      "name" : "Georgia D.",
      "dormId" : 203,
      "studentId" : 4
    },
    {
      "name" : "Jatin S.",
      "dormId" : 204,
      "studentId" : 7
    },
    {
      "name" : "Luis F.",
      "dormId" : 201,
      "studentId" : 9
    },
    {
      "name" : "Ming L.",
      "dormId" : 202,
      "studentId" : 10
    }
  ]
}

{
  "_id" : 102,
  "_metadata" :
  {
    "etag" : "657E2A688F0A086D948A557ABB1FE3BC",
    "asof" : "000000000042348C"
  },
  "name" : "Betty Z.",
  "salary" : 300000,
  "department" : "Computer Science",
  "phoneNumber" : "222-555-022",
  "coursesTaught" :
  [
    {
      "name" : "Algorithms",
      "courseId" : "CS101",
      "classType" : "Online"
    },
    {
      "name" : "Data Structures",
      "courseId" : "CS102",
      "classType" : "In-person"
    }
  ],
  "studentsAdvised" :
  [
    {
      "name" : "Donald P.",
      "dormId" : 201,
      "studentId" : 1
    },
    {
      "name" : "Ileana D.",
      "dormId" : 205,
      "studentId" : 6
    },
    {
      "name" : "Katie H.",
      "dormId" : 205,
      "studentId" : 8
    }
  ]
}

{
  "_id" : 103,
  "_metadata" :
  {
    "etag" : "1F2DB9CBCD6F7E5E558785D78CA7D116",
    "asof" : "000000000042348C"
  },
  "name" : "Colin J.",
  "salary" : 220000,
  "department" : "Mathematics",
  "phoneNumber" :
  [
    "222-555-023"
  ],
  "coursesTaught" :
  [
    {
      "name" : "Advanced Algebra",
      "courseId" : "MATH103",
      "classType" : "Online"
    }
  ],
  "studentsAdvised" :
  [
    {
      "name" : "Elena H.",
      "dormId" : 202,
      "studentId" : 2
    },
    {
      "name" : "Francis K.",
      "dormId" : 204,
      "studentId" : 3
    },
    {
      "name" : "Hye E.",
      "dormId" : 201,
      "studentId" : 5
    }
  ]
}

{
  "_id" : 104,
  "_metadata" :
  {
    "etag" : "D4D644FB68590D5A00EC53778F0E7226",
    "asof" : "000000000042348C"
  },
  "name" : "Natalie C.",
  "salary" : 180000,
  "department" : "Computer Science",
  "phoneNumber" : "222-555-044",
  "coursesTaught" :
  [
  ],
  "studentsAdvised" :
  [
  ]
}
```

Example 21-26 Course Duality View Document Collection (useFlexFields = true)

Compare this with the input course document set, in School Administration Example, Migrator Input Documents, which (with conversion using `minFieldFrequency` = `25` and `minTypeFrequency` = `15`) has only one outlier field: `Notes` (with a field occurrence frequency of 20%).

Field `Notes` is nevertheless *present* in the duality-view document for course `MATH101`, because conversion was done with `useFlexFields` = `true`, which means the converter created flex columns in the duality views — the importer *stored field `Notes` in a flex column*.

The only other difference from the input documents (ignoring field order, which is irrelevant) is that document identifier field `_id` and document-state field `_metadata` have been added. Every document supported by a duality view has these fields.

Note that there's no difference for field `creditHours`. It's of mixed type, number-or-string (that is, the value can be a number or a string). And even though only one document (for course `MATH103`) uses a string value, the field is not a type-occurrence outlier because a string occurs in one of the five documents (20%), which is greater than `minTypeFrequency` = `15`.

Note too that field `_id` has a string value, such as `"MATH101"`, because it is mapped to input field `courseId`. A document-identifier field need not be a number; its value just needs to uniquely identify a document.

```
{
  "_id" : "CS101",
  "_metadata" :
  {
    "etag" : "FE5B789404D0B9945EB69D7036759CF2",
    "asof" : "0000000000423494"
  },
  "name" : "Algorithms",
  "teacher" :
  {
    "name" : "Betty Z.",
    "teacherId" : 102
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 1,
      "name" : "Donald P.",
      "studentId" : 1
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 2,
      "name" : "Elena H.",
      "studentId" : 2
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 4,
      "name" : "Georgia D.",
      "studentId" : 4
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 7,
      "name" : "Jatin S.",
      "studentId" : 7
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 9,
      "name" : "Luis F.",
      "studentId" : 9
    }
  ],
  "creditHours" : 5,
  "courseId" : "CS101"
}

{
  "_id" : "CS102",
  "_metadata" :
  {
    "etag" : "D2A2D30D1998AAABC4D6EC5FDAFB2472",
    "asof" : "0000000000423494"
  },
  "name" : "Data Structures",
  "teacher" :
  {
    "name" : "Betty Z.",
    "teacherId" : 102
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 1,
      "name" : "Donald P.",
      "studentId" : 1
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 2,
      "name" : "Elena H.",
      "studentId" : 2
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 5,
      "name" : "Hye E.",
      "studentId" : 5
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 7,
      "name" : "Jatin S.",
      "studentId" : 7
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 8,
      "name" : "Katie H.",
      "studentId" : 8
    }
  ],
  "creditHours" : 3,
  "courseId" : "CS102"
}

{
  "_id" : "MATH101",
  "_metadata" :
```

```
  {
    "etag" : "3509714A03884A40BC1EBE0952E3F5CE",
    "asof" : "0000000000423494"
  },
  "name" : "Algebra",
  "teacher" :
  {
    "name" : "Abdul J.",
    "teacherId" : 101
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 1,
      "name" : "Donald P.",
      "studentId" : 1
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 5,
      "name" : "Hye E.",
      "studentId" : 5
    }
  ],
  "creditHours" : 3,
  "Notes" : "Prerequisite for Advanced Algebra",
  "courseId" : "MATH101"
}

{
  "_id" : "MATH102",
  "_metadata" :
  {
    "etag" : "3193D7B3FC1EC95210D4ABF12DF1558E",
    "asof" : "0000000000423494"
  },
  "name" : "Calculus",
  "teacher" :
  {
    "name" : "Abdul J.",
    "teacherId" : 101
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 2,
      "name" : "Elena H.",
      "studentId" : 2
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 4,
      "name" : "Georgia D.",
      "studentId" : 4
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 9,
      "name" : "Luis F.",
      "studentId" : 9
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 10,
      "name" : "Ming L.",
      "studentId" : 10
    }
  ],
  "creditHours" : 4,
  "courseId" : "MATH102"
}

{
  "_id" : "MATH103",
  "_metadata" :
  {
    "etag" : "8AC5912C1CB56D431FF4F979EB121E60",
    "asof" : "0000000000423494"
  },
  "name" : "Advanced Algebra",
  "teacher" :
  {
    "name" : "Colin J.",
    "teacherId" : 103
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 3,
      "name" : "Francis K.",
      "studentId" : 3
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 4,
      "name" : "Georgia D.",
      "studentId" : 4
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 6,
      "name" : "Ileana D.",
      "studentId" : 6
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 8,
      "name" : "Katie H.",
      "studentId" : 8
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 9,
      "name" : "Luis F.",
      "studentId" : 9
    }
  ],
  "creditHours" : "3",
  "courseId" : "MATH103"
}
```

The example that follows, "Create JSON Data Guides for Document Collections Supported By Duality Views" , creates data-guide JSON schemas for each of the duality views, that is, for the document sets supported by the views. You can compare the schema for each duality view with a data-guide JSON schema that describes the corresponding input document set (see the example "Create JSON Data Guides for Input Document Sets" in Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas).

A data-guide schema serves as a shortcut (proxy) for comparing the documents supported by a duality view with the corresponding input documents. Such comparison can help decide how you might want to (1) change some of the documents, (2) change some of the configuration fields used to infer and generate the database objects, or (3) change the definition of a duality view or table.

It's important to note that comparing JSON schemas between input and output database objects (input transfer table and output duality view) is not the same as comparing the input and output documents. Comparing schemas can suggest things you might want to change, but it isn't a complete substitute for comparing documents. After you import the original documents into the duality views you can and should compare documents.

Example 21-27   Create JSON Data Guides For Document Collections Supported By Duality Views

This code is identical to the data-guide JSON schema creation code in the example "Create JSON Data Guides for Input Document Sets" in Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas, except that the data guides here are created on duality views, not input tables.

```
SELECT json_dataguide(data,
                      DBMS_JSON.FORMAT_SCHEMA,
                      DBMS_JSON.PRETTY+DBMS_JSON.GATHER_STATS)
  FROM student;

SELECT json_dataguide(data,
                      DBMS_JSON.FORMAT_SCHEMA,
                      DBMS_JSON.PRETTY+DBMS_JSON.GATHER_STATS)
  FROM teacher;

SELECT json_dataguide(data,
                      DBMS_JSON.FORMAT_SCHEMA,
                      DBMS_JSON.PRETTY+DBMS_JSON.GATHER_STATS)
  FROM course;
```

Example 21-28   Student Duality View Data Guide

This data guide JSON schema summarizes the collection of student documents supported by duality view `student`.

The differences from the data guide for the student input documents, in the example "Create JSON Data Guides for Input Student Document Set" in Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas) reflect the differences between the two sets of student documents (see "Student Duality View Document Collection (useFlexFields = true)" in Import After Default Conversion).

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 10,
  "required" : true,
  "properties" :
  {
    "_id" :
    {
      "type" : "number",
      "o:preferred_column_name" : "_id",
      "o:frequency" : 100,
      "o:low_value" : 1,
      "o:high_value" : 10,
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 10,
      "required" : true,
      "maximum" : 10,
      "minimum" : 1
    },
    "age" :
    {
      "oneOf" :
      [
        {
          "type" : "null",
          "o:preferred_column_name" : "age",
          "o:frequency" : 10,
          "o:low_value" : null,
          "o:high_value" : null,
          "o:num_nulls" : 1,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 10
        },
        {
          "type" : "number",
          "o:preferred_column_name" : "age",
          "o:frequency" : 90,
          "o:low_value" : 19,
          "o:high_value" : 21,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 10,
          "maximum" : 21,
          "minimum" : 19
        }
      ]
    },
    "name" :
    {
      "type" : "string",
      "o:length" : 16,
      "o:preferred_column_name" : "name",
      "o:frequency" : 100,
      "o:low_value" : "Donald P.",
      "o:high_value" : "Ming L.",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 10,
      "required" : true,
      "maxLength" : 10,
      "minLength" : 6
    },
    "courses" :
    {
      "type" : "array",
      "o:preferred_column_name" : "courses",
      "o:frequency" : 100,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 10,
      "required" : true,
      "items" :
      {
        "properties" :
        {
          "name" :
          {
            "type" : "string",
            "o:length" : 16,
            "o:preferred_column_name" : "name",
            "o:frequency" : 100,
            "o:low_value" : "Advanced Algebra",
            "o:high_value" : "Data Structures",
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2025-01-10T17:29:04",
            "o:sample_size" : 10,
            "required" : true,
            "maxLength" : 16,
            "minLength" : 7
          },
          "avgGrade" :
          {
            "oneOf" :
            [
              {
                "type" : "number",
                "o:preferred_column_name" : "avgGrade",
                "o:frequency" : 100,
                "o:low_value" : 75,
              "o:high_value" : 95,
              "o:num_nulls" : 0,
              "o:last_analyzed" : "2025-01-10T17:29:04",
              "o:sample_size" : 10,
              "required" : true,
              "maximum" : 95,
              "minimum" : 75
            },
            {
              "type" : "string",
              "o:length" : 4,
              "o:preferred_column_name" : "avgGrade",
              "o:frequency" : 50,
              "o:low_value" : "TBD",
              "o:high_value" : "TBD",
              "o:num_nulls" : 0,
              "o:last_analyzed" : "2025-01-10T17:29:04",
              "o:sample_size" : 10,
              "maxLength" : 3,
              "minLength" : 3
            }
          ]
        },
        "courseNumber" :
        {
          "type" : "string",
          "o:length" : 8,
          "o:preferred_column_name" : "courseNumber",
          "o:frequency" : 100,
          "o:low_value" : "CS101",
          "o:high_value" : "MATH103",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 10,
          "required" : true,
          "maxLength" : 7,
          "minLength" : 5
        },
        "ora$mapCourseId" :
        {
          "type" : "string",
          "o:length" : 8,
          "o:preferred_column_name" : "ora$mapCourseId",
          "o:frequency" : 100,
          "o:low_value" : "CS101",
          "o:high_value" : "MATH103",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 10,
          "required" : true,
          "maxLength" : 7,
          "minLength" : 5
        },
        "ora$mapStudentId" :
        {
          "type" : "number",
          "o:preferred_column_name" : "ora$mapStudentId",
          "o:frequency" : 100,
          "o:low_value" : 1,
          "o:high_value" : 10,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 10,
          "required" : true,
          "maximum" : 10,
          "minimum" : 1
        }
      }
    }
  },
  "_metadata" :
  {
    "type" : "object",
    "o:preferred_column_name" : "_metadata",
    "o:frequency" : 100,
    "o:last_analyzed" : "2025-01-10T17:29:04",
    "o:sample_size" : 10,
    "required" : true,
    "properties" :
    {
      "asof" :
      {
        "type" : "binary",
        "o:length" : 8,
        "o:preferred_column_name" : "asof",
        "o:frequency" : 100,
        "o:low_value" : "",
        "o:high_value" : "",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 10,
        "required" : true
      },
      "etag" :
      {
        "type" : "binary",
        "o:length" : 16,
        "o:preferred_column_name" : "etag",
        "o:frequency" : 100,
        "o:low_value" : "",
        "o:high_value" : "",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 10,
        "required" : true
      }
    }
  },
  "advisorId" :
  {
    "type" : "number",
    "o:preferred_column_name" : "advisorId",
"o:frequency" : 100,
"o:low_value" : 101,
"o:high_value" : 103,
"o:num_nulls" : 0,
"o:last_analyzed" : "2025-01-10T17:29:04",
"o:sample_size" : 10,
"required" : true,
"maximum" : 103,
"minimum" : 101
},
"dormitory" :
{
"type" : "object",
"o:preferred_column_name" : "dormitory",
"o:frequency" : 100,
"o:last_analyzed" : "2025-01-10T17:29:04",
"o:sample_size" : 10,
"required" : true,
"properties" :
{
  "dormId" :
  {
    "type" : "number",
    "o:preferred_column_name" : "dormId",
    "o:frequency" : 100,
    "o:low_value" : 201,
    "o:high_value" : 205,
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2025-01-10T17:29:04",
    "o:sample_size" : 10,
    "required" : true,
    "maximum" : 205,
    "minimum" : 201
  },
  "dormName" :
  {
    "type" : "string",
    "o:length" : 4,
    "o:preferred_column_name" : "dormName",
    "o:frequency" : 100,
    "o:low_value" : "ABC",
    "o:high_value" : "XYZ",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2025-01-10T17:29:04",
    "o:sample_size" : 10,
    "required" : true,
    "maxLength" : 3,
    "minLength" : 3
  }
}
},
"studentId" :
{
"type" : "number",
"o:preferred_column_name" : "studentId",
"o:frequency" : 100,
      "o:low_value" : 1,
      "o:high_value" : 10,
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 10,
      "required" : true,
      "maximum" : 10,
      "minimum" : 1
    }
  }
}
```

Example 21-29 Teacher Duality View Data Guide

This data guide JSON schema summarizes the collection of teacher documents supported by duality view `teacher`.

The differences from the data guide for the teacher input documents in the example "JSON Data Guide for Input Teacher Document Set" in Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas reflect the differences between the two sets of teacher documents (see "Teacher Duality View Document Collection (useFlexFields = true)" in Import After Default Conversion).

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 4,
  "required" : true,
  "properties" :
  {
    "_id" :
    {
      "type" : "number",
      "o:preferred_column_name" : "_id",
      "o:frequency" : 100,
      "o:low_value" : 101,
      "o:high_value" : 104,
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 4,
      "required" : true,
      "maximum" : 104,
      "minimum" : 101
    },
    "name" :
    {
      "type" : "string",
      "o:length" : 16,
      "o:preferred_column_name" : "name",
      "o:frequency" : 100,
      "o:low_value" : "Abdul J.",
      "o:high_value" : "Natalie C.",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 4,
  "required" : true,
  "maxLength" : 10,
  "minLength" : 8
},
"salary" :
{
  "type" : "number",
  "o:preferred_column_name" : "salary",
  "o:frequency" : 100,
  "o:low_value" : 180000,
  "o:high_value" : 300000,
  "o:num_nulls" : 0,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 4,
  "required" : true,
  "maximum" : 300000,
  "minimum" : 180000
},
"_metadata" :
{
  "type" : "object",
  "o:preferred_column_name" : "_metadata",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 4,
  "required" : true,
  "properties" :
  {
    "asof" :
    {
      "type" : "binary",
      "o:length" : 8,
      "o:preferred_column_name" : "asof",
      "o:frequency" : 100,
      "o:low_value" : "",
      "o:high_value" : "",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 4,
      "required" : true
    },
    "etag" :
    {
      "type" : "binary",
      "o:length" : 16,
      "o:preferred_column_name" : "etag",
      "o:frequency" : 100,
      "o:low_value" : "",
      "o:high_value" : "",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 4,
      "required" : true
    }
  }
},
"department" :
{
  "type" : "string",
  "o:length" : 16,
  "o:preferred_column_name" : "department",
  "o:frequency" : 100,
  "o:low_value" : "Computer Science",
  "o:high_value" : "Mathematics",
  "o:num_nulls" : 0,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 4,
  "required" : true,
  "maxLength" : 16,
  "minLength" : 11
},
"phoneNumber" :
{
  "oneOf" :
  [
    {
      "type" : "string",
      "o:length" : 16,
      "o:preferred_column_name" : "phoneNumber",
      "o:frequency" : 50,
      "o:low_value" : "222-555-022",
      "o:high_value" : "222-555-044",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 4,
      "maxLength" : 11,
      "minLength" : 11
    },
    {
      "type" : "array",
      "o:preferred_column_name" : "phoneNumber",
      "o:frequency" : 50,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 4,
      "items" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "scalar_string",
        "o:frequency" : 50,
        "o:low_value" : "222-555-011",
        "o:high_value" : "222-555-023",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maxLength" : 11,
        "minLength" : 11
      }
    }
  ]
},
"coursesTaught" :
```

```
{
  "type" : "array",
  "o:preferred_column_name" : "coursesTaught",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 4,
  "required" : true,
  "items" :
  {
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "name",
        "o:frequency" : 75,
        "o:low_value" : "Advanced Algebra",
        "o:high_value" : "Data Structures",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maxLength" : 16,
        "minLength" : 7
      },
      "courseId" :
      {
        "type" : "string",
        "o:length" : 8,
        "o:preferred_column_name" : "courseId",
        "o:frequency" : 75,
        "o:low_value" : "CS101",
        "o:high_value" : "MATH103",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maxLength" : 7,
        "minLength" : 5
      },
      "classType" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "classType",
        "o:frequency" : 75,
        "o:low_value" : "In-person",
        "o:high_value" : "Online",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maxLength" : 9,
        "minLength" : 6
      }
    }
  }
},
"studentsAdvised" :
{
  "type" : "array",
  "o:preferred_column_name" : "studentsAdvised",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 4,
  "required" : true,
  "items" :
  {
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 16,
        "o:preferred_column_name" : "name",
        "o:frequency" : 75,
        "o:low_value" : "Donald P.",
        "o:high_value" : "Ming L.",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maxLength" : 10,
        "minLength" : 6
      },
      "dormId" :
      {
        "type" : "number",
        "o:preferred_column_name" : "dormId",
        "o:frequency" : 75,
        "o:low_value" : 201,
        "o:high_value" : 205,
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maximum" : 205,
        "minimum" : 201
      },
      "studentId" :
      {
        "type" : "number",
        "o:preferred_column_name" : "studentId",
        "o:frequency" : 75,
        "o:low_value" : 1,
        "o:high_value" : 10,
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 4,
        "maximum" : 10,
        "minimum" : 1
      }
    }
  }
}
```

```
  }
}
```

Example 21-30   Course Duality View Data Guide, for Default Case

This data guide JSON schema summarizes the collection of course documents supported by duality view `course`, for the conversion case where `useFlexFields` is `true`.

The differences from the data guide for the course input documents, in the example "JSON Data Guide for INput Course Document Set" in Before Using the Converter (2): Optionally Create Data-Guide JSON Schemas reflect the differences between the two sets of course documents (see "Course Duality View Document Collection (useFlexFields = true)" in Import After Default Conversion).

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 5,
  "required" : true,
  "properties" :
  {
    "_id" :
    {
      "type" : "string",
      "o:length" : 8,
      "o:preferred_column_name" : "_id",
      "o:frequency" : 100,
      "o:low_value" : "CS101",
      "o:high_value" : "MATH103",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 5,
      "required" : true,
      "maxLength" : 7,
      "minLength" : 5
    },
    "name" :
    {
      "type" : "string",
      "o:length" : 16,
      "o:preferred_column_name" : "name",
      "o:frequency" : 100,
      "o:low_value" : "Advanced Algebra",
      "o:high_value" : "Data Structures",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 5,
      "required" : true,
      "maxLength" : 16,
      "minLength" : 7
    },
    "Notes" :
    {
      "type" : "string",
      "o:length" : 64,
  "o:preferred_column_name" : "Notes",
  "o:frequency" : 20,
  "o:low_value" : "Prerequisite for Advanced Algebra",
  "o:high_value" : "Prerequisite for Advanced Algebra",
  "o:num_nulls" : 0,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 5,
  "maxLength" : 33,
  "minLength" : 33
},
"teacher" :
{
  "type" : "object",
  "o:preferred_column_name" : "teacher",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 5,
  "required" : true,
  "properties" :
  {
    "name" :
    {
      "type" : "string",
      "o:length" : 8,
      "o:preferred_column_name" : "name",
      "o:frequency" : 100,
      "o:low_value" : "Abdul J.",
      "o:high_value" : "Colin J.",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 5,
      "required" : true,
      "maxLength" : 8,
      "minLength" : 8
    },
    "teacherId" :
    {
      "type" : "number",
      "o:preferred_column_name" : "teacherId",
      "o:frequency" : 100,
      "o:low_value" : 101,
      "o:high_value" : 103,
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 5,
      "required" : true,
      "maximum" : 103,
      "minimum" : 101
    }
  }
},
"courseId" :
{
  "type" : "string",
  "o:length" : 8,
  "o:preferred_column_name" : "courseId",
    "o:frequency" : 100,
    "o:low_value" : "CS101",
    "o:high_value" : "MATH103",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2025-01-10T17:29:04",
    "o:sample_size" : 5,
    "required" : true,
    "maxLength" : 7,
    "minLength" : 5
  },
  "students" :
  {
    "type" : "array",
    "o:preferred_column_name" : "students",
    "o:frequency" : 100,
    "o:last_analyzed" : "2025-01-10T17:29:04",
    "o:sample_size" : 5,
    "required" : true,
    "items" :
    {
      "properties" :
      {
        "name" :
        {
          "type" : "string",
          "o:length" : 16,
          "o:preferred_column_name" : "name",
          "o:frequency" : 100,
          "o:low_value" : "Donald P.",
          "o:high_value" : "Ming L.",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 5,
          "required" : true,
          "maxLength" : 10,
          "minLength" : 6
        },
        "studentId" :
        {
          "type" : "number",
          "o:preferred_column_name" : "studentId",
          "o:frequency" : 100,
          "o:low_value" : 1,
          "o:high_value" : 10,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-10T17:29:04",
          "o:sample_size" : 5,
          "required" : true,
          "maximum" : 10,
          "minimum" : 1
        },
        "ora$mapCourseId" :
        {
          "type" : "string",
          "o:length" : 8,
          "o:preferred_column_name" : "ora$mapCourseId",
        "o:frequency" : 100,
        "o:low_value" : "CS101",
        "o:high_value" : "MATH103",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 5,
        "required" : true,
        "maxLength" : 7,
        "minLength" : 5
      },
      "ora$mapStudentId" :
      {
        "type" : "number",
        "o:preferred_column_name" : "ora$mapStudentId",
        "o:frequency" : 100,
        "o:low_value" : 1,
        "o:high_value" : 10,
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-10T17:29:04",
        "o:sample_size" : 5,
        "required" : true,
        "maximum" : 10,
        "minimum" : 1
      }
    }
  }
},
"_metadata" :
{
  "type" : "object",
  "o:preferred_column_name" : "_metadata",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-10T17:29:04",
  "o:sample_size" : 5,
  "required" : true,
  "properties" :
  {
    "asof" :
    {
      "type" : "binary",
      "o:length" : 8,
      "o:preferred_column_name" : "asof",
      "o:frequency" : 100,
      "o:low_value" : "",
      "o:high_value" : "",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-10T17:29:04",
      "o:sample_size" : 5,
      "required" : true
    },
    "etag" :
    {
      "type" : "binary",
      "o:length" : 16,
      "o:preferred_column_name" : "etag",
      "o:frequency" : 100,
              "o:low_value" : "",
              "o:high_value" : "",
              "o:num_nulls" : 0,
              "o:last_analyzed" : "2025-01-10T17:29:04",
              "o:sample_size" : 5,
              "required" : true
            }
          }
        },
        "creditHours" :
        {
          "oneOf" :
          [
            {
              "type" : "number",
              "o:preferred_column_name" : "creditHours",
              "o:frequency" : 80,
              "o:low_value" : 3,
              "o:high_value" : 5,
              "o:num_nulls" : 0,
              "o:last_analyzed" : "2025-01-10T17:29:04",
              "o:sample_size" : 5,
              "maximum" : 5,
              "minimum" : 3
            },
            {
              "type" : "string",
              "o:length" : 1,
              "o:preferred_column_name" : "creditHours",
              "o:frequency" : 20,
              "o:low_value" : "3",
              "o:high_value" : "3",
              "o:num_nulls" : 0,
              "o:last_analyzed" : "2025-01-10T17:29:04",
              "o:sample_size" : 5,
              "maxLength" : 1,
              "minLength" : 1
            }
          ]
        }
      }
    }
```

You can also use `DBMS_JSON_SCHEMA.describe` to create a JSON schema that shows different information about the duality views.

See Also:

- DBMS_JSON_DUALITY in *Oracle Database PL/SQL Packages and Types Reference* for information about function `validate_import_report`
- JSON Patch and JSON Pointer for information about the error content reported by `DBMS_JSON_DUALITY.validate_import_report`
- JSON Schemas Generated with DBMS_JSON_SCHEMA.DESCRIBE in *Oracle Database JSON Developer's Guide*
- DESCRIBE Function in *Oracle Database PL/SQL Packages and Types Reference*

21.12 Using the Converter with useFlexFields=false

Use of the JSON-to-duality converter with `useFlexFields` = `false` is illustrated. Otherwise the configuration is default (except for `minFieldFrequency` and `minTypeFrequency`). The database objects needed to support the document sets are inferred, and the SQL DDL code to construct them is generated.

We pass `useFlexFields` with a *`false`* value to `DBMS_JSON_DUALITY.infer_schema`. Otherwise, that call is the same as in the example "INFER_SCHEMA and GENERATE_SCHEMA with useFlexFields = true" in Using the Converter, Default Behavior. And the JSON schema returned by `infer_schema` is the same for both `true` and `false` `useFlexFields`.

The call to `DBMS_JSON_DUALITY.generate_schema` is also the same. The only differences in the generated DDL code are these:

- The tables underlying the duality views have no flex columns when `useFlexFields` is `false`.
- The duality views don't refer to flex columns when `useFlexFields` is `false`.

Example 21-31 DDL Code from GENERATE_SCHEMA with useFlexFields = false

Function `DBMS_JSON_DUALITY.generate_schema`, produces the generated DDL code shown here if passed the JSON schema produced by function `infer_schema` with `useFlexFields` = `false`.

The only difference in this DDL code from that generated with `useFlexFields` = `true` is that here there are no flex columns in the underlying tables and no references to flex columns in the duality views.

```
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE student_dormitory(
   dorm_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   dorm_name  varchar2(64),
   PRIMARY KEY(dorm_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE map_course_root_to_student_root(
   map_course_id  varchar2(64)  DEFAULT ON NULL SYS_GUID(),
   map_student_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   PRIMARY KEY(map_course_id,map_student_id)
)';
```

```
EXECUTE IMMEDIATE 'CREATE TABLE student_root(
   age  number,
   name  varchar2(64),
   dorm_id  number,
   advisor_id  number,
   student_id  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   PRIMARY KEY(student_id)
)';

EXECUTE IMMEDIATE 'CREATE TABLE teacher_root(
   "_id"  number  GENERATED BY DEFAULT ON NULL AS IDENTITY,
   name  varchar2(64),
   salary  number,
   department  varchar2(64),
   phone_number  json VALIDATE ''{"oneOf" : [{ "type" :"string"},
                                          { "type" :"array"}]}'',
   PRIMARY KEY("_id")
)';

EXECUTE IMMEDIATE 'CREATE TABLE course_root(
   name  varchar2(64),
   avg_grade  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
                                       { "type" :"string"}]}'',
   course_id  varchar2(64)  DEFAULT ON NULL SYS_GUID(),
   class_type  varchar2(64),
   credit_hours  json VALIDATE ''{"oneOf" : [{ "type" :"number"},
                                          { "type" :"string"}]}'',
   "_id_teacher_root"  number,
   PRIMARY KEY(course_id)
)';

EXECUTE IMMEDIATE 'ALTER TABLE map_course_root_to_student_root
ADD CONSTRAINT fk_map_course_root_to_student_root_to_course_root FOREIGN KEY
(map_course_id) REFERENCES course_root(course_id) DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE map_course_root_to_student_root
ADD CONSTRAINT fk_map_course_root_to_student_root_to_student_root FOREIGN KEY
(map_student_id) REFERENCES student_root(student_id) DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE student_root
ADD CONSTRAINT fk_student_root_to_student_dormitory FOREIGN KEY (dorm_id)
REFERENCES student_dormitory(dorm_id) DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE student_root
ADD CONSTRAINT fk_student_root_to_teacher_root FOREIGN KEY (advisor_id)
REFERENCES teacher_root("_id") DEFERRABLE';
EXECUTE IMMEDIATE 'ALTER TABLE course_root
ADD CONSTRAINT fk_course_root_to_teacher_root FOREIGN KEY
("_id_teacher_root") REFERENCES teacher_root("_id") DEFERRABLE';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_map_course_root_to_student_root_to_course_root_index ON
map_course_root_to_student_root(map_course_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_map_course_root_to_student_root_to_student_root_index ON
map_course_root_to_student_root(map_student_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_student_root_to_student_dormitory_index ON student_root(dorm_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
```

```
fk_student_root_to_teacher_root_index ON student_root(advisor_id)';
EXECUTE IMMEDIATE 'CREATE INDEX IF NOT EXISTS
fk_course_root_to_teacher_root_index ON course_root("_id_teacher_root")';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW STUDENT AS
student_root @insert @update @delete
{
  _id : student_id
  age
  name
  courses: map_course_root_to_student_root @insert @update @delete @array
  {
    ora$mapCourseId: map_course_id
    ora$mapStudentId: map_student_id
    course_root @unnest @insert @update @object
    {
      name
      avgGrade: avg_grade
      courseNumber: course_id
    }
  }
  advisorId:advisor_id
  dormitory: student_dormitory @insert @update @object
  {
    dormId: dorm_id
    dormName: dorm_name
  }
  studentId @generated (path: "$._id")
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW TEACHER AS
teacher_root @insert @update @delete
{
  "_id"
  name
  salary
  department
  phoneNumber: phone_number
  coursesTaught: course_root @insert @update @delete @array
  {
    name
    courseId: course_id
    classType: class_type
  }
  studentsAdvised: student_root @insert @update @delete @array
  {
    name
    dormId:dorm_id
    studentId: student_id
  }
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW COURSE AS
course_root @insert @update @delete
{
  _id : course_id
  name
  teacher: teacher_root @insert @update @object
  {
    name
    teacherId: "_id"
  }
  courseId @generated (path: "$._id")
  students: map_course_root_to_student_root @insert @update @delete @array
  {
    ora$mapCourseId: map_course_id
    ora$mapStudentId: map_student_id
    student_root @unnest @insert @update @object
    {
      name
      studentId: student_id
    }
  }
  creditHours: credit_hours
}';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER INSERT_TRIGGER_STUDENT
  BEFORE INSERT
  ON STUDENT
  FOR EACH ROW
DECLARE
  inp_jsonobj json_object_t;
BEGIN
  inp_jsonobj := json_object_t(:new.data);
  IF NOT inp_jsonobj.has(''_id'')
  THEN
    inp_jsonobj.put(''_id'', inp_jsonobj.get(''studentId''));
    :new.data := inp_jsonobj.to_json;
  END IF;
END;';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER INSERT_TRIGGER_COURSE
  BEFORE INSERT
  ON COURSE
  FOR EACH ROW
DECLARE
  inp_jsonobj json_object_t;
BEGIN
  inp_jsonobj := json_object_t(:new.data);
  IF NOT inp_jsonobj.has(''_id'')
  THEN
    inp_jsonobj.put(''_id'', inp_jsonobj.get(''courseId''));
    :new.data := inp_jsonobj.to_json;
  END IF;
END;';
END;
```

The example that follows, "VALIDATE_SCHEMA_REPORT with useFlexFields = false", shows that rare field `Notes` is an outlier for the course document set.

With no flex columns created by the converter, errors will be logged during *import* for any fields that are unmapped by the converter (for example, fields that can't be stored in a simple SQL scalar column). When `useFlexFields` is `true` (the default value) unmapped field `Notes` is retained in course documents, by being stored in a flex field. But with `useFlexFields false` field `Notes` is logged during import as an error.

When you generate the DDL code and then execute it, the duality views and their underlying tables are created.

After executing the DDL code, you run converter function `DBMS_JSON_DUALITY.validate_schema_report` for each kind (student, teacher, course) of input table and duality view, to validate the conversion.

Example 21-32 VALIDATE_SCHEMA_REPORT with useFlexFields = false

This code is the same as that in the example "VALIDATE_SCHEMA_REPORT for Default Case (useFlexFields = true) " in Using the Converter, Default Behavior.

```
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'STUDENT_TAB',
                                  view_name  => 'STUDENT');
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name  => 'TEACHER');
SELECT * FROM DBMS_JSON_DUALITY.validate_schema_report(
                                  table_name => 'COURSE_TAB',
                                  view_name  => 'COURSE');
```

The error reported for the student table and view is the same as in the example "VALIDATE_SCHEMA_REPORT for Default Case (useFlexFields = true) " in Using the Converter, Default Behavior. (That is, the string value of `"Nineteen"` for field `age`).

There are no errors reported for the teacher table and view (just as in the `useFlexFields` = `true` case).

Instead of no errors reported for the course table and view (as in the `useFlexFields` = `true` case), however, an error is reported for field occurrence-outlier field `Notes` in the `useFlexFields` = `false` case. This is because in the *default* case that field is stored in a flex column.

```
DATA
----
{"courseId":"MATH101","name":"Algebra","creditHours":3,"students":[{"studentId":
1,"name":"Donald P."},{"studentId":5,"name":"Hye E."}],"teacher":{"teacherId":10
1,"name":"Abdul J."},"Notes":"Prerequisite for Advanced Algebra"}



ERRORS
------
[{"schemaPath":"$","instancePath":"$","code":"JZN-00501","error":"JSON schema va
lidation failed"},{"schemaPath":"$.additionalProperties","instancePath":"$","cod
e":"JZN-00518","error":"invalid additional properties:  'Notes'"},{"schemaPath":
"$.additionalProperties","instancePath":"$.Notes","code":"JZN-00502","error":"JS
ON boolean schema was false"}]
```

Just as for the `useFlexFields` = `true` case, before importing you could choose to change the `age` field in the student input document for Luis J., to give a number value of `19` instead of the string value "Nineteen". And you might want to remove field Notes from the data or relax (lower) the value of minTypeFrequency to allow its inclusion.

- DBMS_JSON_DUALITY in *Oracle Database PL/SQL Packages and Types Reference* for information about subprograms generate_schema, infer_schema, and validate_import_report
- VALIDATE_REPORT Function in *Oracle Database PL/SQL Packages and Types Reference* for information about function DBMS_JSON_SCHEMA.validate_report

21.13 Import After Conversion with useFlexFields=false

After trying to import, error-log tables are created and queried to show import errors and imported documents.

The process of creating error logs and importing the input document sets (in tables student_tab, teacher_tab, and course_tab) into the duality views created in Using the Converter with useFlexFields=false is exactly the same as in the simplified recipe case: see the example "Creating Error Logs for No Outlier Use Case" and "Importing Document Sets, for No Outlier Use Case" in Migrating To Duality, Simplified Recipe. But checking the error logs for the default case tells a different story.

Example 21-33 Checking Error Logs from Import, for useFlexFields = false Case

As with the default case (see "Checking Error Logs from Import, for Default Case" in Import After Default Conversion), import into the student duality view logs an error for the type-occurrence outlier for field age with value "Nineteen", and no error is logged for the teacher view.

But unlike the default case, import also logs an error for the missing Notes field. Field Notes is not mapped to any column, and since there are no flex columns, the field is not supported by the duality view.

```
ORA_ERR_NUMBER$
---------------
ORA_ERR_MESG$
-------------
ORA_ERR_TAG$
------------
42555

ORA-42555: Cannot insert into JSON Relational Duality View 'COURSE': The input JSON document
is invalid.
JZN-00651: field 'Notes' is unknown or undefined

Import Error
```

Select that culprit course document from the input course table:

```
SELECT * FROM "JANUS".course_tab
  WHERE ROWID IN (SELECT ora_err_rowid$ FROM course_err_log);
```

```
DATA
----
```

```
{"courseId":"MATH101","name":"Algebra","creditHours":3,"students":[{"studentId":
1,"name":"Donald P."},{"studentId":5,"name":"Hye E."}],"teacher":{"teacherId":10
1,"name":"Abdul J."},"Notes":"Prerequisite for Advanced Algebra"}
```

We next use `DBMS_JSON_DUALITY.validate_import_report` to report on any problems with *documents that have been imported* successfully. Unlike the default case and the simplified recipe case, for the conversion with `useFlexFields` = `false`, there are validation problems for imported student and course documents. (There are no validation problems for imported teacher documents.)

Example 21-34 VALIDATE_IMPORT_REPORT for useFlexFields = false Case

There are no validation problems for imported teacher documents.

For imported *student* data, the problematic document with `age` having a string value is reported.

```
SELECT * FROM DBMS_JSON_DUALITY.validate_import_report(
                                  table_name => 'STUDENT_TAB',
                                  view_name  => 'STUDENT');
```

```
DATA
----
{"studentId":9,"name":"Luis F.","age":"Nineteen","advisorId":101,"courses":[{"co
urseNumber":"CS101","name":"Algorithms","avgGrade":75},{"courseNumber":"MATH102"
,"name":"Calculus","avgGrade":95},{"courseNumber":"MATH103","name":"Advanced Alg
ebra","avgGrade":82}],"dormitory":{"dormId":201,"dormName":"ABC"}}
```

```
ERRORS
------
[{"op":"replace","path":"/age","value":null}]
```

PL/SQL function `validate_import_report` compares input documents with documents imported into the duality views, ignoring any additional fields added by the converter (`_id`, `_metadata`, `ora$map*`, `ora$*_flex`). It uses the format of JSON Patch to identify the problematic fields and specify editing operations you can perform on the input data to resolve the problems (differences).

- Field **`path`** specifies the location — the syntax of its value is that of JSON Pointer, *not* that of a SQL/JSON path expression. In this case, the `path` value `/age` targets the `age` field at the top level of the document.
- Field **`op`** specifies the editing operation. For the problematic student document, the operation is to replace the value of its top-level field `age` with JSON `null`. (That may or may not be the resolution you want.)

The problematic student document fails to import into the `student` duality view. However, it is still "supported" by that view. It is present in the view because importing to views `course` and `teacher` causes a row to be added to underlying table `student_root` for that document (with `student_id` = `9`).

The value of field `age` in that document has value (JSON) `null`, however, because there's no field in the other two document sets that maps to column `student_root.age`, so that the value of that column is SQL `NULL`. And that `NULL` column value maps to JSON `null` in the student documents.

For imported *course* data, the problematic document containing field `Notes` is reported.

```
SELECT * FROM DBMS_JSON_DUALITY.validate_import_report(
                                   table_name => 'COURSE_TAB',
                                   view_name  => 'COURSE');
```

```
DATA
----
{"courseId":"MATH101","name":"Algebra","creditHours":3,"students":[{"studentId":
[{"op":"remove","path":"/Notes"},{"op":"remove","path":"/creditHours"}]
1,"name":"Donald P."},{"studentId":5,"name":"Hye E."}],"teacher":{"teacherId":10
1,"name":"Abdul J."},"Notes":"Prerequisite for Advanced Algebra"}
```

```
ERRORS
------
[{"op":"remove","path":"/Notes"},{"op":"remove","path":"/creditHours"}]
```

The problematic course document fails to import into the `course` duality view.

For that document there are two error operations reported: *remove* its top-level fields `Notes` and `creditHours`. (This may or may not be the resolution you want.)

Import of the problematic course document fails because of its field `Notes`, which was pruned because, as an occurrence outlier it wasn't mapped to any column. And as an unmapped field the importer can't store it in a flex column because there are no flex columns (the input data was converted with `useFlexFields` = `false`).

But the `course` view's underlying table `course_root` anyway gets a row that corresponds to that problematic document (where field `courseId` has value `MATH101`), *because of importing the student and teacher data*, that is, populating the `student` and `teacher` views. Importing to those views populates columns `course_id` and `name` of table `course_root`, which are used by student and teacher documents. It does *not*, however, populate field `Notes` or `creditHours`.

The examples that follow, "Student Duality View Document Collection (useFlexFields = False)" and "Course Duality View Document Collection (useFlexFields = false) ", show the student and course document collections supported by the duality views, that is, the result of importing into those views, respectively.

The teacher duality-view collection is the same as for conversion with `useFlexFields` = `true` — see the example "Teacher Duality View Document Collection (useFlexFields = true)" in Import After Default Conversion.

Example 21-35 Student Duality View Document Collection (useFlexFields = false)

Compare this with the input student document set, "Student Document Set (Migrator Input)" in School Administration Example, Migrator Input Documents, which (with conversion using `minFieldFrequency` = `25` and `minTypeFrequency` = `15`) has only one outlier field: `age` (with a type-occurrence frequency of 10%).

These are the only differences (ignoring field order, which is irrelevant):

- Document identifier field `_id` and document-state field `_metadata` have been added. (Every document supported by a duality view has these fields.)

- Fields `ora$mapCourseId` and `ora$mapStudentId` have been added. These correspond to the identifying columns (primary-key columns in this case) for underlying mapping table `map_table_course_root_to_student_root`. Their values are the same as the values of fields `courseNumber` and `studentId`, respectively.
- Even though the document for student Luis F. (`studentId` = `9`) *failed import* into the `student` duality view (because field `age` has the *string* value `"Nineteen"`, and its 10% occurrence is a type-occurrence outlier), that document is nevertheless *present* in the duality view. When we import documents into the `course` and `teacher` duality views, a row is added to table `student_root` that has `9` as the value for column `student_root.student_id`, because `studentId` with value `9` is present in both input tables `course_tab` and `teacher_tab`.

  The `age` field value for that student document for Luis F. is (JSON) `null`, however (not `"Nineteen"` and not `19`). No `age` field exists in either of the course or teacher input document sets, so importing their student data for Luis F. into the `course` and `teacher` views stores SQL `NULL` in the `age` column in table `student_root`. And that `NULL` column value maps to JSON `null` in the student documents.

There are no other differences. In particular, mixed-type field `avgGrade` is unchanged from the input data, as it is not an outlier: each of its types occurs in more than 15% of the documents.

```
{
  "_id" : 1,
  "_metadata" :
  {
    "etag" : "4F39C8B86F4295AD2958B18A77B0AACC",
    "asof" : "00000000004DB839"
  },
  "age" : 20,
  "name" : "Donald P.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 1,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 1,
      "name" : "Data Structures",
      "avgGrade" : "TBD",
      "courseNumber" : "CS102"
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 1,
      "name" : "Algebra",
      "avgGrade" : 90,
      "courseNumber" : "MATH101"
    }
  ],
  "advisorId" : 102,
    "dormitory" :
    {
      "dormId" : 201,
      "dormName" : "ABC"
    },
    "studentId" : 1
  }

  {
    "_id" : 2,
    "_metadata" :
    {
      "etag" : "758A4F3E6EF3152A4FA0892AB38635D4",
      "asof" : "00000000004DB839"
    },
    "age" : 21,
    "name" : "Elena H.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS101",
        "ora$mapStudentId" : 2,
        "name" : "Algorithms",
        "avgGrade" : 75,
        "courseNumber" : "CS101"
      },
      {
        "ora$mapCourseId" : "CS102",
        "ora$mapStudentId" : 2,
        "name" : "Data Structures",
        "avgGrade" : "TBD",
        "courseNumber" : "CS102"
      },
      {
        "ora$mapCourseId" : "MATH102",
        "ora$mapStudentId" : 2,
        "name" : "Calculus",
        "avgGrade" : 95,
        "courseNumber" : "MATH102"
      }
    ],
    "advisorId" : 103,
    "dormitory" :
    {
      "dormId" : 202,
      "dormName" : "XYZ"
    },
    "studentId" : 2
  }

  {
    "_id" : 3,
    "_metadata" :
    {
      "etag" : "06905F120EF74124C5985354BBCE5CC1",
      "asof" : "00000000004DB839"
    },
    "age" : 20,
    "name" : "Francis K.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 3,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 103,
    "dormitory" :
    {
      "dormId" : 204,
      "dormName" : "QWE"
    },
    "studentId" : 3
  }

  {
    "_id" : 4,
    "_metadata" :
    {
      "etag" : "50847D1AB63537118A6133A4CC1B8708",
      "asof" : "00000000004DB839"
    },
    "age" : 19,
    "name" : "Georgia D.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS101",
        "ora$mapStudentId" : 4,
        "name" : "Algorithms",
        "avgGrade" : 75,
        "courseNumber" : "CS101"
      },
      {
        "ora$mapCourseId" : "MATH102",
        "ora$mapStudentId" : 4,
        "name" : "Calculus",
        "avgGrade" : 95,
        "courseNumber" : "MATH102"
      },
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 4,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 101,
    "dormitory" :
    {
      "dormId" : 203,
      "dormName" : "LMN"
    },
    "studentId" : 4
  }

  {
    "_id" : 5,
    "_metadata" :
    {
      "etag" : "FD6E27A868C56D1EF9C7AEB3F08C7F9B",
      "asof" : "00000000004DB839"
    },
    "age" : 21,
    "name" : "Hye E.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS102",
        "ora$mapStudentId" : 5,
        "name" : "Data Structures",
        "avgGrade" : "TBD",
        "courseNumber" : "CS102"
      },
      {
        "ora$mapCourseId" : "MATH101",
        "ora$mapStudentId" : 5,
        "name" : "Algebra",
        "avgGrade" : 90,
        "courseNumber" : "MATH101"
      }
    ],
    "advisorId" : 103,
    "dormitory" :
    {
      "dormId" : 201,
      "dormName" : "ABC"
    },
    "studentId" : 5
  }

  {
    "_id" : 6,
    "_metadata" :
    {
      "etag" : "2BDA7862330B0687F22F830F3E314E34",
      "asof" : "00000000004DB839"
    },
    "age" : 21,
    "name" : "Ileana D.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 6,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 102,
    "dormitory" :
    {
      "dormId" : 205,
      "dormName" : "GHI"
    },
    "studentId" : 6
  }

  {
    "_id" : 7,
    "_metadata" :
    {
      "etag" : "F1EF0CCD54EDFA78D2263D7E742D6CE8",
      "asof" : "00000000004DB839"
    },
    "age" : 20,
    "name" : "Jatin S.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS101",
        "ora$mapStudentId" : 7,
        "name" : "Algorithms",
        "avgGrade" : 75,
        "courseNumber" : "CS101"
      },
      {
        "ora$mapCourseId" : "CS102",
        "ora$mapStudentId" : 7,
        "name" : "Data Structures",
        "avgGrade" : "TBD",
        "courseNumber" : "CS102"
      }
    ],
    "advisorId" : 101,
    "dormitory" :
    {
      "dormId" : 204,
      "dormName" : "QWE"
    },
    "studentId" : 7
  }

  {
    "_id" : 8,
    "_metadata" :
    {
      "etag" : "9A25A267BC08858E0F754E0C00B32F9E",
      "asof" : "00000000004DB839"
    },
    "age" : 21,
    "name" : "Katie H.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "CS102",
        "ora$mapStudentId" : 8,
        "name" : "Data Structures",
        "avgGrade" : "TBD",
        "courseNumber" : "CS102"
      },
      {
        "ora$mapCourseId" : "MATH103",
        "ora$mapStudentId" : 8,
        "name" : "Advanced Algebra",
        "avgGrade" : 82,
        "courseNumber" : "MATH103"
      }
    ],
    "advisorId" : 102,
    "dormitory" :
    {
      "dormId" : 205,
      "dormName" : "GHI"
    },
    "studentId" : 8
  }

  {
    "_id" : 10,
    "_metadata" :
    {
      "etag" : "94376DA05B92E47718AF70A31FBE56E7",
      "asof" : "00000000004DB839"
    },
    "age" : 20,
    "name" : "Ming L.",
    "courses" :
    [
      {
        "ora$mapCourseId" : "MATH102",
        "ora$mapStudentId" : 10,
        "name" : "Calculus",
        "avgGrade" : 95,
        "courseNumber" : "MATH102"
      }
    ],
    "advisorId" : 101,
    "dormitory" :
    {
      "dormId" : 202,
      "dormName" : "XYZ"
    },
    "studentId" : 10
  }
```

```
{
  "_id" : 9,
  "_metadata" :
  {
    "etag" : "579824C71904C46901BBA605E8539943",
    "asof" : "00000000004DB839"
  },
  "age" : null,
  "name" : "Luis F.",
  "courses" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 9,
      "name" : "Algorithms",
      "avgGrade" : 75,
      "courseNumber" : "CS101"
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 9,
      "name" : "Calculus",
      "avgGrade" : 95,
      "courseNumber" : "MATH102"
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 9,
      "name" : "Advanced Algebra",
      "avgGrade" : 82,
      "courseNumber" : "MATH103"
    }
  ],
  "advisorId" : 101,
  "dormitory" :
  {
    "dormId" : 201,
    "dormName" : "ABC"
  },
  "studentId" : 9
}
```

Example 21-36   Course Duality View Document Collection (useFlexFields = false)

Compare this with the input course document set in the example "Course Document Set (Migrator Input)" in School Administration Example, Migrator Input Documents, which (with conversion using `minFieldFrequency` = `25` and `minTypeFrequency` = `15`) has only one outlier field: `Notes` (with an occurrence frequency of 20%).

These are the only differences (ignoring field order, which is irrelevant):

- Document identifier field `_id` and document-state field `_metadata` have been added. (Every document supported by a duality view has these fields.)
- Fields `ora$mapCourseId` and `ora$mapStudentId` have been added. These correspond to the identifying columns (primary-key columns in this case) for underlying mapping table

map_table_course_root_to_student_root. Their values are the same as the values of fields courseNumber and studentId, respectively.

- Even though the document with courseId = "MATH101") *failed import* into the course duality view (because field Notes occurs in only 20% of the documents and is thus an occurrence outlier), that document is *present* in the duality view, but without field Notes (it was not mapped to any column by the converter, and there is no flex column in which to store its value because useFlexFields was false) *and* without field creditHours.

  The problematic document is supported by the view, because when we import documents into the student and teacher duality views, a row is added to table course_root that has "MATH101" as the value for column course_root.course_id. This is because column course_id with value "MATH101" is present in both input tables student_tab and teacher_tab.

  Because the course document with field Notes failed to import, that input document's field creditHours is also missing from the document supported by the view. Field creditHours isn't provided for that document by importing any documents into the student or teacher view. Only table course_tab contains column credit_hours.

```
{
  "_id" : "CS101",
  "_metadata" :
  {
    "etag" : "7600B24570B58297702B95B8DE4F1B00",
    "asof" : "00000000004DB847"
  },
  "name" : "Algorithms",
  "teacher" :
  {
    "name" : "Betty Z.",
    "teacherId" : 102
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 1,
      "name" : "Donald P.",
      "studentId" : 1
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 2,
      "name" : "Elena H.",
      "studentId" : 2
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 4,
      "name" : "Georgia D.",
      "studentId" : 4
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 7,
      "name" : "Jatin S.",
      "studentId" : 7
    },
    {
      "ora$mapCourseId" : "CS101",
      "ora$mapStudentId" : 9,
      "name" : "Luis F.",
      "studentId" : 9
    }
  ],
  "creditHours" : 5,
  "courseId" : "CS101"
}

{
  "_id" : "CS102",
  "_metadata" :
  {
    "etag" : "C3813410219036CF0E210FFCE3917FEB",
    "asof" : "00000000004DB847"
  },
  "name" : "Data Structures",
  "teacher" :
  {
    "name" : "Betty Z.",
    "teacherId" : 102
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 1,
      "name" : "Donald P.",
      "studentId" : 1
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 2,
      "name" : "Elena H.",
      "studentId" : 2
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 5,
      "name" : "Hye E.",
      "studentId" : 5
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 7,
      "name" : "Jatin S.",
      "studentId" : 7
    },
    {
      "ora$mapCourseId" : "CS102",
      "ora$mapStudentId" : 8,
      "name" : "Katie H.",
      "studentId" : 8
    }
  ],
  "creditHours" : 3,
  "courseId" : "CS102"
}

{
  "_id" : "MATH101",
  "_metadata" :
  {
    "etag" : "5E24FBF3B13A297A89FE1D4C68C705BE",
    "asof" : "00000000004DB847"
  },
  "name" : "Algebra",
  "teacher" :
  {
    "name" : "Abdul J.",
    "teacherId" : 101
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 1,
      "name" : "Donald P.",
      "studentId" : 1
    },
    {
      "ora$mapCourseId" : "MATH101",
      "ora$mapStudentId" : 5,
      "name" : "Hye E.",
      "studentId" : 5
    }
  ],
  "courseId" : "MATH101"
}

{
  "_id" : "MATH102",
  "_metadata" :
  {
    "etag" : "4B55E2EF38E6DDAF6777251168DD07A5",
    "asof" : "00000000004DB847"
  },
  "name" : "Calculus",
  "teacher" :
  {
    "name" : "Abdul J.",
    "teacherId" : 101
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 2,
      "name" : "Elena H.",
      "studentId" : 2
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 4,
      "name" : "Georgia D.",
      "studentId" : 4
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 9,
      "name" : "Luis F.",
      "studentId" : 9
    },
    {
      "ora$mapCourseId" : "MATH102",
      "ora$mapStudentId" : 10,
      "name" : "Ming L.",
      "studentId" : 10
    }
  ],
  "creditHours" : 4,
  "courseId" : "MATH102"
}

{
  "_id" : "MATH103",
  "_metadata" :
  {
    "etag" : "C59E6274FE813279ECC28C73CA4AB121",
    "asof" : "00000000004DB847"
  },
  "name" : "Advanced Algebra",
  "teacher" :
  {
    "name" : "Colin J.",
    "teacherId" : 103
  },
  "students" :
  [
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 3,
      "name" : "Francis K.",
      "studentId" : 3
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 4,
      "name" : "Georgia D.",
      "studentId" : 4
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 6,
      "name" : "Ileana D.",
      "studentId" : 6
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 8,
      "name" : "Katie H.",
      "studentId" : 8
    },
    {
      "ora$mapCourseId" : "MATH103",
      "ora$mapStudentId" : 9,
      "name" : "Luis F.",
      "studentId" : 9
    }
  ],
  "creditHours" : "3",
  "courseId" : "MATH103"
}
```

Creating data-guide JSON schemas for the duality views is identical to doing so for the `useFlexFields` = `true` case. — see the example "Create JSON Data Guides for Document Collections Supported by Duality Views", "" in Import After Default Conversion. And the resulting data guides for the `student` and `teacher` views are the same as for that case — see "Student Duality View Data Guide" and "Teacher Duality View Data Guide"in Import After Default Conversion.

But the data guide created for the `course` view is not the same:

Example 21-37   Course Duality View Data Guide, for useFlexFields = false Case

This data guide JSON schema summarizes the collection of course documents supported by duality view `course`, for the conversion case where `useFlexFields` is `false`. It is identical to the data guide for the conversion case where `useFlexFields` is `true`, *except* that it is missing the `Notes` field.

```
{
  "type" : "object",
  "o:frequency" : 100,
  "o:last_analyzed" : "2025-01-15T21:19:03",
  "o:sample_size" : 5,
  "required" : true,
  "properties" :
  {
    "_id" :
    {
      "type" : "string",
      "o:length" : 8,
      "o:preferred_column_name" : "_id",
      "o:frequency" : 100,
      "o:low_value" : "CS101",
      "o:high_value" : "MATH103",
      "o:num_nulls" : 0,
      "o:last_analyzed" : "2025-01-15T21:19:03",
      "o:sample_size" : 5,
    "required" : true,
    "maxLength" : 7,
    "minLength" : 5
  },
  "name" :
  {
    "type" : "string",
    "o:length" : 16,
    "o:preferred_column_name" : "name",
    "o:frequency" : 100,
    "o:low_value" : "Advanced Algebra",
    "o:high_value" : "Data Structures",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2025-01-15T21:19:03",
    "o:sample_size" : 5,
    "required" : true,
    "maxLength" : 16,
    "minLength" : 7
  },
  "teacher" :
  {
    "type" : "object",
    "o:preferred_column_name" : "teacher",
    "o:frequency" : 100,
    "o:last_analyzed" : "2025-01-15T21:19:03",
    "o:sample_size" : 5,
    "required" : true,
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "o:length" : 8,
        "o:preferred_column_name" : "name",
        "o:frequency" : 100,
        "o:low_value" : "Abdul J.",
        "o:high_value" : "Colin J.",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-15T21:19:03",
        "o:sample_size" : 5,
        "required" : true,
        "maxLength" : 8,
        "minLength" : 8
      },
      "teacherId" :
      {
        "type" : "number",
        "o:preferred_column_name" : "teacherId",
        "o:frequency" : 100,
        "o:low_value" : 101,
        "o:high_value" : 103,
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-15T21:19:03",
        "o:sample_size" : 5,
        "required" : true,
        "maximum" : 103,
        "minimum" : 101
      }
    }
  },
  "courseId" :
  {
    "type" : "string",
    "o:length" : 8,
    "o:preferred_column_name" : "courseId",
    "o:frequency" : 100,
    "o:low_value" : "CS101",
    "o:high_value" : "MATH103",
    "o:num_nulls" : 0,
    "o:last_analyzed" : "2025-01-15T21:19:03",
    "o:sample_size" : 5,
    "required" : true,
    "maxLength" : 7,
    "minLength" : 5
  },
  "students" :
  {
    "type" : "array",
    "o:preferred_column_name" : "students",
    "o:frequency" : 100,
    "o:last_analyzed" : "2025-01-15T21:19:03",
    "o:sample_size" : 5,
    "required" : true,
    "items" :
    {
      "properties" :
      {
        "name" :
        {
          "type" : "string",
          "o:length" : 16,
          "o:preferred_column_name" : "name",
          "o:frequency" : 100,
          "o:low_value" : "Donald P.",
          "o:high_value" : "Ming L.",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-15T21:19:03",
          "o:sample_size" : 5,
          "required" : true,
          "maxLength" : 10,
          "minLength" : 6
        },
        "studentId" :
        {
          "type" : "number",
          "o:preferred_column_name" : "studentId",
          "o:frequency" : 100,
          "o:low_value" : 1,
          "o:high_value" : 10,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-15T21:19:03",
          "o:sample_size" : 5,
          "required" : true,
          "maximum" : 10,
          "minimum" : 1
        },
        "ora$mapCourseId" :
        {
          "type" : "string",
          "o:length" : 8,
          "o:preferred_column_name" : "ora$mapCourseId",
          "o:frequency" : 100,
          "o:low_value" : "CS101",
          "o:high_value" : "MATH103",
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-15T21:19:03",
          "o:sample_size" : 5,
          "required" : true,
          "maxLength" : 7,
          "minLength" : 5
        },
        "ora$mapStudentId" :
        {
          "type" : "number",
          "o:preferred_column_name" : "ora$mapStudentId",
          "o:frequency" : 100,
          "o:low_value" : 1,
          "o:high_value" : 10,
          "o:num_nulls" : 0,
          "o:last_analyzed" : "2025-01-15T21:19:03",
          "o:sample_size" : 5,
          "required" : true,
          "maximum" : 10,
          "minimum" : 1
        }
      }
    }
  },
  "_metadata" :
  {
    "type" : "object",
    "o:preferred_column_name" : "_metadata",
    "o:frequency" : 100,
    "o:last_analyzed" : "2025-01-15T21:19:03",
    "o:sample_size" : 5,
    "required" : true,
    "properties" :
    {
      "asof" :
      {
        "type" : "binary",
        "o:length" : 8,
        "o:preferred_column_name" : "asof",
        "o:frequency" : 100,
        "o:low_value" : "",
        "o:high_value" : "",
        "o:num_nulls" : 0,
        "o:last_analyzed" : "2025-01-15T21:19:03",
            "o:sample_size" : 5,
            "required" : true
          },
          "etag" :
          {
            "type" : "binary",
            "o:length" : 16,
            "o:preferred_column_name" : "etag",
            "o:frequency" : 100,
            "o:low_value" : "",
            "o:high_value" : "",
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2025-01-15T21:19:03",
            "o:sample_size" : 5,
            "required" : true
          }
        }
      },
      "creditHours" :
      {
        "oneOf" :
        [
          {
            "type" : "number",
            "o:preferred_column_name" : "creditHours",
            "o:frequency" : 60,
            "o:low_value" : 3,
            "o:high_value" : 5,
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2025-01-15T21:19:03",
            "o:sample_size" : 5,
            "maximum" : 5,
            "minimum" : 3
          },
          {
            "type" : "string",
            "o:length" : 1,
            "o:preferred_column_name" : "creditHours",
            "o:frequency" : 20,
            "o:low_value" : "3",
            "o:high_value" : "3",
            "o:num_nulls" : 0,
            "o:last_analyzed" : "2025-01-15T21:19:03",
            "o:sample_size" : 5,
            "maxLength" : 1,
            "minLength" : 1
          }
        ]
      }
    }
  }
```

This is the missing `Notes` entry (from the `useFlexFields` = `true` case):

```
"Notes" : {"type"                     : "string",
           "o:length"                 : 64,
           "o:preferred_column_name" : "Notes",
           "o:frequency"              : 20,
           "o:low_value"              : "Prerequisite for Advanced Algebra",
           "o:high_value"             : "Prerequisite for Advanced Algebra",
           "o:num_nulls"              : 0,
           "o:last_analyzed"          : "2025-01-15T21:11:48",
           "o:sample_size"            : 5,
           "maxLength"                : 33,
           "minLength"               : 33}
```

See Also:

- DBMS_JSON_DUALITY IN *Oracle Database PL/SQL Packages and Types Reference* for information about function `validate_import_report`
- JSON Patch and JSON Pointer for information about the error content reported by `DBMS_JSON_DUALITY.validate_import_report`

21.14 Errors That Migrator Configuration Alone Can't Fix

Even if you configure the migrator to not consider any fields or their values to be outliers, the migrator can detect other kinds of problems. A simple example shows detection of data contradiction between different document sets.

Trying to migrate document sets can sometimes uncover other potential data problems, besides the existence of outlier fields, which means that the migrator can raise errors even when converting with zero values for configuration fields `minFieldFrequency` and `minTypeFrequency`. This can be another reason you might want to *begin* a migration to duality views by using the simplified migration recipe: default behavior except no outliers.

An example of this is shown here: two document sets that apparently contradict each other, making it impossible to reconcile them without some data changes. Each document set is coherent on its own, both structurally and in terms of field types, but the two sets don't fit together.

(The migrator can help you discover some data coherency problems such as this, even if you're *not* migrating any data!)

The data used here is the same as that presented in other migrator topics, except for this difference: a *course* document says that `Natalie C.` teaches course `MATH101` and a *teacher* document says that `Abdul J.` teaches it. The data used in other topics has `Abdul J.` as the teacher in the course document as well; it has `Natalie C.` teaching no courses.

Example 21-38   Course MATH101 Document with Teacher Natalie C.

This is the input course document that has `Natalie C.` as the teacher of `MATH101`. All other course documents are as in the example "Course Document Set (Migrator Input)" in School

Administration Example, Migrator Input Documents, and the teacher and student documents are all as before.

```
{"courseId"          : "MATH101",
 "name"              : "Algebra",
 "creditHours"       : 3,
 "students"          : [ {"studentId" : 1, "name" : "Donald P."},
                         {"studentId" : 5, "name" : "Hye E."} ],
 "teacher"           : {"teacherId" : 104, "name" : "Natalie C."},
 "Notes"             : "Prerequisite for Advanced Algebra"}
...
```

- The *teacher* document for `Natalie C.` shows *no* `coursesTaught`, but the *course* document for `MATH101` shows `Natalie C.` as the `teacher`.
- The *teacher* document for `Abdul J.` shows that Abdul teaches both `MATH101` and `MATH102`, but the *course* document for `MATH101` shows `Natalie C.`, not Abdul, as the `teacher`.

The importer succeeds, as before (with zero values for `minFieldFrequency` and `minTypeFrequency`). But `DBMS_JSON_DUALITY.validate_import_report` finds and reports those contradictions.

Example 21-39 VALIDATE_IMPORT_REPORT for Contradictory Document Sets

See Migrating To Duality, Simplified Recipe for the simplified recipe that we assume is followed here as well. All of the steps are the same; the only difference is that the course document for `MATH101` used here is as shown in the preceding example, "Course MATH101 Document with Teacher Natalie C".

It's the import validation report for the teacher document set that reports the error; the documents for teachers Abdul J. and Natalie C. are reported.

```
SELECT * FROM DBMS_JSON_DUALITY.validate_import_report(
                                  table_name => 'TEACHER_TAB',
                                  view_name => 'TEACHER');


DATA
----
{"_id":101,"name":"Abdul J.","phoneNumber":["222-555-011","222-555-012"],"salary
":200000,"department":"Mathematics","coursesTaught":[{"courseId":"MATH101","name
":"Algebra","classType":"Online"},{"courseId":"MATH102","name":"Calculus","class
Type":"In-person"}],"studentsAdvised":[{"studentId":4,"name":"Georgia D.","dormI
d":203},{"studentId":7,"name":"Jatin S.","dormId":204},{"studentId":9,"name":"Lu
is F.","dormId":201},{"studentId":10,"name":"Ming L.","dormId":202}]}

{"_id":104,"name":"Natalie C.","phoneNumber":"222-555-044","salary":180000,"depa
rtment":"Computer Science","coursesTaught":[],"studentsAdvised":[]}

ERRORS
------
[{"op":"remove","path":"/coursesTaught/0"}]
[{"op":"replace","path":"/coursesTaught","value":[{"name":"Algebra","courseId":"
MATH101","classType":"Online"}]}]
```

Compare this example with the example "VALIDATE_IMPORT_REPORT for No Outlier Use Case" in Migrating To Duality, Simplified Recipe.

The import validation report suggests fixing the problem by removing `MATH101` from Abdul's teacher document and adding it to Natalie's.

- In the first error-log row returned, column `data` has Abdul's teacher document; in the second row, column `data` has Natalie's document.
- Column `errors` in the first row (which corresponds to Adbul's document) says to *remove* the *first* element of *array* `coursesTaught` (array indexing is zero-based, so the path is `/coursesTaught/0)`. That's this object, which describes course `MATH101`: `{"courseId":"MATH101","name":"Algebra","classType":"Online"}`.

  Column `errors` in the second row (which corresponds to Natalie's document) says to *replace* the *empty* array that's the value of field `coursesTaught` with this array: `[{"name":"Algebra","courseId":"MATH101","classType":"Online"}]`.

The validation report suggests that one remedy. But an alternative fix would be to change the course document for `MATH101` to fit the teacher documents: change its teacher to Abdul. Only *you* know, for your application, whether some particular data is an anomaly, according to your use of it, and only *you* know which ways to reconcile misfits are more appropriate.

If in fact that seemingly conflicted data is *correct*, then presumably the teacher and course documents *should not share* their teacher-course assignments. In that case, the remedy would be to use separate underlying tables in the teacher and course duality-view definitions.