2

Introduction To Car-Racing Duality Views Example

Data for Formula 1 car races is used here to present the features of JSON-relational duality views. This use-case example starts from an analysis of the kinds of JSON documents needed. It then defines corresponding entities and their relationships, relational tables, and duality views built on those tables.

> **Note:**
>
> An alternative approach to creating duality views is available to *migrate* an application that has *existing* sets of related documents, so that it uses duality views.
>
> For that you can use the *JSON-to-duality migrator*, which *automatically infers and generates* the appropriate duality views. No need to manually analyze the different kinds of documents to discover implicit entities and relationships, and then define and populate the relevant duality views and their underlying normalized tables.
>
> The migrator does all of that for you. By default, whatever document parts can be shared within or across views are shared, and the views are defined for maximum updatability.
>
> See Migrating From JSON To Duality in *Oracle Database Utilities*.

For the car-racing example we suppose a document-centric application that uses three kinds of JSON documents: driver, race, and team. Each of these kinds *shares* some data with another kind. For example:

- A *driver document* includes, in its information about a driver, identification of the driver's *team* and information about the *races* the driver has participated in.
- A *race document* includes, in its information about a particular race, information about the podium standings (first-, second-, and third-place winners), and the results for each *driver* in the race. Both of these include driver and *team* names. The racing data is for a single season of racing.
- A *team document* includes, in its information about a team, information about the *drivers* on the team.

Operations the application might perform on this data include the following:

- Adding or removing a driver, race, or team to/from the database
- Updating the information for a driver, race, or team
- Adding a driver to a team, removing a driver from a team, or moving a driver from one team to another
- Adding race results to the driver and race information

The intention in this example is that *all common information be shared*, so that, say, the driver with identification number 302 in the driver duality view is the same as driver number 302 in the team view.

You *specify the sharing* of data that's common between two duality views by including the *same data* from underlying tables.

When you define a given duality view you can control whether it's possible to insert into, delete from, or update the *documents* supported by the view and, overriding those constraints, whether it's possible to insert, delete, or update a given *field* in a supported document. By default, a duality view is read-only: no inserting, deleting, or updating documents.

> **See Also:**
>
> - Working with JSON Relational Duality Views using SQL, a SQL script that mirrors the examples in this document
> - Formula One (Wikipedia)

2.1 Car-Racing Example, JSON Documents

The car-racing example has three kinds of documents: a team document, a driver document, and a race document.

A document supported by a duality view always includes, at its top (root) level, a **document-identifier** field, **`_id`**, which corresponds to (all of) the identifying columns of the root table that underlies the view. See Document-Identifier Field for Duality Views. (In the car-racing example the root table of each duality view has a single identifying column, which is a primary-key column.)

The following naming convention is followed in this documentation:

- The document-identifier field (`_id`) of each kind of document (team, driver, or race) corresponds to the root-table identifying columns of the duality view that supports those documents. For example, field `_id` of a team document corresponds to identifying (primary-key) column `team_id` of table `team`, which is the root table underlying duality view `team_dv`.

- Documents of one kind (e.g. team), supported by one duality view (e.g. `team_dv`) can include other fields named `...Id` (e.g. `driverId`), which represent *foreign-key* references to identifying columns in tables underlying *other* duality views — columns that contain data that's shared. For example, in a team document, field `driverId` represents a foreign key that refers to the document-identifier field (`_id`) of a driver document.

> **Note:**
>
> Only the *application-logic* document content, or **payload** of each document, is shown here. That is, the documents shown here do not include the automatically generated and maintained, top-level field `_metadata` (whose value is an object with fields `etag` and `asof`). However, this *document-handling* field is always included in documents supported by a duality view. See Creating Duality Views for information about field `_metadata`.

Example 2-1 A Team Document

A team document includes information about the drivers on the team, in addition to information that's relevant to the team but not necessarily relevant to its drivers.

- Top-level field `_id` uniquely identifies a team document. It is the document-identifier field. Column `team_id` of table `team` corresponds to this field; it is the table's *primary key*.
- The team information that's *not shared* with driver documents is in field `_id` and top-level fields `name` and `points`.
- The team information that's *shared* with driver documents is in fields `driverId`, `name`, and `points`, under field `driver`. The value of field `driverId` is that of the document-identifier field (`_id`) of a driver document.

```
{"_id"    : 302,
 "name"   : "Ferrari",
 "points" : 300,
 "driver" : [ {"driverId" : 103,
               "name"     : "Charles Leclerc",
               "points"   : 192},
              {"driverId" : 104,
               "name"     : "Carlos Sainz Jr",
               "points"   : 118} ]}
```

Example 2-2 A Driver Document

A driver document includes identification of the driver's team and information about the races the driver has participated in, in addition to information that's relevant to the driver but not necessarily relevant to its team or races.

- Top-level field `_id` uniquely identifies a driver document. It is the document-identifier field. Column `driver_id` of the `driver` table corresponds to this field; it is that table's *primary key*.
- The driver information that's *not shared* with race or team documents is in fields `_id`, `name`, and `points`.
- The driver information that's *shared* with *race* documents is in field `race`. The value of field `raceId` is that of the document-identifier field (`_id`) of a race document.

- The driver information that's *shared* with a *team* document is in fields such as `teamId`, whose value is that of the document-identifier field (`_id`) of a team document.

Two alternative versions of a driver document are shown, with and without nested team and race information.

Driver document, with nested team and race information:

Field `teamInfo` contains the nested team information (fields `teamId` and `name`). Field `raceInfo` contains the nested race information (fields `raceId` and `name`).

```
{"_id"      : 101,
 "name"     : "Max Verstappen",
 "points"   : 258,
 "teamInfo" : {"teamId" : 301, "name" : "Red Bull"},
 "race"     : [ {"driverRaceMapId" : 3,
                 "raceInfo"        : {"raceId" : 201,
                                      "name"   : "Bahrain Grand Prix"},
                 "finalPosition"   : 19},
                {"driverRaceMapId" : 11,
                 "raceInfo"        : {"raceId" : 202,
                                      "name"   : "Saudi Arabian Grand Prix"},
                 "finalPosition"   : 1} ]}
```

Driver document, without nested team and race information:

Fields `teamId` and `team` are not nested in a `teamInfo` object. Fields `raceId` and `name` are not nested in a `raceInfo` object.

```
{"_id"      : 101,
 "name"     : "Max Verstappen",
 "points"   : 25,
 "teamId"   : 301,
 "team"     : "Red Bull",
 "race"     : [ {"driverRaceMapId" : 3,
                 "raceId"          : 201,
                 "name"            : "Bahrain Grand Prix",
                 "finalPosition"   : 19},
                {"driverRaceMapId" : 11,
                 "raceId"          : 202,
                 "name"            : "Saudi Arabian Grand Prix",
                 "finalPosition"   : 1} ]}
```

Example 2-3 A Car-Race Document

A race document includes, in its information about a particular race, information about the podium standings (first, second, and third place), and the results for each driver in the race. The podium standings include the driver and team names. The result for each driver includes the driver's name.

Both of these include driver and team names.

- Top-level field `_id` uniquely identifies a race document. It is the document-identifier field. Column `race_id` of the `race` table corresponds to this field; it is that table's *primary key*.
- The race information that's *not shared* with driver or team documents is in fields `_id`, `name` (top-level), `laps`, `date`, `time`, and `position`.

- The race information that's *shared* with driver documents is in fields such as `driverId`, whose value is that of the document-identifier field (`_id`) of a driver document.
- The race information that's *shared* with team documents is in field `team` (under `winner`, `firstRunnerUp`, and `secondRunnerUp`, which are under `podium`).

Two alternative versions of a race document are shown, with and without nested driver information.

Race document, with nested driver information:

```
{"_id"    : 201,
 "name"   : "Bahrain Grand Prix",
 "laps"   : 57,
 "date"   : "2022-03-20T00:00:00",
 "podium" : {"winner"         : {"name" : "Charles Leclerc",
                                 "team" : "Ferrari",
                                 "time" : "02:00:05.3476"},
             "firstRunnerUp"  : {"name" : "Carlos Sainz Jr",
                                 "team" : "Ferrari",
                                 "time" : "02:00:15.1356"},
             "secondRunnerUp" : {"name" : "Max Verstappen",
                                 "team" : "Red Bull",
                                 "time" : "02:01:01.9253"}},
 "result" : [ {"driverRaceMapId" : 3,
               "position"        : 1,
               "driverInfo"      : {"driverId" : 103,
                                    "name"     : "Charles Leclerc"},
              {"driverRaceMapId" : 4,
               "position"        : 2,
               "driverInfo"      : {"driverId" : 104,
                                    "name"     : "Carlos Sainz Jr"},
              {"driverRaceMapId" : 9,
               "position"        : 3,
               "driverInfo"      : {"driverId" : 101,
                                    "name"     : "Max Verstappen"},
              {"driverRaceMapId" : 10,
               "position"        : 4,
               "driverInfo"      : {"driverId" : 102,
                                    "name"     : "Sergio Perez"} ]}
```

Race document, without nested driver information:

```
{"_id"    : 201,
 "name"   : "Bahrain Grand Prix",
 "laps"   : 57,
 "date"   : "2022-03-20T00:00:00",
 "podium" : {"winner"         : {"name" : "Charles Leclerc",
                                 "team" : "Ferrari",
                                 "time" : "02:00:05.3476"},
             "firstRunnerUp"  : {"name" : "Carlos Sainz Jr",
                                 "team" : "Ferrari",
                                 "time" : "02:00:15.1356"},
             "secondRunnerUp" : {"name" : "Max Verstappen",
                                 "team" : "Red Bull",
                                 "time" : "02:01:01.9253"}},
```

```
"result" : [ {"driverRaceMapId" : 3,
              "position"        : 1,
              "driverId"        : 103,
              "name"            : "Charles Leclerc"},
             {"driverRaceMapId" : 4,
              "position"        : 2,
              "driverId"        : 104,
              "name"            : "Carlos Sainz Jr"},
             {"driverRaceMapId" : 9,
              "position"        : 3,
              "driverId"        : 101,
              "name"            : "Max Verstappen"},
             {"driverRaceMapId" : 10,
              "position"        : 4,
              "driverId"        : 102,
              "name"            : "Sergio Perez"} ]}
```

Related Topics

- Document-Identifier Field for Duality Views
  A document supported by a duality view always includes, at its top level, a **document-identifier** field, `_id`, which corresponds to the *identifying columns* (primary-key columns, identity columns, or columns with a unique constraint or unique index) of the *root* table underlying the view. The field value can take different forms.

2.2 Car-Racing Example, Entity Relationships

Driver, car-race, and team entities are presented, together with the relationships among them. You define entities that correspond to your application documents in order to help you determine the tables needed to define the duality views for your application.

From the documents to be used by your application you can establish entities and their relationships. *Each entity corresponds to a document type*: driver, race, team.

Unlike the corresponding documents, the entities we use have *no content overlap* — they're **normalized**. The content of an entity (what it represents) is *only that which is specific* to its corresponding document type; it doesn't include anything that's also part of another document type.

- The *driver* entity represents only the content of a driver document that's not in a race or team document. It includes only the driver's name and points, corresponding to document fields `name` and `points`.
- The *race* entity represents only the content of a race document that's not in a driver document or a team document. It includes only the race's name, number of laps, date, and podium information, corresponding to document fields `name`, `laps`, `date`, and `podium`.
- The *team* entity represents only the content of a team document that's not in a document or race document. It includes only the team's name and points, corresponding to document fields `name` and `points`.

Two entities are related according to their cardinality. There are three types of such relationships:[1]

[1] In the notation used here, **N** does not represent a number; it's simply an abbreviation for "many", or more precisely, "*one or more*".

One-to-one (1:1)
An instance of entity *A* can only be associated with *one* instance of entity **B**. For example, a driver can only be on one team.

One-to-many (1:N)
An instance of entity *A* can be associated with *one or more* instances of entity *B*. For example, a team can have many drivers.

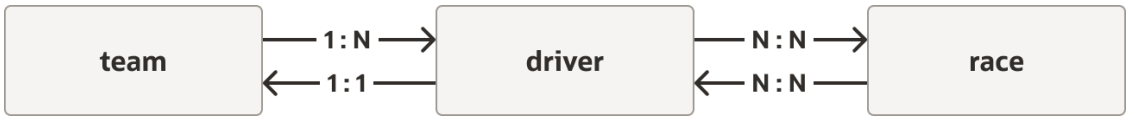
Many-to-many (N:N)
An instance of entity *A* can be associated with *one or more* instances of entity *B*, and *conversely*. For example, a race can have many drivers, and a driver can participate in many races.

> **See Also:**
>
> Entity-relationship model

A many-to-one (N:1) relationship is just a one-to-many relationship looked at from the opposite viewpoint. We use only one-to-many.

See Figure 2-1. An arrow indicates the relationship direction, with the arrowhead pointing to the second cardinality. For example, the 1:N arrow from entity *team* to entity *driver* points toward *driver*, to show that one team relates to many drivers.

Figure 2-1 Car-Racing Example, Directed Entity-Relationship Diagram (1)



A driver can only be associated with one team (1:1). A team can be associated with multiple drivers (1:N). A driver can be associated with multiple races (N:N). A race can be associated with multiple drivers (N:N).

Related Topics

- Creating Duality Views
  You use SQL with (1) SQL/JSON generation-function queries or (2) GraphQL queries to create JSON-relational duality views. Example team, driver, and race duality views are created to provide the JSON documents used by a car-racing application.
- Car-Racing Example, Tables
  Normalized entities are modeled as database tables. Entity relationships are modeled as joins between participating tables. Tables `team`, `driver`, and `race` are used to implement the duality views that provide and support the team, driver, and race JSON documents used by the car-racing application.

> **See Also:**
>
> Database normalization (Wikipedia)

2.3 Car-Racing Example, Tables

Normalized entities are modeled as database tables. Entity relationships are modeled as joins between participating tables. Tables `team`, `driver`, and `race` are used to implement the duality views that provide and support the team, driver, and race JSON documents used by the car-racing application.

The normalized *entities* have no content overlap. But we need the database *tables* that implement the entities to overlap logically, in the sense of a table referring to some content that is stored in another table. To realize this we add columns that are linked to other tables using *foreign-key constraints*. It is these foreign-key relations among tables that implement their sharing of common content.

The tables used to define a duality view must satisfy these requirements (otherwise an error is raised when you try to create the view):

- For a table underlying a duality view to be *updatable* indirectly, through the view (that is, by updating documents supported by the view), the individual rows of the table must be identifiable.

  For this requirement, you define one or more columns, called **identifying columns** for the table, which together *identify a row*. Identifying columns are *primary-key* columns, *identity* columns, or columns with *unique constraints* or *unique indexes*.

  Columns with unique constraints and columns with unique indexes are sometimes called **unique-key** columns. A unique key or a primary key is thus a set of one or more columns that uniquely identify a row in a table.

  For the *root* table of a duality view, if one or more unique-key columns are used for this purpose, then at least one of them must, in addition, be marked `NOT NULL`. This prevents any ambiguity that could arise from using a `NULL`able unique key or a unique key that has some `NULL` columns.

  The identifying columns for the root table in a duality view correspond to the *document-identifier* field, `_id`, of the JSON documents that the view is designed to support — see Document-Identifier Field for Duality Views.

- Oracle recommends that you also define an index on each foreign-key column. References (links) between primary and foreign keys must be defined, but they need not be enforced.

> **Note:**
>
> Primary and unique indexes are generally created implicitly when you define primary-key and unique-key integrity constraints. But this is not guaranteed, and indexes can be dropped after their creation. It's up to you to ensure that the necessary indexes are present. See Creating Indexes in *Oracle Database Administrator's Guide*.

In general, a value in a foreign-key column can be `NULL`. Besides the above requirements, if you want a foreign-key column to not be `NULL`able, then mark it as `NOT NULL` in the table definition.

There's only one identifying column for each of the tables used in the car-racing example, and it is a primary-key column. In this documentation we sometimes speak of primary, foreign, and unique keys as single-column keys, but keep in mind that they can in general be **composite**: composed of multiple columns.

In the car-racing example, entities team, driver, and race are implemented by tables `team`, `driver`, and `race`, which have the following columns:

- **`team`** table:
  - **`team_id`** — primary key
  - **`name`** — unique key
  - `points`
- **`driver`** table:
  - **`driver_id`** — primary key
  - **`name`** — unique key
  - `points`
  - **`team_id`** — foreign key that links to column `team_id` of table `team`
- **`race`** table:
  - **`race_id`** — primary key
  - **`name`** — unique key (so the table has no duplicate rows: there can't be two races with the same name)
  - `laps`
  - `race_date`
  - `podium`

The logic of the car-racing application mandates that there be only one team with a given team name, only one driver with a given driver name, and only one race with a given race name, so column `name` of each of these tables is made a *unique key*. (This in turn means that there is only one team document with a given `name` field value, only one driver document with a given `name`, and only one race document with a given `name`.)

Table `driver` has an additional column, `team_id`, which is data that's logically shared with table `team` (it corresponds to document-identifier field `_id` of the team document). This sharing is defined by declaring the column to be a **foreign key** in table `driver`, which links to (primary-key) column `team_id` of table `team`. That link implements both the 1:1 relationship from driver to team and the 1:N relationship from team to driver.

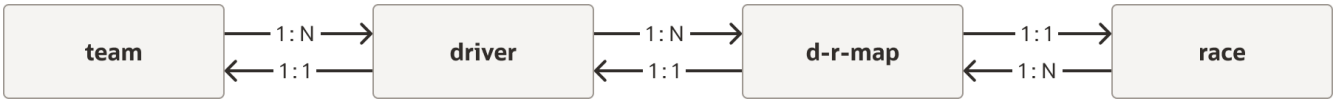
But what about the other sharing: the race information in a driver document that's shared with a race document, and the information in a race document that's shared with a driver document or with a team document?

That information sharing corresponds to the many-to-many (N:N) relationships between entities driver and race. The database doesn't implement N:N relationships directly. Instead, we need to add another table, called a **mapping table** (or an **associative table**), to bridge the relationship between tables `driver` and `race`. A mapping table includes, as foreign keys, the primary-key columns of the two tables that it associates.

An N:N entity relationship is equivalent to a 1:N relationship followed by a 1:1 relationship. We use this equivalence to implement an N:N entity relationship using database tables, by adding mapping table **`driver_race_map`** between tables `driver` and `race`.

Figure 2-2 is equivalent to Figure 2-1. Intermediate entity *d-r-map* is added to expand each N:N relationship to a 1:N relationship followed by a 1:1 relationship.[2]

Figure 2-2 Car-Racing Example, Directed Entity-Relationship Diagram (2)



Mapping table **driver_race_map** implements intermediate entity *d-r-map*. It has the following columns:

- **driver_race_map_id** — primary key
- **race_id** — (1) foreign key that links to primary-key column `race_id` of table `race` and (2) unique key (so the table has no duplicate rows: there can't be two entries for the same driver for a particular race)
- **driver_id** — foreign key that links to primary-key column `driver_id` of table `driver`
- **position**

Together with the relations defined by their foreign-key and primary-key links, the car-racing tables form a *dependency graph*. This is shown in Figure 3-1.

Example 2-4 Creating the Car-Racing Tables

This example creates each table with a primary-key column, whose values are automatically generated as a sequence of integers, and a unique-key column, `name`. This implicitly also creates unique indexes on the primary-key columns. The example also creates foreign-key indexes.

Column `podium` of table `race` has data type `JSON`. Its content is flexible: it need not conform to any particular structure or field types. Alternatively, its content could be made to conform to (that is, validate against) a particular JSON schema.

```
CREATE TABLE team
  (team_id    INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY,
   name       VARCHAR2(255) NOT NULL UNIQUE,
   points     INTEGER NOT NULL,
   CONSTRAINT team_pk PRIMARY KEY(team_id));

CREATE TABLE driver
  (driver_id  INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY,
   name       VARCHAR2(255) NOT NULL UNIQUE,
   points     INTEGER NOT NULL,
   team_id    INTEGER,
   CONSTRAINT driver_pk PRIMARY KEY(driver_id),
   CONSTRAINT driver_fk FOREIGN KEY(team_id) REFERENCES team(team_id));

CREATE TABLE race
  (race_id    INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY,
   name       VARCHAR2(255) NOT NULL UNIQUE,
   laps       INTEGER NOT NULL,
   race_date  DATE,
   podium     JSON,
   CONSTRAINT race_pk PRIMARY KEY(race_id));
```

[2] In the notation used here, **N** does not represent a number; it's simply an abbreviation for "many", or more precisely, "*one or more*".

```
-- Mapping table, to bridge the tables DRIVER and RACE.
--
CREATE TABLE driver_race_map
  (driver_race_map_id INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY,
   race_id            INTEGER NOT NULL,
   driver_id          INTEGER NOT NULL,
   position           INTEGER,
   CONSTRAINT driver_race_map_uk  UNIQUE (race_id, driver_id),
   CONSTRAINT driver_race_map_pk  PRIMARY KEY(driver_race_map_id),
   CONSTRAINT driver_race_map_fk1 FOREIGN KEY(race_id)
                                  REFERENCES race(race_id),
   CONSTRAINT driver_race_map_fk2 FOREIGN KEY(driver_id)
                                  REFERENCES driver(driver_id));
-- Create foreign-key indexes
--
CREATE INDEX driver_fk_idx ON driver (team_id);
CREATE INDEX driver_race_map_fk1_idx ON driver_race_map (race_id);
CREATE INDEX driver_race_map_fk2_idx ON driver_race_map (driver_id);
```

Note:

Primary-key, unique-key, and foreign-key integrity constraints *must be defined* for the tables that underlie duality views (or else an error is raised), but they *need not be enforced*.

In some cases you might know that the conditions for a given constraint are satisfied, so you don't need to validate or enforce it. You might nevertheless want the constraint to be present, to improve query performance. In that case, you can put the constraint in the **`RELY`** state, which asserts that the constraint is believed to be satisfied. See RELY Constraints in a Data Warehouse in *Oracle Database Data Warehousing Guide*.

You can also make a foreign key constraint **`DEFERRABLE`**, which means that the validity check is done at the end of a transaction. See Deferrable Constraints in *Oracle Database Concepts*

Note:

The SQL data types allowed for a column in a table underlying a duality view are `BINARY_DOUBLE`, `BINARY_FLOAT`, `BLOB`, `BOOLEAN`, `CHAR`, `CLOB`, `DATE`, `JSON`, `INTERVAL DAY TO SECOND`, `INTERVAL YEAR TO MONTH`, `NCHAR`, `NCLOB`, `NUMBER`, `NVARCHAR2`, `VARCHAR2`, `RAW`, `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `VECTOR`. An error is raised if you specify any other column data type.

Related Topics

- Car-Racing Example, Entity Relationships
  Driver, car-race, and team entities are presented, together with the relationships among them. You define entities that correspond to your application documents in order to help you determine the tables needed to define the duality views for your application.

- Creating Duality Views
  You use SQL with (1) SQL/JSON generation-function queries or (2) GraphQL queries to create JSON-relational duality views. Example team, driver, and race duality views are created to provide the JSON documents used by a car-racing application.

> **See Also:**
>
> - JSON Schema in *Oracle Database JSON Developer's Guide*
> - CREATE TABLE in *Oracle Database SQL Language Reference*