Yes—{**dm**} is built precisely to *surface data-quality problems* in a collection of related tables, whether those tables live in memory or inside a SQL database that you reach through **DBI**/**dplyr**.

**1. Bring the tables into a dm object**

library(dm)

library(DBI)

library(dplyr)

con <- dbConnect(RMariaDB::MariaDB(), dbname = "my\_prod\_db")

# Pick the tables you need (or dm\_from\_con() to take everything)

my\_dm <- dm\_from\_con(

con,

tables = c("customers", "orders", "order\_items", "products")

)

dm keeps only *table headers* and the key metadata locally; the rows stay on the server until you ask for them.

**2. Declare (or infer) the constraints**

If your database already defines primary- and foreign-key constraints, dm\_from\_con() will import them automatically.  
If not, you can add them in R so that dm knows what to check:

my\_dm <- my\_dm |>

dm\_add\_pk(customers, customer\_id, check = TRUE) |> # check = TRUE verifies uniqueness

dm\_add\_pk(products, product\_id, check = TRUE) |>

dm\_add\_fk(orders, customer\_id, customers) |>

dm\_add\_fk(order\_items, order\_id, orders) |>

dm\_add\_fk(order\_items, product\_id, products)

(Helpers such as enum\_pk\_candidates() and dm\_enum\_fk\_candidates() will suggest plausible key columns if you’re unsure.)

**3. Find the problems**

| **Function** | **What it flags** | **Typical outcome** |
| --- | --- | --- |
| dm\_examine\_constraints() | Every PK/FK you defined, marked is\_key = TRUE (OK) or FALSE (violated) with a problem message | Gives a tidy summary you can filter or join back to the data ([krlmlr.github.io](https://krlmlr.github.io/dm/reference/dm_examine_constraints.html?utm_source=chatgpt.com)) |
| dm\_check\_constraints() | Same check, but *errors* if anything fails (good for tests/CI) | Stops the pipeline early ([krlmlr.github.io](https://krlmlr.github.io/dm/news/index.html?utm_source=chatgpt.com)) |
| check\_key() / check\_subset() / check\_set\_equality() | Low-level helpers to test uniqueness or referential integrity between any two tables/columns | Useful during exploratory work ([dm.cynkra.com](https://dm.cynkra.com/articles/tech-dm-low-level.html?utm_source=chatgpt.com)) |

Example:

bad <- my\_dm |>

dm\_examine\_constraints() |>

filter(!is\_key)

bad

# A tibble:

# table kind columns ref\_table is\_key problem

# <chr> <chr> <list> <chr> <lgl> <chr>

# 1 orders FK <order\_id> order\_items FALSE "Orphan orders"

# 2 order\_items FK <product\_id> products FALSE "Missing product"

**4. Drill down to the offending rows**

orphans <- dm\_get\_tables(my\_dm)$orders |>

anti\_join(dm\_get\_tables(my\_dm)$order\_items, by = "order\_id")

Repeat for any other violations the summary highlighted.

**5. Why use dm instead of ad-hoc SQL?**

* **Unified view** – you can keep remote and local tables in one object.
* **Reproducible checks** – your constraint declarations live in your script, so they can be run in tests or CI.
* **Graph-aware workflows** – once integrity issues are fixed you can use dm\_flatten\_to\_tbl(), dm\_draw(), etc., without worrying that joins silently drop rows.

**Take-home**

*dm\_examine\_constraints() (or its stricter sibling dm\_check\_constraints()) is the primary workhorse for spotting duplicates, missing foreign-key targets, and other integrity breaches.*  
So, yes—load your schema into a dm, add or import the key definitions, and let **dm** show you exactly where the relational “cracks” are.