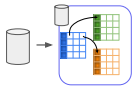
Relational data with dm: Cheat sheet

If you have more than one table, use dm.

Create dm…

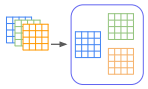
from database: `dm_from_con()`

```
con <- DBI::dbConnect(...)
dm_from_con(con)
```



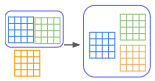
from data frames: `dm(df1, df2, ...)`

```
dm(df1, df2, df3)
```

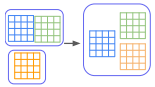


from dm: `dm(dm1, df1...)`

```
dm(dm1, df1)
```



```
dm(dm1, dm2)
```



Add keys: `dm_add_pk()`, `dm_add_fk()`

Automatic for MariaDB, SQL Server, Postgres, and others.

Primary keys

Identify potential primary keys: `dm_enum_pk_candidates()`: columns, candidate, why.

Add primary keys:

```
dm1 |>
dm_add_pk(table, columns)
```

Identify potential foreign keys: `dm_enum_fk_candidates()`: columns, candidate, why.

Add foreign keys:

```
dm1 |>
dm_add_fk(table, column)
```

dm objects: relational data models

The dm package provides a **grammar of relational data models**. It helps maintain **referential integrity**.

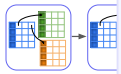
A dm behaves like a list of tables (data frames or lazy tables) capturing **relationships** between the tables.

Shiny app: `dm_gui(dm = dm1)`

Resize dm

Select tables: `dm_select_tbl(dm1, ...)`

```
dm1 |>
  dm_select_tbl(-df3)
```

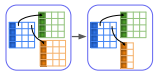


Rename tables: `dm_rename_tbl(dm1, ...)`

Select columns: `dm_select(dm1, table, ...)`

Automatic update of dm meta-information and table relations.

```
dm1 |>
  dm_select(df3, -c3, -c4)
```



Rename columns: `dm_rename(dm1, table, ...)`

Filter rows: `dm_filter(dm1, table = (pred))`

Filter rows in the table where the condition is defined, but also all directly/indirectly connected tables.

Visualize dm: `dm_draw()`

Control diagram level of detail: display…

Only keys (default): `dm_draw(view_type = "keys_only")`.

All variables: `dm_draw(view_type = "all")`.

Only table names: `dm_draw(view_type = "title_only")`.

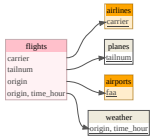
```
dm |>
dm_draw(
  view_type = "title_only",
  rankdir = "TB"
)
```

Control diagram scope

To visualize fewer tables first use `dm_select_tbl()`.

Control diagram colors: `dm_set_colors()`

```
dm |>
  dm_set_colors(
    pink = flights,
    orange = starts_with("air")
  ) |>
  dm_draw()
```



Data checks

`dm_examine_constraints()`

tibble with information about which key constraints are met or violated.

`dm_examine_cardinalities()`

tibble with information about the cardinality of the foreign keys constraints.

`check_key(df, col1)`

returns an error if not an unique key.

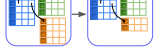
`check_subset(df1, df2)`

returns an error if `df1` is not a subset of `df2`.

`check_set_equality(df1, df2)`

returns an error if `df1` and `df2` are not the same sets.

```
dm1 |>
  dm_filter(df3 = (x == "val"))
```



Fix column names: `dm_disambiguate_cols()`

`dm_disambiguate_cols(dm1)` ensures that all columns in a dm have unique names.

Relational data with dm: Cheat sheet

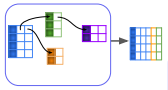
If you have more than one table, use dm.

Transform dm into tibble

Wide tibble: Cascade joins with `dm_flatten_to_tbl()`

Only direct neighbours: `dm_flatten_to_tbl()`

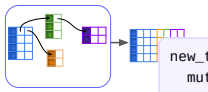
```
dm1 |>
  dm_flatten_to_tbl(
.start = df1
)
```



All neighbours:

`dm_flatten_to_tbl(.recursive = TRUE)`

```
dm1 |>
  dm_flatten_to_tbl(
.start = df1,
.recursive = TRUE
)
```

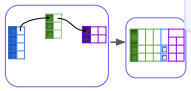


Single tibble dm: `dm_wrap_tbl()`

Parent tables are packed — `dm_pack_tbl()`.

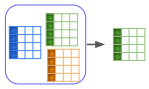
Child tables are nested — `dm_nest_tbl()`.

```
dm1 |>
  dm_wrap_tbl(
root = green_df
)
```



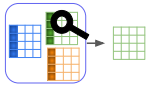
Retrieve one table of the dm: `pull_tbl()`

```
dm1 |>
  pull_tbl(
dm1,
green_df,
keyed = TRUE
)
```



If the dm is zoomed, retrieve zoomed table automatically.

```
dm1 |>
  dm_zoom_to(green_df) |>
pull_tbl()
```

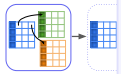


Mutate, create, analyze tables

Method 1: deconstruct and reconstruct

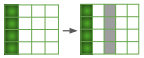
1. `dm_get_tables(keyed = TRUE)`:
   to keep information on primary and foreign keys).

```
dm_tbl <- dm1 |>
  dm_get_tables(keyed = TRUE)
```



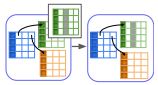
2. tidyverse pipeline on the table of interest.

```
new_table1 <- dm_tbl$table1 |>
  mutate(...)
```



3. Optional: update the dm object:

```
dm1 |>
  dm_select_tbl(-table1) |>
  dm(table1 = new_table1)
```



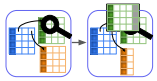
Method 2: zoom

1. `dm_zoom_to()`: Zoom on a table.

```
zoomed_dm1 <- dm1 |>
  dm_zoom_to(green_df)
```

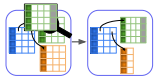
2. tidyverse pipeline (`mutate()`, etc.).

```
zoomed_dm2 <- zoomed_dm1 |>
  mutate(var = thing)
```



3. `dm_update_zoomed()` (replace) / `dm_insert_zoomed()`

```
dm3 <- zoomed_dm2 |>
  dm_update_zoomed()
```



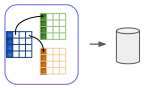
Modify database source of a dm

Export dm object to database: `copy_dm_to()`

Need a **database connection — `DBI::dbConnect()`**.

```
con <- DBI::dbConnect(...)
  # Persistent tables:
persistent_dm &lt;- copy_dm_to(
con,
dm1,
temporary = FALSE
)

DBI::dbDisconnect(con)
```



± Insert, update or remove rows in a dm

Methods:

`dm_rows_insert(dm1, dm2)`: adds new rows

`dm_rows_update(dm1, dm2)`: changes values in rows

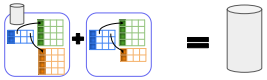
`dm_rows_patch(dm1, dm2)`: fills in missing values

`dm_rows_upsert(dm1, dm2)`: adds new or changes rows

`dm_rows_delete(dm1, dm2)`: deletes rows

```
dm1 |>
dm_rows_insert(dm2, in_place = FALSE)
```

```
dm1 |>
dm_rows_insert(dm2, in_place = TRUE)
```



A dm is immutable, except with

✓ these functions AND

✓ a mutable backend (database) AND

✓ `in_place = TRUE`.