Data grain

Grain is the combination of columns at which records in a table are unique. Ideally, this is captured in a single column, a unique [primary key](https://docs.getdbt.com/terms/primary-key), but even then, there is descriptive grain behind that unique id. Let’s look at some examples to better understand this concept.

| **user\_id** | **address** |
| --- | --- |
| 1 | 123 Jaffle Ln |
| 2 | 456 Waffle St |
| 3 | 789 Raffle Rd |

In the above table, each user\_id is unique. This table is at the *user* *grain*.

| **user\_id** | **address** |
| --- | --- |
| 1 | 123 Jaffle Ln |
| 1 | 420 Jaffle Ln |
| 2 | 456 Waffle St |
| 3 | 789 Raffle Rd |

***In the above table, user\_id is no longer unique. The combination of user\_id and address creates a unique combination***, thus this table is at the *user* *address* *grain*. We generally describe the grain conceptually based on the names of the columns that make it unique. A more realistic combination you might see in the wild would be a table that capture the state of all users at midnight every day. The combination of the captured updated\_date and user\_id would be unique, meaning our table is at *user per day* grain.

In both examples listed in the previous paragraph, we’d want to create a [surrogate key](https://docs.getdbt.com/terms/surrogate-key) of some sort from the combination of columns that comprise the grain. This gives our table a primary key, which is crucial for testing and optimization, and always a best practice. Typically, we’ll name this primary key based on the verbal description of the grain. For the latter example, we’d have user\_per\_day\_id. This will be more solid and efficient than testing than repeatedly relying on the combination of those two columns.

Thinking deeply about grain is a crucial part of data modeling. As we design models, we need to consider the entities we’re describing, and what dimensions (time, attributes, etc.) might fan those entities out so they’re no longer unique, as well as how we want to deal with those. Do we need to apply transformations to reduplicate and collapse the grain? Or do we intentionally want to expand the grain out, like in our *user per day* example?

There’s no right answer here, we have the power to do either as it meets our needs. The key is just to make sure we have a clear sense of our grain for every model we create, that we’ve captured it in a primary key, and that we’re applying tests to ensure that our primary key column is unique and not null.

conclusion

***The grains mean that when in the table there is no primary key or independently the columns are not unique but if we collectively consider 2 or 3 columns and they show unique behavior then it is a grain but separately there wouldn’t be any unique column or any key defined. So we would have to create a surrogate key for this table and a hash function would be used and a unique valued column would create. So now the surrogated key column would possess all the values of the columns on which the key is defined.***

| **user\_id** | **address** |
| --- | --- |
| 1 | 123 Jaffle Ln |
| 2 | 456 Waffle St |
| 3 | 789 Raffle Rd |

In the above table, each user\_id is unique. This table is at the *user* *grain*.

| **user\_id** | **address** |
| --- | --- |
| 1 | 123 Jaffle Ln |
| 1 | 420 Jaffle Ln |
| 2 | 456 Waffle St |
| 3 | 789 Raffle Rd |