```
{“talk_title":
    “Column Name Contracts with dbt”,
 “talk_author”: {
   “author_name”: “Emily Riederer”,
   “author_twtr”: “@emilyriederer”,
   “author_site”: “emilyriederer.com”
  },
 “talk_forum”: {
   “forum_name”: “DataFold”,
   “forum_locn”: “Online”,
   “forum_date”: “2023-05-11”
  }

}
```

Bio

## Emily Riederer

Senior Manager at Capital One

Lead teams focused on data products, analytics, modeling

Involved in open-source data, dbt, and R communities

Agenda

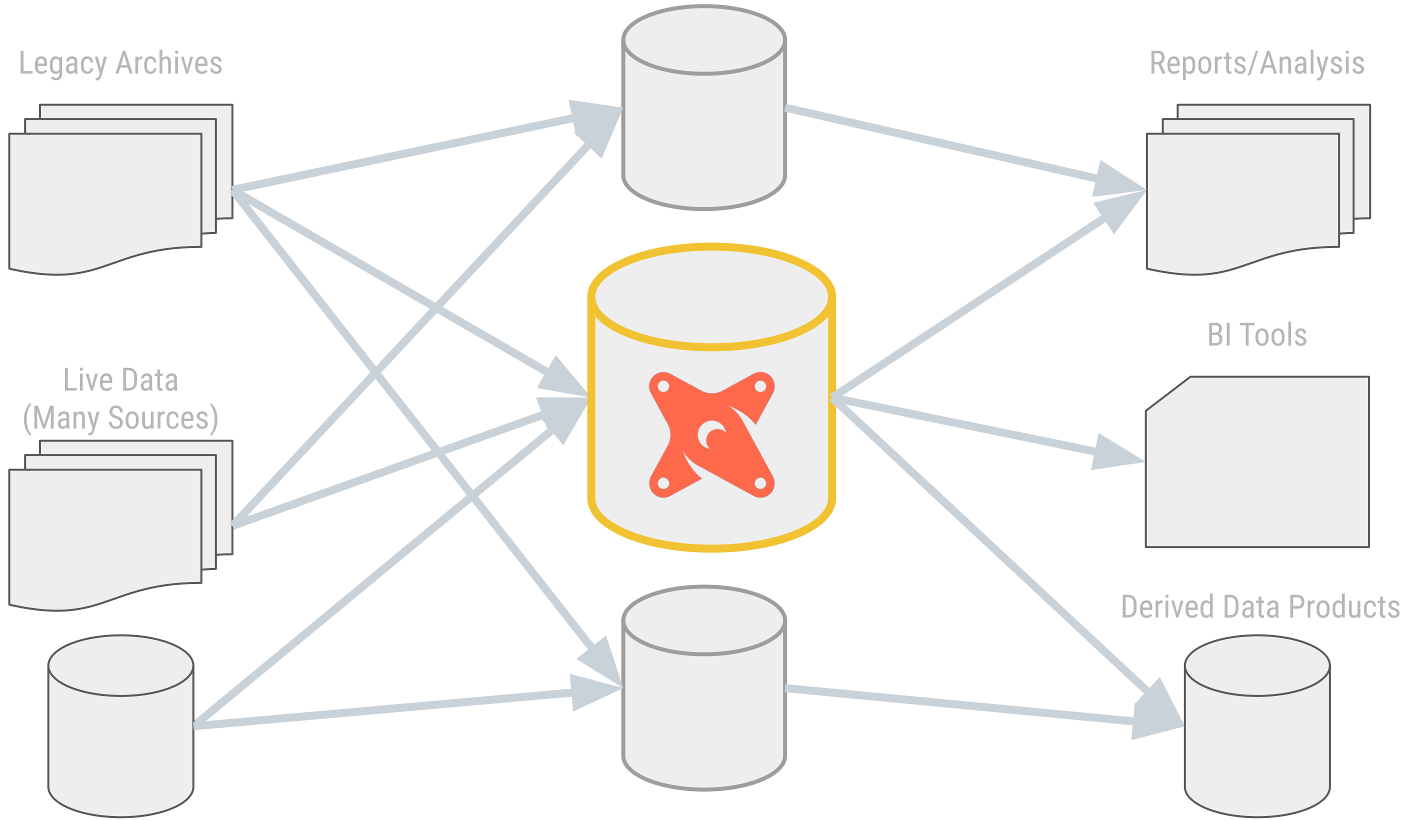
## Column Names as Contracts

The need for latent communication in the data stack

How column-name contracts provide just-in-time context

Using dbt to build scalable, resilient data pipelines with column-name contracts

## Heterogeneous production and consumption patterns motivate the question…



**How to share data context across a complex stack?**

## Controlled vocabularies can create a shared language for our data

| Types |
|---|
| ID |
| IND / IS |
| BIN |
| N |
| AMT |
| VAL |
| DT |
| TM |
| CAT |
| ... |

X

| Subjects |
|---|
| USER |
| LOGIN |
| SESSION |
| CLICK |
| ... |

X

| Details |
|---|
| UTM |
| DURATION |
| ... |

```
{DT | TM}_{LOGIN | SESSION}

ID_{USER | SESSION | LOGIN | VIEW}

{CAT | CD}_SOURCE_UTM

{CAT | CD}_MEDIUM_UTM

AMT_{SESSION | VIEW}_DURATION

...
```

## Controlled vocabulary can embed contracts

| **Stub** | **Semantics** | **Contracts** |
|---|---|---|
| `ID` | Unique entity identifier | Numeric, primary / surrogate key |
| `IND / IS` | Binary 0/1 indicator; **rest of name describes 1 condition** | Always 0 or 1, **non-null** |
| `BIN` | Binary 0/1 indicator; rest of name describes 1 condition | Always 0 or 1 |
| `N` | Count of quantity or event occurrences | Non-negative integer, non-null |
| `DT` | Date of an event | Date, **ISO 8601** (YYYY-MM-DD) |
| `...` | ... | ... |

## Controlled vocabulary can embed contracts

| **Stub** | **Semantics** | **Consequence** |
|---|---|---|
| `USER` | Unique site visitor as determined by IP address | Does **not unique**ly identify a person **across devices** |
| `LOGIN` | A successful authentication (password, MFA) by a confirmed human actor (**after passing Captcha**)<br><br>~~A session beginning with a visit to the login screen~~<br><br>~~The click of the login button after typing username and password~~ | |
| `...` | … | … |

## Contracts create liabilities that dbt can help make good

Inconsistency

Misspelled or free-style column names

▼

**Jinja templates**

*Create valid names and avoid typos*

Infidelity

Incorrect transformation based on contracts

▼

**Custom macros + dbtplyr**

*Iteratively apply transformation based on columns names*

Evasion

Creating problems instead of fixing

▼

**In-pipeline testing**

*Test validity of operations and contract adherence*

## dbtplyr helps maximize the benefits of column-name contracts

| Key Functions | |
|---|---|
| Subset columns by name | `starts_with()`<br>`ends_with()`<br>`contains()`<br>`not_contains()`<br>`one_of()`<br>`not_one_of()`<br>`matches()`<br>`everything()` |
| Iterate over transformations | `across()`<br>`c_across()` |
| Iterate over filters | `if_any()`<br>`if_all()` |

inspired by R's dplyr syntax!

## dbtplyr helps maximize the benefits of column-name contracts

Key Functions

Subset columns by name

```
{% set cols =
          dbtplyr.get_column_names(ref(‘data’)) %}
{% set cols_ind =
          dbtplyr.starts_with(cols, ‘ind’) %}
{% set cols_notnull = [‘x’, ‘y’] %}
```

```
[‘x’,‘y’,‘ind_a’,‘ind_b’]
```

Iterate over transformations

Iterate over filters

# Broken contracts frustrate users

| | ✓ | ✓ | ✗ | ✗ |
|---|---|---|---|---|
| ID_VARIANT | N_CLICK_07 | N_CLICK_14 | N_CLIK_21 | N_28_CLICK |
| 1 | 100 | 172 | 202 | 291 |
| 2 | 112 | 136 | 154 | 191 |
| 3 | 156 | 181 | 202 | 235 |

```
select
  n_click_07,
  n_clik_14..?
from table
```

## Jinja templates enforce consistent naming and definitions

```
{% set lags = [‘07’,’14’,’21’,’24’]%}

select

  id_variant,

  {% for l in var('lags') %}

      count_if(n_days <= {{l}})
          as n_click_{{l}}

  {% if not loop.last %},{% endif %}
  {% endfor %}
```

```
select

  id_variant,

    count_if(n_days <= 07)
        as n_click_07,

    count_if(n_days <= 14)
        as n_click_14
```

## Broken contracts lie to users

```
select count(*)

from logins

where dt_login <= ‘2021-01-01’
```

| DT_LOGIN | ID_LOGIN | IND_LOGIN |
|---|---|---|
| ✗ 2021-01-01T 10:25:28 | 123 | 1 |
| ✗ 2021-01-01T 02:10:53 | 456 | 1 |
| ✗ 2021-01-02T 07:20:00 | 789 | 0 |

| DT_LOGIN | ID_LOGIN | IND_LOGIN |
|---|---|---|
| ✓ 2021-01-01 | 123 | 1 |
| ✓ 2021-01-01 | 456 | 1 |
| ✗ 2021-01-02 | 789 | 0 |

## Custom macros + dbtplyr enforce contracts systemically

```
{% set cols =
         dbtplyr.get_column_names( ref('data') )
%}
{% set cols_n =
         dbtplyr.starts_with(cols, 'n') %}
{% set cols_dt =
         dbtplyr.starts_with(cols, 'dt') %}
{% set cols_ind =
         dbtplyr.starts_with(cols, 'ind') %}

select

  {{ dbtplyr.across(cols_n,
                    "cast({var} as int)
                     as n_{var}")}},
  {{ dbtplyr.across(cols_dt,
                    "date({var})
                     as dt_{var})")}},
  {{ dbtplyr.across(cols_ind,
                    "coalesce({c}, 0)
                     as ind_{var}") }}
```

```
select

  cast(n_a as int64) as n_a,
  cast(n_c as int64) as n_c,

  date(dt_b) as dt_b,
  date(dt_d) as dt_d,

  coalesce(ind_b,0) as ind_b,
  coalesce(ind_c,0) as ind_c
```

## Custom macros + dbtplyr enforce contracts systemically

```
{% set cols =
         dbtplyr.get_column_names( ref('data') )
%}
{% set cols_n =
         dbtplyr.starts_with(cols, 'n') %}
{% set cols_dt =
         dbtplyr.starts_with(cols, 'dt') %}
{% set cols_ind =
         dbtplyr.starts_with(cols, 'ind') %}

select

  {{ dbtplyr.across(cols_n,
                 "cast({var} as int)
                  as n_{var}")}},
  {{ dbtplyr.across(cols_dt,
                 "date({var})
                  as dt_{var})")}},
  {{ dbtplyr.across(cols_ind,
                 "coalesce({c}, 0)
                  as ind_{var}") }}
```

```
select

  cast(n_a as int64) as n_a,
  cast(n_c as int64) as n_c,

  date(dt_b) as dt_b,
  date(dt_d) as dt_d,

  coalesce(ind_b,0) as ind_b,
  coalesce(ind_c,0) as ind_c
```

## Broken contracts evade detection

```
{{ dbtplyr.across(cols_n, “cast({var} as int) as n_{var}”)}}
```

| N_A | N_B |
|---|---|
| 12.00 | 3.25 |
| 19.00 | 4.67 |
| 27.00 | 8.99 |

| N_A ✓ | N_B ✗ |
|---|---|
| 12 | 3 |
| 19 | 5 |
| 27 | 9 |

## Testing confirms any non-enforceable contracts are upheld

```
{% set cols = get_column_names(ref('prep')) %}
{% set cols_n = starts_with(cols, 'n') %}

select *
from {{ ref('my_source') }}
where

 {%- for c in cols_n %}

  abs({{c}} - cast({{c}} as int64)) > 0.01 or

 {% endfor %}

  FALSE
```

```
with dbt__CTE__INTERNAL_test as (

select *
from `db`.`dbt_emily`.`my_source`
where

    abs(n_a - cast(n_a as int64)) > 0.01 or
    abs(n_b - cast(n_b as int64)) > 0.01 or
    abs(n_c - cast(n_c as int64)) > 0.01 or

   FALSE
)

select count(*) from dbt__CTE__INTERNAL_test
```

## Column-name contracts can help deliver intuitive data products at scale

### Better for Users

- Intuitive UI & UX
- Consistent column names
- Credible contracts

### Better for Producers

- Aligned intent across teams
- Dynamic wrangling
- Dynamic testing

### Better for Scale

- Scale wrangling
- Scale testing
- Scale *communication*

```
{“talk_title":
    “Column Name Contracts with dbt”,
 “talk_author”: {
   “author_name”: “Emily Riederer”,
   “author_twtr”: “@emilyriederer”,
   “author_site”: “emilyriederer.com”
  },
 “talk_forum”: {
   “forum_name”: “DataFold”,
   “forum_locn”: “Online”,
   “forum_date”: “2023-05-11”
  }

}
```