

### How to connect to Druid?

#### Port-forward to the Druid cluster

```
kubectl port-forward -n
wiremind-druid-{env} druid-
{env}-broker-{id-container}
8082:8082
```

Example: `kubectl port-forward -n wiremind-druid-staging druid--staging-broker-795c756456-sjbm9 8082:8082`

#### Optional: configure a SQL client

If you want to query Druid with an SQL client, you can do so using [this driver](#). In Database URL, specify: `jdbc:avatica:remote:url=http://localhost:8082/druid/v2/sql/avatica/`

Tools: DBVizualizer, DBeaver, DataGrip, ...

### What data sources can I query?

#### fact\_passenger\_event

Granularity `od_id X event_type X ticket_key`

Refresh `daily`

Time reference `Exact timestamp of the event.`  
(`__time`)

`event_type` can be: `abandon_cart, confirm_purchase, cancel_purchase`

#### fact\_daily\_od\_bucket

Granularity `od_id X bucket_id X day_x`

Refresh `daily`

Time reference `UTC time at train timezone`  
(`__time`) `midnight corresponding to this day_x`

#### fact\_daily\_leg\_physical\_inventory

Granularity `leg_id X physical_inventory_id X day_x`

Refresh `daily`

Time reference `UTC time at train timezone`  
(`__time`) `midnight corresponding to this day_x`

### Query tips

#### Useful SQL functions

Cast datetime to date `CAST(departure_datetime TO DATE)`

#### Most common columns

`__time` If possible, **filter on**.

`day_x` Compute using the service timezone.

`[origin/destination]_station_[id/name/zone]`

`[departure/arrival]_datetime`

`service_[id/number/status]`

`market_[id/name]`

`ordere` You must use ARRAY functions.

`d_-eg_ids`

`od_[capacity/lid/id]`

`bucket_name`

`availability[_physical]_seats_[start/ended]_day`

`[cumulative_]sum_[cancelled/confirmed/net]_bookings`

`[cumulative_]sum_[net]_revenue_vat_[inc/exc]`

`has_event_-occurred` If the line is exactly the same as the day before.

`price_vat_[inc/exc]`

For `fact_passenger_event`, it might be 0 (`abandon_cart`) or negative (`cancelled_purchase`). Use `base_price_vat_inc` for initial price.

`travel_time_minute`

[More information here.](#)



By **tmp\_7890**  
[cheatography.com/tmp-7890/](https://cheatography.com/tmp-7890/)

Not published yet.  
Last updated 5th June, 2020.  
Page 1 of 2.

Sponsored by **CrosswordCheats.com**  
Learn to solve cryptic crosswords!  
<http://crosswordcheats.com>

### Rune tips

#### Bootstrap on Rune

```
from rune.query import Rune
from rune.registry import serpe_factory
from rune.helpers import druid_host_to_connection
SerpeFactory = serpe_factory()
sr = SerpeFactory(druid_host_to_connection('http://localhost:8082'), 'ouigo')
rune = Rune(sr)
```

The rune object has `execute_native_to_df` and `execute_sql_to_df` methods.

#### Native queries

`execute_native` or `execute_native_to_df` have the same interfaces.

First argument: the table you want to query; taken from serpe object (ie: `sr.serpe.fact_passenger_event`).

Optional arguments:

**intervals:** List[str], pre-filter on `__time` column. Must be specified most of the time, allows great optimization.

**columns:** List[Union[str, SerpeEntity]], columns you want to select. You can put either strings or objects from serpe.

**where:** conditions on objects taken from serpe. Ex: `sr.serpe.departure_datetime >= '2019-01-01'`

Check rune documentation for other args: `timeout`, `limit`, `priority`, `querytype`, `use_cache`, ...

Example:

```
rune.execute_native_to_df(sr.serpe.fact_passenger_event, columns=['departure_datetime'], where= sr.serpe.service_number == 7699, limit=10, intervals=['2018-01-01/2018-01-02'])
```

#### How to use CLI?

```
rune ouigo --druid-host http://localhost:8082
```

rune --help if you want more information

