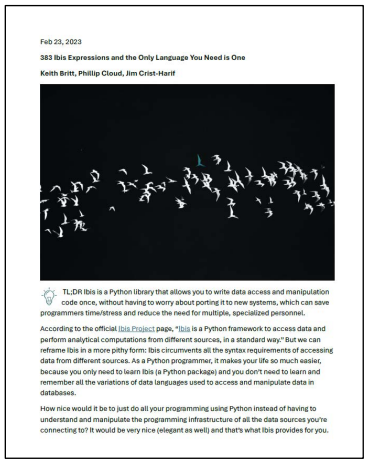
VOLTRON DATA

carahsoft



Feb 23, 2023

383 Ibis Expressions and the Only Language You Need is One

Keith Britt, Phillip Cloud, Jim Crist-Harif

TL;DR Ibis is a Python library that allows you to write data access and manipulation code once, without having to worry about porting it to new systems, which can save programmers time/stress and reduce the need for multiple, specialized personnel.

According to the official Ibis Project page, "Ibis is a Python framework to access data and perform analytical computations from different sources, in a standard way." But we can reframe Ibis in a more pithy form: Ibis circumvents all the syntax requirements of accessing data from different sources. As a Python programmer, it makes your life so much easier, because you only need to learn Ibis (a Python package) and you don't need to learn and remember all the variations of data languages used to access and manipulate data in databases.

How nice would it be to just do all your programming using Python instead of having to understand and manipulate the programming infrastructure of all the data sources you're connecting to? It would be very nice (elegant as well) and that's what Ibis provides for you.

383 Ibis Expressions and the Only Language You Need is One

Thank you for downloading this Voltron Data resource. Carahsoft is the distributor for Voltron Data's AI and ML solutions available via NASA SEWP V, ITES-SW2, NASPO ValuePoint, and many more contract vehicles.

To learn how to take the next step toward acquiring Voltron Data's solutions, please check out the following resources and information:

For Voltron Data overview:
carah.io/Voltron-Data

For upcoming events:
carah.io/Voltron-Events

For additional resources:
carah.io/Voltron-Resources

For additional Artificial Intelligence:
carah.io/ai-solutions

To set up a meeting:
VoltronData@carahsoft.com

To purchase, check out the contract vehicles available for procurement:
carah.io/procurement

Feb 23, 2023

383 Ibis Expressions and the Only Language You Need is One

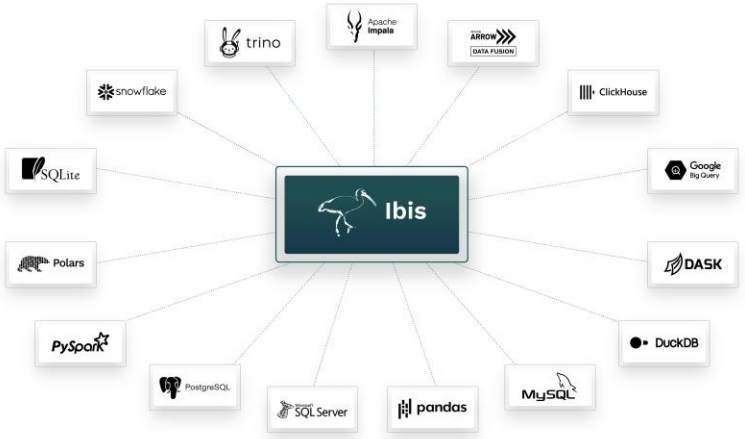
Keith Britt, Phillip Cloud, Jim Crist-Harif

TL;DR Ibis is a Python library that allows you to write data access and manipulation code once, without having to worry about porting it to new systems, which can save programmers time/stress and reduce the need for multiple, specialized personnel.

According to the official Ibis Project page, "Ibis is a Python framework to access data and perform analytical computations from different sources, in a standard way." But we can reframe Ibis in a more pithy form: Ibis circumvents all the syntax requirements of accessing data from different sources. As a Python programmer, it makes your life so much easier, because you only need to learn Ibis (a Python package) and you don't need to learn and remember all the variations of data languages used to access and manipulate data in databases.

How nice would it be to just do all your programming using Python instead of having to understand and manipulate the programming infrastructure of all the data sources you're connecting to? It would be very nice (elegant as well) and that's what Ibis provides for you.

The first question you might ask is, what backends does Ibis support? Good question, with an easy answer. The backends currently supported in Ibis 4.1 are:



The first three (Impala, ClickHouse, Big Query) work the same as they turn the Python instructions you write into strings that are then executed as queries using the respective engines. The latter eleven (Dask, Datafusion, DuckDB, MS SQL Server, Polars, MySQL, PostgreSQL, PySpark, SQLite, Snowflake, and Trino) translate your Python instructions directly into API-specific languages provided by the execution engines. A final backend, pandas, directly executes the backend, as Pandas is a Python library.

The second question you might ask is, what are the expressions for all these backends that are executable by Ibis? Well, let's take a look using some code examples. First, let's just import the necessary packages and do some basic setup:

PYTHON

```python
import ibis
import os
import ibis.expr.operations as ops

ibis.options.interactive = True
```

Now, let's just take a look at the number of operations in the dictionary of Ibis expressions:

PYTHON

```python
print(len(ops.__dict__))
> 383
```

383 operations are supported across all 15 backends. Not all the expressions are supported in each backend, however. Some of the expressions are supported in just a couple of backends, while others are supported across all sixteen. Likewise, the full API for each expression-generating backend is not fully implemented in Ibis, with varying degrees of coverage. The API coverage in Ibis spans a large range from 21% in DataFusion to 84% in Postgres. For a full breakdown of the API and expression coverages, see the table at ibis-project.org.

Let's take a look at how two expressions vary across backends, and how Ibis helps you avoid the complexity associated with moving between different data managers.

Trim

trim is a function that exists in most modern databases. It goes by different names—strip—for example. In Ibis, the string API is modeled after the methods on Python's str object, so it's fittingly called strip.

trim may seem like a trivial thing to abstract over, but it turns out that there are some non-trivial differences in the behavior and spelling of the various databases' implementation of this function.

Postgres and DuckDB

Postgres and DuckDB both spell this function the same way:

```
SELECT trim(some_string)
```

JSX

The function accepts a second argument that is the set of characters to strip from either end of the string:

```
SELECT trim(some_string, 'abc')
```

PYTHON

Note that if any of 'a', 'b', or 'c' are at either end of some_string then all of the matching strings will be trimmed. Here's DuckDB:

```
D SELECT trim('abcdefcba', 'abc');
┌──────────────────────────────────┐
│ main.trim('abcdefcba', 'abc') │
│            varchar             │
├──────────────────────────────────┤
│ def                              │
└──────────────────────────────────┘
```

JSX

MySQL

MySQL takes a different approach. The function name is the same (trim), but the syntax and meaning of arguments are very different from the Postgres/DuckDB approach.

To replicate the same function call in MySQL as the DuckDB example above we have to jump through some hoops:

```sql
SELECT TRIM(BOTH 'c' FROM TRIM(BOTH 'b' FROM TRIM(BOTH 'a' FROM some_string)))
```

SQL

With Ibis, you use a single expression and it handles the syntax conversion for you during execution:

```python
>>> t = ibis.table(dict(some_string="string"), name="t")
>>> expr = t.some_string.strip().name("stripped")
>>> ibis.show_sql(expr, dialect="duckdb")
```

PYTHON

```sql
SELECT
  TRIM('

', t0.some_string) AS stripped
FROM t AS t0
```

SQL

Now for MySQL:

```python
>>> ibis.show_sql(expr, dialect="mysql")
```

PYTHON

```sql
SELECT
  TRIM(BOTH '
            ' FROM TRIM(BOTH '
' FROM TRIM(BOTH '              ' FROM TRIM(BOTH '
' FROM TRIM(BOTH '       ' FROM TRIM(BOTH ' ' FROM (
    t0.some_string
  ))))))) AS stripped
FROM t AS t0
```

You're probably wondering what the heck all the whitespace is. They are the ASCII-defined whitespace characters, which is what strip removes.

Unnest

Moving to more advanced features, we have unnest which turns a column of array values into a flattened column of scalar values. For example, here's an implementation in Ibis:

```python
In [15]: t
Out[15]:
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ arr                       ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ array<string>             │
├───────────────────────────┤
│ ['a', 'b', ... +1]        │
│ ∅                         │
│ []                        │
│ ['b', None]               │
└───────────────────────────┘

In [16]: t.arr.unnest()
Out[16]:
┏━━━━━━━━━━┓
┃ arr      ┃
┡━━━━━━━━━━┩
│ string   │
├──────────┤
│ a        │
│ b        │
│ c        │
│ b        │
│ ∅        │
└──────────┘
```

The syntax for this operation differs wildly across backends. Here's a sampling:

- DuckDB: SELECT unnest(arr)

- Postgres: SELECT unnest(arr)
- Snowflake: SELECT arr FROM FLATTEN(input => t.arr) arr
- BigQuery: SELECT arr FROM t CROSS JOIN UNNEST(arr) AS arr
- PySpark: F.explode(t.arr)

So, the story is even more complicated, but again Ibis saves the day here and captures the variation in most of the supported backends:

DuckDB

```sql
SELECT
  UNNEST(t0.arr) AS arr
FROM t AS t0
```

Postgres

```sql
SELECT
  UNNEST(t0.arr) AS arr
FROM t AS t0
```

Snowflake

```sql
SELECT
  CAST(NULLIF(anon_1.value, '') AS TEXT) AS arr
FROM t AS t0
JOIN LATERAL FLATTEN(
  INPUT => (
    SPLIT(
      ARRAY_TO_STRING(t0.arr, 'b154b027fd4c4f88b11f7334d67dbb11'),
      'b154b027fd4c4f88b11f7334d67dbb11'
    )
  ),
  MODE => 'ARRAY'
) AS anon_1
  ON TRUE
```

Whew, that last one is a doozy! Ibis even takes care of handling nulls the same way across backends so that as you scale up or down you don't see a change in UNNEST's behavior.

That leads to some slightly more complex SQL, but peace of mind as you transition between systems.

Just from these simple examples, we can see that without Ibis, an institution could easily need multiple developers to collaborate to move from one computational code base to another or from one database management system to another. Ibis puts the power to easily migrate in the hands of a single programmer and the code changes are rudimentary. Not too shabby for an open-source software project, eh?

Get started with Ibis, by visiting ibis-project.org's install page, and be sure to visit the excellent tutorial Ibis for SQL Programmers that will walk you step-by-step through getting started with Ibis. Of course, what we do here at Voltron Data is help people stand up and optimize their use of Ibis, Arrow, and other open-source systems so if we can be of any help, please check out our enterprise support options.