

# hail Tables

<https://hail.is/docs/0.2/hail.Table.html>

## Creating Tables

```
ht = hl.read_table('path/table.ht')  
Read in a hail formatted table file.
```

```
ht = hl.import_table('path/dat.csv',  
                    delimiter=',')  
Read in data from a CSV.
```

```
ht = hl.Table.from_pandas(df)  
Create a Table from pandas dataframe.
```

```
ht = hl.utils.range_table(10)  
Create a Table with 10 rows and one field, idx.
```

```
ht = hl.Table.parallelize(  
    hl.literal(  
        [{"a": 4, "b": 7, "c": 10},  
         {"a": 5, "b": 8, "c": 11},  
         {"a": 6, "b": 9, "c": 12}],  
        'array<struct{a:int,b:int,c:int}>'))  
Create a hail table by specifying each row.
```

## Exporting Tables

```
ht.write('path/file.ht')  
Write out the table as hail formatted ht file  
ht.export('path/file.csv', delimiter=',')  
Write out table to a csv.  
df = ht.to_pandas()  
Make a local hail dataframe from the table  
df = ht.to_spark()  
Make a distributed spark dataframe from the table
```

## Globals

Globals are extra table fields that are identical for every row, but are only stored once for efficiency. Globals can be used in hail expressions just like row fields.

```
ht.annotate_globals(source="broad")  
Add a global field called "source" equal to "broad"  
ht.globals.show()  
Show the global fields for this table.
```

## Laziness and Actions – Understanding hail's computational model

For performance reasons, most hail methods are **lazy**. Calling a lazy method does not immediately begin a computation. Instead, it creates a python object representing that computation, which we call an **Expression**. Because of this, many standard python methods won't work on hail expressions.

Python	Hail
3 if x>0 else 2	hl.cond(x>0,3,2)
len(arr)	hl.len(arr)
"foo" in a	a.contains("foo")

Expressions only get evaluated when an **action** is performed. Actions are functions which force hail to compute a result, either by printing some information, returning a local python value, or writing to a file.

Some examples of actions:  
**ht.show()**  
**ht.write(path)**  
**ht.take(k)**  
**ht.collect()**

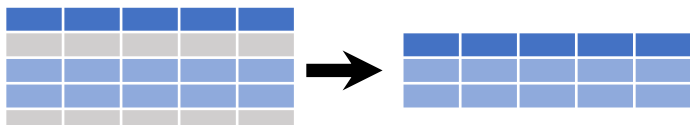
## Exploring Tables

```
ht.describe()  
Print information about the types of each field  
ht.summarize()  
Basic descriptive statistics for each field  
ht.count()  
# of rows in table  
ht.show(n)  
Print first n rows of table (forces computation!)  
ht.n_partitions()  
Check how many partitions your table has.
```

## Adding Keys


```
ht.key_by("year")  
Keys the table by the "year" field.  
ht.key_by()  
Key by with no elements unkeys the table.
```

## Subset Observations (Rows)



```
ht.filter(ht.length > 7)  
Keep rows that meet criteria.  
ht.distinct()  
Remove rows with duplicate keys  
ht.sample(.05)  
Randomly select fraction of rows.  
ht.head(n)  
Subset table to first n rows  
ht.tail(n)  
Subset table to last n rows
```

## Add New Fields



```
ht.annotate(area= ht.length*ht.width)  
Compute and append one or more new fields to each row.  
ht.transmute(area= ht.length*ht.width)  
Like annotate, but deletes referenced fields (length and width above)  
ht.add_index()  
Add a column called "idx" to table that numbers each row in order.
```

## Reshaping Data – Change the layout of a data set

a	b	c
1	'x_14'	False
2	'y_37'	True
3	'g_12'	False
4	'r_1'	False
2	'y_37'	True

**ht1.union(ht2, ht3, ...)**  
Append rows of multiple tables

**ht.order\_by('mpg')**  
Order rows by values of 'mpg' field (low to high).

**ht.order\_by(hl.dsc('mpg'))**  
Order rows by values of 'mpg' field (low to high).


animal	ids
"cat"	[0, 5]
"dog"	[8, 9, 6]

**ht.explode(ht.ids)**  
Create one row for each entry in array field

**ht.rename({'y': 'year'})**  
Rename the fields of a Table

**ht.drop('length', 'height')**  
Drop fields from the table

## Subset Variables (Fields)



```
ht.select('a', 'b')  
Select several fields by name  
ht['a'] or ht.a  
Select single field with specific name  
ht.select(*(x for x in ht.row if re.match(pattern, x)))  
Select fields whose name matches regular expression `pattern`  
ht.drop(*(x for x in ht.row if re.match(pattern, x)))  
Drop fields whose name matches regular expression `pattern`
```

### regex (Regular Expressions) Examples


regex	Examples
'\.'	Matches strings containing a period '.'
'Length\$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]\$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species\$).*	Matches strings except the string 'Species'

## Aggregations

```
ht.aggregate(hl.agg.counter(ht.a))
```

Count number of rows with each unique value for field **a**

id	a
4	"cat"
7	"dog"
9	"cat"



`{"cat":2, "dog":1}`

Besides the above, hail provides a large set of **aggregation functions** that operate on fields of the hail table. They are found in the **hl.agg** module. You can call these functions using **ht.aggregate**.

```
hl.agg.sum(ht.a)
```

Sum values of field **a**.

```
hl.agg.approx_median(ht.a)
```

Median value of field **a**.

```
hl.agg.approx_quantiles(
  ht.a, [.2, .7, .9])
```

Approximate quantiles of field **a**.

```
hl.agg.std(ht.a)
```

Standard deviation of field **a**.

```
hl.agg.min(ht.a)
```

Minimum value of field **a**.

```
hl.agg.max(ht.a)
```

Maximum value of field **a**.

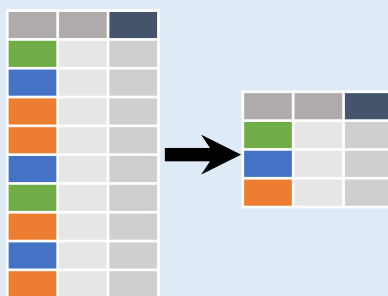
```
hl.agg.mean(ht.a)
```

Mean value of field **a**.

```
hl.agg.var(ht.a)
```

Variance of field **a**.

## Group Data



```
ht.group_by("col")
```

Return a GroupedTable object, grouped by values in column named "col".


```
ht.group_by(level=ht.col % 10)
```

Return a GroupedTable object that is grouped based on the newly computed value **level**

Any call to **group\_by** should always be followed by a call to **aggregate** to get back a Table. See aggregation functions above.

## Scans

idx	num
0	7
1	3
2	5
3	11



idx	num	sum	prod	max
0	7	0	1	NA
1	3	7	7	7
2	5	10	21	7
3	11	15	105	7

```
ht.annotate(sum = hl.scan.sum(ht.num),
  prod = hl.scan.product(ht.num),
  max = hl.scan.max(ht.num))
```

Scans allow rolling aggregations along rows of a table. Each aggregator function has a corresponding scan function.

## Handling Missing Data

```
ht.annotate(x=hl.coalesce(ht.x, val))
```

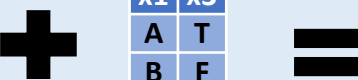
Create a new table where missing values in **x** are replaced by **val**

```
ht.filter(hl.is_defined(ht.x))
```

Create a new table where rows with a missing value for **x** are removed.

## Combine Data Sets

ht1		ht2	
x1	x2	x1	x3
A	1	A	T
B	2	B	F
C	3	B	T
		D	T



Standard Joins (**x1** is the key for both tables)

x1	x2	x3
A	1	T
B	2	F
B	2	T
C	3	NA

```
ht1.join(ht2, how='left')
```

Join matching rows from ht2 to ht1.

x1	x2	x3
A	1	T
B	2	F
B	2	T
D	NA	T

```
ht1.join(ht2, how='right')
```

Join matching rows from ht1 to ht2.

x1	x2	x3
A	1	T
B	2	F
B	2	T

```
ht1.join(ht2, how='inner')
```

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
B	2	T
C	3	NA
D	NA	T

```
ht1.join(ht2, how='outer')
```

Join data. Retain all values, all rows.

x1	x2	x3
A	1	T
B	2	F
C	3	NA

```
ht1.annotate(**ht2[ht1.x1])
```

Join matching rows from ht2 to ht1, does not keep duplicates.

Filtering Joins

x1	x2
A	1
B	2

```
ht1.semi_join(ht2)
```

Keep rows whose keys appear in both ht1 and ht2

x1	x2
C	3

```
ht1.anti_join(ht2)
```

Keep rows whose keys appear in ht1 but not ht2

## Plotting

Hail plotting functions return a figure which can be shown with

```
hl.plot.show(fig)
```

```
hl.plot.histogram(ht.y) hl.plot.scatter(ht.x, ht.y)
```

Histogram of values of field **y** Scatter chart using pairs of points

## Interacting with MatrixTable

From Table to MatrixTable

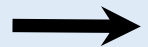
row	col	ent



```
mt = ht.to_matrix_table(row_key=['row'], col_key=['col'])
```

Convert a Table in coordinate representation to a MatrixTable

rk	a	b	c



```
mt = ht.to_matrix_table_row_major(columns=['a', 'b'],
```

```
  entry_field_name='ent', col_field_name='col')
```

Convert a Table in row-major representation to a MatrixTable

From MatrixTable to Table

```
mt.rows()
```

Returns a table with all row fields in the MatrixTable.

```
mt.cols()
```

Returns a table with all col fields in the matrix.

```
mt.entries()
```

Converts the matrix to a table in coordinate form.

```
mt.globals_table()
```

Returns a table with a single row containing the globals.

### Useful Hail Functions

<code>hl.literal(py_obj)</code>	Turn a python object into equivalent hail expression.
<code>hl.if_else(pred, consequent, alternate)</code>	If <b>pred</b> is true, return consequent, else return alternate.
<code>hl.sorted(a)</code>	Sorts array <b>a</b>
<code>hl.argmax(a)</code> <code>hl.argmin(a)</code>	Index of min/max element in <b>a</b>
<code>hl.min(a)/hl.max(a)</code>	Min/max element in <b>a</b>
<code>hl.coalesce(*args)</code>	Return first nonmissing value of args.
<code>a.contains("foo")</code>	Check if array <b>a</b> contains "foo"
<code>hl.is_missing(expr)</code>	Check if an expr is missing
<code>hl.is_nan(expr)</code>	Check if an expr is NaN
<code>&amp;,  , ~, ^</code>	Logical and/or/not/xor