IPFROMINT Function

Computes a four-octet internet protocol (IP) address from a 32-bit integer input.

Source value must be a valid integer within the range specified by the formula below. A valid IPv4 address is in the following format:

```
aaa.bbb.ccc.ddd
```

NOTE: IPv6 addresses are not supported.

The formula used to compute the integer equivalent of an IP address is the following:

```
(aaa * 256^3) + (bbb * 256^2) + (ccc * 256) + (ddd)
```

So, the formula to compute this IP address is the following:

Input	aaa	bbb	ccc	ddd
X	aaa = floor(Input / (256^3)) remainderA = Input - aaa	bbb = floor(remainderA / (256^2)) remainderB = remainderA - bbb	ccc = floor(remainderB / 256) remainderC = remainderB - ccc	ddd = remainderC

Output value:

```
Output = aaa + '.' + bbb + '.' + ccc + '.' + ddd
```

Basic Usage

Numeric literal example:

```
derive type:single value: IPFROMINT('16909060' ) as:'ip_addr'
```

Output: Generates a column containing the IP address `1.2.3.4`.

Column reference example:

```
derive type:single value: IPFROMINT(IpInt) as: 'ip_addr'
```

Output: Generates the new `ip_addr` column containing the values of the `IpInt` column converted to an IP address value.

Syntax and Arguments

```
derive type:single value: IPFROMINT(column_int)
```

Argument	Required?	Data Type	Description
column_int	Y	string or integer	Name of column or integer literal that is to be converted to an IP address value

For more information on syntax standards, see *Language Documentation Syntax Notes*.

column_int

Name of the column or integer literal whose values are used to compute the equivalent IP address value.

- Missing input values generate missing results.
- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	Integer literal or column reference	`16909060`

Examples

> **Tip:** For additional examples, see *Common Tasks*.

Example - Convert IP addresses to integers

This examples illustrates how you can convert IP addresses to numeric values for purposes of comparison and sorting. This example illustrates the following functions:

- `IPTOINT` - converts an IP address to an integer value according to a formula. See *IPTOINT Function*.
- `IPFROMINT` - converts an integer value back to an IP address according to formula. See *IPFROMINT Function*.

Source:

Your dataset includes the following values for IP addresses:

IpAddr
192.0.0.1
10.10.10.10
1.2.3.4
1.2.3
http://12.13.14.15
https://16.17.18.19

Transform:

When the above data is imported, the application initially types the column as URL values, due to the presence of the `http://` and `https://` protocol identifiers. Select the IP Address data type for the column. The last three values are listed as mismatched values. You can fix the issues with the last two entries by applying the following transform, which matches on both `http://` and `https://` strings:

```
replace col:IpAddr with:'' on:`http%?://`
```

> **NOTE:** The `%?` Trifacta® pattern matches zero or one time on any character, which enables the matching on both variants of the protocol identifier.

Now, only the `1.2.3` value is mismatched. Perhaps you know that there is a missing zero at the end of it. To add it back, you can do the following:

```
replace col: IpAddr on: `1.2.3{end}` with: '1.2.3.0' global: true
```

All values in the column should be valid for the IP Address data type. To convert these values to their integer equivalents:

```
derive type:single value:IPTOINT(IpAddr) as:'ip_as_int'
```

You can now manipulate the data based on this numeric key. To convert the integer values back to IP addresses for checking purposes, use the following:

```
derive type:single value:IPFROMINT(ip_as_int) as:'ip_check'
```

Results:

X	ip_as_int	ip_check
192.0.0.1	3221225473	192.0.0.1
10.10.10.10	168430090	10.10.10.10
1.2.3.4	16909060	1.2.3.4
1.2.3.0	16909056	1.2.3.0
12.13.14.15	202182159	12.13.14.15
16.17.18.19	269554195	16.17.18.19