One of the design goals of the cdata [R](https://www.r-project.org/) package is that very powerful and arbitrary record transforms should be convenient and take only one or two steps. In fact it is the goal to take just about any record shape to any other in two steps: first convert to row-records, then re-block the data into arbitrary record shapes

But as with all general ideas, it is much easier to see what we mean by the above with a concrete example.

Let’s consider the following artificial (but simple) example. Suppose we have the following data.

library("cdata")

data <- build\_frame(

'record\_id', 'row', 'col1', 'col2', 'col3' |

1, 'row1', 1, 2, 3 |

1, 'row2', 4, 5, 6 |

1, 'row3', 7, 8, 9 |

2, 'row1', 11, 12, 13 |

2, 'row2', 14, 15, 16 |

2, 'row3', 17, 18, 19 )

knitr::kable(data)

| **record\_id** | **row** | **col1** | **col2** | **col3** |
| --- | --- | --- | --- | --- |
| 1 | row1 | 1 | 2 | 3 |
| 1 | row2 | 4 | 5 | 6 |
| 1 | row3 | 7 | 8 | 9 |
| 2 | row1 | 11 | 12 | 13 |
| 2 | row2 | 14 | 15 | 16 |
| 2 | row3 | 17 | 18 | 19 |

In the above the records are the triples of rows with matching record\_id and the different rows within the record are identified by the value in the row column. So The data items are named by the triplet record\_id, row and renaming column name (col1, col2, or col2).

Suppose we want to transpose each of the records- swapping the row and column notions. With cdata this is easy. First you design a transform to flatten each complex record into a single wide row. Essentially that is just specifying the following control variables. We define how to identify records (the key columns) and the structure of the records (giving the interior of the record arbitrary names we will re-use later).

keyColumns = 'record\_id'

incoming\_shape <- qchar\_frame(

row, col1, col2, col3 |

row1, v11, v12, v13 |

row2, v21, v22, v23 |

row3, v31, v32, v33 )

And we specify (using the same principles) the desired final record shape, re-using the interior names from the first step to show where values are to be mapped.

outgoing\_shape <- qchar\_frame(

column, row1, row2, row3 |

col1, v11, v21, v31 |

col2, v12, v22, v32 |

col3, v13, v23, v33 )

Once you have done this the conversion is accomplished in two function calls.

rowrecs <- blocks\_to\_rowrecs(

data,

keyColumns = keyColumns,

controlTable = incoming\_shape)

transformed <- rowrecs\_to\_blocks(

rowrecs,

controlTable = outgoing\_shape,

columnsToCopy = keyColumns)

knitr::kable(transformed)

| **record\_id** | **column** | **row1** | **row2** | **row3** |
| --- | --- | --- | --- | --- |
| 1 | col1 | 1 | 4 | 7 |
| 1 | col2 | 2 | 5 | 8 |
| 1 | col3 | 3 | 6 | 9 |
| 2 | col1 | 11 | 14 | 17 |
| 2 | col2 | 12 | 15 | 18 |
| 2 | col3 | 13 | 16 | 19 |

And the transform is done, each record has been transposed. The principle is "draw a picture." First we draw a picture of the block record structure we have, and then we draw a picture of the block record structure we want. The intermediate form (rowrecs) is a special form where the concepts of records and rows exactly agree. In this form each record is one exactly row and each row is exactly one record. This data looks like the following.

knitr::kable(rowrecs)

| **record\_id** | **v11** | **v21** | **v31** | **v12** | **v22** | **v32** | **v13** | **v23** | **v33** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 4 | 7 | 2 | 5 | 8 | 3 | 6 | 9 |
| 2 | 11 | 14 | 17 | 12 | 15 | 18 | 13 | 16 | 19 |

We have complete freedom to re-name columns and record-piece labels (the labels that tell us which portion of a block-record each row fits into).

In the development version of cdata we can make things even easier

t2 <- convert\_records(

data,

keyColumns = keyColumns,

incoming\_shape = incoming\_shape,

outgoing\_shape = outgoing\_shape)

knitr::kable(t2)

| **record\_id** | **column** | **row1** | **row2** | **row3** |
| --- | --- | --- | --- | --- |
| 1 | col1 | 1 | 4 | 7 |
| 1 | col2 | 2 | 5 | 8 |
| 1 | col3 | 3 | 6 | 9 |
| 2 | col1 | 11 | 14 | 17 |
| 2 | col2 | 12 | 15 | 18 |
| 2 | col3 | 13 | 16 | 19 |

Also these conversions can also be translated into rquery operators, and therefore saved to be run either in memory or directly on a database.

table\_desciption <- rquery::local\_td(data)

ops <- table\_desciption %.>%

convert\_records(

.,

keyColumns = keyColumns,

incoming\_shape = incoming\_shape,

outgoing\_shape = outgoing\_shape)

cat(format(ops))

#> table(data;

#> record\_id,

#> row,

#> col1,

#> col2,

#> col3) %.>%

#> non\_sql\_node(., blocks\_to\_rowrecs(.)) %.>%

#> non\_sql\_node(., rowrecs\_to\_blocks(.))

rquery::column\_names(ops)

#> [1] "record\_id" "column" "row1" "row2" "row3"