

# Package ‘CellaRepertorium’

February 27, 2021

**Type** Package

**Title** Data structures, clustering and testing for single cell immune receptor repertoires (scRNAseq RepSeq/AIRR-seq)

**Version** 1.0.0

**Description** Methods to cluster and analyze high-throughput single cell immune cell repertoires, especially from the 10X Genomics VDJ solution. Contains an R interface to CD-HIT (Li and Godzik 2006). Methods to visualize and analyze paired heavy-light chain data. Tests for specific expansion, as well as omnibus oligoclonality under hypergeometric models.

**License** GPL-3

**Depends** R (>= 4.0)

**Imports** dplyr, tibble, stringr, Biostrings, Rcpp, reshape2, methods, rlang (>= 0.3), purrr, Matrix, S4Vectors, BiocGenerics, tidyr, forcats, progress, stats, utils

**Suggests** testthat, readr, knitr, rmarkdown, ggplot2, BiocStyle, ggdendro, broom, lme4, RColorBrewer, SingleCellExperiment, scater, broom.mixed, cowplot

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**Encoding** UTF-8

**NeedsCompilation** yes

**RoxygenNote** 7.1.1

**URL** <https://github.com/amcdavid/CellaRepertorium>

**BugReports** <https://github.com/amcdavid/CellaRepertorium/issues>

**Roxygen** list(markdown = TRUE)

**biocViews** RNASeq, Transcriptomics, SingleCell, TargetedResequencing, Technology, ImmunoOncology, Clustering

**git\_url** <https://git.bioconductor.org/packages/CellaRepertorium>

**git\_branch** RELEASE\_3\_12

**git\_last\_commit** 4e5b113

**git\_last\_commit\_date** 2020-10-27

**Date/Publication** 2021-02-26

**Author** Andrew McDavid [aut, cre],  
 Yu Gu [aut],  
 Erik VonKaenel [aut],  
 Thomas Lin Pedersen [ctb]

**Maintainer** Andrew McDavid <Andrew\_McDavid@urmc.rochester.edu>

## R topics documented:

.cluster_permute_test . . . . .	3
canonicalize_cell . . . . .	3
canonicalize_cluster . . . . .	5
ccdb_ex . . . . .	6
ccdb_join . . . . .	6
cdhit . . . . .	7
cdhit_ccdb . . . . .	8
cluster_filterset . . . . .	9
cluster_germline . . . . .	10
cluster_permute_test . . . . .	11
cluster_plot . . . . .	12
cluster_test_by . . . . .	13
ContigCellDB . . . . .	14
contigs_qc . . . . .	16
crosstab_by_celltype . . . . .	16
entropy . . . . .	17
equalize_ccdb . . . . .	18
fancy_name_contigs . . . . .	19
filter_cdb . . . . .	19
fine_clustering . . . . .	20
fine_cluster_seqs . . . . .	21
generate_pseudobulk . . . . .	22
guess_celltype . . . . .	23
hushWarning . . . . .	24
ig_chain_recode . . . . .	24
map_axis_labels . . . . .	25
pairing_tables . . . . .	26
purity . . . . .	28
rank_prevalence_ccdb . . . . .	28
rbind,ContigCellDB-method . . . . .	29
right_join_warn . . . . .	30
split_cdb . . . . .	31
[[,ContigCellDB,character,missing-method . . . . .	32
\$/,ContigCellDB-method . . . . .	33

---

.cluster\_permute\_test *Cell permutation tests (internal)*

---

### Description

Cell permutation tests (internal)

### Usage

```
.cluster_permute_test(  
  labels,  
  covariates,  
  strata,  
  statistic,  
  n_perm,  
  alternative,  
  ...  
)
```

### Arguments

labels	factor of length n
covariates	data.frame of length n
strata	factor
statistic	function of label (vector) and covariate (data.frame). Must return a scalar
n_perm	number of permutations to run
alternative	character naming the direction statistic should be fall under the alternative hypothesis
...	passed along to statistic

### Value

a list containing the observed value of the statistic, its expectation (under independence), a p-value, and the Monte Carlo standard error (of the expected value).

---

canonicalize\_cell *Find a canonical contig to represent a cell*

---

### Description

Using filtering in `contig_filter_args` and sorting in `tie_break_keys` and order find a single, canonical contig to represent each cell Fields in `contig_fields` will be copied over to the `cell_tbl`.

**Usage**

```
canonicalize_cell(
  ccdb,
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads"),
  contig_fields = tie_break_keys,
  order = 1,
  overwrite = TRUE
)
```

**Arguments**

<code>ccdb</code>	<a href="#">ContigCellDB()</a>
<code>contig_filter_args</code>	an expression passed to <a href="#">dplyr::filter()</a> . Unlike <code>filter</code> , multiple criteria must be & together, rather than using commas to separate. These act on <code>ccdb\$contig_tbl</code>
<code>tie_break_keys</code>	(optional) character naming fields in <code>contig_tbl</code> that are used sort the contig table in descending order. Used to break ties if <code>contig_filter_args</code> does not return a unique contig for each cluster
<code>contig_fields</code>	Optional fields from <code>contig_tbl</code> that will be copied into the <code>cluster_tbl</code> from the canonical contig.
<code>order</code>	The rank order of the contig, based on <code>tie_break_keys</code> to return. If <code>tie_break_keys</code> included an ordered factor (such as <code>chain</code> ) this could be used to return the second chain.
<code>overwrite</code>	logical – should non-key fields in <code>y</code> be overwritten using <code>x</code> , or should a suffix ( <code>".y"</code> ) be added

**Value**

[ContigCellDB\(\)](#) with some number of clusters/contigs/cells but with "canonical" values copied into `cell_tbl`

**See Also**

[canonicalize\\_cluster\(\)](#)

**Examples**

```
# Report beta chain with highest umi-count, breaking ties with reads
data(ccdb_ex)
beta = canonicalize_cell(ccdb_ex, chain == 'TRB',
  tie_break_keys = c('umis', 'reads'),
  contig_fields = c('umis', 'reads', 'chain', 'v_gene', 'd_gene', 'j_gene'))
head(beta$cell_tbl)

# Stable: only adds fields to `cell_tbl`
stopifnot(dplyr::all_equal(beta$cell_tbl[ccdb_ex$cell_pk],
  ccdb_ex$cell_tbl[ccdb_ex$cell_pk], ignore_row_order = TRUE))

#Report cdr3 with highest UMI count, but only when > 5 UMIs support it
umi5 = canonicalize_cell(ccdb_ex, umis > 5,
  tie_break_keys = c('umis', 'reads'), contig_fields = c('umis', 'cdr3'))
```

```
stopifnot(all(umi5$cell_tbl$umis > 5, na.rm = TRUE))
```

---

canonicalize\_cluster *Find a canonical contig to represent a cluster*

---

## Description

Find a canonical contig to represent a cluster

## Usage

```
canonicalize_cluster(
  ccdb,
  contig_filter_args,
  tie_break_keys = character(),
  order = 1,
  representative = ccdb$cluster_pk[1],
  contig_fields = c("cdr3", "cdr3_nt", "chain", "v_gene", "d_gene", "j_gene"),
  overwrite = TRUE
)
```

## Arguments

ccdb	<a href="#">ContigCellDB()</a>
contig_filter_args	an expression passed to <a href="#">dplyr::filter()</a> . Unlike <a href="#">filter</a> , multiple criteria must be & together, rather than using commas to separate. These act on <code>ccdb\$contig_tbl</code>
tie_break_keys	(optional) character naming fields in <code>contig_tbl</code> that are used sort the contig table in descending order. Used to break ties if <code>contig_filter_args</code> does not return a unique contig for each cluster
order	The rank order of the contig, based on <code>tie_break_keys</code> to return. If <code>tie_break_keys</code> included an ordered factor (such as <code>chain</code> ) this could be used to return the second chain.
representative	an optional field from <code>contig_tbl</code> that will be made unique. Serve as a surrogate <code>cluster_pk</code> .
contig_fields	Optional fields from <code>contig_tbl</code> that will be copied into the <code>cluster_tbl</code> from the canonical contig.
overwrite	logical – should non-key fields in <code>y</code> be overwritten using <code>x</code> , or should a suffix ( <code>".y"</code> ) be added

## Value

[ContigCellDB\(\)](#) with some number of clusters/contigs/cells but with "canonical" values copied into `cluster_tbl`

## See Also

[canonicalize\\_cell\(\)](#) [left\\_join\\_warn\(\)](#)

**Examples**

```

library(dplyr)
data(ccdb_ex)
ccdb_ex_small = ccdb_ex
ccdb_ex_small$cell_tbl = ccdb_ex_small$cell_tbl[1:200,]
ccdb_ex_small = cdhit_ccdb(ccdb_ex_small,
sequence_key = 'cdr3_nt', type = 'DNA', cluster_name = 'DNA97',
identity = .965, min_length = 12, G = 1)
ccdb_ex_small = fine_clustering(ccdb_ex_small, sequence_key = 'cdr3_nt', type = 'DNA')

# Canonicalize with the medoid contig is probably what is most common
ccdb_medoid = canonicalize_cluster(ccdb_ex_small)

# But there are other possibilities.
# To pass multiple "AND" filter arguments must use &
ccdb_umi = canonicalize_cluster(ccdb_ex_small,
contig_filter_args = chain == 'TRA' & length > 500, tie_break_keys = 'umis',
contig_fields = c('chain', 'length'))
ccdb_umi$cluster_tbl %>% dplyr::select(chain, length) %>% summary()

```

---

`ccdb_ex`*A preconstructed ContigClusterDB from the contigs\_qc data*

---

**Description**

A preconstructed ContigClusterDB from the contigs\_qc data

**Usage**`data(ccdb_ex)`**Format**

```

ccdb_ex = ContigCellDB_10XVDJ(contigs_qc, contig_pk = c('pop', 'sample', 'barcode', 'contig_id'), cell_pk
= c('pop', 'sample', 'barcode'))

```

**See Also**[contigs\\_qc](#)

---

`ccdb_join`*Join dataframe or SingleCellExperiment object with ContigCellDB object*

---

**Description**

Join dataframe or SingleCellExperiment object with ContigCellDB object

**Usage**`ccdb_join(template, ccdb, join_fun = dplyr::left_join, by = ccdb$cell_pk)`

**Arguments**

template	data.frame or SingleCellExperiment object to be joined with ccdb.
ccdb	A ContigCellDB object.
join_fun	Function used for the join operation.
by	A character vector of variables to join by.

**Value**

`ContigCellDB()`

**Examples**

```
data(ccdb_ex)
to_join = dplyr::bind_rows(ccdb_ex$cell_tbl[1:10,],
  dplyr::tibble(barcode = c('extra1', 'extra2'), sample = LETTERS[1:2],
  pop = LETTERS[1:2]))
ccdb_join(to_join, ccdb_ex)
```

---

cdhit

*R interface to CDHIT/CDHITest*

---

**Description**

CDHIT is a greedy algorithm to cluster amino acid or DNA sequences based on a minimum identity. By default, in this package it is configured perform ungapped, global alignments with no clipping at start or end. The `identity` is the number of identical characters in alignment divided by the full length of the shorter sequence. Set `s < 1` to change the minimum coverage of the shorter sequence, which will allow clipping at start or end. Changing `G = 0` changes the meaning of the `identity` to be the number of identical characters in the alignment divided by the length of the alignment. In this case, you must also set the alignment coverage controls `aL`, `AL`, `aS`, `AS`.

**Usage**

```
cdhit(
  seqs,
  identity = NULL,
  kmerSize = NULL,
  min_length = 6,
  s = 1,
  only_index = FALSE,
  showProgress = interactive(),
  ...
)
```

**Arguments**

seqs	AAseq or DNaseq
identity	minimum proportion identity
kmerSize	word size. If NULL, it will be chosen automatically based on the identity. You may need to lower it below 5 for AAseq with identity less than .7.

min_length	Minimum length for sequences to be clustered. An error if something smaller is passed.
s	fraction of shorter sequence covered by alignment.
only_index	if TRUE only return the integer cluster indices, otherwise return a tibble.
showProgress	show a status bar
...	other arguments that can be passed to cdhit, see <a href="https://github.com/weizhongli/cdhit/wiki/3.-User's-Guide#CDHIT">https://github.com/weizhongli/cdhit/wiki/3.-User's-Guide#CDHIT</a> for details. These will override any default values.

### Details

CDHit is by Fu, Niu, Zhu, Wu and Li (2012). The R interface is originally by Thomas Lin Pedersen and was transcribed here because it is not exported from the package FindMyFriends, which is orphaned.

### Value

vector of integer of length seqs providing the cluster ID for each sequence, or a tibble. See details.

### Examples

```
fasta_path = system.file('extdata', 'demo.fasta', package='CellaRepertorium')
aaseq = Biostrings::readAAStringSet(fasta_path)
# 100% identity, global alignment
cdhit(aaseq, identity = 1, only_index = TRUE)[1:10]
# 100% identity, local alignment with no padding of endpoints
cdhit(aaseq, identity = 1, G = 0, aL = 1, aS = 1, only_index = TRUE)[1:10]
# 100% identity, local alignment with .9 padding of endpoints
cdhit(aaseq, identity = 1, G = 0, aL = .9, aS = .9, only_index = TRUE)[1:10]
# a tibble
tbl = cdhit(aaseq, identity = 1, G = 0, aL = .9, aS = .9, only_index = FALSE)
```

---

cdhit\_ccdb

*Use `cdhit()` to cluster a `ContigCellDB()`*

---

### Description

Use `cdhit()` to cluster a `ContigCellDB()`

### Usage

```
cdhit_ccdb(
  ccdb,
  sequence_key,
  type = c("DNA", "AA"),
  cluster_pk = "cluster_idx",
  ...
)
```



**Arguments**

ccdb	An object of class <code>ContigCellDB()</code>
sequence_key	character naming the column in the <code>contig_tbl</code> containing the sequence to be clustered
type	one of 'DNA' or 'AA'
cluster_pk	character specifying key, and name for the clustering.
...	Arguments passed on to <code>cdhit</code>
	<code>identity</code> minimum proportion identity
	<code>kmerSize</code> word size. If NULL, it will be chosen automatically based on the <code>identity</code> . You may need to lower it below 5 for AAseq with identity less than .7.
	<code>min_length</code> Minimum length for sequences to be clustered. An error if something smaller is passed.
	<code>s</code> fraction of shorter sequence covered by alignment.
	<code>showProgress</code> show a status bar

**Value**

`ContigCellDB()`

**See Also**

`cdhit()`

**Examples**

```
data(ccdb_ex)
res = cdhit_ccdb(ccdb_ex, 'cdr3_nt', type = 'DNA',
  cluster_name = 'DNA97', identity = .965, min_length = 12, G = 1)
res$cluster_tbl
res$contig_tbl
res$cluster_pk
```

---

cluster\_filterset      *A filtration of clusters*

---

**Description**

Return clusters that match all provided conditions

**Usage**

```
cluster_filterset(min_number = 0, min_freq = 0, white_list = NULL)
```

**Arguments**

min_number	integer At least this many cells
min_freq	numeric At least this frequency
white_list	data.frame keyed by <code>cluster_pk</code> that must match

**Value**

object representing the filtration (currently a list)

**Examples**

```
cluster_filterset(min_number = 1, min_freq = 0)
```

---

cluster_germline	<i>Cluster contigs by germline properties</i>
------------------	---

---

**Description**

Cluster contigs by germline properties

**Usage**

```
cluster_germline(  
  ccdb,  
  segment_keys = c("v_gene", "j_gene", "chain"),  
  cluster_pk = "cluster_idx"  
)
```

**Arguments**

ccdb	<a href="#">ContigCellDB()</a>
segment_keys	fields in contig_tbl that identify a cluster
cluster_pk	name of cluster to be added to cluster_tbl

**Value**

[ContigCellDB\(\)](#)

**Examples**

```
data(ccdb_ex)  
ccdb_ex = cluster_germline(ccdb_ex)  
ccdb_ex$cluster_tbl
```

---

cluster\_permute\_test *Tests for independence between labels and covariates using permutation of cells*

---

### Description

This tests a statistic for association between labels (for instance, cluster/clonal ID) and covariates (for instance, subject or treatment) by permuting the link between the two. Each observation represents a cell. statistic is any function of labels

### Usage

```
cluster_permute_test(
  ccdb,
  cell_covariate_keys,
  cell_label_key = ccdb$cluster_pk,
  cell_stratify_keys,
  statistic,
  n_perm,
  alternative = c("two.sided", "less", "greater"),
  sanity_check_strata = TRUE,
  ...
)
```

### Arguments

ccdb	ContigCellDB
cell_covariate_keys	character naming fields in ccdb\$cell_tbl
cell_label_key	character naming a single field in ccdb\$cell_tbl
cell_stratify_keys	optional character naming fields in ccdb\$cell_tbl under which permutations of cell_label_key will occur. This means that the test will occur conditional on these covariates. Must be disjoint from cell_covariate_keys.
statistic	function of label (vector) and covariate (data.frame). Must return a scalar
n_perm	number of permutations to run
alternative	character naming the direction statistic should be fall under the alternative hypothesis
sanity_check_strata	logical, should cell_stratify_keys be checked for sanity?
...	passed to statistic

### Value

a list containing the observed value of the statistic, its expectation (under independence), a p-value, and the Monte Carlo standard error (of the expected value).

### See Also

[purity\(\)](#)

**Examples**

```
library(dplyr)
# covariate should name one or more columns in `cell_tbl`

cluster_idx = c(1, 1, 1, 2, 2, 3, 3)
subject = c('A', 'A', 'B', 'B', 'B', 'C', 'C')
contig_tbl = tibble(contig_pk = seq_along(cluster_idx), cluster_idx, subject)
ccdb_test = ContigCellDB(contig_tbl = contig_tbl, contig_pk = 'contig_pk',
cell_pk = c('contig_pk', 'subject', 'cluster_idx'), cluster_pk = 'cluster_idx')
ccdb_test$cell_tbl

cluster_permute_test(ccdb_test, 'subject', 'cluster_idx',
statistic = purity, n_perm = 50)
```

---

cluster\_plot

---

*Make a plot showing properties of the clustering*


---

**Description**

The number of elements per cluster and the average distance between the medoid and other elements are plotted.

**Usage**

```
cluster_plot(cdb, return_plotlist = FALSE)
```

**Arguments**

cdb                    A fine\_clustering ContigCellDB object

return\_plotlist        should a list of ggplot2 plots be returned. If FALSE, a cowplot composite is returned.

**Value**

a cowplot composite or a list of plots.

**Examples**

```
library(dplyr)
data(ccdb_ex)
ccdb_ex_small = ccdb_ex
ccdb_ex_small$cell_tbl = ccdb_ex_small$cell_tbl[1:200,]
ccdb_ex_small = cdhit_ccdb(ccdb_ex_small,
sequence_key = 'cdr3_nt', type = 'DNA', cluster_name = 'DNA97',
identity = .965, min_length = 12, G = 1)
ccdb_ex_small = fine_clustering(ccdb_ex_small, sequence_key = 'cdr3_nt', type = 'DNA')

# Canonicalize with the medoid contig is probably what is most common
ccdb_medoid = canonicalize_cluster(ccdb_ex_small)

# But there are other possibilities.
```

```
# To pass multiple "AND" filter arguments must use &
ccdb_umi = canonicalize_cluster(ccdb_ex_small,
contig_filter_args = chain == 'TRA' & length > 500, tie_break_keys = 'umis',
contig_fields = c('chain', 'length'))
ccdb_umi$cluster_tbl %>% dplyr::select(chain, length) %>% summary()
cluster_plot(ccdb_ex_small)
```

---

cluster_test_by	<i>Test clusters for differential usage</i>
-----------------	---

---

## Description

Typically one will want to stratify by chain by calling `cluster_test_by`, as this will calculate the number of cell "trials" separately depending on the chain recovered.

## Usage

```
cluster_test_by(ccdb, fields = "chain", tbl = "cluster_tbl", ...)
```

```
cluster_logistic_test(
  formula,
  ccdb,
  filterset = cluster_filterset(),
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads"),
  add_cluster_tbl = FALSE,
  keep_fit = FALSE,
  fitter = glm_glmmer,
  silent = FALSE
)
```

## Arguments

<code>ccdb</code>	<code>ContigCellDB()</code>
<code>fields</code>	character naming fields in <code>tbl</code>
<code>tbl</code>	one of <code>contig_tbl</code> , <code>cell_tbl</code> or <code>cluster_tbl</code>
<code>...</code>	passed to <code>cluster_logistic_test</code>
<code>formula</code>	the <b>right-hand side</b> of a glmer or glm-style formula.
<code>filterset</code>	a call to <code>cluster_filterset()</code> that will be used to subset clusters.
<code>contig_filter_args</code>	an expression passed to <code>dplyr::filter()</code> . Unlike <code>filter</code> , multiple criteria must be & together, rather than using commas to separate. These act on <code>ccdb\$contig_tbl</code>
<code>tie_break_keys</code>	(optional) character naming fields in <code>contig_tbl</code> that are used sort the contig table in descending order. Used to break ties if <code>contig_filter_args</code> does not return a unique contig for each cluster
<code>add_cluster_tbl</code>	logical should the output be joined to the <code>cluster_tbl</code> ?
<code>keep_fit</code>	logical as to whether the fit objects should be returned as a list column

fitter	a function taking arguments formula, data, is_mixed and keep_fit that is run on each cluster. Should return a tibble or data.frame
silent	logical. Should warnings from fitting functions should be suppressed?

**Value**

table with one row per cluster/term.

**Functions**

- cluster\_test\_by: split ccdb and conduct tests within strata

**Examples**

```
library(dplyr)
data(ccdb_ex)
ccdb_ex = cluster_germline(ccdb_ex)
trav1 = filter(ccdb_ex$cluster_tbl, v_gene == 'TRAV1')
cluster_logistic_test(~pop + (1|sample), ccdb_ex,
  filterset = cluster_filterset(white_list= trav1))
# Fixed effect analysis of each cluster, by chain
prev4 = ccdb_ex$contig_tbl %>% group_by(cluster_idx) %>%
  summarize(n()) %>% filter(`n()`>= 4)
cluster_test_by(ccdb = ccdb_ex, fields = 'chain',
  tbl = 'cluster_tbl', formula = ~ pop, filterset = cluster_filterset(white_list= prev4))
```

---

ContigCellDB

---

*Construct a ContigCellDB*


---

**Description**

Construct a ContigCellDB

**Usage**

```
ContigCellDB(
  contig_tbl,
  contig_pk,
  cell_tbl,
  cell_pk,
  cluster_tbl,
  cluster_pk = character(),
  equalize = TRUE
)

ContigCellDB_10XVDJ(
  contig_tbl,
  contig_pk = c("barcode", "contig_id"),
  cell_pk = "barcode",
  ...
)
```

**Arguments**

contig_tbl	a data frame of contigs, and additional fields describing their properties
contig_pk	character vector naming fields in contig_tbl that uniquely identify a row/contig
cell_tbl	a data frame of cell barcodes, and (optional) additional fields describing their properties
cell_pk	character vector naming fields in cell_tbl that uniquely identify a cell barcode
cluster_tbl	A data frame that provide cluster assignments for each contig
cluster_pk	If cluster_tbl was provided, a character vector naming fields in cluster_tbl that uniquely identify a cluster
equalize	logical. Should the contig, cells and clusters be equalized by taking the intersection of their common keys?
...	passed to <a href="#">ContigCellDB()</a>

**Value**

ContigCellDB

**Functions**

- [ContigCellDB\\_10XVDJ](#): provide defaults that correspond to identifiers in 10X VDJ data

**Accessors/mutators**

See [\\$,ContigCellDB-method](#) for more on how to access and mutate slots. See [mutate\\_cdb\(\)](#) and [filter\\_cdb\(\)](#) for endomorphic filtering/mutation methods. See [split\\_cdb\(\)](#) to split into a list, and [rbind.ContigCellDB\(\)](#) for the inverse operation.

**See Also**

[\\$,ContigCellDB-method](#)

**Examples**

```
data(contigs_qc)
contigs_qc

cdb = ContigCellDB(contigs_qc, contig_pk = c('barcode', 'pop', 'sample', 'contig_id'),
  cell_pk = c('barcode', 'pop', 'sample'))
cdb

# everything that was in contigs_qc
cdb$contig_tbl

# Only the cell_pk are included by default (until clustering/canonicalization)
cdb$cell_tbl

# Empty, since no cluster_pk was specified
cdb$cluster_tbl

# Keys
cdb$contig_pk
cdb$cell_pk
cdb$cluster_pk
```

---

 contigs\_qc

*Filtered and annotated contigs of TCR from mice*


---

### Description

Data for c57bl6 and balbc mice TCR were downloaded from 10x Genomics website as shown in `system.file('script/10XMouseTCR_v3_chem.R', package = 'CellaRepertorium')`. Additional processing of these data is done in the vignette `mouse_tcell_qc` and are serialized to serve as an examples for other vignettes and documentation.

### Usage

```
data(contigs_qc)
```

### Format

A data frame of 3399 contigs and 22 fields, all except 4 are originally defined in <https://support.10xgenomics.com/single-cell-vdj/software/pipelines/latest/output/annotation#contig>. The following fields were defined ex post facto.

1. anno\_file: Path to original csv file
2. pop: Mouse strain.
3. sample: An artificial "replicate" from the original data defined by subsampling with replacement
4. celltype: The putative cell type of the contig.

---

 crosstab\_by\_celltype

*Count contig UMIs by celltype*


---

### Description

Count contig UMIs by celltype

### Usage

```
crosstab_by_celltype(ccdb)
```

### Arguments

ccdb                    A ContigCellDB object

### Value

a table, keyed by cell\_pk counting UMIs per celltype

### See Also

[guess\\_celltype\(\)](#)



**Examples**

```
data(ccdb_ex)
nrow(ccdb_ex$cell_tbl)
total_umi = crosstab_by_celltype(ccdb_ex)
nrow(total_umi)
```

---

entropy	<i>Calculate the entropy of a vector</i>
---------	--

---

**Description**

Calculate the entropy of a vector

**Usage**

```
entropy(v, pseudo_count = length(v)/1000, na.action = na.fail)

np(v, p = 0.05, pseudo_count = p/5, na.action = na.fail)

modal_category(v, na.action = na.fail)
```

**Arguments**

v	categorical vector
pseudo_count	number of pseudo counts to add on, to stabilize empty categories
na.action	how to handle NA values
p	proportion threshold

**Value**

the sample entropy

**Functions**

- np: The number of categories exceeding p proportion of the total
- modal\_category: The modal category of v. Ties are broken by lexicographic order of the factor levels.

**Examples**

```
v2 = gl(2, 4)
v4 = gl(4, 4)
stopifnot(entropy(v2) < entropy(v4))
v_empty = v2[1:4] #empty level 2
stopifnot(is.finite(entropy(v_empty))) # pseudo_count

np(v4, p = .2, pseudo_count = 0)
np(v4, p = .25, pseudo_count = 0)
np(v4, p = .25, pseudo_count = .0001)

modal_category(v4)
modal_category(v4[-1])
```

---

equalize_ccdb	<i>Take the intersection of keys in tables in x</i>
---------------	---

---

### Description

The cells in `cell_tbl`, and clusters in `cluster_tbl` can potentially be a superset of the `contig_tbl`.

### Usage

```
equalize_ccdb(x, cell = TRUE, contig = TRUE, cluster = TRUE, sort = FALSE)
```

### Arguments

<code>x</code>	<code>ContigCellDB()</code>
<code>cell</code>	logical equalize cells
<code>contig</code>	logical equalize contigs
<code>cluster</code>	logical equalize clusters
<code>sort</code>	logical should equalized fields also be <code>order()</code> ed by their primary keys?

### Details

- `equalize_ccdb(x, cell = TRUE)` trims cells that aren't in `contig_tbl` or `cluster_tbl`.
- `equalize_ccdb(x, cluster = TRUE)` trims clusters that aren't in `contig_tbl`.
- `equalize_ccdb(x, contig = TRUE)` trims contigs that aren't `cell_tbl` or `cluster_tbl`.

### Value

`ContigCellDB()`

### Default equalization

Modification to `contig_tbl` (with `$`) always equalizes contigs and clusters. Modification to `cell_tbl` equalizes only contigs. Modification to `cluster_tbl` equalizes contigs and clusters.

### Examples

```
library(dplyr)
tbl = tibble(clust_idx = gl(3, 2), cell_idx = rep(1:3, times = 2), contig_idx = 1:6)
ccdb = ContigCellDB(tbl, contig_pk = c('cell_idx', 'contig_idx'),
  cell_pk = 'cell_idx', cluster_pk = 'clust_idx')
# 3 cells
ccdb
ccdb$cell_tbl = bind_rows(ccdb$cell_tbl, tibble(cell_idx = 0))
# 4 cells now
ccdb
# 3 cells again
equalize_ccdb(ccdb)
# remove all contigs from cell 1, and one contig from cell 2
ccdb$contig_tbl = ccdb$contig_tbl[-c(1, 2, 4),]
# no changes to cell_tbl yet
ccdb
```

```
# trim cell_tbl to 2 cells, keep all clusters
equalize_ccdb(ccdb, cluster = FALSE)
# trim both cells and clusters
equalize_ccdb(ccdb, cluster = TRUE)
```

---

fancy\_name\_contigs      *Generate a legible name for a series of contigs*

---

### Description

Generate a legible name for a series of contigs

### Usage

```
fancy_name_contigs(contig_tbl, prefix)
```

### Arguments

contig\_tbl      An all\_contig\_annotations.csv file, output from VDJ Cell ranger. Importantly, this should contain columns chain, v\_gene, d\_gene, j\_gene

prefix          an optional prefix added to each contig, eg, possibly a sample id.

### Value

character

### Examples

```
library(dplyr)
contig_anno_path = system.file('extdata', 'all_contig_annotations_balbc_1.csv.xz',
  package = 'CellaRepertorium')
contig_anno = readr::read_csv(contig_anno_path)
contig_anno = contig_anno %>% mutate(fancy_name =
  fancy_name_contigs(., prefix = 'b6_1'))
stopifnot(!any(duplicated(contig_anno$fancy_name)))
```

---

filter\_cdb          *Create new or update existing columns of ContigCellDB tables*

---

### Description

Create new or update existing columns of ContigCellDB tables

### Usage

```
filter_cdb(ccdb, ..., tbl = "contig_tbl")

mutate_cdb(ccdb, ..., tbl = "contig_tbl")
```

**Arguments**

ccdb	<a href="#">ContigCellDB()</a>
...	name and value pair of column that will be updated
tbl	character. One of <code>contig_tbl</code> , <code>cell_tbl</code> or <code>cluster_tbl</code> , naming the table to be updated.

**Value**

ContigCellDB object with updated table

**Functions**

- `filter_cdb`: Filter rows of a table in a ContigCellDB object

**See Also**

[dplyr::mutate\(\)](#)

[dplyr::filter\(\)](#)

**Examples**

```
data(ccdb_ex)
subset_contig = filter_cdb(ccdb_ex,full_length, productive == 'True',
high_confidence, chain != 'Multi', nchar(cdr3) > 5)
subset_cell = filter_cdb(ccdb_ex, sample == 4, tbl = 'cell_tbl')
data(ccdb_ex)
new_contig = mutate_cdb(ccdb_ex, new_col = 1)
new_cell = mutate_cdb(ccdb_ex, new_col = 1, tbl = 'contig_tbl')
```

---

`fine_clustering`

*Perform additional clustering of sequences within groups*

---

**Description**

Perform additional clustering of sequences within groups

**Usage**

```
fine_clustering(
  ccdb,
  sequence_key,
  type,
  max_affinity = NULL,
  keep_clustering_details = FALSE,
  ...
)
```

**Arguments**

ccdb	A <code>ContigCellDB()</code> object
sequence_key	character naming column in <code>contig_tbl</code> with sequence
type	'AA' or 'DNA'
max_affinity	numeric naming the maximal affinity for the sparse affinity matrix that is constructed. Not currently used.
keep_clustering_details	logical – should output of <code>fine_cluster_seqs</code> be kept as a list column
...	Arguments passed on to <code>fine_cluster_seqs</code>
big_memory_brute	attempt to cluster more than 4000 sequences? Clustering is quadratic, so this will take a long time and might exhaust memory
method	one of 'substitutionMatrix' or 'levenshtein'
substitution_matrix	a character vector naming a substitution matrix available in Biostrings, or a substitution matrix itself

**Value**

`ContigCellDB()` object with updated `contig_tbl` and `cluster_tbl`

**Examples**

```
library(dplyr)
data(ccdb_ex)
ccdb_ex_small = ccdb_ex
ccdb_ex_small$cell_tbl = ccdb_ex_small$cell_tbl[1:200,]
ccdb_ex_small = cdhit_ccdb(ccdb_ex_small,
  sequence_key = 'cdr3_nt', type = 'DNA', cluster_name = 'DNA97',
  identity = .965, min_length = 12, G = 1)
ccdb_ex_small = fine_clustering(ccdb_ex_small, sequence_key = 'cdr3_nt', type = 'DNA')

# Canonicalize with the medoid contig is probably what is most common
ccdb_medoid = canonicalize_cluster(ccdb_ex_small)

# But there are other possibilities.
# To pass multiple "AND" filter arguments must use &
ccdb_umi = canonicalize_cluster(ccdb_ex_small,
  contig_filter_args = chain == 'TRA' & length > 500, tie_break_keys = 'umis',
  contig_fields = c('chain', 'length'))
ccdb_umi$cluster_tbl %>% dplyr::select(chain, length) %>% summary()
```

---

<code>fine_cluster_seqs</code>	<i>Calculate distances and perform hierarchical clustering on a set of sequences</i>
--------------------------------	--

---

**Description**

The distances between AA sequences is defined to be  $1 - \text{score} / \max(\text{score})$  times the median length of the input sequences. The distances between nucleotide sequences is defined to be  $\text{edit\_distance} / \max(\text{edit\_distance})$  times the median length of input sequences.

**Usage**

```

fine_cluster_seqs(
  seqs,
  type = "AA",
  big_memory_brute = FALSE,
  method = "levenshtein",
  substitution_matrix = "BLOSUM100",
  cluster_fun = "none",
  cluster_method = "complete"
)

```

**Arguments**

seqs	character vector, DNASTringSet or AAStringSet
type	character either AA or DNA specifying type of seqs
big_memory_brute	attempt to cluster more than 4000 sequences? Clustering is quadratic, so this will take a long time and might exhaust memory
method	one of 'substitutionMatrix' or 'levenshtein'
substitution_matrix	a character vector naming a substitution matrix available in Biostrings, or a substitution matrix itself
cluster_fun	character, one of "hclust" or "none", determining if distance matrices should also be clustered with hclust
cluster_method	character passed to hclust

**Value**

list

**See Also**

[hclust\(\)](#), [Biostrings::stringDist\(\)](#)

**Examples**

```

fasta_path = system.file('extdata', 'demo.fasta', package='CellaRepertorium')
aaseq = Biostrings::readAAStringSet(fasta_path)[1:100]
cls = fine_cluster_seqs(aaseq, cluster_fun = 'hclust')
plot(cls$cluster)

```

---

generate\_pseudobulk     *Generate "pseudobulk" data from a ContigCellDB*

---

**Description**

Tabulate contigs with a unique combination of class\_keys per total\_keys. For instance, total\_keys might be a sample identifier, and class\_keys might be the V- and J- gene identities. The idea is that this might mimic the data generated in a bulk experiment.

**Usage**

```
generate_pseudobulk(ccdb, class_keys, total_keys, type = c("cell", "umi"))
```

**Arguments**

ccdb	<a href="#">ContigCellDB()</a>
class_keys	character naming fields in contig_tbl that define unique classes of the repertoire
total_keys	character naming fields to be conditioned upon when calculating the total.
type	one of "cell" or "umi"

**Details**

This function is currently rather 10x-specific, in that it is assumed that columns barcode and umis exist.

**Value**

tibble

**Examples**

```
data(ccdb_ex)
ccdb_ex = cluster_germline(ccdb_ex)
pseudo = generate_pseudobulk(ccdb_ex, c('v_gene', 'j_gene', 'chain'), c('pop', 'sample'))
```

---

guess\_celltype

*Guess the cell type of a contig from the chain ID*

---

**Description**

This function is likely dependent on annotations from 10X and may change or break as their pipeline changes.

**Usage**

```
guess_celltype(chain)
```

**Arguments**

chain	character which will be parsed to try to infer celltype
-------	---

**Value**

contig table with celltype column

**See Also**

[crosstab\\_by\\_celltype\(\)](#)

**Examples**

```
data(ccdb_ex)
table(guess_celltype(ccdb_ex$contig_tbl$chain))
```

---

hushWarning	<i>Selectively muffle warnings based on output</i>
-------------	--

---

**Description**

Selectively muffle warnings based on output

**Usage**

```
hushWarning(expr, regexp)
```

**Arguments**

expr	an expression
regexp	a regexp to be matched (with str_detect)

**Value**

the result of expr

**Examples**

```
CellaRepertorium::hushWarning(warning('Beware the rabbit'), 'rabbit')
CellaRepertorium::hushWarning(warning('Beware the rabbit'), 'hedgehog')
```

---

ig_chain_recode	<i>Categorize the pairing present in a cell</i>
-----------------	---

---

**Description**

For each cell (defined by `ccdb$cell_pk`) count the number of each level of `chain_key` occurs, and cross tabulate. Also for each cell, paste together all values `chain_key`. Return a tibble, keyed by cells that includes the counts of the chains, the `raw_chain_type` and any additional output from running `chain_recode_fun`.

**Usage**

```
ig_chain_recode(tbl)

tcr_chain_recode(tbl)

enumerate_pairing(ccdb, chain_key = "chain", chain_recode_fun = NULL)
```



**Arguments**

tbl	output from <code>enumerate_pairing</code> containing TRA/TRB or IGH/IHK/IHL columns
ccdb	ContigCellDB
chain_key	character naming the field in the <code>contig_tbl</code> identifying chain
chain_recode_fun	a function that operates on the output of this function that further reduces the chain combinations to some other summary. Set to 'guess' to apply functions that may work for 10X data or NULL to skip. See <code>CellaRepertorium::tcr_chain_recode</code> for an example.

**Value**

a tibble keyed by cells.

**Functions**

- `ig_chain_recode`: Recode a table with IG chains
- `tcr_chain_recode`: Recode a table with TCR chains

**Examples**

```
data(ccdb_ex)
enumerate_pairing(ccdb_ex)
enumerate_pairing(ccdb_ex, chain_recode_fun = 'guess')
```

---

map_axis_labels	<i>Color axis labels</i>
-----------------	--------------------------

---

**Description**

Color axis labels

**Usage**

```
map_axis_labels(
  plt,
  label_data_x = NULL,
  label_data_y = NULL,
  aes_label,
  scale = ggplot2::scale_color_hue(aesthetics = "axis_color")
)
```

**Arguments**

plt	<code>ggplot2::ggplot()</code> object
label_data_x	<code>data.frame()</code> containing the mapping between x-axis labels and <code>aes_label</code>
label_data_y	<code>data.frame()</code> containing the mapping between y-axis labels and <code>aes_label</code>
aes_label	character or bare symbol giving the column in <code>label_data</code> to be mapped
scale	ggplot2 discrete color

**Value**

plt with axis text modified

**Examples**

```
require(ggplot2)
require(dplyr)
plt = ggplot(mpg, aes(x = manufacturer, y = drv)) + geom_jitter()
label_data = mpg %>% select(manufacturer) %>% unique() %>%
mutate(euro = manufacturer %in% c('audi', 'volkswagen'))
map_axis_labels(plt, label_data_x = label_data, aes_label = euro)
```

---

pairing\_tables

*Generate a list of tables representing clusters paired in cells*

---

**Description**

A contingency table of every combination of cluster\_idx up to table\_order is generated. Combinations that are found in at least min\_expansion number of cells are reported. All cells that have these combinations are returned, as well as cells that only have orphan\_level of matching cluster\_idx.

**Usage**

```
pairing_tables(
  ccdb,
  ranking_key = "grp_rank",
  table_order = 2,
  min_expansion = 2,
  orphan_level = 1,
  cluster_keys = character(),
  cluster_whitelist = NULL,
  cluster_blacklist = NULL
)
```

**Arguments**

ccdb	ContigCellDB
ranking_key	field in ccdb\$contig_tbl giving the ranking of each contig per cell. Probably generated by a call to <a href="#">rank_prevalence_ccdb()</a> or <a href="#">rank_chain_ccdb()</a> .
table_order	Integer larger than 1. What order of cluster_idx will be paired, eg, order = 2 means that the first and second highest ranked contigs will be sought and paired in each cell
min_expansion	the minimal number of times a pairing needs to occur for it to be reported
orphan_level	Integer in interval [1, table_order]. Given that at least min_expansion cells are found that have table_order chains identical, how many cluster_idx pairs will we match on to select other cells. Example: ophan_level=1 means that cells that share just a single chain with an expanded pair will be reported.
cluster_keys	optional character naming additional columns in ccdb\$cluster_tbl to be reported in the pairing

**cluster\_whitelist**

a table of pairings or clusters that should always be reported. Here the clusters must be named "cluster\_idx.1", "cluster\_idx.2" (if order-2 pairs are being selected) rather than with 'ccdb\$cluster\_pk'

**cluster\_blacklist**

a table of pairings or clusters that will never be reported. Must be named as per cluster\_whitelist.

**Details**

For example, if table\_order=2 and min\_expansion=2 then heavy/light or alpha/beta pairs found two or more times will be returned (as well as alpha-alpha pairs, etc, if those are present). If orphan\_level=1 then all cells that share just a single chain with an expanded clone will be returned.

The cluster\_idx.1\_fct and cluster\_idx.2\_fct fields in cell\_tbl, idx1\_tbl, idx2\_tbl are cast to factors and ordered such that pairings will tend to occur along the diagonal when they are cross-tabulated. This facilitates plotting.

**Value**

list of tables. The cell\_tbl is keyed by the cell\_identifiers, with fields "cluster\_idx.1", "cluster\_idx.2", etc, IDing the contigs present in each cell. "cluster\_idx.1\_fct" and "cluster\_idx.2\_fct" cast these fields to factors and are reordered to maximize the number of pairs along the diagonal. The idx1\_tbl and idx2\_tbl report information (passed in about the cluster\_idx by feature\_tbl.) The cluster\_pair\_tbl reports all pairings found of contigs, and the number of times observed.

**See Also**

[rank\\_prevalence\\_ccdb\(\)](#)

**Examples**

```
library(dplyr)
tbl = tibble(clust_idx = gl(3, 2), cell_idx = rep(1:3, times = 2), contig_idx = 1:6)
ccdb = ContigCellDB(tbl, contig_pk = c('cell_idx', 'contig_idx'),
cell_pk = 'cell_idx', cluster_pk = 'clust_idx')
# add `grp_rank` to ccdb$contig_tbl indicating how frequent a cluster is
ccdb = rank_prevalence_ccdb(ccdb, tie_break_keys = character())
# using `grp_rank` to determine pairing
# no pairs found twice
pt1 = pairing_tables(ccdb)
# all pairs found, found once.
pt2 = pairing_tables(ccdb, min_expansion = 1)
pt2$cell_tbl
tbl2 = bind_rows(tbl, tbl %>% mutate(cell_idx = rep(4:6, times = 2)))
ccdb2 = ContigCellDB(tbl2, contig_pk = c('cell_idx', 'contig_idx'), cell_pk = 'cell_idx',
cluster_pk = 'clust_idx') %>% rank_prevalence_ccdb(tie_break_keys = character())
#all pairs found twice
pt3 = pairing_tables(ccdb2, min_expansion = 1)
pt3$cell_tbl
ccdb2$contig_tbl = ccdb2$contig_tbl %>%
  mutate(umis = 1, reads = 1, chain = rep(c('TRA', 'TRB'), times = 6))
ccdb2 = rank_chain_ccdb(ccdb2, tie_break_keys = character())
pt4 = pairing_tables(ccdb2, min_expansion = 1, table_order = 2)
```

---

purity	<i>Calculate number of cluster-subject singletons for the purposes of permutation testing</i>
--------	---

---

**Description**

Calculate number of cluster-subject singletons for the purposes of permutation testing

**Usage**

```
purity(cluster_idx, subject)
```

**Arguments**

cluster_idx	factor-like cluster variable
subject	factor-like subject

**Value**

average number of singletons

**See Also**

[cluster\\_permute\\_test\(\)](#)

**Examples**

```
message("see example(cluster_permute_test)")
```

---

rank_prevalence_ccdb	<i>Rank contigs, per cell, by experiment-wide prevalence of cluster_pk, which is added as the prevalence field</i>
----------------------	--

---

**Description**

Rank contigs, per cell, by experiment-wide prevalence of cluster\_pk, which is added as the prevalence field

**Usage**

```
rank_prevalence_ccdb(
  ccdb,
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads")
)

rank_chain_ccdb(
  ccdb,
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads"),
```

```

chain_key = "chain",
contig_fields = tie_break_keys,
chain_levels = c("IGL", "IGK", "TRA", "TRB", "IGH")
)

```

### Arguments

`ccdb` [ContigCellDB\(\)](#)

`contig_filter_args` an expression passed to `dplyr::filter()`. Unlike `filter`, multiple criteria must be & together, rather than using commas to separate. These act on `ccdb$contig_tbl`

`tie_break_keys` (optional) character naming fields in `contig_tbl` that are used sort the contig table in descending order. Used to break ties if `contig_filter_args` does not return a unique contig for each cluster

`chain_key` character naming the field in `contig_tbl` to be sorted on.

`contig_fields` Optional fields from `contig_tbl` that will be copied into the `cluster_tbl` from the canonical contig.

`chain_levels` an optional character vector providing the sort order of the chain column in `tbl`. If set to length zero, then the the ordering will be alphabetical

### Value

ContigCellDB with modified `contig_tbl`

### Functions

- `rank_chain_ccdb`: return a canonical contig by chain type, with TRB/IGH returned first. By default, ties are broken by `umis` and `reads`.

### Examples

```

data(ccdb_ex)
ccdb_ex = cluster_germline(ccdb_ex)
rank_prev = rank_prevalence_ccdb(ccdb_ex)
rank_prev$contig_tbl
rank_chain = rank_chain_ccdb(ccdb_ex)
rank_chain$contig_tbl

```

---

rbind,ContigCellDB-method

*Combine ContigCellDB along rows (contigs, cells or clusters).*

---

### Description

The union of the rows in each of the objects is taken, thus removing any rows that has an exact duplicate. This includes all fields, not just the primary key for that table. The union of the various primary keys is taken.

**Usage**

```
## S4 method for signature 'ContigCellDB'
rbind(..., deparse.level = 1)
```

**Arguments**

```
...          ContigCellDB\(\)
deparse.level ignored
```

**Value**

```
ContigCellDB\(\)
```

**Examples**

```
data(ccdb_ex)
splat = split_cdb(ccdb_ex, 'chain', 'contig_tbl')
unite = equalize_ccdb(rbind(splat$TRA, splat$TRB), sort = TRUE)
stopifnot(all.equal(unite, ccdb_ex))
```

---

right_join_warn	<i>Perform a <code>dplyr::left_join()</code> but check for non-key overlapping fields</i>
-----------------	---

---

**Description**

Perform a `dplyr` join, but either warn if the two tables share non-key fields. If `overwrite = TRUE`, then shared columns will pull from `x` otherwise a suffix will be added to `y`. To perform this check, `by` must be specified, and it is an error if it is not.

**Usage**

```
right_join_warn(...)

left_join_warn(x, y, by, overwrite = FALSE, join = left_join, ...)
```

**Arguments**

```
...          passed to joining function
x            A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames
             (e.g. from dbplyr or dtplyr). See Methods, below, for more details.
y            A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames
             (e.g. from dbplyr or dtplyr). See Methods, below, for more details.
by           character specifying columns in x and y to key on.
overwrite    logical – should non-key fields in y be overwritten using x, or should a suffix
             (".y") be added
join         function giving the type of join to perform, eg, left, right, inner, outer.
```

**Value**

data.frame or tibble

**Functions**

- `right_join_warn`: perform a `dplyr::right_join()`

**Examples**

```
left_join_warn(mtcars, mtcars, by = 'mpg')
left_join_warn(mtcars, mtcars, by = 'mpg', overwrite = TRUE)
```

---

split\_cdb

*Split into a list of [ContigCellDB\(\)](#) by named fields*


---

**Description**

Split into a list of [ContigCellDB\(\)](#) by named fields

**Usage**

```
split_cdb(ccdb, fields, tbl = "contig_tbl", drop = FALSE, equalize = TRUE)
```

**Arguments**

<code>ccdb</code>	<a href="#">ContigCellDB()</a>
<code>fields</code>	character naming fields in <code>tbl</code>
<code>tbl</code>	one of <code>contig_tbl</code> , <code>cell_tbl</code> or <code>cluster_tbl</code>
<code>drop</code>	logical indicating if levels that do not occur should be dropped (if <code>f</code> is a factor or a list).
<code>equalize</code>	logical. Should the <code>contig</code> , <code>cells</code> and <code>clusters</code> be equalized by taking the intersection of their common keys?

**Value**

list of `ContigCellDB`

**Examples**

```
data(ccdb_ex)
splat = split_cdb(ccdb_ex, 'chain', 'contig_tbl')
stopifnot(all(splat$TRA$contig_tbl$chain == 'TRA'))
stopifnot(all(splat$TRB$contig_tbl$chain == 'TRB'))
```

---

```
[[,ContigCellDB,character,missing-method
      data.frame-like mutation/accessor generics for ContigCellDB ob-
      jects
```

---

### Description

A ContigCellDB pretend to be a cell\_tbl data.frame in several regards. This is to enable nesting ContigCellDB objects in the colData of a SingleCellExperiment and so that various plotting functionality in scater can do something sensible.

### Usage

```
## S4 method for signature 'ContigCellDB,character,missing'
x[[i, j, ...]]

## S4 method for signature 'ContigCellDB,ANY,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ContigCellDB'
dim(x)

## S4 method for signature 'ContigCellDB'
dimnames(x)

## S4 method for signature 'ContigCellDB'
nrow(x)

## S4 method for signature 'ContigCellDB'
ncol(x)
```

### Arguments

x	ContigCellDB
i	integer or character index
j	ignored
...	ignored
drop	ignored

### Details

If x a ContigCellDB, then dim(x) and dimnames(x) return dim(x\$cell\_tbl) and dimnames(x\$cell\_tbl), respectively, and x[[col]] returns x\$cell\_tbl[[col]]. Likewise indexing with x[i,] returns cells indexed by i. Finally as.data.frame(x) returns x\$cell\_tbl.

### Value

See details.



**Examples**

```

data(ccdb_ex)
ccdb_ex[1:10,]
head(ccdb_ex[['barcode']])
dim(ccdb_ex)
dimnames(ccdb_ex)

```

---

\$,ContigCellDB-method *Access public members of ContigCellDB object.*

---

**Description**

Modification to members will trigger various forms of equalization. See [equalize\\_ccdb\(\)](#) for details.

**Usage**

```

## S4 method for signature 'ContigCellDB'
x$name

## S4 replacement method for signature 'ContigCellDB'
x$name <- value

```

**Arguments**

x	A ContigCellDB object
name	a slot of a ContigCellDB object (one of c('contig_tbl', 'cell_tbl', 'contig_pk', 'cell_pk', 'pop'))
value	The value assigned to a slot of ContigCellDB object

**Value**

Update or return a slot of [ContigCellDB\(\)](#)

**See Also**

[equalize\\_ccdb\(\)](#)

**Examples**

```

data(ccdb_ex)
ccdb_ex$contig_tbl
ccdb_ex$cell_tbl
ccdb_ex$cluster_tbl
data(ccdb_ex)
ccdb_ex$contig_pk = c("sample", "barcode", "contig_id") # 'pop' is technically redundant with 'sample'
# Take a subset of ccdb_ex
ccdb_ex
ccdb_ex$contig_tbl = dplyr::filter(ccdb_ex$contig_tbl, pop == 'b6')
ccdb_ex

```

# Index

- \* **datasets**
  - ccdb\_ex, 6
  - contigs\_qc, 16
  - .cluster\_permute\_test, 3
  - [,ContigCellDB,ANY,missing,ANY-method
    - ([[,ContigCellDB,character,missing-method), 32
  - [,ContigCellDB,ANY,missing-method
    - ([[,ContigCellDB,character,missing-method), 32
  - [[,ContigCellDB,character,missing-method, 32
  - \$,ContigCellDB-method, 33
  - \$<-,ContigCellDB-method
    - (\$,ContigCellDB-method), 33
- Biostrings::stringDist(), 22
- canonicalize\_cell, 3
- canonicalize\_cell(), 5
- canonicalize\_cluster, 5
- canonicalize\_cluster(), 4
- ccdb\_ex, 6
- ccdb\_join, 6
- cdhit, 7, 9
- cdhit(), 8, 9
- cdhit\_ccdb, 8
- cluster\_filterset, 9
- cluster\_filterset(), 13
- cluster\_germline, 10
- cluster\_logistic\_test
  - (cluster\_test\_by), 13
- cluster\_permute\_test, 11
- cluster\_permute\_test(), 28
- cluster\_plot, 12
- cluster\_test\_by, 13
- ContigCellDB, 14
- ContigCellDB(), 4, 5, 7–10, 13, 15, 18, 20, 21, 23, 29–31, 33
- ContigCellDB-mutate
  - (\$,ContigCellDB-method), 33
- ContigCellDB\_10XVDJ(ContigCellDB), 14
- contigs\_qc, 6, 16
- crosstab\_by\_celltype, 16
- crosstab\_by\_celltype(), 23
- data.frame(), 25
- dim,ContigCellDB-method
  - ([[,ContigCellDB,character,missing-method), 32
- dimnames,ContigCellDB-method
  - ([[,ContigCellDB,character,missing-method), 32
- dplyr::filter(), 4, 5, 13, 20, 29
- dplyr::mutate(), 20
- entropy, 17
- enumerate\_pairing(ig\_chain\_recode), 24
- equalize\_ccdb, 18
- equalize\_ccdb(), 33
- fancy\_name\_contigs, 19
- filter\_cdb, 19
- filter\_cdb(), 15
- fine\_cluster\_seqs, 21, 21
- fine\_clustering, 20
- generate\_pseudobulk, 22
- ggplot2::ggplot(), 25
- guess\_celltype, 23
- guess\_celltype(), 16
- hclust(), 22
- hushWarning, 24
- ig\_chain\_recode, 24
- left\_join\_warn(right\_join\_warn), 30
- left\_join\_warn(), 5
- map\_axis\_labels, 25
- modal\_category(entropy), 17
- mutate\_cdb(filter\_cdb), 19
- mutate\_cdb(), 15
- ncol,ContigCellDB-method
  - ([[,ContigCellDB,character,missing-method), 32
- np(entropy), 17

nrow, ContigCellDB-method  
([, ContigCellDB, character, missing-method),  
32

order(), 18

pairing\_tables, 26

purity, 28

purity(), 11

rank\_chain\_ccdb(rank\_prevalence\_ccdb),  
28

rank\_chain\_ccdb(), 26

rank\_prevalence\_ccdb, 28

rank\_prevalence\_ccdb(), 26, 27

rbind, ContigCellDB-method, 29

rbind.ContigCellDB  
(rbind, ContigCellDB-method), 29

rbind.ContigCellDB(), 15

right\_join\_warn, 30

split\_cdb, 31

split\_cdb(), 15

tcr\_chain\_recode(ig\_chain\_recode), 24