Infinitely Scalable Data Analysis

[tile]DB

Using R with TileDB

Aaron Wolen and Dirk Eddelbuettel

BioC2021 — 8 August 2021

Overview

Outline

Brief Introduction

Some Key Topics

- Dense Arrays
- Sparse Arrays
- Full TileDB API
- S3 Access
- Arrow Format

Genomic Applications

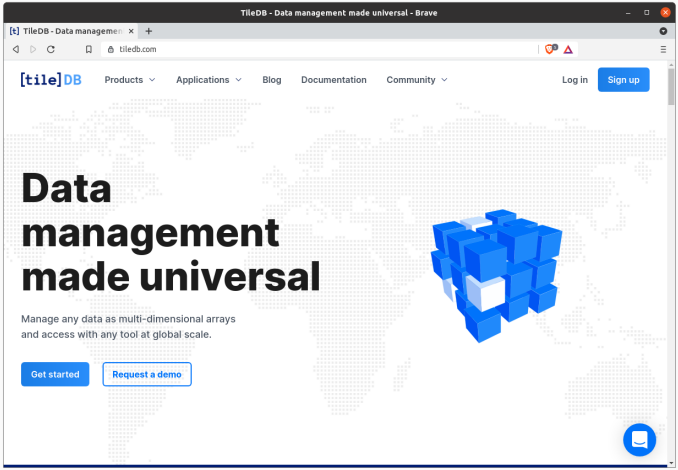
- Annotations
- Matrices
- Ranges

Wrap-Up

Further References

Introduction

TileDB



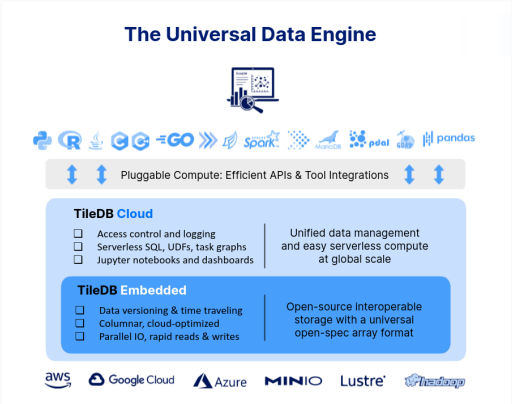
Universal Data Management

Support Any Data in Multi-Dimensional Arrays

Serverless, and at Global Scale

Multiple Languages – In this Talk with an R focus

Architected for Scalability



Key Features

Native cloud object stores access

No extra software/frameworks required

Unlimited storage & massive bandwidth

Read/write queries are fully parallelized and thread- / process-safe

Multi-dimensional indexing

Query times grow with *result* not data size

Tutorial Resources (initially for useR! 2021)

To install the package with code examples and the slides, use

```
remotes::install_github('eddelbuettel/tiledb-user2021')
## or alternatively
repos <- c("https://eddelbuettel.github.io/tiledb-user2021/",
           "https://cloud.r-project.org")
install.packages("tiledb.user2021", repos=repos)
```

Loading the package will show where the example files are located.

Introductory Example

```
# if needed: install.packages("tiledb")     # installation from CRAN
library(tiledb)                             # load the package
library(palmerpenguins)                     # example data
setwd("/tmp")                               # or other scratch space

# create array from data frame with default settings
fromDataFrame(penguins, "penguins")

# read array as data.frame and without (default, added) row index
arr <- tiledb_array("penguins", as.data.frame=TRUE, extended=FALSE)
show(arr)                                   # some array information
```

Introductory Example (cont.)

```
> df <- arr[]
> str(df)
'data.frame':   344 obs. of  8 variables:
 $ species          : chr  "Adelie" "Adelie" "Adelie" "Adelie" ...
 $ island           : chr  "Torgersen" "Torgersen" "Torgersen" "Torgersen" ...
 $ bill_length_mm   : num  39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
 $ bill_depth_mm    : num  18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
 $ flipper_length_mm: int  181 186 195 NA 193 190 181 195 193 190 ...
 $ body_mass_g      : int  3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
 $ sex              : chr  "male" "female" "female" NA ...
 $ year             : int  2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
>
```

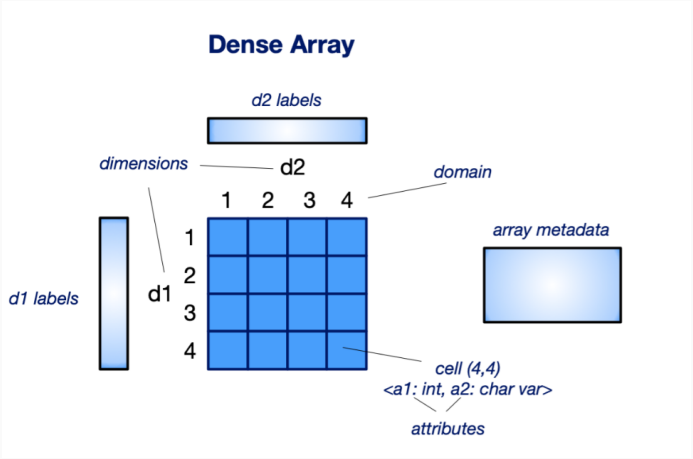
Introductory Example (cont.)

Key Features

- We will discuss available options to create arrays
  - dense arrays versus sparse arrays
  - one or multiple indices (on sparse arrays)
  - options for creating and accessing arrays
  - but we mention tuning (tile extent, tile layout, ...) only in passing
- We will show different ways to read arrays back into R

Dense Arrays

Dense Arrays



- All cells filled
- Homogeneous dimensions
- One or two or … dimensions
- Cells can containing multiple values
- Possibly more performant

Dense Data

The introductory example `quickstart_dense.R` creates an array with two integer domains and a single integer attribute:

```
# The array will be 4x4 with dims "rows" and "cols" and domain [1,4]
dom <- tiledb_domain(dims = c(tiledb_dim("rows", c(1L, 4L), 4L, "INT32"),
                              tiledb_dim("cols", c(1L, 4L), 4L, "INT32")))
# The array will be dense with a single attribute "a" so
# each cell (i,j) cell can store an integer.
schema <- tiledb_array_schema(dom, attrs=c(tiledb_attr("a", type="INT32")))
# Create the (empty) array on disk.
uri <- "quickstart_dense"
tiledb_array_create(uri, schema)
```

Dense Data (cont.)

Having created the array we can now open it for writing and add data.

```
# equivalent to matrix(1:16, 4, 4, byrow=TRUE)
data <- array(c(c(1L, 5L, 9L, 13L),
                c(2L, 6L, 10L, 14L),
                c(3L, 7L, 11L, 15L),
                c(4L, 8L, 12L, 16L)), dim = c(4,4))
# Open the array and write to it.
A <- tiledb_array(uri = uri)
A[] <- data
```

Dense Data (cont.)

Data can be read back with different convenience wrappers:

```
arr <- tiledb_array(uri); arr[]                       # list of columns

arr <- tiledb_array(uri, as.data.frame=TRUE); arr[] # a data.frame

arr <- tiledb_array(uri, as.matrix=TRUE); arr[]     # a matrix

arr <- tiledb_array(uri, as.array=TRUE); arr[]      # an array
```

Dense Data (cont.)

A data.frame example for dense arrays:

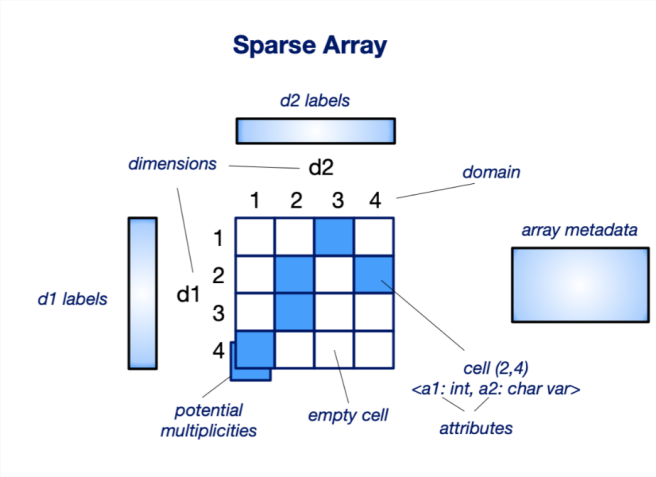
```
library(tiledb)           # load our package
uri <- tempfile()         # any local directory, more later on cloud access

## any data.frame, data.table, tibble ...; here we use penguins_raw
fromDataFrame(palmerpenguins::penguins_raw,  uri)

# we want a data.frame, and we skip the implicit row numbers added as index
x <- tiledb_array(uri, as.data.frame = TRUE, extended = FALSE)

newdf <- x[]              # full array (we can index rows and/or cols too)
```

Dense Data (cont.)

```
> str(newdf[, 1:14])   # omitting last three cols for brevity
'data.frame':   344 obs. of  17 variables:
 $ studyName          : chr  "PAL0708" "PAL0708" "PAL0708" "PAL0708" ...
 $ Sample Number      : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Species            : chr  "Adelie Penguin (Pygoscelis adeliae)"  ...
 $ Region             : chr  "Anvers" "Anvers" "Anvers" "Anvers" ...
 $ Island             : chr  "Torgersen" "Torgersen" "Torgersen" "Torgersen" ...
 $ Stage              : chr  "Adult, 1 Egg Stage" "Adult, 1 Egg Stage" ...
 $ Individual ID      : chr  "N1A1" "N1A2" "N2A1" "N2A2" ...
 $ Clutch Completion  : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ Date Egg           : Date, format: "2007-11-11" ...
 $ Culmen Length (mm) : num  39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
 $ Culmen Depth (mm)  : num  18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
 $ Flipper Length (mm): num  181 186 195 NA 193 190 181 195 193 190 ...
 $ Body Mass (g)      : num  3750 3800 3250 NA 3450 ...
 $ Sex                : chr  "MALE" "FEMALE" "FEMALE" NA ...
```

Sparse Arrays

Sparse Array



(Possibly many) cells empty

Heterogeneous dimension

Multiple cells per dimension tuple possible

Very flexible

Maps very well to data frame objects

Sparse Array: Numeric

```
library(tiledb)                         # TileDB package
library(Matrix)                         # for sparse matrix functionality
uri <- tempfile()                       # array location
set.seed(123)                           # fix RNG seed

mat <- matrix(0, nrow=8, ncol=20)
mat[sample(seq_len(8*20), 15)] <- seq(1, 15)
spmat <- as(mat, "dgTMatrix")           # new sparse 'dgTMatrix'

fromSparseMatrix(spmat, uri)            # store the sparse matrix in TileDB
chk <- toSparseMatrix(uri)              # and retrieve it to check
```

Sparse Array: Numeric (cont.)

```
> chk     # to check retrieved sparse matrix
8 x 20 sparse Matrix of class "dgTMatrix"

[1,] . . . . . . . . . . . . . . . . . . 13 . 8
[2,] . . . . . . 3 . . . . 9 . . . . . . . . .
[3,] . . . . . 5 . . . . . 10 14 . . . . . . .
[4,] . . . . . . . . . . . 12 . . . . . . . .
[5,] . . . . . . . . . . . . . . . . . . . . 7
[6,] . 2 . . . . . . . . . . . 4 . . . . . .
[7,] . . . . . . . . . . . . . . . . . . 11 1
[8,] . . . . . . . . 15 . . . . . . . . . . 6
>
```

Sparse Array: Data Frame

```
library(tiledb)          # load our package
uri <- tempfile()        # any local directory, more later on cloud access
## now sparse with a character and integer ('year') index colum
## with wider range than seen in the data for year we allow appending
fromDataFrame(palmerpenguins::penguins,  uri, sparse = TRUE,
              col_index = c("species", "year"),
              tile_domain=list(year=c(2000L, 2021L)))

x <- tiledb_array(uri, as.data.frame = TRUE, extended = FALSE)
newdf <- x[]             # full array (we can index rows and/or cols too)
```

Sparse Array: Data Frame (cont.)

```
x <- tiledb_array(uri, as.data.frame = TRUE, extended = FALSE)
selected_ranges(x) <- list(year=cbind(2007L, 2008L),
                           species=cbind("Gentoo", "Gentoo"))
newdf <- x[]
```

Now we retrieve with two constraints: ‘years’ from 2007 to 2008 (both included), and ‘species’ equal to “Gentoo” (given as lower and upper range which implies equality). Note that both are *dimension* columns.

Sparse Array: Data Frame (cont.)

```
qc <- tiledb_query_condition_init("body_mass_g", 6000, "INT32", "GE")
query_condition(x) <- qc
newdf <- x[]
```

This selects rows based on the given attribute value, here `body_mass_g` which is required to be greater or equal to 6000 (grams).

Query conditions on attributes can also be combined (via standard Boolean operators) and parsed. String column support is in the GitHub version.

```
qc <- parse_query_condition(body_mass_g >= 6000 && sex == 'male')
```

Sparse Array: Select Attribute Columns

```
x <- tiledb_array(uri, as.data.frame = TRUE, extended = FALSE)
attrs(x) <- c("island", "sex")
```

This results in just the two selected attribute columns being returned (along with the two dimension columns).

Column selections can be combined with row selections.

Sparse Array: Incremental Writes

Setting the initial *domain* of the dimension columns (to ranges that accomodate future writes) allows incremental writes in batches.

As TileDB is serverless and inherently parallel, multiple writes can be made at the same time.

fromDataFrame & tiledb_array

fromDataFrame

High-level Array Writer

- Helper function to *create* arrays from existing data.frame data in R
- Can write dense arrays as well as sparse arrays
  - can add ad-hoc row-indices (dense and sparse)
  - or can use multiple index colums (sparse)
  - these can use int, numeric, or char data
- Defaults to using a ZStd compression filter
- Can set different TileDB array attributes and parameters
- Can support *append* mode via explicit dimension domain values
- We will see some examples later

tiledb_array

High-level Array Reader

- General array accessor for both dense and sparse arrays
- Supports multiple options to return as
  - data.frame
  - matrix
  - array
- Supports selection of row ranges (via dimension constraint)
- Supports selection of returned columns

AWS S3

S3

```
uri <- "s3://namespace/bucket"          # change URI as needed

## you need either these two environment variables
##   AWS_SECRET_ACCESS_KEY
##   AWS_ACCESS_KEY_ID
## or set this in the TileDB config object

fromSparseMatrix(spmat, uri)     # stored
chk <- toSparseMatrix(uri)       # retrieved

## lazy eval: e.g. for subsets only requested data transferred to client
```

S3 (cont.)

```
> pp <- tiledb_array("s3://tiledb-conferences/useR-2021/palmer_penguins", as.data.frame=TRUE)
> dat <- pp[]
> head(dat)
  species year    island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g    sex
1  Adelie 2007     Dream           36.0          17.9               190        3450 female
2  Adelie 2007     Dream           42.3          21.2               191        4150   male
3  Adelie 2007 Torgersen           40.3          18.0               195        3250 female
4  Adelie 2007 Torgersen           34.6          21.1               198        4400   male
5  Adelie 2007 Torgersen           36.6          17.8               185        3700 female
6  Adelie 2007 Torgersen           36.7          19.3               193        3450 female
>
```

Arrow

Arrow

```
suppressMessages( { library(tiledb); library(arrow) } )
val <- 1:3     # arbitrary, could be rnorm() too
typ <- int8()  # any Arrow type
vec <- Array$create(val, typ)           # Arrow vector

aa <- tiledb_arrow_array_ptr()
as <- tiledb_arrow_schema_ptr()
on.exit( { tiledb_arrow_array_del(aa); tiledb_arrow_schema_del(as) } )
arrow:::ExportArray(vec, aa, as)  # export Arrow to TileDB

newvec <- arrow::Array$create(arrow:::ImportArray(aa, as))
stopifnot(all.equal(vec, newvec))
print(newvec)   # show round-turn
```

Arrow (cont.)

```
> print(newvec)   # show round-turn
Array
<int8>
[
  1,
  2,
  3
]
>
```

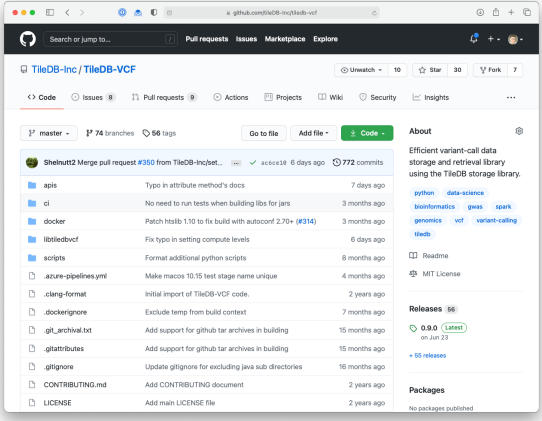
Additional examples demonstrate zero-copy transfer from Arrow into TileDB Arrays, and the inverse from TileDB to Arrow.

Additional higher-level functions will likely get added soon.

TileDB + Genomics

TileDB-VCF

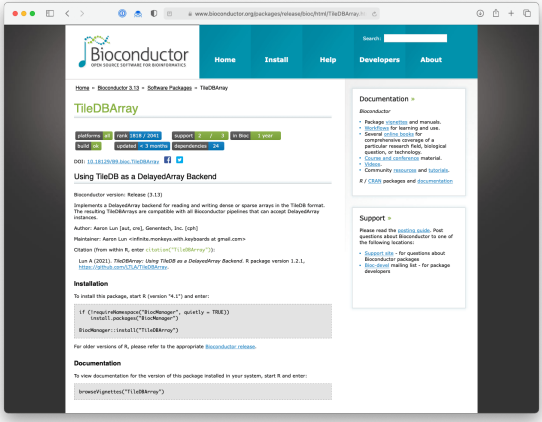
A free and open source application for efficient/scalable storage and retrieval of genomic variant-call data.



https://github.com/TileDB-Inc/TileDB-VCF

TileDBArray Bioconductor Package

Implements a DelayedArray backend for reading and writing dense or sparse arrays in the TileDB format. The resulting TileDBArrays are compatible with all Bioconductor pipelines that can accept DelayedArray instances.



Lun A (2021). *TileDBArray* version 1.2.1, https://github.com/LTLA/TileDBArray.

Genomics Use Case Examples

1. Genomic Annotations
2. 2D Matrices
3. Genomic Ranges

Genomic Annotations

Ensembl Gene Annotation

Example: Store Ensembl annotations for gene, transcripts, exons, *etc*.

- Original format: Parsed GTF file with the latest *Homo sapien* Ensembl annotations courtesy of `AnnotationHub` (❤)
- Array: Sparse array with a single dimension for the relevant type's ID (e.g., `gene_id`s)
- Typical query: Retrieve genomic locations for a particular gene or group of genes

Ensembl Genomic Annotations Data

```
# A tibble: 3,413,874 x 18
   gene_id      chrom pos_start  pos_end type       gene_name gene_source  gene_biotype
   <chr>        <chr>     <int>    <int> <chr>      <chr>     <chr>        <chr>
 1 ENSG00000000003 X     100636695   1.01e8 five_prim… TSPAN6    ensembl_hav… protein_codi…
 2 ENSG00000000003 X     100633405   1.01e8 CDS        TSPAN6    ensembl_hav… protein_codi…
 3 ENSG00000000003 X     100627108   1.01e8 gene       TSPAN6    ensembl_hav… protein_codi…
 4 ENSG00000000003 X     100627108   1.01e8 transcript TSPAN6    ensembl_hav… protein_codi…
 5 ENSG00000000003 X     100636608   1.01e8 exon       TSPAN6    ensembl_hav… protein_codi…
 6 ENSG00000000003 X     100636608   1.01e8 CDS        TSPAN6    ensembl_hav… protein_codi…
 7 ENSG00000000003 X     100636692   1.01e8 start_cod… TSPAN6    ensembl_hav… protein_codi…
 8 ENSG00000000003 X     100635558   1.01e8 exon       TSPAN6    ensembl_hav… protein_codi…
 9 ENSG00000000003 X     100635558   1.01e8 CDS        TSPAN6    ensembl_hav… protein_codi…
10 ENSG00000000003 X     100635178   1.01e8 exon       TSPAN6    ensembl_hav… protein_codi…
# … with 3,413,864 more rows, and 9 more variables: transcript_id <chr>, transcript_name <chr>,
#   transcript_source <chr>, transcript_biotype <chr>, transcript_version <int>,
#   transcript_support_level <chr>, exon_id <chr>, exon_number <int>, exon_version <int>
```

Ensembl Genomic Annotations Array

```
tdb_genes <- tiledb_array(
  "s3://tiledb-conferences/bioc-2021/ensembl-grch38-ens104",
  as.data.frame = TRUE
)
```

Genomic Annotations: Query 1

Retrieve annotations for a single gene using `[]`-style indexings.

```
tdb_genes["ENSG00000152822", ]
```

```
# A tibble: 122 x 18
   gene_id   chrom pos_start  pos_end type    gene_name gene_source gene_biotype
   <chr>     <chr>     <int>    <int> <chr>   <chr>     <chr>       <chr>
 1 ENSG0000… 6     146352250   1.46e8 CDS     GRM1      ensembl_ha… protein_cod…
 2 ENSG0000… 6     146357526   1.46e8 exon    GRM1      ensembl_ha… protein_cod…
 3 ENSG0000… 6     146357526   1.46e8 CDS     GRM1      ensembl_ha… protein_cod…
 4 ENSG0000… 6     146386890   1.46e8 exon    GRM1      ensembl_ha… protein_cod…
 5 ENSG0000… 6     146386890   1.46e8 CDS     GRM1      ensembl_ha… protein_cod…
 6 ENSG0000… 6     146398769   1.46e8 exon    GRM1      ensembl_ha… protein_cod…
 7 ENSG0000… 6     146398769   1.46e8 CDS     GRM1      ensembl_ha… protein_cod…
 8 ENSG0000… 6     146433872   1.46e8 exon    GRM1      ensembl_ha… protein_cod…
 9 ENSG0000… 6     146433872   1.46e8 CDS     GRM1      ensembl_ha… protein_cod…
10 ENSG0000… 6     146434794   1.46e8 stop_c… GRM1      ensembl_ha… protein_cod…
# … with 112 more rows, and 10 more variables: gene_version <int>,
#   transcript_id <chr>, transcript_name <chr>, transcript_source <chr>,
#   transcript_biotype <chr>, transcript_version <int>,
#   transcript_support_level <chr>, exon_id <chr>, exon_number <int>,
```

Genomic Annotations: Query 2

Add a **query condition** to filter for the gene's *transcript* level annotations.

```
query_condition(tdb_genes) <- parse_query_condition(type == "transcript")
tdb_genes["ENSG00000152822",]
```

```
# A tibble: 7 x 18
  gene_id    chrom pos_start  pos_end type    gene_name gene_source gene_biotype
  <chr>      <chr>     <int>    <int> <chr>   <chr>     <chr>       <chr>
1 ENSG00000… 6     146029486   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
2 ENSG00000… 6     146027782   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
3 ENSG00000… 6     146029062   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
4 ENSG00000… 6     146029283   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
5 ENSG00000… 6     146027782   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
6 ENSG00000… 6     146027703   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
7 ENSG00000… 6     146027646   1.46e8 transc… GRM1      ensembl_ha… protein_cod…
# … with 10 more variables: gene_version <int>, transcript_id <chr>,
#   transcript_name <chr>, transcript_source <chr>, transcript_biotype <chr>,
#   transcript_version <int>, transcript_support_level <chr>, exon_id <chr>,
#   exon_number <int>, exon_version <int>
```

Genomic Annotations: Query 3

Specify a subset of attributes to reduce the amount of data that's retrieved and returned.

```
attrs(tdb_genes) <-
  c("gene_name", "transcript_id", "chrom", "pos_start", "pos_end")
tdb_genes["ENSG00000152822",]

# A tibble: 7 x 6
  gene_id         gene_name transcript_id   chrom pos_start   pos_end
  <chr>           <chr>     <chr>           <chr>     <int>     <int>
1 ENSG00000152822 GRM1      ENST00000507907 6     146029486 146434885
2 ENSG00000152822 GRM1      ENST00000361719 6     146027782 146437598
3 ENSG00000152822 GRM1      ENST00000282753 6     146029062 146437601
4 ENSG00000152822 GRM1      ENST00000355289 6     146029283 146434885
5 ENSG00000152822 GRM1      ENST00000492807 6     146027782 146437595
6 ENSG00000152822 GRM1      ENST00000502405 6     146027703 146159891
7 ENSG00000152822 GRM1      ENST00000507005 6     146027646 146159692
```

2D Assay Matrices

2D Example: RNA-Seq Expression Matrix

Example: Store RNA-seq data generated by the Genotype-Tissue Expression (GTEx) project

- Original format: 56,200 gene × 17,382 sample matrix
- Array: 2D sparse array with dimensions for `gene_id` and `sample` names
- Typical query: Retrieve gene expression values for a particular gene across all or a subset of samples

The *GTEx Project* was supported by the Common Fund of the Office of the Director of the National Institutes of Health, and by NCI, NHGRI, NHLBI, NIDA, NIMH, and NINDS. The data used for this app were obtained from the GTEx Portal on 05/12/21.

GTEx RNA-Seq Raw Data

Name	GTEX-1117F-0226-SM-5GZZ7	GTEX-1117F-0426-SM-5EGHI	GTEX-1117F-0526-SM-5EGHJ
ENSG00000223972.5	0	0	0
ENSG00000227232.5	8.764	3.861	7.349
ENSG00000278267.1	0	0	1.004
ENSG00000243485.5	0.07187	0	0
ENSG00000237613.2	0	0	0
ENSG00000268020.3	0	0.056	0
ENSG00000240361.1	0.06621	0.05004	0

GTEx RNA-Seq Array Layout

gene_id	sample	tpm
"ENSG00000227232.5"	"GTEX-1117F-0226-SM-5GZZ7"	8.76
"ENSG00000227232.5"	"GTEX-1117F-0426-SM-5EGHI"	3.86
"ENSG00000227232.5"	"GTEX-1117F-0526-SM-5EGHJ"	7.35
"ENSG00000278267.1"	"GTEX-1117F-0526-SM-5EGHJ"	1.00
"ENSG00000243485.5"	"GTEX-1117F-0226-SM-5GZZ7"	0.0719

RNA-seq Array: Query 1

Retrieve TPM expression values for a single gene across all samples:

```
tdb_gtex <- tiledb_array(
  "s3://tiledb-conferences/bioc-2021/gtex-rnaseq
  as.data.frame = TRUE
)

tdb_gtex["ENSG00000202059.1", ]
```

```
# A tibble: 3,183 x 3
   gene_id           sample                    tpm
   <chr>             <chr>                   <dbl>
 1 ENSG00000202059.1 GTEX-1117F-0226-SM-5GZZ7 0.464
 2 ENSG00000202059.1 GTEX-1117F-0626-SM-5N9CS 0.437
 3 ENSG00000202059.1 GTEX-1117F-1326-SM-5EGHH 0.768
 4 ENSG00000202059.1 GTEX-1117F-2526-SM-5GZY6 1.06
 5 ENSG00000202059.1 GTEX-111CU-0526-SM-5EGHK 0.275
 6 ENSG00000202059.1 GTEX-111FC-0526-SM-5GZZ8 0.436
 7 ENSG00000202059.1 GTEX-111FC-1126-SM-5GZWU 0.439
 8 ENSG00000202059.1 GTEX-111FC-2926-SM-5GZY7 0.366
 9 ENSG00000202059.1 GTEX-111VG-0526-SM-5N9BW 0.888
10 ENSG00000202059.1 GTEX-111VG-0726-SM-5GIDC 0.992
# … with 3,173 more rows
```

Sample Metadata

Use associated metadata to retrieve identifiers for a particular subset of samples.

```
tdb_samples <- tiledb_array(
  "s3://tiledb-conferences/bioc-2021/gtex-analysis-sample-attributes",
  as.data.frame = TRUE,
  attrs = "SMTS",
  query_condition = parse_query_condition(
    SMTS == "Skin"
  )
)

tdb_samples[]
```

```
# A tibble: 424 x 2
   sample                  SMTS
   <chr>                   <chr>
 1 GTEX-1117F-2926-SM-5GZYI Skin
 2 GTEX-111CU-1126-SM-5EGIM Skin
 3 GTEX-111CU-1926-SM-5GZYZ Skin
 4 GTEX-111FC-0126-SM-5N9DL Skin
 5 GTEX-111FC-2526-SM-5GZXU Skin
 6 GTEX-111VG-0008-SM-5Q5BG Skin
 7 GTEX-111VG-1626-SM-5EGIO Skin
 8 GTEX-111VG-2426-SM-5GZXD Skin
 9 GTEX-111YS-0008-SM-5Q5BH Skin
10 GTEX-111YS-1526-SM-5GZYW Skin
# … with 414 more rows
```

RNA-seq Array: Query 2

Slice across both dimensions to retrieve expression values for a gene across a subset of samples.

```
gene_id <- "ENSG00000202059.1"
samples <- tdb_samples[]$sample

# specify the query range
selected_ranges(tdb_gtex) <- list(
  gene_id = cbind(gene_id, gene_id),
  sample = cbind(samples, samples)
)

# run the query
tdb_gtex[]
```

```
# A tibble: 82 × 3
   gene_id           sample                    tpm
   <chr>             <chr>                   <dbl>
 1 ENSG00000202059.1 GTEX-111VG-2426-SM-5GZXD 0.328
 2 ENSG00000202059.1 GTEX-113IC-0008-SM-5QGRF 0.424
 3 ENSG00000202059.1 GTEX-113IC-0126-SM-5HL6T 0.466
 4 ENSG00000202059.1 GTEX-117YW-2626-SM-5GZZH 0.324
 5 ENSG00000202059.1 GTEX-117YX-2326-SM-5H12W 0.318
 6 ENSG00000202059.1 GTEX-1192W-2626-SM-5Q5AF 0.361
 7 ENSG00000202059.1 GTEX-1192X-0226-SM-5H12D 0.325
 8 ENSG00000202059.1 GTEX-1192X-2726-SM-5N9DN 0.305
 9 ENSG00000202059.1 GTEX-11DXW-0826-SM-5H118 0.316
10 ENSG00000202059.1 GTEX-11DXY-0126-SM-5H11Q 0.282
# … with 72 more rows
```

Genomic Ranges

Genomic Ranges

(but not `GenomicRanges`, sorry...)

Genomic Range Indexing

Example: Store GWAS summary statistics from the UK Biobank for all analyzed phenotypes.

- Original dataset: Contains ~ 12,000 results files for each individual GWAS trait. Individual files contain genomic location information and summary stats for ~10 million variants.
- Array: 3D sparse array with dimensions for `phenotype`, `chromosome`, and chromosome start `pos`.
- Typical query: slice by a particular genomic location for a trait of interest

UK Biobank announcement:

`http://www.nealelab.is/uk-biobank/ukbround2announcement`

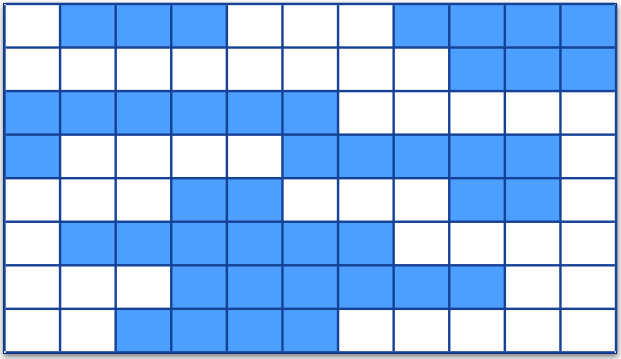
GWAS Raw Data

variant	beta	se	tstat	pval
1:15791:C:T	-1.70174e+01	5.66755e+01	-3.00260e-01	7.63979e-01
1:69487:G:A	-5.70053e-02	1.11014e-01	-5.13496e-01	6.07605e-01
1:69569:T:C	-2.30684e-03	1.99098e-02	-1.15865e-01	9.07760e-01
1:139853:C:T	-5.62416e-02	1.11017e-01	-5.06603e-01	6.12434e-01
1:692794:CA:C	7.72562e-04	9.22074e-04	8.37852e-01	4.02114e-01
1:693731:A:G	1.31202e-03	8.71218e-04	1.50596e+00	1.32078e-01
1:707522:G:C	8.77269e-04	9.79498e-04	8.95631e-01	3.70450e-01
1:717587:G:A	-8.32431e-05	2.33724e-03	-3.56160e-02	9.71589e-01
1:723329:A:T	-1.15975e-02	6.88597e-03	-1.68422e+00	9.21406e-02
1:730087:T:C	4.23934e-05	1.21371e-03	3.49286e-02	9.72137e-01

GWAS Array Layout

Dimension 3:
Chromosome Position

Dimension 2:
Chromosome



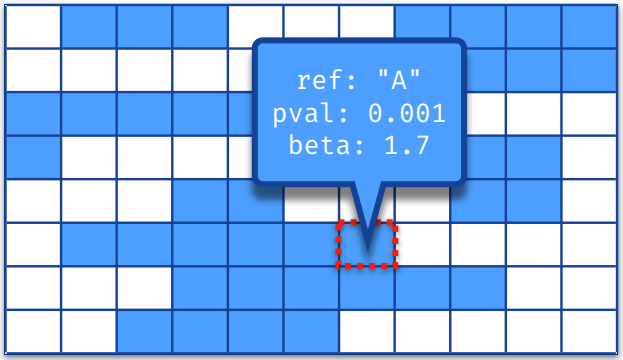
GWAS Array Layout

Dimension 3:
Chromosome Position

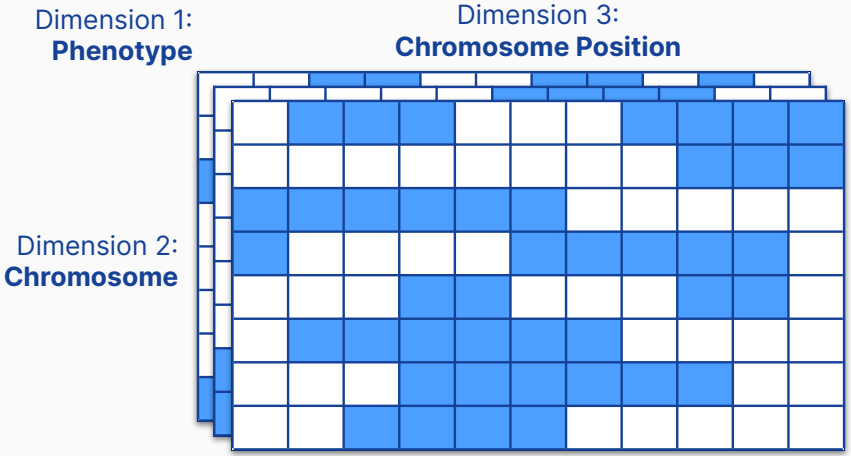
543

Dimension 2:
Chromosome

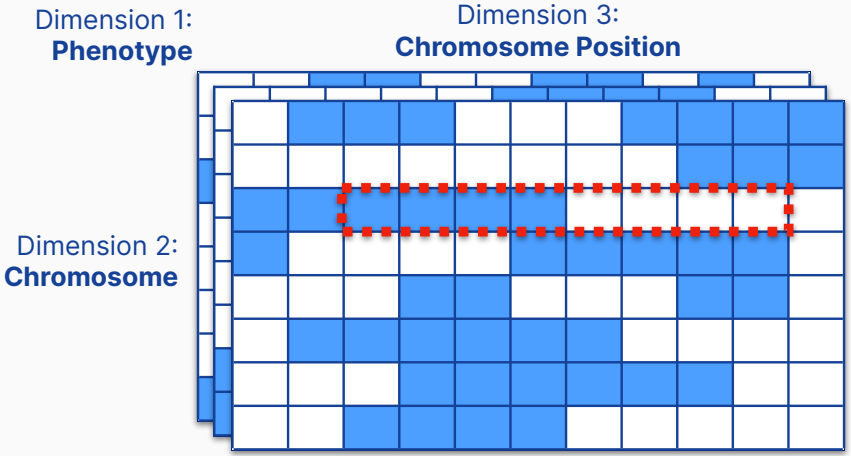
chr6



GWAS Array Layout



GWAS Array Layout



GWAS Results: Query 1

Use `[]` indexing to query the first 2 dimensions (i.e., `phenotype` and `chr`).

```
tdb_gwas <- tiledb_array(
  "tiledb://aaronwolen/ukbiobank-gwasdb",
  as.data.frame = TRUE,
  attrs = c("beta", "se", "pval")
)
```

```
tdb_gwas["Water intake", "20"]

# A tibble: 290,721 x 6
   phenotype    chr     pos       beta       se  pval
   <fct>        <fct> <int>      <dbl>    <dbl> <dbl>
 1 Water intake 20    61098  0.00199   0.00246 0.417
 2 Water intake 20    61270 -0.00113   0.00719 0.876
 3 Water intake 20    61795  0.000381  0.00218 0.861
 4 Water intake 20    62731 -0.00200   0.00328 0.541
 5 Water intake 20    63231  0.00219   0.00683 0.749
 6 Water intake 20    63244  0.00201   0.00254 0.429
 7 Water intake 20    63452 -0.0000762 0.00719 0.992
 8 Water intake 20    63799  0.000339  0.00218 0.876
 9 Water intake 20    63967 -0.000765  0.00725 0.916
10 Water intake 20    65288 -0.00194   0.00327 0.552
# … with 290,711 more rows
```

GWAS Results: Query 2

Use `selected_ranges` to query all 3-dimensions and extract data for a specific genomic region.

```
selected_ranges(tdb_gwas) <- list(
  phenotype = cbind("Water intake", "Water intake"),
  chr = cbind("20", "20"),
  pos = cbind(5e6, 6e6)
)
tdb_gwas[]

# A tibble: 5,140 x 6
   phenotype    chr       pos      beta       se   pval
   <chr>        <chr>   <int>     <dbl>   <dbl>  <dbl>
 1 Water intake 20    5000142  0.0138  0.0103  0.180
 2 Water intake 20    5000146 -0.00457 0.00529 0.388
 3 Water intake 20    5000279  0.00523 0.0181  0.773
 4 Water intake 20    5000280 -0.00605 0.00246 0.0139
 5 Water intake 20    5000337 -0.00459 0.00529 0.386
 6 Water intake 20    5000581 -0.00545 0.00551 0.323
 7 Water intake 20    5000753 -0.0199  0.0279  0.475
 8 Water intake 20    5000780 -0.00464 0.00529 0.380
# … with 5,132 more rows
```

Wrap Up

In Summary

TileDB

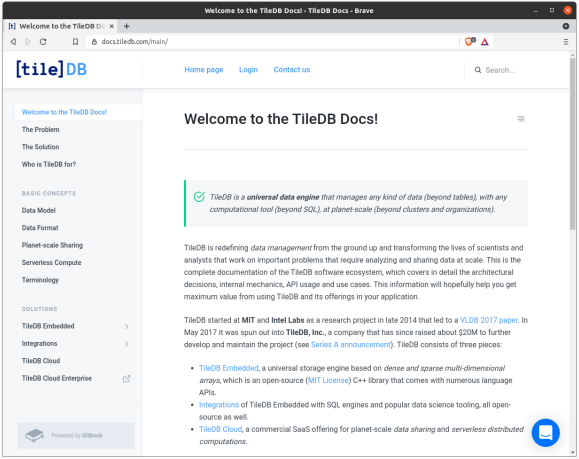
- an open-source embeddable storage engine
- an open-source format for modeling any type of data
- fully cloud-native on AWS, GCS, Azure
- limitless scalability
- offers time travel
- offers encryption

In Summary

TileDB R Package

- available on CRAN, and already used by Bioconductor
- high-level R-friendly interface for creating/query TileDB arrays
- also includes low-level access to the full TileDB API
- fully interoperable with DBI, Arrow, …

Documentation



Extensive documentation on TileDB, APIs, Usage, and more

`docs.tiledb.com`

`github.com/TileDB-Inc/TileDB-R`

`github.com/TileDB-Inc/TileDB`

Talk to TileDB

email	hello@tiledb.com
web	https://tiledb.com/
docs	https://docs.tiledb.com/main
forum	https://forum.tiledb.com/
github	https://github.com/TileDB-Inc/TileDB
twitter	https://twitter.com/tiledb
slack	https://tiledb-community.slack.com/
jobs	https://apply.workable.com/tiledb/

we're hiring!!