National Library of Medicine
National Center for Biotechnology Information

# Sequence Read Archive (SRA) Data
## Technical User Manual

**Document:** v1.1
**SRA Toolkit:** v3.2.1
**Last updated:** June 16, 2025

*www.ncbi.nlm.nih.gov/sra*

# *Table of Contents*

# Overview

This document describes the data format that SRA stores, as well as the archive sequence data and underlying Virtual Database (VDB) that the SRA format is built on top of.

This document is intended to serve users who want a deep technical understanding of the data available in SRA.

---

# 1. SRA Design

## *1.1. Sequence Read Archive (SRA) Data*

The Sequence Read Archive (SRA) is designed to store large-scale Next-Generation Sequencing (NGS) data. This data often quite large, sometimes exceeding 1 TB per file.

SRA data includes essential nucleotide basecalls which represent DNA bases (A, C, G, T) or RNA bases (A, C, G, U). In addition to basecalls, most data additionally contain confidence scores, also known as quality scores, that indicate the reliability of these basecalls, provided either per base or as an overall score for each read.

Some data submissions may also include alignment data, which maps the sequences to reference genomes or custom references provided by submitters. All SRA data is stored using the Virtual Database (VDB), an efficient and scalable data storage system.

## *1.2. SRA Normalized Format*

Data are submitted to the SRA in their original BAM, CRAM, or FASTQ file formats. These submissions undergo an Extract, Transform, and Load (ETL) process to convert them into a standardized, harmonized format known as the **SRA Normalized Format**. This process ensures:

- **Interoperability**: the data from diverse studies and sequencing platforms can be compared effectively.
- **Size Reduction**: The harmonized format is more storage-efficient than the original submission files.

Because sequencing platforms may produce data in varying file types, the Normalized Format standardizes this information.

For data aligned to a reference genome, SRA employs a specialized compression method known as **cSRA Format**. This serialized, read-only format, resembling a tar archive structure, optimizes storage while retaining alignment data. Access to cSRA formatted data is available through the SRA Toolkit.

The SRA Normalized Format is structured as a columnar database, integrating sequence data with corresponding reference and alignment information. (See diagram in ***Fig. 1.***)

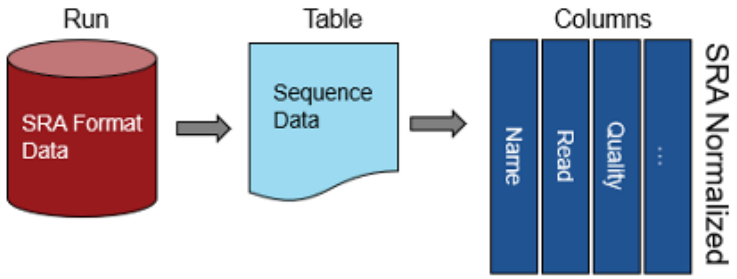


***Figure 1.*** *The logical structure of the SRA format. Each SRA run has at least one Table, which contains sequence, reference, and/or alignment data. Runs may have additional tables to store additional data-like alignments. Each of these Tables is composed of multiple Columns which store types of data.*

## *1.3. SRA Lite Format*

The **SRA Lite Format** is a streamlined format based on the SRA Normalized format designed to further reduce data size by simplifying sequence quality score information.

In this format:

- Individual base quality scores (as seen in the SRA Normalized Format) are replaced with a binary read filter that represents the overall quality of the read.
- Alignments are preserved if they were included in the original submission.
- Compression by reference (as seen in cSRA) is also maintained.

For workflows requiring base-level quality scores, the SRA Lite Format allows simulated quality scores to be generated using the read filter value. These scores can be uniformly set to either a high value (Phred 30) or a low value (Phred 3) for each base in the run.

## 1.4. Key Terms

| | |
|---|---|
| *Basecall* | The genomic base that has been proposed for a given location. This could include A, T, C, G, N or other ambiguity codes defined by IUPAC. https://pmc.ncbi.nlm.nih.gov/articles/PMC341218/ |
| *Kar* | An archive format like the Unix tar utility that is used to store a directory tree and many files as a single file. |
| *Quality Score* | A confidence score for a basecall in sequencing data. Quality scores typically use the Phred scoring system. https://pmc.ncbi.nlm.nih.gov/articles/PMC341218/ |
| *Read* | The individual fragments of sequence that have a specific meaning within a spot. Primer, adapter, forward, and reverse reads are each considered to be different reads. |
| *Spot* | The entire combined base content for a single measured template of sequence. For platforms with X and Y coordinates this would be all bases produced for a unique X and Y coordinate on the detection media. |
| *SRA Lite* | The SRA sequence data storage format produced with the SRA Toolkit that replaces base quality scores with binary read quality scores. |
| *SRA Normalized* | The SRA sequence data storage format produced with the SRA Toolkit and containing full resolution base quality scores. |

# *1.5. Types of Tables*

## 1.5.1. Table Overview

***Table 1.*** *A description of the four tables that may be present in an SRA Normalized archive. The alignment and reference tables are only used for runs submitted with alignments to a reference sequence or genome.*

| Table Name | Required? | Description |
|---|---|---|
| SEQUENCE | REQUIRED | Contains the bases, qualities and additional contents of the sequencing data. (See ***Table 2***.) |
| PRIMARY_ALIGNMENT | Optional | Information for how the sequence data maps to the reference data. (See ***Table 3***.) |
| SECONDARY_ALIGNMENT | Optional | Alternative mappings of the sequence data to the reference. A read will not have a secondary alignment without a primary alignment. (See ***Table 3***.) |
| REFERENCE | Required if PRIMARY_ALIGNMENT table present | The reference sequence either stored fully in the run or externally stored and accessed using a unique identifier, such as a reference genome (e.g., GRCh38) or an individual sequence record (e.g., Genbank, RefSeq). (See ***Table 4***.) |

## 1.5.2. Sequence Table Columns

***Table 2.*** *A description of the sequence table (presented in* ***Table 1)****, which is present in all SRA Normalized archives. It contains the bases, qualities and additional contents of the sequencing data.*

| Column Name | Description | Typically static (same value for all rows) |
|---|---|---|
| ALIGNMENT_COUNT | vector of integers, how many alignments per read | No |
| BASE_COUNT | number of bases in the sequence table | Yes |
| BIO_BASE_COUNT | number of bases (excluding submitter-noted technical sequences like adapters, primers, etc.) in the sequence table | Yes |
| CMP_BASE_COUNT | number of unaligned bases in the sequence table | Yes |
| CMP_READ | compressed read, only the unaligned reads [Required to output FASTQ from cSRA] | No |
| COLOR_MATRIX | describes the translation between color-space and base-space | Yes |
| CSREAD | translated READ-column into color-space | No |
| CS_KEY | key for translation between color-space and base-space | Yes |
| CS_NATIVE | flag that indicates if sequence was produced in color-space | Yes |
| FIXED_SPOT_LEN | Flag that indicates if all reads have the same length | Yes |
| MAX_SPOT_ID | id of the last spot in the table | Yes |
| MIN_SPOT_ID | id of the first spot in the table | Yes |
| NAME | name of the spot, generated from the row-id | No |
| PLATFORM | name of the sequencing platform used to generate the reads in the table | Yes |
| PRIMARY_ALIGNMENT_ID | Reference to corresponding row-id in primary alignment table [Required to output FASTQ from cSRA] | No |
| QUALITY | stored quality values, ordered in the same direction as the corresponding read | No |

| | | |
|---|---|---|
| READ | assembled or stored bases, ordered in the direction of sequencing [Required to output FASTQ] | No |
| READ_FILTER | vector of flags, one for each read | No |
| READ_LEN | vector of integers, one for each read, length of each read [Required to output FASTQ] | No |
| READ_SEG | vector of integer-pairs, one for each read, [zero-based start offset of read, length of read] | No |
| READ_START | vector of integers, one for each read, zero-based start offset of read | No |
| READ_TYPE | vector of flags, one for each read, tells if read is biological or technical and the direction it was sequenced [Required to output FASTQ] | No |
| SIGNAL_LEN | lengths of recorded signal | No |
| SPOT_COUNT | number of spots in the table (=MAX_SPOT_ID) | Yes |
| SPOT_GROUP | describes grouping in the reads, equivalent to read-group in BAM | No |
| SPOT_ID | row-id of each spot (1-based) | No |
| SPOT_LEN | number of bases in the spot | No |
| TRIM_LEN | length of sequence remaining after trimming | No |
| TRIM_START | coordinate of first untrimmed base | No |

## 1.5.3. Alignment Tables Columns

***Table 3.*** *A description of* `PRIMARY_ALIGNMENT` *and* `SECONDARY_ALIGNMENT` *alignment tables columns (presented in* ***Table 1)****, which are optionally present in SRA Normalized archives.* `PRIMARY_ALIGNMENT` *columns contain information for how the sequence data maps to the reference data, while* `SECONDARY_ALIGNMENT` *columns contain alternative mappings of the sequence data to the reference. A read will not have a secondary alignment without a primary alignment.*

| Column Name | Description | Typically static (same value for all rows) |
|---|---|---|
| ALIGN_ID | row-id of each spot (1 based) | No |
| BASE_COUNT | number of bases in the alignment table | Yes |
| BIO_BASE_COUNT | number of bases (excluding submitter-noted technical sequences like adapters, primers, etc.) in the sequence table | Yes |
| CIGAR_LONG | long form of the CIGAR (Compact Idiosyncratic Gapped Alignment Report) string | No |
| CIGAR_SHORT | short form of the CIGAR string | No |
| COLOR_MATRIX | describes the translation between color-space and base-space | Yes |
| CS_KEY | key for translation between color-space and base-space | Yes |
| CS_NATIVE | flag that indicates if the sequence was produced in color-space | Yes |
| EDIT_DISTANCE | number of mismatches | No |
| GLOBAL_REF_START | global position in the reference table | No |
| HAS_MISMATCH | bitfield of mismatches | No |
| HAS_REF_OFFSET | bitfield of offsets in the reference, used to represent indels | No |
| LABEL | alignment label, provides future compatibility multi-ploid alignment representation | No |
| LABEL_LEN | length of the label-part to be used | No |
| LABEL_START | start offset of the label-part to be used | No |
| MAPQ | PHRED mapping quality | No |

| | | |
|---|---|---|
| MATE_ALIGN_ID | row-id of the mate of this read (paired reads only) | No |
| MATE_CIGAR_LONG | long form of the CIGAR-string of the mate (paired-reads only) | No |
| MATE_CIGAR_SHORT | short form of the CIGAR-string of the mate (paired-reads only) | No |
| MATE_EDIT_DISTANCE | number of mismatches in the mate (paired-reads only) | No |
| MATE_REF_ID | row-id in the reference-table in the mate (paired-reads only) | No |
| MATE_REF_LEN | mate alignment lines in reference coordinates (paired-reads only) | No |
| MATE_REF_NAME | reference-name to which the mate is aligned (paired-reads only) | No |
| MATE_REF_ORIENTATION | orientation of the mate (paired-reads only) | No |
| MATE_REF_POS | mate position on the reference (paired-reads only) | No |
| MAX_SPOT_ID | id of the last spot in the table | Yes |
| MIN_SPOT_ID | id of the first spot in the table | Yes |
| MISMATCH | base values of the mismatches | No |
| MISMATCH_QUAL | qualities of the mismatches | No |
| NAME | auto-generated name of the alignment from row-id | No |
| PLATFORM | name of the sequencing platform used to generate the reads in the table | Yes |
| QUALITY | stored quality values, ordered in the same direction as the corresponding read | No |
| RAW_READ | original sequence ordered in the direction of sequencing | No |
| RD_FILTER | vector of flags, one for each read | No |
| READ | assembled or stored bases, ordered in the direction of sequencing | No |
| READ_FILTER | vector of flags, one for each read | No |
| READ_LEN | vector of integers, one for each read, length of each read | No |

| | | |
|---|---|---|
| READ_START | vector of integers, one for each read, zero-based start offset of read | No |
| READ_TYPE | vector of flags, one for each read, tells if read is biological or technical and the direction it was sequenced | No |
| REF_ID | row-id in the reference table | No |
| REF_LEN | length of alignment in reference coordinates | No |
| REF_NAME | name of the reference | No |
| REF_OFFSET | orientation of original sequence relative to the reference | No |
| REF_POS | position on the reference at which the alignment starts | No |
| REF_READ | subsequence of reference on which alignment is projected | No |
| REF_SEQ_ID | sequence id used in the submitted reference | No |
| REF_START | offset in the row-id of the reference at which alignment starts | No |
| REF_TABLE | name of the reference table | No |
| SAM_FLAGS | flags to be used in SAM-format | No |
| SAM_QUALITY | quality score converted to ASCII presentation from sequence-row-id | No |
| SEQ_NAME | SRA auto-generated name of the sequence from sequence-row-id | No |
| SEQ_READ_ID | read-id of aligned sequence | No |
| SPOT_COUNT | number of spots in the table (=MAX_SPOT_ID) | Yes |
| SPOT_GROUP | describes grouping in the reads, equivalent to read-group in BAM | No |
| SEQ_SPOT_ID | sequence spot id | No |
| SPOT_LEN | number of bases in spot | No |
| TEMPLATE_LEN | template length | No |
| TRIM_LEN | length of sequence remaining after trimming | No |
| TRIM_START | coordinate of first untrimmed base | No |

## 1.5.4. Reference Table Columns

***Table 4.*** *The reference table columns (presented in* ***Table 1)*** *are required in an SRA Normalized archive if the alignment table is present. The reference sequence to which data are aligned may be present as a locally stored sequence or as an external sequence.*

| Column Name | Description | Typically static (same value for all rows) |
|---|---|---|
| BASE_COUNT | number of bases in the reference table | Yes |
| BIO_BASE_COUNT | number of bases (excluding submitter noted technical sequence like adapters, primers, etc.) in the sequence table | Yes |
| CGRAPH_HIGH | maximum coverage depth in this alignment fragment | No |
| CGRAPTH_INDELS | total number of indels in this alignment fragment | No |
| CGRAPH_LOW | minimum coverage depth in this alignment fragment | No |
| CGRAPH_MISMATCHES | total number of mismatches between sequence and this alignment fragment | No |
| CIRCULAR | flag if this reference is circular | No |
| CMP_BASE_COUNT | count of unaligned bases in the reference table | Yes |
| CMP_READ | compressed read (unaligned reads only) | No |
| COLOR_MATRIX | describes the translation between color-space and base-space | Yes |
| CSREAD | translated READ-column into color-space | No |
| CS_KEY | key for translation between color-space and base-space | Yes |
| CS_NATIVE | flag that indicates if the sequence was produced in color-space | Yes |
| LABEL | description of this sequence fragment | No |
| LABEL_LEN | length of description | No |
| LABEL_START | start offset of description | No |
| MAX_SEQ_LEN | maximum size for the sequence fragment in this table | No |

| | | |
|---|---|---|
| MAX_SPOT_ID | id of the last spot in the table | Yes |
| MIN_SPOT_ID | id of the first spot in this table | Yes |
| NAME | name of the sequence, equivalent what BAM used in the reference-sequence-name-field | No |
| NAME_RANGE | used internally by SRA for index lookup | No |
| PRIMARY_ALIGNMENT_IDS | list of row-id's from primary alignment table which start their alignment in this alignment fragment | No |
| QUALITY | stores the quality of the reference, auto-generated when not available | No |
| RD_FILTER | vector of flags, one for each read | No |
| READ | the sequence of the reference, merges remote and local reference into one column | No |
| READ_FILTER | vector of flags, one for each read | No |
| READ_LEN | vector of integers, one for each read, length of each read | No |
| READ_START | vector of integers, one for each read, zero-based start offset of read | No |
| READ_TYPE | vector of flags, one for each read, tells if read is biological or technical and the direction it was sequenced | No |
| SECONDARY_ALIGNMENT_IDS | list of row-id's from secondary alignment table which start their alignment in this alignment fragment | No |
| SEQ_ID | id of remotely stored sequence, used as a key to find the sequence | No |
| SEQ_LEN | the length of the fragment from the remotely stored sequence | No |
| SEQ_START | the start of this fragment on the remote sequence | No |
| SPOT_COUNT | number of spots in the table (=MAX_SPOT_ID) | No |
| SPOT_GROUP | describes grouping in the reads, equivalent to read-group in BAM | No |
| SPOT_ID | row-id of each spot (1 based) | No |
| SPOT_LEN | how many bases are in this spot | No |

| | | |
|---|---|---|
| TRIM_LEN | length of the section not subject to be trimmed | No |
| TRIM_START | start of the section not subject to be trimmed | No |

# 2. Sequence Compression Format

## *2.1. Compression Strategy*

Compression in SRA is more complex than simply running a compression tool on the resulting files. In addition to storing basecalls as numeric representations, several tactics are used to improve compression of the data, such as compression by reference and applying standard compression algorithms in ways to maximize efficiency.

Data types are stored as columns within the cSRA format. These columns are then individually compressed, which reduces the complexity of the characters seen by the compression algorithm and therefore increases the compression efficiency. Data types such as bases can be packed or stored as numeric values within VDB to reduce the required storage space substantially while still allowing the SRA toolkit to easily output the data using the text characters that users expect and need in their workflows.

## *2.2. Compression Format (cSRA)*

The cSRA format is a serialized read-only file structure similar to the tar archive format. Access to the data in the cSRA format is through the SRA Toolkit

The cSRA format uses **compression by reference** to reduce the size of the SRA storage footprint. Compression by reference uses the reference sequences in the alignment to reduce the storage size of the SRA runs themselves. Compression by reference requires a reference sequence, either in the REFERENCE table within the cSRA or as an external VDB format file, as a method to compress the sequence data information.  Consequently, reading the data in a cSRA archive file may require the availability and download of many additional references.

## 2.3. cSRA Format Structure

A cSRA archive file contains multiple tables. In the example shown in ***Fig. 2***, these tables are SEQUENCE, PRIMARY_ALIGNMENT, SECONDARY_ALIGNMENT, and REFERENCE.

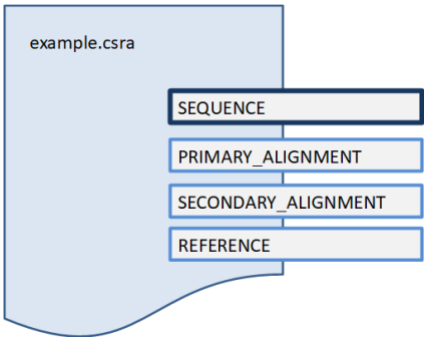


***Figure 2.*** *An example of the cSRA format structure. The example cSRA archive file (`example.csra`) may contain the tables SEQUENCE, PRIMARY_ALIGNMENT, SECONDARY_ALIGNMENT, and/or REFERENCE. Not all runs will include a SECONDARY_ALIGNMENT.*

## 2.4. Example of Aligned Sequence

An example of an alignment with both primary (PRIM) and secondary (SEC) alignments is shown in ***Fig. 3***. Data in the sequence table can have primary and/or secondary alignments, or might not have an alignment at all.

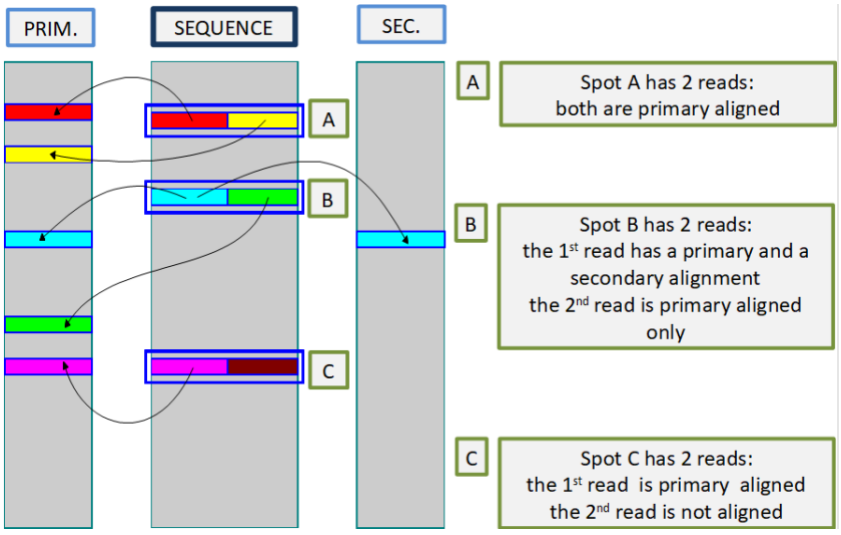


***Figure 3.*** *An alignment with both primary (PRIM) and secondary (SEC) alignments.*

## 2.5. Difference from Reference Format Storage

Below are several examples of how the differences in a read from the reference sequence are stored in a cSRA file. The columns HAS_MISMATCH, HAS_REF_OFFSET, MISMATCH, and REF_OFFSET from the Alignment table are used to store information about how the sequence data differs from the reference sequence. Visual representations of these examples are presented in ***Fig. 4***.

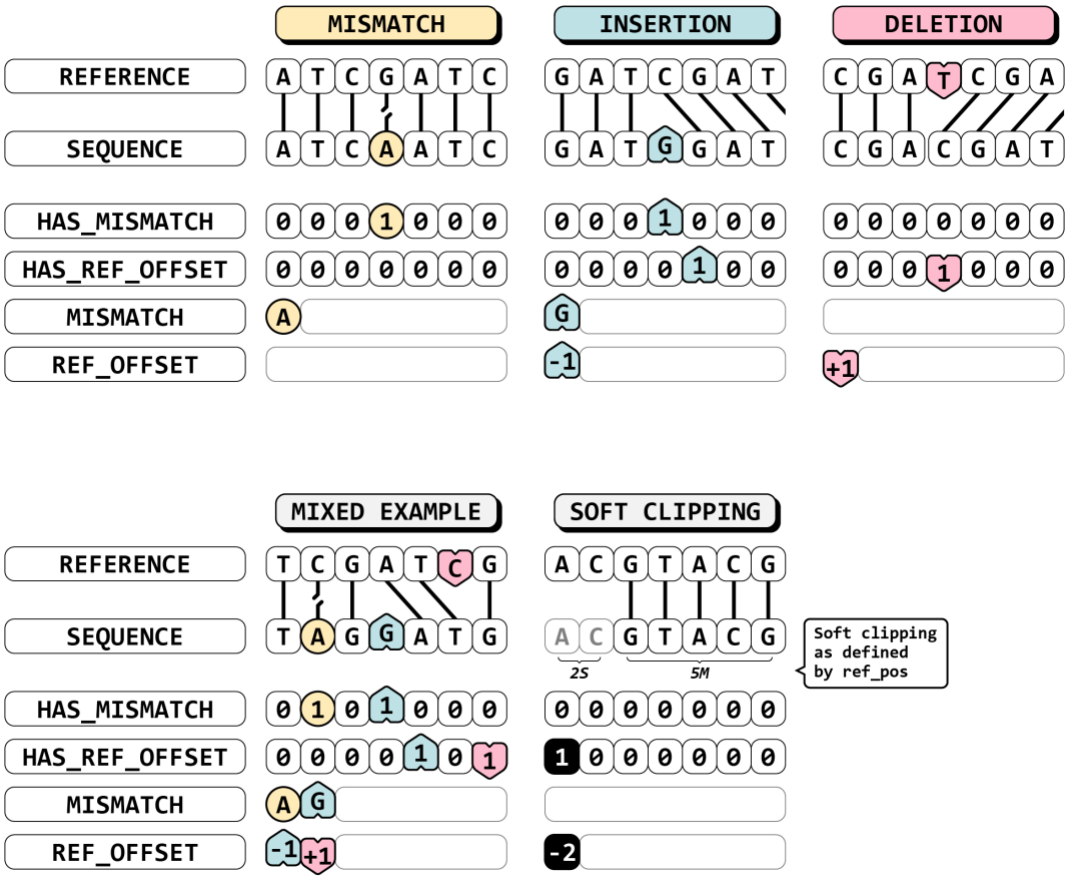


***Figure 4.*** *Representation of mismatches, insertions, deletions, softclipping, and combinations thereof in the cSRA format. Each panel compares reference and sequence reads, followed by descriptor columns (HAS_MISMATCH, HAS_REF_OFFSET, MISMATCH, REF_OFFSET). In the "Mismatch" example, a single base difference is flagged by HAS_MISMATCH and recorded in MISMATCH. The "Insertion" example shows an extra base in the read, with HAS_REF_OFFSET and a -1 value in REF_OFFSET indicating insertion before the next reference base. The "Deletion" example marks a missing base with HAS_REF_OFFSET and a +1 offset in REF_OFFSET, indicating a skipped reference base. The "Mixed" example combines multiple events across a single read. In the "Soft Clipping" example, unaligned bases are represented using negative REF_OFFSET values. Alignment start is defined by ref_pos in the sequence file, with the first two bases being soft-clipped (2S), and the next five bases matching the reference (5M).*

### 2.5.1. Single Base Mismatch

An example of how a single base mismatch is represented in the cSRA format is presented in ***Fig. 4.*** The mismatch (from G to A) is identified as a “1” in the HAS_MISMATCH column. Non mismatched bases are retained as a “0” in this column. The actual sequence difference (in this case an “A”) is stored in the MISMATCH column.

### 2.5.2. Single Base Insert

An example of how a single base insert is represented in the cSRA format is presented in ***Fig. 4***. The inserted base (a “G”) is identified as a “1” in the HAS_MISMATCH column as well as a “1” in the HAS_REF_OFFSET column. The mismatched base (the “G”) is stored in the MISMATCH column, and the REF_OFFSET column has a “-1” to signify the corresponding sequence shift compared to the reference sequence.

### 2.5.3. Single Base Deletion

An example of how a single base deletion is represented in the cSRA format is presented in ***Fig. 4***. The single base deletion (a “T” that is present in the reference but not in the sequence) is identified as a “1” in the HAS_REF_OFFSET. The REF_OFFSET column has a “+1” to signify the corresponding sequence shift compared to the reference sequence.

### 2.5.4. Multiple Changes

It is not uncommon for sequences to contain multiple changes compared to the reference — insertions, deletions, and mismatches. The rules for representing these changes with the cSRA format remains the same, with all mismatches and offsets being represented in their respective columns (***Fig. 4***).

The “Mixed Example” panel shows how a mismatch, insert, and deletion are all represented within a single cSRA format example.

### 2.5.5. Soft Clip

Soft clipping — in which portions of a read sequence that don't align well to the reference sequence are temporarily masked (or "clipped") without being removed from the alignment file — can also be represented in cSRA format (***Fig. 4***). These portions of the sequences are defined by the REF_POS column, which is the position on the reference at which the alignment starts.

# 3. Quality Scores Compression Format

## *3.1. SRA Lite Format*

The SRA Lite conversion process removes the individual base quality scores from the processed runs and replaces them with a read filter that represents the quality of the entire read with a single value. The low (PHRED 3) filter value is determined by examining individual reads and identifying reads that either start or end with a run of quality scores below Phred 20 or have more than half of the entire quality scores below Phred 20. All other reads will have the high (PHRED 30) value output for all bases in the read.

## *3.2. SRA Lite Conversion*

SRA converts all supported submitted sequence data to the SRA Lite format. Beginning with version 2.11 of the SRA Toolkit, this is done as part of the ETL process. The SRA Toolkit creates a read filter column and allows for the removal of the quality score data from the runs without any other modification needed to produce an SRA Lite format run. This substantially simplifies the creation of the SRA Lite format. In the absence of per base quality score data, the read filter is used to determine whether the read is marked as pass or fail and to generate synthetic per-base quality scores accordingly. However, much of the data in the SRA archive was processed with earlier versions of the Toolkit that did not include a read filter as part of the loading process. To convert these data to Lite format, it was necessary to re-open and read the quality scores from the Normalized format to which they had already been converted, generate a read filter mask, and then apply that mask to the runs and remover the per base quality scores. Described below is the method for converting data in the SRA archive that was processed before a read filter was included as part of ETL from the SRA Normalized format to SRA Lite format.

## *3.3. SRA Normalized vs. SRA Lite*

Below is a description of how SRA Normalized and SRA Lite formats differ from one another.

When converting runs from SRA Normalized format to SRA, the SRA schema (see section 5.1) will be updated according to a mapping in the SRA Lite conversion configuration.

The following columns, if present in the SRA Normalized run, will be removed upon conversion to SRA Lite.

- QUALITY
- QUALITY2
- CMP_QUALITY
- POSITION
- SIGNAL

Columns added to SRA Lite:

- RD_FILTER

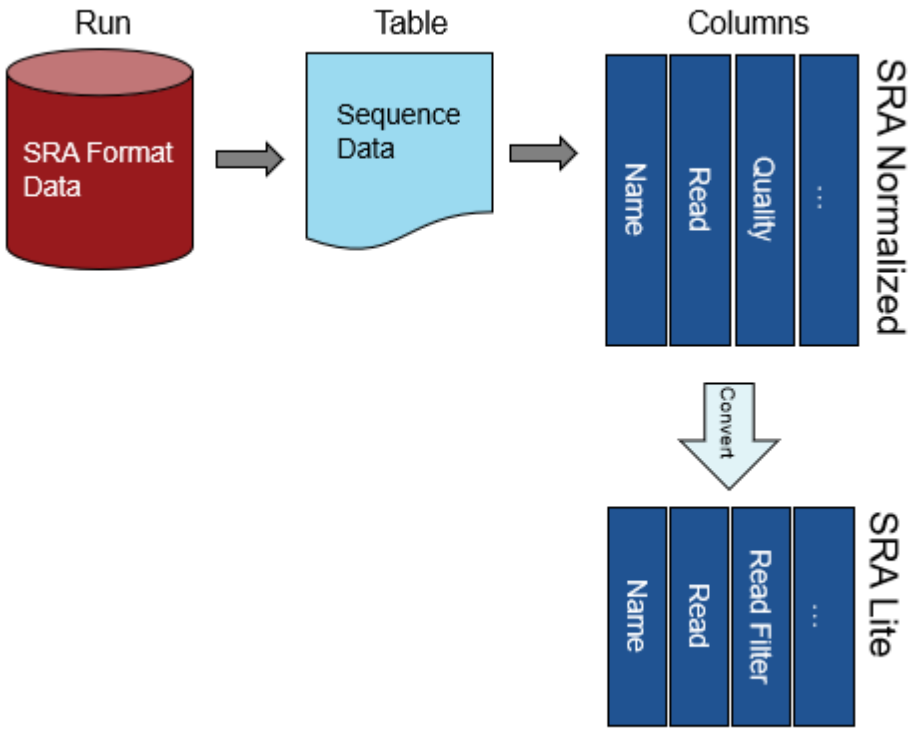


***Figure 5.*** *A representation of how the SRA Lite format is produced from the SRA Normalized format.*

# 4. Toolkit Binaries Used in Legacy Data Conversion to SRA Lite

The information present in ***Table 5*** is a list of SRA Toolkit Binaries used in legacy data conversion. These binaries are available with the SRA Toolkit.

***Table 5.*** *List of SRA Toolkit Binaries. These tools are used to create the read filter values on data submitted to SRAprior to the creation of the SRA Lite format.*

| Binary | Description |
|---|---|
| kar+ | Used to unpack the kar into the directory and file structure of VDB to update the contents. |
| kar+meta | A modified version of the kar program that includes additional features for handling metadata. |
| make-read-filter | Updates the filter flag column in a run. |
| vdb-dump | A tool to read and output the contents of a run in a configurable way. |
| vdb-lock | Creates a lock file to prevent changes to a run. |
| vdb-unlock | Removes the lock file to make the run available to be updated. |
| vdb-validate | Checks that the contents of a given object are self-consistent according to vdb specifications. |

## *4.1. Sequencing Platforms Excluded from SRA Lite Conversion*

Data from some sequencing platforms are excluded from conversion to SRA Lite format. These excluded platforms include:

- AB SOLID "colorspace run"
- COMPLETE GENOMICS
- PACBIO_SMRT "pacbio run"
- OXFORD_NANOPORE "ONT run"

Data from these platforms are excluded for two major reasons.

The first reason is that long read technologies from PacBio and Oxford Nanopore are not well represented in the algorithm used to determine overall read quality. Longer reads are more likely to have long strings of low-quality bases or to end with 10 or more bases under quality score value of 20. Future research may yield a conversion algorithm that converts these data to SRA Lite format without flagging a high number of

reads as “fail”. Until then, these long read platforms are not converted to SRA Lite format.

Additionally, the unique data characteristics of the two legacy platforms — Complete Genomics and AB SOLiD (Applied Biosystems Sequencing by Oligonucleotide Ligation and Detection) — make conversion more difficult. These legacy platforms are rarely seen in new data submissions, and SRA has no plans at this time to perform their conversion to SRA Lite.

## *4.2. make-read-filter*

The *make-read-filter* reads the quality scores and uses rules to create a read filter column. This is used for loaders that were applied prior to read filter value incorporation into the ETL process.

Reads that have more than half of quality score values < Phred 20 will be flagged ‘reject’.

Reads that begin or end with a run of more than 10 quality scores < Phred 20 are also flagged `reject`. The process stores the following reasons for read filtering in the associated metadata for the run:

- *low_quality_count* >50% of quality scores for this read were below Phred 20
- *low_quality_back* 10+ quality scores at the end of the read below Phred 20
- *low_quality_front* 10+ quality scores at the start of the read below Phred 20
- *original_filter* fastq file included a filter flag for this read in the original file

### 4.2.1. Filter Metadata Example

An example of the run SRR000001 that has gone through the SRA Lite conversion process is shown below. The *kdbmeta* program is used to display the metadata stored within the run to display what changes were made as part of the conversion to SRA Lite.

```
$ kdbmeta -u SRR000001 READ_FILTER_CHANGES

<READ_FILTER_CHANGES>
  <BACK_LOW_QUALITY_BASES>634322</BACK_LOW_QUALITY_BASES>
  <BACK_LOW_QUALITY_READS>3110</BACK_LOW_QUALITY_READS>
  <FILTERED_READS>6222</FILTERED_READS>
  <FRONT_LOW_QUALITY_BASES>0</FRONT_LOW_QUALITY_BASES>
  <FRONT_LOW_QUALITY_READS>0</FRONT_LOW_QUALITY_READS>
  <ORIGINAL_FILTERED_BASES>0</ORIGINAL_FILTERED_BASES>
  <ORIGINAL_FILTERED_READS>0</ORIGINAL_FILTERED_READS>
  <TOTAL_LOW_QUALITY_BASES>349298</TOTAL_LOW_QUALITY_BASES>
  <TOTAL_LOW_QUALITY_READS>3341</TOTAL_LOW_QUALITY_READS>
</READ_FILTER_CHANGES>
```

# 5. Custom Metadata

SRA stores metadata describing the data contents of the run and key information about the conversion process from the submitted data into the SRA format.

## *5.1. Schema*

The tables, columns, and format of the data in the columns for an SRA object are described by the schema. There are schemas defined for each major sequencing platform processed by SRA. The schemas can be found in the *ncbi-vdb* github repository with `.vschema` in their name. https://github.com/ncbi/ncbi-vdb/tree/master/interfaces/sra

Users can also use the *vdb-dump* tool to access the schema for a particular run with the following command using an SRA run accession:

```
vdb-dump -A <accession>
```

## *5.2. Metadata*

Each run contains metadata that includes information about that run, the loader used, and the vdb schema used to make the run. Run metadata can be viewed using the *kdbmeta* tool in the SRA Toolkit.

# 6. Virtual Database (VDB) Format

The **Virtual Database (VDB) format** is a structured and efficient system for storing biological information in a portable file format.

A VDB is a metadata container for components used to integrate data from multiple data sources, so that they can be accessed in an integrated manner through a single, uniform API.

VDB employs a schema to define tables and columns, functioning similarly to a relational database. This schema-based structure is supported by a set of libraries that enable data access and modification.

## *6.1. Introduction to VDB*

VDB is a library developed with the goal to efficiently store biological information. It is written in the C language (C89) and consists of approximately 410,000 lines of code across 2,000 files. Prebuilt binaries for x86 are available for Linux (AlmaLinux and Ubuntu), Windows, and MacOS, with both static and dynamic 64-bit libraries. Additionally, ARM64 prebuilt binaries are also available for MacOS.

VDB supports both read-only and read-write access, with language bindings for C++ and Python. The system employs Git for version control and CMake as its build system.

- Source Code: https://github.com/ncbi/ncbi-vdb
- Build and Installation Instructions: https://github.com/ncbi/ncbi-vdb/wiki/Building-and-Installing-from-Source

### 6.1.1. VDB System Overview

The VDB system has several notable characteristics:

- Uses **GCC** for Linux, **Clang** for macOS, and **MSVC** for Windows.
- Isolates platform-dependent code where necessary without relying on third-party solutions.
- Includes static copies of key third-party libraries:
    - bzip2
    - mbedtls
    - regex
    - zlib
    - zstd

### 6.1.2. VDB Dependencies

The following tools are required to compile VDB:

- **CMake** (version 3.16 or higher)
- **flex** (version 2.6 or higher)
- **bison** (version 3 or higher)

### 6.1.3. Key Features

VDB provides the following features:

- Data can be accessed locally or remotely via HTTPS.
- An optional encryption layer enhances data security.
- Limited index support is available for improved data retrieval.

## *6.2. VDB Structure*

### 6.2.1. Global Level

At the global level, VDB is composed of two primary layers: a **storage layer** (Key-Database, or KDB) and a **presentation layer** (Schema).

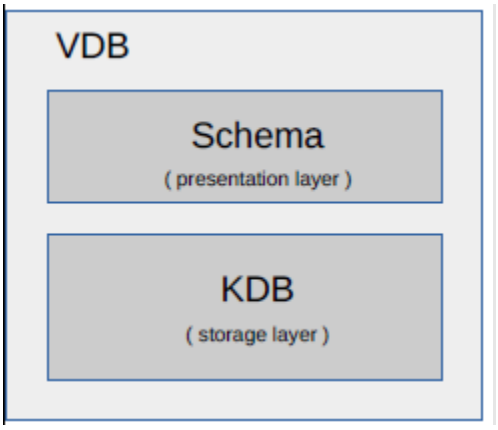


***Figure 6.*** *A representation of the structure of VDB, composed of a Key-Database (KDB) acting as the storage layer, and a Schema acting as the presentation layer.*

### 6.2.2. Storage Layer

The VDB storage layer uses a columnar layout within the **KDB** system. The storage structure follows this table organization:

- Each **table** is a container of **columns**.
- Each **column** is a sequence of **blobs**.
- Each **blob** contains at least one row of **data**, with the number of rows dependent on data size.

Blobs may be compressed, with each column using a different compression-algorithm for its blobs. The available compression algorithms are **GZIP**, **BZIP**, **ZSTD**, and integer-compression (a custom algorithm tailored for QUALITY values). In addition to compression, data can be pre-processed with a variety of schema-functions.

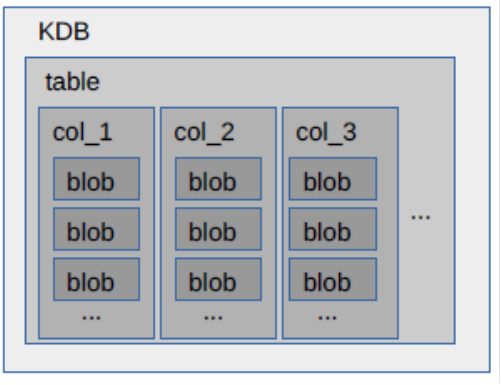


***Figure 7.*** *A representation of the structure of a Key-Database (KDB), composed of a table containing multiple columns, and each column containing a sequence of blobs. Each blob also contains data.*

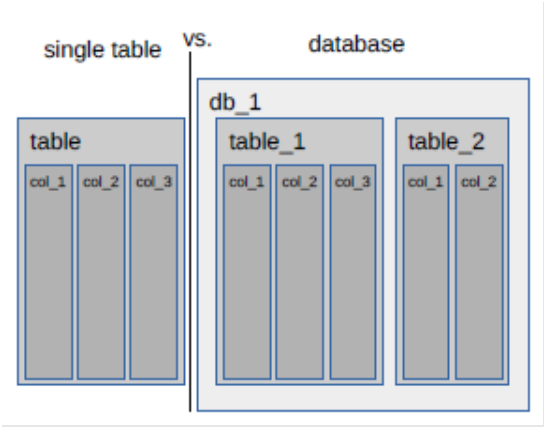


***Figure 8.*** *A VDB-object can be just a single table, or a database of tables.*

For each row in a table, a blob stores a cell. In the example presented in ***Fig. 9***, the number of cells in `blob #1` in `col_1` must match the number of cells in `blob #1` in `col_2` and all other columns in this table. The set of cells over all columns is a row.

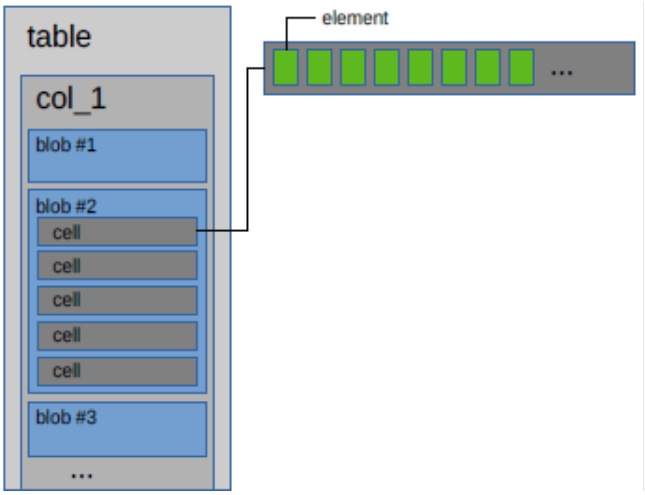


***Figure 9.*** *Blobs store cells in a column. Each blob will have at least one cell.*

Each row in a table corresponds to a cell stored within a blob. All cells in corresponding blobs across different columns align to form a complete row. Cells may contain between 0 and 2^32 bits of data.

While individual cells can vary in size, elements within the same column maintain uniform size. For instance, storing DNA bases in the 2na format (just A, C, T, G) requires only 2 bits per element. To allow for N-values, it is best to add a second column with 1-bit elements, and let the schema combine the 2 columns into human readable ASCII.

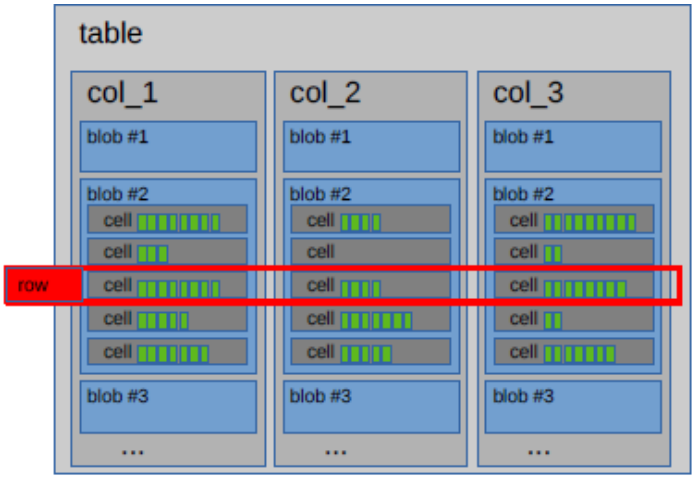


***Figure 10.*** *A row is the set of all cells across each column in a row.*

When a VDB object is created, data is stored in a structured directory layout (***Fig. 11***):

- **Table Structure:**
  - col - Contains physical column data.
  - idx - Provides limited index support.
  - md - Stores metadata.

- **Files in Table Structure:**
  - lock - Prevents accidental modifications.
  - md5 - Ensures data integrity with an MD5 checksum.

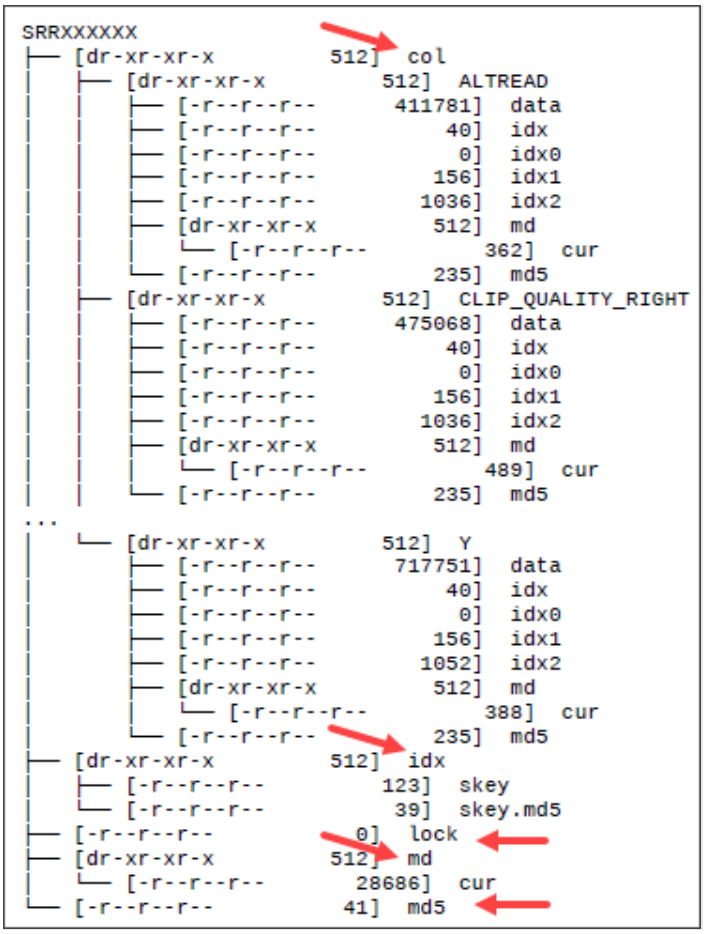


***Figure 11.*** *Directory layout for a VDB object.*

For each column in the table, the “col” directory contains the following layout (***Fig. 12***):

- data - Sequentially stored blobs.
- idx* - Binary dictionary mapping rows to data file offsets.
- md - Metadata storage for the column.
- md5 - MD5 checksums for data integrity.

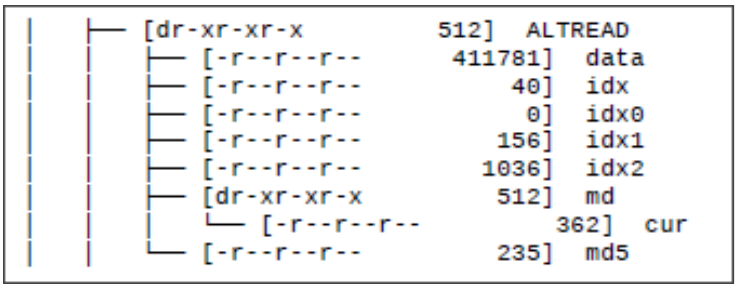


***Figure 12.*** *Directory layout of the “col” directory.*

This data structure can be combined (archived) into a single file (***Fig. 13***) containing the following information:

- **HDR** - Header information
- **TOC** - Table of contents listing file locations and lengths
- **FILES** - Sorted file data by size


***Figure 13.*** *A representation of the Single File Archive Directory layout of the “col” directory.*

## 6.2.3. Schema Layer

The **schema layer** is parsed when opening a database or table for read or write into a chain of productions. It plays a critical role in VDB by:

- Storing text-based rules in column, table, and database metadata
- Defining rules for writing and retrieving data
- Controlling compression algorithms for each column

The need for the schema language resulted from the many different formats of biological data arriving at the NIH National Library of Medicine. Instead of burdening tools with detailed knowledge about the different layouts of data used in the archive, it was decided to implement this flexibility in the schema layer.

The tools can now code against a stable layout without the need to change them whenever a new data layout is necessary. To demonstrate some of the capabilities of the schema, the Python bindings for the VDB library are used.

## 6.3. Example Scripts

### 6.3.1. Running a Simple Example Script

If the user wants to verify the following examples (on a Debian/Ubuntu machine) the following steps must be taken:

1) Ensure that *python3*, *git*, *cmake*, and *build-essentials* are installed.
2) Ensure that the read- and the write-version of the library is installed
3) In order to obtain the read- and write-versions of the library, it is necessary to checkout and build the source code from GitHub.

```
$sudo apt install git build-essential cmake python3
$git clone https://github.com/ncbi/ncbi-vdb.git
$cd ncbi-vdb
$git checkout engineering
$./configure
$make
$mkdir -p ~/.ncbi/lib64
$cp ~/ncbi-outdir/ncbi-vdb/linux/gcc/x86_64/rel/lib/libncbivdb.
so.3.0.4 ./libncbi-vdb.so
$cp ~/ncbi-outdir/ncbi-vdb/linux/gcc/x86_64/rel/lib/libncbiwvdb.
so.3.0.4 ./libncbi-wvdb.so
$mkdir -p ~/schema-doc
$cd ~/schema-doc
$cp ~/ncbi-vdb/py-vdb/vdb.py .
```

**Note:** Always use the newest version created by the build step.

4) In the schema-doc directory, create a python-file (`*schema-doc.py*`) with the following content:

```
#!/usr/bin/env python3
from vdb import *
try:
       with manager() as mgr :
            print( f"mgr.vers = {mgr.Version()}" )
except vdb_error as e :
      print( e )
```

You can run this file using the following commands:

```
$chmod +x schema-doc.py
$./schema-doc.py
```

5) If the steps above have been correctly entered, the following line will print:

```
mgr.vers = 2.7.49
```

In the future, the reported version might be higher.

6) After executing the steps above, a simple example-script (shown below) can be run. Draw your attention in particular to the code in the schema section, which is highlighted, as we will be modifying this section in the upcoming examples.

```
#!/usr/bin/env python3

import os, shutil
from vdb import *
def make_table( tbl_name ) :

    #SCHEMA SECTION
    schema = '''
        version 1;
        table T1 #1.0
        {
            column U8 C1;
            column U32 C2;
        }; '''

    if os.path.exists( tbl_name ) :
        shutil.rmtree( tbl_name )
    wmgr = manager( OpenMode.Write )
    s = wmgr.MakeSchema( schema )
    tbl = wmgr.CreateTable( s, "T1", tbl_name )
    cur = tbl.CreateCursor( OpenMode.Write )
    cols = cur.OpenColumns( [ "C1", "C2" ] )
    for idx in range( 0, 3 ) :
        cur.OpenRow()
        cols[ "C1" ].write_rand( idx+2, 100 )
        cols[ "C2" ].write_rand( 1, 100000 )
        cur.CommitRow()
        cur.CloseRow()
    cur.Commit()
try:
    make_table( "e1" )
    manager().OpenTable( "e1" ).print_rows()
except vdb_error
    print( e )
```

7) Store this script as `*e1.py*` and execute it, which will then create a vdb-table as a directory. This vdb-table has 2 columns: C1 and C2.C1 is of the type U8 (an 8-bit unsigned integer) and C2 is of type U32 (a 32-bit unsigned integer). The table will have 3 rows. The column C1 will have 2, 3, and 4 random elements in the range of 1-100. The column C2 will have 1 random value in the range of 1.100,00. Notice that the highlighted name of the table in the schema (`"T1"`) must be used in `CreateTable()`.

8) If the sra-toolkit is installed, the created vdb-object can be inspected with the `$vdb-dump ./e1` command.

### 6.3.2. Example Script - Compression and Computed Column

The next script demonstrates compression and a computed column. The schema section is highlighted, as in the previous example. Three lines of code beginning with `s = wmgr.MakeSchema()` are necessary for the encoding function.

```
#!/usr/bin/env python3

import os, shutil
from vdb import *

def make_table( tbl_name ) :

    #SCHEMA SECTION
    schema = '''
        version 1;
        include 'vdb/vdb.vschema';
        table T1 #1.0
        {
            column < U8 > zip_encoding C1;
            readonly column U32 C1L = row_len( C1 );
        }; '''

    if os.path.exists( tbl_name ) :
        shutil.rmtree( tbl_name )
    wmgr = manager( OpenMode.Write )

    s = wmgr.MakeSchema()
    s.AddIncludePath( "../ncbi-vdb/interfaces" )
    s.ParseText( schema )

    tbl = wmgr.CreateTable( s, "T1", tbl_name )
    cur = tbl.CreateCursor( OpenMode.Write )
    cols = cur.OpenColumns( [ "C1" ] )
    for idx in range( 0, 3 ) :
        cur.OpenRow()
        N = random.randint( 5, 10 );
        cols[ "C1" ].write_rand( N, 100 )
        cur.CommitRow()
        cur.CloseRow()
    cur.Commit()
try:
    make_table( "e2" )
    manager().OpenTable( "e2" ).print_rows()
except vdb_error as e :
    print( e )
```

Store this script as `*e2.py*` and execute it.

This second table has 2 columns: C1 and C1L. C1 is of the type U8 and stored as compressed blobs. C2 is of type U32 and not stored at all, it is computed while reading.

### 6.3.3. Example Script - Tables can inherit Columns from other Tables

This script demonstrates that tables can inherit columns from other tables:

```
#!/usr/bin/env python3

import os, shutil
from vdb import *

def make_table( tbl_name ) :

    #SCHEMA SECTION
    schema = '''
        version 1;
        include 'vdb/vdb.vschema';
        table T1 #1.0
        {
            column < U8 > zip_encoding C1;
            readonly column U32 C1L = row_len( C1 );
        };
        table T2 #1.0 =T1 #1.0
        {
            column ascii C2;
        }; '''

    if os.path.exists( tbl_name ) :
        shutil.rmtree( tbl_name )
    wmgr = manager( OpenMode.Write )
    s = wmgr.MakeSchema()
    s.AddIncludePath( "../ncbi-vdb/interfaces" )
    s.ParseText( schema )
    tbl = wmgr.CreateTable( s, "T1", tbl_name )
    cur = tbl.CreateCursor( OpenMode.Write )
    cols = cur.OpenColumns( [ "C1", "C2" ] )
    for idx in range( 0, 3 ) :
        cur.OpenRow()
        N = random.randint( 5, 10 );
        cols[ "C1" ].write_rand( N, 100 )
        Cols[ "C2" ].write( f"row #{idx} N={N}" )
        cur.CommitRow()
        cur.CloseRow()
    cur.Commit()
try:
    make_table( "e3" )
    manager().OpenTable( "e3" ).print_rows()
except vdb_error as e :
    print ( e )
```

Store this script as `*e3.py*` and execute it.

The Table T2 inherits 2 columns from Table T1. The table from which it inherits columns can also be included from a different schema file.

### 6.3.4. Example Script – Using echo

This script shows the usage of a different schema-function, `echo`:

```
#!/usr/bin/env python3

import os, shutil
from vdb import *
def make_table( tbl_name ) :

    #SCHEMA SECTION
    schema = '''
        version 1;
        include 'vdb/vdb.vschema';
        table T1 #1.0
        {
            column < U8 > zip_encoding C1;
            readonly column utf8 hello
            = <utf8> echo <" world">();
        };'''

    if os.path.exists( tbl_name ) :
        shutil.rmtree( tbl_name )
    wmgr = manager( OpenMode.Write )
    s = wmgr.MakeSchema()
    s.AddIncludePath( "../ncbi-vdb/interfaces" )
    s.ParseText( schema )
    tbl = wmgr.CreateTable( s, "T1", tbl_name )
    cur = tbl.CreateCursor( OpenMode.Write )
    cols = cur.OpenColumns( [ "C1" ] )
    for idx in range( 0, 3 ) :
        cur.OpenRow()
        cols[ "C1" ].write_rand( 2, 100 )
        cur.CommitRow()
        cur.CloseRow()
    cur.Commit()
try:
    make_table( "e4" )
    manager().OpenTable( "e4" ).print_rows()
except vdb_error as e :
    print( e )
```

Store this script as `*e4.py*` and execute it.

The read-only column `hello` does not refer to any other column; it just produces text for each row in the table. If column C1 were not populated, the table would be empty and no rows would be produced while reading.

### 6.3.5. Example Script - Schema Mapping, Encoding, and Validation

This script uses a more complicated pattern:

```
#!/usr/bin/env python3

import os, shutil
from vdb import *
def make_table( tbl_name ) :

    #SCHEMA SECTION
    schema = '''
        version 1;
        include 'vdb/vdb.vschema';
        table T1 #1.0
        {
            extern column ascii C1 {
            read = out_c1;
            validate=< ascii >compare( in_c1_upper, out_c1 );
        }
        ascii in_c1_upper =
            < ascii,ascii > map < "abcd", "ABCD" >( C1 );
        U8 in_c1_bin =
            < ascii, U8 > map < "ABCD", [0,1,2,3] >( in_c1_upper);
        physical column <U8> zip_encoding .C1 = in_c1_bin;
        ascii out_c1 =
            < U8, ascii > map < [0,1,2,3], "ABCD" >( .C1 );
        };'''

    if os.path.exists( tbl_name ) :
        shutil.rmtree( tbl_name )
    wmgr = manager( OpenMode.Write )
    s = wmgr.MakeSchema()
    s.AddIncludePath( "../ncbi-vdb/interfaces" )
    s.ParseText( schema )
    tbl = wmgr.CreateTable( s, "T1", tbl_name )
    cur = tbl.CreateCursor( OpenMode.Write )
    cols = cur.OpenColumns( [ "C1" ] )
    for idx in range( 0, 3 ) :
        cur.OpenRow()
        cols[ "C1" ].write( "abcdABCD" )
        cur.CommitRow()
        cur.CloseRow()
    cur.Commit()
try:
    make_table( "e5" )
    manager().OpenTable( "e5" ).print_rows()
except vdb_error as e :
    print( e )
```

Store this script as `*e5.py*` and execute it.

- The schema describes a column C1. It takes ASCII-text when writing to it and produces ASCII-text when reading from it. Internally it stores the values as zip-encoded unsigned 8-bit values. It also converts the input to all uppercase. Only text containing characters out of the set `"abcdABCD"` are allowed. If a string of `"abcde"` would be written to the column, the script would terminate and print an error.
- To perform the conversion to uppercase, the schema uses the map function. The same function is also used to translate ASCII-text to binary numbers.
- This schema uses temporary productions like `'in_c1_upper'`, `'in_c1_bin'`, and `'out_c1'`. For instance, the temporary production `'out_c1'` is used twice: in the read as well as in the validate production. Because of the conversion to uppercase, the validate function has to be overwritten. Otherwise, writing data would fail since the vdb-library always performs a comparison between the data that has just been written and the given input.

### 6.3.6. Example Script - Usage of a Database

This script demonstrates the usage of a database:

```
#!/usr/bin/env python3
import os, shutil
from vdb import *
def fill_table( tbl, num_rows, num_elements ) :
    cur = tbl.CreateCursor( OpenMode.Write )
    cols = cur.OpenColumns( [ "C1" ] )
    for idx in range( 0, num_rows ) :
        cur.OpenRow()
        cols[ "C1" ].write_rand( num_elements, 100 )
        cur.CommitRow()
        cur.CloseRow()
    cur.Commit()
def make_database( db_name ) :

    #SCHEMA SECTION
    schema = '''
        version 1;
        table T1 #1.0 { column U8 C1; };
        table T2 #1.0 { column U32 C1; };
        database DB1 #1.0 {
            table T1 #1.0 t1;
            table T2 #1.0 t2;
        }; '''

    if os.path.exists( db_name ) :
        shutil.rmtree( db_name )
    with manager( OpenMode.Write ) as wmgr :
        s = wmgr.MakeSchema( schema )
        with wmgr.CreateDB( s, "DB1", db_name ) as db :
            fill_table( db.CreateTable( "t1" ), 4, 5 )
            fill_table( db.CreateTable( "t2" ), 5, 3 )
try:
    make_database( "e6" )
    with manager().OpenDB( "e6" ) as db :
        db.OpenTable( "t1" ).print_rows()
        db.OpenTable( "t2" ).print_rows()
except vdb_error as e :
    print( e )
```

Store this script as `*e6.py*` and execute it.

The schema defines a database DB1, which contains two tables T1 and T2, which are defined in the same schema above. The table-definitions could also be included from another schema. Both tables have a column C1, but in the table T1 this column has the type U8 and in the table T2 it has the type U32. They are independent of each other.

### 6.3.7. Example Script - Produce FASTQ

This script demonstrates a simple way to produce FASTQ from an SRA-accession:

```
#!/usr/bin/env python3
import os, shutil
from vdb import *
QNAME = "(INSDC:quality:text:phred_33)QUALITY"
def fastq( name, read, qual ) :
    print( f"@{name}\n{read}\n+{name}\n{qual}")
def whole_spots( cur, count ) :
    cols = cur.OpenColumns( [ "NAME", "READ", QNAME ] )
    for row_id, _ in zip( cols[ "READ" ].range(), range( count ) ) :
        fastq( cols[ "NAME" ].Read( row_id ),
            cols[ "READ" ].Read( row_id ),
            cols[ QNAME ].Read( row_id ) )
def split_spots( cur, count ) :
    cols = cur.OpenColumns( [ "NAME", "READ", QNAME,
        "READ_START", "READ_LEN" ] )
    for row_id, _ in zip( cols[ "READ" ].range(), range( count ) ) :
        name = cols[ "NAME" ].Read( row_id )
        rd = cols[ "READ" ].Read( row_id )
        q = cols[ QNAME ].Read( row_id )
        rs = cols[ "READ_START" ].Read( row_id )
        rl = cols[ "READ_LEN" ].Read( row_id )
        for i, ( st, cnt ) in enumerate( zip( rs, rl ) ) :
            if cnt > 0 :
                R = rd[ st : st + cnt ]
                Q = q[ st : st + cnt ]
                fastq( f"{name}.{i}", R, Q )
try:
    ACC = "SRR000001"
    with manager().OpenTable( ACC ) as tbl :
            whole_spots( tbl.CreateCursor(), 10 )
            print( '-'*70 )
            split_spots( tbl.CreateCursor(), 10 )
except vdb_error as e :
    print( e )
```

No schema needs to be created while reading data. The schema is stored in the VDB-object and used implicitly when reading data from it.

# 7. Appendix

### 7.1.1. Example schema: vdb-dump

The following is an example of a schema using `vdb-dump`, which outputs the native VDB format of SRA data. To output the schema of a run the following command is used.

```
vdb-dump -A <accession>
```

The schema output for an SRA run will look like the following.

```
version 1;

typedef I64 vdb:row_id_range [ 2 ];
typedef ascii INSDC:dna:text;
alias INSDC:dna:text INSDC:fasta;
typedef U8 INSDC:4na:bin;
typedef B1 INSDC:4na:packed [ 4 ];
typedef U8 INSDC:2na:bin;
typedef U8 INSDC:x2na:bin;
typedef B1 INSDC:2na:packed [ 2 ];
alias INSDC:2na:packed INSDC:dna:2na;
alias INSDC:2na:packed NCBI:2na;
typedef ascii INSDC:color:text;
typedef U8 INSDC:2cs:bin;
typedef U8 INSDC:x2cs:bin;
typedef B1 INSDC:2cs:packed [ 2 ];
alias INSDC:2cs:packed INSDC:color:2cs;
alias INSDC:2cs:packed NCBI:2cs;
typedef U8 INSDC:quality:phred;
alias INSDC:quality:phred NCBI:qual1;
typedef I8 INSDC:quality:log_odds;
typedef ascii INSDC:quality:text:phred_33;
typedef ascii INSDC:quality:text:phred_64;
typedef I32 INSDC:coord:val;
typedef U32 INSDC:coord:len;
typedef INSDC:coord:val INSDC:coord:zero;
typedef INSDC:coord:val INSDC:coord:one;
typedef U8 INSDC:SRA:read_filter;
alias INSDC:SRA:read_filter NCBI:SRA:read_filter;
typedef U8 INSDC:SRA:spot_filter;
typedef U8 INSDC:SRA:xread_type;
typedef INSDC:SRA:xread_type INSDC:SRA:read_type;
alias INSDC:SRA:read_type NCBI:SRA:read_type;
typedef U32 INSDC:SRA:spotid_t;
typedef U64 INSDC:SRA:spot_ids_found [ 4 ];
typedef U8 INSDC:SRA:platform_id;
alias INSDC:SRA:platform_id NCBI:SRA:platform_id;
typedef U16 NCBI:SRA:Segment [ 2 ];
typedef B8 NCBI:SRA:SpotDesc [ 16 ];
typedef B8 NCBI:SRA:ReadDesc [ 80 ];
typedef U32 NCBI:align:ploidy;
typedef U8 NCBI:align:ro_type;
```

```
typeset text_set { utf8, utf16, utf32, ascii };
typeset text8_set { utf8, ascii };
typeset pack_set { B8, B16, B32, B64, U8, U16, U32, U64, I8, I16, I32, I64 };
typeset izip_set { U8, U16, U32, U64, I8, I16, I32, I64 };
typeset NCBI:SRA:stats:qual_type { INSDC:quality:phred,
INSDC:quality:log_odds, INSDC:quality:log_odds [ 4 ] };
typeset NCBI:spot_filter_read_set { INSDC:4na:bin, INSDC:2na:bin,
INSDC:x2na:bin };
fmtdef izip_fmt;
fmtdef zlib_fmt;
extern function any cast #1 ( any in ) = vdb:cast;
extern function < type T > T bit_or #1 < U8 align > ( T A, T B ) =
vdb:bit_or;
extern function < type T > T trim #1 < U8 align, T val > ( T A ) = vdb:trim;
extern function I64 row_id #1 () = vdb:row_id;
extern function U32 row_len #1 ( any in ) = vdb:row_len;
extern function U32 fixed_row_len #1 ( any in ) = vdb:fixed_row_len;
validate function < type T > void compare #1 < * U32 sig_bits > ( T src, T
cmp ) = vdb:compare;
extern function < type T > T range_validate #1 < T lower, T upper > ( T in )
= vdb:range_validate;
extern function < type T > T is_configuration_set #1 < ascii node, ascii
value > ( T target ) = vdb:is_configuration_set;
extern function < type T > T meta:read #1 < ascii node * bool deterministic >
();
extern function < type T > T meta:value #1 < ascii node * bool deterministic
> ();
extern function text8_set idx:text:project #1.1 < ascii index_name * U8
case_sensitivity > ( * text8_set substitute );
extern function text8_set idx:text:insert #1.1 < ascii index_name * U8
case_sensitivity > ( text8_set key );
extern function vdb:row_id_range idx:text:lookup #1.1 < ascii index_name,
ascii query_by_name * U8 case_sensitivity > ();
extern function < type T > T echo #1 < T val > ( * any row_len ) = vdb:echo;
extern function < type A, type B > B map #1 < A from, B to > ( A in * B src )
= vdb:map;
extern function < type T > T clip #1 < T lower, T upper > ( T in ) =
vdb:clip;
extern function < type T > T sum #1 < * T k > ( T a, ... ) = vdb:sum;
extern function < type T > T diff #1 < * T k > ( T a * T b ) = vdb:diff;
extern function < type T > T add_row_id #1 ( T in ) = vdb:add_row_id;
extern function < type T > T sub_row_id #1 ( T in ) = vdb:sub_row_id;
extern function < type T > T [ * ] cut #1 < U32 idx, ... > ( T [ * ] in ) =
vdb:cut;
extern function < type T > T [ * ] paste #1 ( T [ * ] in, ... ) = vdb:paste;
extern function B1 [ * ] pack #1 ( pack_set in ) = vdb:pack;
extern function pack_set unpack #1 ( B1 [ * ] in ) = vdb:unpack;
extern function izip_fmt izip #2.1 ( izip_set in ) = vdb:izip;
extern function izip_set iunzip #2.1 ( izip_fmt in ) = vdb:iunzip;
extern function zlib_fmt zip #1 < * I32 strategy, I32 level > ( any in ) =
vdb:zip;
extern function any unzip #1 ( zlib_fmt in ) = vdb:unzip;
extern function < type T > T simple_sub_select #1 < ascii tbl, ascii col > (
I64 row * I32 idx ) = vdb:simple_sub_select_1;
extern function text_set sprintf #1 < ascii fmt > ( any p1, ... ) =
vdb:sprintf;
extern function INSDC:2na:bin INSDC:SEQ:rand_4na_2na #1 ( INSDC:4na:bin
```

```
rd_bin );
extern function ascii INSDC:SRA:format_spot_name #1 ( ascii name_fmt, I32 X,
I32 Y * ascii spot_name );
extern function ascii INSDC:SRA:format_spot_name_no_coord #1 ( ascii name_fmt
* ascii spot_name );
extern function INSDC:SRA:read_filter INSDC:SRA:spot2read_filter #1 (
INSDC:SRA:spot_filter out_spot_filter, INSDC:SRA:xread_type out_read_type );
extern function INSDC:SRA:spot_filter INSDC:SRA:read2spot_filter #1 (
INSDC:SRA:read_filter out_read_filter );
extern function U8 NCBI:SRA:stats_trigger #1 ( U8 read_bin, U32 read_len,
INSDC:SRA:xread_type read_type * ascii spot_group );
extern function U8 NCBI:SRA:cmp_stats_trigger #1 ( B8 cmp_read_bin,
NCBI:SRA:stats:qual_type qual_bin, U32 read_len, INSDC:SRA:xread_type
read_type * ascii spot_group );
extern function U8 NCBI:SRA:cmpf_stats_trigger #1 ( B8 cmp_read_bin, U32
spot_len, U32 read_len, INSDC:SRA:xread_type read_type * ascii spot_group );
extern function U8 NCBI:SRA:cmpb_stats_trigger #1 ( B8 cmp_read_bin * ascii
spot_group );
extern function U8 NCBI:SRA:readlen_stats_trigger #1 ( U32 read_len,
INSDC:SRA:xread_type read_type );
extern function U8 NCBI:SRA:phred_stats_trigger #1 ( INSDC:quality:phred
qual_bin );
extern function INSDC:SRA:spot_filter NCBI:SRA:make_spot_filter #1 < * U32
min_length, U8 min_quality, U8 no_quality > ( NCBI:spot_filter_read_set
bin_read, INSDC:quality:phred quality, INSDC:coord:zero read_start,
INSDC:coord:len read_len, INSDC:SRA:xread_type read_type,
INSDC:SRA:spot_filter spot_filter );
extern function INSDC:quality:phred NCBI:SRA:syn_quality #1 <
INSDC:quality:phred good_quality, INSDC:quality:phred bad_quality > (
INSDC:coord:len read_len, INSDC:SRA:spot_filter spot_filter );
extern function INSDC:quality:phred NCBI:SRA:syn_quality_read #1 <
INSDC:quality:phred good_quality, INSDC:quality:phred bad_quality > (
INSDC:coord:zero read_start, INSDC:coord:len read_len, INSDC:SRA:xread_type
read_type, INSDC:SRA:read_filter read_filter );
extern function INSDC:x2cs:bin NCBI:color_from_dna #1.1 ( INSDC:x2na:bin
bin_x2na, INSDC:coord:zero read_start, INSDC:coord:len read_len,
INSDC:dna:text cs_key, U8 color_matrix );
extern function INSDC:dna:text NCBI:SRA:setRnaFlag #1 ( INSDC:dna:text
in_read );
extern function INSDC:dna:text NCBI:SRA:useRnaFlag #1 ( INSDC:dna:text
in_read );
extern function INSDC:coord:len [ 2 ] NCBI:SRA:fix_read_seg #1 ( U16 [ 2 ]
rd_seg, INSDC:coord:len spot_len );
extern function NCBI:SRA:SpotDesc NCBI:SRA:make_spot_desc #1 (
INSDC:coord:len spot_len, INSDC:coord:len fixed_len, INSDC:coord:len sig_len,
INSDC:coord:zero trim_start, INSDC:coord:len trim_len, U8 num_reads );
extern function NCBI:SRA:ReadDesc NCBI:SRA:make_read_desc #1 ( U8 num_reads,
INSDC:coord:zero read_start, INSDC:coord:len read_len, INSDC:SRA:xread_type
read_type, INSDC:SRA:read_filter read_filt, INSDC:dna:text cs_key,
INSDC:coord:zero label_start, INSDC:coord:len label_len, ascii label );
extern function < type T > T NCBI:align:cigar #2 < U8 ctype > ( bool
has_mismatch, bool has_ref_offset, I32 ref_offset, INSDC:coord:len read_len *
INSDC:coord:len ref_len, NCBI:align:ro_type ref_offset_type ) =
ALIGN:cigar_2;
extern function U32 NCBI:align:edit_distance #1 ( bool has_mismatch, bool
has_ref_offset, I32 ref_offset );
extern function U32 NCBI:align:edit_distance #2 ( bool has_mismatch, bool
```

```
has_ref_offset, I32 ref_offset, INSDC:coord:len ref_len * INSDC:coord:len
read_len ) = NCBI:align:edit_distance_2;
extern function U32 NCBI:align:edit_distance #3 ( bool has_mismatch, bool
has_ref_offset, I32 ref_offset, NCBI:align:ro_type ref_offset_type,
INSDC:coord:len read_len ) = NCBI:align:edit_distance_3;
extern function ascii NCBI:align:rna_orientation #1 ( NCBI:align:ro_type
ref_offset_type );
extern function < type T > T NCBI:align:project_from_sequence #1 < ascii col
> ( I64 seq_spot_id, INSDC:coord:one seq_read_id ) =
ALIGN:project_from_sequence;
extern function INSDC:4na:bin NCBI:align:align_restore_read #1 (
INSDC:4na:bin ref_read, bool has_mismatch, INSDC:4na:bin mismatch, bool
has_ref_offset, I32 ref_offset * INSDC:coord:len read_len ) =
ALIGN:align_restore_read;
extern function INSDC:4na:bin NCBI:align:raw_restore_read #1 ( INSDC:4na:bin
align_read, bool ref_orientation ) = ALIGN:raw_restore_read;
extern function INSDC:quality:phred NCBI:align:raw_restore_qual #1 (
INSDC:quality:phred align_qual, bool ref_orientation );
extern function INSDC:4na:bin NCBI:align:ref_sub_select #1 ( I64 id,
INSDC:coord:zero start, INSDC:coord:len len * U32 ref_ploidy ) =
ALIGN:ref_sub_select;
extern function INSDC:4na:bin NCBI:align:ref_restore_read #1 ( INSDC:4na:bin
cmp_rd, ascii seq_id, INSDC:coord:one seq_start, INSDC:coord:len seq_len ) =
ALIGN:ref_restore_read;
extern function INSDC:4na:bin NCBI:align:seq_restore_read #1 ( INSDC:4na:bin
cmp_rd, I64 align_id, INSDC:coord:len read_len, INSDC:SRA:xread_type rd_type
) = ALIGN:seq_restore_read;
extern function ascii NCBI:align:seq_restore_linkage_group #1 ( ascii
cmp_linkage_group, I64 align_id ) = ALIGN:seq_restore_linkage_group;
extern function bool NCBI:align:generate_has_mismatch #1 ( INSDC:4na:bin
reference, INSDC:4na:bin subject, bool has_ref_offset, I32 ref_offset ) =
ALIGN:generate_has_mismatch;
extern function INSDC:4na:bin NCBI:align:generate_mismatch #1 ( INSDC:4na:bin
reference, INSDC:4na:bin subject, bool has_ref_offset, I32 ref_offset ) =
ALIGN:generate_mismatch;
extern function INSDC:coord:zero NCBI:align:ref_pos #1 ( I64 ref_id,
INSDC:coord:zero ref_start );
extern function ascii NCBI:align:ref_name #1 ( I64 ref_id );
extern function ascii NCBI:align:ref_seq_id #1 ( I64 ref_id );
extern function I64 NCBI:align:local_ref_id #1 ( U64 global_ref_start );
extern function INSDC:coord:zero NCBI:align:local_ref_start #1 ( U64
global_ref_start );
extern function I32 NCBI:align:template_len #1 ( INSDC:coord:zero pos,
INSDC:coord:zero mate_pos, INSDC:coord:len reflen, INSDC:coord:len
mate_reflen, ascii ref_name, ascii mate_ref_name, INSDC:coord:one read_id );
extern function U32 NCBI:align:get_sam_flags #1 ( INSDC:coord:len read_len,
INSDC:coord:one read_id, I32 template_len, bool strand, bool mate_strand,
bool is_secondary * INSDC:SRA:read_filter filter );
extern function U32 NCBI:align:get_sam_flags #2 ( I64 mate_id,
INSDC:coord:one read_id, I32 template_len, bool strand, bool mate_strand,
bool is_secondary * INSDC:SRA:read_filter filter ) =
NCBI:align:get_sam_flags_2;
extern function INSDC:coord:len NCBI:align:get_left_soft_clip #2 ( bool
has_ref_offset, I32 ref_offset, INSDC:coord:len read_len ) =
NCBI:align:get_left_soft_clip_2;
extern function INSDC:coord:len NCBI:align:get_right_soft_clip #2 ( bool
has_mismatch, INSDC:coord:len left_clip, bool has_ref_offset, I32 ref_offset
```

```
) = NCBI:align:get_right_soft_clip_2;
extern function INSDC:coord:len NCBI:align:get_right_soft_clip #3 ( bool
has_ref_offset, I32 ref_offset, INSDC:coord:len ref_len ) =
NCBI:align:get_right_soft_clip_3;
extern function INSDC:coord:len NCBI:align:get_right_soft_clip #4 ( bool
has_ref_offset, I32 ref_offset, INSDC:coord:len read_len, INSDC:coord:len
ref_len ) = NCBI:align:get_right_soft_clip_4;
extern function INSDC:coord:len NCBI:align:get_right_soft_clip #5 ( bool
has_ref_offset, I32 ref_offset, NCBI:align:ro_type ref_offset_type,
INSDC:coord:len read_len ) = NCBI:align:get_right_soft_clip_5;
extern function < type T > T NCBI:align:get_clipped_cigar #2 ( ascii cigar,
INSDC:coord:len cigar_len ) = NCBI:align:get_clipped_cigar_2;
extern function I32 NCBI:align:get_clipped_ref_offset #1 ( bool
has_ref_offset, I32 ref_offset );
extern function < type T > T NCBI:align:clip #1 ( T object, INSDC:coord:len
left_clip, INSDC:coord:len right_clip );
extern function < type T > T NCBI:align:clip #2 ( T object, INSDC:coord:len
read_len, INSDC:coord:len left_clip, INSDC:coord:len right_clip ) =
NCBI:align:clip_2;
extern function INSDC:coord:len NCBI:align:get_ref_len #1 ( bool
has_ref_offset, I32 ref_offset * INSDC:coord:len right_clip );
extern function INSDC:coord:len NCBI:align:get_ref_len_2 #2 ( bool
has_ref_offset, I32 ref_offset ) = NCBI:align:get_ref_len_2;
extern function ascii NCBI:align:get_mismatch_read #1 ( bool has_mismatch,
INSDC:dna:text mismatch );
extern function bool NCBI:align:get_ref_mismatch #1 ( bool has_mismatch, bool
has_ref_offset, I32 ref_offset, INSDC:coord:len ref_len );
extern function bool NCBI:align:get_ref_insert #1 ( bool has_mismatch, bool
has_ref_offset, I32 ref_offset, INSDC:coord:len ref_len );
extern function bool NCBI:align:get_ref_delete #1 ( bool has_mismatch, bool
has_ref_offset, I32 ref_offset, INSDC:coord:len ref_len );
extern function I64 NCBI:align:get_mate_align_id #1 ( I64 spot_id );
physical < type T > T izip_encoding #1

{
        encode
        { return izip #2.1 ( @ ); }
        decode
        { return ( T ) iunzip #2.1 ( @ ); }
}

physical < type T > T zip_encoding #1 < * I32 strategy, I32 level >
{
        encode
        { return zip #1 < strategy, level > ( @ ); }
        decode
        { return unzip #1 ( @ ); }
}

physical bool bool_encoding #1
{
        encode
        {
                U8 lim = < U8 > clip #1 < 0, 1 > ( @ );
                B1 bit = pack #1 ( lim );
                return zip #1 < 3, 1 > ( bit );
        }
```

```
        decode
        {
                B1 bit = unzip #1 ( @ );
                return ( bool ) unpack #1 ( bit );
        }
}

table INSDC:tbl:sequence #1.0.1
{
        default column INSDC:dna:text READ
        {
                read = out_dna_text;
                validate = < INSDC:dna:text > compare #1 ( in_dna_text,
out_dna_text );
        }

        column INSDC:4na:bin READ = out_4na_bin;
        column INSDC:4na:packed READ = out_4na_packed;
        column INSDC:x2na:bin READ = out_x2na_bin;
        column INSDC:2na:bin READ = out_2na_bin;
        column INSDC:2na:packed READ = out_2na_packed;
        default column INSDC:color:text CSREAD

        {
                read = out_color_text;
                validate = < INSDC:color:text > compare #1 ( in_color_text,
out_color_text );
        }

        column INSDC:x2cs:bin CSREAD = out_x2cs_bin;
        column INSDC:2cs:bin CSREAD = out_2cs_bin;
        column INSDC:2cs:packed CSREAD = out_2cs_packed;
        readonly column bool CS_NATIVE = cs_native;
        column INSDC:dna:text CS_KEY

        {
                read = out_cs_key;
                validate = < INSDC:dna:text > compare #1 ( in_cs_key,
out_cs_key );
        }

        column U8 COLOR_MATRIX = out_color_matrix;
        default column INSDC:quality:phred QUALITY

        {
                read = out_qual_phred;
                validate = < INSDC:quality:phred > compare #1 (
in_qual_phred, phys_qual_phred );
        }

        column INSDC:quality:text:phred_33 QUALITY = out_qual_text_phred_33 |
( INSDC:quality:text:phred_33 ) < B8 > sum #1 < 33 > ( out_qual_phred );
        column INSDC:quality:text:phred_64 QUALITY = out_qual_text_phred_64 |
( INSDC:quality:text:phred_64 ) < B8 > sum #1 < 64 > ( out_qual_phred );
        INSDC:coord:len signal_len = ( INSDC:coord:len ) row_len #1 (
out_signal ) | < INSDC:coord:len > echo #1 < 0 > ();
```

```
}

table INSDC:SRA:tbl:spotcoord #1
{
        default column INSDC:coord:val X = out_x_coord;
        default column INSDC:coord:val Y = out_y_coord;
        readonly column U16 X = cast #1 ( x_clip_U16 );
        readonly column U16 Y = cast #1 ( y_clip_U16 );
        INSDC:coord:val x_clip_U16 = < INSDC:coord:val > clip #1 < 0, 65535 >
( out_x_coord );
        INSDC:coord:val y_clip_U16 = < INSDC:coord:val > clip #1 < 0, 65535 >
( out_y_coord );
}

table INSDC:SRA:tbl:spotname #1.1 = INSDC:SRA:tbl:spotcoord #1
{
        column ascii NAME = _out_name;
        readonly column INSDC:SRA:spot_ids_found SPOT_IDS_FOUND =
spot_ids_found;
        ascii _out_name = INSDC:SRA:format_spot_name #1 ( out_name_fmt,
out_x_coord, out_y_coord, out_spot_name ) | INSDC:SRA:format_spot_name #1 (
out_name_fmt, out_x_coord, out_y_coord ) |
INSDC:SRA:format_spot_name_no_coord #1 ( out_name_fmt ) | out_spot_name |
out_trace_name;
}

table INSDC:SRA:tbl:spotdesc #1.0.2 = INSDC:tbl:sequence #1.0.1
{
        column U8 NREADS = out_nreads;
        readonly column INSDC:coord:len SPOT_LEN = spot_len;
        readonly column INSDC:coord:len FIXED_SPOT_LEN = fixed_spot_len;
        readonly column INSDC:coord:zero TRIM_START = trim_start | <
INSDC:coord:zero > echo #1 < 0 > ();
        readonly column INSDC:coord:one TRIM_START = ( INSDC:coord:one ) <
I32 > sum #1 < 1 > ( trim_start ) | < INSDC:coord:one > echo #1 < 1 > ();
        readonly column INSDC:coord:len TRIM_LEN = trim_len | spot_len;
        column ascii LABEL = out_label;
        column INSDC:coord:zero LABEL_START = out_label_start;
        column INSDC:coord:len LABEL_LEN = out_label_len;
        readonly column U16 LABEL_START = cast #1 ( out_label_start );
        readonly column U16 LABEL_LEN = cast #1 ( out_label_len );
        default column INSDC:SRA:xread_type READ_TYPE = out_read_type;
        readonly column INSDC:SRA:read_type READ_TYPE = out_read_type | <
INSDC:SRA:xread_type, INSDC:SRA:read_type > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7
], [ 0, 1, 0, 1, 0, 1, 0, 1 ] > ( out_read_type );
        default column INSDC:coord:zero READ_START = out_read_start;
        column INSDC:coord:one READ_START = ( INSDC:coord:one ) < I32 > sum
#1 < 1 > ( out_read_start );
        column INSDC:coord:len READ_LEN = out_read_len;
        readonly column U16 READ_START = cast #1 ( out_read_start );
        readonly column U16 READ_LEN = cast #1 ( out_read_len );
        column INSDC:SRA:read_filter READ_FILTER = out_rd_filter |
INSDC:SRA:spot2read_filter #1 ( out_spot_filter, out_read_type ) | <
INSDC:SRA:read_filter > echo #1 < 0 > ( out_read_type );
        readonly column INSDC:SRA:read_filter RD_FILTER = out_rd_filter;
        column INSDC:SRA:spot_filter SPOT_FILTER = out_spot_filter |
INSDC:SRA:read2spot_filter #1 ( out_rd_filter ) | < INSDC:SRA:spot_filter >
```

```
echo #1 < 0 > ();
        INSDC:SRA:xread_type in_read_type = READ_TYPE | _alt_in_read_type;
        INSDC:coord:zero in_read_start = READ_START;
        INSDC:coord:len in_read_len = READ_LEN | _alt_in_read_len;
        INSDC:SRA:read_filter in_read_filter = READ_FILTER;
        INSDC:SRA:spot_filter in_spot_filter_0 = SPOT_FILTER |
INSDC:SRA:read2spot_filter #1 ( in_read_filter ) | < INSDC:SRA:spot_filter >
echo #1 < 0 > ();
        INSDC:coord:len spot_len = base_space_spot_len | color_space_spot_len
| align_spot_len;
        INSDC:coord:len fixed_spot_len = static_fixed_spot_len |
base_space_fixed_spot_len | color_space_fixed_spot_len;
}

table INSDC:SRA:tbl:stats #1.1
{
        readonly column INSDC:SRA:spotid_t MIN_SPOT_ID = min_spot_id | <
INSDC:SRA:spotid_t > echo #1 < 1 > ();
        readonly column INSDC:SRA:spotid_t MAX_SPOT_ID = max_spot_id | cast
#1 ( spot_count );
        readonly column U64 SPOT_COUNT = spot_count;
        readonly column U64 BASE_COUNT = base_count;
        readonly column U64 BIO_BASE_COUNT = bio_base_count;
        readonly column U64 CMP_BASE_COUNT = cmp_base_count | base_count;
        U8 stats_dummy = in_stats_bin;
}

table INSDC:SRA:tbl:sra #1.0.4 = INSDC:tbl:sequence #1.0.1,
INSDC:SRA:tbl:spotname #1.1, INSDC:SRA:tbl:spotdesc #1.0.2,
INSDC:SRA:tbl:stats #1.1
{
        column INSDC:SRA:platform_id PLATFORM = .PLATFORM | out_platform;
        readonly column ascii PLATFORM = platform_name;
        column INSDC:SRA:spotid_t SPOT_ID = < INSDC:SRA:spotid_t > add_row_id
#1 ( .SPOT_ID ) | cast #1 ( rowid_64 );
        column ascii SPOT_GROUP = out_spot_group | .SPOT_GROUP | < ascii >
echo #1 < '' > ();
        I64 rowid_64 = row_id #1 ();
        ascii in_spot_group = SPOT_GROUP;
        physical column < INSDC:SRA:platform_id > zip_encoding #1 .PLATFORM =
PLATFORM;
        physical column < INSDC:SRA:spotid_t > izip_encoding #1 .SPOT_ID = <
INSDC:SRA:spotid_t > sub_row_id #1 ( SPOT_ID );
        physical column < ascii > zip_encoding #1 < 0, 1 > .SPOT_GROUP =
in_spot_group;
}

table NCBI:SRA:tbl:stats #1.2.1 = INSDC:SRA:tbl:stats #1.1, INSDC:SRA:tbl:sra
#1.0.4
{
        INSDC:SRA:spotid_t min_spot_id = < INSDC:SRA:spotid_t > meta:value #1
< 'STATS/TABLE/SPOT_MIN' > ();
        INSDC:SRA:spotid_t max_spot_id = < INSDC:SRA:spotid_t > meta:value #1
< 'STATS/TABLE/SPOT_MAX' > ();
        U64 spot_count = < U64 > meta:value #1 < 'STATS/TABLE/SPOT_COUNT' >
();
        U64 base_count = < U64 > meta:value #1 < 'STATS/TABLE/BASE_COUNT' >
```

```
();
        U64 bio_base_count = < U64 > meta:value #1 <
'STATS/TABLE/BIO_BASE_COUNT' > ();
        U64 cmp_base_count = < U64 > meta:value #1 <
'STATS/TABLE/CMP_BASE_COUNT' > () | base_count;
        trigger meta_stats = NCBI:SRA:stats_trigger #1 ( in_stats_bin,
in_read_len, in_read_type, in_spot_group ) | NCBI:SRA:stats_trigger #1 (
in_stats_bin, in_read_len, in_read_type ) | NCBI:SRA:cmp_stats_trigger #1 (
in_cmp_stats_bin, in_stats_qual, in_read_len, in_read_type, in_spot_group ) |
NCBI:SRA:cmp_stats_trigger #1 ( in_cmp_stats_bin, in_stats_qual, in_read_len,
in_read_type ) | NCBI:SRA:cmpf_stats_trigger #1 ( in_cmp_stats_bin,
in_spot_len, in_read_len, in_read_type, in_spot_group ) |
NCBI:SRA:cmpf_stats_trigger #1 ( in_cmp_stats_bin, in_spot_len, in_read_len,
in_read_type ) | NCBI:SRA:cmpb_stats_trigger #1 ( in_cmp_stats_bin,
in_spot_group ) | NCBI:SRA:cmpb_stats_trigger #1 ( in_cmp_stats_bin );
        trigger readlen_stats = NCBI:SRA:readlen_stats_trigger #1 (
in_read_len, in_read_type );
        trigger qual_stats = NCBI:SRA:phred_stats_trigger #1 ( in_qual_phred
);
}

table NCBI:tbl:seqloc #1
{
        column < ascii > zip_encoding #1 SEQ_ID;
        column < INSDC:coord:one > izip_encoding #1 SEQ_START;
        readonly column INSDC:coord:zero SEQ_START = ( INSDC:coord:zero ) <
INSDC:coord:one > diff #1 < 1 > ( .SEQ_START );
        column < INSDC:coord:len > izip_encoding #1 SEQ_LEN;
}

table NCBI:tbl:dcmp_base_space #1
{
        INSDC:dna:text dcmp_virtual_productions = out_dcmp_4na_bin |
out_dcmp_x2na_bin | out_dcmp_2na_bin | out_dcmp_2na_packed;
}

table NCBI:tbl:base_space_common #1.0.3 = INSDC:tbl:sequence #1.0.1,
INSDC:SRA:tbl:spotdesc #1.0.2, INSDC:SRA:tbl:stats #1.1,
NCBI:tbl:dcmp_base_space #1
{
        bool cs_native = < bool > echo #1 < false > ();
        INSDC:dna:text out_cs_key = .CS_KEY | < INSDC:dna:text > echo #1 <
'T' > ( out_read_type ) | < INSDC:dna:text > echo #1 < 'T' > ( out_read_len )
| < INSDC:dna:text > echo #1 < 'T' > ();
        INSDC:2cs:bin out_2cs_bin = < INSDC:x2cs:bin, INSDC:2cs:bin > map #1
< [ 0, 1, 2, 3, 4 ], [ 0, 1, 2, 3, 0 ] > ( out_x2cs_bin );
        INSDC:2na:bin out_2na_bin = out_dcmp_2na_bin | ( INSDC:2na:bin )
unpack #1 ( out_2na_packed );
        INSDC:x2cs:bin out_x2cs_bin = NCBI:color_from_dna #1.1 (
out_x2na_bin, out_read_start, out_read_len, out_cs_key, out_color_matrix );
        INSDC:2cs:packed out_2cs_packed = ( INSDC:2cs:packed ) pack #1 (
out_2cs_bin );
        INSDC:4na:packed out_4na_packed = ( INSDC:4na:packed ) pack #1 (
out_4na_bin );
        INSDC:color:text out_color_text = < INSDC:x2cs:bin, INSDC:color:text
> map #1 < [ 0, 1, 2, 3, 4 ], '0123.' > ( out_x2cs_bin );
        U8 out_color_matrix = < U8 > echo #1 < [ 0, 1, 2, 3, 4, 1, 0, 3, 2,
```

```
4, 2, 3, 0, 1, 4, 3, 2, 1, 0, 4, 4, 4, 4, 4, 4 ] > ();
        INSDC:coord:len base_space_spot_len = ( INSDC:coord:len ) row_len #1
( out_2na_packed );
        INSDC:coord:len base_space_fixed_spot_len = ( INSDC:coord:len )
fixed_row_len #1 ( out_2na_packed );
}

table NCBI:tbl:base_space #3 = NCBI:tbl:base_space_common #1.0.3,
NCBI:tbl:dcmp_base_space #1
{
        INSDC:dna:text in_dnarna_text_upper = < INSDC:dna:text,
INSDC:dna:text > map #1 < '.acmgrsvtwyhkdbnu', 'NACMGRSVTWYHKDBNU' > ( READ
);
        INSDC:dna:text in_dna_text = NCBI:SRA:setRnaFlag #1 (
in_dnarna_text_upper );
        INSDC:4na:bin in_4na_bin = < INSDC:4na:bin > range_validate #1 < 0,
15 > ( READ ) | ( INSDC:4na:bin ) unpack #1 ( in_4na_packed ) | <
INSDC:dna:text, INSDC:4na:bin > map #1 < '.ACMGRSVTWYHKDBN', [ 0, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ] > ( in_dna_text ) | < INSDC:x2na:bin,
INSDC:4na:bin > map #1 < [ 0, 1, 2, 3, 4 ], [ 1, 2, 4, 8, 15 ] > (
in_x2na_bin );
        INSDC:4na:packed in_4na_packed = READ;
        INSDC:x2na:bin in_x2na_bin = < INSDC:x2na:bin > range_validate #1 <
0, 4 > ( READ ) | < INSDC:4na:bin, INSDC:x2na:bin > map #1 < [ 0, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ], [ 4, 0, 1, 4, 2, 4, 4, 4, 3, 4, 4,
4, 4, 4, 4, 4 ] > ( in_4na_bin );
        INSDC:2na:bin in_2na_bin = < INSDC:2na:bin > range_validate #1 < 0, 3
> ( READ ) | ( INSDC:2na:bin ) unpack #1 ( in_2na_packed ) |
INSDC:SEQ:rand_4na_2na #1 ( in_4na_bin );
        INSDC:2na:packed in_2na_packed = READ;
        INSDC:4na:bin in_alt_4na_bin = ( INSDC:4na:bin ) < INSDC:4na:bin,
INSDC:4na:bin > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15 ], [ 15, 0, 0, 3, 0, 5, 6, 7, 0, 9, 10, 11, 12, 13, 14, 15 ] > (
in_4na_bin );
        U8 in_stats_bin = in_2na_bin;
        INSDC:2na:packed out_2na_packed = .READ | out_dcmp_2na_packed;
        INSDC:x2na:bin out_x2na_bin = out_dcmp_x2na_bin | < INSDC:4na:bin,
INSDC:x2na:bin > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15 ], [ 4, 0, 1, 4, 2, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4 ] > ( out_4na_bin );
        INSDC:4na:bin out_2na_4na_bin = < INSDC:2na:bin, INSDC:4na:bin > map
#1 < [ 0, 1, 2, 3 ], [ 1, 2, 4, 8 ] > ( out_2na_bin );
        INSDC:4na:bin out_4na_bin = < INSDC:4na:bin > bit_or #1 < 1 > (
out_2na_4na_bin, .ALTREAD ) | out_dcmp_4na_bin | out_2na_4na_bin;
        INSDC:dna:text out_dnarna_text = < INSDC:4na:bin, INSDC:dna:text >
map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ],
'.ACMGRSVTWYHKDBN' > ( out_4na_bin );
        INSDC:dna:text out_dna_text = NCBI:SRA:useRnaFlag #1 (
out_dnarna_text );
        physical column INSDC:2na:packed .READ = in_2na_packed | (
INSDC:2na:packed ) pack #1 ( in_2na_bin );
        physical column < INSDC:4na:bin > zip_encoding #1 .ALTREAD = <
INSDC:4na:bin > trim #1 < 0, 0 > ( in_alt_4na_bin );
}

table NCBI:tbl:phred_quality #2.0.5 = INSDC:tbl:sequence #1.0.1
{
        INSDC:quality:phred phys_qual_phred = .ORIGINAL_QUALITY | .QUALITY;
```

```
        INSDC:quality:phred const_qual_phred = < INSDC:quality:phred > echo
#1 < 30 > ( out_2na_bin ) | < INSDC:quality:phred > echo #1 < 30 > (
out_4na_bin );
        INSDC:quality:phred raw_qual_phred = < INSDC:quality:phred >
is_configuration_set #1 < '/sra/quality_type', 'raw_scores' > (
phys_qual_phred );
        INSDC:quality:phred syn_qual_phred = NCBI:SRA:syn_quality_read #1 <
30, 3 > ( out_read_start, out_read_len, out_read_type, out_rd_filter ) |
NCBI:SRA:syn_quality #1 < 30, 3 > ( out_read_len, out_spot_filter ) |
const_qual_phred;
        INSDC:quality:phred out_qual_phred = raw_qual_phred | syn_qual_phred;
        INSDC:SRA:spot_filter in_spot_filter = NCBI:SRA:make_spot_filter #1 (
in_4na_bin, in_qual_phred, in_read_start, in_read_len, in_read_type,
in_spot_filter_0 );
        INSDC:quality:text:phred_33 in_qual_text_phred_33 = QUALITY;
        INSDC:quality:text:phred_64 in_qual_text_phred_64 = QUALITY;
        INSDC:quality:phred in_qual_phred = QUALITY | ( INSDC:quality:phred )
< B8 > diff #1 < 33 > ( in_qual_text_phred_33 ) | ( INSDC:quality:phred ) <
B8 > diff #1 < 64 > ( in_qual_text_phred_64 );
        trigger pull_spot_filter = < INSDC:SRA:spot_filter > compare #1 (
in_spot_filter, out_spot_filter );
        INSDC:quality:phred in_stats_qual = in_qual_phred;
        physical column < INSDC:quality:phred > zip_encoding #1
.ORIGINAL_QUALITY = in_qual_phred;
}

table NCBI:SRA:tbl:spotdesc_nocol #1.1 = INSDC:tbl:sequence #1.0.1,
INSDC:SRA:tbl:spotdesc #1.0.2
{
        readonly column NCBI:SRA:Segment LABEL_SEG = out_label_seg | cast #1
( out_label_seg32 ) | cast #1 ( _out_label_seg32 );
        readonly column NCBI:SRA:Segment READ_SEG = out_read_seg | cast #1 (
out_read_seg32 ) | cast #1 ( _out_read_seg32 );
        readonly column NCBI:SRA:ReadDesc READ_DESC = NCBI:SRA:make_read_desc
#1 ( out_nreads, out_read_start, out_read_len, out_read_type, _out_rd_filter,
out_cs_key, _out_label_start, _out_label_len, _out_label );
        readonly column NCBI:SRA:SpotDesc SPOT_DESC = NCBI:SRA:make_spot_desc
#1 ( spot_len, fixed_spot_len, signal_len, trim_start, trim_len, out_nreads
);
        readonly column INSDC:coord:len SIGNAL_LEN = signal_len;
        readonly column U16 SIGNAL_LEN = cast #1 ( signal_len );
        U32 _out_label_startU32 = ( U32 ) out_label_start;
        U32 [ 2 ] _out_label_seg32 = < U32 > paste #1 ( _out_label_startU32,
out_label_len );
        U32 _out_read_startU32 = ( U32 ) out_read_start;
        U32 [ 2 ] _out_read_seg32 = < U32 > paste #1 ( _out_read_startU32,
out_read_len );
        INSDC:SRA:read_filter _out_rd_filter = out_rd_filter | <
INSDC:SRA:read_filter > echo #1 < 0 > ( out_read_start );
        ascii _out_label = out_label | < ascii > echo #1 < '' > ();
        INSDC:coord:zero _out_label_start = out_label_start | <
INSDC:coord:zero > echo #1 < 0 > ( out_read_start );
        INSDC:coord:len _out_label_len = out_label_len | < INSDC:coord:len >
echo #1 < 0 > ( out_read_start );
}

table NCBI:SRA:tbl:spotdesc_nophys #1.1 = NCBI:SRA:tbl:spotdesc_nocol #1.1
```

```
{
        U8 out_nreads = .NREADS;
        ascii out_label = .LABEL;
        INSDC:SRA:xread_type out_read_type = .READ_TYPE;
        INSDC:SRA:read_filter out_rd_filter = .RD_FILTER;
        INSDC:SRA:spot_filter out_spot_filter = .SPOT_FILTER;
        INSDC:coord:zero out_label_start = .LABEL_START | ( INSDC:coord:zero
) < U32 > cut #1 < 0 > ( out_label_seg32 );
        INSDC:coord:len out_label_len = .LABEL_LEN | ( INSDC:coord:len ) <
U32 > cut #1 < 1 > ( out_label_seg32 );
        U32 [ 2 ] out_label_seg32 = cast #1 ( .LABEL_SEG );
        INSDC:coord:zero out_read_start = .READ_START | ( INSDC:coord:zero )
< U32 > cut #1 < 0 > ( out_read_seg32 );
        INSDC:coord:len out_read_len = .READ_LEN | ( INSDC:coord:len ) < U32
> cut #1 < 1 > ( out_read_seg32 );
        U32 [ 2 ] out_read_seg32 = NCBI:SRA:fix_read_seg #1 ( .READ_SEG,
spot_len );
}

table NCBI:SRA:tbl:spotdesc #1.1 = NCBI:SRA:tbl:spotdesc_nophys #1.1
{
        physical column < U8 > zip_encoding #1 .NREADS = NREADS;
        physical column < ascii > zip_encoding #1 .LABEL = LABEL;
        physical column < INSDC:coord:zero > izip_encoding #1 .LABEL_START =
LABEL_START;
        physical column < INSDC:coord:len > izip_encoding #1 .LABEL_LEN =
LABEL_LEN;
        physical column < INSDC:coord:zero > izip_encoding #1 .READ_START =
in_read_start;
        physical column < INSDC:coord:len > izip_encoding #1 .READ_LEN =
in_read_len;
        physical column < INSDC:SRA:xread_type > zip_encoding #1 .READ_TYPE =
in_read_type;
        physical column < INSDC:SRA:read_filter > zip_encoding #1 .RD_FILTER
= in_read_filter;
        physical column < INSDC:SRA:spot_filter > zip_encoding #1
.SPOT_FILTER = in_spot_filter;
}

table NCBI:align:tbl:cmp_base_space #1 = INSDC:tbl:sequence #1.0.1,
NCBI:tbl:dcmp_base_space #1
{
        default column INSDC:dna:text CMP_READ
        {
                read = out_cmp_dna_text;
                validate = < INSDC:dna:text > compare #1 ( in_cmp_dna_text,
out_cmp_dna_text );
        }
        column INSDC:4na:bin CMP_READ = out_cmp_4na_bin;
        column INSDC:4na:packed CMP_READ = out_cmp_4na_packed;
        column INSDC:x2na:bin CMP_READ = out_cmp_x2na_bin;
        column INSDC:2na:bin CMP_READ = out_cmp_2na_bin;
        column INSDC:2na:packed CMP_READ = out_cmp_2na_packed;
        INSDC:dna:text in_cmp_dnarna_text = < INSDC:dna:text, INSDC:dna:text
> map #1 < '.acmgrsvtwyhkdbnu', 'NACMGRSVTWYHKDBNU' > ( CMP_READ );
        INSDC:dna:text in_cmp_dna_text = NCBI:SRA:setRnaFlag #1 (
in_cmp_dnarna_text );
```

```
        INSDC:4na:bin in_cmp_4na_bin = < INSDC:4na:bin > range_validate #1 <
0, 15 > ( CMP_READ ) | ( INSDC:4na:bin ) unpack #1 ( in_cmp_4na_packed ) | <
INSDC:dna:text, INSDC:4na:bin > map #1 < '.ACMGRSVTWYHKDBN', [ 0, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ] > ( in_cmp_dna_text ) | <
INSDC:x2na:bin, INSDC:4na:bin > map #1 < [ 0, 1, 2, 3, 4 ], [ 1, 2, 4, 8, 15
] > ( in_cmp_x2na_bin );
        INSDC:4na:packed in_cmp_4na_packed = CMP_READ;
        INSDC:x2na:bin in_cmp_x2na_bin = < INSDC:x2na:bin > range_validate #1
< 0, 4 > ( CMP_READ ) | < INSDC:4na:bin, INSDC:x2na:bin > map #1 < [ 0, 1, 2,
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ], [ 4, 0, 1, 4, 2, 4, 4, 4, 3,
4, 4, 4, 4, 4, 4, 4 ] > ( in_cmp_4na_bin );
        INSDC:2na:bin in_cmp_2na_bin = < INSDC:2na:bin > range_validate #1 <
0, 3 > ( CMP_READ ) | ( INSDC:2na:bin ) unpack #1 ( in_cmp_2na_packed ) |
INSDC:SEQ:rand_4na_2na #1 ( in_cmp_4na_bin );
        INSDC:2na:packed in_cmp_2na_packed = CMP_READ;
        INSDC:4na:bin in_cmp_alt_4na_bin = < INSDC:4na:bin, INSDC:4na:bin >
map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ], [ 15, 0,
0, 3, 0, 5, 6, 7, 0, 9, 10, 11, 12, 13, 14, 15 ] > ( in_cmp_4na_bin );
        U8 in_cmp_stats_bin = in_cmp_2na_bin;
        INSDC:2na:packed out_cmp_2na_packed = .CMP_READ;
        INSDC:2na:bin out_cmp_2na_bin = ( INSDC:2na:bin ) unpack #1 (
out_cmp_2na_packed );
        INSDC:x2na:bin out_cmp_x2na_bin = < INSDC:4na:bin, INSDC:x2na:bin >
map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ], [ 4, 0, 1,
4, 2, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4 ] > ( out_cmp_4na_bin );
        INSDC:4na:bin out_cmp_2na_4na_bin = < INSDC:2na:bin, INSDC:4na:bin >
map #1 < [ 0, 1, 2, 3 ], [ 1, 2, 4, 8 ] > ( out_cmp_2na_bin );
        INSDC:4na:bin out_cmp_4na_bin = < INSDC:4na:bin > bit_or #1 < 1 > (
out_cmp_2na_4na_bin, .CMP_ALTREAD ) | out_cmp_2na_4na_bin;
        INSDC:4na:packed out_cmp_4na_packed = ( INSDC:4na:packed ) pack #1 (
out_cmp_4na_bin );
        INSDC:dna:text out_cmp_dnarna_text = < INSDC:4na:bin, INSDC:dna:text
> map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ],
'.ACMGRSVTWYHKDBN' > ( out_cmp_4na_bin );
        INSDC:dna:text out_cmp_dna_text = NCBI:SRA:useRnaFlag #1 (
out_cmp_dnarna_text );
        INSDC:x2na:bin out_dcmp_x2na_bin = < INSDC:4na:bin, INSDC:x2na:bin >
map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ], [ 4, 0, 1,
4, 2, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4 ] > ( out_dcmp_4na_bin );
        INSDC:2na:bin out_dcmp_2na_bin = < INSDC:x2na:bin, INSDC:2na:bin >
map #1 < [ 0, 1, 2, 3, 4 ], [ 0, 1, 2, 3, 0 ] > ( out_dcmp_x2na_bin );
        INSDC:2na:packed out_dcmp_2na_packed = ( INSDC:2na:packed ) pack #1 (
out_dcmp_2na_bin );
        physical column INSDC:2na:packed .CMP_READ = in_cmp_2na_packed | (
INSDC:2na:packed ) pack #1 ( in_cmp_2na_bin );
        physical column < INSDC:4na:bin > zip_encoding #1 .CMP_ALTREAD = <
INSDC:4na:bin > trim #1 < 0, 0 > ( in_cmp_alt_4na_bin );
}

table NCBI:align:tbl:qstat #1
{
        column < ascii > zip_encoding #1 SPOT_GROUP;
        column < U32 > izip_encoding #1 CYCLE;
        column INSDC:dna:text KMER;
        column < INSDC:coord:len > izip_encoding #1 HPRUN;
        column < U32 > izip_encoding #1 GC_CONTENT;
        column < INSDC:quality:phred > zip_encoding #1 ORIG_QUAL;
```

```
        column < INSDC:quality:phred > zip_encoding #1 MAX_QUAL;
        column < U8 > zip_encoding #1 NREAD;
        column < U32 > izip_encoding #1 TOTAL_COUNT;
        column < U32 > izip_encoding #1 MISMATCH_COUNT;
}

table NCBI:align:tbl:ref_block_cmn #1
{
        readonly column ascii REF_TABLE = < ascii > meta:read #1 <
'CONFIG/REF_TABLE' > () | < ascii > echo #1 < 'REFERENCE' > ();
        column I64 REF_ID = out_ref_id;
        column INSDC:coord:zero REF_START = out_ref_start;
        column U64 GLOBAL_REF_START = out_global_ref_start;
        column INSDC:coord:len REF_LEN = out_ref_len;
        column bool_encoding #1 REF_ORIENTATION;
        column < U32 > izip_encoding #1 REF_PLOIDY;
        readonly column INSDC:coord:zero REF_POS = NCBI:align:ref_pos #1 (
out_ref_id, out_ref_start );
        readonly column ascii REF_NAME = NCBI:align:ref_name #1 ( out_ref_id
);
        readonly column ascii REF_SEQ_ID = NCBI:align:ref_seq_id #1 (
out_ref_id ) | < ascii > echo #1 < '' > ();
        INSDC:coord:len out_ref_len_internal = NCBI:align:get_ref_len_2 #2 (
out_has_ref_offset, out_ref_offset ) | NCBI:align:get_ref_len #1 (
out_has_ref_offset, out_ref_offset );
        INSDC:coord:len out_ref_len = .REF_LEN | out_ref_len_internal;
        physical column < INSDC:coord:len > izip_encoding #1 .REF_LEN =
REF_LEN;
}

table NCBI:align:tbl:global_ref_block #1 = NCBI:align:tbl:ref_block_cmn #1
{
        U64 out_global_ref_start = .GLOBAL_REF_START;
        I64 out_ref_id = NCBI:align:local_ref_id #1 ( .GLOBAL_REF_START );
        INSDC:coord:zero out_ref_start = NCBI:align:local_ref_start #1 (
.GLOBAL_REF_START );
        physical column < U64 > izip_encoding #1 .GLOBAL_REF_START =
GLOBAL_REF_START;
}

table NCBI:align:tbl:align_cmn #2.1.1 = NCBI:tbl:base_space_common #1.0.3,
NCBI:SRA:tbl:stats #1.2.1, NCBI:align:tbl:ref_block_cmn #1
{
        column < U32 > izip_encoding #1 TMP_KEY_ID;
        column < ascii > zip_encoding #1 LINKAGE_GROUP;
        column < I64 > izip_encoding #1 SEQ_SPOT_ID;
        column < INSDC:coord:one > izip_encoding #1 SEQ_READ_ID;
        readonly column INSDC:coord:len LEFT_SOFT_CLIP =
NCBI:align:get_left_soft_clip #2 ( HAS_REF_OFFSET, REF_OFFSET, out_read_len
);
        readonly column INSDC:coord:len RIGHT_SOFT_CLIP = out_right_clip;
        readonly column ascii CLIPPED_CIGAR_LONG = < ascii >
NCBI:align:get_clipped_cigar #2 ( CIGAR_LONG, CIGAR_LONG_LEN );
        readonly column INSDC:coord:len CLIPPED_CIGAR_LONG_LEN = <
INSDC:coord:len > NCBI:align:get_clipped_cigar #2 ( CIGAR_LONG,
CIGAR_LONG_LEN );
        readonly column ascii CLIPPED_CIGAR_SHORT = < ascii >
```

```
NCBI:align:get_clipped_cigar #2 ( CIGAR_SHORT, CIGAR_SHORT_LEN );
        readonly column INSDC:coord:len CLIPPED_CIGAR_SHORT_LEN = <
INSDC:coord:len > NCBI:align:get_clipped_cigar #2 ( CIGAR_SHORT,
CIGAR_SHORT_LEN );
        readonly column ascii CLIPPED_HAS_MISMATCH = < U8, ascii > map #1 < [
0, 1 ], '01' > ( out_clipped_has_mismatch );
        readonly column bool CLIPPED_HAS_MISMATCH = out_clipped_has_mismatch;
        readonly column ascii CLIPPED_HAS_REF_OFFSET = < U8, ascii > map #1 <
[ 0, 1 ], '01' > ( out_clipped_has_ref_offset );
        readonly column bool CLIPPED_HAS_REF_OFFSET =
out_clipped_has_ref_offset;
        readonly column INSDC:dna:text CLIPPED_MISMATCH = < INSDC:dna:text >
NCBI:align:clip #1 ( out_mismatch_dna_text, LEFT_SOFT_CLIP, RIGHT_SOFT_CLIP
);
        readonly column I32 CLIPPED_REF_OFFSET =
NCBI:align:get_clipped_ref_offset #1 ( HAS_REF_OFFSET, REF_OFFSET );
        readonly column INSDC:quality:phred CLIPPED_QUALITY = <
INSDC:quality:phred > NCBI:align:clip #2 ( out_qual_phred, out_read_len,
LEFT_SOFT_CLIP, RIGHT_SOFT_CLIP );
        readonly column INSDC:dna:text CLIPPED_READ = < INSDC:dna:text >
NCBI:align:clip #2 ( READ, out_read_len, LEFT_SOFT_CLIP, RIGHT_SOFT_CLIP );
        column < NCBI:align:ploidy > izip_encoding #1 PLOIDY;
        column INSDC:quality:phred CMP_QUALITY = .CMP_QUALITY |
out_cmp_quality;
        readonly column INSDC:quality:text:phred_33 SAM_QUALITY = QUALITY;
        column ascii SEQ_NAME = .SEQ_NAME | < ascii > simple_sub_select #1 <
'SEQUENCE', 'NAME' > ( .SEQ_SPOT_ID ) | sprintf #1 < '%u' > ( tmp_seq_spot_id
);
        readonly column U32 SAM_FLAGS = NCBI:align:get_sam_flags #1 (
projected_read_len, .SEQ_READ_ID, out_template_len, REF_ORIENTATION,
out_mate_ref_orientation, is_secondary, out_rd_filter ) |
NCBI:align:get_sam_flags #2 ( out_mate_align_id, .SEQ_READ_ID,
out_template_len, REF_ORIENTATION, out_mate_ref_orientation, is_secondary,
out_rd_filter );
        readonly column ascii MISMATCH_READ = NCBI:align:get_mismatch_read #1
( out_has_mismatch, out_mismatch_dna_text );
        column < I32 > izip_encoding #1 MAPQ;
        column INSDC:coord:zero MATE_REF_POS = out_mate_ref_pos;
        column INSDC:coord:len MATE_REF_LEN = out_mate_ref_len;
        column I64 MATE_REF_ID = out_mate_ref_id;
        column I32 TEMPLATE_LEN = out_template_len;
        column bool MATE_REF_ORIENTATION = out_mate_ref_orientation;
        readonly column ascii MATE_REF_NAME = NCBI:align:ref_name #1 (
out_mate_ref_id );
        readonly column ascii MATE_REF_SEQ_ID = NCBI:align:ref_seq_id #1 (
out_mate_ref_id );
        readonly column U8 ALIGNMENT_COUNT = out_alignment_count;
        readonly column ascii HAS_REF_OFFSET = < U8, ascii > map #1 < [ 0, 1
], '01' > ( out_has_ref_offset );
        column bool_encoding #1 HAS_REF_OFFSET;
        column < I32 > izip_encoding #1 REF_OFFSET;
        column < NCBI:align:ro_type > izip_encoding #1 REF_OFFSET_TYPE;
        readonly column I64 ALIGN_ID = row_id #1 ();
        readonly column INSDC:dna:text REF_READ = < INSDC:4na:bin,
INSDC:dna:text > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15 ], '.ACMGRSVTWYHKDBN' > ( REF_READ );
        readonly column INSDC:4na:bin REF_READ = NCBI:align:ref_sub_select #1
```

```
( out_ref_id, out_ref_start, out_ref_len, .REF_PLOIDY ) |
NCBI:align:ref_sub_select #1 ( out_ref_id, out_ref_start, out_ref_len );
        readonly column INSDC:dna:text RAW_READ = < INSDC:4na:bin,
INSDC:dna:text > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15 ], '.ACMGRSVTWYHKDBN' > ( out_raw_read );
        readonly column INSDC:4na:bin RAW_READ = out_raw_read;
        readonly column ascii CIGAR_LONG = < ascii > NCBI:align:cigar #2 < 1
> ( out_has_mismatch, out_has_ref_offset, out_ref_offset, out_read_len,
out_ref_len, out_ro_type ) | < ascii > NCBI:align:cigar #2 < 1 > (
out_has_mismatch, out_has_ref_offset, out_ref_offset, out_read_len,
out_ref_len ) | < ascii > NCBI:align:cigar #2 < 1 > ( out_has_mismatch,
out_has_ref_offset, out_ref_offset, out_read_len );
        readonly column INSDC:coord:len CIGAR_LONG_LEN = < INSDC:coord:len >
NCBI:align:cigar #2 < 1 > ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_read_len, out_ref_len, out_ro_type ) | < INSDC:coord:len
> NCBI:align:cigar #2 < 1 > ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_read_len, out_ref_len ) | < INSDC:coord:len >
NCBI:align:cigar #2 < 1 > ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_read_len );
        readonly column ascii CIGAR_SHORT = < ascii > NCBI:align:cigar #2 < 0
> ( out_has_mismatch, out_has_ref_offset, out_ref_offset, out_read_len,
out_ref_len, out_ro_type ) | < ascii > NCBI:align:cigar #2 < 0 > (
out_has_mismatch, out_has_ref_offset, out_ref_offset, out_read_len,
out_ref_len ) | < ascii > NCBI:align:cigar #2 < 0 > ( out_has_mismatch,
out_has_ref_offset, out_ref_offset, out_read_len );
        readonly column INSDC:coord:len CIGAR_SHORT_LEN = < INSDC:coord:len >
NCBI:align:cigar #2 < 0 > ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_read_len, out_ref_len, out_ro_type ) | < INSDC:coord:len
> NCBI:align:cigar #2 < 0 > ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_read_len, out_ref_len ) | < INSDC:coord:len >
NCBI:align:cigar #2 < 0 > ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_read_len );
        readonly column ascii RNA_ORIENTATION = NCBI:align:rna_orientation #1
( out_ro_type );
        readonly column U32 EDIT_DISTANCE = NCBI:align:edit_distance #3 (
out_has_mismatch, out_has_ref_offset, out_ref_offset, out_ro_type,
out_read_len ) | NCBI:align:edit_distance #2 ( out_has_mismatch,
out_has_ref_offset, out_ref_offset, out_ref_len, out_read_len ) |
NCBI:align:edit_distance #2 ( out_has_mismatch, out_has_ref_offset,
out_ref_offset, out_ref_len ) | NCBI:align:edit_distance #1 (
out_has_mismatch, out_has_ref_offset, out_ref_offset );
        readonly column ascii HAS_MISMATCH = < U8, ascii > map #1 < [ 0, 1 ],
'01' > ( out_has_mismatch );
        readonly column ascii SEQ_SPOT_GROUP = out_spot_group;
        readonly column ascii REF_MISMATCH = < U8, ascii > map #1 < [ 0, 1 ],
'01' > ( out_ref_mismatch );
        readonly column bool REF_MISMATCH = out_ref_mismatch;
        readonly column ascii REF_INSERT = < U8, ascii > map #1 < [ 0, 1 ],
'01' > ( out_ref_insert );
        readonly column bool REF_INSERT = out_ref_insert;
        readonly column ascii REF_DELETE = < U8, ascii > map #1 < [ 0, 1 ],
'01' > ( out_ref_delete );
        readonly column bool REF_DELETE = out_ref_delete;
        bool is_secondary = out_is_secondary;
        INSDC:coord:len out_right_clip = NCBI:align:get_right_soft_clip #5 (
out_has_ref_offset, out_ref_offset, out_ro_type, out_read_len ) |
NCBI:align:get_right_soft_clip #4 ( out_has_ref_offset, out_ref_offset,
```

```
out_read_len, out_ref_len ) | NCBI:align:get_right_soft_clip #3 (
out_has_ref_offset, out_ref_offset, out_ref_len ) |
NCBI:align:get_right_soft_clip #2 ( out_has_mismatch, LEFT_SOFT_CLIP,
out_has_ref_offset, out_ref_offset );
        bool out_clipped_has_mismatch = < bool > NCBI:align:clip #2 (
out_has_mismatch, out_read_len, LEFT_SOFT_CLIP, RIGHT_SOFT_CLIP );
        bool out_clipped_has_ref_offset = < bool > NCBI:align:clip #2 (
HAS_REF_OFFSET, out_read_len, LEFT_SOFT_CLIP, RIGHT_SOFT_CLIP );
        U32 out_nreads = .PLOIDY | < U32 > echo #1 < 1 > ();
        INSDC:coord:zero out_read_start = .READ_START | < INSDC:coord:zero >
echo #1 < 0 > ();
        INSDC:coord:len align_spot_len = ( INSDC:coord:len ) row_len #1 (
out_has_ref_offset );
        INSDC:coord:len out_read_len = .READ_LEN | align_spot_len;
        INSDC:quality:phred out_raw_qual = < INSDC:quality:phred >
NCBI:align:project_from_sequence #1 < '( INSDC:quality:phred ) QUALITY' > (
.SEQ_SPOT_ID, .SEQ_READ_ID );
        INSDC:quality:phred out_qual_phred = NCBI:align:raw_restore_qual #1 (
out_raw_qual, .REF_ORIENTATION ) | < INSDC:quality:phred > echo #1 < 30 > (
out_4na_bin );
        ascii out_spot_group = < ascii > simple_sub_select #1 < 'SEQUENCE',
'SPOT_GROUP' > ( .SEQ_SPOT_ID );
        INSDC:SRA:spotid_t tmp_seq_spot_id = cast #1 ( .SEQ_SPOT_ID );
        INSDC:coord:len projected_read_len = < INSDC:coord:len >
simple_sub_select #1 < 'SEQUENCE', 'READ_LEN' > ( .SEQ_SPOT_ID );
        ascii out_name_fmt = < ascii > echo #1 < '$R' > ();
        INSDC:coord:zero trim_start = < INSDC:coord:zero > echo #1 < 0 > ();
        INSDC:coord:len trim_len = align_spot_len;
        ascii out_label = .LABEL | < ascii > echo #1 < 'ploidy1' > ();
        INSDC:coord:zero out_label_start = .LABEL_START | < INSDC:coord:zero
> echo #1 < 0 > ();
        INSDC:coord:len out_label_len = .LABEL_LEN | < INSDC:coord:len > echo
#1 < 7 > ();
        INSDC:SRA:read_filter out_rd_filter = .RD_FILTER | <
INSDC:SRA:read_filter > NCBI:align:project_from_sequence #1 < 'READ_FILTER' >
( .SEQ_SPOT_ID, .SEQ_READ_ID ) | < INSDC:SRA:read_filter > echo #1 < 0 > (
out_read_len );
        INSDC:SRA:platform_id out_platform = .PLATFORM | <
INSDC:SRA:platform_id > simple_sub_select #1 < 'SEQUENCE', 'PLATFORM' > (
.SEQ_SPOT_ID ) | < INSDC:SRA:platform_id > echo #1 < 0 > ();
        U8 out_alignment_count = < U8 > NCBI:align:project_from_sequence #1 <
'ALIGNMENT_COUNT' > ( .SEQ_SPOT_ID, .SEQ_READ_ID );
        INSDC:SRA:xread_type out_read_type = < INSDC:SRA:xread_type > echo #1
< 3 > ( out_read_len );
        bool in_stats_bin = HAS_REF_OFFSET;
        INSDC:coord:len _alt_in_read_len = READ_LEN | ( INSDC:coord:len )
row_len #1 ( HAS_REF_OFFSET );
        INSDC:SRA:xread_type _alt_in_read_type = READ_TYPE | <
INSDC:SRA:xread_type > echo #1 < 1 > ( _alt_in_read_len );
        bool out_has_ref_offset = .HAS_REF_OFFSET;
        I32 out_ref_offset = .REF_OFFSET;
        NCBI:align:ro_type out_ro_type = .REF_OFFSET_TYPE;
        INSDC:4na:bin ref_read_internal = NCBI:align:ref_sub_select #1 (
out_ref_id, out_ref_start, out_ref_len_internal, .REF_PLOIDY ) |
NCBI:align:ref_sub_select #1 ( out_ref_id, out_ref_start,
out_ref_len_internal );
        INSDC:dna:text out_dna_text = < INSDC:4na:bin, INSDC:dna:text > map
```

```
#1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ],
'.ACMGRSVTWYHKDBN' > ( out_4na_bin );
        bool out_ref_mismatch = NCBI:align:get_ref_mismatch #1 (
out_has_mismatch, out_has_ref_offset, out_ref_offset, out_ref_len );
        bool out_ref_insert = NCBI:align:get_ref_insert #1 (
out_has_mismatch, out_has_ref_offset, out_ref_offset, out_ref_len );
        bool out_ref_delete = NCBI:align:get_ref_delete #1 (
out_has_mismatch, out_has_ref_offset, out_ref_offset, out_ref_len );
        physical column < INSDC:coord:zero > izip_encoding #1 .READ_START =
READ_START;
        physical column < INSDC:coord:len > izip_encoding #1 .READ_LEN =
READ_LEN;
        physical column < INSDC:quality:phred > zip_encoding #1 .CMP_QUALITY
= CMP_QUALITY;
        physical column < ascii > zip_encoding #1 .SEQ_NAME = SEQ_NAME;
        physical column < INSDC:SRA:read_filter > zip_encoding #1 .RD_FILTER
= READ_FILTER;
}

table NCBI:align:tbl:align_full #1.1.1 = NCBI:align:tbl:align_cmn #2.1.1
{
        column bool_encoding #1 TMP_HAS_MISMATCH;
        readonly column bool HAS_MISMATCH = out_has_mismatch;
        column < INSDC:dna:text > zip_encoding #1 TMP_MISMATCH;
        readonly column INSDC:dna:text MISMATCH = out_mismatch_dna_text;
        readonly column INSDC:4na:bin MISMATCH = out_mismatch_4na_bin;
        column I64 MATE_ALIGN_ID = out_mate_align_id;
        column I64 PRIMARY_ALIGNMENT_ID = .PRIMARY_ALIGNMENT_ID | < I64 >
simple_sub_select #1 < 'SEQUENCE', 'PRIMARY_ALIGNMENT_ID' > ( .SEQ_SPOT_ID,
.SEQ_READ_ID );
        bool out_is_secondary = < bool > echo #1 < true > ();
        INSDC:4na:bin out_raw_read = < INSDC:4na:bin > simple_sub_select #1 <
'PRIMARY_ALIGNMENT', '( INSDC:4na:bin ) RAW_READ' > ( .PRIMARY_ALIGNMENT_ID )
| < INSDC:4na:bin > NCBI:align:project_from_sequence #1 < '( INSDC:4na:bin )
READ' > ( .SEQ_SPOT_ID, .SEQ_READ_ID );
        INSDC:4na:bin out_4na_bin = NCBI:align:align_restore_read #1 (
ref_read_internal, out_has_mismatch, tmp_out_mismatch_4na_bin,
out_has_ref_offset, out_ref_offset, .READ_LEN ) |
NCBI:align:align_restore_read #1 ( ref_read_internal, out_has_mismatch,
tmp_out_mismatch_4na_bin, out_has_ref_offset, out_ref_offset ) |
NCBI:align:raw_restore_read #1 ( out_raw_read, .REF_ORIENTATION );
        bool out_has_mismatch = .TMP_HAS_MISMATCH |
NCBI:align:generate_has_mismatch #1 ( REF_READ, READ, out_has_ref_offset,
out_ref_offset );
        INSDC:4na:bin out_mismatch_4na_bin = NCBI:align:generate_mismatch #1
( REF_READ, READ, out_has_ref_offset, out_ref_offset );
        INSDC:4na:bin tmp_out_mismatch_4na_bin = < INSDC:dna:text,
INSDC:4na:bin > map #1 < '.ACMGRSVTWYHKDBN', [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15 ] > ( .TMP_MISMATCH );
        INSDC:dna:text out_mismatch_dna_text = .TMP_MISMATCH | <
INSDC:4na:bin, INSDC:dna:text > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15 ], '.ACMGRSVTWYHKDBN' > ( out_mismatch_4na_bin );
        INSDC:coord:zero out_mate_ref_pos = .MATE_REF_POS | <
INSDC:coord:zero > simple_sub_select #1 < '', 'REF_POS' > ( MATE_ALIGN_ID );
        I64 out_mate_ref_id = .MATE_REF_ID | < I64 > simple_sub_select #1 <
'', 'REF_ID' > ( MATE_ALIGN_ID );
        INSDC:coord:len out_mate_ref_len = < INSDC:coord:len >
```

```
simple_sub_select #1 < '', 'REF_LEN' > ( MATE_ALIGN_ID );
        I32 out_template_len = .TEMPLATE_LEN | NCBI:align:template_len #1 (
REF_POS, out_mate_ref_pos, out_ref_len, out_mate_ref_len, REF_NAME,
MATE_REF_NAME, SEQ_READ_ID );
        bool out_mate_ref_orientation = .MATE_REF_ORIENTATION | < bool >
simple_sub_select #1 < '', 'REF_ORIENTATION' > ( MATE_ALIGN_ID );
        I64 out_mate_align_id = .MATE_ALIGN_ID;
        I32 read_idx = < I32 > cast #1 ( .SEQ_READ_ID );
        physical column < INSDC:coord:zero > izip_encoding #1 .MATE_REF_POS =
MATE_REF_POS;
        physical column < I64 > izip_encoding #1 .MATE_REF_ID = MATE_REF_ID;
        physical column < I32 > izip_encoding #1 .TEMPLATE_LEN =
TEMPLATE_LEN;
        physical column < bool > izip_encoding #1 .MATE_REF_ORIENTATION =
MATE_REF_ORIENTATION;
        physical column < I64 > izip_encoding #1 .MATE_ALIGN_ID =
MATE_ALIGN_ID;
        physical column < I64 > izip_encoding #1 .PRIMARY_ALIGNMENT_ID =
PRIMARY_ALIGNMENT_ID;
}

table NCBI:align:tbl:compressed_by_reference #1.2.1 =
NCBI:align:tbl:align_cmn #2.1.1
{
        column bool_encoding #1 HAS_MISMATCH;
        column INSDC:dna:text MISMATCH
        {
                read = out_mismatch_dna_text;
                validate = < INSDC:dna:text > compare #1 (
in_mismatch_dna_text, out_mismatch_dna_text );
        }
        column < ascii > zip_encoding #1 ALIGN_GROUP;
        column I64 MATE_ALIGN_ID = out_mate_align_id;
        readonly column U32 MATE_EDIT_DISTANCE = < U32 > simple_sub_select #1
< '', 'EDIT_DISTANCE' > ( MATE_ALIGN_ID );
        readonly column ascii MATE_CIGAR_LONG = < ascii > simple_sub_select
#1 < '', 'CIGAR_LONG' > ( MATE_ALIGN_ID );
        readonly column ascii MATE_CIGAR_SHORT = < ascii > simple_sub_select
#1 < '', 'CIGAR_SHORT' > ( MATE_ALIGN_ID );
        readonly column INSDC:coord:len MATE_CIGAR_LONG_LEN = <
INSDC:coord:len > simple_sub_select #1 < '', 'CIGAR_LONG_LEN' > (
MATE_ALIGN_ID );
        readonly column INSDC:coord:len MATE_CIGAR_SHORT_LEN = <
INSDC:coord:len > simple_sub_select #1 < '', 'CIGAR_SHORT_LEN' > (
MATE_ALIGN_ID );
        bool out_is_secondary = < bool > echo #1 < false > ();
        bool out_has_mismatch = .HAS_MISMATCH;
        INSDC:dna:text in_mismatch_dna_text = < INSDC:dna:text,
INSDC:dna:text > map #1 < '.acmgrsvtwyhkdbn', 'NACMGRSVTWYHKDBN' > ( MISMATCH
);
        INSDC:4na:bin in_mismatch_4na_bin = < INSDC:dna:text, INSDC:4na:bin >
map #1 < '.ACMGRSVTWYHKDBN', [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15 ] > ( in_mismatch_dna_text );
        INSDC:4na:bin out_mismatch_4na_bin = .MISMATCH;
        INSDC:dna:text out_mismatch_dna_text = < INSDC:4na:bin,
INSDC:dna:text > map #1 < [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15 ], '.ACMGRSVTWYHKDBN' > ( out_mismatch_4na_bin );
```

```
        I64 out_mate_align_id = .MATE_ALIGN_ID | NCBI:align:get_mate_align_id
#1 ( .SEQ_SPOT_ID );
        INSDC:4na:bin out_4na_bin = NCBI:align:align_restore_read #1 (
ref_read_internal, out_has_mismatch, .MISMATCH, out_has_ref_offset,
out_ref_offset, .READ_LEN ) | NCBI:align:align_restore_read #1 (
ref_read_internal, out_has_mismatch, .MISMATCH, out_has_ref_offset,
out_ref_offset );
        INSDC:4na:bin out_raw_read = NCBI:align:raw_restore_read #1 (
out_4na_bin, .REF_ORIENTATION );
        I64 primary_align_pair = < I64 > simple_sub_select #1 < 'SEQUENCE',
'PRIMARY_ALIGNMENT_ID' > ( .SEQ_SPOT_ID );
        I64 out_mate_ref_id = < I64 > simple_sub_select #1 < '', 'REF_ID' > (
MATE_ALIGN_ID );
        bool out_mate_ref_orientation = < bool > simple_sub_select #1 < '',
'REF_ORIENTATION' > ( MATE_ALIGN_ID );
        INSDC:coord:zero out_mate_ref_pos = < INSDC:coord:zero >
simple_sub_select #1 < '', 'REF_POS' > ( MATE_ALIGN_ID );
        INSDC:coord:len out_mate_ref_len = < INSDC:coord:len >
simple_sub_select #1 < '', 'REF_LEN' > ( MATE_ALIGN_ID );
        I32 out_template_len = NCBI:align:template_len #1 ( REF_POS,
out_mate_ref_pos, out_ref_len, out_mate_ref_len, REF_NAME, MATE_REF_NAME,
SEQ_READ_ID );
        physical column < INSDC:4na:bin > zip_encoding #1 .MISMATCH =
in_mismatch_4na_bin;
        physical column < I64 > izip_encoding #1 .MATE_ALIGN_ID =
MATE_ALIGN_ID;
}

table NCBI:align:tbl:align_sorted #1.2.1 =
NCBI:align:tbl:compressed_by_reference #1.2.1,
NCBI:align:tbl:global_ref_block #1
{
        column default limit = 131072;
}

table NCBI:align:tbl:align_mate_sorted #1.1.1 = NCBI:align:tbl:align_full
#1.1.1, NCBI:align:tbl:global_ref_block #1
{
        column default limit = 131072;
}

table NCBI:align:tbl:seq #2 = NCBI:tbl:base_space #3, NCBI:tbl:phred_quality
#2.0.5, NCBI:align:tbl:cmp_base_space #1, NCBI:SRA:tbl:spotdesc #1.1,
NCBI:SRA:tbl:stats #1.2.1
{
        column default limit = 131072;
        column < I64 > izip_encoding #1 PRIMARY_ALIGNMENT_ID;
        column < U8 > zip_encoding #1 ALIGNMENT_COUNT;
        column < ascii > zip_encoding #1 RAW_NAME;
        column < U64 > izip_encoding #1 TMP_KEY_ID;
        column < U64 > izip_encoding #1 TI;
        column < ascii > zip_encoding #1 CMP_LINKAGE_GROUP;
        readonly column ascii LINKAGE_GROUP =
NCBI:align:seq_restore_linkage_group #1 ( .CMP_LINKAGE_GROUP,
.PRIMARY_ALIGNMENT_ID ) | .CMP_LINKAGE_GROUP;
        INSDC:coord:zero trim_start = < INSDC:coord:zero > echo #1 < 0 > ();
        INSDC:coord:len trim_len = _spot_len;
```

```
        ascii out_name_fmt = .RAW_NAME | < ascii > echo #1 < '$R' > ();
        INSDC:4na:bin out_dcmp_4na_bin = NCBI:align:seq_restore_read #1 (
out_cmp_4na_bin, .PRIMARY_ALIGNMENT_ID, .READ_LEN, .READ_TYPE );
}

table NCBI:align:tbl:reference #3 = NCBI:align:tbl:cmp_base_space #1,
NCBI:tbl:base_space #3, NCBI:tbl:seqloc #1, NCBI:SRA:tbl:stats #1.2.1
{
        column < U32 > izip_encoding #1 MAX_SEQ_LEN;
        column bool_encoding #1 CIRCULAR;
        column utf8 NAME = out_spot_name_utf8;
        column < U8 > izip_encoding #1 CGRAPH_HIGH;
        column < U8 > izip_encoding #1 CGRAPH_LOW;
        column < U32 > izip_encoding #1 CGRAPH_MISMATCHES;
        column < U32 > izip_encoding #1 CGRAPH_INDELS;
        column < I64 > izip_encoding #1 PRIMARY_ALIGNMENT_IDS;
        column < I64 > izip_encoding #1 SECONDARY_ALIGNMENT_IDS;
        column < I64 > izip_encoding #1 EVIDENCE_INTERVAL_IDS;
        column < INSDC:coord:zero > izip_encoding #1 OVERLAP_REF_POS;
        column < INSDC:coord:len > izip_encoding #1 OVERLAP_REF_LEN;
        readonly column vdb:row_id_range NAME_RANGE = idx:text:lookup #1 <
'i_name', 'QUERY_SEQ_NAME' > ();
        INSDC:quality:phred out_qual_phred = < INSDC:quality:phred > echo #1
< 30 > ( out_dcmp_4na_bin );
        INSDC:dna:text in_cs_key = < INSDC:dna:text, INSDC:dna:text > map #1
< 'acgtn', 'ACGTN' > ( CS_KEY );
        U32 in_spot_len = SEQ_LEN;
        INSDC:coord:len _alt_in_read_len = READ_LEN | SEQ_LEN;
        INSDC:SRA:xread_type _alt_in_read_type = READ_TYPE | <
INSDC:SRA:xread_type > echo #1 < 1 > ();
        INSDC:coord:zero out_read_start = < INSDC:coord:zero > echo #1 < 0 >
();
        INSDC:coord:len out_read_len = .SEQ_LEN;
        utf8 out_spot_name_utf8 = idx:text:project #1 < 'i_name' > ( .NAME );
        ascii out_spot_name = cast #1 ( out_spot_name_utf8 );
        INSDC:coord:zero trim_start = < INSDC:coord:zero > echo #1 < 0 > ();
        INSDC:coord:len trim_len = base_space_spot_len;
        ascii out_label = < ascii > echo #1 < 'reference' > ();
        INSDC:coord:zero out_label_start = < INSDC:coord:zero > echo #1 < 0 >
();
        INSDC:coord:len out_label_len = < INSDC:coord:len > echo #1 < 9 > ();
        U32 out_nreads = < U32 > echo #1 < 1 > ();
        INSDC:SRA:xread_type out_read_type = < INSDC:SRA:xread_type > echo #1
< 3 > ();
        INSDC:SRA:read_filter out_rd_filter = < INSDC:SRA:read_filter > echo
#1 < 0 > ();
        INSDC:4na:bin out_dcmp_4na_bin = NCBI:align:ref_restore_read #1 (
out_cmp_4na_bin, .SEQ_ID, .SEQ_START, .SEQ_LEN );
        physical column < INSDC:dna:text > zip_encoding #1 .CS_KEY =
in_cs_key;
        physical column utf8 .NAME = idx:text:insert #1 < 'i_name' > ( NAME
);
}

database NCBI:align:db:alignment_sorted #2
{
        table NCBI:align:tbl:reference #3 REFERENCE;
```

```
        table NCBI:align:tbl:align_sorted #1.2.1 PRIMARY_ALIGNMENT;
        table NCBI:align:tbl:align_mate_sorted #1.1.1 SECONDARY_ALIGNMENT;
        table NCBI:align:tbl:seq #2 SEQUENCE;
        table NCBI:align:tbl:qstat #1 QUAL_STAT;
}
```