StatsDB: platform-agnostic storage and understanding of next generation sequencing run metrics [version 1; peer review: 2 approved, 1 approved with reservations]

Ricardo H. Ramirez-Gonzalez, Richard M. Leggett, Darren Waite, Anil Thanki, Nizar Drou, Mario Caccamo, Robert Davey

The Genome Analysis Centre, Norwich Research Park, Norwich, NR4 7UH, UK

Abstract
Modern sequencing platforms generate enormous quantities of data in ever-decreasing amounts of time. Additionally, techniques such as multiplex sequencing allow one run to contain hundreds of different samples. With such data comes a significant challenge to understand its quality and to understand how the quality and yield are changing across instruments and over time. As well as the desire to understand historical data, sequencing centres often have a duty to provide clear summaries of individual run performance to collaborators or customers. We present StatsDB, an open-source software package for storage and analysis of next generation sequencing run metrics. The system has been designed for incorporation into a primary analysis pipeline, either at the programmatic level or via integration into existing user interfaces. Statistics are stored in an SQL database and APIs provide the ability to store and access the data while abstracting the underlying database design. This abstraction allows simpler, wider querying across multiple fields than is possible by the manual steps and calculation required to dissect individual reports, e.g. “provide metrics about nucleotide bias in libraries using adaptor barcode X, across all runs on sequencer A, within the last month”. The software is supplied with modules for storage of statistics from FastQC, a commonly used tool for analysis of sequence reads, but the open nature of the database schema means it can be easily adapted to other tools. Currently at The Genome Analysis Centre (TGAC), reports are accessed through our LIMS system or through a standalone GUI tool, but the API and supplied examples make it easy to develop custom reports and to interface with other packages.
Corresponding authors: Richard M. Leggett (richard.leggett@tgac.ac.uk), Robert Davey (robert.davey@tgac.ac.uk)

Competing interests: The authors declare that there are no competing interests.

Grant information: The development of StatsDB has been funded by a Biotechnology and Biological Sciences Research Council (BBSRC) National Capability Grant at TGAC.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2013 Ramirez-Gonzalez RH et al. This is an open access article distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. Data associated with the article are available under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original data is properly cited.

How to cite this article: Ramirez-Gonzalez RH, Leggett RM, Waite D et al. StatsDB: platform-agnostic storage and understanding of next generation sequencing run metrics [version 1; peer review: 2 approved, 1 approved with reservations] F1000Research 2013, 2:248 (https://doi.org/10.12688/f1000research.2-248.v1)

Introduction

Next generation short-read sequencers are now capable of generating hundreds of gigabases of sequence data per run. This increase in throughput is complemented by technologies such as long-read single molecule platforms, multiplex sequencing and RADSeq, that lead to differing analytical techniques and enable hundreds of samples to be combined into a single sequencing experiment, respectively. With this data heterogeneity and abundance comes a significant challenge to keep track of samples, to understand data quality and to understand how quality control (QC) and yield are changing across instruments and over time. As well as the desire to understand historical data, centres often have a duty to provide clear summaries of individual run performance to collaborators or customers.

A user’s initial attempts to understand the QC profile of a run usually involve using manufacturer-supplied software, for example Illumina’s Sequence Analysis Viewer or PacBio’s SMRTAnalysis. As well as these, a number of community-developed tools have emerged for assessing run quality and they are often included in the primary analysis pipelines of sequencing centres. FastQC is a popular tool for analysing FASTQ files and is able to report a wide range of information related to the quality profile of the reads, as well as analysing QC content and over-represented sequence, such as PCR duplicates or over-abundance of adaptors. HTQC, another toolkit for FASTQ data analysis, is composed of a set of six tools for analysis and trimming of reads. PRINSEQ also analyses and trims reads, with an emphasis on metagenomic datasets. Other tools include NGSQC, QRQC and SAMStat. The latter tool, as the name implies, works with SAM files rather than FASTQ files.

The new generation of single molecule sequencing technologies, such as the RS platform from Pacific Biosciences with longer reads and different error profiles, have brought their own QC challenges. While it is still possible to get useful information from next generation tools such as FastQC, new tools are emerging which are tailored to the platform. Examples include stsPlots, which provides graphical summaries of data included in the .sts.csv files output by the instrument and PacBioEDA which operates on the .bas.h5 files produced by the PacBio primary analysis.

While there are a range of useful tools available to generate QC statistics for individual runs, we are not aware of any currently available solution for facilitating the easy storage and access of this valuable information. For this reason, we have created StatsDB, a platform-independent, tool-independent run QC and metadata database with APIs in Perl and Java. StatsDB features a generic database schema which enables the storage of data from any QC tool designed for any sequencing platform. StatsDB is designed to automate the storage of run QC metrics, enabling more granular queries over the data held within. Installation is simple and use of the software and API requires no knowledge of SQL.

Methods and implementation

Figure 1 illustrates the overall structure of the StatsDB system. At the core of StatsDB is a MySQL database which stores run metrics on a per-base, per-partition or per-run basis. On top of the database sits an API - currently implemented in both Perl and Java - which abstracts the database design from the tools that use the data and provides a simple interface for adding or querying data.

We envisage two types of programs utilising the API - parsers and consumers. Parsers process external data output from QC tools and use the StatsDB API to write the data. Conversely, consumers query the data stored in StatsDB and present it to the user, typically in the form of text and graphs. The StatsDB package includes parser implementations and examples of consumers, and the API enables others to be developed quickly and easily. Additionally, integrating these consumers into third-party applications is promoted - the open source LIMS system currently in development at TGAC, MISO, provides support for accessing data in StatsDB out-of-the-box.

Database installation

The StatsDB framework supplies two SQL files that are used to set up the database for use, comprising the schema and the stored procedures, respectively, and detailed in the following sections.

The first step is to create a new MySQL database called ‘statsdb’ and grant a user read-write access to it, e.g. a new ‘statsdb’ user with a suitable password, e.g. ‘statsdb’:

```
mysql > CREATE DATABASE statsdb ;
mysql > USE statsdb ;
```

We then grant the user all privileges:

```
mysql > GRANT ALL ON ‘statsdb’.*
   TO ’statsdb’@’localhost’;
```

The two SQL files are then imported into the database as follows:

```
$ mysql -D statsdb -u statsdb -p < statsdb_schema . sql
$ mysql -D statsdb -u statsdb -p < stored_procedures . sql
```
This will populate the ‘statsdb’ database with the tables and procedures required.

Testing a successfully installed database can be undertaken by running the FastQC parser on the supplied example data, as follows:

```bash
$ cd Perl
$ perl parse_fastqc.pl
   -i examples/metadata_test.txt
   -d examples/template_db.txt
```

This will result in data being inserted into the database. In order to revert back to an empty state, reimport the schema SQL as detailed previously.

Database design

The StatsDB database is designed to be flexible and to hold virtually any type of QC analysis. Figure 2 illustrates the database schema. The database is normalised to the third normal form (3NF) and has stored procedures and views to facilitate consistent access to the stored information. The tables in the core of the database are as follows:

- **analysis**: holds the ID and timestamp of when the analysis was recorded.

- **analysis_property**: holds general information about the analysis and the run from which it originated. The values contained within this table are populated directly from a user-specified tab-delimited table of property headings and respective values, i.e. the RunTable object (see ‘Parsers’ below). The following properties are used to define common denominators across platforms and analyses:
  - **tool**: the name of the tool that was used to undertake the analysis (for example, FastQC, PRINSEQ).
  - **encoding**: the encoding of the input, for example ‘Illumina 1.5’, ‘fasta+qual’.
  - **chemistry**: a short name for the chemistry or type of experiment run.
  - **instrument**: the ID of the instrument.
  - **software**: the basecaller software or software version of the instrument.
  - **type**: type of experiment, for example ‘RNA-Seq’, ‘WGS’.
  - **pair**: if the experiment is paired end or mate pair, then 1 if the first read or 2 if the second read.
  - **sample_name**: the name of the sample as assigned during the library construction.

Figure 2. StatsDB database schema.
• lane - the lane or flowcell of the instrument.
• run - the name of the run, as given by the instrument.
• barcode - if the sample is multiplexed, the adaptor barcode of the sample, for example ‘ACCTTG’.

`list_selectable_values_from_property` will list all available properties.

`list_selectable_properties` will list the properties that can be used as queries.

`list_selectable_values_from_property` will list the values for a given property.

`list_runs` will list all available runs.

`database:`

The following stored procedures are used by the APIs to query the database: as views and stored procedures. This enables it to have lightweight APIs which behave consistently across languages and means that the API can easily be ported to other languages should the implementation require it.

The StatsDB framework provides two APIs, one in the Perl language and the other in Java, both of which offer the same functionality. They call the stored procedures and provide a sufficient layer of abstraction such that parsers and consumers do not need to access the stored procedures directly. Therefore, the typical method of accessing the data held within StatsDB is through the following APIs.

**API**

**Perl API**

Before using the Perl API, a connection to the database has to be created as defined by the Perl DBI API\(^2\). A template for the configuration file is provided in `Perl/examples/template_db.txt`, and the values need to match those used in the database installation instructions above:

```
    db_string  dbi:mysql:statsdb;host=localhost
    db_user    statsdb
    db_password statsdb
```

The Perl API comprises modules to import analysis information to the StatsDB database, as well as functions to query the database.

The abstraction of the database is contained in the `QCAnalysis` module, which is used to add an analysis to the database. It automatically fills the missing types of value in the database, so it is not necessary to have a comprehensive list of types of analysis that are going to be stored `a priori`. However, a parser needs to define valid types for the `QCAnalysis` object, i.e. `value_type`, and to which scope they should be assigned, i.e. the `type_scope`, as described in the Database design section above, and in the following code:

```
// function signature
$analysis->add_valid_type ($value_type,
                        $value_scope);

// examples
$analysis->add_valid_type ("general_gc_content",
                        "analysis");
```

The following stored procedures are used to generate reports. To make queries, the common arguments to specify the run or runs to group are: `instrument_in, run_in, lane_in, pair_in` and `barcode`. The arguments are optional and if more than one analysis meets the query criteria, a summary is produced. This can be used, for example, to query the quality of a given lane and use this information to find systematic issues.

- `general_summaries_for_run` returns a summary of all the values global to the analysis.
- `summary_per_position_for_run` returns all the values queried from the `per_partition_value` table.
- `summary_value_with_comment` returns the summary values with a descriptive text, if it was present as a note for a value. For example, this can be used to retrieve further description of the over-represented sequences in a FastQC report.

**Stored procedures**

Most of the functionality of StatsDB is embedded directly in the database as views and stored procedures. This enables it to have lightweight APIs which behave consistently across languages and means that the API can easily be ported to other languages should the implementation require it.

The following stored procedures are used by the APIs to query the database:

- `list_runs` will list all available runs.
- `summary_per_scope` will return the descriptions present for a given scope. This is useful to know if a `value_type` is already present or not, before trying to insert it again.

In addition to the tables, StatsDB has three auxiliary views, used to facilitate complex queries:

- `property` transposes the `analysis_property` table, transforming the row values to columns.
- `run` merges the corresponding values from `property`, according to the run.
- `latest_run` selects the latest run from the `run` view, so if an analysis has been run more than once, only the latest results are stored.

### Stored procedures

- `analysis_value` is used to store the properties that can be represented as a value for the whole run, e.g. overrepresented sequences, counts of multiplexed tags.
- `per_position_value` stores values related to a absolute position that is never grouped, such as counts of how many times a certain quality is represented.
- `per_partition_value` is similar to `per_position_value`, but allows grouping by some range or factor.
- `value_type` is an auxiliary table that holds descriptions for the values and the scope of the value.
- `type_scope` is an auxiliary table defining the scope relevant to the analysis to group the values consistently. Currently, the following are the scopes supported by StatsDB:
  - `analysis` - global values for the analysis, for example ‘Total GC content’.
  - `base_partition` - values of summaries per base, allowing grouping of ranges, for example quality mean per position.
  - `sequence_cumulative` - for cumulative counts, such as quality scores or read counts.
  - `GC_content'.
  - `per_position_value` stores values related to a position that is never grouped, such as counts of how many times a certain value belongs to a set.
  - `per_partition_value` is similar to `per_position_value`, but allows grouping by some range.
  - `value_type` is an auxiliary table that holds descriptions for the values and the scope of the value.
  - `type_scope` is an auxiliary table defining the scope relevant to the analysis to group the values consistently. Currently, the following are the scopes supported by StatsDB:
    - `analysis` - global values for the analysis, for example ‘Total GC content’.
    - `base_partition` - values of summaries per base, allowing grouping of ranges, for example quality mean per position.
    - `sequence_cumulative` - for cumulative counts, such as quality scores or read counts.

In addition to the tables, StatsDB has three auxiliary views, used to facilitate complex queries:

- `property` transposes the `analysis_property` table, transforming the row values to columns.
- `run` merges the corresponding values from `property`, according to the run.
- `latest_run` selects the latest run from the `run` view, so if an analysis has been run more than once, only the latest results are stored.

### Stored procedures

Most of the functionality of StatsDB is embedded directly in the database as views and stored procedures. This enables it to have lightweight APIs which behave consistently across languages and means that the API can easily be ported to other languages should the implementation require it.

The following stored procedures are used by the APIs to query the database:

- `list_runs` will list all available runs.
- `list_selectable_properties` will list the properties that can be used as queries.
- `list_selectable_values_from_property` will list the values for a given property.

The following stored procedures are used to generate reports. To make queries, the common arguments to specify the run or runs to group are: `instrument_in, run_in, lane_in, pair_in` and `barcode`. The arguments are optional and if more than one analysis meets the query criteria, a summary is produced.

The following stored procedures are used by the APIs to query the database:

- `list_runs` will list all available runs.
- `list_selectable_properties` will list the properties that can be used as queries.
- `list_selectable_values_from_property` will list the values for a given property.

The following stored procedures are used to generate reports. To make queries, the common arguments to specify the run or runs to group are: `instrument_in, run_in, lane_in, pair_in` and `barcode`. The arguments are optional and if more than one analysis meets the query criteria, a summary is produced.

The following stored procedures are used to generate reports. To make queries, the common arguments to specify the run or runs to group are: `instrument_in, run_in, lane_in, pair_in` and `barcode`. The arguments are optional and if more than one analysis meets the query criteria, a summary is produced.

The following stored procedures are used to generate reports. To make queries, the common arguments to specify the run or runs to group are: `instrument_in, run_in, lane_in, pair_in` and `barcode`. The arguments are optional and if more than one analysis meets the query criteria, a summary is produced.

The following stored procedures are used to generate reports. To make queries, the common arguments to specify the run or runs to group are: `instrument_in, run_in, lane_in, pair_in` and `barcode`. The arguments are optional and if more than one analysis meets the query criteria, a summary is produced.
After defining the valid values and scopes, the properties of the analysis should be added. The following example adds a 'tool' property with a 'FastQC' value, to represent a FastQC analysis property type:

```perl
$analysis->add_property ("tool","FastQC");
```

Global values are supported in StatsDB to represent generalised properties that are permissible across analyses. To add global values to the analysis, the `add_general_value` function is used. General values can have an optional description. When the description is present, and if the value is new to the database, the description is added. If the value is already present in the database, the description is ignored. This is by design and allows for consistency across analyses:

```perl
// function signature
$analysis->add_general_value ($key, $value, $description);
```

### Examples

```perl
$analysis->add_general_value ("gc_content_percentage", 100);
$analysis->add_general_value ("average_length", 100);
```

To add values with a discrete count the `add_position_value` function is called. The following example specifies that 15,000 reads in the analysis had a quality score of 30:

```perl
// function signature
$analysis->add_position_value ($position, $key, $value);
```

```perl
$analysis->add_position_value(30, "quality_score_count", 15000);
```

Finally, to add values that can be grouped in ranges the function `add_partition_value` is called. A range is an array specifying the first and last position (inclusive). The following example inserts a quality mean from position 10 to position 14 (5 values) of the run:

```perl
// function signature
$analysis->add_partition_value ($range, $key, $value);
```

```perl
$analysis->add_partition_value ([10-14], "quality_mean", 38.7);
```

An auxiliary function, `parse_range`, is provided to convert a string representation of a range into an array type. If a single string value is provided, it returns an array with the value repeated, representing a partition of size 1. If a range string is passed, it returns an array with the positions as needed by `add_partition_value`:

```perl
$analysis->parse_range ("10") -> [10,10]
$analysis->parse_range ("10-14") -> [10,14]
```

Once the QC Analysis object is constructed with all the required values, it can be inserted to the database with `db-> > insert_analysis ($analysis)`.

To query the database from the Perl API, the `Reports.pm` module is used. To allow flexibility in the querying and to be able to get summaries at different granularities (barcode, lane, pair, etc) the convention is that all the queries accept as an argument a properties hash comprising the required key-value pairs to build the query. A constant, declared in the `Reports.pm` module, represents the set of controlled platform-agnostic keys and is defined as follows:

```perl
use constant {
    ENCODING => "encoding",
    CHEMISTRY => "chemistry",
    INSTRUMENT => "instrument",
    SOFTWARE_ON_INSTRUMENT => "softwareOnInstrument",
    TYPE_OF_EXPERIMENT => "typeOfExperiment",
    PAIR => "pair",
    SAMPLE_NAME => "sampleName",
    LANE => "lane",
    BARCODE => "barcode",
    RUN => "run"
};
```

An example script to query the different types of tables is provided with the StatsDB framework, i.e. `examples/example_consumer.pl`.

### Java API

The Java API is supplied as a Maven 
project to ease building and testing. For convenience, a pre-built JAR file is available to use in existing Java projects by simply downloading the JAR file from the TGAC Maven repository, or including the following repository and dependency Maven declarations in your pom.xml build descriptor to download the artifact:
The API is built using the standard Maven command:

```bash
cmvn clean install
```

This will compile the source code and provide a library JAR comprising the API, but does not attempt the StatsDB database-level tests. These unit tests make sure the database is accessible, that the schema is correct, and that the API calls available work correctly. To turn these tests on, supply the relevant database connection properties in `Java/statsdb – api/src/test/resources/test.statsdb.properties`:

```properties
statsdb.driver=com.mysql.jdbc.Driver
database.url=jdbc:mysql://localhost:3306/statsdb
database.username=statsdb
database.password=statsdb
```

Then use the following profile activation when building the library:

```bash
cmvn clean install -DdbTests=true
```

An option to build an executable JAR file is available which includes a dedicated command-line application that allows API access for loading and querying a StatsDB database. To enable this option, use the following build command:

```bash
cmvn clean install -Donejar=true
```

The resulting JAR can then be executed by the user. This helper application requires either an input file representing the analysis report to be parsed, e.g. a `fastqc_data.txt` file, or a StatsDB metadata table file (see Table 1) comprising multiple analysis reports. The helper application will then process the analysis file(s) and load the data into StatsDB. Supplying the `-t` option allows testing of a given parser without writing any information into the database.

```
$ java -jar statsdb-api.one-jar.jar -h
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-f &lt;file&gt;</code></td>
<td>Use given input report file</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Print this help</td>
</tr>
<tr>
<td><code>-m &lt;file&gt;</code></td>
<td>Process multiple reports using a StatsDB metadata table file</td>
</tr>
<tr>
<td><code>-p &lt;fastqc,other&gt;</code></td>
<td>Use specified parser type</td>
</tr>
</tbody>
</table>

Before using the Java API library, a connection to the database has to be created in a similar way to the Perl API. This file needs to be called `statsdb.properties`, needs to reside on the classpath (in the case of the executable JAR, this would simply be in the same directory, for example), and contains the same fields as the test properties example above. Finally, as with the Perl API, the values need to match those used in the database installation instructions above.

In terms of building and initiating queries within custom applications, the Java API mirrors the Perl API whereby the abstraction of the database is contained in the `QCAnalysis` class. Similarly, the `Reports` class is used to query the database from the Java API, and the query key granularity is represented as a `Map <RunProperty, String>` comprising the required key-value pairs to build the query. For convenience, the API tolerates missing values and, in such a case, the average is returned by default. When explicitly specifying these values, in the same way as the Perl API constants, the `RunProperty` enumeration represents the set of controlled platform-agnostic keys.

The following is a minimal example to query each of the different types of tables. Note that the first step is to construct a map of the values to query. In this particular example, the barcode is explicitly specified, i.e. "ACCGTT". If this was to be omitted, a general summary for lane 1 would be returned instead of a specific sample, allowing for an overview assessment of the run:

```java
String fields = new String[]{
    "TYPE_OF_EXPERIMENT: NGS",
    "SAMPLE_NAME: TEST_SAMPLE",
    "LANE: 1",
    "PAIR: 1",
    "RUN: RUN_NAME",
    "BARCODE: AACTGA"
};
```

**Table 1. Example fields in an analysis metadata table file.**

<table>
<thead>
<tr>
<th>Field</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_OF_EXPERIMENT</td>
<td>NGS</td>
</tr>
<tr>
<td>PATH_TO_ANALYSIS</td>
<td>/path/to/fastqc_data.txt</td>
</tr>
<tr>
<td>ANALYSIS_TYPE</td>
<td>FastQC</td>
</tr>
<tr>
<td>INSTRUMENT</td>
<td>MISEQ-1</td>
</tr>
<tr>
<td>CHEMISTRY_VERSION</td>
<td>TRUSEQ_SBS_V3</td>
</tr>
<tr>
<td>SOFTWARE_ON_INSTRUMENT_VERSION</td>
<td>MCS_2.2.0_RTA_1.17.28.0</td>
</tr>
<tr>
<td>CASAVA_VERSION</td>
<td>1.8.2</td>
</tr>
<tr>
<td>RUN_FOLDER</td>
<td>/path/to/run_folder</td>
</tr>
<tr>
<td>SAMPLE_NAME</td>
<td>TEST_SAMPLE</td>
</tr>
<tr>
<td>LANE</td>
<td>1</td>
</tr>
<tr>
<td>BARCODE</td>
<td>AACTGA</td>
</tr>
<tr>
<td>PAIR</td>
<td>1</td>
</tr>
<tr>
<td>RUN</td>
<td>RUN_NAME</td>
</tr>
</tbody>
</table>
// Setup the query arguments
Map <RunProperty, String> properties = new HashMap<>();
properties.put (RunProperty.lane, "1");
properties.put (RunProperty.barcode, "ACCGTT");
properties.put (RunProperty.run, "RUN-123");

// Get summary values of the run
ReportTable table = r.getAverageValues (properties);
log.info (table.toJSON());

// Get the quality mean of the selected run across partitions
table = r.getPerPartitionValues ("quality_mean", properties);
log.info (table.toCSV());

// Get the quality score count per base position
table = r.getPerPositionValues ("quality_score_count", properties);
log.info (table.toCSV());

All javadoc for the StatsDB Java API can be found at https://repos.tgac.ac.uk/statsdb/javadoc/latest/.

Parsers
StatsDB parsers are small programs, usually scripts, which take the output of a QC tool and use one of the APIs to store the data. Parsers for StatsDB should have the same structure, e.g. adhering to the contractual Java interfaces, and only ever need implement the specific parsing code for the analysis to be added.

Both parsing APIs require analyses to be defined in an analysis metadata table (see Table 1). This is represented as a simple tab-delimited flat file describing the list of analysis fields and values. This file should supply one analysis per line, for example a path to a FastQC data file and related properties.

Every new parser should conform to the contract shown in Figure 3. To write a parser it is not necessary to know the schema of the database or to modify the contents directly. StatsDB conveniently provides the database access objects within its APIs which then connect to the database. The Wrapper object represents a managerial entity that calls a SpecificParser to actually undertake the parsing, but also manages connections to the underlying StatsDB DB object. The Wrapper should open a connection to the database, then create a RunTable object to contain a list of runs, its properties and a path to the analysis to be stored, as described by the metadata table. The path

Figure 3. StatsDB parser interaction diagram, using the Perl API. To add new parsers, only the SpecificParser has to be written. The rest of the classes abstract the interaction with the database and ensure the consistency of the data. The Java API utilises different class and method names but employs the same generalised interactions.
An example \textit{Wrapper} application would be implemented in Perl as follows:

```perl
# Opens a connection to the database
my $db = QCAnalysis::DB->new();
$db->connect ($config);

# Reads the metadata table file (Figure 3)
my @analysis = QCAnalysis::RunTable->parse_file ($input);

# Iterates over each analysis to add
foreach (@analysis){
  # Gets the path to the file
  # The argument is the column name
  my $fast_qc_file = $_->get_property ("path_to_counts");

  # Executes specific TagCount parser
  QCAnalysis::TagCount->parse_file ($fast_qc_file, $_);

  # Inserts the analysis to the database
  $db->insert_analysis ($_);
}

# Closes the connection to the database.
$db->disconnect();
```

A concrete \textit{Wrapper} and \textit{SpecificParser} implementation that interrogates FastQC output can be found in \texttt{Perl/parse_fastqc.pl} and \texttt{Figure 4}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image.png}
\caption{Example of the execution of a query to the database. The client only needs to be aware of the Java/Perl API and StatsDB will format the result in CSV or JSON, so that the client can display the summary.}
\end{figure}
Perl/QCA/alysis/FastQC.pm in the Github repository. Similarly, the Java version of a SpecificParser can be found in FastQCReporterParser.java. Examples of wrapper construction of the parser and analysis objects can be seen in the StatsDBApp.java and TestFastQCParser.java classes.

Consumers

Consumers are programs that process StatsDB data through the API. StatsDB provides a comprehensive API to query for summaries or results of an analysis related to a specific run. The client needs to know if the value is global to the run, per position, or per partition in order to select the relevant method to call. To provide a consistent interface, all the queries are summaries and the following properties can be used as selecting criteria: encoding, chemistry, instrument, softwareOnInstrument, typeOfExperiment, pair, sampleName, lane, barcode, run. If all the properties are specified, only the latest corresponding analysis for the run is returned. This approach allows consumers to make complex comparative analyses.

The Perl API examples/example_consumer.pl script provides example calls to the Reports.pm reporting module, which forms the basis for any consumer implementation. Similarly, to facilitate these complex queries in the Java API, we provide a ReportsDecorator class which contains specific methods to interact with each report, but also encapsulates related reports. One example is getPerPositionBaseContent() which produces a matrix of the base content of each base per position, rather than producing individual queries for each base.

Figure 4 shows the general approach to query the database. The consumers call the corresponding summary or the report decorator. The queried properties are then used to call the stored procedures. StatsDB reads the result set to produce an internal representation of the table, which can then be used directly by the consumer, or can be formatted as CSV or JSON. For further details and the available methods, the inline perldoc or javadoc documentation provides a comprehensive description of each method.

Reporting tools such as FastQC produce informative plots to visualise run metrics. StatsDB supplies example plots based on those produced by FastQC, written using the d3.js Javascript library, to demonstrate downstream processing and representation of consumer-generated StatsDB JSON. In this way, developers can easily integrate StatsDB plots on their own web pages. A dedicated helper class to generate a set of required JSON fragments is supplied, i.e. D3PlotConsumer.java, and downstream Javascript and HTML that renders the JSON fragments can also be found in the GitHub repository in the Web directory. Examples of interactive plots generated from FastQC per-base quality and per-base GC content can be seen in Figure 6.

Use case

At TGAC, we use StatsDB as part of our Primary Analysis Pipeline. Each Illumina run sequenced at TGAC, both with HiSeq and MiSeq instruments, passes through this pipeline. Two important steps in the process are QC analysis with FastQC and contamination

Figure 5. StatsDB Reporter allows overlay plots of QC data across instruments and runs.
analysis using an in-house kmer-based screening tool. The output of both of these tools is parsed using two separate Perl scripts provided as part of the StatsDB package, and loaded into StatsDB. As well as these QC output, we load details of the instrument, chemistry version, RTA version and Casava version into StatsDB.

Our PacBio primary analysis pipeline is still under development, but this currently includes using FastQC to analyse FASTQ files output as part of the process. We are also working on parsers for the sts.csv files that are produced by the instrument. With StatsDB, it is perfectly possible to mix data from different platforms and different tools into one database. When querying the data, the consumer application can make the decision about what data comparisons are meaningful.

We currently access data stored in StatsDB using two consumers: TGAC's open-source LIMS, MISO, and the prototype StatsDB Reporter tool. Using the MISO web-based interface, it is possible to access StatsDB information and produce graph plots of FastQC data, which are based on the d3.js consumer examples supplied with the framework (Figure 6). StatsDB Reporter allows selection

![Figure 6. Examples of d3.js plots, generated from FastQC data parsed into StatsDB.](image)
of runs by instrument, lane, run, sample or barcode and provides comparison of data across runs (Figure 5).

Conclusions
Many software tools exist for the generation of run quality statistics and FastQC is possibly the most notable example. However, until now, there has been no easy solution to the problem of storage and analysis of historical run metadata and statistics. StatsDB has been designed to address this problem and offers a flexible, easy to use, platform-agnostic and tool-independent framework for consolidated access to run metrics.

Installation of StatsDB and integration with existing analysis pipelines is achieved with minimal effort. To perform data entry into StatsDB, a parser is included for the popular FastQC tool, and parsers for other tools can be written in less than a day by a competent programmer or scripter. Similarly, command line tools are provided in both Perl and Java to load parsed data into StatsDB.

To perform downstream analysis and visualisation of data held within StatsDB, reporting helper entities are provided. Furthermore, an example consumer tool is supplied in the form of the StatsDB Reporter application which currently exists in prototype form but a mature version will be available in due course from the GitHub repository for StatsDB, as well as parsers for other tools (for example, PacBio sts files).

Software details

Author contributions
MC initiated the project, which RD now leads. RHRG designed the database, implemented the Perl API and, with RD, implemented the Java API. RML and DW wrote parsers and consumers to use the API. AT developed the d3.js plots. RML, RHRG, RD, DW and AT all contributed to the manuscript.

Competing interests
The authors declare that there are no competing interests.

Grant information
The development of StatsDB has been funded by a Biotechnology and Biological Sciences Research Council (BBSRC) National Capability Grant at TGAC.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements
We are grateful to the Library Preparation and Sequencing Operations teams at TGAC for generating the data that has been used to test StatsDB.

References
10. Skelly T: PacBio Exploratory Data Analysis. Reference Source
12. Perl DBI. Perl DBI. Reference Source
Open Peer Review

Current Peer Review Status: ✔️  ❓  ✔️

Version 1

Reviewer Report 27 December 2013

https://doi.org/10.5256/f1000research.2894.r2794

© 2013 Kumar A. This is an open access peer review report distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Anuj Kumar
Department of Molecular, Cellular & Developmental Biology, University of Michigan, Ann Arbor, MI, USA

The manuscript by Ramirez-Gonzalez et al. presents a software package to store and analyze metrics from next-generation sequencing runs. The open-source software package, StatsDB, stores statistics from sequencing reactions in a MySQL database, and APIs provide means to access and analyze the data. StatsDB provides modules that can be used in conjunction with FastQC as well as with other tools suited for the analysis of sequence data.

From the perspective of a wet-lab genomicist, the StatsDB package seems to be a useful tool, and I can certainly see the utility in storing data from the sequencing reactions, particularly for a genome analysis center. I only have a few minor comments.

1. Since FastQC is commonly used for the analysis of sequencing runs, and since the StatsDB package works with FastQC, it would be useful to provide a little more explicit information describing the output/types of data provided by FastQC. Some of these data are presented in the StatsDB schema, but it would be nice to add a bit more detail in this regard (possibly a very simple figure) into the Introduction.

2. I appreciate the use case provided, but I think it would be helpful to provide an additional small example of how the StatsDB package could be useful to an individual lab that uses a moderate level of next-generation sequencing in their research (as opposed to a genome center). Possibly, a summary of the StatsDB Reporter application would suffice.

3. It would be helpful to add a few sentences to the Introduction to indicate how other sequencing centers, etc. store output from FastQC or other similar tools. This does not have to be an extensive overview.

Competing Interests: No competing interests were disclosed.
I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Robert Davey, Earlham Institute, UK

We would like to thank Anuj for taking the time to read and comment on our manuscript.

We have added a sentence in to the Introduction, outlining briefly the data formats outputted by FASTQC, i.e. a set of HTML files, and a single plain-text flat file from which we parse the data to be loaded into StatsDB.

We envisage no differences in scaling terms between a small-scale lab with one or two sequencers and a large multi-platform centre. StatsDB has applications in the smaller centre where quick but potentially more sporadic access to historical run data would be investigated through the StatsDB Reporter tool rather than the more "heavyweight" integration with a LIMS or via a web server. As such, we have added a sentence describing its relevance in this context. We will be publishing the StatsDB Reporter application separately, or as a software update to this publication, in due course.

We have added a short overview of how the output of QC tools might be kept in different centres, highlighting the usefulness of StatsDB.

Competing Interests: No competing interests were disclosed.
2. Page 4, Database design - The relationship between the analysis table and the auxiliary views is unclear. For example, the analysis table stores a unique ID and timestamp of when an analysis run was performed. But the latest_run view mentions that an analysis can be run more than once. Please clarify.

3. Page 10, Use case - Add a figure with a flowchart showing the TGAC pipeline that is explained in the text. It is critical to show how StatsDB plugs into a standard analysis pipeline.

4. For the source code, an LGPL license might be more appropriate than GPL because StatsDB is an API, rather than a standalone tool in a workflow. LGPL would also allow parsers/consumers for manufacturer-supplied metrics.

Minor comments:
1. Page 3, Database installation - The code snippets show file names with accidental whitespace. Specifically: statsdb_schema.sql, stored_procedures.sql, and parse_fastqc.pl

2. Page 3, Introduction - "With this data heterogeneity and abundance, comes a..." - missing comma.

3. Page 5, Database design - How would you store metrics that depend on genomic positions or ranges? e.g targeted sequencing coverage. If tables per_position_value and per_partition_value stored values based on genomic positions or ranges, it would also depend on which reference sequence build (e.g. NCBI36, GRCh37) was in use for that analysis. Would we define that build as a value_type? And in the stored procedures used by the APIs, how can we handle a mix of different reference builds used across runs?

4. Page 7, Java API - accidental whitespace in filename fastqc_data.txt

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 24 Jan 2014

Robert Davey, Earlham Institute, UK

We would like to thank Cyriac for taking the time to read and comment on our manuscript.

In terms of the Use Case expansion, we believe the usefulness is inherent in the ability to store, retrieve and therefore compare historical run metrics, based on a variety of user- or analysis-specific attributes. We envisage tools like StatsDB Reporter will be the simplest and more widely-used interface with StatsDB for the average lab technician or bioinformatician. Similarly, we believe performance metrics at this level would be unhelpful rather than beneficial, as these would be very dependent on infrastructure, hardware and DMBS used. We aim to publish incremental updates to the API and surrounding tools, e.g. a full StatsDB Reporter release, and a Python API.

The StatsDB schema was designed to allow the same analysis to be stored multiple times, with potentially differing or identical parameters. We foresee that if analysis uniqueness is required, this would be down to an API implementation to check for previous analyses with the same tool and
parameters supplied, rather than at the database level. We have added a description to the Database Design section (analysis and latest_run table outlines) of the manuscript.

The pipeline that utilises StatsDB has been covered in detail in our recent open-access FrontiersIn publication, so we have added a reference to this paper in the Use Case main text.

The GPLv3 aims to give free software developers an advantage over proprietary developers. A parser that supports a proprietary format does not fit within the scope of our vision - we aim to continue the trend in bioinformatics software whereby fully open-source licences are preferred to maximise the reusability of a given tool or library.

Yes, you are correct in using value_type for this kind of attribute. There are API calls that let you pull out analyses by a value_type value, so this should be supported out of the box.

The accidental whitespace has been introduced in the HTML view, from which the PDF download is generated. We shall contact the editorial office to ensure these are corrected. Thank you for spotting those!

**Competing Interests:** No competing interests were disclosed.

---

Reviewer Report 10 December 2013

https://doi.org/10.5256/f1000research.2894.r2467

© 2013 Watson M. This is an open access peer review report distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

---

Mick Watson
Department of Genomics and Genetics, Roslin Institute, Royal (Dick) School of Veterinary Studies, University of Edinburgh, Edinburgh, UK

The article is well written and describes a tool which will be very useful to the community.

I only have a few minor issues:

1. Throughout I felt that the terms used were confusing: "analysis" and "run" for example. These can take on multiple meanings in sequencing (for example, what the SRA calls a run is not what I call a run). I feel these terms need a better definition when they are first introduced, with examples where appropriate.

2. Is it valid to still use the term "short-read" sequencers?

3. There is a lot of technical information about the design, database and APIs, but only a single example (using Illumina data) at the end; examples using PacBio and Ion Torrent data would show the true flexibility of the tool.

**Competing Interests:** No competing interests were disclosed.
I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 24 Jan 2014

Robert Davey, Earlham Institute, UK

We would like to thank Mick for taking the time to read and comment on our manuscript.

Admittedly, there is much interplay when considering terminology such as “run” and “analysis”, but throughout the paper we use the term “run” to represent a sequencing run, and similarly, the term “analysis” to represent a QC process. We feel this is adequate given the focus of the paper, but we have clarified one potential misuse (changed “run” to “carried out”, where appropriate).

Yes, the field refers to these terms regularly, given that “long read” sequencers are available and distinct from their “short-read” counterparts, i.e. we consider the new 2x300bp and upcoming 2x400bp Illumina techniques to still be “short-read”.

As long as the data produced from non-Illumina machines is in the FASTQ format (as we state in the existing Use Case text), then there are no differences from the method outlined in the paper, e.g. FASTQC output parsed and stored in StatsDB. Where differences may exist, e.g. STS files from PacBio, different parsers can be written to accommodate this, and we are working on producing such a parser. By no means is StatsDB inherently tied to the FASTQ format, but we feel this reflects the most common QC methods available currently.

**Competing Interests:** No competing interests were disclosed.