An SQL-based approach to physics analysis

Dr. Maaike Limper
CERN IT department, CH-1211 Geneva 23, Switzerland
E-mail: maaike.limper@cern.ch

Abstract. As part of the CERN openlab collaboration a study was made into the possibility of performing analysis of the data collected by the experiments at the Large Hadron Collider (LHC) through SQL-queries on data stored in a relational database. Currently LHC physics analysis is done using data stored in centrally produced “ROOT-ntuple” files that are distributed through the LHC computing grid. The SQL-based approach to LHC physics analysis presented in this paper allows calculations in the analysis to be done at the database and can make use of the database’s in-built parallelism features. Using this approach it was possible to reproduce results for several physics analysis benchmarks. The study shows the capability of the database to handle complex analysis tasks but also illustrates the limits of using row-based storage for storing physics analysis data, as performance was limited by the I/O read speed of the system.

1. Introduction
Within the context of the CERN openlab collaboration with Oracle [1] an investigation was made into using Oracle database software for performing analysis of the data taken by the LHC experiments at CERN, with the aim of demonstrating the capability and possible limitations of the relational database to handle complex analysis tasks. Currently most of the analysis of data taken by the LHC experiments is done using ROOT [2], the dedicated C++ analysis framework developed for and by the High Energy Physics community. This study was based on experience from physics analysis in the ATLAS experiment, but the approach is similar for the other LHC experiments. ATLAS physics analysis is typically done using flat ROOT ntuples that are accessible through the LHC computing grid [3]. An SQL-based physics analysis using data stored in a centrally accessible database could provide an alternative to the distributed computing systems currently used to access data, and allows performance optimization through the use of the Oracle database’s in-built parallelism features [4].

In these proceedings we present a possible schema for storing LHC analysis data and we show how analysis code can be written as SQL. This approach was tested using a small cluster of machines running an Oracle database and compared with the standard ROOT-ntuple analysis running on the same hardware, using two different analysis benchmarks. The first benchmark is a simplified version of the actual analysis and uses relatively few variables, while the second benchmark is based on actual ATLAS analysis code and involves a larger number of variables and computations made through external library functions.

2. Database design
The tests done for this study used a subset of ntuples produced for the ATLAS top-physics group. This subset contained 7.2 million events stored in 127 ntuple-files with around 4000
variables stored per event.

The database design used in this study was chosen to facilitate any generic analysis, which in principle could involve any possible combination of the variables in the data-set. The large number of variables stored per event meant it was not feasible to store all the data in a single table using one column per variable. Instead, the database design was based on creating one table per physics-object. This design helps to reduce I/O, as users need only access the tables relevant for their specific analysis, and makes it easier to apply a pre-selection on the physics-objects involved in the analysis.

Table 1 illustrates the database schema used to store the analysis data. Physics-objects can have zero, one or multiple entries per event. The largest table, holding information about reconstructed photon-objects, had nearly 90 million rows of data for our test-sample of 7.2 million events. The physics-objects are described by many different variables reconstructed in the events, which resulted in the tables having hundreds of columns.

| Table Name      | Millions of rows | Number of Columns | Size (GB) |
|-----------------|------------------|-------------------|-----------|
| photon          | 89.9             | 216               | 114.4     |
| electron        | 49.5             | 340               | 94.6      |
| jet             | 26.8             | 171               | 26.3      |
| muon            | 7.7              | 251               | 14.2      |
| primary_vertex  | 89.5             | 25                | 11.9      |
| EF              | 7.2              | 490               | 7.9       |
| MET_RefFinal    | 7.2              | 62                | 2.3       |
| eventData       | 7.2              | 52                | 7.3       |

Table 1. Table illustrating the database schema used to store the analysis data, showing the number of rows and columns of the database tables for the test sample.

The tables were partitioned by the “RunNumber”-variable, which indicates the data-taking period when the collision events were recorded. This provides a logical partition-unit as it facilitates run-dependent selection strategies and allows the user to limit their analysis to data taken during specific periods.

The data in the database was uncompressed as the volume-reduction using compression was found to be relatively small for our data-set\(^1\). In contrast the ROOT-ntuple format saves data in a binary compressed format and has column-based storage optimized for compression.

2.1. Table constraints and indexes

The PrimaryKey constraint in the `eventData`-table is defined by `(RunNumber,EventNumber)` column-pair as this uniquely identifies collision events in each run. For data consistency all other tables have a ForeignKey that refers to the PrimaryKey in the `eventData`-table, as well as their own PrimaryKey that uniquely identify the objects they hold.

The SQL performance can be influenced through the creation of indexes on specific table-columns. In this database all tables have an index on `(RunNumber,EventNumber)` to allow users to directly access specific events. For uniform performance results, and as the analysis-query might use any of the hundreds of columns in each table, no additional indexes were created on any other columns in the tables.

\(^1\) Using basic-compression on the tables reduced overall data-size by approximately 10%.
3. Analysis code in SQL

LHC physics analysis typically starts with the pre-selection of specific objects from the collision events, after which the information of selected objects is combined to determine what type of event took place. The SQL-versions of our analysis benchmarks were built through a series of select statements on each object-table, each with a WHERE-clause to apply selection criteria. These pre-selection can be defined within the query using the WITH-AS statement as follows:

```sql
WITH
goodmuon AS (SELECT E, px, py, pz, ... FROM muon WHERE pt > 25. AND ...),
goodjet AS (SELECT E, px, py, pz, ... FROM jet WHERE pt > 20. AND ...),
```

Or, alternatively, pre-selection can also be done by explicitly creating new tables holding the objects, and using these as input for further analysis. This increases analysis time by the time needed to create these tables, and requires the user to have sufficient space available in his/her schema, but has the advantage of speeding up future analysis using the same pre-selection.

JOIN statements on (RunNumber,EventNumber) can be used to put information from the different selections together. The final select will determine the variables that are returned to the user. PL/SQL functions can be used in the selection to calculate new variables. For example, a SELECT-statement returning the invariant mass of selected muon- and jet-pairs looks like:

```sql
SELECT RunNumber, EventNumber, inv_mass(jet1, jet2), inv_mass(mu1, mu2)
FROM goodmuon mu1
INNER JOIN goodmuon mu2 USING (RunNumber, EventNumber)
INNER JOIN goodjet jet1 USING (RunNumber, EventNumber)
INNER JOIN goodjet jet2 USING (RunNumber, EventNumber)
WHERE mu1.muon_id != mu2.muon_id AND jet1.muon_id != jet2_muon_id ...
```

3.1. Using hints

The Oracle SQL optimizer uses statistics gathered on the table-data to determine the optimum execution plan. In practice, the SQL optimizer did not always manage to identify the optimum plan with our data and queries, so we used “hints” inside the analysis-query to optimize the query performance.

All sub-selections involved in the final JOIN were resolved as global temporary tables before starting the join-procedure, using the MATERIALIZE-hint:

```sql
muon_event AS (SELECT /*+ MATERIALIZE */ RunNumber, EventNumber ... )
```

The FULL hint was used when the optimizer underestimated the number of rows returned from a sub-selection and the JOIN statements on the RunNumber,EventNumber attributes would trigger an “access by index” while a full table scan would have been faster:

```sql
goodjet AS (SELECT /*+ FULL */ E, px, py, pz, ... FROM jet WHERE ...)
```

3.2. External code

The user might be not be able (or willing) to write all analysis code as SQL statements and PL/SQL functions, especially as LHC analysis often involves references to existing C++ classes for specific calculations. Functions from C++ libraries can be called from inside the SQL-query by writing a PL/SQL stored subprogram using the EXTERNAL clause, which lets C-code be called through Oracle’s external procedure agent. However, calling the external libraries in this way introduces an additional overhead from the external procedure agent. In this study we used PL/SQL wrappers to invoke java-methods that call the C++-libraries through the Java Native Interface (JNI)^2, as this significantly reduced the overall analysis time.

^2 The Oracle documentation states that: “Oracle Database does not support the use of JNI in Java applications”
Table 2 illustrates the time gain of calling the external functions via Java on our test setup, for one of the pre-selections used in the analysis benchmarks described in the section 4.

| “b-jet selection”-query with:                      | Query time   |
|---------------------------------------------------|--------------|
| PL/SQL calling C library                          | 852 seconds  |
| PL/SQL calling Java calling C library             | 368 seconds  |

**Table 2.** Query time for the “b-jet selection” (without parallel execution) with two different approaches to call the external C++-library involved in this selection.

4. **Benchmarks**

Two SQL-based analysis benchmarks were implemented that could produce the exact same results as the equivalent analysis from ROOT-ntuples. These benchmarks were executed using ROOT’s *TOracleServer*-class, which sends a string (the SQL-query) to an Oracle-database and returns the results within the ROOT-macro. The string for the SQL-query is constructed within the C++-classes by adding lines for the different sub-selections within the query using the WITH-AS clause. This approach allows the output to be directly displayed using the ROOT-histogram tools, and could also be used to do bulk data-processing through SQL while more complex processing is done afterwards within the C++ framework.

4.1. **Benchmark 1: Higgs+Z selection**

This benchmark is a simplified version of the search for the Higgs (decaying into two bottom-quarks) in association with a Z-boson (decaying into two leptons) which involved the selection of two good leptons (muons or electrons) and two good b-jets\(^3\), applying additional constraints on the selected particle-pairs and finally returning the invariant mass of the b-jet pair.

This analysis was implemented both as a single ROOT-macro, containing a loop over all the events in the ntuple-files, and as a ROOT-macro calling an SQL-query. This benchmark required 40 different variables and used external code to re-calculate the b-tagging likelihood that identified which jet-objects were likely to have been produced by b-quarks.

4.2. **Benchmark 2: ttbar cutflow**

In addition, a cutflow analysis for the top-pair production cross-section measurement was implemented as a more complicated benchmark. This analysis returns the cutflow of the top-quark-pair (ttbar) selection which is the number of events remaining after the addition of each event-selection cut.

For this benchmark the original “RootCore”-packages were compared to a modified set of packages that construct a string forming the SQL-query used to retrieve the cutflow result. This benchmark involved 262 variables, and used data from the same tables as the Higgs+Z benchmark as well as data from the photon-table. The photon and electron selection involved the use of external library functions and an INNER JOIN with the *primary_vertex*-table, to apply pile-up\(^4\) corrections to the reconstructed variables before the selection criteria were applied. As for the Higgs+Z selection, the jet-selection also used an external library to identify b-jets.

---

\(^3\) Here a jet defined as a spray of particles, which possibly indicates quark-production in the collision-event. A b-jet is a jet resulting from bottom-quark production.

\(^4\) Here pile-up refers to the number of collisions that occurred within the event.
5. Test results
The two benchmarks were used to make a comparison of analysis times and resource usage between the SQL-based approach running on the database and the standard approach from ROOT-ntuples. All reported analysis times are “cold”, the file-system was flushed prior to running the benchmarks, and the database buffer-cache was flushed for the SQL-based analysis.

5.1. Test setup
The test setup used to obtain the results presented in this paper consisted of five machines (8-core each) connected via two network links to five disk-arrays, with each array holding 12 disks. The limit on the I/O read speed in this cluster is 2.5 GB/s due to the 4 Gb/s speed of the host-bus adapter on the disk-arrays. The test setup was also used for a comparison of sequential data access between Oracle and Hadoop and more information about our test setup can be found in the paper corresponding to this study [5].

On this setup we created an Oracle Real Application Cluster (RAC, version 12c) to run the database software simultaneously on all five machines while accessing a single database. The Oracle database made use of the Automatic Storage Management (ASM) feature to spread the data evenly over all devices. To make a fair comparison with the ntuple-analysis on the same setup, ROOT (version 5.34.09) was installed on all five nodes and the ntuple-files were distributed evenly over all disks.

5.2. I/O reads
The analysis-queries accessed the data in the database primarily through full table scans, which due to the row-based storage in combination with relatively wide tables resulted in a large volume of I/O reads. In contrast, the ROOT-ntuple analysis required less I/O reads as it used column-based storage to allow accessing only the branches containing the variables needed for each analysis. Table 3 shows the volume of the I/O reads for the SQL-based analysis running on the database and for the ROOT-analysis from ntuples, for the two different benchmarks.

| Benchmark:          | I/O read volume from DB | I/O read volume from ntuples |
|---------------------|-------------------------|------------------------------|
| Higgs+Z selection   | 154 GB                  | 14 GB                        |
| ttbar cutflow       | 317 GB                  | 87 GB                        |

Table 3. Total volume of I/O reads for the benchmarks on our test sample.

For the Higgs+Z benchmark the volume of the I/O reads was a factor of 11 larger for the analysis from the database compared to the ntuple-analysis, while for the ttbar cutflow benchmark, which involved more physics-objects and more variables, the volume of I/O reads was “only” a factor of 3.6 larger.

5.3. Effect of parallelism
An SQL query can be executed in serial or in parallel and the degree of parallelism can be set on the table or by a hint inside the query. For the ntuple analysis, parallelism was mimicked by running multiple simultaneous ROOT-jobs, each analyzing a subset of files. A comparison of the benchmark analysis times between the database-analysis and ntuple-analysis for increasing degrees of parallelism is shown in figures 1 and 2.

The results show that the ntuple-version initially gains significantly more from parallelism than the database-version of the analysis. This is because the DB-version is limited by I/O read speed as a much larger volume of data is scanned to find the relevant variables.
On our test setup, the Higgs+Z benchmark from ntuples becomes faster than the database-version starting from 20 simultaneous ROOT-jobs. In contrast, for the ttbar cutflow benchmark the SQL-based version was always faster than the standard version, and the time for the ntuple-version even increased from 30 to 40 ROOT-jobs as the CPU resources on the cluster got completely saturated. This was mainly due to the significantly higher CPU-usage of the ntuple-based version of the ttbar cutflow benchmark, as will be shown in the next section.

5.4. CPU usage
Figures 3, 4, 5, and 6 show the average CPU usage on the machines, while running the two analysis benchmarks for the two different approaches, using 40 sub-jobs for the ntuple-analysis and a parallel degree of 40 for the database-version. The results show that for the simplified Higgs+Z benchmark, for both versions of the analysis, the time was dominated by iowait-events.

For the ttbar cutflow benchmark, the analysis time from the ntuples using 40 sub-jobs is shown to complete saturate the CPU on our system. The database-version of the ttbar cutflow benchmark, while clearly needing more CPU compared to the Higgs+Z benchmark, still has a significant amount of iowait-events.

6. Conclusion
The studies done in this paper demonstrate the possibility of doing complex data analysis through SQL-queries on data stored in a relational database. The database design was based on creating separate database-tables for different physics-objects, so users need only access those tables containing the data relevant for their analysis. In practice, the physics-objects described in the test data-set still had many more variables than were needed for the benchmark analysis. This resulted in a larger volume of I/O reads required for the analysis from the database than for the corresponding analysis from the ROOT-ntuple files, as the ROOT-ntuple has a column-based storage structure making it possible to read only the data for the relevant variables.

The comparison of the SQL-based approach with the standard ROOT-ntuple analysis for different benchmarks in our test-setup showed that the performance depended on the number of variables used in the analysis and the CPU usage required in the selection. The improvement of analysis time with increasing degree of parallelism on the Oracle database was relatively little, as the analysis was limited by the I/O read speed of system. Future studies will focus on the use of column store features to improve the performance of the SQL-based physics analysis approach.
Figure 3. CPU usage for the ntuple-analysis running the simplified Higgs+Z benchmark, with 40 simultaneous sub-jobs.

Figure 4. CPU usage for the SQL-analysis running the simplified Higgs+Z benchmark, using PARALLEL 40.

Figure 5. CPU usage for the standard ntuple-analysis running the ttbar cutflow benchmark, with 40 simultaneous sub-jobs.

Figure 6. CPU usage for the SQL-analysis running the ttbar cutflow benchmark on the test-setup, using PARALLEL 40.

References
[1] Gaillard M 2013 First Annual report of CERN openlab phase IV See also http://openlab.web.cern.ch
[2] Brun R and Rademakers F 1997 ROOT - An object oriented data analysis framework Proc. AIHENP’96 Workshop, Nucl. Inst. & Meth. in Phys. Res. A. 389 81-86. See also http://root.cern.ch
[3] Legger F on behalf of the ATLAS Collaboration 2013 The ATLAS distributed analysis system Proc. Computing in High Energy Physics (CHEP) 2013 (these proceedings)
[4] Bellamkonda S, Li H, Jagtap U, Zhu Y, Liang V and Crunanes T 2013 Adaptive and big data scale parallel execution in Oracle Proc. of the VLDB Endowment 6 1102-13
[5] Baranowski Z 2013 Sequential data access with Oracle and Hadoop: a performance comparison Proc. Computing in High Energy Physics (CHEP) 2013 (these proceedings)