

DM

Data Management Group

CERN IT  
Department

# Partitioning in COOL

**Andrea Valassi (CERN IT-DM)**

**R. Basset (CERN IT-DM)**

***Distributed Database Operations Workshop,  
11th November 2008***



- **Motivation**
- **Option 1 – homemade partitioning**
  - COOL database ‘federation’
- **Option 2 – Oracle partitioning**
  - Table and index partitioning
  - Performance implications and tests
- **Conclusions**





# Motivation

- **Data management – for instance:**
  - Several smaller, more manageable data chunks
  - Declare old partitions as read-only
  - Take old partitions offline
  - Export individual partitions by file copy
  - Import partition into existing COOL database
- **Performance is not the argument**
  - Scalability tests with 10 years of Atlas data show good SQL performance for index-based queries
  - But must ensure performance is not degraded by whichever partitioning solution we choose





# Use cases in COOL

- **Single-version data**
  - e.g. temperatures (copied from PVSS to COOL)
  - Very large samples (Atlas ~200 GB/year)
  - No write access for old data
    - Measured data is strictly historical data
  - Infrequent read access to old data?
- **Multi-version data: not a candidate?**
  - e.g. calibration, alignment
  - Relatively small data samples
  - Write access for old data
    - Compute new calibrations using new algorithms
  - Frequent read access to old data





# Homemade partitioning

- **COOL database federation**
  - Attach validity range to each COOL ‘database’
  - Federation (‘super-database’) includes several databases covering different validity ranges
    - MUST ensure there is no overlap between databases: start- and end-of-validity must fall within same partition
- **This partitioning is not ‘transparent’**
  - New metadata tables needed in COOL schema
    - Keep track of which data is in which COOL database



# Oracle partitioning

- **Transparent partitioning**
  - No new tables needed in COOL schema
  - Data management using Oracle tools (add/split, offline, read-only, transportable tablespaces...)
- **Table and index partitioning**
  - Partitioning key: start-of-validity (IOV\_SINCE)
    - Better choose partition boundaries so that end-of-validity (IOV\_UNTIL) also falls within the same partition
  - Must use local (partitioned) indexes
    - Global indexes hinder partition-wise data management
  - Technicality: must declare tables as partitioned at creation time (not an 'alter table' property)



# Oracle partitioning – performance

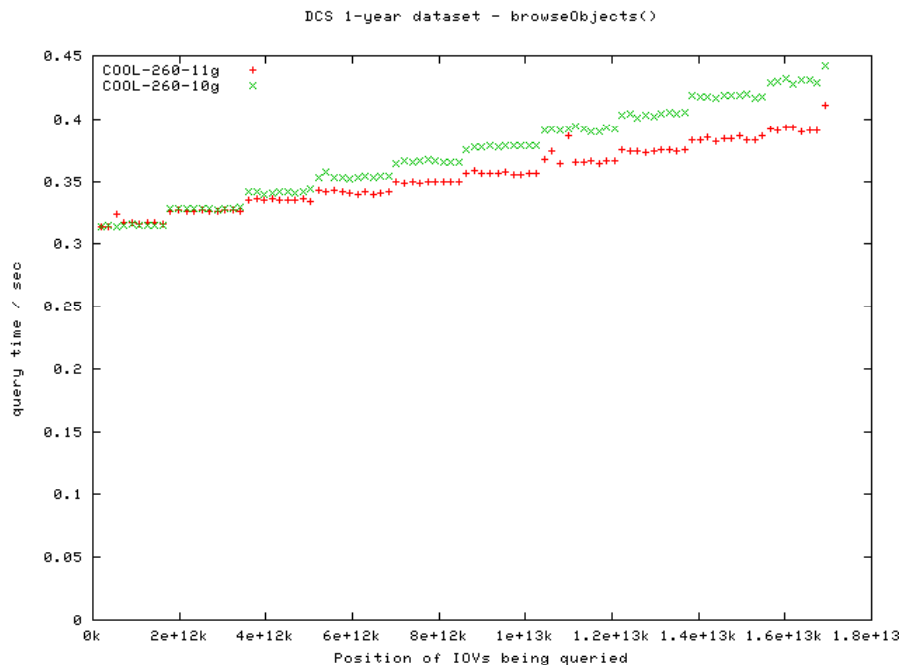
- **IOV retrieval from a partitioned table**
  - Ideally, Oracle partition pruning ensures that:
    - First, the relevant partition is located
    - Then, the IOV is located within the partition
  - Do not expect large (any?) benefit over non-partitioned query if indexes were already used
    - Loop over partitions vs. loop over index branches
- **Test with 3 samples of ‘Atlas 10 years’**
  - Non-partitioned
  - Partitioned tables, global indexes
  - Partitioned tables, local indexes
  - *Work done by Romain Basset*





# Oracle partitioning – tests

- **Observe slight performance degradation**
  - **Caution: work in progress!!!**
  - Loop on partitions less efficient than index scan?
  - Complex SQL with SELECT MAX() subquery
  - Differences between 10g and 11g also observed
    - Affected by bug 6029469 fixed in 10.2.0.5 / 11.1.0.7?



Query response time is flat within each partition, but increases from one partition to the next







# Conclusions

- **Two options are being considered**
  - Custom partitioning (database ‘federation’)
    - No work in progress on this option
  - Oracle partitioning
    - Performance tests are in progress
- **They are not mutually exclusive**
  - Internally, a database in a COOL federation can be partitioned using Oracle partitioning





# Reserve slides





# COOL data model

- **Modeling of conditions data objects**
  - System-managed common **“metadata”**
    - Data items: many tables, each with many channels
    - *Interval of validity* - “IOV” [since, until]
    - *Versioning information* - with handling of interval overlaps
  - User-defined schema for **“data payload”**
    - Support for fields of simple C++ types
  
- **Main use case: event reconstruction**
  - *Lookup data payload valid at a given event time*

<i>objectID</i>	<i>channelID</i>	<i>since</i>	<i>until</i>	<i>pressure</i>	<i>temperature</i>

**Metadata**

System-controlled  
*(versioning metadata not shown)*

**Data payload**

User-defined schema  
*(different tables for different schemas)*

