Conditions Database

Status Review

Andrea Valassi

(CERN IT-DB)

Outline

- Status report
- General comments: the scope of the project
- Software review: data model, API, implementation
- Summary of my proposal and conclusions

Introduction

- **Project launched in summer 2003 (within LCG Persistency Framework)**
  - Background in 2000-2003:
    - C++ API definition and Objectivity implementation
    - Oracle implementation of the original API ("BLOB" data payload)
    - API extensions and MySQL implementation (user-defined relational data payload)
  - Two goals for the common project:
    - Integrate the existing Oracle and MySQL packages into LCG Application Area
    - Coordinate new development of API, software and tools

- **Status overview**
  - Kick-off workshop at CERN in December 2003
  - Activity along two directions in parallel
    - Integrate the existing software into LCG Application Area
    - Review two APIs and implementations, coordinate discussion about new developments
  - *Main problem so far: lack of committed manpower*
    - *New developments also slowed down by the divergence in the two APIs*

Status: organization and manpower

- **Organization**
  - Weekly phone meetings since mid-April (typical attendance 5-10 people, mainly Atlas)
  - General mailing list ~ 60 people, low traffic (general discussions)
  - Developers mailing list ~ 30 people, higher traffic (specific issues and meeting follow-up)
  - No formal work package structure, very few people
  - Software release built in LCG AFS area (CondDBMySQL also in parallel for Atlas)

- **Manpower in first 9 months of 2004**
  - Andrea (~0.5 FTE in 2004 on LCG CondDB): ~ 5 man-months on LCG CondDB
    - General coordination, meeting organization, project web doc, status reviews, HVS design ~ 3 m.m.
    - Build infrastructure, integration/release, API refactoring, CondDBOracle maintenance ~ 2 m.m.
    - Rest of the time: Oracle Physics Database Services (incl. Oracle client kit development/support)
  - Antonio, Luis, Dinis, Nuno, Tiago (~3 FTEs? in 2004 on Atlas CondDB + LCG CondDB)
    - Atlas CondDB: Athena integration, user support, data management, data browser, MySQL servers
    - Atlas and LCG CondDB: CondDBMySQL code and tools maintenance/doc and LCG integration
  - Sven (~0.8 FTE since May on LCG CondDB): ~ 4 man-months on LCG CondDB
    - DataCopy and CondDBCommon code and tools development/doc
  - In addition: Vakho (since June)
    - Atlas Detector Description and HVS (may become an LCG CondDB component, but is not yet)

Status: software releases

- **Release CONDDB_0_1_0 (April 2004) – first public release**
  - Most recent Oracle and MySQL implementations (integrated in LCG CVS and SCRAM)
  - CondDBOracle: original common API (only BLOBs) – only for gcc2.95.2
  - CondDBMySQL: Lisbon extended API (BLOBs and ICondDBTable)
  - Separate API and examples for the two packages
- **Release CONDDB_0_1_1 (May 2004)**
  - Full support for gcc3.2.3 (Oracle OCCI for gcc3.2.3), functionality as in CONDDB_0_1_0
- **Release CONDDB_0_2_0 (July 2004)**
  - Common dependency on API package ConditionsDB (~original API, only BLOBs)
  - Lisbon extensions (ICondDBTable and others) in CondDBMySQL
  - Same functionality and packaging as CONDDB_0_1_1, only packaging changed
- **Next release CONDDB_0_3_0 (October 2004?)**
  - Common dependency on library package CondDBCommon (SimpleTime implementation)
  - DataCopy and Utilities packages with libraries/tools to extract/copy MySQL data
  - Maybe: possibility to link together both packages and copy data across implementations?
- **Future releases CONDDB_0_4_x**
  - Integration (common dependency?) with SEAL: CondDBOracle/MySQL as SEAL plugins
  - Integration with POOL: POOL string token example, copy POOL data too from DataCopy

Status: users *(to my knowledge)*

- Only one production user so far: Atlas test beams
  - *Essentially using only the work done by the Lisbon group*
    - Only *MySQL version* used in production
    - Only *extended API* used in production (no BLOBs)
    - Atlas-specific software installation (not from central LCG installation)
    - Software integration with Athena and PVSS
  - Writers: online and offline
    - Online (PVSS interface): all data from DCS, no filtering, stored when values change
    - Offline: output from Muon alignment program
    - Data size ~10 GB in 2000 folders/tables
  - Readers: online and offline
    - Online: experts debugging their detector (CondDB used as/instead of PVSS archive)
    - Offline: input to Muon alignment program
    - Offline: Athena code reading output from Muon alignment program
- Other activities
  - Tests in LHCb: plan offline readers only, BLOBs or POOL only (Oracle/MySQL)
  - Tests in CMS; also ideas on registering in CondDB data from preexisting tables
  - No production use of CondDBOracle (except for pre-LCG version in Harp)

Outline

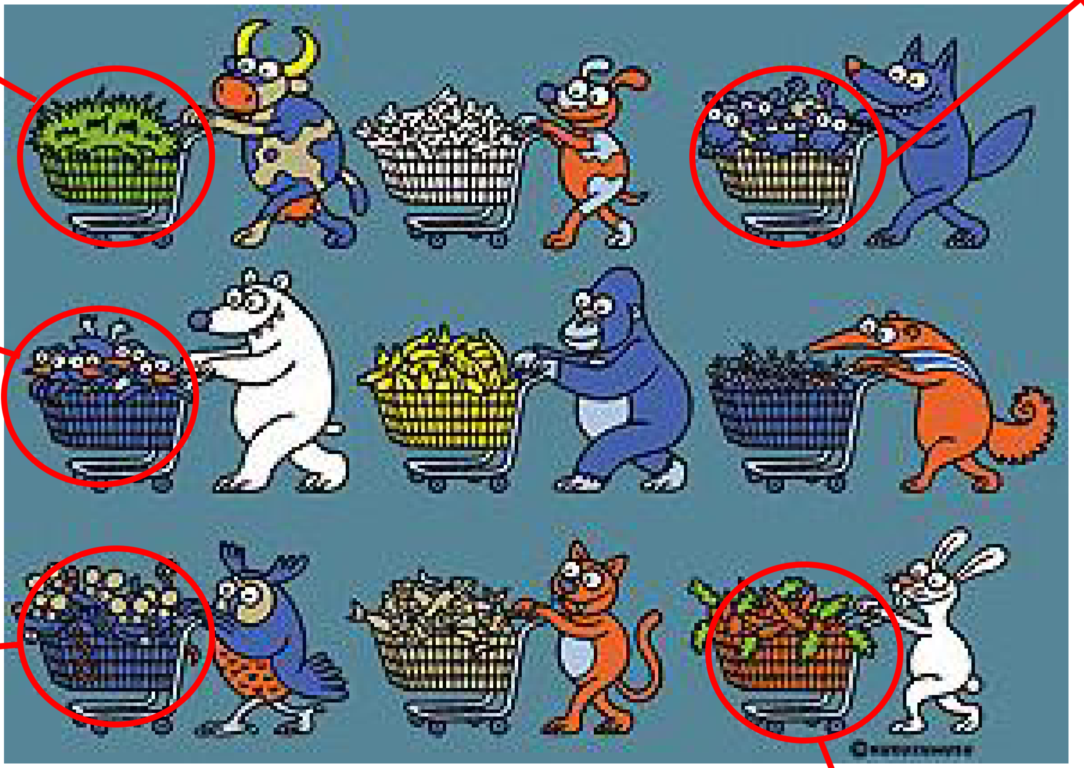
- Status report
- General comments: the scope of the project
- Software review: data model, API, implementation
- Summary of my proposal and conclusions

About the "common" project…

- Is there anything *in common* between what you want?

Reading the XML BLOB containing the LHCb calibration data valid for the event processed

Retrieving the POOL alignment object for the run processed



© KUKUXUMUSU

Storing Atlas high voltages from PVSS into MySQL whenever the values change (every few seconds)

Registering that the CMS detector geometry in a set of Oracle tables is valid for 2008 and 2009

Reading Alice alignment from the ROOT files for the run processed

Commonalities? Project goals and non-goals

- **Project non-goals (experiment-common, not conditionsDB-specific)**
  - Generic C++ access to relational databases (→ POOL project: RAL)
  - Generic relational database deployment and data distribution (→ 3D project)
    - Integration with data distribution infrastructure, however, is a project goal

- **Project goals (experiment-common, conditionsDB-specific)**
  - Common software and tools for non-event time-varying versioned data
    - NB 1: you will need to work a lot to customize the common solution to your needs!
    - NB 2: even this may still be too generic! (see the next slide)

- **Project non-goals (experiment-specific)**
  - Specific data models for calibration/geometry/… (→ experiments)
  - Specific payload format encoding (→ experiments)
    - That is to say: how you use relational databases, RAL or POOL is up to you!
  - Specific time encoding and other conventions (→ experiments)

Scope of the project *(IMO)*

- **Online and offline write and read access patterns are very different**
  - Better to focus on one than to design a software that neither can use
  - No "silver bullet": flexibility vs. performance, especially with databases!
  - *Keep in mind most frequent read access pattern: databases, not data dumps*

- **Proposal: focus on conditions data needed for offline analysis**
  - Just like the BaBar CDB project and the original common project
  - Keep the flexibility of an "online" option too (e.g. data with no versioning), but do not consider this as the main performance target
    - Different time variation patterns (every few seconds), data channel number (no filtering and aggregation) and data sizes (no filtering and averaging)

- ***One experiment may use more than one solution for time-varying data!***
  - BaBar uses two (one for unfiltered online data, one for offline analysis)
    - The "ambient" database (online controls data from EPICS, ~400 GB)
    - The CDB "conditions" database for offline (alignment, calibration, ~32 GB)
  - LHCb plans to use two as well: the PVSS archive and the Conditions DB
  - The "common" solution may not solve all the needs of your experiment

Outline

- Status report
- General comments: the scope of the project
- Software review: data model, API, implementation
- Summary of my proposal and conclusions

Main limitations of current software *(IMO)*

- CondDBOracle (original API)
  - *Data model:* only BLOBs, *no user-defined data payload* (~ à la CondDBMySQL)
    - BLOBs also imply performance overhead if you only need to store POOL *string* tokens
  - *Implementation:* *slow*, need reengineering (bulk inserts, speed up versioning)

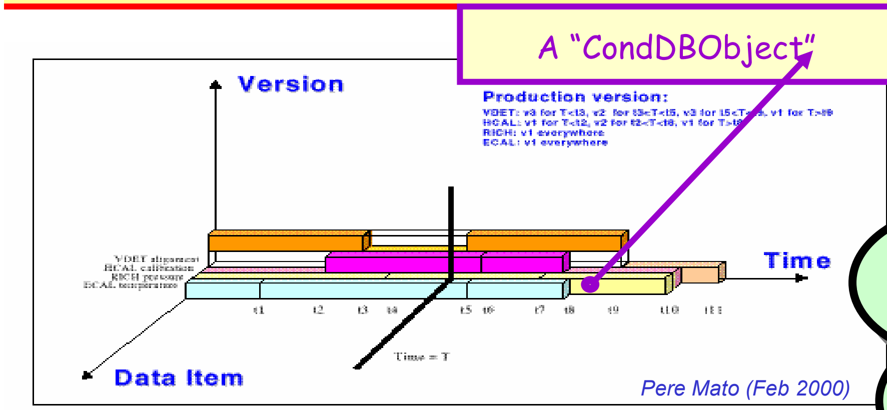
- CondDBMySQL (original API, plus Lisbon API extensions)
  - *Data model & C++ API:* too many ad-hoc solutions, lacks a consistent approach
    - *for instance: BLOBs and relational attributes handled by two different APIs*
    - for instance: versioning/tagging and “channel ID” not provided for all “folder types”
  - *C++ API:* ICondDBTable interface is confusing, many concepts mixed up
    - *schema vs. contents; one vs. many objects*; persistent table vs. transient objects
  - *Duplication of effort:* large overlap with POOL relational access

- CondDBOracle and (vs.) CondDBMySQL:
  - *Differences in data model & C++ API:* *new common developments very difficult*
  - *Implementation:* schemas differ even in tables providing same functionality
    - data copy between CondDBOracle and CondDBMySQL more complex than it could be
  - *Duplication of effort:* code/schema implemented separately in Oracle/MySQL
    - any new features (e.g. partitioning, user tags) would need to be implemented twice
  - *Data distribution:* lack consistent data model and API for partitioning/cloning
  - *Integration:* foresee components to handle referenced data in POOL or tables

Solution *(IMO)*

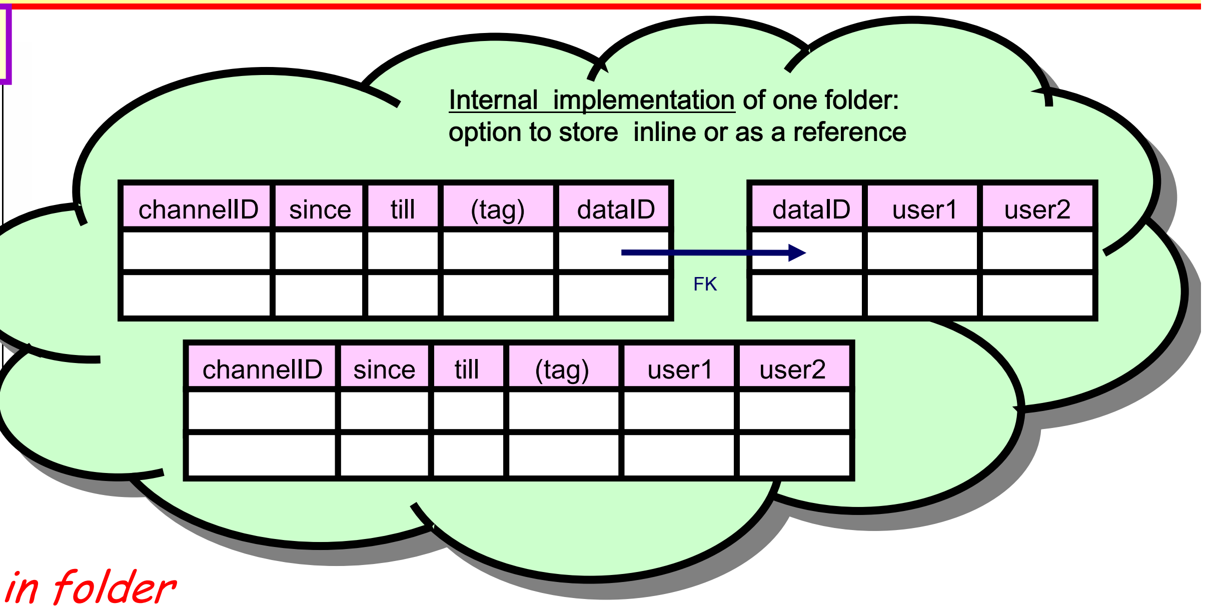
- **Do not simply implement extended data model and API in CondDBOracle**
  - Would not remove internal limitations of CondDBMySQL data model and API
  - Would require significant effort anyway: better spend it on extensive redesign
- **Instead: extend/redesign the data model and the C++ API**
  - Drop the ICondDBTable interface, extend the ICondDBObject interface
    - Replace BLOB data payload by AttributeList (with BLOBs) data payload
    - Single customizable approach instead of many independent ad-hoc solutions
    - Consistent approach to relational data payload and BLOB data payload
    - Clean separation of payload data from payload schema
    - Clean separation of atomic CondDBObject's from their collections/containers
  - Implementation: the POOL Relational Access Layer (RAL) may be appropriate
    - Single implementation for Oracle and MySQL; dependency on RAL, not on all of POOL
  - Decompose required functionalities into several components (eg BLOB encoding)
- **Comments**
  - Take into account need to keep/migrate existing data from Atlas test beam
  - ***Need feedback and commitment from all interested experiments***

Proposal: extend CondDBObject API *(IMO)*



Pere Mato (Feb 2000)

Figure 1 The three axes for identifying uniquely each data item in the condition database



Internal implementation of one folder: option to store inline or as a reference

channelID	since	till	(tag)	dataID

dataID	user1	user2

channelID	since	till	(tag)	user1	user2

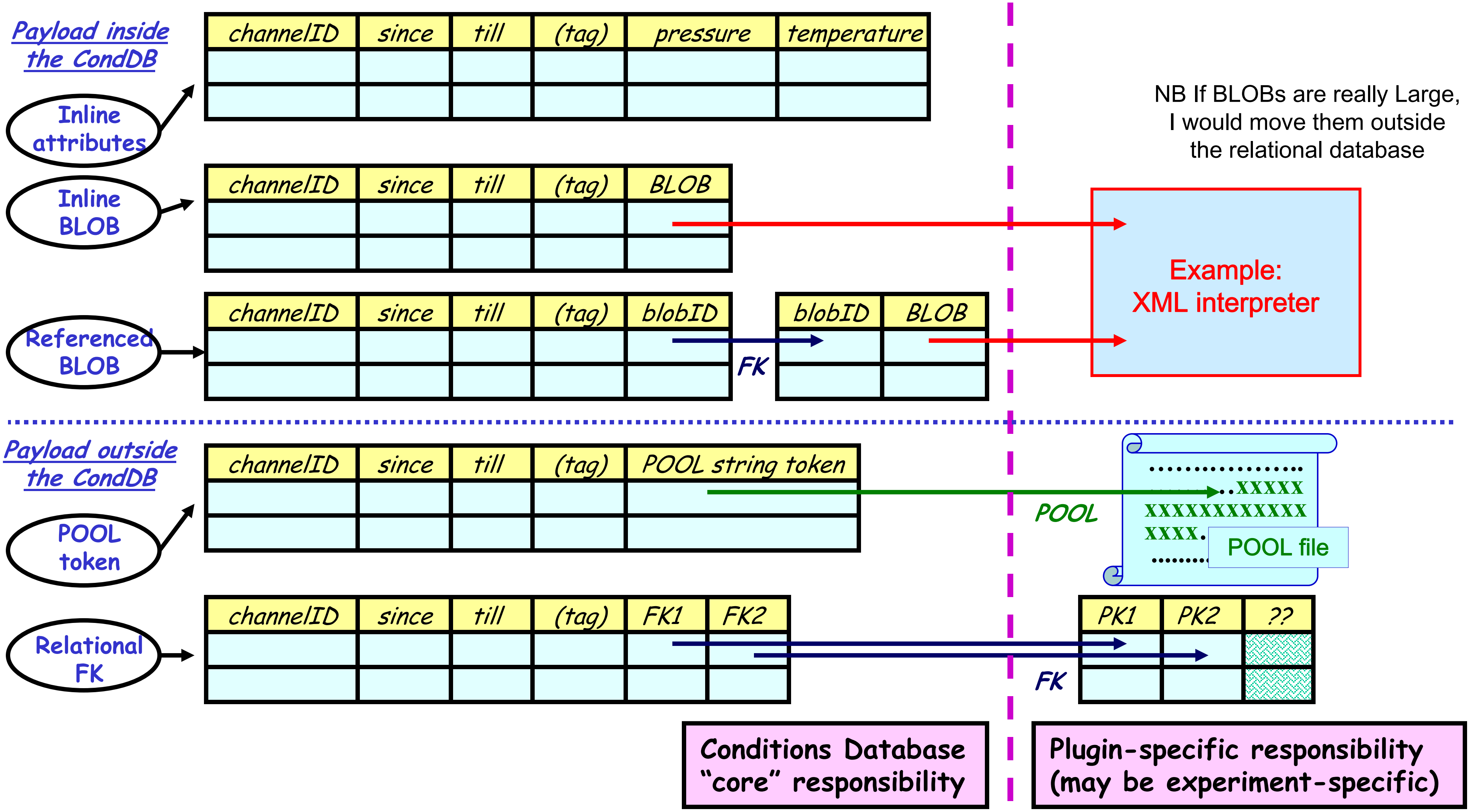
Metadata for one CondDBObject:

1. Data item id: folder name *+ channelID in folder*
   - Options at folder creation: specify channelID schema (AttributeListSpecification); no channelID (only one channel)
2. Interval of validity: [since, till]
3. Version info: *insertion time* (not layer number)
   - Options at folder creation: *no versioning*; versioning with *inline user data*; versioning with referenced user data (stored only once)

Payload for one CondDBObject:

1. User data (AttributeList)
   - Simple C++ types, BLOB; no arrays
   - At folder creation: specify user data AttributeListSpecification
   - Different folders have different schemas; different channels in the same folder have the same schema)

Data payload: typical use cases



Payload inside the CondDB

Inline attributes

channelID	since	till	(tag)	pressure	temperature

Inline BLOB

channelID	since	till	(tag)	BLOB

Referenced BLOB

channelID	since	till	(tag)	blobID

FK

blobID	BLOB

NB If BLOBs are really Large, I would move them outside the relational database

Example: XML interpreter

Payload outside the CondDB

POOL token

channelID	since	till	(tag)	POOL string token

POOL

POOL file

Relational FK

channelID	since	till	(tag)	FK1	FK2

FK

PK1	PK2	??
		▨
		▨

Conditions Database "core" responsibility

Plugin-specific responsibility (may be experiment-specific)

Outline

- Status report
- General comments: the scope of the project
- Software review: data model, API, implementation
- Summary of my proposal and conclusions

Summary of my proposal (1): *main points*

- Keep one main focus for the software: data for offline analysis
  - "Online" option (no versioning, PVSS) too, but not the main performance target
- Extend original API: user-defined data payload (AttributeList)
  - Different payload schemas in different folders (AttributeListSpecification)
- Drop the "ICondDBTable" Lisbon API, extend the ICondDBObject API
  - Keep the same API and metadata model for BLOBs and user-defined data
  - Clean separation of schema vs. data and of one vs. many objects
- Extend original API: foresee partition management and data cloning
  - Many physical partitions may be created within the same logical folder
  - Add special methods to insert "cloned" data (user-specified insertion time)
- Component decomposition: develop components above new extended API
  - Handlers of specific payload types (POOL tokens, relational FKs, BLOBs…)
  - Slicing and copy tools (including deep copy of referenced data, eg POOL)
  - Synchronization manager: keep registered data items in sync with event time
  - Browsing and visualization tools (accepting plugins for user payload)
- Maximise integration with existing LCG solutions
  - Infrastructure: support only SCRAM builds on the official LCG platforms
  - Software: take AttributeList and "generic" relational tables from RAL

Summary of my proposal (2): other points

- **Extend original API: option to switch off versioning ("online")**
  - Cross-check that: "since" of new IOV ≥ "till" of last inserted IOV
- **Extend original API: option to store data payload inline**
  - Implementation detail (transparent to users): storage vs performance overhead
  - Used by default if versioning is switched off ("online" option)
- **Extend original API: tagging extensions (inside each folder)**
  - "Tag HEAD at time XXX" (also: *replace layer number by insertion time*)
  - Option to specify one user tag when a CondDBObject is inserted
- **Extend original API: option to define many "channels" in one folder**
  - The different channels in one folder share the same data payload schema
  - Useful to prevent inflation in the number of folders and tables
- **Extend original API: "hierarchical versioning" in parallel to global tags**
  - Persistently store the parent-to-child association between folder set versions

Summary of my proposal (3): other points

- Implementation technology: use the Relational Abstraction Layer (RAL)
  - Avoid duplication of effort: but direct MySQL implementation is still possible
  - SQL-level data distribution between Oracle and MySQL (same table schema)
- Publish relational schema for read access (but use it at your own risk!)
  - No guarantee that this will not change: keep C++ API stability as main target
  - Direct write access strictly forbidden (unless in data cloning by experts)
  - Translate user requirements (?) for SQL read access into API extensions?
- Discourage the storage of (really large) BLOBs in relational databases
  - Management overhead for DBAs, no user benefit (no SQL queries on BLOBs)
  - If really large BLOBs are foreseen, store them in external files instead?

Work packages (proposal)

- **WP1 – Infrastructure**
  - Build configuration, software integration and release, documentation
- **WP2 – CondDB core software**
  - *Define new extended API and implement it* (e.g. RAL)
  - Generic hierarchical versioning system (if required)
  - Generic data synchronization component
- **WP3 – Experiment integration**
  - *End-user feedback to the project from the experiments*
  - *Experiment-specific guidelines for end-users*
  - *Define and implement realistic tests and examples*
- **WP4 – Specific payload handlers**
  - PVSS "online" manager (write data from PVSS)
  - Handler of POOL string tokens (integration with POOL StorageSvc)
  - Handler of relational FKs (integration with POOL ObjectRelationalAccess)
  - Generic handler for BLOB payload encoding/decoding
- **WP5 – Data management**
  - Tools for data slicing, data copy and data distribution
  - Tools for interactive data browsing

Most urgent: agree on *general direction* for API
-Original BLOBs only
-Lisbon extensions
-My AttributeList proposal
-Other alternatives

Minimal area where ALL the interested experiments should commit manpower

ATLAS manpower?

CMS manpower?

LHCb manpower?

Conclusions

- **This project is about *your data*: your feedback is needed**
  - The goal is to develop the software that *you need* and *you will use*
  - Nobody wants to develop software that would not be used

- **Agreement and active involvement from >1 experiment is needed**
  - Not a "common project" if only Atlas will develop and use the software
  - Target: agreement on commonalities in spite of the many differences

- ***Your* decision is needed now about the direction to take**
  - *Cannot afford to maintain two different APIs at the same time*
  - I made a proposal that tries to take the best from both of them

- **Support your requirements by committing users and developers**

Reserve slides

API review - 1 *(IMO)*

```
class ICondDBObject {
  CondDBKey validSince() const;
  CondDBKey validTill() const;
  …
  void data( string blob ) const;
}
```

🟨 = Original (BLOB) API

🟦 = Extended (ICondDBTable) API

```
class ICondDBTable {
  enum cdb_types {cdbNull=0, cdbBool, cdbInt, …};
  int getNames (vector<string>& names) ;
  int getTypes (vector<cdb_types>& types) ;
  int setName (unsigned n_column, string& name);
  int setType (unsigned n_column, cdb_types type);
  …
  int getRow (unsigned rowNumber, vector<string>& values);
  int [get/set]Cell (unsigned n_column, unsigned n_row, [int/float/..] data);
  …
  int get[Since/Till]Time (unsigned n_row, SimpleTime& time) ;
  int set[Since/Till]Time (vector<SimpleTime>& times) ;
}
```

1. Keep one interface (ICondDBObject) for BLOBs and user fields: use **AttributeList**
   - Replace data() by method returning an AttributeList (intrinsic payload)
   - Support all simple C++ types (int, float, string…) and only them: no user-defined types
     - BLOB type presently not in AttributeList, should be included to support CondDB BLOBs (LHCb)
     - IMO: remove POOL Token from AttributeList, store as strings, interpret in external component
   - Maximise reuse of existing LCG solutions (no dependency on POOL if moved to SEAL)
2. Differentiate schema vs. data: use **AttributeListSpecification**
   - Encapsulated in an ICondDBObjectSpecification?
3. Differentiate one vs many: a table is just a set of objects

API review - 2 *(IMO)*

Create a folder

```
createCondDBFolder(
string fullPath,
string attributes = "",
string description = "",
bool parents = false);
```

```
createCondDBFolder(
string fullPath,
ICondDBTable* table,
string attributes = "",
string description = "",
bool parents = false,
folder_types ftype = STRUCT);
```

Store an object

```
storeCondDBObject(
string folder,
ICondDBObject* CondObject );
```

```
storeCondDBObject(
string folder,
ICondDBTable* table);
```

🟨 = Original (BLOB) API

🟦 = Extended (ICondDBTable) API

Find an object

```
findCondDBObject(
ICondDBObject*& oblock,
string folder,
CondDBKey& point,
string tag = "" );
```

```
findCondDBObject(
ICondDBTable *table,
string folder,
CondDBKey& point,
string id = "“,
string& selection = "",
vector<string>* nullValues = 0,
string tag = "");
```

?

1- Keep one interface (ICondDBObject) for BLOBs and user fields: use AttributeList

2- Differentiate schema vs. data: use ~ AttributeListSpecification at creation time

3- Differentiate one vs. many: get ~ ICondDBObjectSet (or vector<ICondDBObject>) when reading; storing a “table” is just a sequential insertion of many rows?

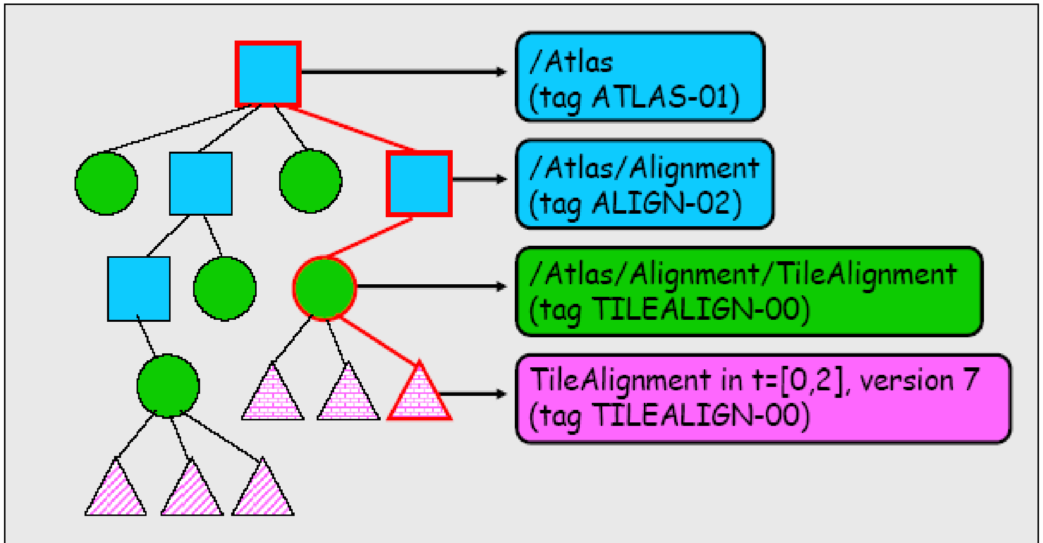
4- IF (?) selection on data content is needed, do not mix it with lookup by time and tag

OK to insert concept of ID and extra folder options (inline/external, online/versioning)

API review - 3 *(IMO)*

- **Define data model and API for partitioning**
  - Useful to keep independent channels in the same folder (same object schema)
- **Replace "layer" by "insertion time" in metadata model?**
  - Inspired again from BaBar (thanks to Igor for many discussions!)
  - Layer/Version "number" not necessarily meaningful
  - Would allow "tag HEAD at time…" extension
  - Would simplify *data cloning* of database slices in validity and insertion times
    - Need dedicated methods anyway for inserting clones
    - Layer structure and original insertion times must be preserved in cloning
- **Channel ID extension?**
  - Useful to keep independent channels in the same folder (same object schema)
- **Tagging extensions (for IOVs within one folder)?**
  - Option to specify one "user tag" at insertion time and keep "user tag HEAD"
- **"Hierarchical" (HVS) tagging extension (for folders within folder sets)?**
  - Useful to keep independent channels in the same folder (same object schema)

"Hierarchical versioning"?



/Atlas
(tag ATLAS-01)

/Atlas/Alignment
(tag ALIGN-02)

/Atlas/Alignment/TileAlignment
(tag TILEALIGN-00)

TileAlignment in t=[0,2], version 7
(tag TILEALIGN-00)

Symbol	HVS	Conditions DB
🟦	Branch Node (Directory)	Folder Set
🟢	Leaf Node (Structure)	Folder
△ △	Structure Element	IOV

Two ways to store the association of "ALIGN-02" and the "TileAlignment in [0,2], version 7" IOV:

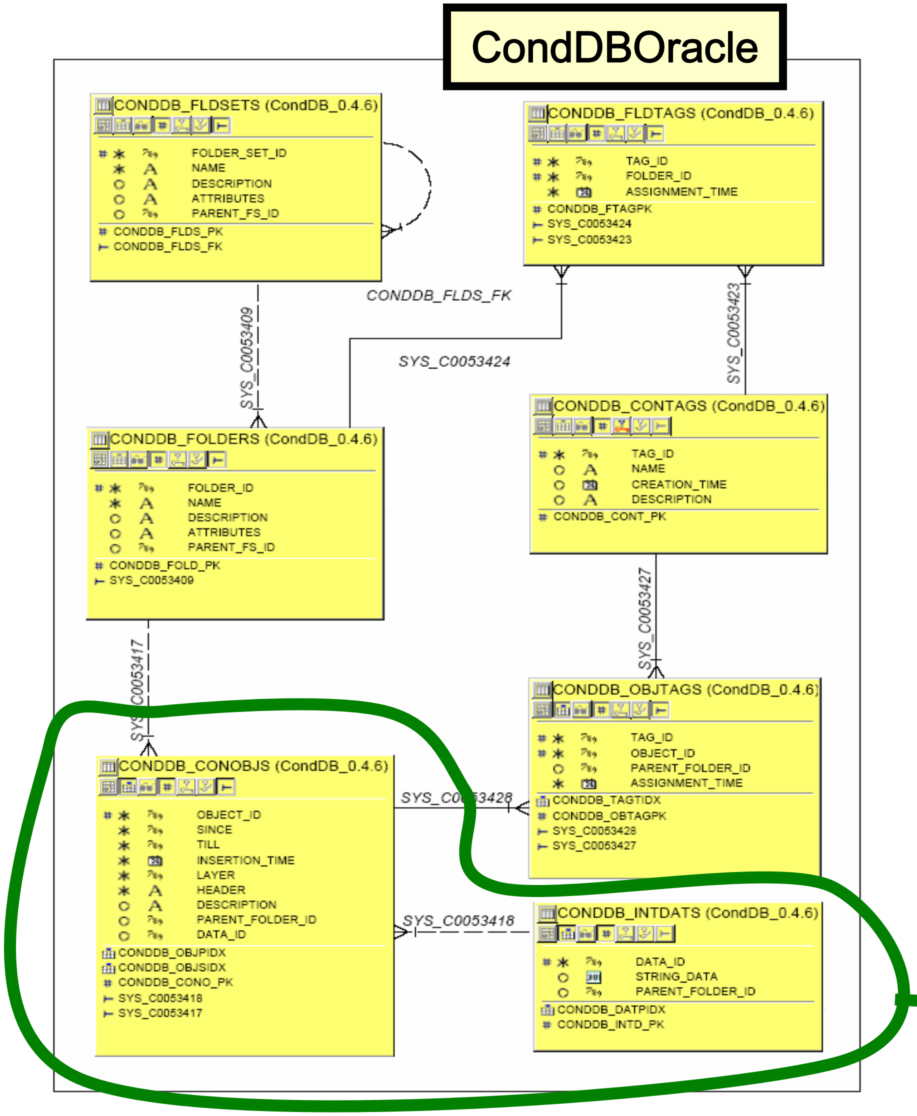
1. Store directly the association between the IOV and the "ALIGN-02" tag; *although "ALIGN-02" is assigned to all IOVs tagged as "TILEALIGN-00", the association is lost*
2. Store the association between the IOV and the local "TILEALIGN-00" tag; then *store the association between the "ALIGN-02" and "TILEALIGN-00" tags*

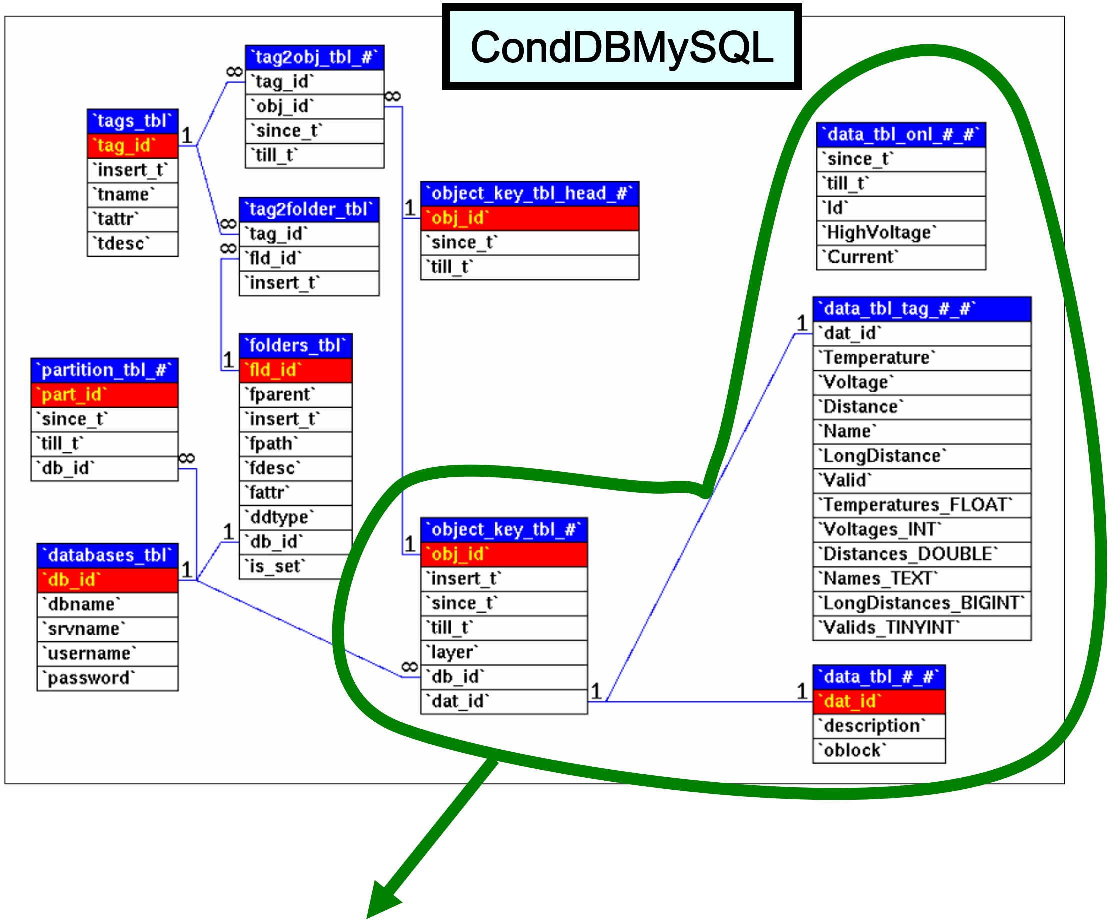
Which way do you prefer?

1. Present Conditions Database tagging (analogous to CVS): "global tags"
2. Hierarchical versioning: "local tags"

- **Scope of possible application to the CondDB: folder set tag management**
  - The association of IOVs to tags within their folder is unchanged

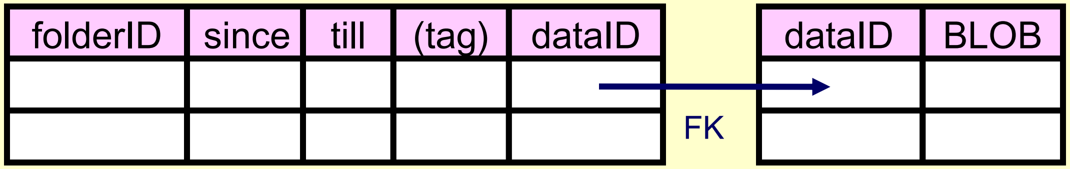
Schema review - 1 *(IMO)*





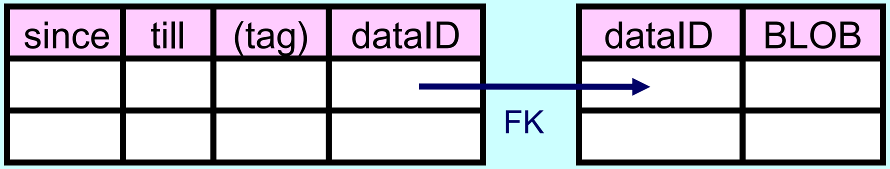
Zoom on *IOV table* and *data payload* table in the next slide

Schema review – 2 *(IMO)*



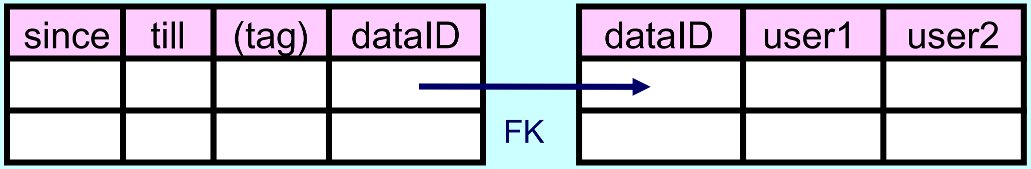
CondDBOracle

1- BLOB API
Versioning.
All folders in one IOV table.
External payload table.

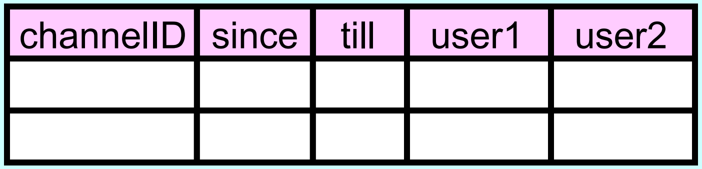


CondDBMySQL

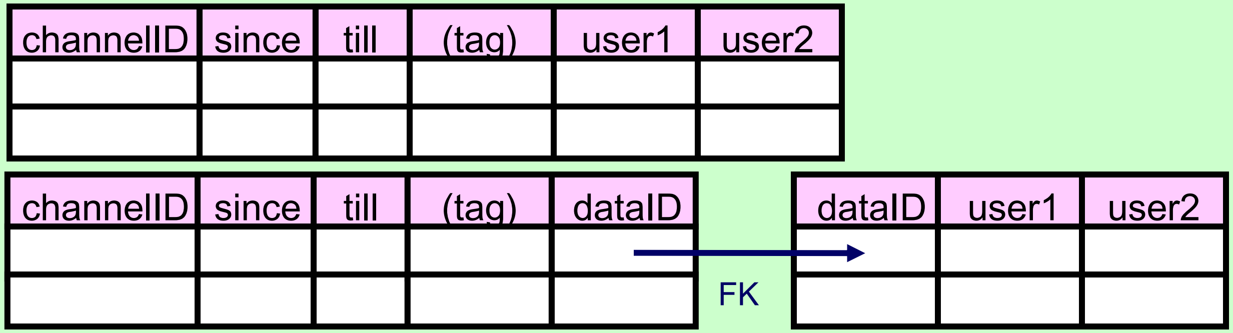
1- BLOB API
Versioning.
One IOV table per folder.
External payload table (BLOBs).



2- Extended API (STRUCTTAG)
Versioning. No ChannelID.
One IOV table per folder.
External payload table (user fields).



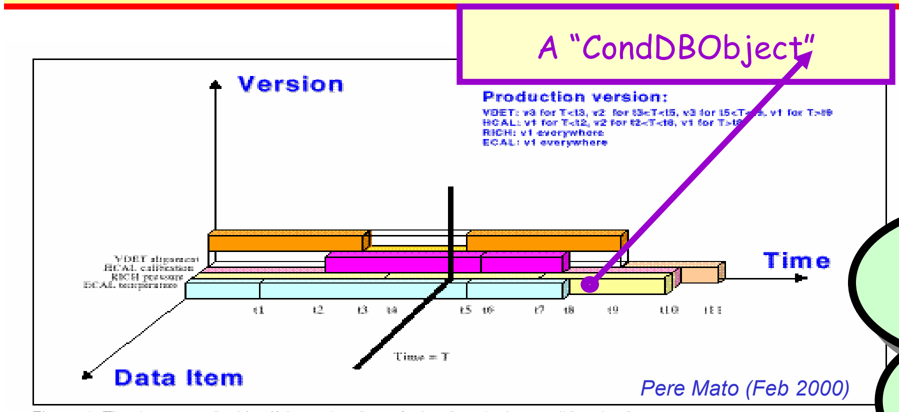
3- Extended API (STRUCTID)
No versioning. ChannelID.
One IOV table per folder.
External payload table (user fields).



New proposed API
One IOV table per folder.
Versioning (unless disabled).
Channel ID (unless disabled).
External payload (unless inlined) – internal implementation detail

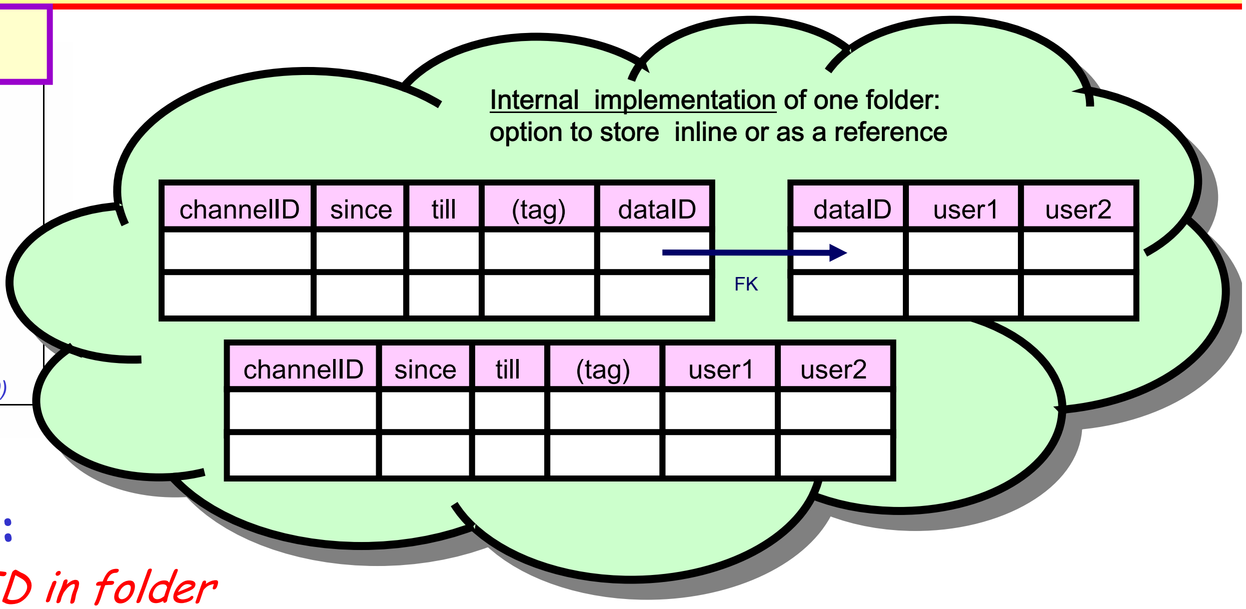
Seen from outside, user interface offers a single consistent view.

Schema review – 3 *(IMO: my proposal)*



Pere Mato (Feb 2000)

Figure 1 The three axes for identifying uniquely each data item in the condition database



Internal implementation of one folder:
option to store inline or as a reference

channelID	since	till	(tag)	dataID

FK

dataID	user1	user2

channelID	since	till	(tag)	user1	user2

Metadata for one CondDBObject:

1. Data item id: folder name *+ channelID in folder*
   - Options at folder creation: specify channelID schema (AttributeListSpecification); no channelID (only one channel)
2. Interval of validity: [since, till]
3. Version info: *insertion time* (not layer number)
   - Options at folder creation: *no versioning*; versioning with *inline user data*; versioning with referenced user data (stored only once)

Payload for one CondDBObject:

1. User data (AttributeList)
   - Simple C++ types, BLOB; no arrays
   - At folder creation: specify user data AttributeListSpecification
   - Different folders have different schemas; different channels in the same folder have the same schema)

Draft requirements from LHCb
(in my understanding; thanks to Beat, Clara and Pere!)

- **CondDB scope: offline analysis**
  - Online writes into CondDB only the pre-filtered data needed for offline
  - PVSS data goes to PVSS archive: CondDB not used for detector debug
- **Technology: anything compatible with distributed analysis**
  - For instance, Oracle at Tier0 and lightweight RDBMS or files at Tier3
  - OK for RAL if: manpower to do it; no performance penalty; few dependencies
- **No API extensions required: original BLOB API would be enough**
  - Priority should be to make that performing and provide tools around it
  - BLOB encoding/decoding (e.g. XML): responsibility of the experiment software
    - But if possible project should provide "standard BLOB" plugins (POOL, AttributeList)
    - Browsing tools should accept plugins to decode BLOBs and display actual data
  - External POOL objects will be referenced by tokens stored as strings/BLOBs
  - *No need for API extensions to store relational data inlined in IOV table*
    - OK for some API changes using AttributeList and option to store strings inline if this helps performance optimization and as long as the BLOB functionality is preserved
  - No need for referencing external relational tables (DetDesc is in XML)
  - Not particularly interested in HVS (tags à la CVS are enough)