FairRoot Database Interface

Denis Bertini (SC-GSI)

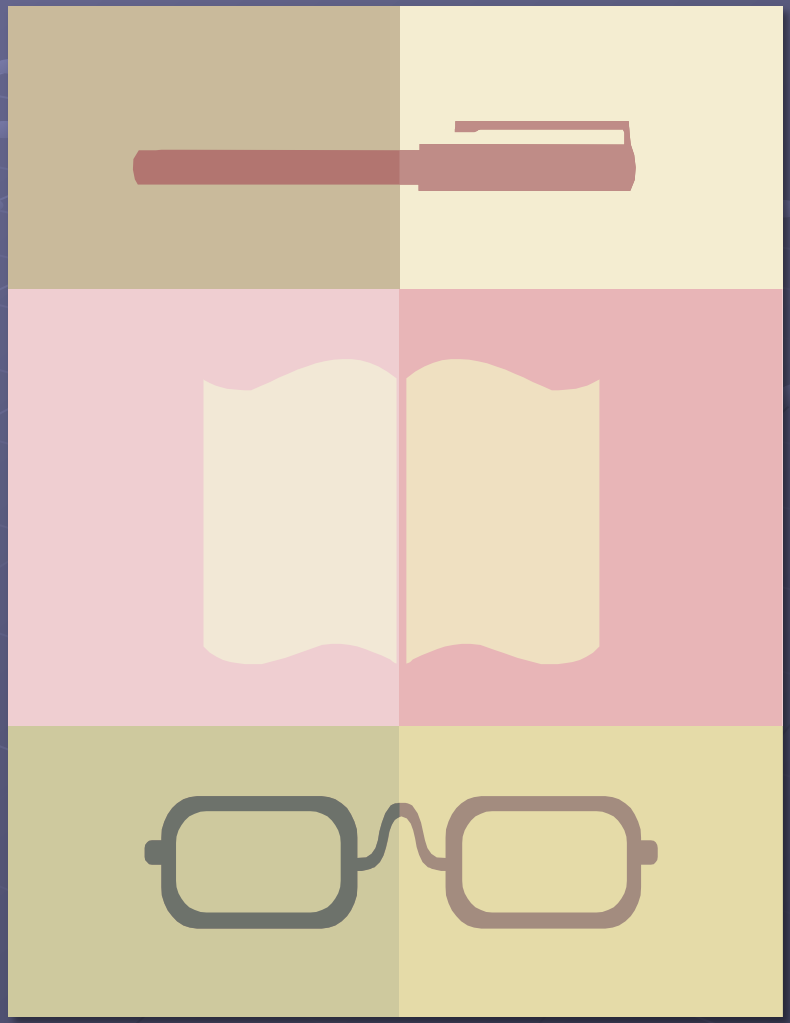
Outline

- A little brainstorming:
  - Why we don't want a database ?
  - Why do we want a database ?
- Where do we start ?
- What do we have ?
  - Database connectivity
  - Version Management
  - Database migration
  - **I/O : Generic Parameter Container**

- **Getting started**

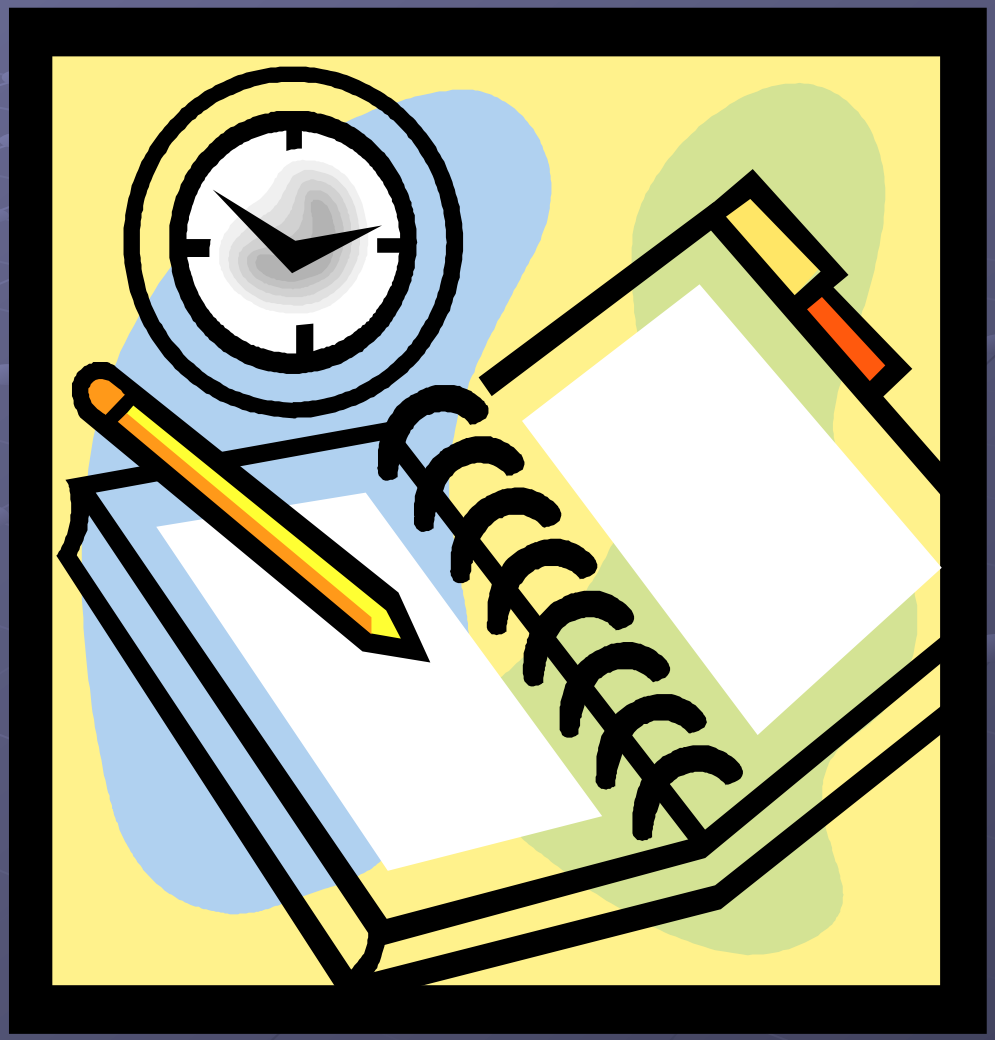
Why we don't want a database?

- **DBs are complex**, difficult and time-consuming to design
- **"garbage in – garbage out".**
  - Who will perform data entry?
  - Learning +Training? On-line help?
  - How data entry will be performed?
- Initial training required for all users
- **Damage to Dbs** will affect everybody
- Extensive **conversion costs** in moving from a DBs to another
- Substantial hardware and software start-up costs



Why do we want a database?

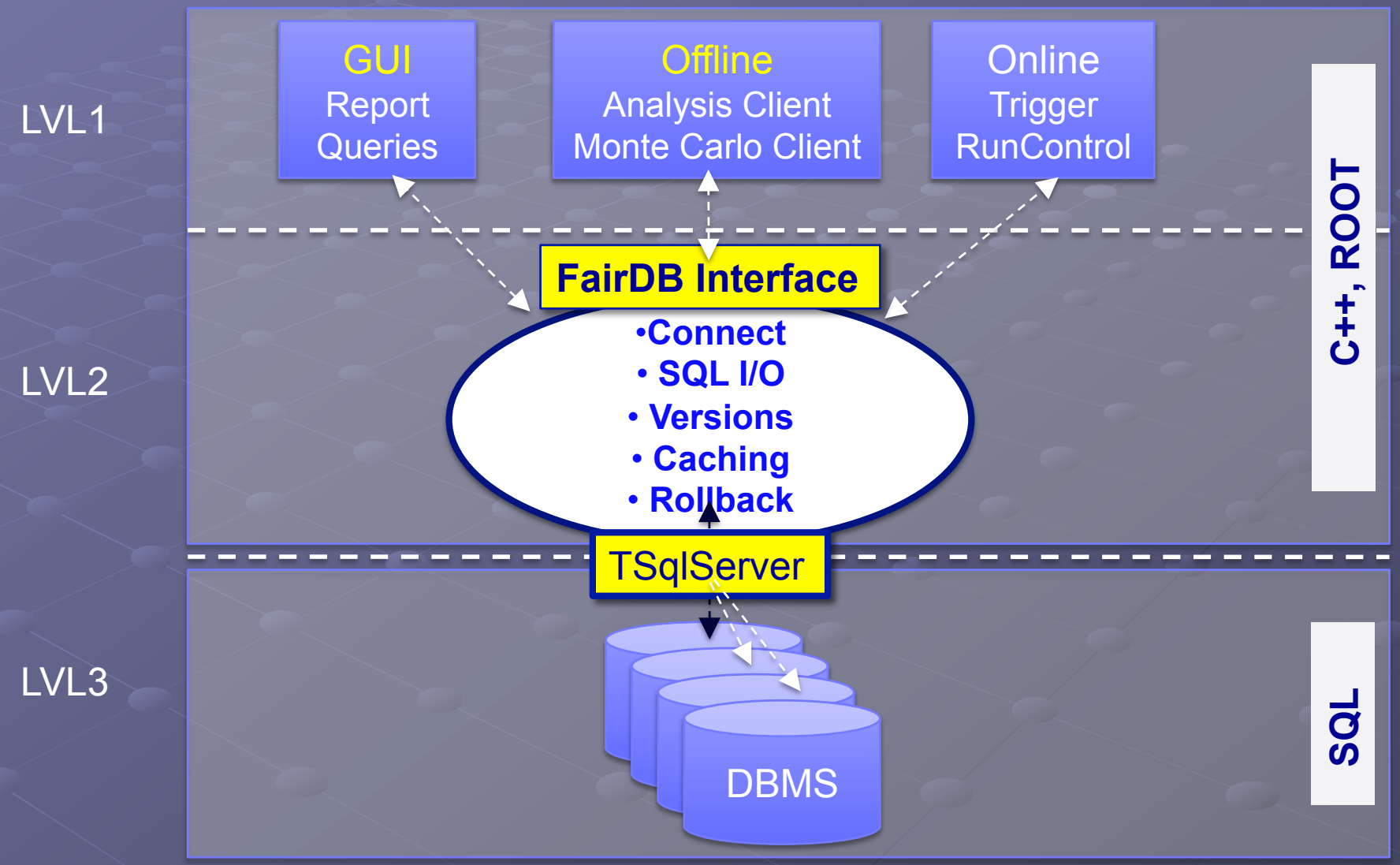
- To keep an "history" of records
  - Online / Offline Parameters
  - Experimental setups
  - etc ..
- To **share these records** between users
- To able users to access the same records concurrently
- To provide efficient access to large amount of data without caring about physical storage format
- To ease communication between users
- To ensure Data consistency & integrity



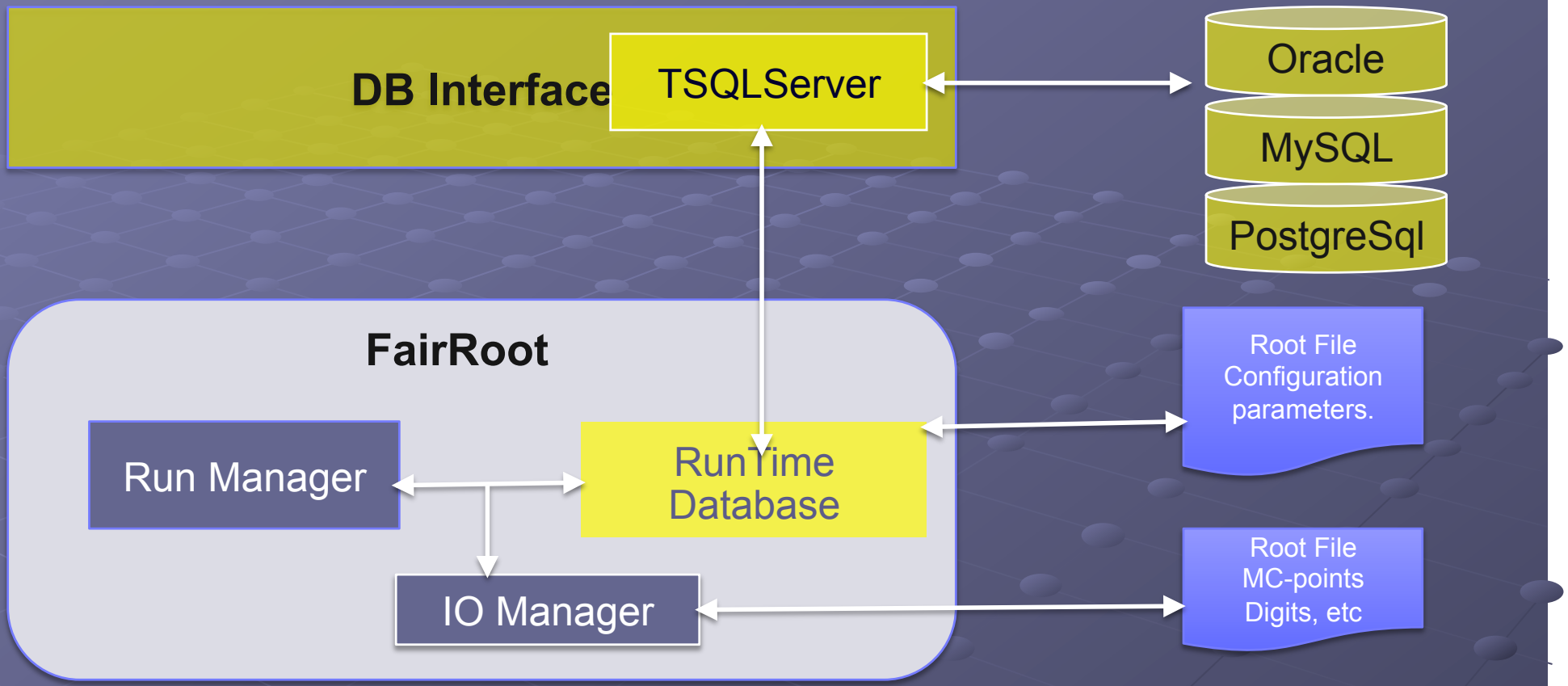
Where do we start?

- **Version Management**
  - Data type: Real and/or MC
  - Detector type
  - Date and Time Range
- **Reduce SQL coding**
  - Simple Predefined Table
  - Only Simple SQL used
  - Ultimately **Generic Container**
- Handle **Write/Read access**
- **Support different DBMS**
  (MySQL, Oracle, PostGres,...)
- Allow **multiple connections** to DB instances at runtime

FairDB interface Concept



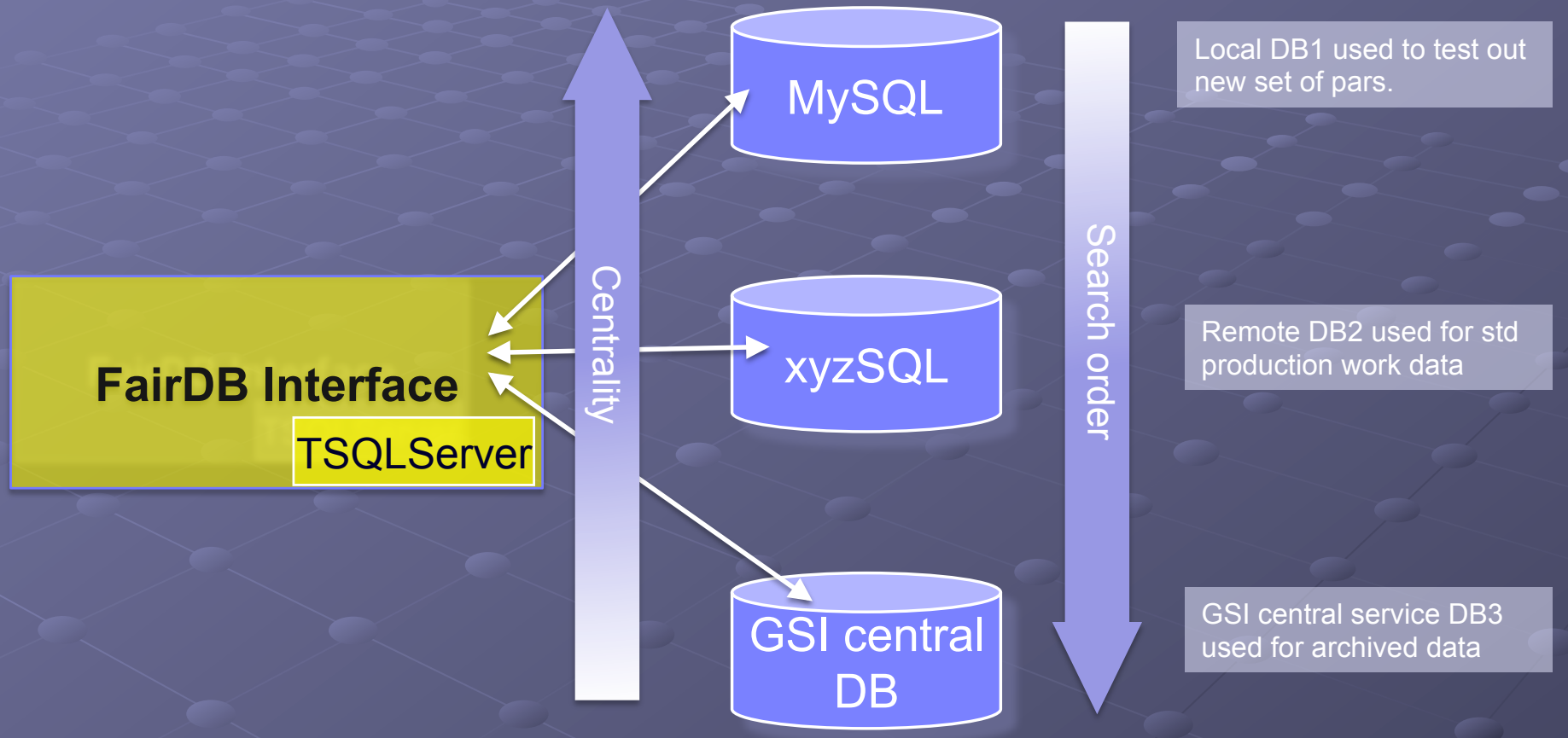
FairDB within FairRoot



Runtime Database

- *FairParRootFileIo*
- FairParAsciiFileIO
- **FairParSQLIo**

FairDB connectivity(1)



Configuring connections

- **FairDB needs a list of DB URLs** , a user name and a password **(JDBC style)**
- Configuration via environment variables of type ENV_DB*
- **Configuration example:**

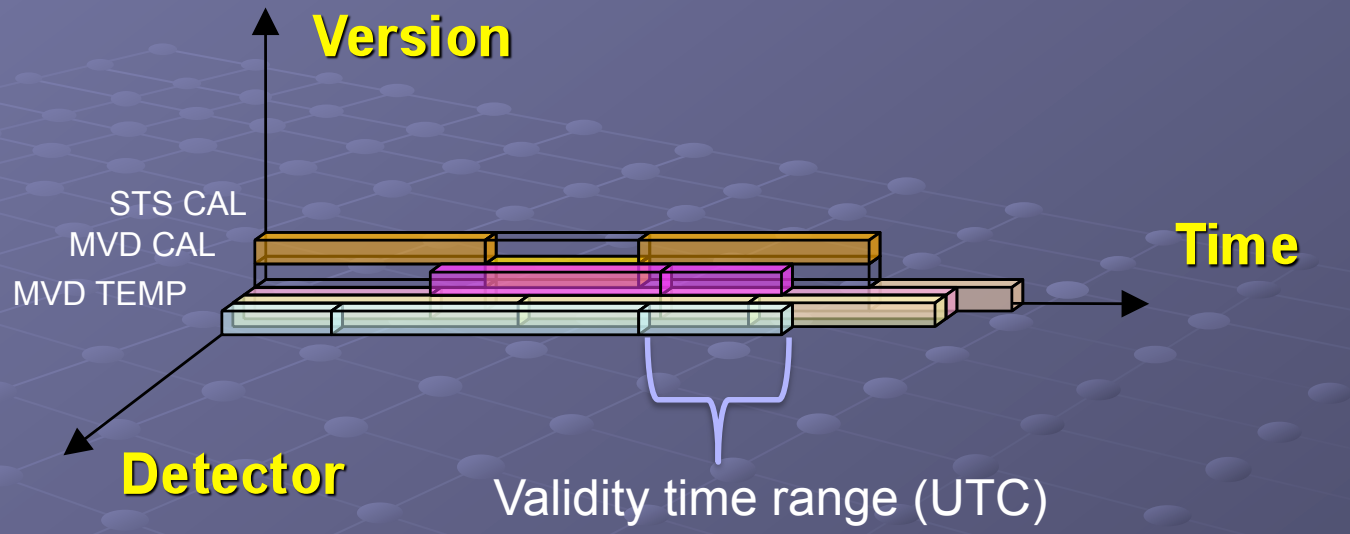
```
// db_config.sh
  export ENV_DB_URL="mysql://demac013.gsi.de/r3b;
                     oracle://bka.oracle.gsi.de/db-test
                     pgsql://demac013.gsi.de:5432/r3b"
  export ENV_DB_USER="denis;scott;denis"
  export ENV_DB_PSWD="passwd_mysql;tiger;passwd_pgsql"
...
}
```

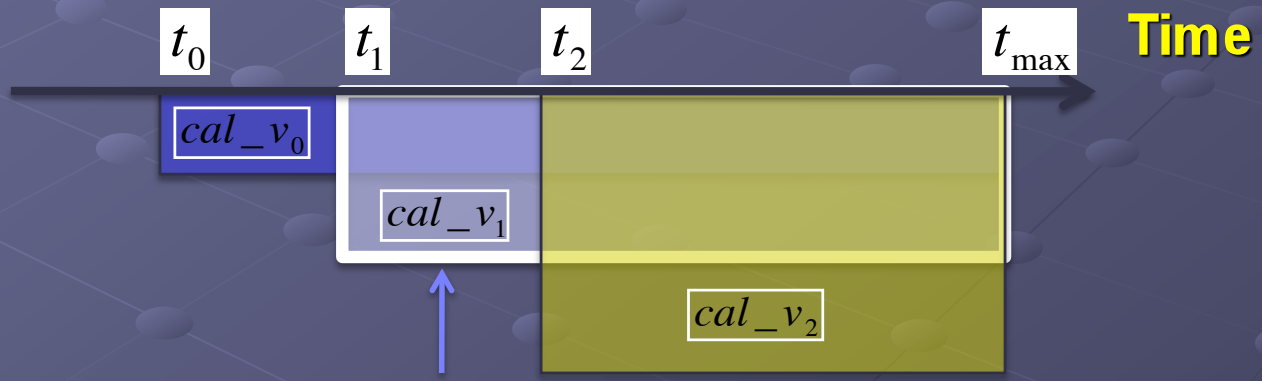
Handling connections

- **Multiple connections:**
  - FairDB can access data for more than one DB
  - DB instances are defined at initialization by a the priority-ordered list of URLs
- **Opening connections**

```
// Create a priority-list of databases

    FairDbMultConnector *fMultConn  = FairDbMultConnector();
    for (Int_t dbId = 0;dbId< fMultConn->GetDnNo(); ++dbId) {
        auto_ptr<FairDbStatement> fStmt(fMultConn->CreateStatement(dbId));

// Execute SQL stmt for dbId
      fStmt->ExecuteUpdate("DROP TABLE IF EXISTS R3BDBDEMODATA1");
…
}
```

Version Management (1)



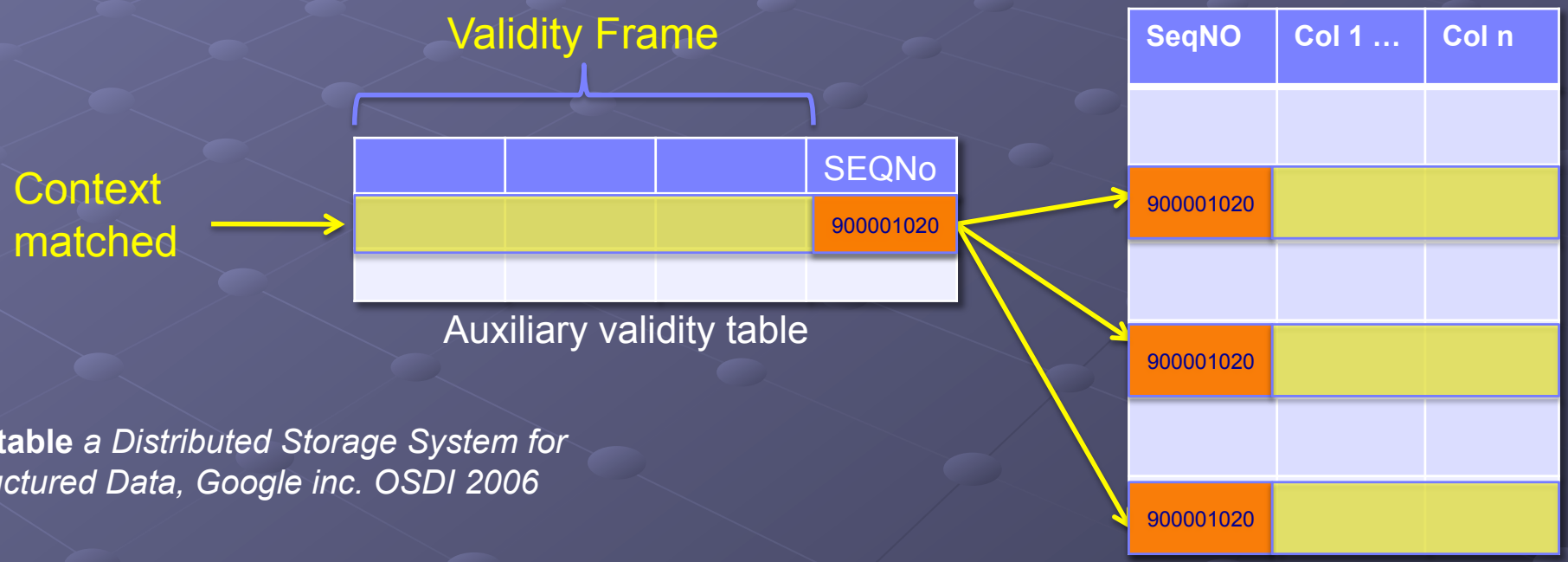


RunID $\Leftrightarrow$ t $\quad t \in [t_1, t_2]$

Version Management (2)

The Query process

1. Context (Timestamp,Detector,Version) is the primary key
2. Context converted to unique SeqNo
3. SeqNo used as keys to access all rows in main table
4. System gives user access of all such rows



Bigtable *a Distributed Storage System for Structured Data, Google inc. OSDI 2006*

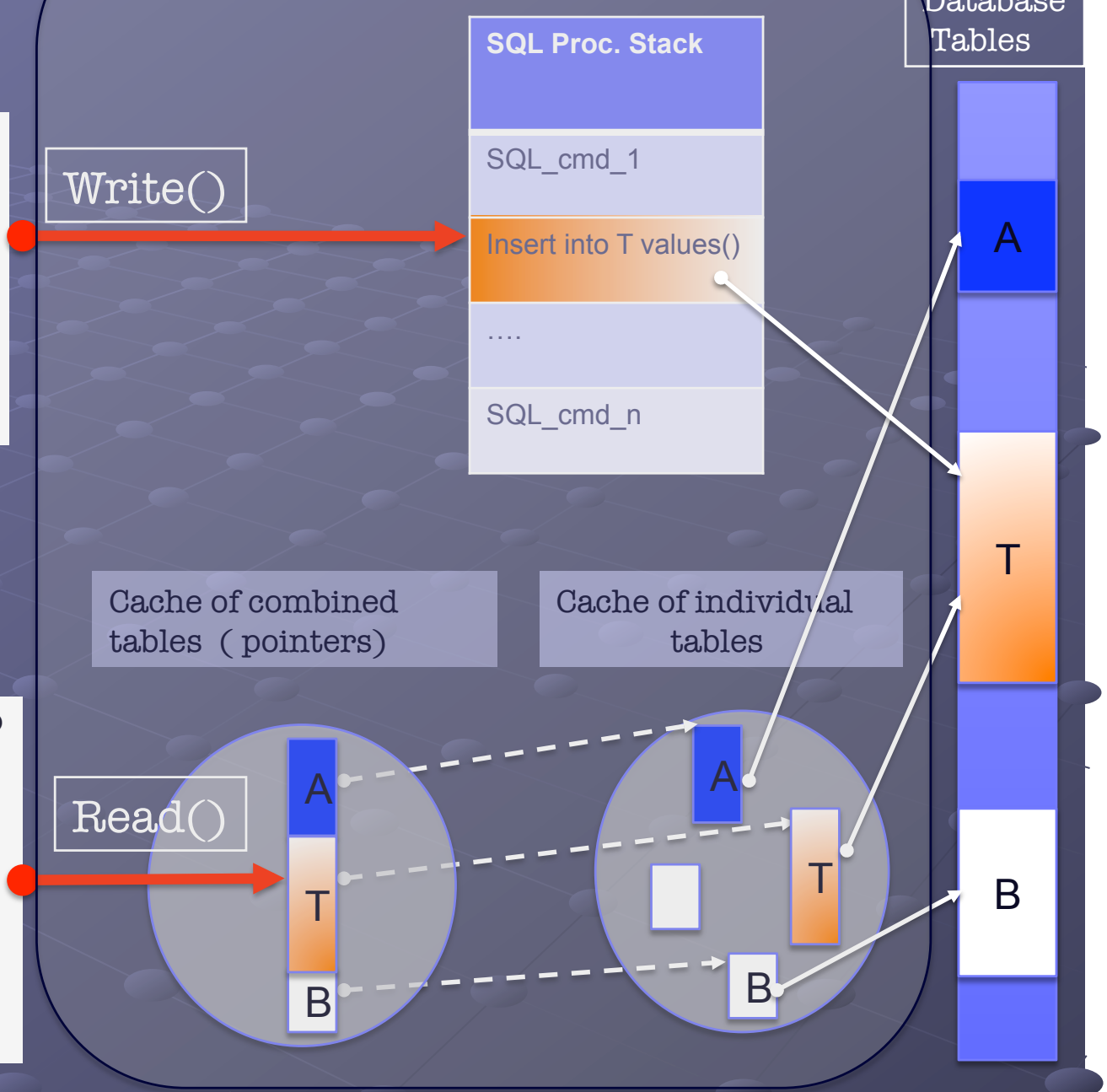
FairDB IO Components

- **Data organisation**
  - **Context**: event date, time, version, detector …
  - **Range**: A context extended to a time window
  - **RowTable**: A single row of a table
  - **Single and/or Composite (**a set of RowTable sharing a Range)
- **I/O Processes**
  - **Write access by Range**
    - Use case:*"do calibrate channel in crate, estimate Range which it remains valid and store in DB"*
  - **Read access by Context**
    - Use case:*"for this event (Context) get calibration csts for every channel in every crate."*
    - Must be efficient (use of Cache that owns query results for reuse, caller just gets const pointers)

FairDB: IO System

```
// Writing by Range (one Comp. at a time)
Int_t Comp_Id;
const ValRange Vr;

FairDbWriter<MyTableRow>  writer(Vr, Comp_Id);

// Fill writer with rows of Agg
MyTableRow row0, row1, ...;

writer<< row0<<row1<< ....;
```

```
// Reading by Context ( multiple Comp at a time)
Int_t Comp_Id;
const ValContext Vc;

//Set up the templated reader
FairDbRes<MyTableRow>  reader(Vc, Comp_Id);

// retrieve all rows
const MyTableRow* row0 = reader.GetRow(0)
const MyTableRow* row1 = reader.GetRow(1);
```



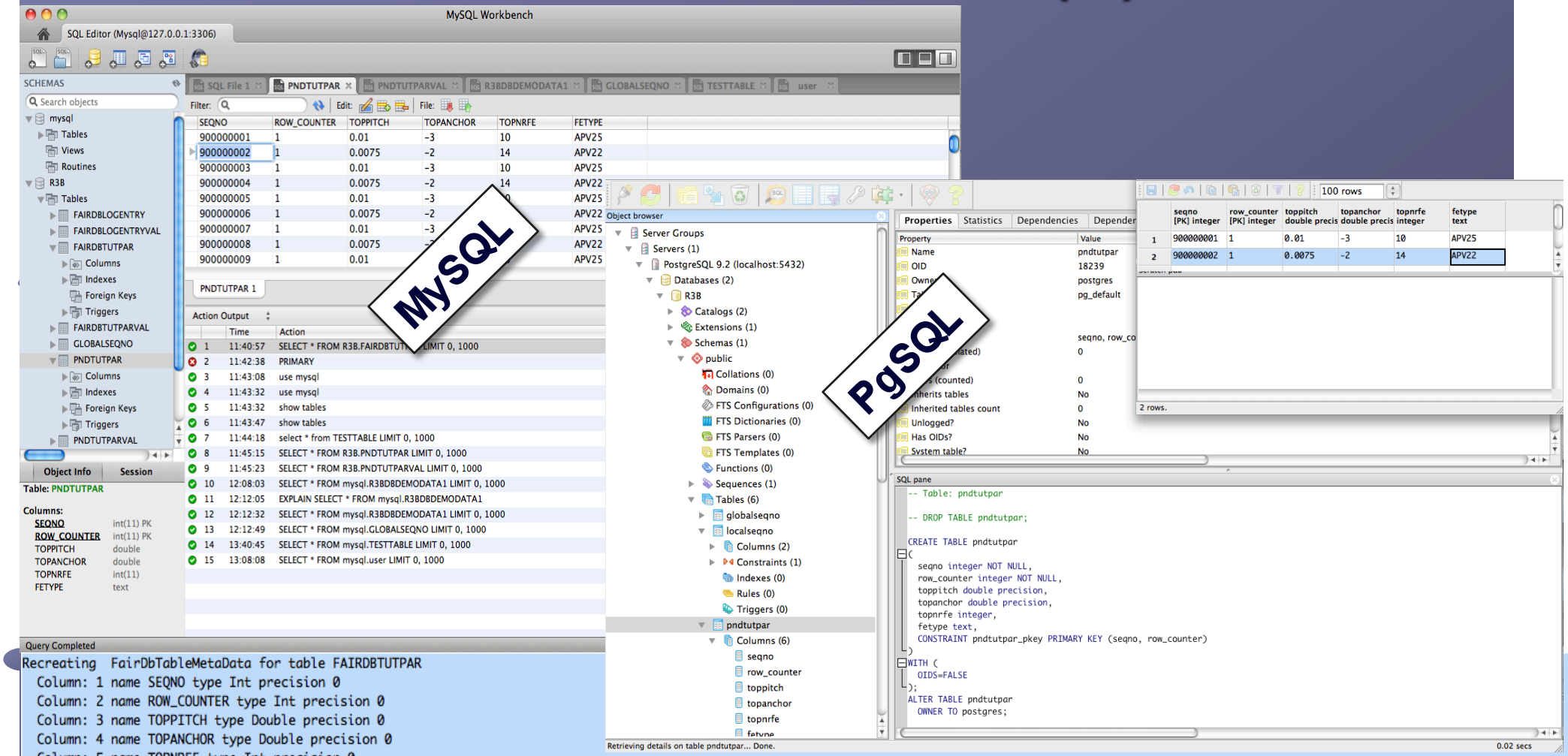
Database Migration

- **Problem** : “S” in SQL does not mean Standard !!!
- **Solution**: keep the SQL code simple, reduce it to minimum
- Vendor-dependent SQL not allowed
- Database Interface supports a **“on-the-fly” SQL conversion**

SQL Translation(1)

- Not meant to be a general purpose SQL translater !!
- Simple translater that FairDB employs which is dialect specific
- Translation (MySQL > Oracle) supported for now
- Translation (MySQL > PostGreSQL) support on-going
  - Set DATE format to be compatible
  - Convert NOW() into SYSDATE
  - Convert WHERE expression of the form A & B to BITAND(A,B) !=0
  - Convert CREATE TABLE: conversion is achieved by creating a FairDBMetaDataTable and then asking it to generate the Oracle / PostGres equivalent

SQL Translation (2)




```
Recreating  FairDbTableMetaData for table FAIRDBTUTPAR
  Column: 1 name SEQNO type Int precision 0
  Column: 2 name ROW_COUNTER type Int precision 0
  Column: 3 name TOPPITCH type Double precision 0
  Column: 4 name TOPANCHOR type Double precision 0
  Column: 5 name TOPNRFE type Int precision 0
  Column: 6 name FETYPE type Char precision 0
-I- FairDbStatement::TranslateSQL  sql: create table FAIRDBTUTPAR( SEQNO         INT NOT NULL,  ROW_COUNTER    INT NOT NULL,  TOPPITCH       DOUBLE,  TOPANCHOR      DOUBLE,  TOPNRFE        INT,  FETYPE         TEXT
ey(SEQNO,ROW_COUNTER))
translates to 1 statements:-
-I- FairDbStatement::TranslateSQL    create table FAIRDBTUTPAR(SEQNO INT not null, ROW_COUNTER INT not null, TOPPITCH DOUBLE PRECISION, TOPANCHOR DOUBLE PRECISION, TOPNRFE INT, FETYPE VARCHAR(65535), primary key (SE
TER))
-I- FairDbStatement::ExecuteUpdate SQL:R3B:create table FAIRDBTUTPAR(SEQNO INT not null, ROW_COUNTER INT not null, TOPPITCH DOUBLE PRECISION, TOPANCHOR DOUBLE PRECISION, TOPNRFE INT, FETYPE VARCHAR(65535), primary k
W_COUNTER))
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "fairdbtutpar_pkey" for table "fairdbtutpar"
-I- FairDbResultPtr:: GetTableProxy tablename: FAIRDBTUTPAR
-I- FairDbTableProxyRegistry create FairDbTableProxy 0x7fff5fbdb250 proxyname# FAIRDBTUTPAR::FairDbTutPar
-I- FairDbTableMetaData:: create for table # FAIRDBTUTPAR
```

I/O using generic container

- Ultimately hides the SQL coding to the user, no DB expertise needed
- Allows simple type data member and also complex ones
- Stores as Table and/or Byte Array **(complex types)**
- (Schema evolution: uses streamer info , ROOT version)

Data types:
Int_t[], Float_t[], Double_t[], Char_t[],
Text_t[], UChar_t[], **class type**

stored as byte array (Binary String)
number of values (single value or array)

any class derived from TObject decoded in the analysis interface by ROOT streamer

Getting Started

- Need new external packages April 2013
  - Root with DB plugins
  - Installation: svn or cvmfs
- Use the trunk version for FairRoot fairbase
  - Edit and change the externals links to point to “trunk”

/base: https://subversion.gsi.de/fairroot/fairbase/trunk/base

- Add the new example directory containing the FairDB tutorials

/example: https://subversion.gsi.de/fairroot/fairbase/trunk/example

- Adapt the FairDB config script “dbconfig.sh” _

SQL aware parameter class

```
class FairDbTutPar : public FairParGenericSet
                    {
 public : // ….
```

```
private:
   // Strip Parameters
 Double_t fTopPitch;     // Strip pitch on top wafer side
 Double_t fTopAnchor;    // Anchor point of top strip#0
 Int_t     fTopNrFE;      // NFE attached to top wafer side
string    fFeType;       // Frontend type name
…
```

```
// I/O  member functions
                virtual void Fill(FairDbResultSet& rs,
                            const FairDbValidityRec* vrec);
                virtual void Store(FairDbOutRowStream& ors,
                             const FairDbValidityRec* vrec) const;
                virtual void Fill(UInt_t rid);
                virtual void Store(UInt_t rid);

                // Validity frame definition
                virtual ValContext GetContextDTF(UInt_t rid) {
                  return ValContext(Detector::kGfi,
                                    SimFlag::kData,
                                    ValTimeStamp(rid));
                }
…
```

sql_write_params.C

```
FairRuntimeDb* db = FairRuntimeDb::instance();
 cout << "-I- FairRuntimeDb created ----> " << db << endl;

 // Create in memory the relevant container
 FairDbTutPar* p1 = (FairDbTutPar*)(db->getContainer("TUTParDefault"));
 FairDbTutPar* p2 = (FairDbTutPar*)(db->getContainer("TUTParAlternative"));

 // Set the Ascii IO as first input
 FairParAsciiFileIo* inp1 = new FairParAsciiFileIo();

 TString work = getenv("VMCWORKDIR");
 TString filename = work + "/example/Tutorial5/macros/ascii-example.par";
 inp1->open(filename.Data(),"in");
 db->setFirstInput(inp1);

 // Set the SQL based IO as second input
FairParTSQLIo* inp2 = new FairParTSQLIo();
inp2->open();
db->setSecondInput(inp2);

 // <INIT> containers from Ascii input
 // with assigned  RunId

db->initContainers(runId);
 p1->Print();
p2->Print();

 // <WRITE> back containers to the user-defined
 // Database using the SQL based IO of the
 // second input.

 db->setOutput(inp2);
 db->writeContainers();
```

sql_read_params.C

```
// Create a Runtime Database singleton.
                FairRuntimeDb* db = FairRuntimeDb::instance();

                // Set the SQL IO as first input
                FairParTSQLIo* inp = new FairParTSQLIo();
                inp->open();
                db->setFirstInput(inp);

                // Create the container via the factory if not already created
                FairDbTutPar* p1 = (FairDbTutPar*)(db->getContainer("TUTParDefault"));
                FairDbTutPar* p2 = (FairDbTutPar*)(db->getContainer("TUTParAlternative"));


                // Create a dummy runID using date in UTC from which
                // corresponding  parameters will be initialised

                ValTimeStamp tStamp(2013,04,08,08,17,00);
                UInt_t runId = tStamp.GetSec();
                cout << "-I- looking for parameters at runID# " << runId << endl;
                cout << "-I- corresponding  time in runID (UTC) " << tStamp.AsString("c") << endl;

                // Use the generated RunID to initialised the parameter
                // using the SQL-based IO input
                db->initContainers(runId);


                pp1->Print();
                pp2->Print();
```

Table Data Model

Data Objects Table rows:

```
mysql> select * from FAIRDBTUTPAR;
+-----------+-------------+----------+-----------+---------+--------+
| SEQNO     | ROW_COUNTER | TOPPITCH | TOPANCHOR | TOPNRFE | FETYPE |
+-----------+-------------+----------+-----------+---------+--------+
| 900000001 |           1 |     0.01 |        -3 |      10 | APV25  |
| 900000002 |           1 |   0.0075 |        -2 |      14 | APV22  |
| 900000003 |           1 |     0.01 |        -3 |      10 | APV25  |
| 900000004 |           1 |   0.0075 |        -2 |      14 | APV22  |
+-----------+-------------+----------+-----------+---------+--------+
4 rows in set (0.00 sec)
```

Auxiliary Validity Table

```
mysql> select * from FAIRDBTUTPARVAL;
+-----------+---------------------+---------------------+-------------+---------+---------+-------------+---------------------+---------------------+
| SEQNO     | TIMESTART           | TIMEEND             | DETECTORMASK | SIMMASK | VERSION | AGGREGATENO | CREATIONDATE        | INSERTDATE          |
+-----------+---------------------+---------------------+-------------+---------+---------+-------------+---------------------+---------------------+
| 900000001 | 2013-04-08 08:16:05 | 2038-01-19 03:14:07 |           8 |       1 |       0 |          -1 | 2013-04-08 08:16:05 | 2013-04-08 08:16:05 |
| 900000002 | 2013-04-08 08:16:05 | 2038-01-19 03:14:07 |           8 |       1 |       1 |          -1 | 2013-04-08 08:16:05 | 2013-04-08 08:16:05 |
| 900000003 | 2013-04-08 08:19:59 | 2038-01-19 03:14:07 |           8 |       1 |       0 |          -1 | 2013-04-08 08:17:05 | 2013-04-08 08:19:59 |
| 900000004 | 2013-04-08 08:19:59 | 2038-01-19 03:14:07 |           8 |       1 |       1 |          -1 | 2013-04-08 08:17:05 | 2013-04-08 08:19:59 |
+-----------+---------------------+---------------------+-------------+---------+---------+-------------+---------------------+---------------------+
```

Conclusions

- **A new Database Interface is available within FairRoot**
  - Fairbase (trunk) + external packages (April 2013)
  - /fairbase/trunk/example/tutorial5
- It does **not fix the DB technologie** FAIR experiments
  - (MySQL, Oracle, PostGres, etc…)
- FairDB is a natural **testing / benchmark framework**
- **Ongoing work:**
  - Including / testing PostGreSQL
  - Consolidating / extending SQL conversion
  - Extending Generic Parameter Container
  - Exceptions & Logging Mechanism
  - Realistic tests