Database Architecture for the European Emulsion Scanning System

E. Barbuto, C. Bozza, G. Grella, G. Romano, C. Sirignano – University of Salerno and INFN.

Abstract

The database system sketched in the following pages is intended to be the backbone in the computing infrastructure for the European Emulsion Scanning System for OPERA. The overall structure and the details of data organization are discussed. Performance and security issues are dealt with. The note ends with the source SQL scripts needed to implement the proposed structure.

1 Overview

The European Emulsion Scanning System (**EESS** from now on) will be used in several laboratories spread over all Europe. In order to have uniform efficiencies and performances and consistent data handling, they will act as a single sub-detector. This is really a unique situation in High Energy Physics. Moreover, OPERA needs a quasi-online scanning. A conventional file-based data storage system would require very large administrative and programming efforts, and much of the developers' work should then be devoted to ensure data safety and consistency. A database system is an ideal choice in such a case. ORACLE DB Server, a very robust and scalable commercial solution, has now been made freely available for CERN experiments. Though general, the DB architecture proposed in the following pages has been practically developed and tested on ORACLE 9iDS, and some details of the implementation depend on the choice of the database system.

In order to clarify the boundaries of the subject, we very briefly recall the organizational structure of the European Emulsion Scanning Group.

From the OPERA detector, the triggered bricks are sent to the Scanning Station for CS

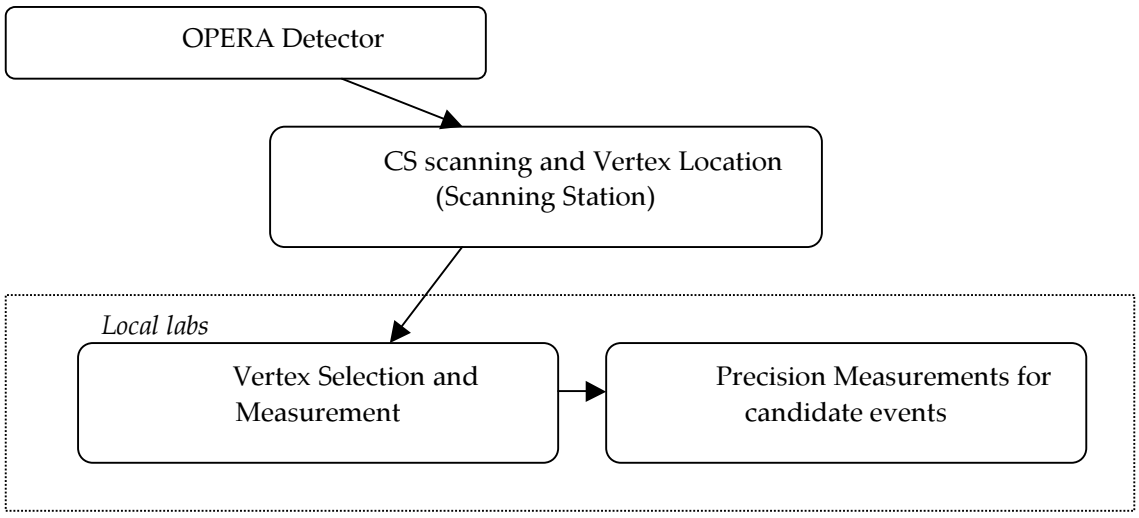

Fig. 1. Scanning organization and brick flow

scanning and vertex location. After the vertex is found, the bricks are sent to local laboratories for vertex selection and precision measurements. The brick sharing should be automatic, since it must follow the interaction triggers in the detector. Likewise, every laboratory must make its results immediately available to the whole European Scanning Group (and possibly to the OPERA Collaboration) so that relevant decisions can be taken (e.g. extracting another brick, require further studies on some event, etc.). Since OPERA emulsions have a continuous refreshing feature due to controlled fading, all decisions and scanning tasks must occur as quickly as interactions are generated in the detector. Every laboratory must have immediate access to all the relevant information about the history of every event, ranging from the electronic sub-detector predictions to the outcome of previous scanning tasks (outcome of CS scanning, of Vertex Location, of Vertex Selection, etc.). The scanning power is also distributed over all Europe: we must take into account the possibility that bricks entering one laboratory be processed there up to some stage, and then go to some other laboratory for detailed study.

Data consistency, safety and security are three important points. Many programs and many persons will write data to the DB. Indeed, most data will be written and almost never read (non-interesting events). Without good control over data insertion and update, mistakes would fatally occur. Replication, automatic backup and access restrictions will help preserving existing data. A good DB design, however, must protect from the insertion of wrong data. As the reader will find in the following, much of the DB schema we envisaged is dedicated to ensure data correctness.

Last, but not least, ORACLE 9iDS is easy to administer remotely. This is very important, because increasing the number of administrators too much would lead to spoiling human resources that would otherwise be dedicated to Physics, which is our ultimate goal.

2 Design guidelines

The schema of the European Emulsion DB (**EEDB** from now on) has been designed to ensure performance as well as tight control over data consistency.

1) Every entry in almost every table has its own ID number.
2) Logical relationships have been made explicit in the DB schema.
3) An effort has been made to establish relationships through single indices, even at the cost of introducing alternate keys in the tables. In general, these alternate keys are all generated automatically by the DB Management System (**DBMS**), so that uniqueness is ensured. In the case of ORACLE 9iDS, *number sequences* can be defined by the designer and managed by the DBMS, to provide a unique ID number for each entry in a table.
4) In order to save space, the number of entries at the end of the experiment has been guessed for each table, and the data size has been specified accordingly.
5) Triggers and / or constraints have been set wherever *a priori* consistency checks can be made on data (e.g., since every brick has 56 plates, it is impossible to write data about plate #74).
6) Data redundancy has been avoided when possible because it is a source of mistakes. However, in order to retrieve some information it is sometimes necessary to execute queries with multiple join operations, which slow down the process of data retrieval; in some cases, and for very frequent queries, some tables have been denormalized (i.e. they contain some columns with replicas of data

stored in other tables). This is a common practice, but it depends on the designer's judgement how far to follow it, and a detailed optimisation needs further study.

7) The EESS should in principle be completely configurable and administrable through the DB. Machines and users must be registered and known to the DB. For example, brick transport to some location must be performed by couriers, trucks or whatever, and the DB system must be aware that, at certain time, the brick is being transported.

8) Access restrictions must be applied according to the criticality level of each table. Only administrators will be able to insert / update / remove scanning and data processing machines and user permissions. People involved in the scanning tasks must be able to write data to scanning tables, but not to final event reconstruction tables (from which they can of course read). Likewise, people involved in offline tasks should have complete access to event reconstruction tables, but they should not be able to modify the primary "raw" data. Users should not be able to read each other's passwords.

9) Other permissions cannot be implemented as access restrictions on DB schema items, but they are important as well. An example will make things clearer: user "*John Smith*", who is a member of scanning group in the site of "*Whoknowswhere*", can write to primary scanning data tables; however, he should be prevented from scheduling a scanning batch on the machines of the "*Fairyplace*" site. In this case, we are not specifying that he can / cannot write data to the DB; we are stating that privileges to do something somewhere must also be recorded in some way; instead of using local configuration files, it is better to have DB tables storing these data as well.

Only every-day use will help to fine-tune the DB structure. It must be clear that this is still a preliminary (but working) design.

3 DB schema

Figure 2 shows a graphical representation of the DB schema obtained through QDesigner (a commercial tool for DB design). This is a preliminary version and is subject to changes. The rectangular regions grouping tables together have been put just to help the reader's comprehension. The green rectangles stand for DB tables, and the arrows stand for DB relationships. The naming convention used is defined by the following rules that:

1) Table names begin with "TB";
2) Index names begin with "IX"; in particular, alternate keys are always called IX_tablename_ID;
3) Identity fields (fields that uniquely identify the record in that table) are always named "ID";
4) Fields referring to related fields in other tables (foreign keys) are named "ID_xxxx" where "xxxx" is the name of the related table;
5) Foreign key relationships are named "FK_primarytable_foreigntable", unless the name results too long;
6) Primary keys must be named "PK_tablename".
7) Unique number sequences must be named "SEQ_xxxx" where "xxxx" is the name of the associated table.

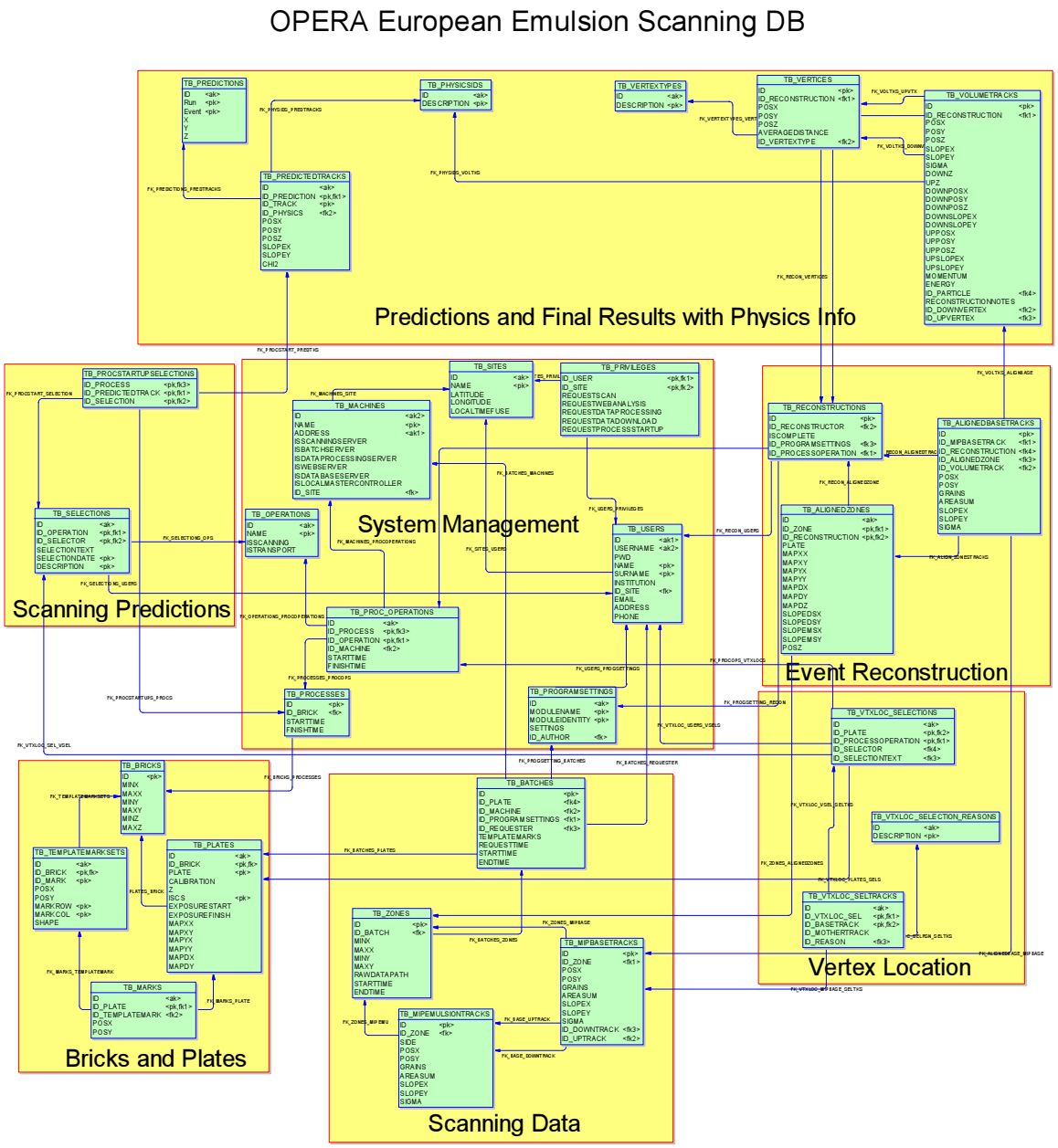

Fig. 2. Overall view of the DB schema

8) Triggers for automatic ID generation ("*before insert*" triggers) must be named "TIB_TB_tablename".

In the following pages, we will run throughout the whole schema to describe in detail the meaning of each table.

3.1. System Management

Management and maintenance of the whole computing infrastructure is supported by eight tables: TB_SITES, TB_USERS, TB_PERMISSIONS, TB_MACHINES, TB_PROCESSES, TB_OPERATIONS, TB_PROC_OPERATIONS and TB_PROGRAMSETTINGS.

TB_SITES contains the definitions of the scanning sites in the European Emulsion Group. The fields of this table are:

a) ID (a unique identifier);
b) Name (the geographical name of the site);
c) Latitude;
d) Longitude;
e) LocalTimeFuse (useful for clock synchronization with the detector).

TB_MACHINES defines the server machines. Personal analysis and development workstations are excluded; only the computers officially entitled for processing and / or scanning must enter this table.

a) ID (a unique identifier);
b) Name (a user-friendly but meaningful name, e.g. SALERNOWEB);
c) Address (DNS name or IP address; the former option should be preferred);
d) ID_SITE (a link to the site where the machine belongs);
e) IsScanningServer (tells whether this machine controls a microscope);
f) IsBatchServer (tells whether this machine runs programs that control the local scheduling of scanning batches; there should be only one Batch Server per site);
g) IsDataProcessingServer (tells whether this machine performs automatic post processing tasks, incremental event reconstructions, and all that is needed to feed the Scanning Servers with meaningful scanning tasks; these machines MUST not be used for personal analysis);
h) IsWebServer (tells whether this machine serves as a Web Interface between humans and all the automatic tasks running on the various servers);
i) IsDataBaseServer (tells whether this machine is part of the DB system; this can be a local copy or replica, or part of a cluster of servers spread over all Europe; however, there should be at least one of these machines per each site);
j) IsLocalMasterController (tells whether this machine runs the local Master Control Program that keeps local brick and event processing under control and applies the scanning policies and strategies).

Many of these services can be grouped together: it should be very natural to have the Master Control Program, the Web Monitor and the Batch Server running on the same machine. Database Servers, Data Processing Servers and Scanning Servers should be dedicated machines.

Bookkeeping is not the only reason to have the list of the machines in the DB. At system start-up, all automatic tasks need to know the local "topology" of the machines they can rely on (e.g. which machines can be used to perform scanning). Instead of having several scattered configuration files, the whole computing infrastructure can be configured and administered in a central, homogeneous way.

TB_USERS keeps track of who are the authorized users of this computing infrastructure. This table is aimed at defining responsibilities in the computing infrastructure, not at securing tables from read / write operations. Integration of this table with the ORACLE server access control should probably be studied carefully. For example, if a rogue user sniffs the password for the account of user "John Smith" for Web access to the computing services, the worst thing he / she can do is to launch meaningless scanning batches or start meaningless event reconstructions, which are easy to detect, but he / she cannot corrupt good data; but if the "John Smith" account is the same that ORACLE uses for DB login, then the hacker can corrupt data or destroy the DB schema, which is much worse. Thus, for the time being, we work in the hypothesis that the account information stored here only keeps track of permissions on the computing infrastructure, not on the DB tables. The fields in this table are:

a) ID (a unique identification number);
b) Username (name the user provides when logging on a computing service);
c) PWD (his / her password, either stored in plain text, or as an irreversibly encrypted hashing code);
d) Name (his / her name);
e) Surname (his / her surname);
f) Institution (name of the institution he / she is working for);
g) ID_SITE (points to the site he belongs to);
h) Email (his / her e-mail address);
i) Address (his / her address);
j) Phone (his / her phone number).

TB_PRIVILEGES is a pivot table: it defines a *many-to-many* relation between sites and users. The actions allowed to each user depend on the site: if "John Smith" belongs to the site of "*Whoknowswhere*" he will be authorized to launch scanning tasks in "*Whoknowswhere*" but not in "*Fairyplace*"; however, he will probably have access to data from both sites. Here below the fields in TB_PRIVILEGES are listed:

a) ID_USER (link to the user that owns these privilege entry);
b) ID_SITE (link to the site these privileges are referred to);
c) RequestScan (tells whether the user can schedule a scanning request in the site);
d) RequestWebAnalysis (tells whether the user can access the local Web site for remote data analysis);
e) RequestDataProcessing (tells whether the user is authorized to start a data processing task on a data processing server in this site);
f) RequestDataDownload (tells whether the user can download large batches of data from local file servers);
g) RequestProcessStartup (tells whether the user can start a new process; see below for the definition of a "process").

TB_PROCESSES is the table that keeps note of what happens to each brick. It is a "journal" of operations. Conceptually, a process starts when a brick is extracted because the Target Tracker triggers it. A process does not necessarily end with a reconstructed event; in order to find an event, it may be necessary to extract several bricks, and a new process starts for each brick. Technical tests and rescanning tasks can be recorded in this table too, so the associated data can be put into the DB. For each process, the following fields are recorded:

a) ID (unique process identifier);
b) ID_BRICK (brick involved);
c) STARTTIME (time/date when the process began);
d) FINISHTIME (time/date when the process finished, or NULL if it is not over yet).

TB_PROC_OPERATIONS keeps track of individual operations. For each process, there are several operations to be carried out: CS scanning, vertex location, vertex selection, precision measurements, and so on; there are also a number of transport operations interleaved with scanning operations, and they also enter the TB_PROC_OPERATIONS table. For each operation, the following fields must be filled:

a) ID (unique operation identifier);
b) ID_PROCESS (link to the owner process);
c) ID_OPERATION (link to the operation descriptor table TB_OPERATIONS; this is a code that identifies the type of operation; see below the explanation of TB_OPERATIONS for more details);
d) ID_MACHINE (link to the machine responsible for carrying out the operation);

e) StartTime (time/date when the operation began);
f) FinishTime (time/date when the operation finished, or NULL if it is not over yet).

In case of critical failures, such as power down, hardware failures and generic interruptions, a simple query on this table shows the pending operations. Therefore, it is possible to define recovery procedures and to code them in programs, so that administrators and people that take care of the scanning tasks always take the correct steps and behave uniformly after every fault.

TB_OPERATIONS defines the various operations a brick can undergo. This table should be filled when setting up the DB and never modified throughout the whole experiment's lifetime. It establishes a correspondence between number codes and scanning activities. Here is a report of its fields:

a) ID (unique operation identifier);
b) Name (a short but meaningful name, such as "CS Scanning" or "Vertex Selection" or "Precision Measurement", or "Transport to Lab");
c) IsScanning (flag that tells whether this is a scanning operation);
d) IsTransport (flag that tells whether this is a transport operation.)

The last two flags are important for Master Control Programs, in order to allocate tasks and resources. Knowing that an operation requires scanning power gives a hint about where to send a brick; transport operations should be grouped in order to optimise usage of trucks and couriers.

TB_PROGRAMSETTINGS permanently stores the program and parameter settings for scanning and reconstruction programs. This helps reproduce efficiencies and background results over long times. For each entry we have:

a) ID (unique identification number);
b) ModuleName (name of the program module);
c) ModuleIdentity (unique identifier, e.g. version number, of the program module);
d) Settings (an XML document storing the parameter settings for the program);
e) ID_AUTHOR (link to the author of the parameter settings entry).

3.2. Predictions

Since this DB is intended to support the scanning activity in Europe, everything begins with a prediction from the electronic sub-detectors. Three tables summarize the information that other sub-detectors can provide to emulsion groups. This part of the DB is somewhat preliminary, and will probably extended with further information. For the time being, only the data that will be surely available have been used to design the tables.

TB_PREDICTIONS yields some basic information:

a) ID (unique event identifier);
a) Run (run number of the neutrino beam);
b) Event (event number for this run);
c) X, Y, Z (3D coordinates in the detector's reference frame of the suggested interaction point).

TB_PREDICTEDTRACKS contains the tracks associated with each predicted event:

a) ID (unique track identifier);
b) ID_PREDICTION (link to the predicted event);
c) ID_TRACK (track number in the corresponding predicted event);
d) ID_PHYSICSID (link to the Physics tag attached to this track; see below for an explanation of Physics IDs);
e) PosX, PosY, PosZ (3D coordinates of a point along the track trajectory);

f) SlopeX, SlopeY (predicted track slopes);
g) Chi2 (χ^2 of the track reconstruction fit; this can be replaced or integrated by any other quality parameter);

TB_PHYSICSIDS lists the Physics tag attached to predicted tracks; depending on the quality of the prediction and on the hits in other sub-detectors, a predicted track might be qualified as a muon, a pion, an electron, an electromagnetic shower, and so on. The entries in this table must be fixed, and they are not intended to vary frequently; in principle, they should stay the same throughout the experiment's lifetime. This table only has two fields:

a) ID (unique Physics tag identifier);
b) Description (a short textual description, such as "muon" or "pion" or "EM shower", etc.);

The importance of this table should not be underestimated. It is much better to have a table with Physics IDs rather than continuously having to remember, e.g. that a "3" in some field means that the electronic sub-detectors have identified the particle as a muon.

3.3. Scanning Predictions

Detector predictions have to undergo some filtering before becoming scanning predictions. One can envisage a policy that avoids starting scanning if the prediction is ambiguous or if its quality is poor, or if the event in the prediction is not likely to be a neutrino beam event. Furthermore, one can decide to search for some predicted track and to disregard other tracks. In order to evaluate efficiency and background of the scanning task as a whole, one needs clear documentation of the selections applied. The EEDB actively supports and documents these policies by two tables.

TB_PROCSTARTUPSELECTIONS links processes (see above) with the selection that accepted the event and defined the tracks to look for. It is a pivot table. It contains the following fields:

a) ID_PROCESS (link to the process in TB_PROCESSES);
b) ID_PREDICTEDTRACK (link to one accepted predicted track);
c) ID_SELECTION (link to the selection that accepted the event and the track);

Each set of three values for ID_PROCESS, ID_PREDICTEDTRACK and ID_SELECTION can of course appear only once.

TB_SELECTIONS keeps track of the selection policy to accept / reject events and tracks for scanning. The design of this table is preliminary, but it is important to foresee its presence from the very beginning. It contains the following fields:

a) ID (unique selection identifier);
b) ID_SELECTOR (link to the user that defined this selection policy and coded it into the DB);
c) ID_OPERATION (the operation the selection refers to: of course, the selection criteria depend on the operation involved; e.g. those for CS scanning are different from those for vertex selection; see above for more details about TB_OPERATIONS);
d) SelectionDate (date when the selection was coded into the DB);
e) SelectionText (text that performs the selection; this is not intended to be a user-friendly description, but rather the text of an SQL query, or an XML document, or the text of some script that actually defines the tracks to be scanned);
f) Description (a user-friendly description of the selection policy).

3.4. Bricks and Plates

It is relatively simple to define the information that must be stored for bricks and for emulsion plates. Though they play a central role in the design of the OPERA experiment, they are simple entities. Plate calibration data (i.e. the reference frame transformations that allow points and vectors on each plate to be referred to the experiment reference frame) are indeed the parameters that deserve more attention; thus we will spend some words about our working assumptions. Since presently the way to identify emulsion plates has not been defined yet, we assume that conventional methods are used; however, the general ideas are always valid.

Each plate of every brick that is extracted from the detector bears a grid of calibration marks that is printed just before development. An emulsion plate can be measured as many times as one wants, but every time it is placed on a microscope stage it is displaced in a slightly different way; the most important effects are translations and rotations, but small contraction / expansion / shear effects can show up, because of temperature changes. The calibration grid has the purpose of providing the readout system with a set of stable reference points. One can assume that this set of points is the same for all the sheets in a brick, i.e., that the negative film used to print the marks undergoes no deformation since it is printed on sheet #1 until it is printed on sheet #56. In principle one can think that the negative film for the grid is exactly the same for all plates during the whole lifetime of the experiment; however, it is safer to foresee that it is replaced, and that it can deform from time to time; so we assume that during the development of the plates of a single brick it stays undeformed. The grid can tell nothing about the misalignments that existed at exposure time. Passing-through tracks are used to recover the original alignment. Transforming the coordinates of each sheet using an affine transformation usually leads to precisions better that 10 μm on a large scale. It is common practice to embed this transformation into the calibration mark grid; thus, instead of having one calibration grid for all plates in a brick, and a different affine transformation for each one, it is possible to generate a ready-to-use calibration grid that already establishes the experiment's reference frame. For operating purposes, it is quicker to use this last grid, whereas for documentation it is important to store the parameters of the affine transformation. At the present time, we choose to store both,

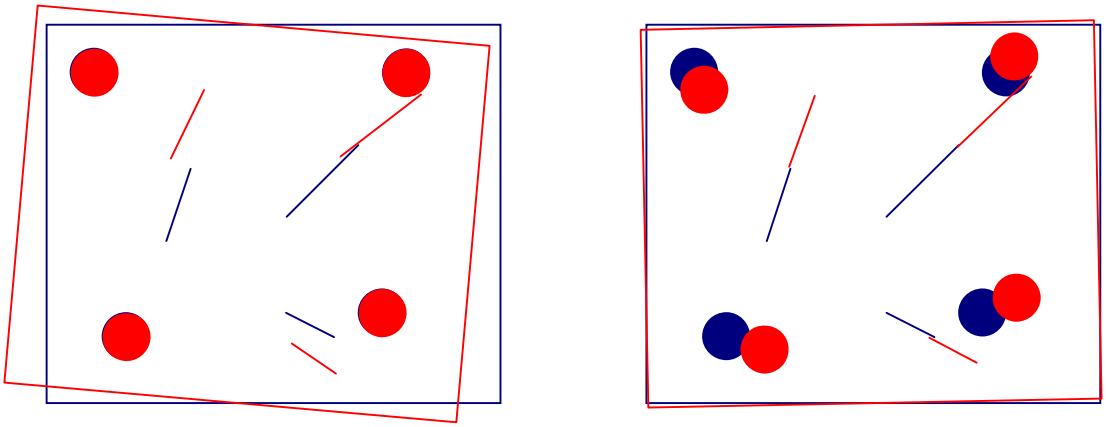

Fig 3. Intercalibration: the red and blue sheets are two consecutive plates in the same brick. Calibration marks are printed after exposure, so they don't mark the same points in the experiment reference frame; after alignment by tracks, the grids differ by an affine transformation, and define a different set of points for each sheet.

but it is clear that the "aligned" calibration grid can be generated on demand as a view. This would avoid the danger of inconsistencies, but there is the concrete risk to perform the same computation thousands of times; of course, this is one possible point to optimise the DB design.

The tables that take care of recording information about bricks and plates are discussed in detail in the following pages.

TB_BRICKS stores general information about bricks, as shown by the list of the fields:

a) ID (unique brick identifier);
b) MinX, MaxX, MinY, MaxY, MinZ, MaxZ (3D extents of the brick shape in experiment reference frame; if the bricks are moved, these are the coordinates it had when the event was triggered within it).

TB_TEMPLATEMARKSETS contains the "template" reference grid for that brick. Since it is impossible to measure the coordinates of the marks on the negative film used for printing, the grid measured on one plate (e.g. CS or Sheet #1) is assumed as the reference grid for all plates in the brick; the "aligned" sets are obtained from this "template" set by applying the alignment affine transformation for each plate. Usually marks lay in a square-cell lattice arrangement. The information we plan to store for each template set are:

a) ID (unique mark identifier);
b) ID_BRICK (link to the owner brick);
c) ID_MARK (number of the mark in the set);
d) PosX, PosY (tentative X and Y positions in the experiment reference frame);
e) MarkRow, MarkCol (row and column number of the grid mark in the square lattice arrangement);
f) Shape (if the marks don't have all the same shape, as in previous experiments with emulsions, this number identifies the shape).

TB_PLATES stores overall information about each plate. The set of fields shown below will probably be enriched with new entries such as the average thickness, the emulsion production year, and so on.

a) ID (unique plate identifier);
b) ID_BRICK (link to the owner brick);
c) Plate (plate number in the brick, ranging from 1 to 56 for target sheets, and an integer greater than 0 for CS);
d) IsCS (tells whether this plate is a CS or a target sheet);
e) Z (adjusted longitudinal position after sheet-to-sheet alignment using cosmic rays);
f) ExposureStart (time / date of the beginning of the exposure of this plate);
g) ExposureFinish (time / date of the end of the exposure of this plate);
h) MapXX, MapXY, MapYX, MapYY, MapDX, MapDY (parameters of the affine transformation that aligns this plate to the experiment reference frame, according to the formula:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} MapXX & MapXY \\ MapYX & MapYY \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} MapDX \\ MapDY \end{pmatrix}$$

TB_MARKS is actually conceived as a table, but, as said above, it can be implemented as a view. It contains the following information for ready-to-use aligned marks:

a) ID (unique mark identifier);
b) ID_PLATE (plate identifier, linked to ID in TB_PLATES; it is NOT the plate number in the brick, but the DB unique plate ID);
c) ID_TEMPLATEMARK (mark identifier, linked to ID in TB_TEMPLATEMARKS);
d) PosX, PosY (aligned X, Y position of the mark in this sheet).

3.5. Scanning Data

The tracks read out from emulsion must be organized in the DB. Although this section of the DB is conceptually very simple, the design choices and the operational policy we will apply on this subject will affect heavily both the performance and the use of the DB, so we must be very careful, and possibly keep doors open for other ideas and future changes. First of all, let's set some solid ground to start working. Our present experience with emulsion from OPERA test beam exposures is that in many cases, we have obtained data files as large as 100 MB for 1 cm^2 scanning areas. The files contained mostly unassociated microtracks (i.e. microtracks that did not pair with other microtracks on the other side of the sheet). Their density increases with decreasing emulsion thickness, with the exposure to cosmic rays and background radiation, and we know that for emulsion plates refreshed, exposed and developed in the conditions of the OPERA detector the density of such meaningless tracks will be much lower than it is now; however, their density also depends on instrumental factors, and it is very difficult to estimate now their amount. A few storage policies can be envisaged and discussed:

a) Store only the base tracks in the DB and their associated microtracks; a single table contains all information; this poses some limitations to offline reconstruction and analysis algorithms, because they cannot use the DB for tasks such as the search for τ decays in the plastic base, or alternative microtrack linking procedures; this would limit the use of the DB; however, this option minimizes its size.
b) Store all microtracks in the DB; base tracks and microtracks are stored in separate tables, linked through a relationship; offline reconstruction and analysis algorithms can exploit the maximum amount of information; if the number of microtracks is too large, the DB size will "explode", and it will mostly behave as a slow, inefficient file system.
c) Store the base tracks in the DB and their associated microtracks in zones far from the vertices, and all microtracks in selected areas around "interesting points" (then a strategy must be decided to define "interesting points"); this should pose very few limitations to offline reconstruction and analysis algorithms, while keeping the DB reasonably small; base tracks and microtracks are stored in separate tables, linked by a relationship.

Option (a) looks too risky; option (b) and option (c) define the same DB structure, and only the usage policy changes. So, at present, we work in the assumption that the final option will be either (b) or (c). The proposed design for this section of the DB presently requires four tables.

TB_BATCHES records the scanning batches; on each single plate, several areas will be scanned; even to align two plates, at least three zones must be scanned, and they will obviously be grouped in a single batch. Presently, no information is stored about environmental conditions during scanning, but this is an upgrade that is easy obtained by modifying this table. Currently we propose to store:

a) ID (unique identification number);
b) ID_PLATE (tells the plate where the scanning was performed, by linking the batch to a related plate in the TB_PLATES table);
c) ID_MACHINE (links the batch to the machine, listed in the TB_MACHINES table, that materially drove the microscope that scanned the areas in the batch);
d) ID_PROGRAMSETTINGS (links the batch to the program settings entry in the TB_PROGRAMSETTINGS table that stores the scanning parameter configuration used for this scanning task);

e) ID_REQUESTER (links the batch to the user that requested it in the TB_USERS table);
f) TemplateMarks (0 if the batch was executed using calibrated marks, 1 if it was executed using template marks);
g) RequestTime (time / date when the batch was requested);
h) StartTime (time / date when the batch started or NULL if it has not started yet);
i) EndTime (time / date when the batch finished or NULL if it is still in progress or not started yet).

TB_ZONES keeps track of the scanning areas that make up a batch. For each zone we store:

a) ID (unique identification number);
b) ID_BATCH (owner batch ID);
c) MinX, MaxX, MinY, MaxY (extents of the scanning area);
d) RawDataPath (full path to the raw data files where the microscope stored its output; post-processing programs use this information to know which files must be reprocessed, e.g. for mechanical systematic error subtraction);
e) StartTime (time / date when the zone started or NULL if it has not started yet);
f) EndTime (time / date when the zone finished or NULL if it is still in progress or not started yet).

Notice that with this information, in case of a power interruption, hardware failures or any other cause of interruption, it is easy to restart from the very point where the process stopped. A failure recovery program can easily check the TB_PROCESSES table and the TB_PROC_OPERATIONS tables to know what was being done, and the TB_BATCHES and TB_ZONES document the progress of each operation.

TB_MIPEMULSIONTRACKS is the table that stores the microtracks for all zones. It records:

a) ID (unique microtrack identification number);
b) ID_ZONE (zone the microtrack belongs to);
c) Side (0 for top side tracks and 1 for bottom side tracks);
d) PosX, PosY (X, Y coordinates of the point of the track at the nearest surface of the plastic base in the sheet);
e) Grains (number of grains in the track);
f) AreaSum (sum of the areas, measured in pixels, of the grains in the track);
g) SlopeX, SlopeY (X, Y components of the track slope);
h) Sigma (average residual deviation of the grains from the trajectory fit).

TB_MIPBASETRACKS is a denormalized table. It contains the base tracks obtained by joining microtracks in either sides of the emulsion plate. As such, most records could in principle be computed on-the-fly from microtrack parameters; however, this table is accessed very often, and the parameters of a base track are not expected to change once they are defined; thus, for the time being, we implement this as a table, rather than as a view. Of course this choice might change if it looks convenient. Each record presents:

a) ID (unique base track identification number);
b) ID_ZONE (links the base track to the zone; this could be obtained as an indirect link through TB_MIPEMULSIONTRACKS, but this would slow down too much);
c) PosX, PosY (X, Y coordinates of the point of the track at the downstream surface of the plastic base in the sheet);
d) Grains (number of grains in the track; indeed this is simply the sum of the number of grains in the two microtracks);
e) AreaSum (sum of the areas, measured in pixels, of the grains in both microtracks);
f) SlopeX, SlopeY (X, Y components of the track slope);

g) Sigma (quality parameter for base tracks);
h) ID_DOWNTRACK (links the base track to the microtrack on the downstream side of the sheet);
i) ID_UPTRACK (links the base track to the microtrack on the upstream side of the sheet).

It is worth to notice that all microtracks from all sheets from all bricks and for any task are stored in the same table. The same is true for base tracks. This simplifies the DB structure, but could in principle be harmful for performance in data retrieval; however, if we set properly the table indices and storage parameters (properly defining the data partitioning and replication policy), there is no performance degradation. Moreover, the fact that all data have the same structure simplifies analysis and reconstruction programs that read from / write to the DB.

3.6. Vertex Location Structures

Most of the DB sub-areas we described up to now are not really specific to the OPERA experiment, and would be very similar for other experiments with emulsions. Now we come to consider aspects more tightly connected to the specific features of OPERA. In the present design of the DB schema, we have taken the general concepts of the experiment into account, bearing in mind the clear consciousness that the details might change as the scanning strategy becomes more and more clear.

Event processing can be divided in 4 stages: CS validation, vertex location, vertex selection and precision measurements. Conceptually, CS validation is very simple; at the present time, it looks that vertex selection and precision measurements require the scanning of certain volumes that can be defined *a priori* once a rough sketch of the event structure is known. On the other hand, the vertex location procedure requires the monitoring tasks to take quasi-online decisions to "drive" the scanning. Since the scanning systems will have to scan back tracks seen on the CS and / or predicted by the target trackers, on each scanning zone they will have to look into the scanning output (TB_MIPEMULSIONTRACKS and TB_MIPBASETRACKS tables), find the best candidate (if any) for the scan-back track, update it from sheet to sheet, project it to the next sheet, and so on; alternatively, the scanning monitors will detect multi-track crossings, and mark them as vertices or kinks, and possibly update the scan-back predictions. We must allow human intervention in this delicate phase, but the computing infrastructure will in most cases take decisions on its own. In the case of vertex selection and precision measurement, one defines the features of the event to be studied, performs the scanning, and then analyses the output data, taking decisions just at the end of each stage; vertex location requires decisions to be taken after every sheet before scanning the next one. These decisions must be documented and recorded; at this point, the DB is the "shared memory" of the whole computing infrastructure. The tables we need, in this preliminary draft, are listed below.

TB_VTXLOC_SELECTIONS is the highest level table. It documents the selection criteria applied, so that the decision procedure can be exactly reproduced. Its records contain:

a) ID (unique identification number for the selection);
b) ID_PLATE (links to the plate in the TB_PLATES table the selection applies to);
c) ID_PROCESSOPERATION (identifies the process operation in the TB_PROC_OPERATIONS table that corresponds to the present Vertex Location procedure);
d) ID_SELECTOR (identifies the author of the selection text in the list of users);
e) ID_SELECTIONTEXT (identifies the selection text in the TB_SELECTIONS table that has been used to select scan back tracks; this is intended to be an SQL statement, or

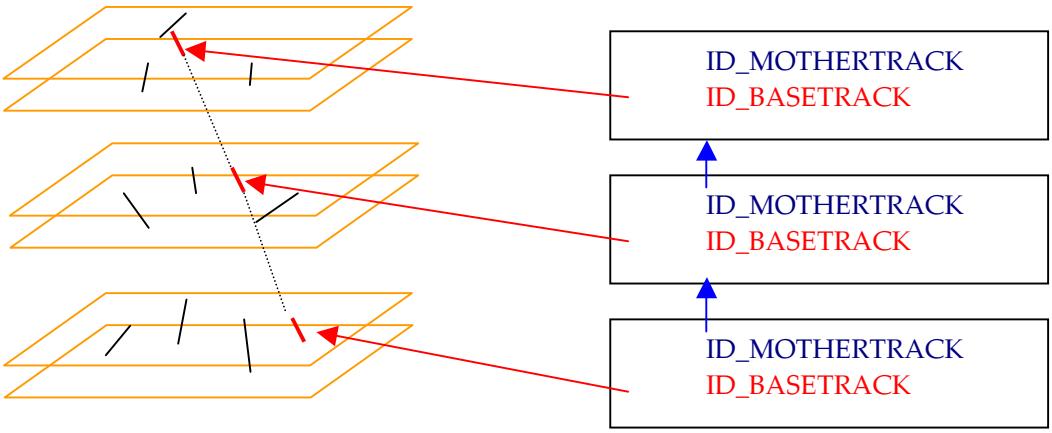

Fig. 4. A scan-back path as defined by records in the TB_VTXLOC_SELTRACKS table.

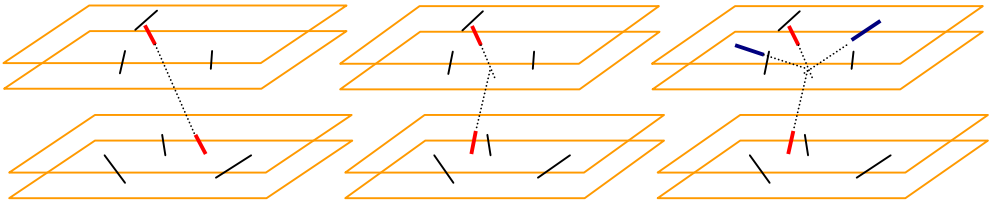

Fig 5. From left to right, the three possible logical relationships between a mother track and its candidate daughter in the scan-back: ***straight line***, ***kink***, *and* ***vertex***.

an XML structure, or some kind of really operating code that actually produces the selection, rather than a user-friendly description).

The entries in this table can be also used in efficiency and background estimate, to run Monte Carlo simulations and check the behaviour of the overall event handling procedure.

TB_VTXLOC_SELTRACKS records which tracks in each sheet have been marked as scan-back track candidates. This is essentially a pivot table that represents just logical relationships. At each sheet, each scan-back path is updated with a new track that satisfies one of three conditions:

I) the new track continues the scan-back path in a straight line;
II) the new track crosses the scan-back path trajectory in a kink topology;
III) the new track crosses the scan-back path trajectory in a vertex topology;

Without executing any code, it is sufficient to read in this table the index of the scan-back track from sheet to sheet for each event, to get the overall event structure. Its fields are listed below:

a) ID (unique identification number);
b) ID_VTXLOC_SEL (the selection that accepted the track as a scan-back candidate; this is a link to an entry in TB_VTXLOC_SELECTIONS);
c) ID_BASETRACK (identifies the base track in the TB_MIPBASETRACKS table that has been identified as the best scan-back candidate; its geometrical track parameters are stored in TB_MIPBASETRACKS, and data replication is avoided);
d) ID_MOTHERTRACK (the ID, in this same table, of the record that describes the last scan-back candidate for this scan-back path; by simply running through this index,

one can examine the scan-back history for each scan-back path, from the vertex, or vertices, up to the CS);

e) ID_REASON (states why the track defined by ID_BASETRACK has been attached to the table entry defined by ID_MOTHERTRACK; links this entry to one of the logical reasons listed in table TB_VTXLOC_SELECTION_REASONS, explained below).

TB_VTXLOC_SELECTION_REASONS is a table that must be filled up at the time when the DB is first set up, and never touched throughout the experiment's lifetime. It establishes a connection between a set of numbers and logical relationships between scan-back paths and their candidates on each sheet. It has only two fields:

a) ID (unique identification number);

b) Description (short user friendly description of the relationship: possible entries should be "STRAIGHT LINE", "KINK" and "VERTEX").

As seen, the DB architecture sketched here completely separates the logical meaning of data from the data themselves, thus allowing different, alternative algorithms to process them without any limitation.

3.7. Event Reconstruction

For each event, already after vertex selection, there is a partial, rough event reconstruction available; this is the basis on which precision measurements are performed; the precision measurement stage ends up with a reconstruction that, even not definitive (we don't want to touch now the subject of the final reconstruction that enters the neutrino oscillation search statistics for the Collaboration), is probably the final product of the European Emulsion Scanning Group. Further, alternative, more detailed reconstructions, obtained by other analysis and reconstruction programs are still possible using the data in the DB; however, the scanning process naturally ends up with a proposed event reconstruction. Intermediate, incremental event reconstructions are also possible and indeed needed at some point in the scanning process. As soon as these reconstruction become of public interest (i.e., they are not private reconstructions for algorithm development purposes) they deserve the right to be stored in the DB, so all members of the European Emulsion Group (and even interested members of the Collaboration, of course) can access them.

Although event reconstructions represent very refined data, and the measurements and analysis task one can perform on them are very sophisticated, their logical structure is very simple. A "volume track" in an ECC is built by connecting base tracks from several sheets together, and vertices and kinks are obtained as topological crossing points of two or more volume tracks. In order to reach the high accuracy required by the OPERA experiment, the local alignment of sheets must be adjusted using reference tracks. This simple description is easily coded into DB tables.

TB_RECONSTRUCTIONS is the main table dealing with event reconstruction. It is useful for documentation and bookkeeping. Its records are defined as having the following fields:

a) ID (unique identification number);

b) ID_PROCESSOPERATION (the event reconstruction is the final product of many process operations; this link is also useful to quickly connect the reconstruction to its associated prediction, but formally it is redundant, so this is a denormal table);

c) ID_RECONSTRUCTOR (the user who requested this reconstruction; this is a link to the TB_USERS table);

d) ID_PROGRAMSETTINGS (link to the entry in the TB_PROGRAMSETTINGS table that contains the reconstruction parameters);

e) IsComplete (flag that tells whether the reconstruction is complete or partial).

TB_ALIGNEDZONES stores local alignment information. For event reconstructions, tracks are aligned on a very local basis, zone by zone. Usually the alignment parameters for track positions are those needed to define a local affine transformation and a longitudinal displacement; for slopes, usually multiplicative and additive adjustments are sophisticated enough to satisfy the needs. We review the parameters, and then we show the related correction formulae:

a) ID (unique zone identifier);
b) ID_ZONE (links the aligned zone to the related scanning zone in the TB_ZONES table);
c) ID_RECONSTRUCTION (links the aligned zone to the owner reconstruction in the TB_RECONSTRUCTIONS table);
d) PLATE (order number of this sheet in the brick); shortcut link to the corresponding plate; because of the presence of ID_ZONE, this is redundant, and the table is denormal; however, it avoids two very frequent joins in SQL queries, so we find it useful);
e) MapXX, MapXY, MapYX, MapYY, MapDX, MapDY, MapDZ (track position transformation parameters);
f) SlopeDSX, SlopeDSY, SlopeMSX, SlopeMSY (track slope transformation parameters).
g) PosZ (adjusted longitudinal position for the plate).

The transformation parameters are applied to tracks in the TB_MIPBASETRACKS table related to the ID_ZONE zone, according to the formulae:

For positions:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} MapXX & MapXY & 0 \\ MapYX & MapYY & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} MapDX \\ MapDY \\ MapDZ \end{pmatrix}$$

For slopes:

$$\begin{pmatrix} S'_x \\ S'_y \end{pmatrix} = \begin{pmatrix} MapXX & MapXY \\ MapYX & MapYY \end{pmatrix} \left[\begin{pmatrix} SlopeMSX & 0 \\ 0 & SlopeMSY \end{pmatrix} \begin{pmatrix} S_x \\ S_y \end{pmatrix} + \begin{pmatrix} SlopeDSX \\ SlopeDSY \end{pmatrix} \right]$$

The position deformation matrix also contributes in the slope transformation, otherwise the coherence between slopes and positions would be lost.

TB_ALIGNEDBASETRACKS is a denormal table that records the result of the application of the above formulae to the tracks in the TB_MIPBASETRACKS table; however, for each reconstruction this computation is to be performed only once, so probably a computed view would be impractical. The fields in this table are almost the same we find in TB_MIPBASETRACKS:

a) ID (unique track identifier);
b) ID_RECONSTRUCTION (link to the related owner reconstruction);
c) ID_VOLUMETRACK (link to the volume track that owns this aligned base track, or NULL if the aligned track is not part of a volume track; see below for a definition of volume tracks);

d) ID_MIPBASETRACK (link to the original base track; since several reconstructions are possible, the same base track can spawn several aligned base tracks);
e) ID_ALIGNEDZONE (link to the aligned zone in the TB_ALIGNEDZONES table to which the aligned track belongs);
f) PosX, PosY (X, Y aligned coordinates of the aligned track);
g) Grains (number of grains in the aligned track);
h) AreaSum (sum of the area, in pixels, of all the grains in the track);
i) SlopeX, SlopeY (X, Y aligned components of the track slope);
j) Sigma (track quality parameter).

TB_VOLUMETRACKS records the volume tracks in the event reconstruction. These are naturally conceived as sequences of aligned base tracks, but in most cases ready-to-use global parameters are needed. For each track, the results of a global fit and of two local fits (most downstream and most upstream) are stored. The table records several parameters for each volume track, as follows:

a) ID (unique identification number);
b) ID_RECONSTRUCTION (links the track to its owner reconstruction);
c) PosX, PosY, PosZ (the 3D coordinates of a point along the track trajectory);
d) SlopeX, SlopeY (components of the global slope);
e) Sigma (track quality parameter);
f) DownZ (Z of the most downstream point of the volume track seen in emulsion);
g) UpZ (Z of the most upstream point of the volume track seen in emulsion);
h) DownPosX, DownPosY, DownPosZ (the 3D coordinates of a point along the track trajectory fit using the most downstream base tracks);
i) DownSlopeX, DownSlopeY (components of the slope computed using the most downstream base tracks);
j) UpPosX, UpPosY, UpPosZ (the 3D coordinates of a point along the track trajectory fit using the most upstream base tracks);
k) UpSlopeX, UpSlopeY (components of the slope computed using the most upstream base tracks);
l) Momentum (the corresponding particle momentum if available; NULL otherwise);
m) Energy (the corresponding particle energy if available; NULL otherwise);
n) ID_PARTICLE (link to the ID_PHYSICSIDS table that sets the possible particle identification);
o) ID_DOWNVERTEX (link to the downstream vertex, if any);
p) ID_UPVERTEX (link to the upstream vertex, if any);
q) ReconstructionNotes (a flag field that can be used to add notes about the reconstruction).

TB_VERTICES is the table that stores vertex and kink information. For the DB schema, kinks are vertices with one upstream track and one downstream track. The fields are:

a) ID (unique vertex identification number);
b) ID_RECONSTRUCTION (links the vertex to its owner reconstruction);
c) PosX, PosY, PosZ (3D coordinates of the vertex point);
d) AverageDistance (average distance of volume tracks extrapolations to the vertex Z from the vertex point; this is a measure of vertex quality);
e) ID_VERTEXTYPE (link to the TB_VERTEXTYPES table that defines the kind of vertex; see below for an explanation).

TB_VERTEXTYPES stores a textual description of vertex types; as other tables in this DB schema, this table should be filled up once when the DB is set up, and should never be modified. It only has two fields:

a) ID (unique identification number);
b) Description (a short description, such as "Decay", "Neutrino Interaction", "Secondary Interaction").

This completes the description of the DB schema. It is worth to notice that it develops as a circle starting from Physics predictions from the electronic sub-detectors, and it ends with the final results of event reconstruction from scanning data and Physics flags to identify particles and interactions.

4 Computing Resources Organization

As stated at the very beginning of this note, the DB must serve as the core of the computing infrastructure for the European Emulsion Scanning System. This means that it will be involved in almost all operations, both as a data source (to retrieve data) and as a data sink (to store data). The usage of this DB indeed will be rather atypical: most events will not be interesting, so they will be written and almost never read. Every time a microscope scans a zone, its results will go to the DB, even if there is just meaningless background. Furthermore, almost all data will be made of background tracks for alignment purposes, accompanying the few tracks that record the history of neutrino events. In this situation, write operations can really be a bottleneck for the whole system. On top of that, we add three goals: immediate availability of all data to the whole European Emulsion Group and to the OPERA Collaboration, reduced network traffic, and data safety. Clearly these are clashing goals. The architecture we propose should be able to reconcile them, taking profit of the scalability features of ORACLE DB systems.

In order to achieve data safety, multiple copies of the whole DB must exist and be synchronized. This system cannot be backed up frequently (during the months of beam activity the system cannot be taken off-line for backups; each backup consists of several hundreds of GBs, so it's not a quick operation). Organizing the DB data files in some clever way can help, but the situation is not easy to solve by means of simple disk backup policies. It is better to have the DB system take care of producing multiple copies: ORACLE servers can be joined together in Multi-Master Replication Groups. Each server is a peer in the group, and all servers contain equivalent copies of data. Every time some data is added or updated on one server, the DB Management System (**DBMS**) takes care of propagating the changes to other servers in a fully automatic way. Therefore, we have multiple copies of data on separated machines, which increases data safety. Changes are propagated as soon as possible (depending on network traffic conditions) in the form of "delayed transactions". The Multi-Master Replication Group should physically reside in a central location, and communications among these machines should be so fast that they act as a single unit. We will call this group the Core DB Group (**CDBG**).

This Multi-Master Replication Group is the core in the DB system. If it were the only place where data are stored and retrieved, the whole system would have a very weak point: the network. First, data are actually produced in local laboratories, and they must be efficiently delivered to the DB system core; second, the laboratories themselves need quick and continuous access to data in order to perform scanning tasks. In order to avoid that network interruptions paralyse completely a laboratory, and to work also in heavy traffic conditions, the only solution is to have local copies of the DB. However, local laboratories don't need to have copies of the whole data set; furthermore, it would be an unneeded waste of network bandwidth to replicate all changes in the CDBG to local DBs. It is intuitively clear that if every time the laboratory of

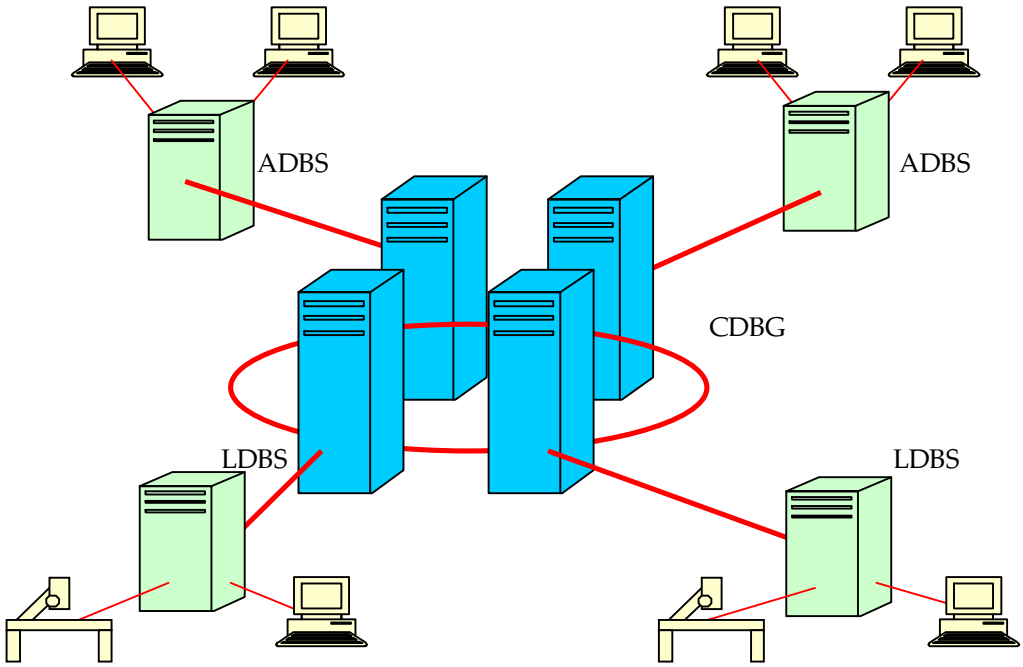

Fig 6. Sketch of the DB system organization. The core is the CDBG. Scanning laboratories rely on Local DB Systems. Offline analysis groups use Analysis DB Systems.

Whoknowswhere inserts a set of tracks into the DB, this change must be propagated over all Europe, the network traffic would increase substantially without any benefit. It is much better that each laboratory stores only a copy of its own data locally; the local system keeps working even in absence of network connection with the Core DB Group, and the bandwidth needed is reduced. Again, we take profit of ORACLE scalability features, and we implement Local DB Systems (**LDBS**) mostly as *materialized views*. For our purposes, a materialized view is a local synchronized copy of a subset of the data contained in a table. The laboratory actually works updating and inserting rows into its materialized views; the DBMS propagates the changes to the CDBG when network traffic allows to, again by delayed transactions.

The last point to deal with is security. A minimal set of access restrictions on tables and stored procedures helps maintain data integrity; however, there are still chances to perform a *query attack*, i.e. to overwhelm the CDBG computing power by executing too complex queries. This kind of attack can be the deed of hackers or rogue users, as well as the result of improper usage by inexperienced users. As in every Physics experiment, several users will be undergraduate students or Ph.D. students, and not everybody can be a DB expert and understand that some operation can affect seriously the overall system performance. Most commonly queried tables and data can be hosted by one or more dedicated machines (Analysis DB Systems – **ADBS**), again containing materialized views rather than the actual tables. So, if some user is performing some very heavy analysis, the worst thing that can happen is that all analysis tasks only at his / her site are slowed down, but the scanning goes on without noticing any change or slowdown; the CDBG and other analysis sites experience no trouble as well. On top of that, it is a good policy to have skilled, Physics-aware DB administrators design some ready-made stored procedures and queries, to help users in most common tasks. This will stimulate people to use the DB system, and discourage the practice of having private copies of data for analysis purposes. Performance-critical tables are then accessed in controlled ways, not directly, but through safe stored procedures.

The proposed architecture qualitatively reaches all design goals. Fine tuning and practical implementation need quantitative analysis of data set size, access performances and network traffic. These items will be addressed in a future note.

Appendix: SQL code for the European Emulsion Scanning DB

This section contains two scripts that implement the draft DB schema shown in the previous pages. The scripts do not contain sophisticated features such as permissions, storage options, replication policies and so on, and are intended to run on stand-alone ORACLE instances.

The first script creates the schema. The second one creates the "Integrity Package", i.e. a set of triggers and procedures that bind sequences to columns to generate unique identifiers for the tables. We have chosen to keep this script separated, since its implementation for replicating environments could change considerably.

Both scripts try to drop previously existing items with the same name as those that are about to be created; if those items do not exist, error messages are generated, but they can be safely ignored. The code has been successfully tested on ORACLE 9iDS running on Windows 2000 and Windows XP Professional, and the database was successfully accessed from Windows and Linux machines. The code creates schema objects in the user's default tablespace and schema. There are no special naming requirements.

The ideal tool to execute these scripts is SQLPlus Worksheet (a standard ORACLE tool).

Schema creation

```
/*==============================================================*/
/* Database name:  OPERA_EUROPEAN_EMULSION_SCANNING_DB          */
/* DBMS name:      ORACLE Version 9i                            */
/* Created on:     21-Feb-03 11:27:05 AM                        */
/*==============================================================*/


alter table TB_ALIGNEDBASETRACKS
   drop constraint FK_ALIGNEDBASE_MIPBASE
/


alter table TB_ALIGNEDBASETRACKS
   drop constraint FK_RECON_ALIGNEDTRACKS
/


alter table TB_ALIGNEDBASETRACKS
   drop constraint FK_ALIGN_ZONESTRACKS
/


alter table TB_ALIGNEDBASETRACKS
   drop constraint FK_VOLTKS_ALIGNBASE
/


alter table TB_ALIGNEDZONES
   drop constraint FK_RECON_ALIGNEDZONE
/


alter table TB_ALIGNEDZONES
   drop constraint FK_ZONES_ALIGNEDZONES
/


alter table TB_BATCHES
   drop constraint FK_BATCHES_MACHINES
/


alter table TB_BATCHES
   drop constraint FK_BATCHES_PLATES
/


alter table TB_BATCHES
   drop constraint FK_BATCHES_REQUESTER
/


alter table TB_BATCHES
   drop constraint FK_PROGSETTING_BATCHES
/


alter table TB_MACHINES
   drop constraint FK_MACHINES_SITE
/


alter table TB_MARKS
```

```
   drop constraint FK_MARKS_PLATE
/


alter table TB_MARKS
   drop constraint FK_MARKS_TEMPLATEMARK
/


alter table TB_MIPBASETRACKS
   drop constraint FK_BASE_DOWNTRACK
/


alter table TB_MIPBASETRACKS
   drop constraint FK_BASE_UPTRACK
/


alter table TB_MIPBASETRACKS
   drop constraint FK_ZONES_MIPBASE
/


alter table TB_MIPEMULSIONTRACKS
   drop constraint FK_ZONES_MIPEMU
/


alter table TB_PLATES
   drop constraint FK_PLATES_BRICK
/


alter table TB_PREDICTEDTRACKS
   drop constraint FK_PHYSIDS_PREDTRACKS
/


alter table TB_PREDICTEDTRACKS
   drop constraint FK_PREDICTIONS_PREDTRACKS
/


alter table TB_PRIVILEGES
   drop constraint FK_SITES_PRIVILEGES
/


alter table TB_PRIVILEGES
   drop constraint FK_USERS_PRIVILEGES
/


alter table TB_PROCESSES
   drop constraint FK_BRICKS_PROCESSES
/


alter table TB_PROCSTARTUPSELECTIONS
   drop constraint FK_PROCSTART_PREDTKS
/


alter table TB_PROCSTARTUPSELECTIONS
   drop constraint FK_PROCSTART_SELECTION
/


alter table TB_PROCSTARTUPSELECTIONS
   drop constraint FK_PROCSTARTUPS_PROCS
/


alter table TB_PROC_OPERATIONS
   drop constraint FK_MACHINES_PROCOPERATIONS
/


alter table TB_PROC_OPERATIONS
   drop constraint FK_OPERATIONS_PROCOPERATIONS
/


alter table TB_PROC_OPERATIONS
   drop constraint FK_PROCESSES_PROCOPS
/


alter table TB_PROGRAMSETTINGS
   drop constraint FK_USERS_PROGSETTINGS
/


alter table TB_RECONSTRUCTIONS
   drop constraint FK_PROCOPERA_RECON
/


alter table TB_RECONSTRUCTIONS
   drop constraint FK_PROGSETTING_RECON
/


alter table TB_RECONSTRUCTIONS
   drop constraint FK_RECON_USERS
/
```

```
alter table TB_SELECTIONS
   drop constraint FK_SELECTIONS_OPS
/


alter table TB_SELECTIONS
   drop constraint FK_SELECTIONS_USERS
/


alter table TB_TEMPLATEMARKSETS
   drop constraint FK_TEMPLATEMARKSETS
/


alter table TB_USERS
   drop constraint FK_SITES_USERS
/


alter table TB_VERTICES
   drop constraint FK_RECON_VERTICES
/


alter table TB_VERTICES
   drop constraint FK_VERTEXTYPES_VERTICES
/


alter table TB_VOLUMETRACKS
   drop constraint FK_VOLTKS_DOWNVTX
/


alter table TB_VOLUMETRACKS
   drop constraint FK_PHYSIDS_VOLTKS
/


alter table TB_VOLUMETRACKS
   drop constraint FK_RECON_VOLTKS
/


alter table TB_VOLUMETRACKS
   drop constraint FK_VOLTKS_UPVTX
/


alter table TB_VTXLOC_SELECTIONS
   drop constraint FK_PROCOPS_VTXLOCS
/


alter table TB_VTXLOC_SELECTIONS
   drop constraint FK_VTXLOC_PLATES_SELS
/


alter table TB_VTXLOC_SELECTIONS
   drop constraint FK_VTXLOC_SEL_VSEL
/


alter table TB_VTXLOC_SELECTIONS
   drop constraint FK_VTXLOC_USERS_VSELS
/


alter table TB_VTXLOC_SELTRACKS
   drop constraint FK_VTXLOC_SELRSN_SELTKS
/


alter table TB_VTXLOC_SELTRACKS
   drop constraint FK_VTXLOC_MIPBASE_SELTKS
/


alter table TB_VTXLOC_SELTRACKS
   drop constraint FK_VTXLOC_VSEL_SELTKS
/


alter table TB_ZONES
   drop constraint FK_BATCHES_ZONES
/


drop table TB_ALIGNEDBASETRACKS cascade constraints
/


drop table TB_ALIGNEDZONES cascade constraints
/


drop table TB_BATCHES cascade constraints
/


drop table TB_BRICKS cascade constraints
/
```

```
drop table TB_MACHINES cascade constraints
/

drop table TB_MARKS cascade constraints
/

drop table TB_MIPBASETRACKS cascade constraints
/

drop table TB_MIPEMULSIONTRACKS cascade constraints
/

drop table TB_OPERATIONS cascade constraints
/

drop table TB_PHYSICSIDS cascade constraints
/

drop table TB_PLATES cascade constraints
/

drop table TB_PREDICTEDTRACKS cascade constraints
/

drop table TB_PREDICTIONS cascade constraints
/

drop table TB_PRIVILEGES cascade constraints
/

drop table TB_PROCESSES cascade constraints
/

drop table TB_PROCSTARTUPSELECTIONS cascade constraints
/

drop table TB_PROC_OPERATIONS cascade constraints
/

drop table TB_PROGRAMSETTINGS cascade constraints
/

drop table TB_RECONSTRUCTIONS cascade constraints
/

drop table TB_SELECTIONS cascade constraints
/

drop table TB_SITES cascade constraints
/

drop table TB_TEMPLATEMARKSETS cascade constraints
/

drop table TB_USERS cascade constraints
/

drop table TB_VERTEXTYPES cascade constraints
/

drop table TB_VERTICES cascade constraints
/

drop table TB_VOLUMETRACKS cascade constraints
/

drop table TB_VTXLOC_SELECTIONS cascade constraints
/

drop table TB_VTXLOC_SELECTION_REASONS cascade constraints
/

drop table TB_VTXLOC_SELTRACKS cascade constraints
/

drop table TB_ZONES cascade constraints
/

drop sequence SEQ_ALIGNEDBASETRACKS
/
```

```
drop sequence SEQ_ALIGNEDZONES
/

drop sequence SEQ_BATCHES
/

drop sequence SEQ_BRICKS
/

drop sequence SEQ_MACHINES
/

drop sequence SEQ_MARKS
/

drop sequence SEQ_MIPBASETRACKS
/

drop sequence SEQ_MIPEMULSIONTRACKS
/

drop sequence SEQ_PLATES
/

drop sequence SEQ_PREDICTEDTRACKS
/

drop sequence SEQ_PREDICTIONS
/

drop sequence SEQ_PROCESSES
/

drop sequence SEQ_PROC_OPERATIONS
/

drop sequence SEQ_PROGRAMSETTINGS
/

drop sequence SEQ_RECONSTRUCTIONS
/

drop sequence SEQ_SELECTIONS
/

drop sequence SEQ_SITES
/

drop sequence SEQ_TEMPLATEMARKSETS
/

drop sequence SEQ_USERS
/

drop sequence SEQ_VERTICES
/

drop sequence SEQ_VOLUMETRACKS
/

drop sequence SEQ_VTXLOC_SELECTIONS
/

drop sequence SEQ_VTXLOC_SELTRACKS
/

drop sequence SEQ_ZONES
/


create sequence SEQ_ALIGNEDBASETRACKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_ALIGNEDZONES
increment by 1
start with 1
nocycle
order
```

```
/


create sequence SEQ_BATCHES
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_BRICKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_MACHINES
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_MARKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_MIPBASETRACKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_MIPEMULSIONTRACKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_PLATES
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_PREDICTEDTRACKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_PREDICTIONS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_PROCESSES
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_PROC_OPERATIONS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_PROGRAMSETTINGS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_RECONSTRUCTIONS
increment by 1
start with 1
nocycle
order
/
```

```
create sequence SEQ_SELECTIONS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_SITES
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_TEMPLATEMARKSETS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_USERS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_VERTICES
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_VOLUMETRACKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_VTXLOC_SELECTIONS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_VTXLOC_SELTRACKS
increment by 1
start with 1
nocycle
order
/


create sequence SEQ_ZONES
increment by 1
start with 1
nocycle
order
/


/*==============================================================*/
/* Table: TB_ALIGNEDBASETRACKS                                  */
/*==============================================================*/


create table TB_ALIGNEDBASETRACKS  (
   ID                   NUMBER(9)                       not null,
   ID_MIPBASETRACK      NUMBER(9)                       not null,
   ID_RECONSTRUCTION    NUMBER(6)                       not null,
   ID_ALIGNEDZONE       NUMBER(7)                       not null,
   ID_VOLUMETRACK       NUMBER(6)                       not null,
   POSX                 NUMBER(9,2)                     not null,
   POSY                 NUMBER(9,2)                     not null,
   GRAINS               NUMBER(2)                       not null,
   AREASUM              NUMBER(4)                       not null,
   SLOPEX               NUMBER(5,4)                     not null,
   SLOPEY               NUMBER(5,4)                     not null,
   SIGMA                NUMBER(5,4)                     not null,
   constraint PK_TB_ALIGNEDBASETRACKS primary key (ID)
)
/


/*==============================================================*/
/* Table: TB_ALIGNEDZONES                                       */
/*==============================================================*/


create table TB_ALIGNEDZONES  (
   ID                   NUMBER(7)                       not null,
   ID_ZONE              NUMBER(7)                       not null,
   ID_RECONSTRUCTION    NUMBER(6)                       not null,
   PLATE                NUMBER(6)                       not null,
   MAPXX                NUMBER(8,7)                     not null,
   MAPXY                NUMBER(8,7)                     not null,
```

```
   MAPYX                NUMBER(8,7)                    not null,
   MAPYY                NUMBER(8,7)                    not null,
   MAPDX                NUMBER(9,2)                    not null,
   MAPDY                NUMBER(9,2)                    not null,
   MAPDZ                NUMBER(9,2)                    not null,
   SLOPEDSX             NUMBER(5,4)                    not null,
   SLOPEDSY             NUMBER(5,4)                    not null,
   SLOPEMSX             NUMBER(5,4)                    not null,
   SLOPEMSY             NUMBER(5,4)                    not null,
   POSZ                 NUMBER(9,2),
   constraint PK_TB_ALIGNEDZONES primary key (ID_ZONE, ID_RECONSTRUCTION),
   constraint AK_IX_ALIGNEDZONES_ID_TB_ALIGN unique (ID)
)
/


/*==============================================================*/
/* Table: TB_BATCHES                                            */
/*==============================================================*/


create table TB_BATCHES  (
   ID                   NUMBER(6)                      not null,
   ID_PLATE             NUMBER(6)                      not null,
   ID_MACHINE           NUMBER(4)                      not null,
   ID_PROGRAMSETTINGS   NUMBER(3)                      not null,
   ID_REQUESTER         NUMBER(4)                      not null,
   TEMPLATEMARKS        NUMBER(1)                      not null,
   REQUESTTIME          DATE                           not null,
   STARTTIME            DATE,
   ENDTIME              DATE,
   constraint PK_BATCHES primary key (ID)
)
/


/*==============================================================*/
/* Table: TB_BRICKS                                             */
/*==============================================================*/


create table TB_BRICKS  (
   ID                   NUMBER(6)                      not null,
   MINX                 NUMBER(9,2)                    not null,
   MAXX                 NUMBER(9,2)                    not null,
   MINY                 NUMBER(9,2)                    not null,
   MAXY                 NUMBER(9,2)                    not null,
   MINZ                 NUMBER(9,2)                    not null,
   MAXZ                 NUMBER(9,2)                    not null,
   constraint PK_BRICKS primary key (ID)
)
/


/*==============================================================*/
/* Table: TB_MACHINES                                           */
/*==============================================================*/


create table TB_MACHINES  (
   ID                   NUMBER(4)                      not null,
   NAME                 VARCHAR2(50)                   not null,
   ADDRESS              VARCHAR2(256)                  not null,
   ISSCANNINGSERVER     NUMBER(1)                      not null,
   ISBATCHSERVER        NUMBER(1)                      not null,
   ISDATAPROCESSINGSERVER NUMBER(1)                      not null,
   ISWEBSERVER          NUMBER(1)                      not null,
   ISDATABASESERVER     NUMBER(1)                      not null,
   ISLOCALMASTERCONTROLLER NUMBER(1)                      not null,
   ID_SITE              NUMBER(2)                      not null,
   constraint PK_MACHINES primary key (NAME),
   constraint IX_MACHINES_ADDRESS unique (ADDRESS),
   constraint IX_MACHINES_ID unique (ID),
   constraint CN_MACHINES check ((ISLOCALMASTERCONTROLLER = 0 OR ISLOCALMASTERCONTROLLER = 1) AND (ISSCANNINGSERVER = 0
OR ISSCANNINGSERVER = 1) AND (ISBATCHSERVER = 0 OR ISBATCHSERVER = 1) AND (ISDATAPROCESSINGSERVER = 0 OR
ISDATAPROCESSINGSERVER = 1) AND (ISDATABASESERVER = 0 OR ISDATABASESERVER = 1) AND (ISWEBSERVER = 0 OR ISWEBSERVER = 1)
AND (ISSCANNINGSERVER + ISBATCHSERVER + ISDATAPROCESSINGSERVER + ISDATABASESERVER + ISWEBSERVER +
ISLOCALMASTERCONTROLLER > 0))
)
/


/*==============================================================*/
/* Table: TB_MARKS                                              */
/*==============================================================*/


create table TB_MARKS  (
   ID                   NUMBER(4)                      not null,
   ID_PLATE             NUMBER(6)                      not null,
   ID_TEMPLATEMARK      NUMBER(6)                      not null,
   POSX                 NUMBER(8,2)                    not null,
   POSY                 NUMBER(8,2)                    not null,
   constraint PK_MARKS primary key (ID_PLATE),
   constraint IX_MARKS_ID unique (ID)
)
/


/*==============================================================*/
/* Table: TB_MIPBASETRACKS                                      */
/*==============================================================*/


create table TB_MIPBASETRACKS  (
   ID                   NUMBER(9)                      not null,
   ID_ZONE              NUMBER(7)                      not null,
```

```
   POSX                 NUMBER(9,2)                    not null,
   POSY                 NUMBER(9,2)                    not null,
   GRAINS               NUMBER(2)                      not null,
   AREASUM              NUMBER(4)                      not null,
   SLOPEX               NUMBER(5,4)                    not null,
   SLOPEY               NUMBER(5,4)                    not null,
   SIGMA                NUMBER(5,4)                    not null,
   ID_DOWNTRACK         NUMBER(9)                      not null,
   ID_UPTRACK           NUMBER(9)                      not null,
   constraint PK_TB_MIPBASETRACKS primary key (ID)
)
/


/*==============================================================*/
/* Table: TB_MIPEMULSIONTRACKS                                  */
/*==============================================================*/


create table TB_MIPEMULSIONTRACKS  (
   ID                   NUMBER(9)                      not null,
   ID_ZONE              NUMBER(7),
   SIDE                 NUMBER(1)                      not null,
   POSX                 NUMBER(9,2)                    not null,
   POSY                 NUMBER(9,2)                    not null,
   GRAINS               NUMBER(2)                      not null,
   AREASUM              NUMBER(4)                      not null,
   SLOPEX               NUMBER(5,4)                    not null,
   SLOPEY               NUMBER(5,4)                    not null,
   SIGMA                NUMBER(5,4)                    not null,
   constraint PK_TB_MIPEMULSIONTRACKS primary key (ID),
   constraint CKT_TB_MIPEMULSIONTRACKS check (SIDE = 1 OR SIDE = 2)
)
/


/*==============================================================*/
/* Table: TB_OPERATIONS                                         */
/*==============================================================*/


create table TB_OPERATIONS  (
   ID                   NUMBER(2)                      not null,
   NAME                 VARCHAR2(50)                   not null,
   ISSCANNING           NUMBER(1)                      not null,
   ISTRANSPORT          NUMBER(1)                      not null,
   constraint PK_OPERATIONS primary key (NAME),
   constraint IX_OPERATIONS_ID unique (ID)
)
/


/*==============================================================*/
/* Table: TB_PHYSICSIDS                                         */
/*==============================================================*/


create table TB_PHYSICSIDS  (
   ID                   NUMBER(2)                      not null,
   DESCRIPTION          VARCHAR2(32)                   not null,
   constraint PK_TB_PHYSICSIDS primary key (DESCRIPTION),
   constraint AK_IX_PHYSICSIDS_ID_TB_PHYSI unique (ID)
)
/


/*==============================================================*/
/* Table: TB_PLATES                                             */
/*==============================================================*/


create table TB_PLATES  (
   ID                   NUMBER(6)                      not null,
   ID_BRICK             NUMBER(6)                      not null,
   PLATE                NUMBER(2)                      not null,
   CALIBRATION          NUMBER(1)                      not null,
   Z                    NUMBER(9,2)                    not null,
   ISCS                 NUMBER(1)                      not null,
   EXPOSURESTART        DATE                           not null,
   EXPOSUREFINISH       DATE                           not null,
   MAPXX                NUMBER(8,7)                    not null,
   MAPXY                NUMBER(8,7)                    not null,
   MAPYX                NUMBER(8,7)                    not null,
   MAPYY                NUMBER(8,7)                    not null,
   MAPDX                NUMBER(9,2)                    not null,
   MAPDY                NUMBER(9,2)                    not null,
   constraint PK_PLATES primary key (ID_BRICK, PLATE, ISCS),
   constraint IX_PLATES unique (ID),
   constraint CN_PLATES check ((EXPOSURESTART < EXPOSUREFINISH) AND ((ISCS = 1 AND PLATE > 0) OR (ISCS = 0 AND PLATE >=
1 AND PLATE <= 56)))
)
/


/*==============================================================*/
/* Table: TB_PREDICTEDTRACKS                                    */
/*==============================================================*/


create table TB_PREDICTEDTRACKS  (
   ID                   NUMBER(8)                      not null,
   ID_PREDICTION        NUMBER(8)                      not null,
   ID_TRACK             NUMBER(2)                      not null,
   ID_PHYSICS           NUMBER(2)                      not null,
   POSX                 NUMBER(8,2)                    not null,
   POSY                 NUMBER(8,2)                    not null,
   POSZ                 NUMBER(8,2)                    not null,
```

```
   SLOPEX               NUMBER(5,4)                    not null,
   SLOPEY               NUMBER(5,4)                    not null,
   CHI2                 NUMBER(7,6)                    not null,
   constraint PK_TB_PREDICTEDTRACKS primary key (ID_PREDICTION, ID_TRACK),
   constraint AK_IX_PREDICTEDTRACKS_TB_PREDI unique (ID)
)
/


/*==============================================================*/
/* Table: TB_PREDICTIONS                                        */
/*==============================================================*/


create table TB_PREDICTIONS  (
   ID                   NUMBER(8)                      not null,
   RUN                  NUMBER(6)                      not null,
   EVENT                NUMBER(6)                      not null,
   X                    NUMBER(8,2)                    not null,
   Y                    NUMBER(8,2)                    not null,
   Z                    NUMBER(8,2)                    not null,
   constraint PK_TB_PREDICTIONS primary key (RUN, EVENT),
   constraint AK_IX_PREDICTIONS_ID_TB_PREDI unique (ID)
)
/


/*==============================================================*/
/* Table: TB_PRIVILEGES                                         */
/*==============================================================*/


create table TB_PRIVILEGES  (
   ID_USER              NUMBER(4)                      not null,
   ID_SITE              NUMBER(2)                      not null,
   REQUESTSCAN          NUMBER(1)                      not null
         constraint CKC_REQUESTSCAN_TB_PRIVI check (REQUESTSCAN between 0 and 1),
   REQUESTWEBANALYSIS   NUMBER(1)                      not null
         constraint CKC_REQUESTWEBANALYSI_TB_PRIVI check (REQUESTWEBANALYSIS between 0 and 1),
   REQUESTDATAPROCESSING NUMBER(1)                      not null
         constraint CKC_REQUESTDATAPROCES_TB_PRIVI check (REQUESTDATAPROCESSING between 0 and 1),
   REQUESTDATADOWNLOAD  NUMBER(1)                      not null
         constraint CKC_REQUESTDATADOWNLO_TB_PRIVI check (REQUESTDATADOWNLOAD between 0 and 1),
   REQUESTPROCESSSTARTUP NUMBER(1)                      not null
         constraint CKC_REQUESTPROCESSSTA_TB_PRIVI check (REQUESTPROCESSSTARTUP between 0 and 1),
   constraint PK_TB_PRIVILEGES primary key (ID_USER, ID_SITE)
)
/


/*==============================================================*/
/* Table: TB_PROCESSES                                          */
/*==============================================================*/


create table TB_PROCESSES  (
   ID                   NUMBER(6)                      not null,
   ID_BRICK             NUMBER(6)                      not null,
   STARTTIME            DATE                           not null,
   FINISHTIME           DATE,
   constraint PK_PROCESSES primary key (ID),
   constraint CN_PROCESSES check (FINISHTIME IS NULL OR (STARTTIME < FINISHTIME))
)
/


/*==============================================================*/
/* Table: TB_PROCSTARTUPSELECTIONS                              */
/*==============================================================*/


create table TB_PROCSTARTUPSELECTIONS  (
   ID_PROCESS           NUMBER(8)                      not null,
   ID_PREDICTEDTRACK    NUMBER(8)                      not null,
   ID_SELECTION         NUMBER(3)                      not null,
   constraint PK_TB_PROCSTARTUPSELECTIONS primary key (ID_PROCESS, ID_PREDICTEDTRACK, ID_SELECTION)
)
/


/*==============================================================*/
/* Table: TB_PROC_OPERATIONS                                    */
/*==============================================================*/


create table TB_PROC_OPERATIONS  (
   ID                   NUMBER(6)                      not null,
   ID_PROCESS           NUMBER(6)                      not null,
   ID_OPERATION         NUMBER(2)                      not null,
   ID_MACHINE           NUMBER(4)                      not null,
   STARTTIME            DATE                           not null,
   FINISHTIME           DATE,
   constraint PK_TB_PROC_OPERATIONS primary key (ID_PROCESS, ID_OPERATION),
   constraint AK_IX_PROCOPERATIONS__TB_PROC_ unique (ID)
)
/


/*==============================================================*/
/* Table: TB_PROGRAMSETTINGS                                    */
/*==============================================================*/


create table TB_PROGRAMSETTINGS  (
   ID                   NUMBER(3)                      not null,
   MODULENAME           VARCHAR2(256)                  not null,
   MODULEIDENTITY       VARCHAR2(256)                  not null,
```

```
   SETTINGS             CLOB                           not null,
   ID_AUTHOR            NUMBER(4),
   constraint PK_TB_PROGRAMSETTINGS primary key (MODULENAME, MODULEIDENTITY),
   constraint AK_IX_RECONSTRUCTIONP_TB_PROGR unique (ID)
)
/


/*==============================================================*/
/* Table: TB_RECONSTRUCTIONS                                    */
/*==============================================================*/


create table TB_RECONSTRUCTIONS  (
   ID                   NUMBER(6)                      not null,
   ID_RECONSTRUCTOR     NUMBER(4)                      not null,
   ISCOMPLETE           NUMBER(1)                      not null,
   ID_PROGRAMSETTINGS   NUMBER(3)                      not null,
   ID_PROCESSOPERATION  NUMBER(6)                      not null,
   constraint PK_TB_RECONSTRUCTIONS primary key (ID),
   constraint CN_TB_RECONSTRUCTIONS check (ISCOMPLETE = 0 OR ISCOMPLETE = 1)
)
/


/*==============================================================*/
/* Table: TB_SELECTIONS                                         */
/*==============================================================*/


create table TB_SELECTIONS  (
   ID                   NUMBER(3)                      not null,
   ID_OPERATION         NUMBER(2)                      not null,
   ID_SELECTOR          NUMBER(4)                      not null,
   SELECTIONTEXT        CLOB                           not null,
   SELECTIONDATE        DATE                           not null,
   DESCRIPTION          VARCHAR2(1024)                 not null,
   constraint PK_TB_SELECTIONS primary key (ID_OPERATION, ID_SELECTOR, SELECTIONDATE, DESCRIPTION),
   constraint AK_IX_SELECTIONS_ID_TB_SELEC unique (ID)
)
/


/*==============================================================*/
/* Table: TB_SITES                                              */
/*==============================================================*/


create table TB_SITES  (
   ID                   NUMBER(2)                      not null,
   NAME                 VARCHAR2(32)                   not null,
   LATITUDE             NUMBER(5,2)                    not null,
   LONGITUDE            NUMBER(5,2)                    not null,
   LOCALTIMEFUSE        NUMBER(1)                      not null,
   constraint PK_SITES primary key (NAME),
   constraint IX_SITES_ID unique (ID)
)
/


/*==============================================================*/
/* Table: TB_TEMPLATEMARKSETS                                   */
/*==============================================================*/


create table TB_TEMPLATEMARKSETS  (
   ID                   NUMBER(6)                      not null,
   ID_BRICK             NUMBER(6)                      not null,
   ID_MARK              NUMBER(2)                      not null,
   POSX                 NUMBER(9,2)                    not null,
   POSY                 NUMBER(9,2)                    not null,
   MARKROW              NUMBER(1)                      not null,
   MARKCOL              NUMBER(1)                      not null,
   SHAPE                NUMBER(1)                      not null,
   constraint PK_TEMPLATEMARKSETS primary key (ID_BRICK, ID_MARK, MARKROW, MARKCOL),
   constraint IX_TEMPLATEMARKSETS_ID unique (ID)
)
/


/*==============================================================*/
/* Table: TB_USERS                                              */
/*==============================================================*/


create table TB_USERS  (
   ID                   NUMBER(4)                      not null,
   USERNAME             VARCHAR2(50)                   not null,
   PWD                  VARCHAR2(50)                   not null,
   NAME                 VARCHAR2(50)                   not null,
   SURNAME              VARCHAR2(50)                   not null,
   INSTITUTION          VARCHAR2(50)                   not null,
   ID_SITE              NUMBER(2)                      not null,
   EMAIL                VARCHAR2(256)                  not null,
   ADDRESS              VARCHAR2(256)                  not null,
   PHONE                VARCHAR2(32)                   not null,
   constraint PK_USERS primary key (NAME, SURNAME),
   constraint IX_USERS_ID unique (ID),
   constraint IX_USERS_USR unique (USERNAME)
)
/


/*==============================================================*/
/* Table: TB_VERTEXTYPES                                        */
/*==============================================================*/
```

```
create table TB_VERTEXTYPES  (
   ID                   NUMBER(2)                      not null,
   DESCRIPTION          VARCHAR2(32)                   not null,
   constraint PK_TB_VERTEXTYPES primary key (DESCRIPTION),
   constraint AK_IX_VERTEXTYPES_ID_TB_VERTE unique (ID)
)
/


/*==============================================================*/
/* Table: TB_VERTICES                                           */
/*==============================================================*/


create table TB_VERTICES  (
   ID                   NUMBER(7)                      not null,
   ID_RECONSTRUCTION    NUMBER(6)                      not null,
   POSX                 NUMBER(9,2)                    not null,
   POSY                 NUMBER(9,2)                    not null,
   POSZ                 NUMBER(9,2)                    not null,
   AVERAGEDISTANCE      NUMBER(7,4)                    not null,
   ID_VERTEXTYPE        NUMBER(2),
   constraint PK_TB_VERTICES primary key (ID)
)
/


/*==============================================================*/
/* Table: TB_VOLUMETRACKS                                       */
/*==============================================================*/


create table TB_VOLUMETRACKS  (
   ID                   NUMBER(6)                      not null,
   ID_RECONSTRUCTION    NUMBER(6)                      not null,
   POSX                 NUMBER(9,2)                    not null,
   POSY                 NUMBER(9,2)                    not null,
   POSZ                 NUMBER(9,2)                    not null,
   SLOPEX               NUMBER(5,4)                    not null,
   SLOPEY               NUMBER(5,4)                    not null,
   SIGMA                NUMBER(5,4)                    not null,
   DOWNZ                NUMBER(9,2)                    not null,
   UPZ                  NUMBER(9,2)                    not null,
   DOWNPOSX             NUMBER(9,2)                    not null,
   DOWNPOSY             NUMBER(9,2)                    not null,
   DOWNPOSZ             NUMBER(9,2)                    not null,
   DOWNSLOPEX           NUMBER(5,4)                    not null,
   DOWNSLOPEY           NUMBER(5,4)                    not null,
   UPPOSX               NUMBER(9,2)                    not null,
   UPPOSY               NUMBER(9,2)                    not null,
   UPPOSZ               NUMBER(9,2)                    not null,
   UPSLOPEX             NUMBER(5,4)                    not null,
   UPSLOPEY             NUMBER(5,4)                    not null,
   MOMENTUM             NUMBER(6,2),
   ENERGY               NUMBER(6,2),
   ID_PARTICLE          NUMBER(2),
   RECONSTRUCTIONNOTES  NUMBER(2)                      not null,
   ID_DOWNVERTEX        NUMBER(7)                      not null,
   ID_UPVERTEX          NUMBER(7)                      not null,
   constraint PK_TB_VOLUMETRACKS primary key (ID)
)
/


/*==============================================================*/
/* Table: TB_VTXLOC_SELECTIONS                                  */
/*==============================================================*/


create table TB_VTXLOC_SELECTIONS  (
   ID                   NUMBER(6)                      not null,
   ID_PLATE             NUMBER(6)                      not null,
   ID_PROCESSOPERATION  NUMBER(6)                      not null,
   ID_SELECTOR          NUMBER(4)                      not null,
   ID_SELECTIONTEXT     NUMBER(3)                      not null,
   constraint PK_TB_VTXLOC_SELECTIONS primary key (ID_PLATE, ID_PROCESSOPERATION),
   constraint AK_IX_VTXLOC_SELECTIO_TB_VTXLO unique (ID)
)
/


/*==============================================================*/
/* Table: TB_VTXLOC_SELECTION_REASONS                           */
/*==============================================================*/


create table TB_VTXLOC_SELECTION_REASONS  (
   ID                   NUMBER(1)                      not null,
   DESCRIPTION          VARCHAR2(256)                  not null,
   constraint PK_TB_VTXLOC_SELECTION_REASONS primary key (DESCRIPTION),
   constraint AK_IX_VTXLOC_SELECTION_REASONS unique (ID)
)
/


/*==============================================================*/
/* Table: TB_VTXLOC_SELTRACKS                                   */
/*==============================================================*/


create table TB_VTXLOC_SELTRACKS  (
   ID                   NUMBER(9)                      not null,
   ID_VTXLOC_SEL        NUMBER(6)                      not null,
   ID_BASETRACK         NUMBER(9)                      not null,
   ID_MOTHERTRACK       NUMBER(9),
   ID_REASON            NUMBER(1)                      not null,
```

```
   constraint PK_TB_VTXLOC_SELTRACKS primary key (ID_VTXLOC_SEL, ID_BASETRACK),
   constraint AK_IX_VTXLOC_SELTRACK_TB_VTXLO unique (ID)
)
/


/*==============================================================*/
/* Table: TB_ZONES                                              */
/*==============================================================*/


create table TB_ZONES  (
   ID                   NUMBER(7)                      not null,
   ID_BATCH             NUMBER(6)                      not null,
   MINX                 NUMBER(9,2)                    not null,
   MAXX                 NUMBER(9,2)                    not null,
   MINY                 NUMBER(9,2)                    not null,
   MAXY                 NUMBER(9,2)                    not null,
   RAWDATAPATH          VARCHAR2(256)                  not null,
   STARTTIME            DATE                           not null,
   ENDTIME              DATE,
   constraint PK_TB_ZONES primary key (ID)
)
/


alter table TB_ALIGNEDBASETRACKS
   add constraint FK_ALIGNEDBASE_MIPBASE foreign key (ID_MIPBASETRACK)
      references TB_MIPBASETRACKS (ID)
/


alter table TB_ALIGNEDBASETRACKS
   add constraint FK_RECON_ALIGNEDTRACKS foreign key (ID_RECONSTRUCTION)
      references TB_RECONSTRUCTIONS (ID)
/


alter table TB_ALIGNEDBASETRACKS
   add constraint FK_ALIGN_ZONESTRACKS foreign key (ID_ALIGNEDZONE)
      references TB_ALIGNEDZONES (ID)
/


alter table TB_ALIGNEDBASETRACKS
   add constraint FK_VOLTKS_ALIGNBASE foreign key (ID_VOLUMETRACK)
      references TB_VOLUMETRACKS (ID)
/


alter table TB_ALIGNEDZONES
   add constraint FK_RECON_ALIGNEDZONE foreign key (ID_RECONSTRUCTION)
      references TB_RECONSTRUCTIONS (ID)
/


alter table TB_ALIGNEDZONES
   add constraint FK_ZONES_ALIGNEDZONES foreign key (ID_ZONE)
      references TB_ZONES (ID)
/


alter table TB_BATCHES
   add constraint FK_BATCHES_MACHINES foreign key (ID_MACHINE)
      references TB_MACHINES (ID) not deferrable
/


alter table TB_BATCHES
   add constraint FK_BATCHES_PLATES foreign key (ID_PLATE)
      references TB_PLATES (ID)
/


alter table TB_BATCHES
   add constraint FK_BATCHES_REQUESTER foreign key (ID_REQUESTER)
      references TB_USERS (ID) not deferrable
/


alter table TB_BATCHES
   add constraint FK_PROGSETTING_BATCHES foreign key (ID_PROGRAMSETTINGS)
      references TB_PROGRAMSETTINGS (ID)
/


alter table TB_MACHINES
   add constraint FK_MACHINES_SITE foreign key (ID_SITE)
      references TB_SITES (ID) not deferrable
/


alter table TB_MARKS
   add constraint FK_MARKS_PLATE foreign key (ID_PLATE)
      references TB_PLATES (ID) not deferrable
/


alter table TB_MARKS
   add constraint FK_MARKS_TEMPLATEMARK foreign key (ID_TEMPLATEMARK)
      references TB_TEMPLATEMARKSETS (ID) not deferrable
/


alter table TB_MIPBASETRACKS
   add constraint FK_BASE_DOWNTRACK foreign key (ID_DOWNTRACK)
      references TB_MIPEMULSIONTRACKS (ID)
```

```
/


alter table TB_MIPBASETRACKS
   add constraint FK_BASE_UPTRACK foreign key (ID_UPTRACK)
      references TB_MIPEMULSIONTRACKS (ID)
/


alter table TB_MIPBASETRACKS
   add constraint FK_ZONES_MIPBASE foreign key (ID_ZONE)
      references TB_ZONES (ID)
/


alter table TB_MIPEMULSIONTRACKS
   add constraint FK_ZONES_MIPEMU foreign key (ID_ZONE)
      references TB_ZONES (ID)
/


alter table TB_PLATES
   add constraint FK_PLATES_BRICK foreign key (ID_BRICK)
      references TB_BRICKS (ID) not deferrable
/


alter table TB_PREDICTEDTRACKS
   add constraint FK_PHYSIDS_PREDTRACKS foreign key (ID_PHYSICS)
      references TB_PHYSICSIDS (ID)
/


alter table TB_PREDICTEDTRACKS
   add constraint FK_PREDICTIONS_PREDTRACKS foreign key (ID_PREDICTION)
      references TB_PREDICTIONS (ID)
/


alter table TB_PRIVILEGES
   add constraint FK_SITES_PRIVILEGES foreign key (ID_SITE)
      references TB_SITES (ID)
/


alter table TB_PRIVILEGES
   add constraint FK_USERS_PRIVILEGES foreign key (ID_USER)
      references TB_USERS (ID)
/


alter table TB_PROCESSES
   add constraint FK_BRICKS_PROCESSES foreign key (ID_BRICK)
      references TB_BRICKS (ID)
/


alter table TB_PROCSTARTUPSELECTIONS
   add constraint FK_PROCSTART_PREDTKS foreign key (ID_PREDICTEDTRACK)
      references TB_PREDICTEDTRACKS (ID)
/


alter table TB_PROCSTARTUPSELECTIONS
   add constraint FK_PROCSTART_SELECTION foreign key (ID_SELECTION)
      references TB_SELECTIONS (ID)
/


alter table TB_PROCSTARTUPSELECTIONS
   add constraint FK_PROCSTARTUPS_PROCS foreign key (ID_PROCESS)
      references TB_PROCESSES (ID)
/


alter table TB_PROC_OPERATIONS
   add constraint FK_MACHINES_PROCOPERATIONS foreign key (ID_MACHINE)
      references TB_MACHINES (ID)
/


alter table TB_PROC_OPERATIONS
   add constraint FK_OPERATIONS_PROCOPERATIONS foreign key (ID_OPERATION)
      references TB_OPERATIONS (ID)
/


alter table TB_PROC_OPERATIONS
   add constraint FK_PROCESSES_PROCOPS foreign key (ID_PROCESS)
      references TB_PROCESSES (ID)
/


alter table TB_PROGRAMSETTINGS
   add constraint FK_USERS_PROGSETTINGS foreign key (ID_AUTHOR)
      references TB_USERS (ID)
/


alter table TB_RECONSTRUCTIONS
   add constraint FK_PROCOPERA_RECON foreign key (ID_PROCESSOPERATION)
      references TB_PROC_OPERATIONS (ID)
/


alter table TB_RECONSTRUCTIONS
   add constraint FK_PROGSETTING_RECON foreign key (ID_PROGRAMSETTINGS)
```

```
      references TB_PROGRAMSETTINGS (ID)
/


alter table TB_RECONSTRUCTIONS
   add constraint FK_RECON_USERS foreign key (ID_RECONSTRUCTOR)
      references TB_USERS (ID)
/


alter table TB_SELECTIONS
   add constraint FK_SELECTIONS_OPS foreign key (ID_OPERATION)
      references TB_OPERATIONS (ID)
/


alter table TB_SELECTIONS
   add constraint FK_SELECTIONS_USERS foreign key (ID_SELECTOR)
      references TB_USERS (ID)
/


alter table TB_TEMPLATEMARKSETS
   add constraint FK_TEMPLATEMARKSETS foreign key (ID_BRICK)
      references TB_BRICKS (ID) not deferrable
/


alter table TB_USERS
   add constraint FK_SITES_USERS foreign key (ID_SITE)
      references TB_SITES (ID)
/


alter table TB_VERTICES
   add constraint FK_RECON_VERTICES foreign key (ID_RECONSTRUCTION)
      references TB_RECONSTRUCTIONS (ID)
/


alter table TB_VERTICES
   add constraint FK_VERTEXTYPES_VERTICES foreign key (ID_VERTEXTYPE)
      references TB_VERTEXTYPES (ID)
/


alter table TB_VOLUMETRACKS
   add constraint FK_VOLTKS_DOWNVTX foreign key (ID_DOWNVERTEX)
      references TB_VERTICES (ID)
/


alter table TB_VOLUMETRACKS
   add constraint FK_PHYSIDS_VOLTKS foreign key (ID_PARTICLE)
      references TB_PHYSICSIDS (ID)
/


alter table TB_VOLUMETRACKS
   add constraint FK_RECON_VOLTKS foreign key (ID_RECONSTRUCTION)
      references TB_RECONSTRUCTIONS (ID)
/


alter table TB_VOLUMETRACKS
   add constraint FK_VOLTKS_UPVTX foreign key (ID_UPVERTEX)
      references TB_VERTICES (ID)
/


alter table TB_VTXLOC_SELECTIONS
   add constraint FK_PROCOPS_VTXLOCS foreign key (ID_PROCESSOPERATION)
      references TB_PROC_OPERATIONS (ID)
/


alter table TB_VTXLOC_SELECTIONS
   add constraint FK_VTXLOC_PLATES_SELS foreign key (ID_PLATE)
      references TB_PLATES (ID)
/


alter table TB_VTXLOC_SELECTIONS
   add constraint FK_VTXLOC_SEL_VSEL foreign key (ID_SELECTIONTEXT)
      references TB_SELECTIONS (ID)
/


alter table TB_VTXLOC_SELECTIONS
   add constraint FK_VTXLOC_USERS_VSELS foreign key (ID_SELECTOR)
      references TB_USERS (ID)
/


alter table TB_VTXLOC_SELTRACKS
   add constraint FK_VTXLOC_SELRSN_SELTKS foreign key (ID_REASON)
      references TB_VTXLOC_SELECTION_REASONS (ID)
/


alter table TB_VTXLOC_SELTRACKS
   add constraint FK_VTXLOC_MIPBASE_SELTKS foreign key (ID_BASETRACK)
      references TB_MIPBASETRACKS (ID)
/


alter table TB_VTXLOC_SELTRACKS
```

```
   add constraint FK_VTXLOC_VSEL_SELTKS foreign key (ID_VTXLOC_SEL)
      references TB_VTXLOC_SELECTIONS (ID)
/


alter table TB_ZONES
   add constraint FK_BATCHES_ZONES foreign key (ID_BATCH)
      references TB_BATCHES (ID)
/
```

Integrity package

```
/*==============================================================*/
/* Database name:  OPERA_EUROPEAN_EMULSION_SCANNING_DB          */
/* DBMS name:      ORACLE Version 9i                            */
/* Created on:     21-Feb-03 11:27:28 AM                        */
/*==============================================================*/



-- Integrity package declaration
create or replace package IntegrityPackage AS
 procedure InitNestLevel;
 function GetNestLevel return number;
 procedure NextNestLevel;
 procedure PreviousNestLevel;
 end IntegrityPackage;
/

-- Integrity package definition
create or replace package body IntegrityPackage AS
 NestLevel number;

-- Procedure to initialize the trigger nest level
 procedure InitNestLevel is
 begin
 NestLevel := 0;
 end;


-- Function to return the trigger nest level
 function GetNestLevel return number is
 begin
 if NestLevel is null then
     NestLevel := 0;
 end if;
 return(NestLevel);
 end;

-- Procedure to increase the trigger nest level
 procedure NextNestLevel is
 begin
 if NestLevel is null then
     NestLevel := 0;
 end if;
 NestLevel := NestLevel + 1;
 end;

-- Procedure to decrease the trigger nest level
 procedure PreviousNestLevel is
 begin
 NestLevel := NestLevel - 1;
 end;

 end IntegrityPackage;
/


drop trigger TIB_TB_ALIGNEDBASETRACKS
/


drop trigger TIB_TB_ALIGNEDZONES
/


drop trigger TIB_TB_BATCHES
/


drop trigger TIB_TB_BRICKS
/


drop trigger TIB_TB_MACHINES
/


drop trigger TIB_TB_MARKS
/


drop trigger TIB_TB_MIPBASETRACKS
/


drop trigger TIB_TB_MIPEMULSIONTRACKS
/


drop trigger TIB_TB_PLATES
/
```

```
drop trigger TIB_TB_PREDICTEDTRACKS
/


drop trigger TIB_TB_PREDICTIONS
/


drop trigger TIB_TB_PROCESSES
/


drop trigger TIB_TB_PROC_OPERATIONS
/


drop trigger TIB_TB_PROGRAMSETTINGS
/


drop trigger TIB_TB_RECONSTRUCTIONS
/


drop trigger TIB_TB_SELECTIONS
/


drop trigger TIB_TB_SITES
/


drop trigger TIB_TB_TEMPLATEMARKSETS
/


drop trigger TIB_TB_USERS
/


drop trigger TIB_TB_VERTICES
/


drop trigger TIB_TB_VOLUMETRACKS
/


drop trigger TIB_TB_VTXLOC_SELECTIONS
/


drop trigger TIB_TB_VTXLOC_SELTRACKS
/


drop trigger TIB_TB_ZONES
/



--  Before insert trigger "TIB_TB_ALIGNEDBASETRACKS" for table "TB_ALIGNEDBASETRACKS"
create trigger TIB_TB_ALIGNEDBASETRACKS before insert
on TB_ALIGNEDBASETRACKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_ALIGNEDBASETRACKS
    select SEQ_ALIGNEDBASETRACKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_ALIGNEDZONES" for table "TB_ALIGNEDZONES"
create trigger TIB_TB_ALIGNEDZONES before insert
on TB_ALIGNEDZONES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_ALIGNEDZONES
    select SEQ_ALIGNEDZONES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_BATCHES" for table "TB_BATCHES"
```

```
create trigger TIB_TB_BATCHES before insert
on TB_BATCHES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_BATCHES
    select SEQ_BATCHES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_BRICKS" for table "TB_BRICKS"
create trigger TIB_TB_BRICKS before insert
on TB_BRICKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_BRICKS
    select SEQ_BRICKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_MACHINES" for table "TB_MACHINES"
create trigger TIB_TB_MACHINES before insert
on TB_MACHINES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_MACHINES
    select SEQ_MACHINES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_MARKS" for table "TB_MARKS"
create trigger TIB_TB_MARKS before insert
on TB_MARKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_MARKS
    select SEQ_MARKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_MIPBASETRACKS" for table "TB_MIPBASETRACKS"
create trigger TIB_TB_MIPBASETRACKS before insert
on TB_MIPBASETRACKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_MIPBASETRACKS
    select SEQ_MIPBASETRACKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/
```

```
--  Before insert trigger "TIB_TB_MIPEMULSIONTRACKS" for table "TB_MIPEMULSIONTRACKS"
create trigger TIB_TB_MIPEMULSIONTRACKS before insert
on TB_MIPEMULSIONTRACKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_MIPEMULSIONTRACKS
    select SEQ_MIPEMULSIONTRACKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_PLATES" for table "TB_PLATES"
create trigger TIB_TB_PLATES before insert
on TB_PLATES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_PLATES
    select SEQ_PLATES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_PREDICTEDTRACKS" for table "TB_PREDICTEDTRACKS"
create trigger TIB_TB_PREDICTEDTRACKS before insert
on TB_PREDICTEDTRACKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_PREDICTEDTRACKS
    select SEQ_PREDICTEDTRACKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_PREDICTIONS" for table "TB_PREDICTIONS"
create trigger TIB_TB_PREDICTIONS before insert
on TB_PREDICTIONS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_PREDICTIONS
    select SEQ_PREDICTIONS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_PROCESSES" for table "TB_PROCESSES"
create trigger TIB_TB_PROCESSES before insert
on TB_PROCESSES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_PROCESSES
    select SEQ_PROCESSES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
```

```
        raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_PROC_OPERATIONS" for table "TB_PROC_OPERATIONS"
create trigger TIB_TB_PROC_OPERATIONS before insert
on TB_PROC_OPERATIONS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_PROC_OPERATIONS
    select SEQ_PROC_OPERATIONS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_PROGRAMSETTINGS" for table "TB_PROGRAMSETTINGS"
create trigger TIB_TB_PROGRAMSETTINGS before insert
on TB_PROGRAMSETTINGS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_PROGRAMSETTINGS
    select SEQ_PROGRAMSETTINGS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_RECONSTRUCTIONS" for table "TB_RECONSTRUCTIONS"
create trigger TIB_TB_RECONSTRUCTIONS before insert
on TB_RECONSTRUCTIONS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_RECONSTRUCTIONS
    select SEQ_RECONSTRUCTIONS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_SELECTIONS" for table "TB_SELECTIONS"
create trigger TIB_TB_SELECTIONS before insert
on TB_SELECTIONS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_SELECTIONS
    select SEQ_SELECTIONS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_SITES" for table "TB_SITES"
create trigger TIB_TB_SITES before insert
on TB_SITES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_SITES
    select SEQ_SITES.NEXTVAL INTO :new.ID from dual;
```

```
--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_TEMPLATEMARKSETS" for table "TB_TEMPLATEMARKSETS"
create trigger TIB_TB_TEMPLATEMARKSETS before insert
on TB_TEMPLATEMARKSETS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_TEMPLATEMARKSETS
    select SEQ_TEMPLATEMARKSETS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_USERS" for table "TB_USERS"
create trigger TIB_TB_USERS before insert
on TB_USERS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_USERS
    select SEQ_USERS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_VERTICES" for table "TB_VERTICES"
create trigger TIB_TB_VERTICES before insert
on TB_VERTICES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_VERTICES
    select SEQ_VERTICES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_VOLUMETRACKS" for table "TB_VOLUMETRACKS"
create trigger TIB_TB_VOLUMETRACKS before insert
on TB_VOLUMETRACKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_VOLUMETRACKS
    select SEQ_VOLUMETRACKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_VTXLOC_SELECTIONS" for table "TB_VTXLOC_SELECTIONS"
create trigger TIB_TB_VTXLOC_SELECTIONS before insert
on TB_VTXLOC_SELECTIONS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
```

```
    --  Column "ID" uses sequence SEQ_VTXLOC_SELECTIONS
    select SEQ_VTXLOC_SELECTIONS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_VTXLOC_SELTRACKS" for table "TB_VTXLOC_SELTRACKS"
create trigger TIB_TB_VTXLOC_SELTRACKS before insert
on TB_VTXLOC_SELTRACKS for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_VTXLOC_SELTRACKS
    select SEQ_VTXLOC_SELTRACKS.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/


--  Before insert trigger "TIB_TB_ZONES" for table "TB_ZONES"
create trigger TIB_TB_ZONES before insert
on TB_ZONES for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;

begin
    --  Column "ID" uses sequence SEQ_ZONES
    select SEQ_ZONES.NEXTVAL INTO :new.ID from dual;

--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/
```

Index