PS/CO/Min. 89-26
1.06.89

Real Time Database.

Presents : Jan Cuperus, G Daems, W. Heinze, M. Lelaizant, J. Lewis, F. di Nais, F. Perrollat, Ch Serre, G. Shering, Clh Sicard.-
InFo : Chefs de section Co : G.P. Benincasa, B. Kuiper, N. de Metz-Noblat

1/ Introduction : But → Discuter les differents possibles options pour une RT DB.-

JC : Résumé des differents options présentées dans le papier de JC
(voir Appendix de PS.CO.WP 89.015 du 22 Mai 89)

R/O Tables
R/O Dictionnary
R/O TORACL (essentiellement pour NODAL et assez lent....)[SPS]
R/O Table File System (Package specifique). [LEP]
R/W File System UNIX (+DBM) : par ex.
R/W CTree = compliqué et difficile à maintenir
SPS a pas mal de pb avec C-Tree.

RDB Management System = ORACLE.

JLW : → Personal Home made Julian's DB : written in P+ with simple basic command. To transport this JLW's DB on "C" : about 6 weeks / 2 months.-
- Fixed lenght records
- Single key commands
* This JLW's DB is actually used in the system by ≃15 AP (Nodal).-

JC → This DB needs to be translated in "C", but also some Tools for management.-

FP → What about Tools for Archiving ? (of the DB itself)
For ex to transfer data for offline analysis

FdM → Pour chacune de DB prévues il faudrait etudier chacune des fonctionalités de ce DB.

2/ Discussion :

GS → Il faut prendre beaucoup de précautions avant d'inventer qq chose d'autre qu'ORACLE.–

- Voir les exemples cités :
  - MIS
  - Accélérateurs (DB pour 1990's sera ORACLE)

* les avantages de n'utiliser qu'ORACLE sont très importants.
* les inconvénients d'inventer et de maintenir une Home made DB sont énormes.– (personnel, discussions, politiques, outils comparables, ect...).

FP → OK pour limiter le nbre de DBMS
MAis de quoi a-t-on besoin au niveau accélérateur ?
- R/O DB (très rapide)
- ORACLE
- On line R/W facilities

FdM → Il existe des limitations au niveau d'ORACLE.

GS : il existe une licence générale pour le site CERN avec possibilité d'utiliser ORACLE dans un ordinateur local.

FP : Quel est le temps d'accès nécessaire dans notre syst actuel ?
JC → actuellement 5 lect OB name par seconde
GS → ORACLE a un temps de réponse d'environ 5 secondes à partir d'un programme.–

JC Pour JC, soit on utilise un R/O Table simple avec des Keys simples, soit on fait le saut et on passe à ORACLE. Il ne doit pas y avoir de choses intermédiaires.–

* Liste des R/O Tables existant pour les AP.

EQINFO

ALARMES

WSET

SETUP

OBname ⟷ EM/FEC/EQNO

SOS + VIDEO

(ARBRES)

(ARCHIVES)

* JC pense que si on doit faire un effort pour apprendre à utiliser qqchose, il faut le faire pour qq chose qui existe et qui est maintenu correctement.-

* FP: RO DB pour les AP
  * Quelles sont les performances prévisibles si on utilise ORACLE pour ~~tous~~ les applications ?

    JC : l'appel à ORACLE sera tjs plus lent que l'appel à une table R/O.-

  * Que peut on faire si ORACLE n'est plus en liaison ?

    JC : on a besoin d'un ensemble qui soit en ligne d'une façon continu ; avec arrêt faisceau si link coupé.-
    → on peut réfléchir à la question, bien sur.-
    mais enfin laissons pour le moment.-

  * Si on a une table R/O comme source RT, la source primaire reste ORACLE.- Il faudrait donc un accès pour générer une "Flat Table".-

* GS = 2 questions =
  - Temp d'accès aux DB d'ORACLE ?
  - R/O de tables d'ORACLE. Comment générer/accéder ?-

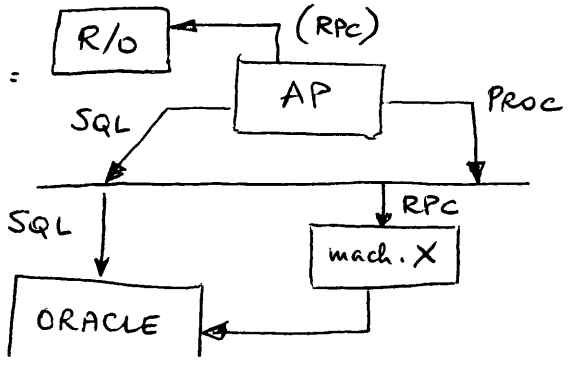
→ il existe des outils interface avec ORACLE commerciaux.-

JC : Voyons les façons d accéder =

a/ SQL

b) PROC. (càd RPC)

c) R/O Tables



3/ local R/O Tables.

- D'après les participants, on a besoin pour les AP d'un ensemble "R/O Tables".

* Donc

- Nécessité d'un mécanisme pour générer ces "R/O Tables" depuis ORACLE.
- Nécessité d'un mécanisme de lecture de ces "R/O Tables" depuis les AP.

* Justification d'une R/O Tables :
  - Temps d'accès faible.
  - Données stables et simples à accéder pendant une période donnée.
  - Simplicité des AP

CHS - Pourquoi n'envisage-t-on pas de prendre "R/O Table File System" du LEP ?

JC Effort de maintenance et d'apprentissage plus grand que ce que je fournis au PS/CO.
Beaucoup des arguments contre le ORACLE primitif n'existe plus maintenant.-

GS N'y-a-t-il pas plus de difficultés de prendre les petits systèmes développés à la maison que d'aller directement vers ORACLE ? (raisons psychologiques par ex.).

JC : la routinette ORACLE → R/O DataTables préserve la cohérence des infos, fait à chaque update !!

* les "R/O DT" seront stockés dans un server du réseau.

GS pour les requêtes de la maintenance (requêtes particulières), les accès seront faits vers ORACLE (câblage, localisation ...).

GD = il faut des outils pour lire et comparer les "R/O DT" qui sont stockés dans le server du réseau.-

JC = NON, il n'y aura rien !!

FdM : localement, une copie unique des infos paramétres, générée à partir d'ORACLE. Cette copie devrait être dans un server (avec un disque) particulier [accés NFS].

= Quelles méthodes de mise à jour ?
- Pas de vérification source/copie.
- On n'installe pas un élément en 5 minutes, donc pour le R/O DT on peut très bien attendre 5 min.

GD = Peut-on être sur de la coherence des multiples DT dans les AP par rapport à la copie "R/O DT" ?

JC = les données seront suffisamment génériques pour que l'on n'est pas besoin de particularités pour rendre efficace les différents AP.

FP = Problème de la performance est suffisamment important pour justifier une copie locale "R/O DT"

* Quelle est cette implémentation ?

ou = Dictionary File
ou = Table File System

= Quel est le côut de ce developpement ?

JC : Faible ....
Sinon nous allons vers ORACLE.

= Quelles sont les spécifications sur les méthodes d'accés ?

JC = Ref : RT Dictionary File Tables
J. Cuperus
DB - Article 4.2 -

* Conclusion pour le R/O DT.
- on n'échappe pas à une copie locale R/O DT ! (Efficacité, simplicité)
- JC presente les specifs de la méthode d'accés.
- utilisation de DBM ou utilisation de Dictionary Files (JC)

* Prochaine Réunion : Discussion sur "Online R/W Database"

REAL-TIME DICTIONARY TABLES

Authors: J. Cupérus
Written: 6 SEP 1988
Revised: 1 JUN 1988

Database Art.: 4.2
Section : RT-DB
Pages : 10

1. Introduction

The central source of our data is the ORACLE database, at present on an IBM mainframe. In principle, real-time programs could access the central database directly but, for the moment at least, it is not possible for us to make remote procedure calls to ORACLE. Even if this were possible, there is a timing problem: a relational database is quite efficient for responding to complicated queries for data sets but needs a minimum of time even for returning a single record and a typical display may need a large number of them, which can result in too long delays, especially in a multi-user database system where everybody has the same priority. The central system may also be down for maintenance at times unrelated to the operational needs of our control system. For these reasons, several of the ORACLE tables and views are downloaded as dictionary files on the TREES computer where they can be queried by simple remote procedure calls (RPCs) to routines which return a complete record when the user produces the key.

To make this service efficient, the records are ordered physically according to the key and the (short) index to the disk pages is permanently in memory. This has the advantage that any record can be can be retrieved with, at most, one disk access and none if the disk page is already in memory, which often happens when closely related records are successively requested.

The application program calls the appropriate procedure header with the correct record structure. In the server, a few lines of code relay the call from the table-specific header to a general access routine XFIND. This routine and associated data structures can give access to up to 8 dictionary tables. This limit is a consequence of the small size of the ND computer segments (24 kWords in this case). At the moment, two copies of the routines are implemented on different segments, giving access to up to 16 dictionary tables. Each table is implemented on a separate, permanently open, SINTRANIII file.

The acces routine has 8 slots. An initialisation routine DINIT allows to connect each slot to a specific dictionary table and to read its index into memory.

The RT database will be ported, as soon as possible on a ULTRIX server where it will exist in parallel with the ND version for a certain time. This is discussed at the end of the note.

2. Structure of the Dictionary Tables

The dictionary table starts with one or more index pages (of 1024 words), followed by data pages with records of information. If a record does not fit completely at the end of the page, the space is left blank and the record begins on the next page.

A record is composed of key fields in proper order (with the most significant key first), followed by data fields. The records are ordered according to the key fields, as if the key was composed of (16 bit) integers. This is done for efficiency and it means that the ordering is correct only for integer and string keys, but these are the only keys that matter in practice (a record is not normally identified with a real). Characters are packed, two per word. A string is always composed of an even number of characters and padded with blanks at the end.

The first positions of the first index page contain:

Pos. 0 : number of records in table
Pos. 1 : number of 16-bit words per record
Pos. 2 : number of records per page
Pos. 3 : offset of last records on page (in words, from start of page)
Pos. 4 : last page in file (from 0)
Pos. 5 : length of key in words
Pos. 6 : number of index pages
Pos. 7 : offset of last key on index (in words from start of file)
Pos. 8,9: spare positions

Starting at Pos. 10 follows a contiguous list of the keys of the last record of each page.

3. The RPC Headers

The RPC headers are defined in file <PRDEV>(DATA-BASE)NPL-DB-1ACCESS:SYMB for the first 8 tables and in file <PRDEV>(DATA-BASE)NPL-DB-2ACCESS:SYMB for the next 8 tables. These ICCI headers are the link between the outside world and the interface routine XFIND (see below). They specify the name of the routines and the number and kind of parameters. They check the array dimensions and other limits and adapt the outside request to the dictionary table structures. Several headers may refer to the same dictionary table and one header may combine calls to several dictionary routines. In general, however, the outside call mirrors directly all or part of the fields of a dictionary table. Appendix 1 gives an idea of the present headers. To have the latest version, refer to the original definitions on file <PRDEV>(DATA-BASE)DEF-DATABASE-FUN:SYMB. The definition file is written for PPLUS but, at present, most of the calls are from NODAL.

4. The DINIT and XFIND Routines

The source code for the initialisation routine DINIT and the access routine XFIND is on <PRDEV>(INFO)NPL-DB-DICSEARCH:SYMB.

Each of 8 buffers can be connected to a particular dictionary table. The binding to permanently open dictionary files is done with routine DINIT which has following integer parameters:

RO FCOUNT : number of file buffers used
RO FILENO[8] : array with file numbers of permanently open files

```
RO RECLEN[8] :  array with length of corresponding table records
WO XBUFNO    :  last processed file (XBUFNO=FCOUNT and XCOCO=0 if all OK)
WO XCOCO     :  complement code (see below)
```

DINIT is called, at initialisation, from the header routine DBIN1 for the first segment and DBIN2 for the second segment. These header routines specify the file numbers. The RECLEN array is redundant and serves to check whether the tables have the expected structure. The DBIN routines are called from SYS-GO when the TREES computer is initialised.

The other header routines communicate with the access routine XFIND. Here, communication is through DISP parameters:

```
RO XBUFNO   :  file buffer number 1..FCOUNT
RO XMODE    :  XMODE=0: get record with sequence number XRECNO
               XMODE=1: search for exact match with key
               XMODE=2: abbreviations are detected (for packed strings)
RO XRECNO   :  sequence number (from 1) of requested record (on input) or
               of returned record (on output)
RW XRECBUF  :  array for receiving the record; if XMODE is 1 or 2, then
               the key must be filled in before calling (padded with
               blanks if abbreviation); the complete record, as found in
               the table, is returned
WO XCOCO    :  complement code (see below);
```

5. Error Codes

The routines DINIT or XFIND may return the following complement codes. These codes may be transformed, in some cases, by the header routines:

```
XCOCO=0        :  requested record found or exact match found
     =1        :  unambiguous abbreviation match found
     =2        :  ambiguous abbreviation match found
     =3        :  not found and record with next higher key returned
     =4        :  not found and last record in the dictionary returned
     =10       :  file buffers not initialised
     =11..17   :  buffer initialisation errors (bad data file ?)
     =11,12,13 :  bad file heading ?
     =14,15    :  index too large ?
     =16       :  bad order in file ?
     =17       :  page keycheck failure
     =SF+&12000:  SF is SINTRAN III file error
```

6. Generation of Dictionary Tables

A dictionary file is generated by FORTRAN program RTDUMP. The program asks a number of questions which are reproduced below, followed by appropriate user input:

Give ORACLE username.tablename: PSC.ALMES

Key field list (1 or more lines) terminated by END: CATNUM/I2,MESNUM/I2,END

Field list (1 or more lines) terminated by END: MESSAGE/I2,END

ALMES is the name of the ORACLE table or view of which you want a dictionary version.

The lists are composed of entries, separated by commas and terminated by end. The list can be continued on several lines, provided each line exept the last ends with a comma.

Each entry is of format AAAA/Xn where AAAA is the ORACLE column name, X the column type (H=hex, I=integer, O=octal, B=binary, R=real, S=string) and n the length of the field (in bytes) in the dictionary table.

The dictionary is written to file: tablename.DUMP. This file is in ND format for integers, reals and characters.

When several dictionary tables must be generated, the program is called once for each one. These calls are done from file <IBM>(PSCO)RTDBGEN.EXEC (see article 4.1).

7. RT Database on the ULTRIX Computers

On the VAX computers, under ULTRIX, many of the space limitations of the ND version will not exist.

The file structure will be the same as on the ND version. The readout routines will be written in C. The parameters for the DINIT and XFIND routines will be the same but all the parameters will now be passed in the subroutine calls:

```
RTDBINIT(FCOUNT,FILENO_ARRAY,RECLEN_ARRAY,COCO_ARRAY)

RECFIND(BUFNO,MODE,RECNO,RECBUF,COCO)
```

All indexes are now grouped in the same buffer which can countain as many file references as necessary.

PS/CO/Min. 89-26

1.06.89

Real Time Database.

Presents : Jan Cuperus, G Daems, W. Heinze, M. Lelaizant, J. Lewis, F. di Maio, F. Perriollat, Ch Serre, G. Shering, Clt Sicard.

Info : Chefs de section Co. Gi. P. Bernincasa, B. Kuiper, N de Metz-Noblat

I N V I T A T I O N

REAL TIME DATABASE

Aux : voir ci-dessus

De : Ch. Serre

MERCREDI 7 JUIN 1989

PETITE SALLE DE CONFERENCE PS
(6-2-002)

À 14H00

Ch. Serre