ODB (Observational Database) and its usage at ECMWF

Anne Fouilloux

Abstract

ODB stands for Observational DataBase and has been developed at ECMWF since mid-1998 by Sami Saarinen. The main goal of ODB is to allow the storage and retrieval of large amounts of meteorological (numerical) data efficiently when used within IFS (Integrated Forecasting System). ODB has replaced the CMA (Central Memory Array) file structure (developed by Drasko Vasiljevic at ECMWF) with less disk space and memory demanding formats. Thanks to its ODB/SQL language, ODB enables serial or MPI-parallel data queries and also coordinates queries for data shuffling between MPI-tasks. ODB maintains high efficiency for storing and recovering data on any platforms (scalar or vector machines).

1 Observational usage over the past decades at ECMWF

One major progress made over the last two decades in Numerical Weather Prediction (NWP) can be attributed to the improved utilization of observations as shown on Figure 1. However this has only been possible thanks to the use of supercomputers as well as the development of efficient strategies to read/write/process these observations. ECMWF has always played a leading role in this area.

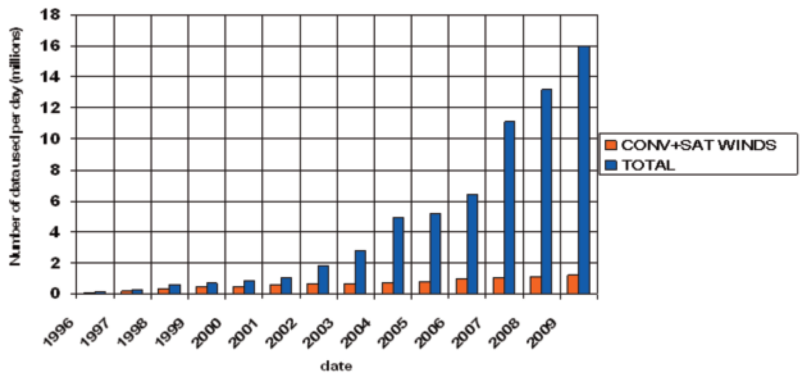
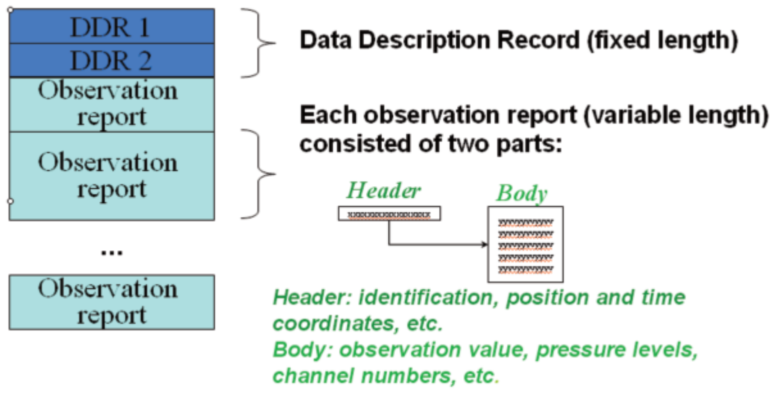


Fig. 1 Use of observational data at ECMWF since 1996.

2 Before ODB...

A first step toward an efficient strategy was achieved with the development of CMA (Central Memory Array) file structure. All observational data were packed report by report into a single IEEE 64 bit floating-point array. The file structure is a simple bit stream as shown on Figure 2 and is composed of two Data Description Records (DDR1 and DDR2) followed by a set of observation reports. The DDRs have a fixed length and are positioned at the beginning of the file. After the DDRs the observation reports follow, one after the other, until the end of the file. The observation reports are of variable length. Once read, CMA is kept in memory for a fast data access.



DDR 1
DDR 2
Observation report
Observation report
...
Observation report
Data Description Record (fixed length)
Each observation report (variable length) consisted of two parts:
Header
Body
Header: identification, position and time coordinates, etc.
Body: observation value, pressure levels, channel numbers, etc.

Fig. 2 CMA file structure.

3 Observational DataBase (ODB)

3.1 *What is ODB?*

With the introduction of 4D-Var in IFS and the ever growing number of satellite observations, there was a need for a new approach to store and access observational data. Sami Saarinen et al. suggested the use of relational database concepts for easier data selection and filtering: the ODB (Observational DataBase) software was born (mid-1998) and became operational in 2000.

ODB is an in-core hierarchical database (these two features were inherited from CMA format) with a data definition and query language: ODB/SQL - language (subset of ANSI SQL). ODB offers a parallel Fortran 90 interface to enable MPI-parallel data queries, but also to coordinate queries for data shuffling between MPI-tasks. Post-processing tools (odbsql, odbdiff, etc.) have been developed to cover user needs.

3.2 *ODB limitations*

ODB cannot restrict the user's ability to retrieve, add or modify data by protecting unauthorized access. However, with Fortran 90 access layer, an ODB database can be opened in READONLY-mode. An ODB database cannot be shared by concurrent users without interfering each other. This is only possible for READONLY-databases. In addition, there is no protection from corruption in the case an ODB database is inconsistently updated or a system failure.

3.3 *ODB/SQL language*

ODB/SQL is a language to manage ODB observational databases. It is a minimum subset of international standard SQL (Structured Query Language) used to manipulate genuine relational databases. Except for the creation of a database where a Fortran program is usually necessary, ODB/SQL is used in an interactive way via ODB-tools.

ODB/SQL is currently capable of handling the following tasks:

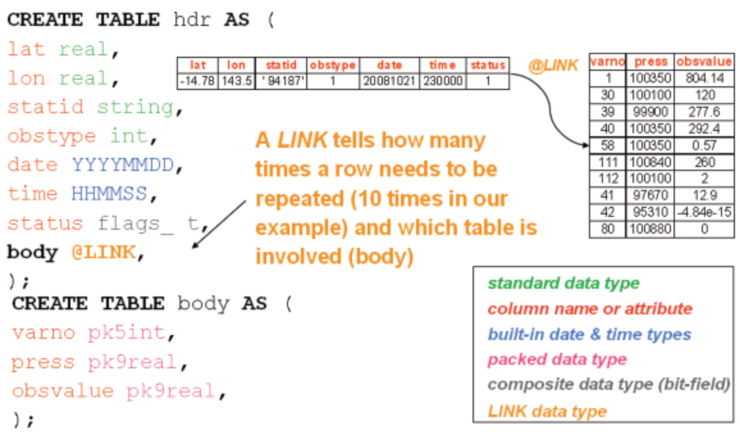
- Data definition: to define structure and organization of data to be stored and relationships between the items;
- Data retrieval: to define data queries in order to retrieve (normally) a subset of data items.

3.3.1 Data Definition Language (DDL)

The CREATE TABLE statement is used to create a table in a database:

```
CREATE TABLE table name AS (
      column_name1 data_type1,
      column_name2 data_type2,
      column_name3 data_type3,
      ....
);
```

The data type specifies what type of data the column can hold. Figure 3 shows how to create tables and link them together (via @LINK data type). The link describes the hierarchy between the two tables.



```
CREATE TABLE hdr AS (
lat real,
lon real,
statid string,
obstype int,
date YYYYMMDD,
time HHMMSS,
status flags_ t,
body @LINK,
);
CREATE TABLE body AS (
varno pk5int,
press pk9real,
obsvalue pk9real,
);
```

lat	lon	statid	obstype	date	time	status
-14.78	143.5	'94187'	1	20081021	230000	1

varno	press	obsvalue
1	100350	804.14
30	100100	120
39	99900	277.6
40	100350	292.4
58	100350	0.57
111	100840	260
112	100100	2
41	97670	12.9
42	95310	-4.84e-15
80	100880	0

@LINK
A *LINK* tells how many times a row needs to be repeated (10 times in our example) and which table is involved (body)
standard data type
column name or attribute
built-in date & time types
packed data type
composite data type (bit-field)
LINK data type

Fig. 3 Example of ODB DDL.

3.3.2 Data Manipulation Language (DML)

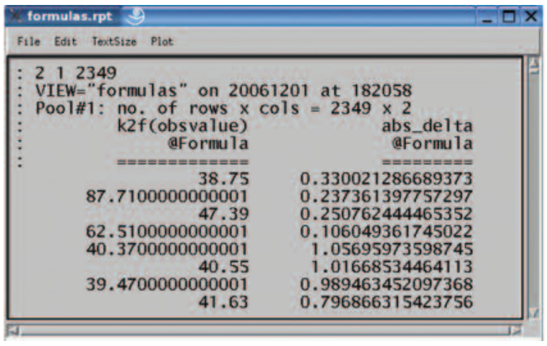
The Data Manipulation Language implements a data query engine in order to search in particular columns of a database satisfying certain conditions. The syntax is as follow:

```
[CREATE VIEW view_name AS]
SELECT [DISTINCT] column_ name( s)
FROM table( s)
[WHERE some_ condition( s)_ to_ be_ met ]
[ORDERBY sort_ column_ name( s) [ASC/ DESC] ]
```

For example:

```
SELECT fahrenheit(obsvalue), // Convert from Kelvin to F
abs(fg_depar - an_depar) AS abs_delta
FROM hdr, body
WHERE
obstype = $synop
AND
varno@body = $t2m
AND
obsvalue is not NULL
;
```

This SQL query will return 2 columns (as shown on Figure 4): the first column is the temperature (2m temperature as indicated in varno@body) in Fahrenheit (ODB Fahrenheit function is used) and the second column is the absolute difference between the first guess departure (fg_depar) and analysis departure (an_depar).



```
: 2 1 2349
: VIEW="formulas" on 20061201 at 182058
: Pool#1: no. of rows x cols = 2349 x 2
:           k2f(obsvalue)             abs_delta
:                @Formula              @Formula
:           =============             =========
                    38.75     0.330021286689373
        87.7100000000001     0.237361397757297
                    47.39     0.250762444465352
        62.5100000000001     0.106049361745022
        40.3700000000001      1.05695973598745
                    40.55      1.01668534464113
        39.4700000000001     0.989463452097368
                    41.63     0.796866315423756
```

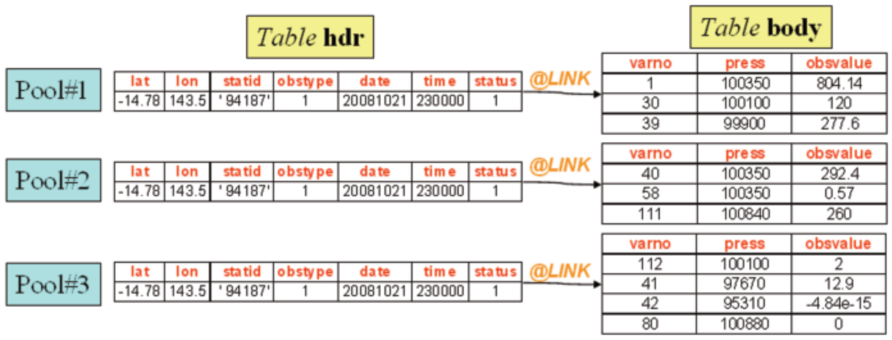
Fig. 4 SQL output as returned by odbviewer

3.4 *Parallelism in ODB*

ODB is a parallel database system: it aims to improve performance through parallelization of various operations, such as loading data, building ODBs and evaluating queries. Data is stored in distributed fashion:

- TABLEs are divided "horizontally" into pools between processors as shown on Figure 5 (pools are assigned to the MPI-tasks in a round-robin fashion);
- Each table can be assigned to an openMP threads.

The number of pools is "decided" in the Fortran 90 layer and can be changed at any time. However, the user has to cater for allocating data to each pool. Therefore we usually distribute data among pools at the creation of the ODB database.



lat	lon	statid	obstype	date	time	status
-14.78	143.5	'94187'	1	20081021	230000	1

varno	press	obsvalue
1	100350	804.14
30	100100	120
39	99900	277.6

lat	lon	statid	obstype	date	time	status
-14.78	143.5	'94187'	1	20081021	230000	1

varno	press	obsvalue
40	100350	292.4
58	100350	0.57
111	100840	260

lat	lon	statid	obstype	date	time	status
-14.78	143.5	'94187'	1	20081021	230000	1

varno	press	obsvalue
112	100100	2
41	97670	12.9
42	95310	-4.84e-15
80	100880	0

Fig. 5 Example of data partitioning.

A single pool forms a "sub-database".

To improve performance, only a subset of pools is selected to perform I/O (read/write ODB files on disk). Similar tables are then concatenated together. The number of I/O pools is fully configurable.

3.5 ODB Fortran 90 access layer

Parallel data queries are possible via the ODB Fortran 90 interface layer. The Fortran 90 layer offers a unique user interface to:

- open & close database;
- execute ODB/SQL queries, update & store queried data;
- inquire information about database metadata.

The same Fortran 90 code can be used in serial or parallel MPI/OpenMP mode (with any number of processors/openMP threads). Selected data can be asked to be shuffled ("part-exchanged") or replicated across processors (by default data selection applies to the local pools only).

3.6 ODB tools

Various ODB-tools are provided to simplify browsing and management of ODB databases. Some of the tools are such that one can completely ignore programming efforts: just retrieve and plot the data.

Much effort has been made to allow access of ODB databases through ECMWF software such as Metview (to access, manipulate and visualise meteorological data) and obstat (our observation monitoring software).

In addition, the two following ODB-tools are available in the ODB package:

- odbsql: a tool to access ODB data in read/only mode
- odbdiff: a tool to compare two ODB databases

4 ECMWF usage of ODB in IFS

Two main ODB databases are used at ECMWF:

- ECMA (Extended CMA)
- CCMA (Compressed CMA)

The ODB hierarchy is as described on Figure 6.

The database ECMA contains all the observations to be used in a 4D-Var analysis. These are the observations converted from external sources like BUFR-files or ASCII text files into ODB. The very first step (traj#0) of the 4D-Var suite screens observations found in ECMA in order to select a suitable set for the subsequent 4D-Var minimization process. These so-called active observations are placed into a separate database called CCMA. Both databases are distributed between MPI-tasks so that every task roughly processes the same number of observations in each timeslot of 4D-Var.

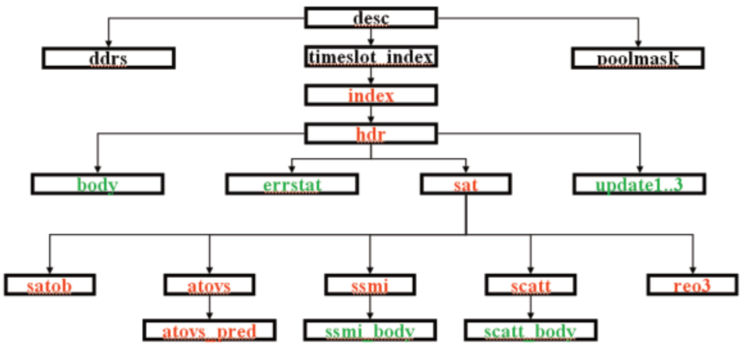


Fig. 6 ECMWF table hierarchies

We do not have a unique/centralized ODB database: a new ODB is created for each analysis and only contains observations needed for this analysis.

ECMWF ODBs are currently stored in ECFS which is a large distributed storage system. Feedback bufr files are created from ODBs at the end of the analysis and archived in MARS (our Meteorological Archive).

5 The way forward

ODB is now more than a tool only dedicated to ECMWF's 4D-Var system, it became also very popular to post-process observational data.

In the short term we plan to better integrate ODB in our full ECMWF system (from receiving observations to the archiving of feedback information).

The first step of this integration will be to archive ODBs in our Meteorological archive (see Peter Kuchta's presentation).

In addition, there is a growing interest on ODB from external Centres: ODB is already used by Australian Bureau of Meteorology (Melbourne) and this triggered some interest by UK Meteorological Office, GMAO (Washington) also investigates the possibilities of ODB for their own usage, etc.

Therefore we plan to make ODB easier to handle by external parties and for this we have two on going projects. The first one consists in revisiting the current hierarchy of tables as well as the ODB attributes (columns) defined in our DDL file and to create a dictionary of all existing ODB attributes with their meaning (description, units, etc.). The second project aims at redefining the ODB user interfaces: the idea is to specify two generic user interface layers, usable not only at ECMWF but also by external centres (UK Met. Office, Bureau of Australian Meteorology, etc.). The first user interface would be able to handle any kind of databases, and the second user interface (built on top of it) would be specific to our schema file (ECMA/CCMA).