Base/OPEN Object–Oriented Data Base Management System, EasyDB – Release 5

Overview

The Base/OPEN Object-Oriented Data Base Management System *EasyDB* represents a new generation of data base technology developed in the wake of relational data base systems.

The Base/OPEN Data Base Management System (DBMS) is optimized for long–lasting and large transactions which are characteristic of engineering applications. It provides facilities to create and maintain a common data base for different applications. The data base is *object-oriented*, which provides for natural, real-world oriented data modelling making it particularly efficient for CAD/CASE-applications. It is available for several modern workstations making it possible to write portable applications.

Two products Base/OPEN Object-Oriented Data Base Management System is shipped in one of the following configurations.

- *EasyDB* EasyDB Release 5.2 is a multi-user DBMS. It allows network distribution, utilizing NCS and NFS or any other global distributed file system for optimized performance.

- *EasyDB LiTe* EasyDB Release 5.1 is a single-user data manager (in the same sense as a UNIX file system), designed to accelerate the development rate for a single tool. The user interface is upward compatible with multi-user EasyDB 4.2.

Product Perspective

The purpose of the Base/OPEN DBMS is to be supremely useful for engineering applications, e.g. CAE, CAD, CASE. We take a practitioner's view, how to make things work. Extensions to traditional database models, like object–orientation, are a means to ths end, not goals by themselves.

Performance The first question a potential application developer asks about a DBMS is invariably. ”What about performance?”

EasyDB is designed to allow applications to keep much of their run-time data structures in the database, without sacrificing too much performance compared to hard-wired tool specific data structures.

Conceptual Simplicity The philosophy is to achieve performance by conceptual simplicity. This is another aspect of performance orientation. The value of added concepts and facilities have been carefully weighed against their cost in terms of performance.

The Classical Database Problem EasyDB provides solutions for ”the classical database problem”: *achieving data independence under the assumption that data is a shared resource*. Here lies much of the motivation for using a DBMS in the first place. Main goals are,

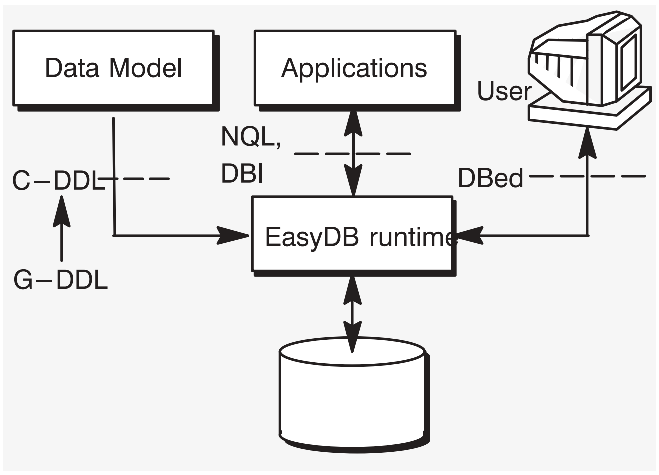
- To make data definitions visible, sharable (in contrast to being buried in application code), and largely programming language independent.
- To free data definitions from storage mechanism considerations (in contrast to being hardware-dependent).
- To allow data definitions to be modified (or at least extended) without invalidating existing data or applications.

Persistence is not enough It is clear that just picking an object-oriented programming language and making its objects persistent does not achieve these goals.

EasyDB Product Description

Interfaces The Base/OPEN EasyDB has two generic interfaces – *DDL* and *DML*. The DDL (Data Definition Language) is based on the well-known Entity/Relationship model[1] and is used to define the structure of data in terms of *entity types*, *relationships* and *attributes*. The DML (Data Manipulation Language) works on two different levels: *interactive* for ad-hoc access and query and as an *embedded language* for highest performance.

1. Concepts and terminology for the Conceptual schema and the information base, ISO/DTR 9007.



Data Definition Language The data description language is strongly influenced by entity-relationship concepts. It allows you to define *entity types* and *relationships* between them. An entity type is defined as a set of *attributes*. Relationships are inherently bi-directional. This is the basic framework upon which two strains of DDL are defined – *C–DDL* and *A–DDL*. All DDL definitions are stored in a *data dictionary* (DD). *G–DDL* is an interactive graphical modeling tool.

C–DDL – Concepts:

- ***entity type***
  single inheritance between entities
- ***attribute***
  entity types may have attributes
  Database keys model global relationships
- ***relationship***
  bi–directional, between entity types, local
- ***cluster type***
  collection of related entity types
  introduces version control level

C–DDL Conceptual DDL (C–DDL) is the language you use to express your data model (schema). With the exception of clusters, C–DDL is kept on a logical level.

Entity types The basic modelling unit is the entity type which has a *name* and may have *attributes*. An attribute has a name and a value. The entity is the *small grain object*.

Attributes There is a rather complete type system for attributes. In addition to conventional data types there are two specialized ones: *Bytestream* for storing large varying length character strings – the source code for a program module, for instance. *Database key* for storing references between entity instances in different clusters in the data base thus used to model global relationships.

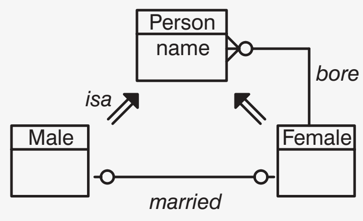
Domains Strictly speaking, an attribute is associated with a *domain* rather than with a type. A domain is defined as a base type plus a range of possible values. Domain definitions are collected in compilation units called *packages*.

Relationships Relationship types in EasyDB are binary and local, connecting two entity types within a cluster. Relationships may be traversed in both directions, i.e. they are *bi–directional*. Thus *referential integrity* is guaranteed. Relationship types may be of different cardinalities. The cardinalities are:

- *one–to–one* and
- *one–to–many*.

Either or both ends of a relationship may be qualified as *required*. This is a means for expressing that entity instances at relationship ends must be attached to each other.

Inheritance An entity type may inherit another entity type. All attributes and relationships are inherited in such case. Any number of inheritance levels are allowed.



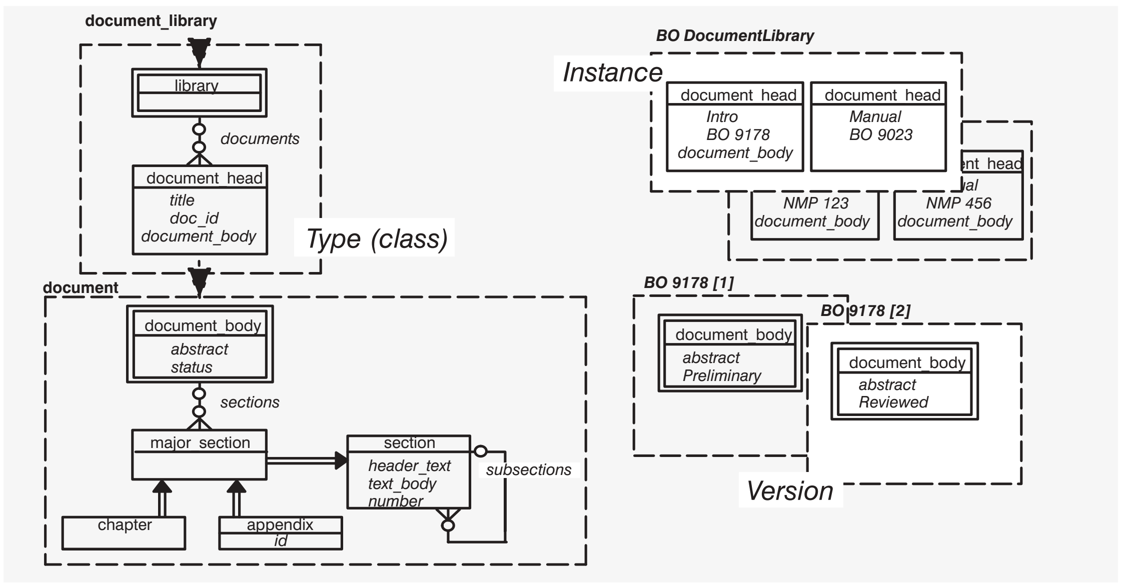
```
-- C-DDL text:
. . .
ENTITY Female ISA Person : femaleType
-- Female inherits Person
. . .
REL bore ONE Female MANY Person
-- A Female may have given birth to many
-- Persons (Male or Female)
```

Cluster types The implementation independent nature of C–DDL is broken by the concept *of cluster type.* This sets a physical boundary around a structure of entity types and relationships thus defining a *large grain object* or aggregation. Relationships crossing a cluster type boundary are called *global* in contrast to *local relationships.* The global relationship is in EasyDB modelled by the Database key attribute which is a one–directional pointer.

Aggregation The main reason for not concealing cluster boundaries behind a logical layer is performance. In CAE a large design is always divided into comprehensible design objects, e.g. a Document consists of section headers, paragraphs etc.. Thus we believe there is generally a natural mapping between design objects and cluster types.

Schema Evolution New cluster types can be added to a universe at any time. A cluster type definition can be modified, as long as it is only extended, without disrupting data base operation. E.g. you can add a new attribute, a new relationship etc. to an entity type without any need of re-compiling old applications. In other words, there may be several versions of one and the same cluster type. Instances of old cluster type versions may co-exist with instances of newer versions.

Versioning The cluster is the unit of physical i/o. A cluster, when opened, is transferred in its entirety to virtual memory. The cluster is also the unit of versioning and access control. Cluster versions are organized in tree structures called version trees. The version tree is the unit of distribution.



Structural OO In the terminology of Dittrich, K.R.[2] EasyDB (ODBMS) is *structurally object–oriented*. That is, methods are not stored in the data base.

Universe C–DDL is also used to define a data *universe* for applications. A universe may be, for example, all data needed by all CAD tools, the ”cad-production-universe”, or a ”test-universe” for testing out new applications and changed schemas. The universe consists of a number of cluster types.

A–DDL A universe may contain many cluster types. Application DDL (A–DDL) is used to define a *view (application schema)*, i.e. a subset of the total schema. A data base application has to be associated with one or more views.

2. Object–Oriented Databases from Entity–Relationship Approach, S. Spaccapietra (ed.). Elsevier Science Publishers 1986.

Data Manipulation Language

EasyDB offers two DML's for program development, NQL (for programs written in Ada and C) and DBI (for programs written in C++ and Ada95).

Dynamic and static DML

There are two strategies when accessing data, *static* and *dynamic* access. Static access is type-safe and the fastest in terms of execution speed. Dynamic access is the flexible way, useful for instance when writing application independent database browsing tools, like DBed.

NQL – a C and Ada Interface

The basic DML is called *NQL* (Navigational Query Language). It is embedded in a host language (C and Ada at the moment). NQL in itself is host-language independent. As its name suggests, NQL is navigational.

NQL Concepts:

- ***cursor***
  refers to an entity instance or is nil
- ***navigation***
  move cursor from entity to entity by select
- ***query and update attribute values***
- ***create/delete, attach/detach entities***
- ***other*** *open/close on clusters*

NQL (C–language binding) – Example:

```
/*
 * Print the name (and age) of all children
 * not older than 18 years. See ”C-DDL” figure.
 */
NQL DECLARE (CURSOR) cur;
. . .
NQL EVERY cur(Female) MEMBER bore
    WHERE { cur(Person)->age <= 18 }
    FROM femaleCur
        printf(”Name %s”, cur(Person)->name);
        printf(” Age %d\n”, cur(Person)->age);
NQL ENDEVERY
. . .
```

Cursors

NQL introduces a cursor concept. A cursor either refers to an entity or is nil. Cursors are declared in a way similar to ordinary variables.

NQL (Ada–language binding) – Example:

```
 -- Print the name (and age) of all children
 -- not older than 18 years. See ”C-DDL” figure.
 --
NQL DECLARE (CURSOR) cur;
. . .
NQL EVERY cur(Female) MEMBER bore
    WHERE { cur(Person)->age <= 18 }
    FROM femaleCur
        Text_Io.put(”Name ” & cur(Person)->name);
        Text_Io.put_line(” Age ” &
            integer’image(cur(Person)->age));
NQL ENDEVERY
. . .
```

DBI – a C++ and Ada95 Interface

DBI is a C++ and Ada95 interface to EasyDB. It deals with data manipulation, i.e. it is a DML (like NQL) as opposed to a DDL. DBI is used for writing C++ and Ada95 application programs to access data in EasyDB. The DBI system has two main parts:

- the *predefined class library*, and
- the *view generator program.*

DBI Classes

The class library interface provides all general functionality of the data base manager interface. The view generator program generates schema-specific classes, types and constants which extend the existing class library with faster, more specialized functionality. A schema-specific class corresponds to the C–DDL entity concept.

DBI Concepts:

- ***entity handle concept***
  typed reference to an entity instance or is nil
- ***navigational***
  move handle from entity to entity by select or iteration –– by entity class operations
- ***query and update attribute values*** *–– by entity/attribute operations*
- ***create/delete, attach/detach entities*** *–– by entity class operations*
- ***other*** *open/close on clusters –– by entity class operations*

A DBI Application

In short, a DBI application is created in the following steps:

- Create a schema description with C–DDL or G–DDL tools. The result will be a representation of the schema which is stored in the *Data Dictionary*.
- Define the application view, a subset of the conceptual schema, by means of A–DDL.
- Generate ready-to-use schema-specific C++ or Ada95 classes, types and constants with the *cxxview* or *ada95view* tool.

DBI C++–language binding – Example:

```
/*
 * Print the name (and age) of all children
 * not older than 18 years. See ”C-DDL” figure.
 */
sreg_Person personH (myModule); // decl. handle
dbi_Iterator myit;         // declare iterator
. . .
// initialize an iterator
femaleH.bore_every_age (myit, DBI_LEQ, 18);
while (myit.next (personH) == DBI_TRUE)
    {
        cout << ”Name ” << personH.name();
        cout << ” Age ” << personH.age() << endl;
    }
. . .
```

DBI Ada95–language binding – Example:

```
--
-- Print the name (and age) of all children
-- not older than 18 years. See ”C-DDL” figure.
--
Person_H : SReg.Person(My_Module); -- decl. handle
My_It : DBI.Iterator;     -- declare iterator
Status : DBI_Standard.DBI_Bool :=
                     DBI_Standard.DBI_True;
. . .
-- initialize an iterator
SReg.Every (SReg.Get_Bore(Female_H), Female_H,
   My_It, DBI_Standard.DBI_LEQ,
   (Sreg.Age, Sreg.Age, 18));
while Status = DBI_Standard.DBI_True loop
   DBI.Next(My_It, Person_H, Status);
   if Status = DBI_Standard.DBI_True then
      Text_IO.Put( ”Name ” & SReg.Name(Person_H)
      Text_IO.Put_Line(” Age ” &
         Integer’Image(SReg.Age(Person_H));
   end if;
end loop;
. . .
```

- Write your C++ or Ada95 application programs utilizing the DBI library and the generated classes to create a *data base* and manipulate its contents. It is possible, and sometimes very useful, to write an application without using generated classes at all. This is called *dynamic DBI* versus static DBI where you use the names defined in the schema. The DBI example uses static DBI – i.e. we call member functions of generated schema-specific classes.

BGL Interfaces provided by a DBMS (either procedural or embedded textual language) are often aimed at optimizing application run–time performance with the disadvantage being the necessity of relying on special programs written in C/C++ or Ada. The Base/OPEN BGL tool set, on the other hand, functions at a higher level and is optimized to minimize development time.

BGL Tool Set The Base/OPEN BGL tool set consists of graphical modeling and interactive query tools. By employing BGL it is possible to develop and maintain conceptual models and to accelerate the development of query and update applications.

Furthermore greater independence from the underlying DBMS concepts is achieved as the work itself is conducted on the well-known, extended entity-relationship concept level. With the BGL tool set it is no longer necessary to iterate in the edit–compile–run–debug loop.

Graphical DDL The Graphical Data Definition Language (‘‘G–DDL’’) is a tool for conceptual modeling and schema definition based on the data modeling concepts Entity–Relationship–Attribute (ERA) combined with Object-Orientation (OO).

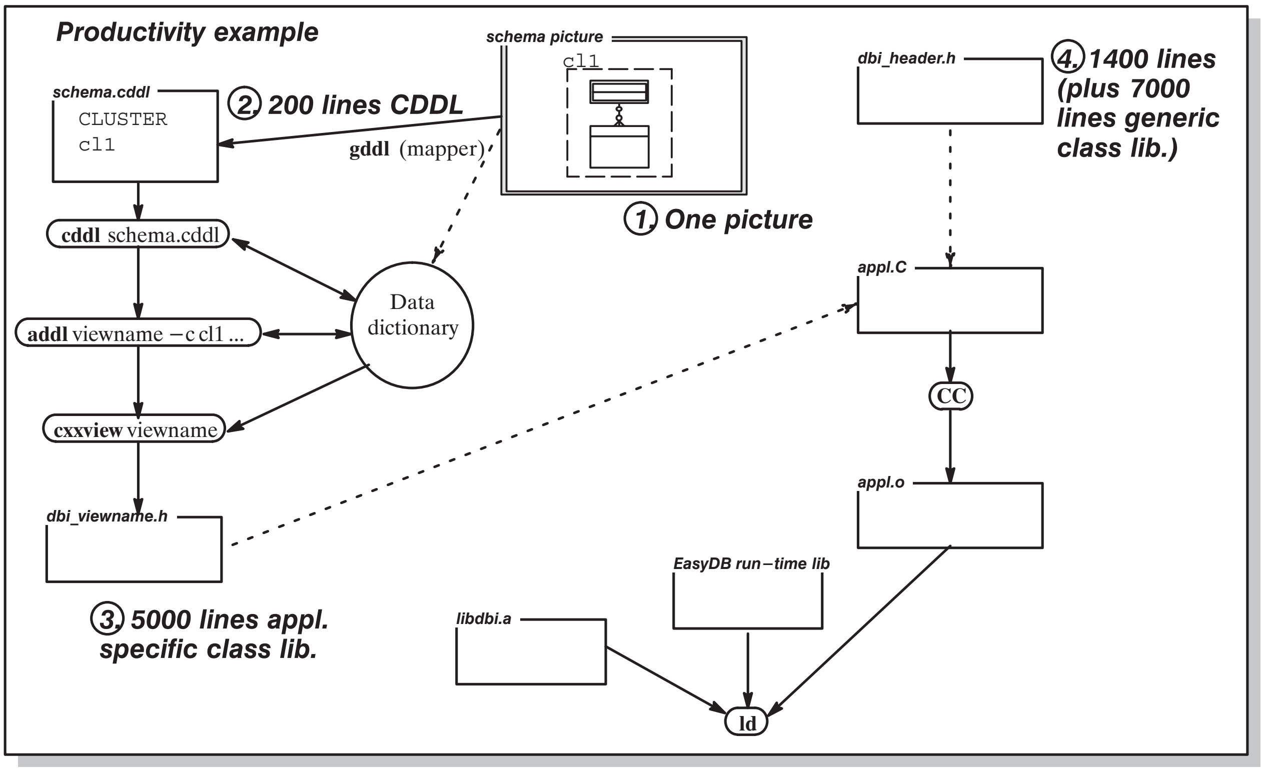
Interactive DML The interactive DML is called *DBed*. DBed is a tool for ad-hoc query (”interactive NQL”) and rapid development of data base applications. DBed also supports scripting and provides interactive query access to the *data dictionary* (DD), of EasyDB. DBed is written entirely in C and dynamic NQL, thus demonstrating the strength and openness of the EasyDB interfaces.

DBed (interactive DML) – Example:

```
-- print the name (and age) of all children,
-- not older than 18 years.
   Female >> every bore (age<=18) {print :name :age}
Attribute: 'name'      = 'John'
Attribute: 'age'       = '13'
Attribute: 'name'      = 'Susan'
Attribute: 'age'       = '9'
(DBed) No more members.
-- Query DataDictionary for relationships,
-- where Female has the role of Head
    Female >> dd head
HEAD --> sreg.Female
           Female -(bore)* Person
           Female -(married)- Male
```

EasyDB Productivity

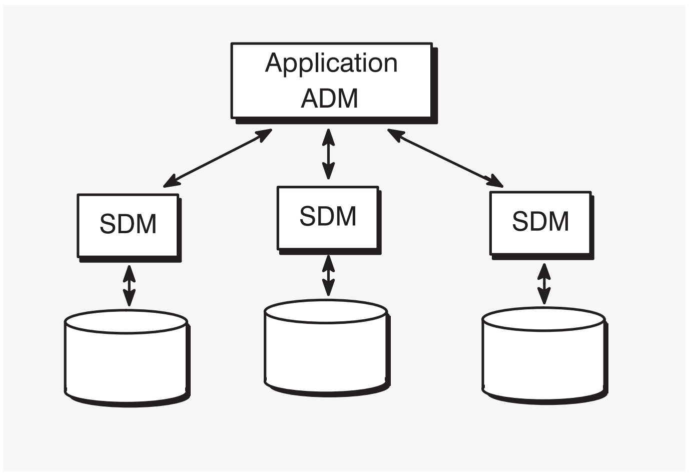
The figure below illustrates the productivity in using DBI – the C++ interface to EasyDB. Assume we start off from one schema picture with 15 entities each having 5 attributes, 15 relationships and 5 clusters, created with the G–DDL modelling tool. From this, about 200 lines C–DDL may be generated. The schema is compiled into the data dictionary with the *cddl* tool. The *cxxview* class generator reads the data dictionary and creates schema specific-classes, query member functions etc., for the view defined with the *addl* tool. The generated code will be about 5000 lines of C++ code in this case, ready to use in application (*appl.C* in the figure) development or for extending DBI with user-defined member function.



EasyDB Architecture

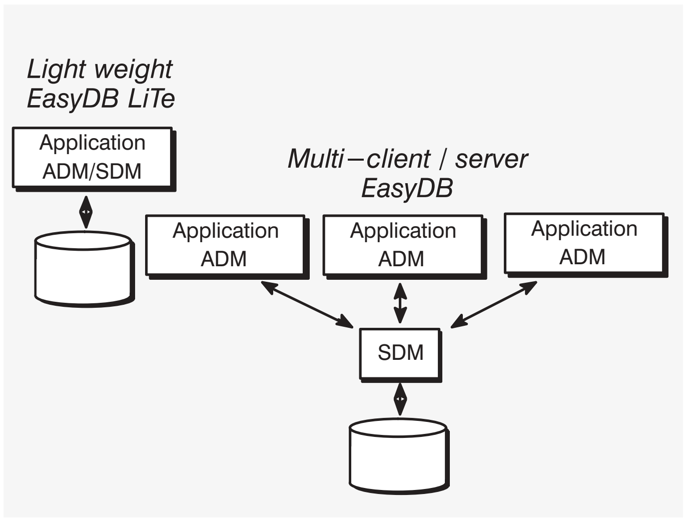
The EasyDB system is implemented as two levels of data managers in a multi-client – multi-server architecture. This is illustrated in the next figure. The data managers are called ADM and SDM.

- ADM – Application Data Manager, library to be linked with the application.
- SDM – Storage Data Manager. One process for each host/disk you want to store database contents on.

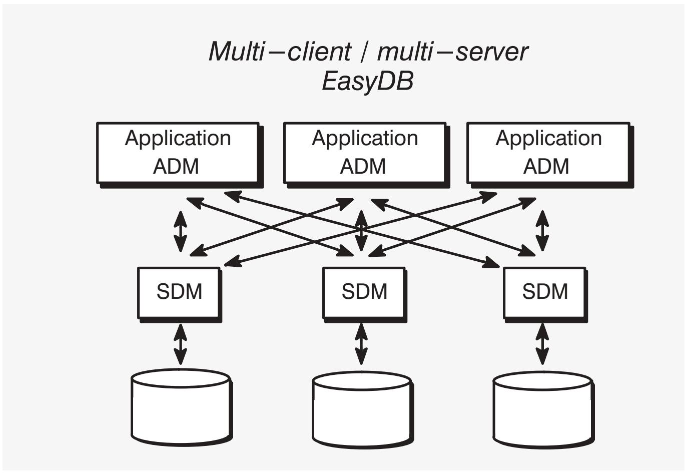


Example Configurations

EasyDB can be configured in numerous configurations as outlined in the following figures.



The above figure first shows light weight EasyDB LiTe and next multi-user EasyDB in a single server and multi-client configurations. The clients need not run on the same host as the servers. Below we have added more EasyDB servers running on different hosts in a local area network, a multi-server configuration. Clients running on different hosts in the network have transparent access to data managed by the different servers. This is configured by the database administrator. EasyDB includes other server processes not included in the figures.



Performance Considerations

Here are some major performance-related design considerations.

- The whole EasyDB design rests on the assumption that a large number of operations are performed on a cluster once it has been opened.
- NQL has strong (static) type checking. As much work as possible is moved from run-time to compile-time. (Most NQL statements appear in two versions, with static or dynamic type checking. Dynamic type checking is sometimes very useful, but not as efficient as static type checking.) This is a strong reason for making *NQL embedded* rather than implementing it as a procedure library.
- The run-time system uses a binary summary of pertinent schema data. This data structure (called a cluster map) is generated by the C–DDL compiler. It eliminates the need for run-time access to the data dictionary.

Performance Factors

Measurements have confirmed that there are two key performance factors. The first is the time for bringing a cluster from disk to virtual memory; the second is ”in-core” operations.

Storage I/O	The cluster transfer speed from a local disk is about 500 kb/s. This is not too far from the OS/UNIX file transfer speed. Across an 10Mbit Ethernet LAN (NFS) about the same cluster reading speed can be reached but the writing speed is decreased.
In–core performance	In–core operations are very fast. As a rule-of-thumb a simple operation (navigating one step, or getting the value of an attribute) takes 40 μs (on a two MIPS machine). This includes complete checking for all kinds of conflicts, including domain boundaries. This is in the order of 100 times faster than a averages of similar operations reported in Duhl, J. & Damon, C.[3] (Data size is also cut to around one third.)
Transactions	EasyDB supports two levels of transactions, *short-term* and *long-lasting*. Short-term transactions include *open, close, checkpoint*. Long-lasting transaction concepts are *reserve, replace, branch, cancel.*

Product Summary

EasyDB releases 5 product highlights are summarized in:

- Multi-user system with network distributed storage – can be scaled down to single-user and single-host configurations.
- Strong modeling system (both graphical and textual). Dynamic schema evolution.
- Well integrated language bindings. Static or dynamic name resolution. Multimedia support.
- Safe and easy to use conflict handling throughout the system.
- Support for Configuration Management.
- Powerful distribution mechanism. Network topology independent.
- Technology independent layers.
- High performance. Short and long–lasting transactions.

3. A Performance Comparison of Object and Relational Databases Using the Sun Benchmark. Proc. of OOPSLA Conf. 1988.

Product Releases

EasyDB multi-user releases 5.2 and EasyDB LiTe releases 5.1 are the versions of Base/OPEN ODBMS available as supported products at this writing. EasyDB 4.x and 5.x supports both 64-bit and 32-bit architectures.

Both EasyDB releases are easy to install and maintain. The EasyDB multiuser releases includes utilities for starting and monitoring storage data managers and for building a configuration file defining the hardware and run-time configuration.

Porting Base and Hardware Requirements

EasyDB releases 5.1 and 5.2 are available on most UNIX platforms (e.g. Sun Sparc, Digital Alpha, SGI) and for Microsoft Windows on Intel.

Cross-platform development is supported. The EasyDB run-time is easily ported to POSIX compliant platforms. Application development may be done on a e.g. standard UNIX platform, and the application may then be re-compiled and linked on the target platform.

Please contact Basesoft Open Systems or your local EasyDB representative for further information on which configurations of tools, compilers etc. are supported on different platforms.

- An EasyDB development installation requires 10–40 Mb of disk storage depending on configuration.
- An EasyDB run-time installation requires 5–20 Mb of disk storage for EasyDB software apart from the disk storage needed for the data base.
- There is no special requirement on physical memory. The EasyDB SDM server uses about 1–3Mb of memory. The EasyDB run-time caches data to clients. Host running client applications should have free physical or virtual memory available for at least twice the size of the largest cluster instance used (e.g. 1–20Mb).

Pricing and Licensing

Price for first multi-user development and 5-user run-time system license is USD 998 or EUR 828. EasyDB LiTe is available starting from USD 860 for one OEM application run-time. Quantity, educational and re-seller discounts are available. Maintenance and support agreements are also available. Evaluation licence available. One or three day introduction courses will get you easily started using the product.

Documentation

The following documentation is provided with the EasyDB product:

- EasyDB C−DDL Reference Manual
  BO 90 082
- EasyDB NQL Reference Manual
  BO 90 060
- EasyDB DBed Reference Manual
  BO 89 206
- EasyDB Tools Reference Manual
  BO 91 013
- EasyDB DBI C++ Reference Manual
  BO 91 088
- EasyDB DBI Ada95 Reference Manual
  BO 98 102
- EasyDB NQL C example
  BO 91 008
- EasyDB NQL Ada example
  BO 93 012
- EasyDB and C++ − example
  BO 91 123
- EasyDB and Ada95 − example
  BO 97 521
- EasyDB Modelling
  BO 91 009
- EasyDB Installation and Release Document
  BO 91 116
- EasyDB LiTe Installation and Release Document
  BO 96 202
- EasyDB Administrator's Guide
  BO 91 115
- EasyDB Glossary
  BO 91 002
- EasyDB DBed sample session
  BO 90 110
- EasyDB GDDL, Ramatic, Reference
  BO 91 137

Media The software, including on-line manual pages, is distributed on a selection of media types depending on the host platform.

For further information, please contact your local EasyDB representative or Jaan Haabma at Basesoft Open Systems AB on:

phone: +46 8 13 17 20, fax: +46 8 13 17 25,
E−mail: request@basesoft.se

Post address:	**Office address:**	**Phone:**
Basesoft Open Systems AB	Gjörwellsgatan 22, 13 TR	+46 8−13 17 20
Box 34 140		**Telefax:**
S−100 26 STOCKHOLM	MARIEBERG−STOCKHOLM	+46 8−13 17 25
Sweden		