



your partner for successful **IT** business

cplusDBAction

The Ultimate C++ DB ORM Framework

Fläckeohof 34
CH-6023 Rothenburg
Phone: +41 41 281 27 40
Fax: +41 42 281 27 34
E-Mail: info@addoit.com

What is an ORM Framework?

- ORM stands for Object-Relation-Mapping
- ORM represents the mapping of in-memory Business Objects to records in relational databases for persistence purposes
- Object Oriented Programming (OOP) mainly deals with in-memory object instances of classes
- Most Database Management Systems (DBMS) can only handle scalar data types; the ORM Framework establishes the connection between these relational DB Systems and the Business Objects

Requirements to an ORM

- To transform the tables and fields of a DBMS into strongly-typed objects using the programming language required by the Business Layer
- To persistently store a Business Object's data into a DBMS
- To restore a Business Object's data from persistent data located in a DBMS
- To store/modify data into a DBMS without violating the ACID properties: Atomicity, Consistency, Isolation and Durability
- To provide the appropriate data manipulation methods and functions to the Business Layer

ORM Product Overview

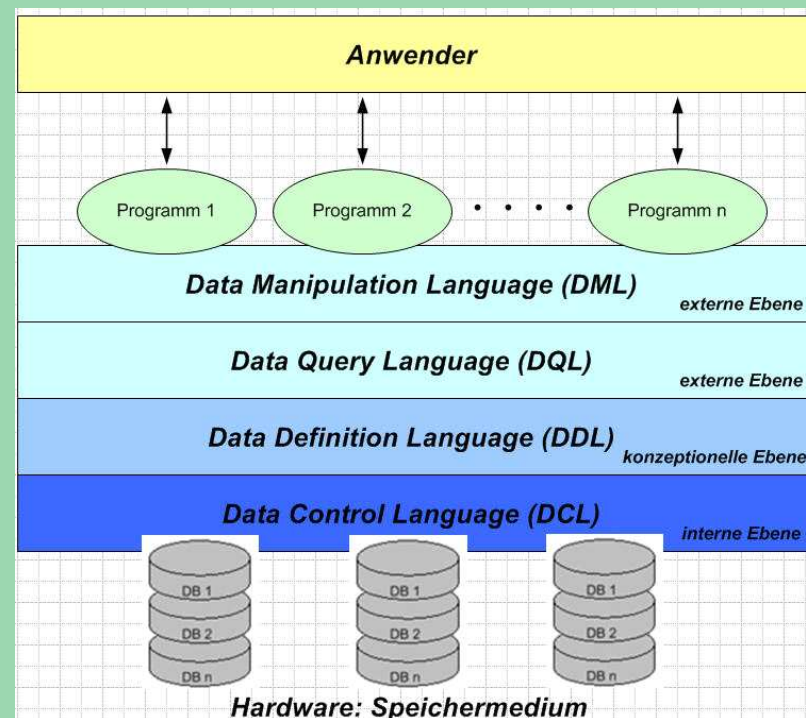
- ORM products are mainly programming language dependant (Java, C#, C++, etc.)
- Some ORMs are based on a specific platform or environment (e.g. Microsoft .NET Framework)
- Not all ORMs are compatible with all DBMS
- Most ORMs are operating-system dependent
- The ORMs define different database development processes and services

cplusDBAction: more than an ORM

- The first real C++ ORM
- Operating system, platform and DBMS independent
- The Framework locates possible errors in the Business Layer already at compilation time (performing compatibility checks between DBMS, DB Layer and Business Layer)
- Features like data export, data import, source code generation, schema generation, etc. are provided as a Database Development Process to the software developer
- Additional services: runtime compatibility checks, key generation, row stamps for inserts and updates, etc.

cplusDBAction Architecture (1/3)

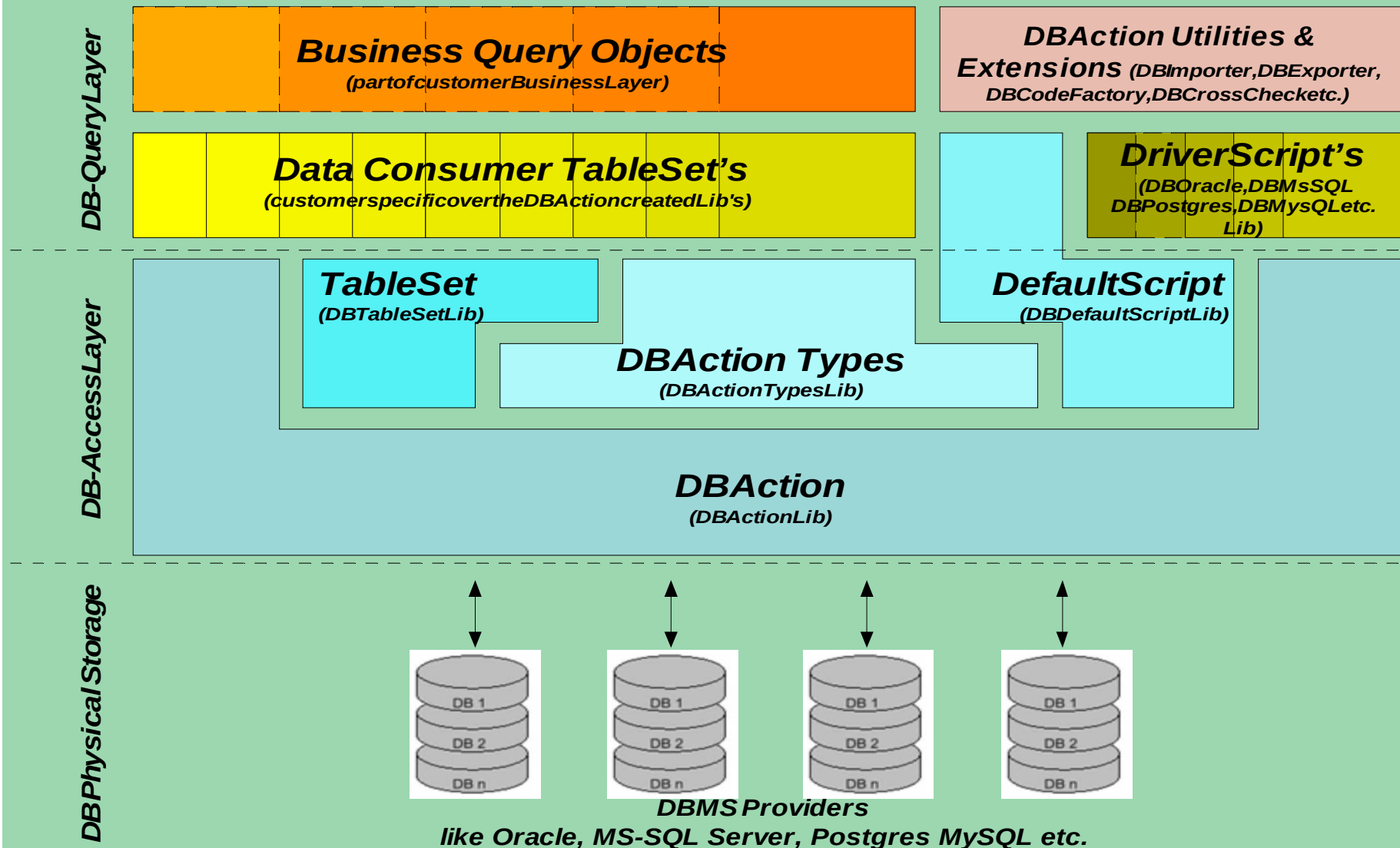
- cplusDBAction operates between the Business Logic and the DBMS and provides at least the DML, DQL, DDL und DCL functionality (also known as Database Layer)



cplusDBAction Architecture (2/3)

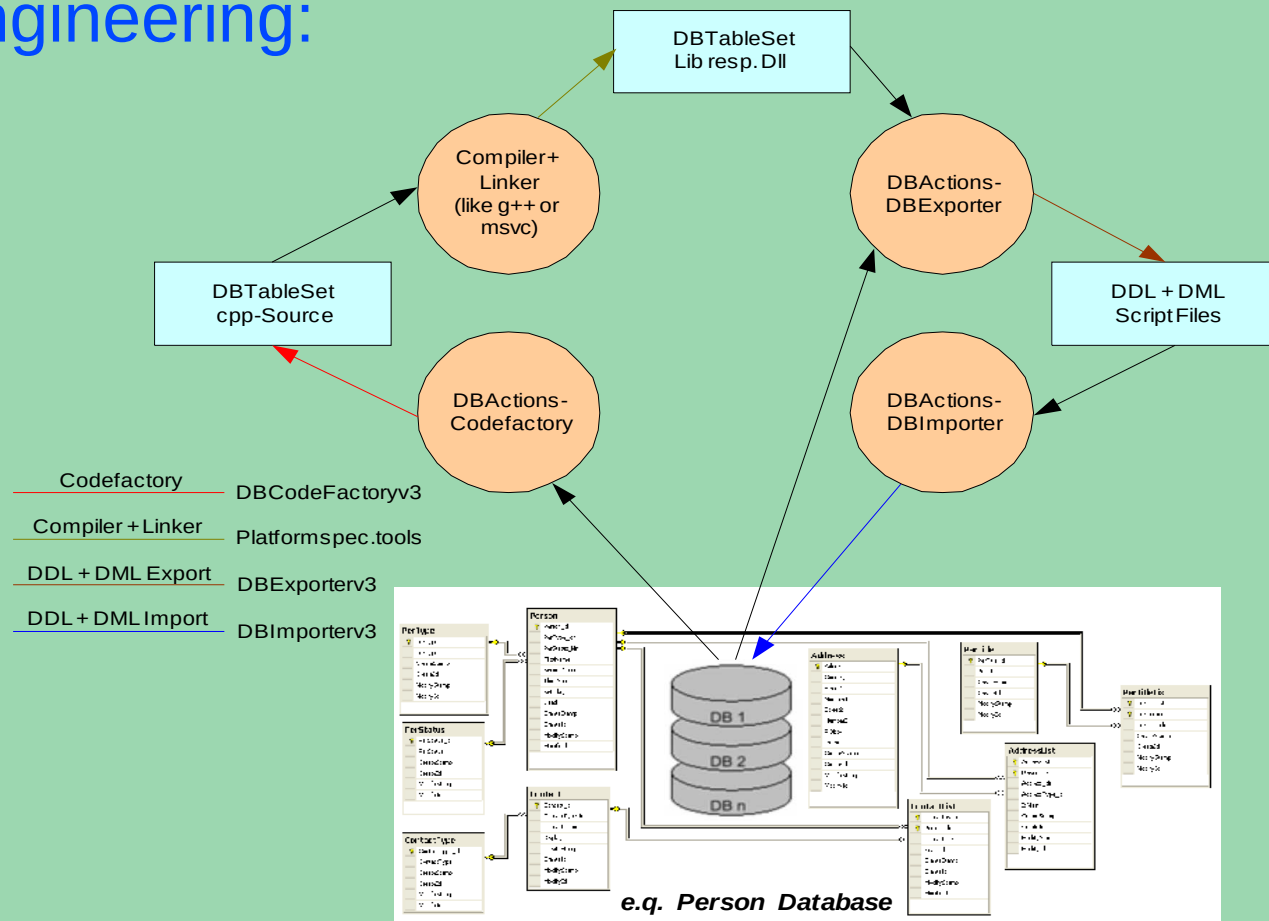
- For clarity purposes, the Database Layer is divided into two layers:
 - **DB-Access Layer** (completely managed internally by the cplusDBAction)
 - **DB-Query Layer**, which consists of:
 - Business Query Objects
 - generated Source Code (Data Consumer Tablesets)
- Software developers only need to write code within the Business Query Objects
- The DBAction Utilities and Service Layer contains several helpful Tools for software developers (for details, refer to the next page →)

cplusDBAction Architecture (3/3)



DB Development Process (1/2)

- cplusDBAction also provides support for Reverse Engineering:



DB Development Process (2/2)

- ***DBCodeFactory*** reads all information out of an existing DBMS-Schema and generates C++ source code
- This generated C++ source code is then compiled into a library using an appropriate compiler for the target platform (g++, msvc, etc.)
- ***DBExporter*** uses this library to generate all DDL specific scripts (and, if any data available, also the DML specific scripts)
- ***DBImporter*** generates the specified Tables, Fields, Views, etc. into a target DBMS (and can optionally import the data using the previously generated DML scripts)

cplusDBAction Features (1/2)

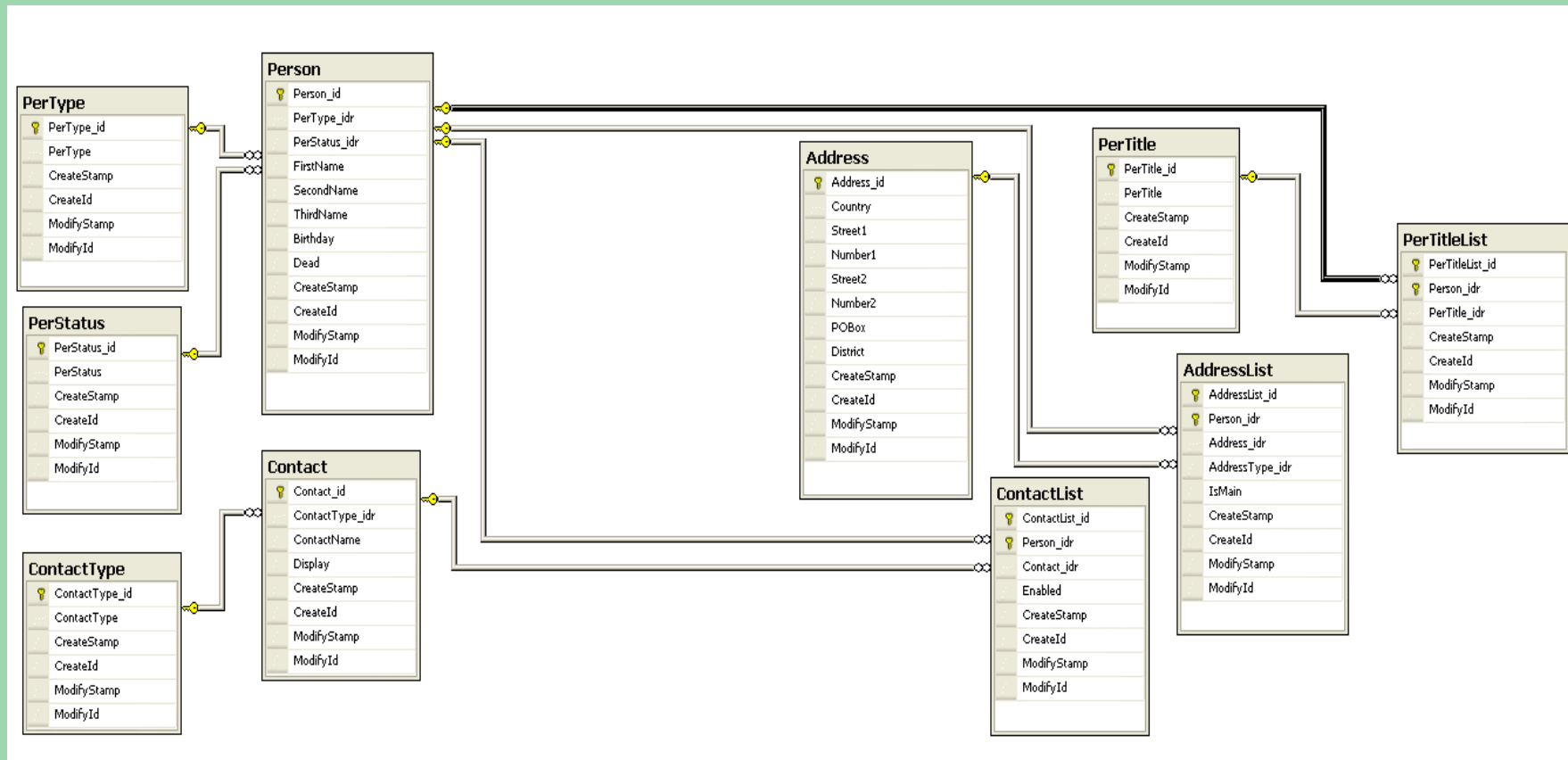
- DBMS, operating system and platform independent
- Generates source code out of a DB Schema (RDBM)
- Supports following DB Objects:
 - Tables
 - Views
 - All SQL92 Data Types
 - Domain Types
 - Defaults
 - Unique und Index Fields
- Fully supports the ACID Transaction Model
- Supports both Optimistic Save and Pessimistic Save (Timestamp comparison)

cplusDBAction Features (2/2)

- Automatically generates:
 - Primary Keys (PK) and Constraints PK
 - Row Timestamps and Row User Stamps for inserted and updated records
- Generates DDL, DML und DQL Export/Import functions (also supporting DB-Provider dialects)
- Provides PK and FK (Foreign Key) Navigation methods (to read/edit the corresponding child or parent records)
- Provides Query Builder Methods for the creation of complex queries
- Checks the consistency of queries at compilation time
- Software developers do not have to write SQL queries

cplusDBAction: Examples

- A DBTableSet Library named “PersExampSet” is created from the following Database Schema:



cplusDBAction: DBTableSet

- The PersExampSet DBTableSet is created using the following command:

```

1
2 //here the command on a window machine
3 DBCodeWriterv3d.exe -P10 //here we identifier the MsSQL Server driver
4 -sdbo //schema name
5 -hlocalhost //server location
6 -usa //DB user
7 -iadmin01 //db password
8 -dPersExamp //database name
9 -tPersExampSet //DBTableSet name
10 -I1 //DBTableSet version
11 -EC:\Source\persExampSet //location where the code will be stored
12
13 //here the command on a linux machine
14 DBCodeWriterv3d -P14
15 -sdbo
16 -hlocalhost
17 -usa
18 -iadmin01
19 -dPersExamp
20 -tPersExampSet
21 -I1
22 -E/home/john/Source/PersExampSet

```

cplusDBAction: Open a Session

```
3 //the user DBTableSet object
4 //this should be global or member from some class
5 TableSetPersExampSet::clsPersExampSet* pObjPersExampSet=NULL;
6 //for all DBSettings
7 DBTableSet::clsDBSettings objDBSet;
8 //to hold the active environment
9 DBEnvironment::clsDBEnvironment objDBEnv;
10 int iRetVal=DBBaseObject::eRetCodeUnknownError;
11 DBString::clsDBString strDBErrInfo="";
12 //here we initialize the connection settings
13 //DBMS Server name
14 objDBSet.SetHostName("Host Address");
15 //if needed a port
16 objDBSet.SetPort(1432);
17 //the db user name
18 objDBSet.SetUserName("user");
19 //the db password
20 objDBSet.SetPassword("admin01");
21 //the db schema name
22 objDBSet.SetSchemaName("tiger");
23 //the db name
24 objDBSet.SetDatabaseName("PersExamp");
25 //driver selection for oracle
26 objDBSet.SetDriverType(DBEnvironment::eDBOCI);
27 //User DBTableSet selection (created with DBCodeFactory tool)
28 objDBSet.SetTableSetName("PerExampSet");
29 //PerExampSet version
30 objDBSet.SetTableSetVersion(1);
31 //here now we open the database
32 iRetVal=TableSetPersExampSet::clsPersExampSet::Open(&objDBSet,&objDBEnv, &pObjPersExampSet, strDBErrInfo);
33 if(iRetVal>=DBBaseObject::eRetCodeSuccess){
34     //the connection is open and ready for working
35 }
```

cplusDBAction: Close a Session

```
4  
5     iRetVal=TableSetPersExampSet::clsPersExampSet::Close(&objDBEnv, &pobjPersExampSet, strDBErrInfo);  
6     if(iRetVal>=DBBaseObject::eRetCodeSuccess){  
7         //the connection is closed and all the object are removed  
8     }
```


cplusDBAction: Read Row Data

```

5  DBString::clsDBString strSQLQueryTmp;
6  DBString::clsDBString strSQLQuery;
7  int iRetVal=DBBaseObject::eRetCodeUnknownError;
8  TableSetPersExampSet::clsCollPerson objPersonContainer;
9  TableSetPersExampSet::clsPerson* pObjPersonItem=NULL;
10
11  iRetVal=pObjPersExampSet->FieldQuery(TableSetPersExampSet::CTBL_Person,
12                                     TableSetPersExampSet::clsPerson::CFL_FirstName,
13                                     DBTableSet::EN_OPER_EQUAL,
14                                     "John",
15                                     strSQLQueryTmp);
16
17  strSQLQuery=strSQLQueryTmp;
18  iRetVal=pObjPersExampSet->FieldQuery(TableSetPersExampSet::CTBL_Person,
19                                     TableSetPersExampSet::clsPerson::CFL_SecondName,
20                                     DBTableSet::EN_OPER_EQUAL,
21                                     "Mercury",
22                                     strSQLQueryTmp);
23
24  strSQLQuery+=" AND ";
25  strSQLQuery+=strSQLQueryTmp;
26  iRetVal=TableSetPersExampSet::clsPerson::Select(pObjPersExampSet, &objPersonContainer, strSQLQuery);

```

- The Select method provides several other optional parameter like RowCount, OrderBy, FieldSelect, Distinct
- Sub-selects can be created using the FieldSubQuery method

cplusDBAction: Data Iteration

```
26 //cursor method
27 objPersonContainer.ResetCursor();
28 while(objPersonContainer.MoveNext()==true){
29     TableSetPersExampSet::clsPerson* pObjPersonItem=NULL;
30     pObjPersonItem=objPersonContainer.Current();
31     std::cout << pObjPersonItem->DB_FirstName()
32               << " is born "
33               << pObjPersonItem->DB_Birthday()
34               << std::endl;
35
36 }
37
38 //direct indexing
39 for(int iPos=0;iPos<objPersonContainer.Count(); iPos++){
40     TableSetPersExampSet::clsPerson* pObjPersonItem=NULL;
41     pObjPersonItem=objPersonContainer.ItemAt(iPos);
42     std::cout << pObjPersonItem->DB_FirstName()
43               << " is born "
44               << pObjPersonItem->DB_Birthday()
45               << std::endl;
46
47 }
```

cplusDBAction: Data Navigation

- Navigation can be done both forward (retrieve all children) as well as backward (retrieve the parent)

```

49 //here we get the AddressList
50 //this is a PK Navigation and we get over the primary key all the children element
51 //here we declare just a container
52 TableSetPersExampSet::clsCollAddressList* pObjCollAddressList=NULL;
53 pObjCollAddressList=pObjPersExampSet->PK_AddressList_Person_idr();
54 pObjCollAddressList->ResetCursor();
55 while(pObjCollAddressList->MoveNext()==true){
56     TableSetPersExampSet::clsAddressList* pObjAddressList=NULL;
57     pObjPersonItem=objPersonContainer.Current();
58     std::cout << pObjAddressList->DB_IsMain()
59         << " for "
60         << pObjPersonItem->DB_FirstName()
61         << std::endl;
62 }
63
64 //here we get the PerType
65 //this is a FK Navigation and we get over the foreign key his parent
66 //here we declare just an item
67 TableSetPersExampSet::clsPerType* pObjPerType=NULL;
68 pObjPerType=pObjPersExampSet->FK_PerType_idr();
69 std::cout << pObjPerType->DB_PerType()
70     << " for "
71     << pObjPersonItem->DB_FirstName()
72     << std::endl;
73

```

cplusDBAction: Data Changes (1)

- An example of an Update, an Insert and a Delete:

```

78 //here we modify data
79 TableSetPersExampSet::clsCollPerson* pObjCollPerson=NULL;
80 pObjCollPerson=TableSetPersExampSet::clsPerson::Update(pObjPersExampSet, strSQLQuery);
81 pObjPersonItem=objPersonContainer.Current();
82 pObjPersonItem->DB_FirstName("NewJohn",DBTableSet::EN_FLD_CONTENT_EDITED_NORMAL);|
83 iRetVal=pObjPersExampSet->Save();
84
85 //here we insert records
86 TableSetPersExampSet::clsCollPerson* pObjCollPerson=NULL;
87 pObjCollPerson=TableSetPersExampSet::clsPerson::Insert(pObjPersExampSet);
88 pObjPersonItem=objPersonContainer.Add();
89 pObjPersonItem->DB_FirstName("FirstName",DBTableSet::EN_FLD_CONTENT_EDITED_NEW);
90 pObjPersonItem->DB_SecondName("SecondName",DBTableSet::EN_FLD_CONTENT_EDITED_NEW);
91 //here we connect the relation to the PerType Table
92 pObjPersonItem->FK_PerType(pObjPerType,DBTableSet::EN_FLD_CONTENT_EDITED_NEW);
93 //and so on
94 iRetVal=pObjPersExampSet->Save();
95
96 //here we delete records
97 TableSetPersExampSet::clsCollPerson* pObjCollPerson=NULL;
98 pObjCollPerson=TableSetPersExampSet::clsPerson::Delete(pObjPersExampSet, strSQLQuery);
99 iRetVal=pObjPersExampSet->Save();

```

cplusDBAction: Data Changes (2)

- The same example, but with a Save at the end (which also performs a *consistency check* to see if the data was meanwhile modified by another process):

```

77 //here we modify data
78 TableSetPersExampSet::clsCollPerson* pobjCollPerson=NULL;
79 pobjCollPerson=TableSetPersExampSet::clsPerson::Update(pobjPersExampSet, strSQLQuery);
80 pobjPersonItem=objPersonContainer.Current();
81 pobjPersonItem->DB_FirstName("NewJohn",DBTableSet::EN_FLD_CONTENT_EDITED_NORMAL);
82
83 //here we insert records
84 TableSetPersExampSet::clsCollPerson* pobjCollPerson=NULL;
85 pobjCollPerson=TableSetPersExampSet::clsPerson::Insert(pobjPersExampSet);
86 pobjPersonItem=objPersonContainer.Add();
87 pobjPersonItem->DB_FirstName("FirstName",DBTableSet::EN_FLD_CONTENT_EDITED_NEW);
88 pobjPersonItem->DB_SecondName("SecondName",DBTableSet::EN_FLD_CONTENT_EDITED_NEW);
89 //here we connect the relation to the PerType Table
90 pobjPersonItem->FK_PerType(pobjPerType,DBTableSet::EN_FLD_CONTENT_EDITED_NEW);
91 //and so on
92
93 //here we delete records
94 TableSetPersExampSet::clsCollPerson* pobjCollPerson=NULL;
95 pobjCollPerson=TableSetPersExampSet::clsPerson::Delete(pobjPersExampSet, strSQLQuery);
96
97 DBString::clsDBStringList strTableList;
98 strTableList.add(TableSetPersExampSet::CTBL_Person);
99 iRetVal=pobjPersExampSet->Save(strTableList,DBTableSet::EN_DBSAVE_PESSIMISTIC);

```