DØOM Review

Database and MSQL Interface

H. Greenlee

Mar. 4, 1998

Outline

- Why MSQL?
- How the C++ data model maps onto the relational data mode.
- MSQL database schema.
- Database I/O interfaces.
- MSQL interface classes.
- Updating MSQL objects.
- Two MSQL implementations.
- Using the MSQL interface.

Why MSQL

- A free relational database (distributable via d0library).
- Prototype for other relational databases (e.g. Oracle). SQL should be nearly the same for different databases.

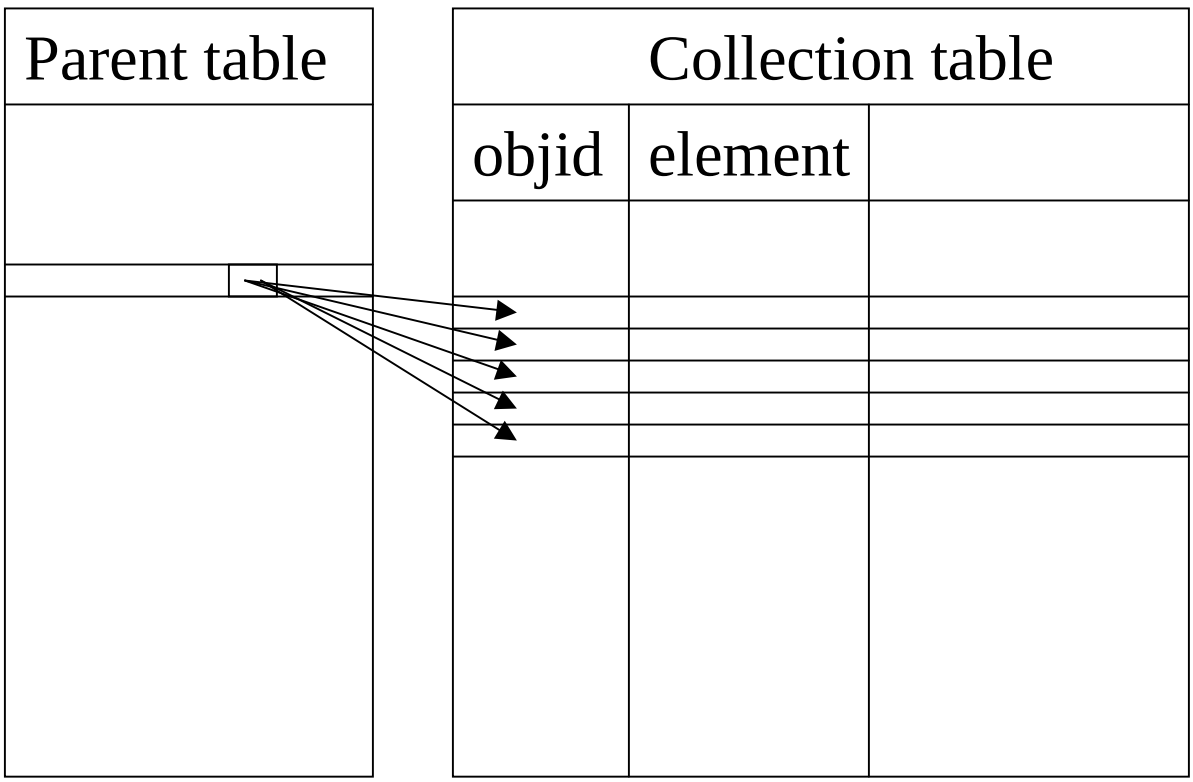
Relational Data Model

- Relational databases organize data into tables with rows and columns.
- Columns contain predefined atomic types.
  - MSQL atomic types:
    - Fixed length character.
    - Variable length character.
    - Signed integer
    - Unsigned integer
    - Real
    - Date
    - Time
    - Money
- Basic mappings

C++ Class	⇒ Database table.
Instance of class	⇒ Row of table.
Field of class	⇒ Column of table.
Pointer	⇒ Index.

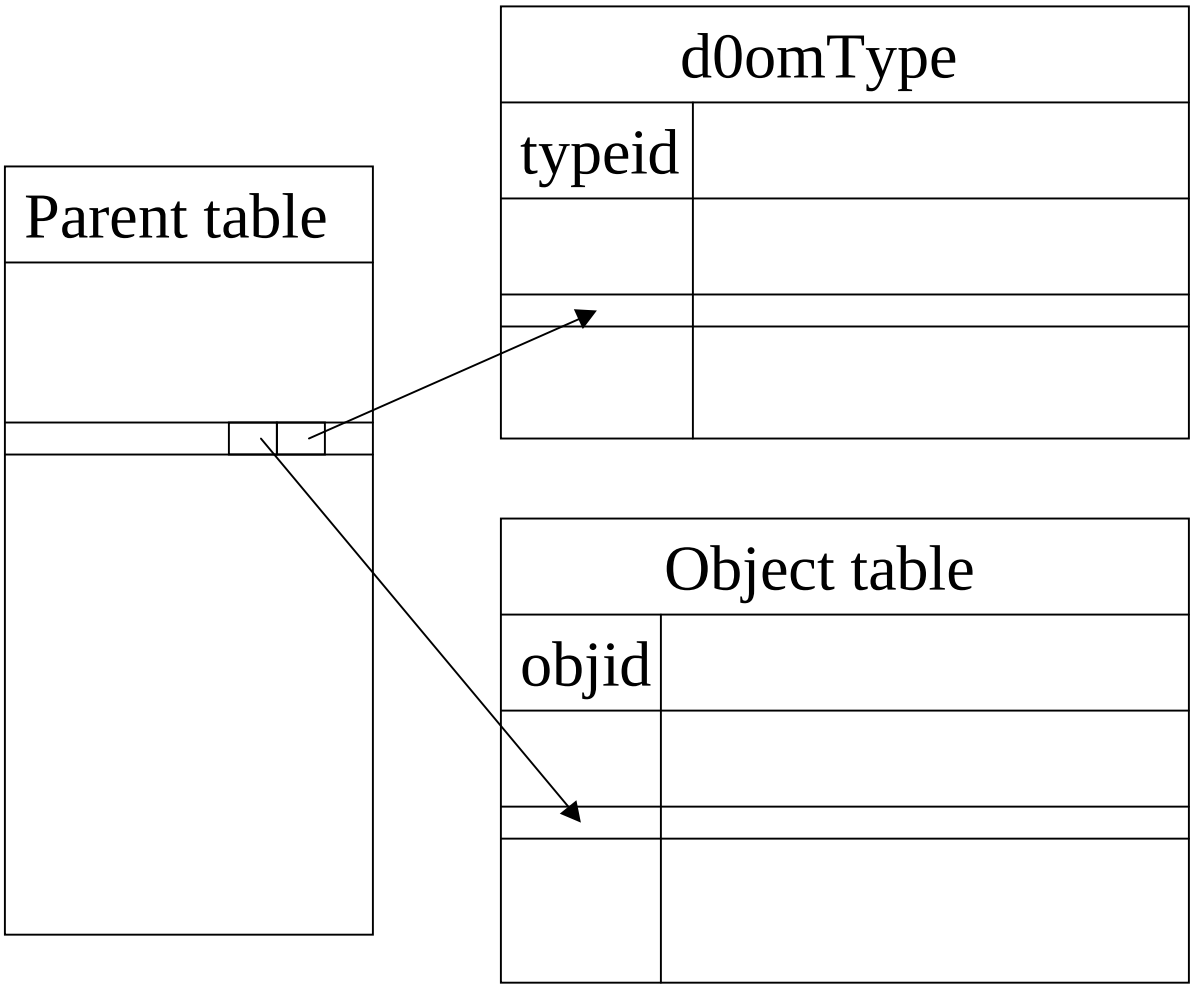
Collections and Arrays

- C++ collection classes (STL, etc.) and built-in arrays.
- Statically typed.
- Single index creates one-to-many association.



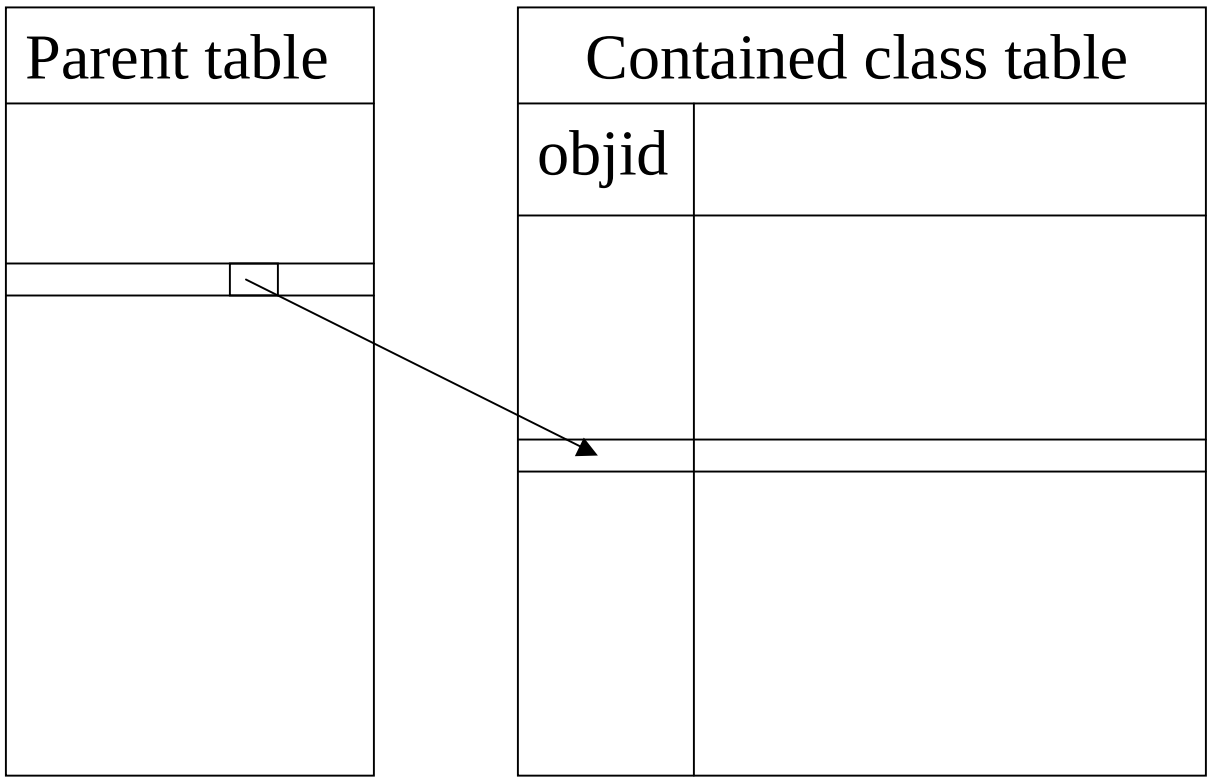
Pointers and References

- Bare C++ pointers, DØOM references (d0_Ref<T>), auto_ptr<T>.
- Dynamically typed.
- Separate indices for object and dynamic type.



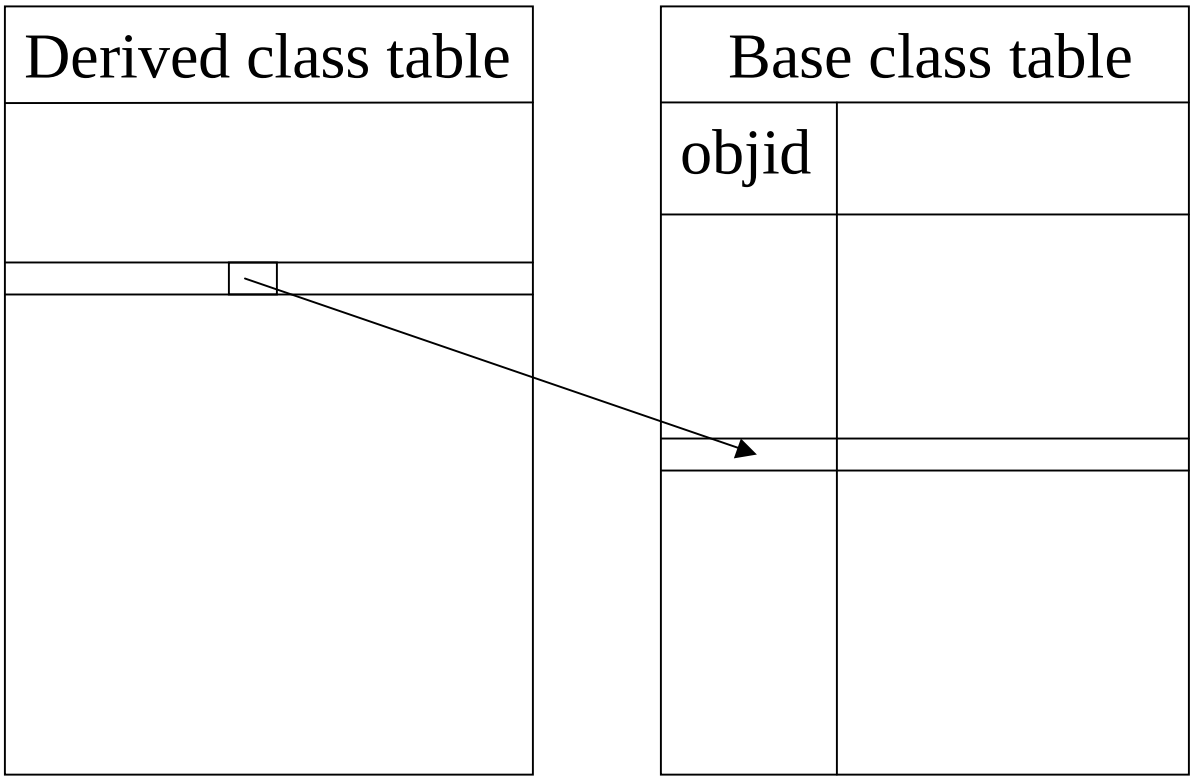
Contained Classes

- C++ by value containment.
- Statically typed
- Single index creates association between parent (containing) and contained table.



Inheritance

- Base classes are treated like contained classes.



Schema Flattening

- The following kinds tables can be eliminated by incorporating their data directly into a parent table.
  - Contained classes.
  - Base classes.
  - Fixed length arrays.
- Schema flattening is generally advantageous, and is done by the MSQL interface wherever possible, except for long fixed length arrays.

MSQL Tables and Metadata

- Types of MSQL tables
  - Object tables (for classes deriving from d0_Object).
  - Non-object tables (for classes that do not derive from d0_Object).
  - Collection and array tables.
  - Type table
  - Key table.

Object Tables

- For classes that derive from d0_Object.
- Metadata
  - objid - A unique key index.
  - refcount - How many pointers, references, and keys are pointing at me?

Object Table		
objid	refcount	data...

Non-Object Tables

- For contained classes and base classes that do not derive from d0_Object.
- Same as object tables, except no reference count column.
- Metadata
  - objid - A unique key index.

Non-Object Table	
objid	data...

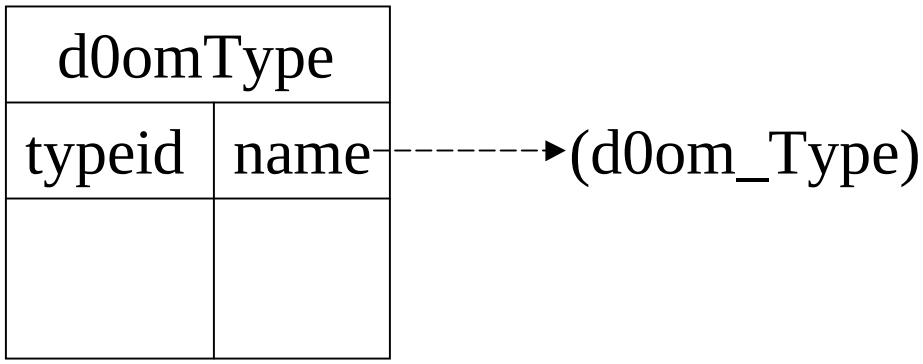
Collection and Array Tables

- For DØOM and STL collection classes and C++ built-in fixed length arrays.
- Metadata
  - objid - A non-unique key index.
  - element - Element number.

Collection Table		
objid	element	data...

Type Table

- A list of known data types (i.e. C++ classes).
- Metadata.
  - typeid - A unique key index.
  - name - Corresponding C++ class name (can be used to locate d0omType class)..



Key Table

- For DØOM keys.
- Metadata.
  - keyid - A unique key index.
  - name - String representation of key.
  - typeid - Type of object associated with key.
  - objid - Object id. of object associated with key.

Key Table			
keyid	name	typeid	objid

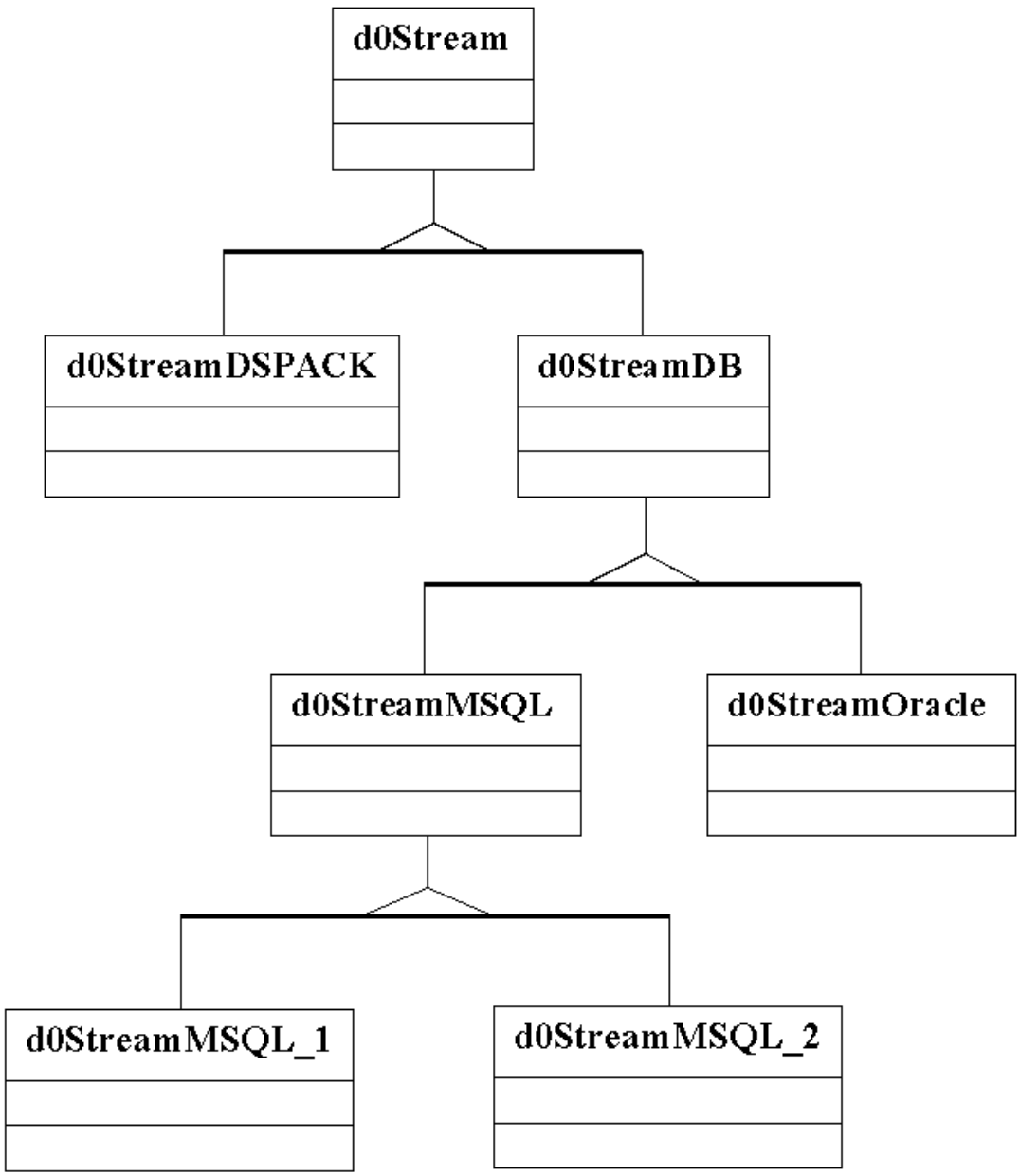
MSQL Name Mangling

- MSQL table and column names are derived from C++ class and field names with name mangling.
- Rules:
  - Special characters (including underscore) are replace by upper case “X.”
  - C++ names are prepended by “XX.”
  - Special rules for the following cases:
    - Dynamic type columns.
    - Array tables.
    - Base classes.
    - Schema flattening.

Database I/O Interface

- Keyed I/O interface via class d0Stream.
  - Same interface as used for DSPACK I/O (except that keys are interpreted differently).
- Database specific I/O operations handled by class d0StreamDB.
  - Queries.
  - Transactions.
- Either d0Stream or d0StreamDB can be instantiated by d0StreamFactory (as described two weeks ago).

d0Stream Class Diagram



Keyed I/O Interface

- Methods of d0Stream:
  - d0_Ref_Any read(const d0Key*);
    - Standard keyed read.
  - void write(const d0_Object&);
    - First time write.
    - Update write (for existing objects).
  - bool bind(d0_Object&, const d0Key&);
    - Associate key with object.
    - Can be called before or after object is written to MSQL.
  - void unbind(const d0Key&);
    - Break association between key and object.
  - int close()
    - Close database.

Database Specific I/O Interface

- Methods of d0StreamDB:
  - void open_transaction(bool readonly);
  - commit_transaction();
  - abort_transactio();
    - MSQL does not support transactions.
  - virtual d0_List<d0_Ref_Any> query(
    const d0_String& classname,
    const d0_String& condition,
    const d0_String& tables = “”,
    const d0_String& fields = “”);
    - Construct query:
      select objid, *columns* from *tables condition*.
    - Condition argument would typically contain a where clause.
    - Use $(…) notation in condition argument to invoke standard MSQL name mangling on C++ names.

MSQL Interface Classes

- Stream classes
  - d0StreamMSQL
  - d0StreamMSQL_1
  - d0StreamMSQL_2
- Type classes
  - d0om_MSQL_Type
  - d0om_MSQL_Type_Def
- Indirect pointer classes
  - d0om_MSQL_Indptr
  - d0om_MSQL_Loc
- Operations on collections and arrays.
  - d0om_MSQL_Collection_Iterator
  - d0om_MSQL_Collection_Iterator_Input
  - doom_MSQL_Collection_Iterator_Output
  - doom_MSQL_Collection_Iterator_Update
  - d0om_MSQL_Array_Adapter

MSQL Interface Classes (cont.)

- Iterator for MSQL query results.
  - d0om_MSQL_Result_Iterator
  - d0om_MSQL_Result_Iterator_1
  - d0om_MSQL_Result_Iterator_2

Class d0StreamMSQL

- Class d0StreamMSQL and its subclasses, d0StreamMSQL_1 and d0StreamMSQL_2 implement basic public I/O methods, including read, write, query, bind, unbind.
- Classes d0StreamMSQL_1 and d0StreamMSQL_2 encapsulate most of the MSQL c api.
- Data:
  - Maps associating keys with objects in memory.
  - A map containing object id.'s of all previously written objects.
  - Output filter (if any).
  - List of database tables and fields.
  - Database file information.

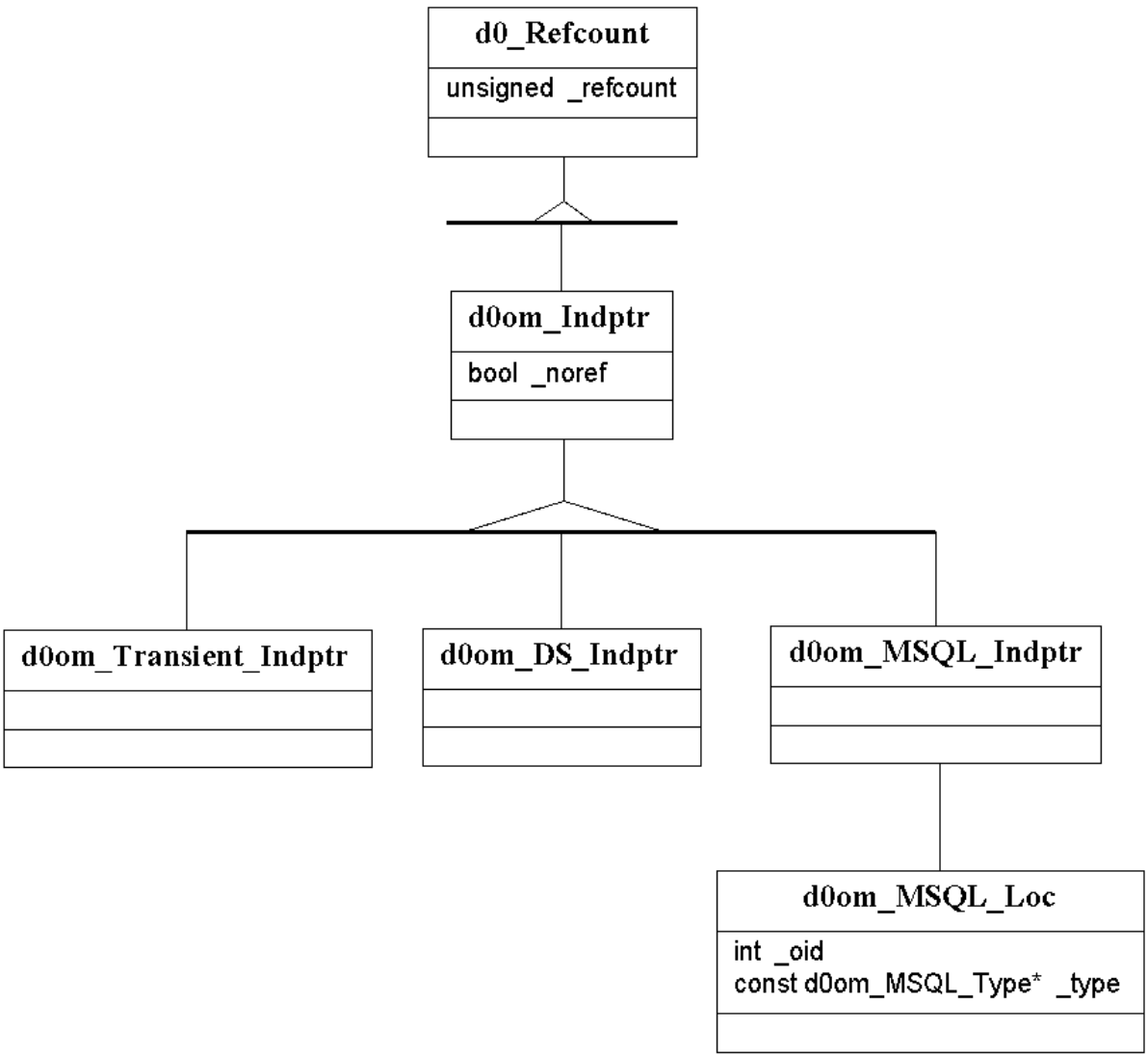
Class d0om_MSQL_Type

- Classes d0om_MSQL_Type encapsulates DØOM (via d0om_Type) and MSQL type information.
- Data:
  - Database schema information for this table.
  - MSQL type id.
  - One instance of d0om_MSQL_Type_Def
    - Pointer to corresponding d0om_Type.
    - Dimensionality (=1 for non-array types).
    - Pointer to open d0StreamMSQL.
- Methods.
  - d0om_Type interface is completely encapsulated within d0om_MSQL_Type.
  - Numerous other type-dependent methods.

Class d0om_MSQL_Indptr

- MSQL indirect pointer class derived from d0om_Indptr.
- d0om_MSQL_Indptr contains enough information, via d0om_MSQL_Loc, to uniquely identify a database object:
  - Object id.
  - Type (pointer to d0om_MSQL_Type).
- Class d0om_MSQL_Indptr allows true deferred I/O.
- All concrete indirect pointer classes must override method makeobj, which is called to instantiate C++ objects in memory.

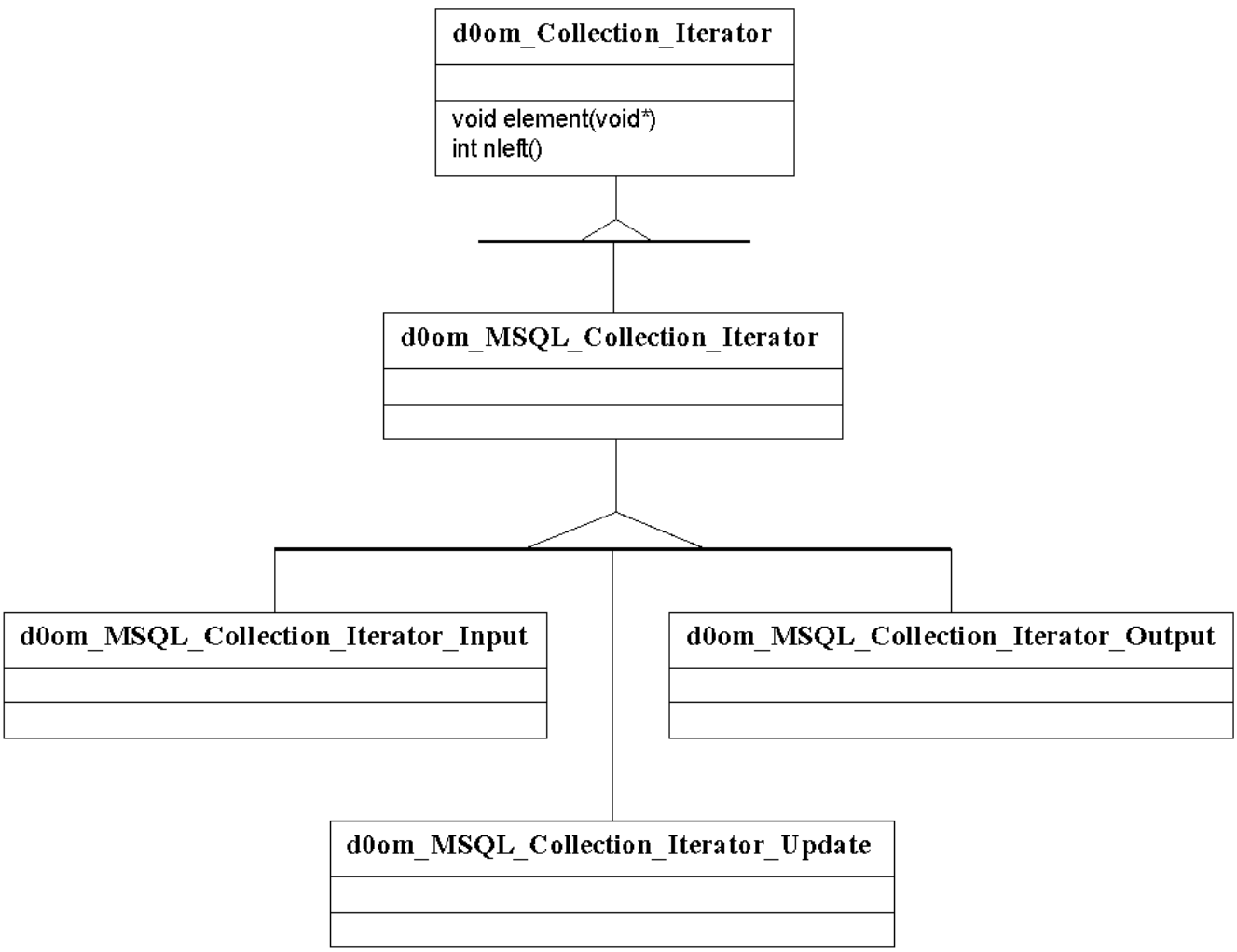
d0om_MSQL_Indptr Class Diagram



Class d0om_MSQL_Collection_Iterator

- Classes
  - d0om_MSQL_Collection_Iterator
  - d0om_MSQL_Collection_Iterator_Input.
  - d0om_MSQL_Collection_Iterator_Output.
  - d0om_MSQL_Collection_Iterator_Update.
- Used for operating on each element of a collection or array.
- These classes derive from the DØOM class d0om_Collection_Iterator.
- They are required to override the following methods:
  - nleft() - Number of elements remaining to iterate.
  - element(void*) - Called for each element of collection or array.

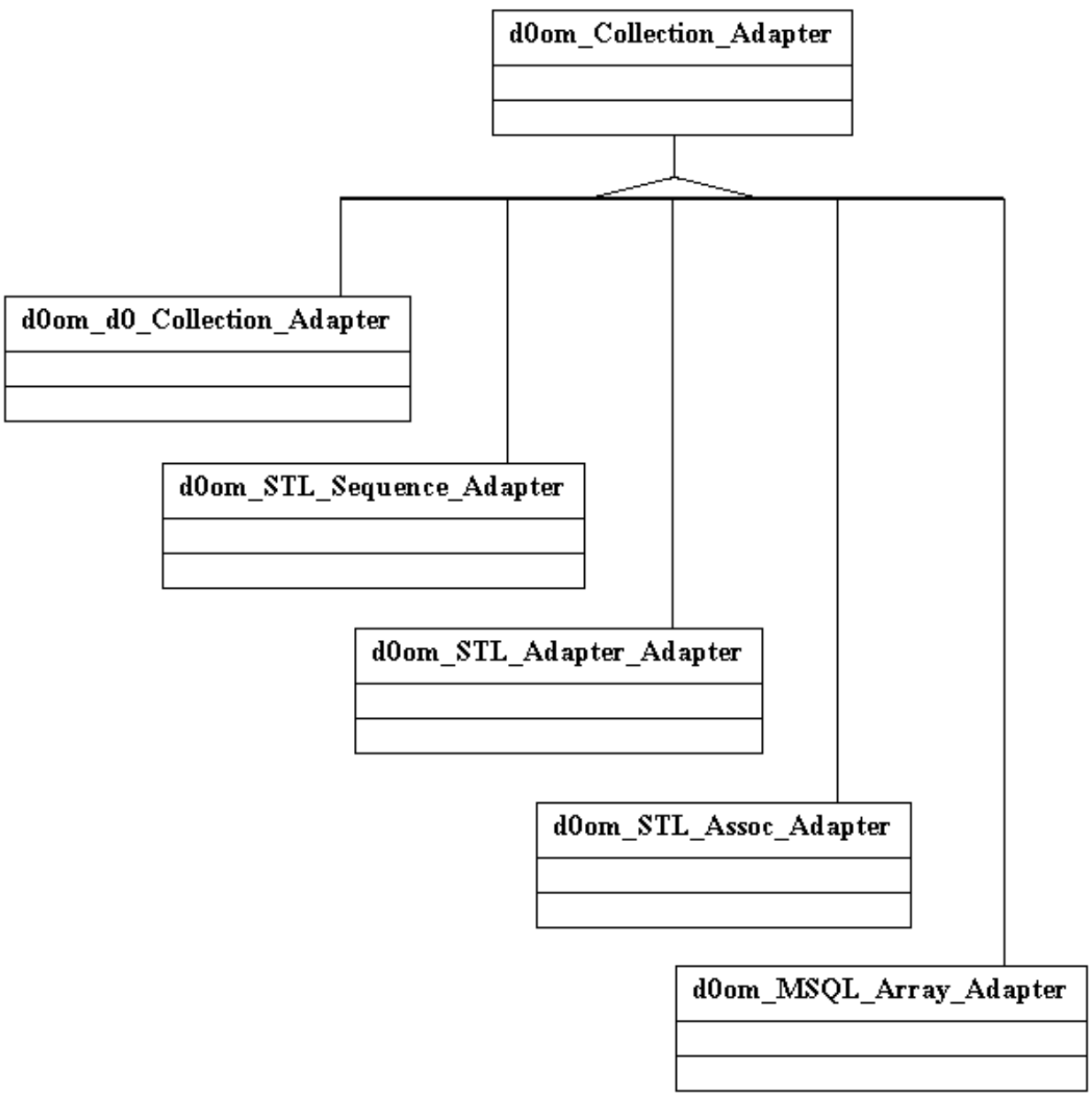
d0om_MSQL_Collection_Iterator Class Diagram



d0om_MSQL_Array_Adapter

- Adapter classes derive from DØOM class d0om_Collection_Adapter.
- The job of adapter classes is to iterate over collections and arrays, and to invoke d0om_Collection_Iterator::element for each element.
- DØOM supplies adapter classes for DØOM and STL collections.
- Class d0om_MSQL_Array_Adapter is an adapter class for fixed length arrays.

d0om_MSQL_Array_Adapter Class Diagram



d0om_MSQL_Result_Iterator

- Class d0om_MSQL_Result_Iterator is an iterator class for results of MSQL queries.
- Classes d0om_MSQL_Result_Iterator_1 and d0om_MSQL_Result_Iterator_2 encapsulate the part of the MSQL c api pertaining to getting the results of queries.

Updating MSQL objects

- If an object id. is already associated with an object passed to method d0Stream::write, then the object is written in update mode.
- An object id. can be associated from a previous read (via d0om_MSQL_Indptr) or from a previous write (via d0StreamMSQL memory cache).
- Reference counts were introduced into the MSQL database schema to facilitate updating.
- Updating is not an issue for DSPACK.

Updating MSQL Objects (cont.)

- All object id.'s are recycled, if possible, including the object id. of the original object.
- The object in memory is compared to the corresponding object in the database (the database object is reread).
- Only those fields that have been modified are written out.
- Links from modified objects are followed.
- Links from unmodified objects are not followed.
- Allowed modifications:
  - Modified atomic fields.
  - Swapping pointed to objects.
  - Changing contents of contained objects and objects in collections.
  - Changing size of collections.

MSQL Reference Counting.

- Column “refcount” of object tables.
- MSQL reference counting for database resident objects operates similarly to, but independently of, DØOM reference counting for memory resident objects.
- Objects are initially created with a reference count of zero.
- The reference count counts the number of database keys and database references that point to an object.
- Memory references to database objects (DØOM indirect pointers) are not counted.
- If the reference count reaches zero (after having been positive) the object is deleted (possibly causing other reference counts to be decremented, etc.).

Two MSQL Implementations

- The following classes, and no others, are physically dependent on the MSQL c language api.
  - d0StreamMSQL_1
  - d0StreamMSQL_2
  - d0om_MSQL_Result_Iterator_1
  - d0om_MSQL_Result_Iterator_2
- Classes X_1 use the standard MSQL database implementation via include:
  - #include “msql.h”
- Classes X_2 use the d0library MSQL database implementation (package MSQL) via include:
  - #include “MSQL/d0msql.h”
- Except for include file, c api is identical.
- Two MSQL implementations are link compatible.

Standard MSQL vs. d0library MSQL

- Standard MSQL:
  - Database server.
  - Global configuration file.
  - Databases owned by administrative user.
  - User access controlled by ACL's.
  - Databases physically located in a single directory.
  - Not supported by FNAL (v2.0.x).
- D0library MSQL.
  - No server.
  - No global configuration file.
  - Databases owned by ordinary users.
  - Access controlled using standard UNIX file permissions.
  - Databases can be located at an arbitrary UNIX pathname.
  - Multiple simultaneous writers not allowed.

Instantiating d0StreamDB using the MSQL Interface

- d0StreamDB is instantiated via method d0StreamFactory::make_d0StreamDB.
- Specify argument streamtype = “MSQL” for standard MSQL implementation.
- Specify argument streamtype = “D0MSQL” for d0library MSQL implementation (recommended).
- Example:

```
d0StreamFactory *factory =
    d0StreamFactory::locateStreamFactory();
d0StreamDB* dbout =
    factory->make_d0StreamDB(fname,
    “D0MSQL”,
    std::ios::out | std::ios::trunc);
```