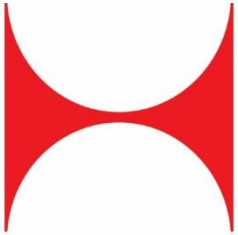
# ABL Buffers and Queries in OO Wrappers

A research report by

Tim Kuehn



TDK CONSULTING

www.tdkcs.com

# Disclaimer…

The material in this presentation is derived from an ongoing research project undertaken by TDK Consulting and is for informational purposes only.

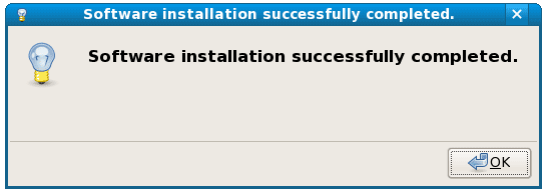
# About TDK Consulting

## TDK Consulting Services

..delivers full life cycle development from a clean sheet of paper



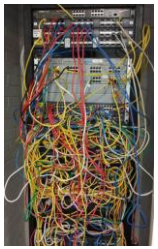
to a finished product




…specializes in the development of software engines

that wrap complex business logic in easy to use APIs

..can refactor code like this:



into code like this:

…develops and delivers training solutions to turn questions

into knowledge and understanding

*"Innovation you can build on."*

*www.tdkcs.com*

# Where it all began..



The Code…

```
DEFINE BUFFER Customer FOR Customer.

DEFINE VARIABLE hCustomer AS HANDLE NO-UNDO.

CREATE BUFFER hCustomer
    FOR TABLE "Customer“.
```

<u>Issues and Opportunities….</u>

- Static declaration for buffers and handles
- Scoped to the procedure, internal procedure, or function
- “Harder” to create and track additional buffers
- Requires manual object lifecycle management
- No OO capabilities
- Not encapsulated

# I have a dream…

The dream….. an OO-Enclosed Buffer:

```
CLASS CustomerDbRecord:
DEFINE PUBLIC PROPERTY CustNum AS INTEGER NO-UNDO
    GET: RETURN(Customer.CustNum).
    END GET.
    SET(iCustNum AS INTEGER):
    ASSIGN Customer.CustNum = iCustNum.
    END SET.

/* Code to setup and control */
/* the record buffer…        */

END CLASS.
```

The advantages…..

- Fully Encapsulated!
- Strong typing
- No passing issues w/in a session
- Can add functionality as needed

The challenges…..

- Fully Encapsulated!
- Different conceptual thought process
- Adding functionality to the right layer
- Scoping generic functionality for a table, record, field, and buffer

Usage:

```
DEFINE VARIABLE oCustomerDbRecord AS CustomerDbRecord NO-UNDO.
oCustomerDbRecord = NEW CustomerDbRecord().
```

# The mission….



Design an OO(ABL) data access structure to provide:

- generic control of a table, record, and buffer,
- a set of OO constructs which encapsulate the ABL "data access" language elements

Required for Implementation:

- Create a set of generic OO base classes for data access and management
- Use the resulting OO objects in an application

That doesn’t sound so hard…..

# Buffer Objects…Oh My!



### Buffer object handle

A handle to a buffer object, corresponding to an underlying ABL buffer, which c
static or dynamic. An example of a static underlying buffer is one you define at c
time by using the DEFINE BUFFER statement, or by implicitly referencing a tabl
ABL construct such as `Customer.CustNum`. An example of a dynamic underlying
is one you create at run time with the CREATE BUFFER statement.

**Syntax**

```
buffer-handle [ :attribute | :method ]
```

*buffer-handle*

An item of type HANDLE representing a handle to a buffer object.

*attribute*

An attribute of the buffer object.

*method*

A method of the buffer object.

**Attributes**

| | | |
|---|---|---|
| ADM-DATA attribute | AFTER-BUFFER attribute | AFTER-ROWID attri |
| AMBIGUOUS attribute | ATTACHED-PAIRLIST attribute | AUTO-DELETE attrib |
| AUTO-SYNCHRONIZE attribute | AVAILABLE attribute | BATCH-SIZE attribut |
| BEFORE-BUFFER attribute | BEFORE-ROWID attribute | BUFFER-GROUP-ID attribute |

(2 of 2)

| | | |
|---|---|---|
| BUFFER-GROUP-NAME attribute | BUFFER-PARTITION-ID attribute | BUFFER-TENANT-ID attribute |
| BUFFER-TENANT-NAME attribute | CAN-CREATE attribute | CAN-DELETE attribute |
| CAN-READ attribute | CAN-WRITE attribute | CRC-VALUE attribute |
| CURRENT-CHANGED attribute | CURRENT-ITERATION attribute (Data Objects) | DATASET attribute |
| DATA-SOURCE attribute | DATA-SOURCE-COMPLETE-MAP attribute | DATA-SOURCE-MODIFIED attribute |
| DATA-SOURCE-ROWID attribute | DBNAME attribute | DYNAMIC attribute |
| ERROR attribute | ERROR-STRING attribute | FILL-MODE attribute |
| HANDLE attribute | HAS-LOBS attribute | INSTANTIATING-PROCEDURE attribute |
| IS-MULTI-TENANT attribute | IS-PARTITIONED attribute | KEYS attribute |
| LAST-BATCH attribute | LOCKED attribute | NAME attribute |
| NAMESPACE-PREFIX attribute | NAMESPACE-URI attribute | NEW attribute |
| NEXT-SIBLING attribute | NUM-CHILD-RELATIONS attribute | NUM-FIELDS attribute |
| NUM-ITERATIONS attribute (data objects) | NUM-REFERENCES attribute | ORIGIN-ROWID attribute |
| PARENT-RELATION attribute | PRIMARY attribute | PRIVATE-DATA attribute |
| QUERY attribute | RECID attribute | RECORD-LENGTH attribute |
| REJECTED attribute | ROWID attribute | ROW-STATE attribute |
| SERIALIZE-NAME attribute | TABLE attribute | TABLE-HANDLE attribute |
| TABLE-NUMBER attribute | TYPE attribute | UNIQUE-ID attribute |
| XML-NODE-NAME attribute | | |

**Methods**

(1 of 2)

| | |
|---|---|
| ACCEPT-CHANGES( ) method | ACCEPT-ROW-CHANGES( ) method |
| APPLY-CALLBACK( ) method | ATTACH-DATA-SOURCE( ) method |
| BUFFER-COMPARE( ) method | BUFFER-COPY( ) method |
| BUFFER-CREATE( ) method | BUFFER-DELETE( ) method |
| BUFFER-FIELD( ) method | BUFFER-RELEASE( ) method |
| BUFFER-VALIDATE( ) method | DETACH-DATA-SOURCE( ) method |
| DISABLE-DUMP-TRIGGERS( ) method | DISABLE-LOAD-TRIGGERS( ) method |

(2 of 2)

| | |
|---|---|
| EMPTY-TEMP-TABLE( ) method | FILL( ) method |
| FIND-BY-ROWID( ) method | FIND-CURRENT( ) method |
| FIND-FIRST( ) method | FIND-LAST( ) method |
| FIND-UNIQUE( ) method | GET-CALLBACK-PROC-CONTEXT( ) method |
| GET-CALLBACK-PROC-NAME( ) method | GET-CHANGES( ) method |
| GET-CHILD-RELATION( ) method | GET-ITERATION( ) method (Data Objects) |
| INDEX-INFORMATION( ) method | MARK-NEW( ) method |
| MARK-ROW-STATE( ) method | MERGE-CHANGES( ) method |
| MERGE-ROW-CHANGES( ) method | RAW-TRANSFER( ) method |
| READ-JSON( ) method | READ-XML( ) method |
| READ-XMLSCHEMA( ) method | REJECT-CHANGES( ) method |
| REJECT-ROW-CHANGES( ) method | SAVE-ROW-CHANGES( ) method |
| SERIALIZE-ROW( ) method | SET-CALLBACK( ) method |
| SET-CALLBACK-PROCEDURE( ) method | SYNCHRONIZE( ) method |
| WRITE-JSON( ) method | WRITE-XML( ) method |
| WRITE-XMLSCHEMA( ) method | – |

| | |
|---|---|
| AFTER-FILL event | AFTER-ROW-FILL event |
| BEFORE-FILL event | BEFORE-ROW-FILL event |
| FIND-FAILED event | OFF-END event |
| ROW-CREATE event | ROW-DELETE event |
| SYNCHRONIZE event | – |

For information on these events, see the "ProDataSet events" section on page 2045.

Buffer-field object handle, CREATE BUFFER statement, DEFINE BUFFER statement, ProDataSet object handle, Query object handle, Temp-table object handle

What have I gotten myself into?

TDK CONSULTING

*"Innovation you can build on."*

# Mapping Buffer Attributes



Mapping a read-only attribute to an OO property:

```
DEFINE PUBLIC PROPERTY BufferHandle AS HANDLE NO-UNDO GET. PRIVATE SET.
```

Invariant properties are set in the class constructor

Mapping a read/write attribute to an OO property:

```
DEFINE PUBLIC PROPERTY SerializeName AS CHARACTER NO-UNDO
      GET: RETURN(THIS-OBJECT:BufferHandle:SERIALIZE-NAME).
      END GET.
      SET(chString AS CHARACTER):
         ASSIGN THIS-OBJECT:BufferHandle:SERIALIZE-NAME = chString.
      END SET.
```

(make sure to get INTEGER and INT64's right!)

# Mapping Buffer Methods



Mapping a buffer method to an OO method:

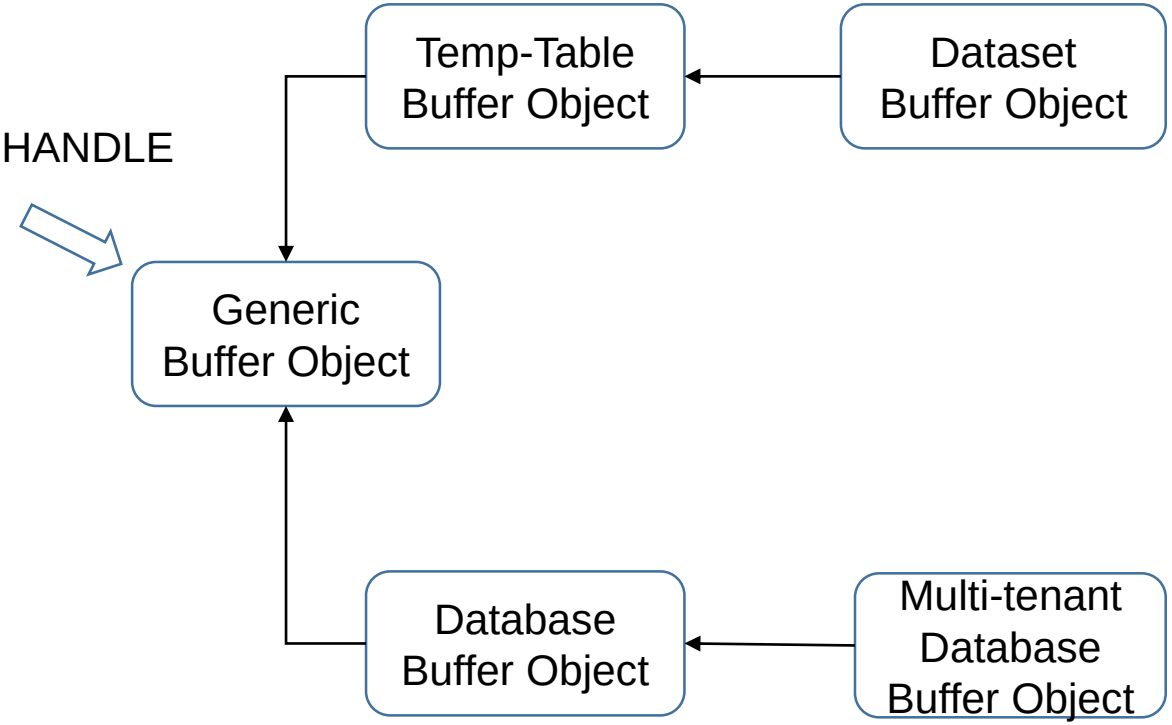
```
METHOD PUBLIC LOGICAL BufferCompare(hSrcBuffer AS HANDLE):
RETURN(THIS-OBJECT:BufferHandle:BUFFER-COMPARE(hSrcBuffer)).
END METHOD.

METHOD PUBLIC LOGICAL BufferCopy(hSrcBuffer AS HANDLE):
RETURN(THIS-OBJECT:BufferHandle:BUFFER-COPY(hSrcBuffer)).
END METHOD.
```

# The things you find…



A hierarchy of “buffer objects”:

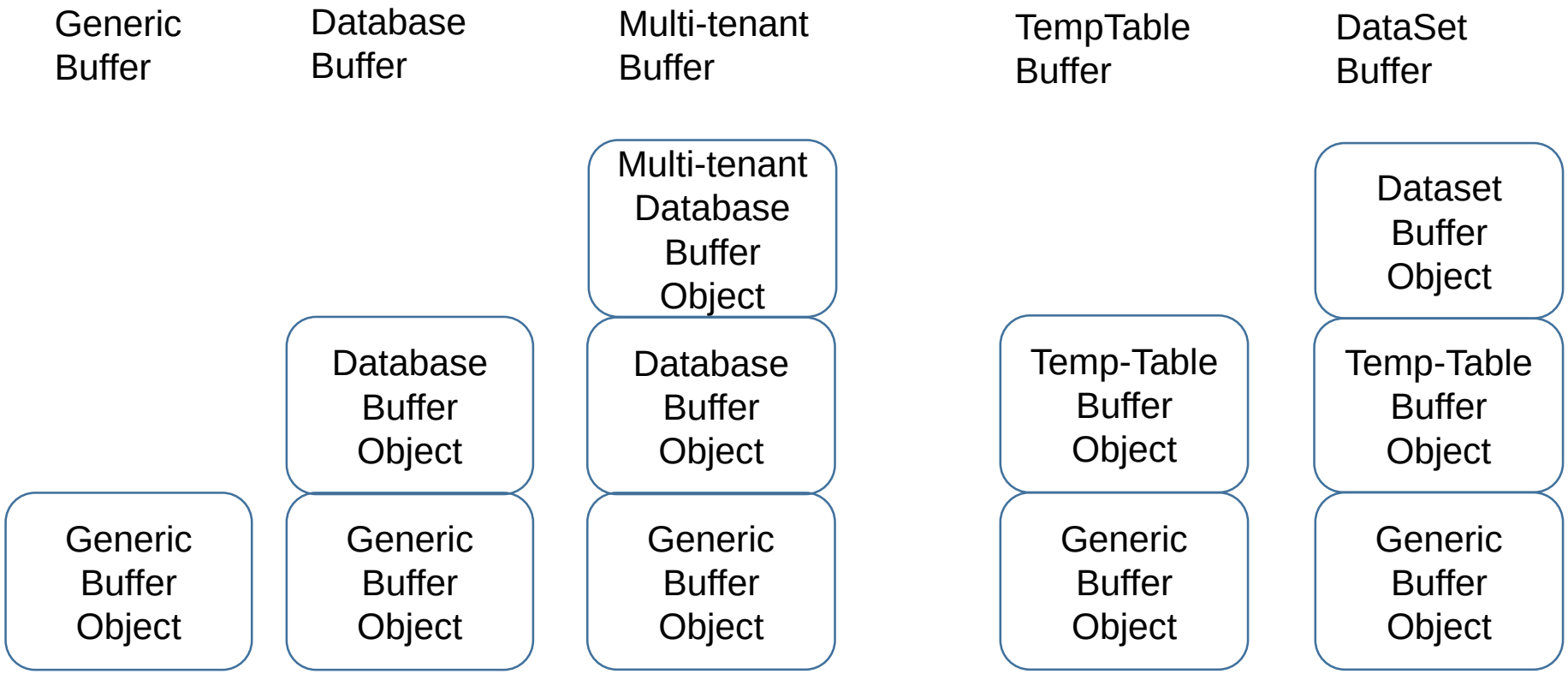
BUFFER TableName:BUFFER-HANDLE



# Buffer Object Hierarchy



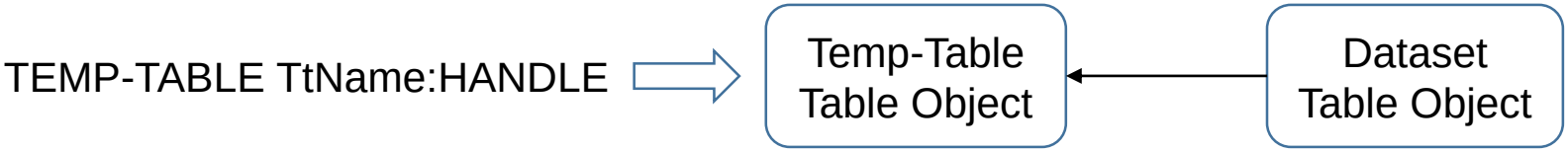
Mapping the “buffer object” hierarchy to OO buffer classes:



# The things you find…



A hierarchy of “Temp Table Objects”:



# Modelling FIND()



## How to model LOCK and WAIT mode specifications?

**Syntax**

```
FIND-BY-ROWID ( rowid
  [ ,
    { SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK }
    [ , NO-WAIT ]
  ] )
```

*rowid*

An expression of type ROWID that represents the rowid of the desired record.

SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK

The type of lock that the AVM places on the record, if found. The default is SHARE-LOCK.

**Note:** For more information on record locks, see *OpenEdge® Getting Started: ABL Essentials.*

NO-WAIT

Causes FIND-BY-ROWID to return FALSE immediately if another user has a lock on the desired record and FIND-BY-ROWID specifies a locking option other than NO-LOCK.

**Note:** To determine whether another user has a lock on the desired record, use the LOCKED attribute of the buffer object.

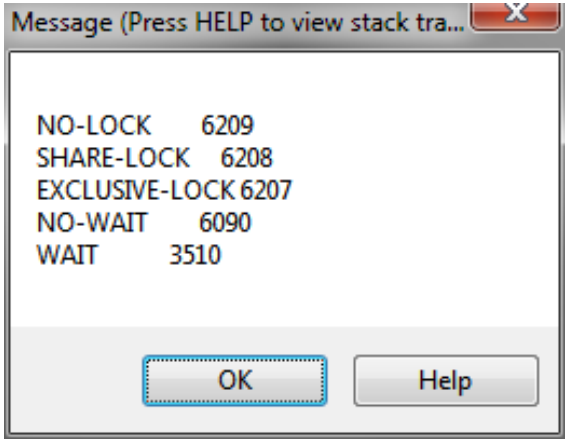
Text is from the 11.5 docs…

# Specifying Lock and Wait

NO-LOCK, SHARE-LOCK, EXCLUSIVE-LOCK, and NO-WAIT are all “integers”.

```
DEFINE VARIABLE iWait AS INTEGER NO-UNDO.

ASSIGN iWait = 3510.

 MESSAGE "NO-LOCK       "   INTEGER(NO-LOCK)        SKIP
         "SHARE-LOCK    "   INTEGER(SHARE-LOCK)     SKIP
         "EXCLUSIVE-LOCK"   INTEGER(EXCLUSIVE-LOCK) SKIP
         "NO-WAIT       "   INTEGER(NO-WAIT)        SKIP
         "WAIT          "   INTEGER(iWait)
     VIEW-AS ALERT-BOX.
```

Because WAIT doesn’t exist as a language element…and Gus says this is the value it has



# Should you be afraid?

Are you afraid yet?

# Wizard Warning!

**Gus Bjorklund** <gus@progress.com>

to peg

you are hereby warned:

the number 6090 is subject to change without notice. in the next major release, it may be some other number.

-gus

# Locking and Waiting



## Lock and Wait Mode Classes

Used to store and pass lock/wait specifications to FIND and related methods

| | |
|---|---|
| LockModeDefinition.cls | Holds all possible locking types and validates a lock mode value is legal. |
| LockMode.cls | Locking mode specification implemented as a value object. |
| WaitModeDefinition.cls | Holds all possible wait types and does validation testing on wait modes |
| WaitMode.cls | Wait mode specification implemented as a value object. |
| LockWaitStatic.cls | A collection of static properties with all possible lock and wait conditions |

# Implementing FIND()s



Finding a record using -

Language default is “SHARE-LOCK, WAIT”

...the default lock/wait specification:

```
METHOD PUBLIC LOGICAL FindByRowid(riRow AS ROWID):
RETURN(THIS-OBJECT:BufferHandle:FIND-BY-ROWID(riRow)).
END METHOD.
```

Passing an OO LockMode instance to specify how to lock the record

... a lock specification:

```
METHOD PUBLIC LOGICAL FindByRowid(riRow AS ROWID, oLockMode AS LockMode):
RETURN(THIS-OBJECT:BufferHandle:FIND-BY-ROWID(riRow, oLockMode:LockMode)).
END METHOD.
```

# Implementing FIND()s



Finding a record using a lock and wait mode specification:

```
METHOD PUBLIC LOGICAL FindByRowid(riRow     AS ROWID,
                                  oLockMode AS LockMode,
                                  oWaitMode AS WaitMode):

RETURN(IF oWaitMode:WaitMode = oWaitModeDefinitions:Wait
        THEN THIS-OBJECT:FindByRowid(riRow, oLockMode)
        ELSE THIS-OBJECT:BufferHandle:FIND-BY-ROWID(riRow,
                                                    oLockMode:LockMode,
                                                    oWaitMode:WaitMode
                                                    )
        ).
END METHOD.
```

Lock / Wait mode OO parameters

If the specification is to wait

Language default is WAIT”

All others calls go here

# Creating Concrete Classes



Accomplished:
Developed a set of classes that maps DB and temp-table buffer handles to a set of *generic* OO buffer object constructs.

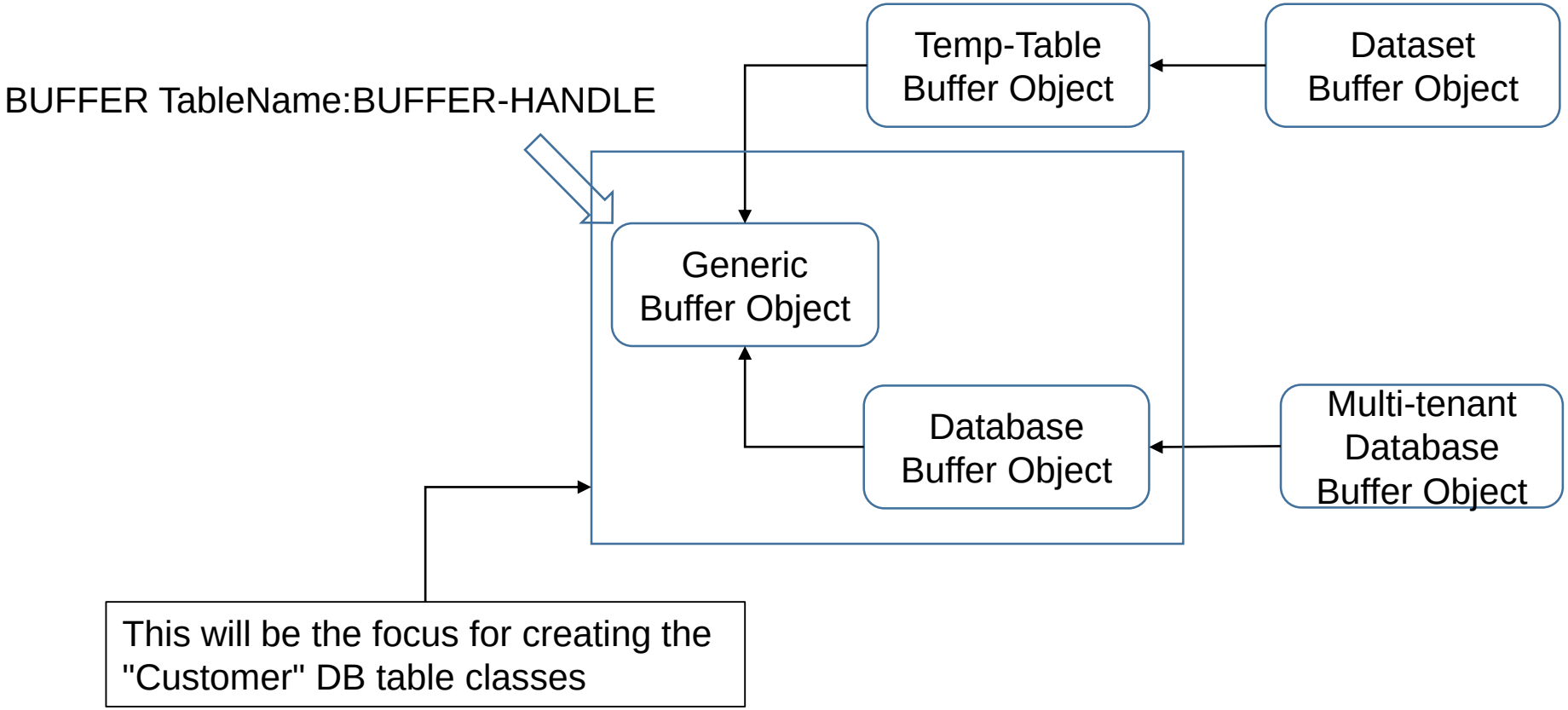
Next Goal:
Use these *generic* OO buffer object constructs to create a set of classes that model *concrete* DB tables, records, and buffers.

First Step:
Map a Sports2000 “Customer” DB table to a set of concrete OO classes.

# DB Buffer Objects



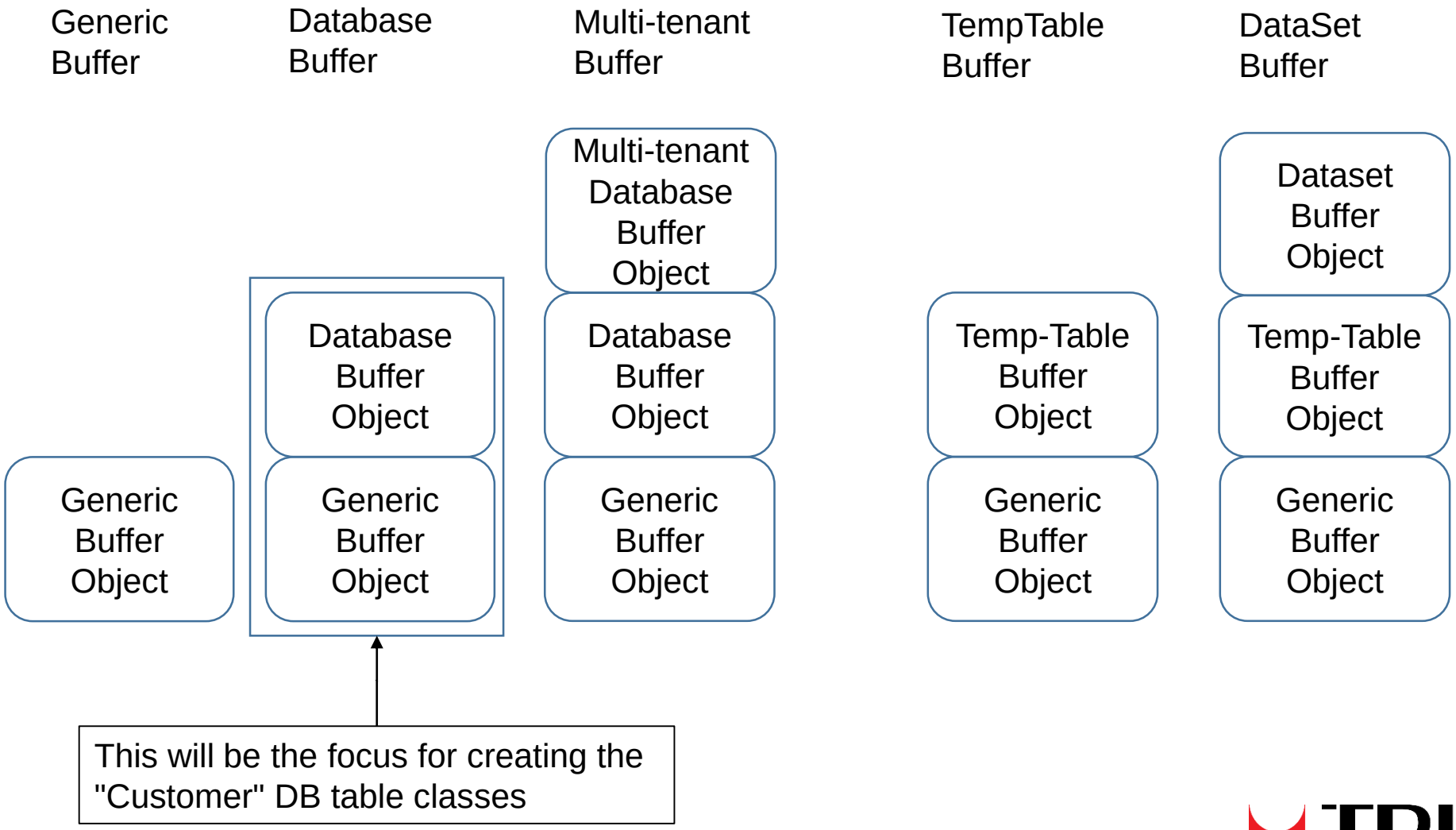
A hierarchy of “buffer objects”:



# Buffer Object Hierarchy



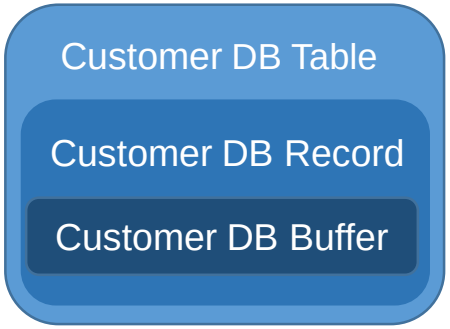
Mapping the “buffer object” hierarchy to OO buffer classes:



# The Final Objective



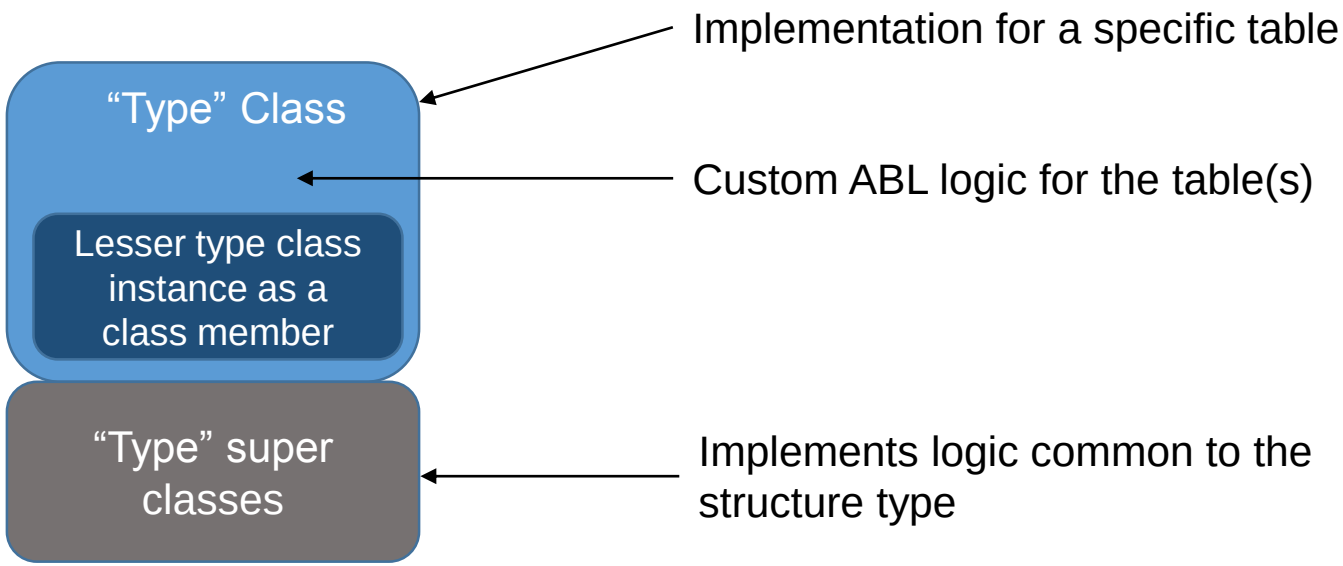
An overall view of the final object model:




| | |
|---|---|
| Customer Table Records | = DB Type III Structure |
| Customer Buffer Record / Fields | = DB Type II Structure |
| Customer Buffer Handle | = DB Type I Structure |

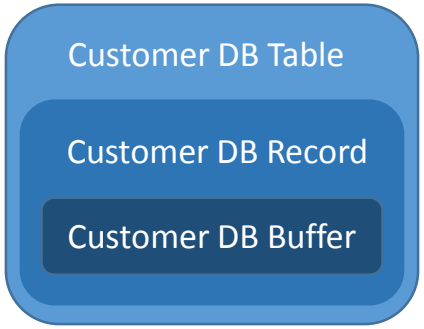
# "Type" Class Common Structure





# DB Type I Structure



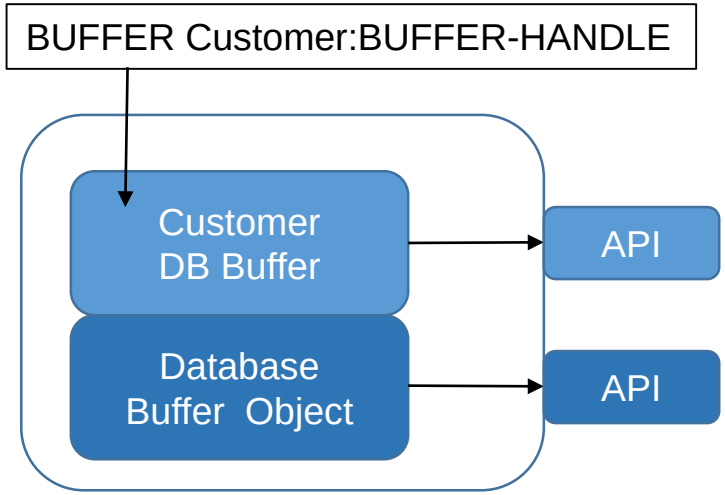



**Customer Buffer Handle = DB Type I Structure**

# DB Type I Structure



## Customer DB Buffer Design Overview




Structure:

- Customer DB Buffer class *inherits* the Database Buffer class
- Customer DB Buffer class holds business logic specific to the Customer table buffer
- Database Buffer Object methods & properties are *exposed*
- Customer DB Buffer Object methods & properties are *exposed*
- Customer DB Buffer can reference a local-to-the-class “Customer” buffer handle, **or** a Customer buffer-handle passed in via the class constructor

Properties & Usage:

- Used for controlling a Customer table buffer
- Appropriate for linking an OO Buffer to other OO(ABL) constructs (Queries, Data-Sources, etc.)

# DB Type I Structure



## Customer DB Buffer Implementation Overview

```
CLASS CustomerDbBuffer
    INHERITS DataBaseBufferObject:


CONSTRUCTOR CustomerDbBuffer():
SUPER(BUFFER Customer:HANDLE).
END CONSTRUCTOR.


CONSTRUCTOR CustomerDbBuffer(hBufferParm AS HANDLE):
SUPER(hBufferParm).
END CONSTRUCTOR.


END CLASS.
```

Class Definition

Create object using a local customer buffer

Create object using an external customer buffer reference

# Extending the Class



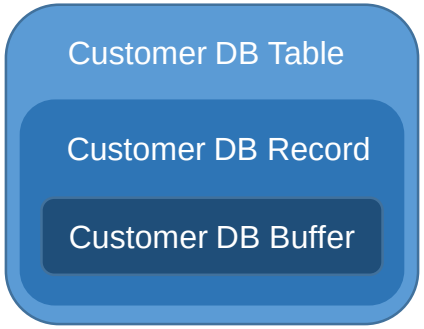
Extend CustomerDbBuffer to “FIND” a Customer

Add to CustomerDbBuffer:

```
METHOD PUBLIC LOGICAL FindCustomerNL( iCustnum  AS INTEGER):
RETURN(THIS-OBJECT:FindUnique(SUBSTITUTE("WHERE &1.custnum = &2",
                                          THIS-OBJECT:Name, STRING(iCustnum)),
                              LockWaitStatic:NoLock)).
END METHOD.
```

# DB Type II Structure





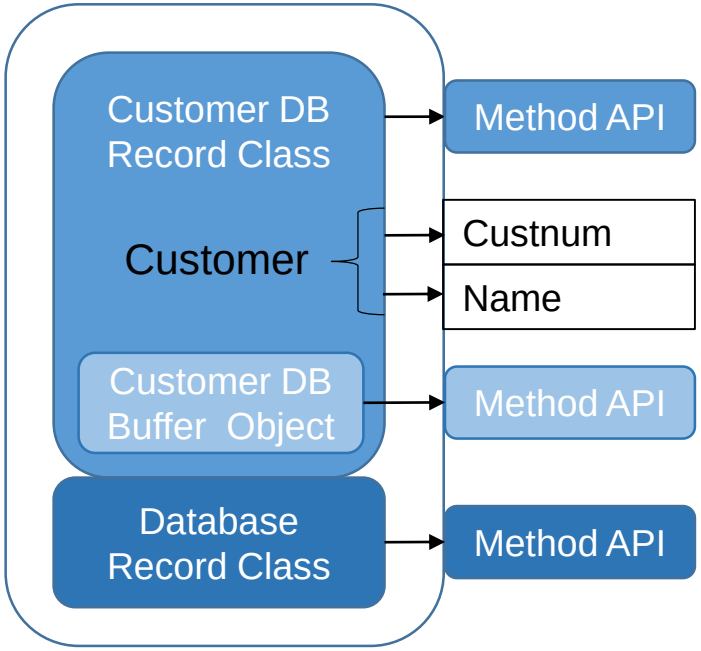

**Customer Buffer Record / Fields = DB Type II Structure**

Customer Buffer Handle = DB Type I Structure

# DB Type II Structure



## Customer DB Record Design Overview - Structure



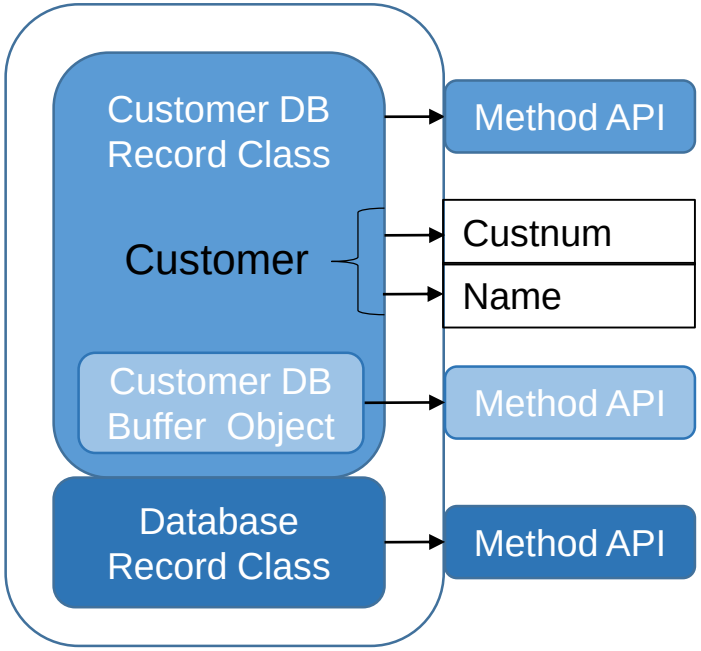

Structure:

- Customer DB Record class *inherits* Database Record Class
- Customer DB Record class *contains* a *private* Customer DB Buffer Object
- Customer DB Buffer Object references the Customer DB Record Customer buffer
- Database Record logic is *exposed* via *methods*
- Customer DB table *fields* are *exposed* via *properties*
- Customer DB Record *logic* is exposed via *methods*
- Customer DB Buffer Object reference can be obtained via a *method* API

# DB Type II Structure

## Customer DB Record Design Overview - Properties






Properties & Usage:

- Class properties mimic Customer table fields
- Record position and control done directly on the local Customer buffer, or indirectly using the Customer DB Buffer object
- Eliminates name-space collision between Customer DB Buffer Object *properties* and Customer DB table *field names*
- Appropriate location for implementing record-level CRUD business logic
- Used to link with other OO(ABL) constructs (Queries, Data-Sources, etc.)

# DB Type II Structure



## Customer DB Record Implementation Overview

```
CLASS CustomerDbRecord
    INHERITS DatabaseRecord:

DEFINE VARIABLE oCustomerDbBo AS CustomerDbBuffer NO-UNDO.


CONSTRUCTOR CustomerDbRecord():
THIS-OBJECT(NEW CustomerDbBuffer(BUFFER Customer:HANDLE)).
END CONSTRUCTOR.


CONSTRUCTOR PRIVATE CustomerDbRecord(oCustomerDbBoParm AS CustomerDbBuffer):
SUPER(oCustomerDbBoParm).
ASSIGN oCustomerDbBo = oCustomerDbBoParm.
END CONSTRUCTOR.

    /* Class Properties defined in next slide */

END CLASS.
```

Class definition

Customer DB Buffer Object

Create a new Customer DB record instance using a local customer buffer

Create a new record instance using a passed-in customer buffer object

Hold the Customer Buffer Object instance reference

# DB Type II Structure



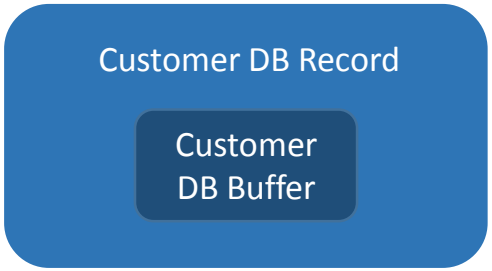
## Customer DB Record Implementation Overview – Field Properties

```
DEFINE PUBLIC PROPERTY CustNum AS INTEGER NO-UNDO
                                GET: RETURN(Customer.CustNum).
                                END GET.
                                SET(iCustNum AS INTEGER):
                                ASSIGN Customer.CustNum = iCustNum.
                                END SET.

DEFINE PUBLIC PROPERTY PostalCode AS CHARACTER NO-UNDO
                                GET: RETURN(Customer.PostalCode).
                                END GET.
                                SET(chTmp AS CHARACTER):
                                ASSIGN Customer.PostalCode = chTmp.
                                END SET.
```

# Extending the Classes



Task: Extend CustomerDBRecord to “FIND” using a Customer Number




Already added to CustomerDbBuffer:

```
METHOD PUBLIC LOGICAL FindCustomerNL( iCustnum  AS INTEGER):
RETURN(THIS-OBJECT:FindUnique(SUBSTITUTE("WHERE &1.custnum = &2",
                                         THIS-OBJECT:Name, STRING(iCustnum)),
                              LockWaitStatic:NoLock)).
END METHOD.
```

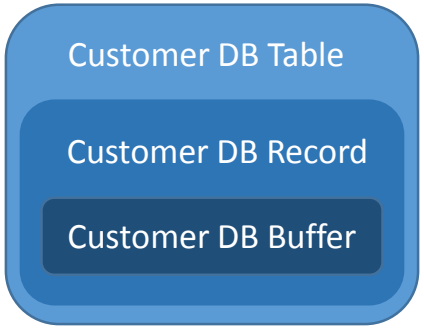
Façade which passes the call to the CustomerDbBuffer API

Now add to CustomerDbRecord:

```
METHOD PUBLIC LOGICAL FindCustomerNL(iCustNum AS INTEGER):
RETURN(oCustomerDbBo:FindCustomerNL(iCustNum)).
END METHOD.
```

# DB Type III Structure

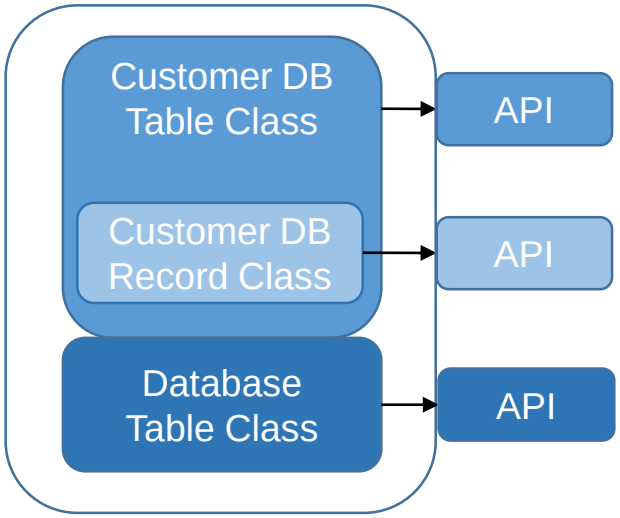





| | |
|---|---|
| **Customer Table Records** | **= DB Type III Structure** |
| Customer Buffer Record / Fields | = DB Type II Structure |
| Customer Buffer Handle | = DB Type I Structure |

# DB Type III Structure

Customer DB Table Design Overview:
Table = Container for 1 or more records




Structure:

- Customer DB Table class *inherits* Database Table class
- Customer DB Record is *private* to the Customer DB Table class
- Database Table class APIs are *exposed*
- Customer DB Table class APIs are *exposed*
- Customer DB Record class *exposed* via an API

Properties & Usage:

- Appropriate location for Customer DB Table scoped business logic
- Customer DB Table can be updated to have more than one Customer Record DB class instance
- Appropriate place for implementing table-level CRUD business logic
- Used to link with other OO(ABL) constructs (Queries, Data-Sources, etc.)

# DB Type III Structure



## Customer DB Table Implementation Overview

```
CLASS CustomerDbTable
    INHERITS DatabaseTable:

DEFINE VARIABLE oCustomerDbRecord AS CustomerDbRecord  NO-UNDO.

CONSTRUCTOR CustomerDbTable():
THIS-OBJECT(NEW CustomerDbRecord()).
END CONSTRUCTOR.

CONSTRUCTOR PRIVATE CustomerDbTable(oCustomerDbRecordParm AS CustomerDbRecord):
SUPER(oCustomerDbRecordParm).
oCustomerDbRecord = oCustomerDbRecordParm.
END CONSTRUCTOR.

METHOD PUBLIC CustomerDbRecord GetCustomerDbRecord():
RETURN(oCustomerDbRecord).
END METHOD.
END CLASS.
```

Class definition

Customer Record DB variable

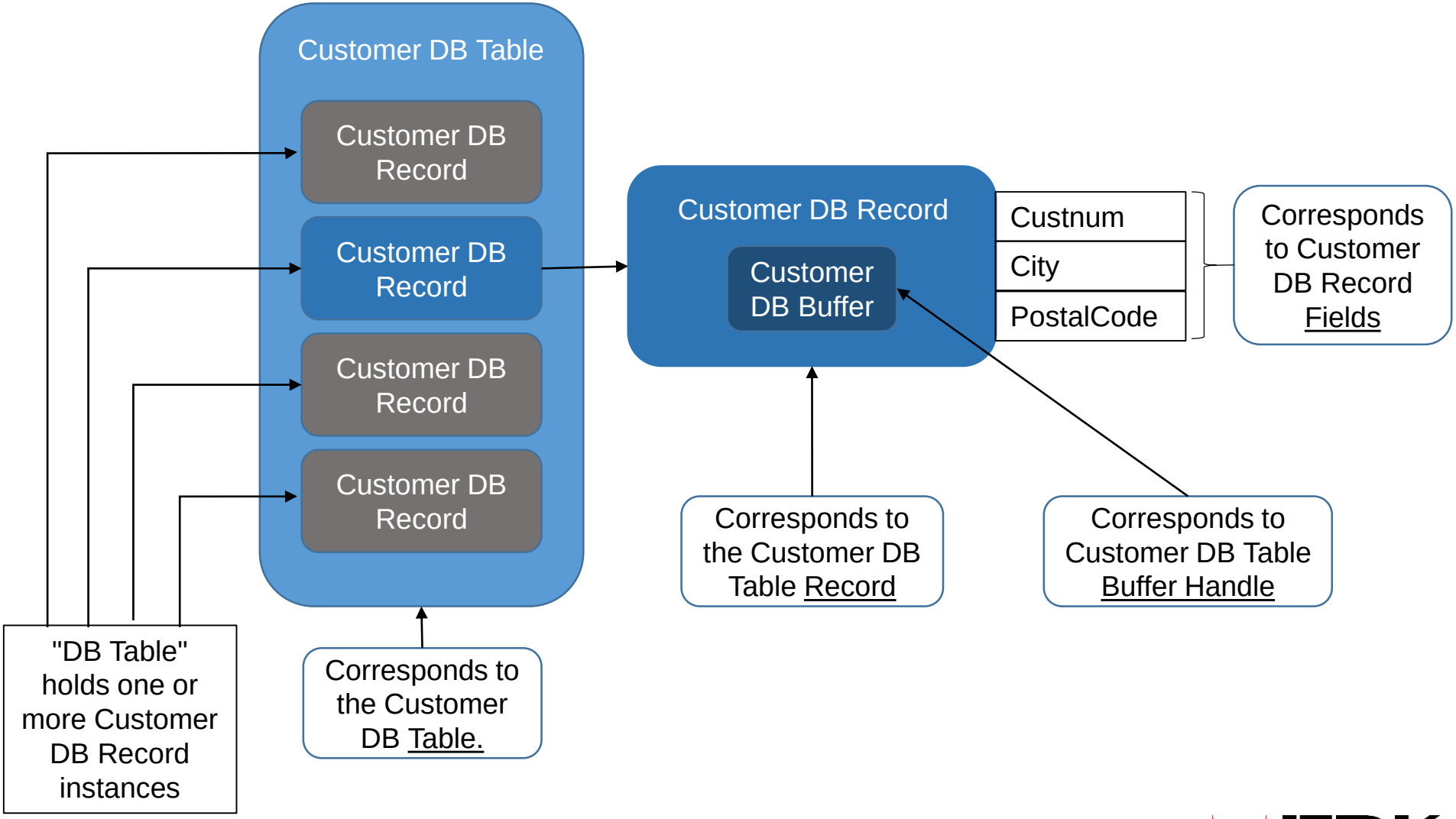
Create a new table using a default Customer DB Record object

Remember the Customer DB Record object used to make this instance

API to get the Customer DB Record instance

# Overall OO Structure

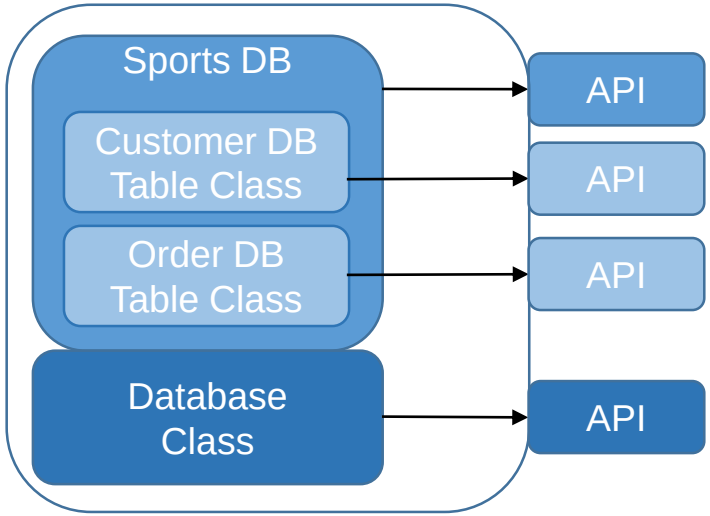




# DB Type IV Structure



## Sports DB Design Overview (not implemented)




Structure:

- Sports DB class *inherits* the Database class
- Sports DB *contains* all DB table classes as private instances
- Database class attributes are *exposed*
- Sports DB attributes are *exposed*
- SportsDB table classes are *exposed* via APIs

Properties & Usage:

- Contains all table classes.
- Appropriate location for all inter-table Data Access BL
- Appropriate place for DB-level CRUD BL

# Extending the Classes



Extend CustomerDBRecord to “FIND” a Customer

Add to CustomerDbBuffer:

```
METHOD PUBLIC LOGICAL FindCustomerNL( iCustnum  AS INTEGER):
RETURN(THIS-OBJECT:FindUnique(SUBSTITUTE("WHERE &1.custnum = &2",
                                         THIS-OBJECT:Name, STRING(iCustnum)),
                              LockWaitStatic:NoLock)).
END METHOD.
```

Add to CustomerDbRecord:

```
METHOD PUBLIC LOGICAL FindCustomerNL(iCustNum AS INTEGER):
RETURN(oCustomerDbBo:FindCustomerNL(iCustNum)).
END METHOD.
```

Façade which passes the call to the CustomerDbBuffer API

# Usage Example #1



Using CustomerDbRecord to find / display a customer

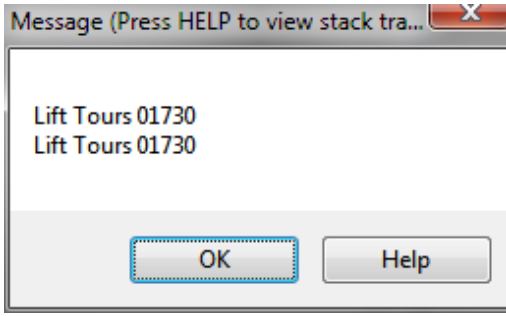
```
DEFINE VARIABLE oCustomerDbRecord AS CustomerDbRecord NO-UNDO.

oCustomerDbRecord = NEW CustomerDbRecord().

oCustomerDbRecord:FindCustomerNL(1).

FIND Customer
    WHERE Customer.CustNum = 1
    NO-LOCK
    NO-ERROR.

MESSAGE oCustomerDbRecord:NAME
        oCustomerDbRecord:PostalCode    SKIP

        Customer.Name
        Customer.PostalCode
   VIEW-AS ALERT-BOX.
```

Customer DB Record object

Customer DB Record object – find customer #1

Local customer buffer – find customer #1

Show the respective results of the two FINDs



# Usage Example #2



Using CustomerDbRecord to find / display an Order's customer

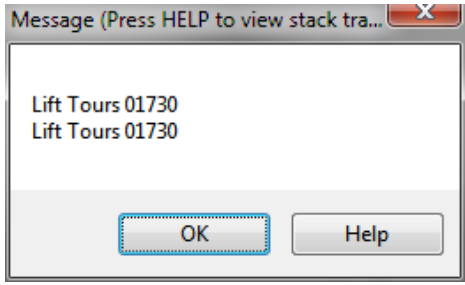
```
DEFINE VARIABLE oCustomerDbRecord AS CustomerDbRecord NO-UNDO.

oCustomerDbRecord = NEW CustomerDbRecord().

oCustomerDbRecord:FindCustomerNL(oOrderMaster).

FIND Customer
    WHERE Customer.CustNum = oOrderMaster:CustNum
    NO-LOCK
    NO-ERROR.

MESSAGE oCustomerDbRecord:NAME
        oCustomerDbRecord:PostalCode    SKIP

        Customer.Name
        Customer.PostalCode
   VIEW-AS ALERT-BOX.
```

Customer DB Record object

Customer DB Record object – find an order's customer

Local customer buffer – find customer for order

Show the respective results of the two FINDs



# Usage Example #3



Using SportsDB to get a CustomerDbRecord for an Order Line

```
DEFINE VARIABLE oSportsDB         AS SportsDB         NO-UNDO.
DEFINE VARIABLE oCustomerDbRecord AS CustomerDbRecord NO-UNDO.

oSportsDB = NEW SportsDB().

oCustomerDbRecord = oSportsDB:FindCustomerNL(oOrderLine).

MESSAGE oCustomerDbRecord:NAME
        oCustomerDbRecord:PostalCode    SKIP
   VIEW-AS ALERT-BOX.
```

Instantiate the database DA object

Return CustomerDbRecord instance / reference for an OrderLine

Show the customer info

# Usage Example #4



Using SportsDB to get a CustomerDbRecord for an Order Master

```
DEFINE VARIABLE oSportsDB         AS SportsDB         NO-UNDO.
DEFINE VARIABLE oCustomerDbRecord AS CustomerDbRecord NO-UNDO.

oSportsDB = NEW SportsDB().

oSportsDB:CustomerDbRecord:FindCustomerNL(oOrderMaster).

MESSAGE oSportsDB:CustomerDbRecord:NAME
        oSportsDB:CustomerDbRecord:PostalCode    SKIP
   VIEW-AS ALERT-BOX.
```

Instantiate the database DA object

Tell the CustomerDbRecord instance in SportsDB to find the customer record for Order Master

Show the customer info

# What about…



What about….

- Record locking?
- Data Updates?
- Transaction Scoping?

Stay tuned for “OO(ABL)” the sequel….

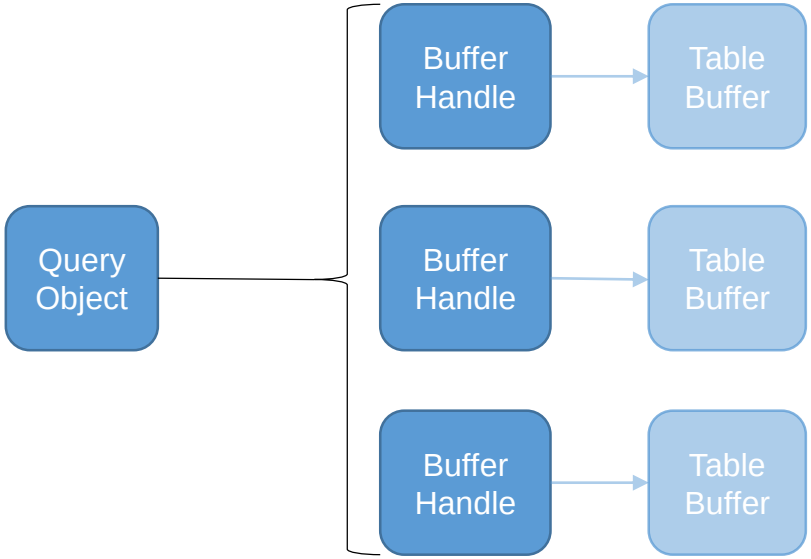
# The Next Step



Using OO(Buffer)s with an OO(Query)

# Common Query Structure

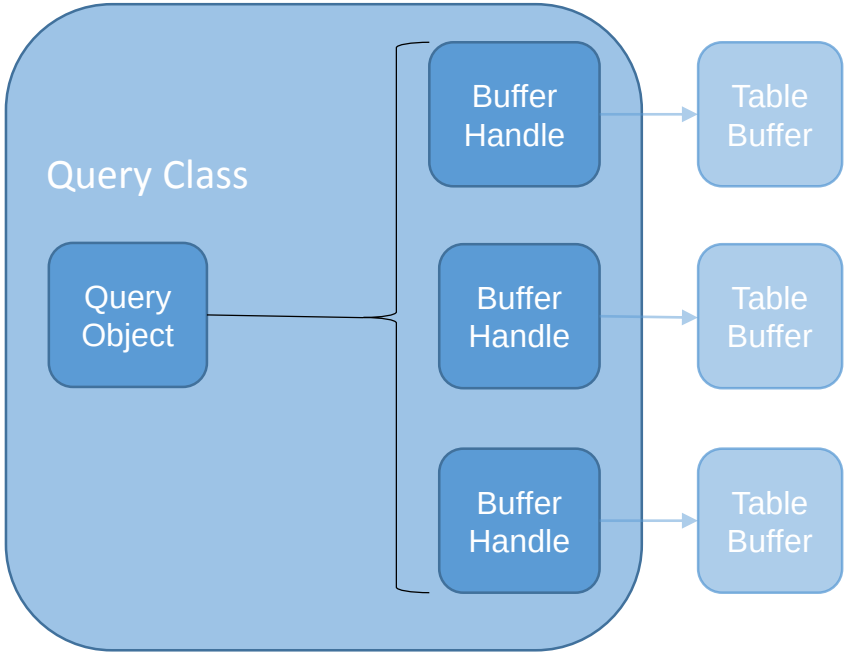




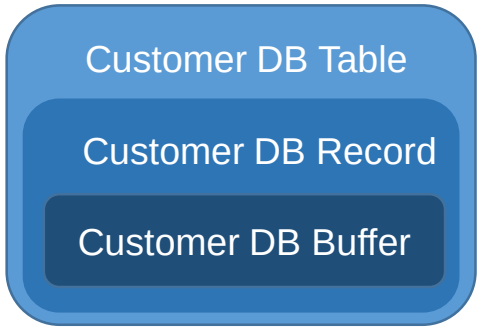
# OO(Query) Structure



OO Query Structure




OO(Table) Structure

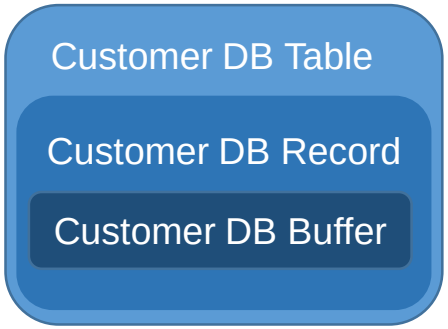



How do we get the buffer handle from an OO(Table) Structure to an OO(Query)?

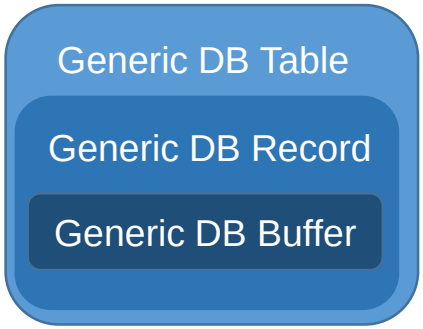
# Query/Buffer Linkage

How to get the buffer handle from an OO(Table) to an OO(Query)?

All these objects inherit …




…from these objects



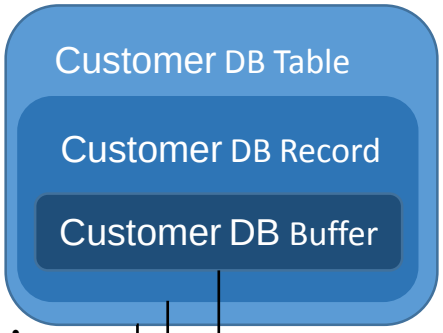

Getting to the buffer object's handle:

| Class | Method/Property |
|---|---|
| GenericTable: | GetGenericRecord() |
| GenericRecord: | GetGenericBo() |
| GenericBufferObject: | BufferHandle |

# Query/Buffer Linkage



How do we get the buffer handle from OO(Table) to OO(Query)?




OO(Query) “AddBuffer” Methods

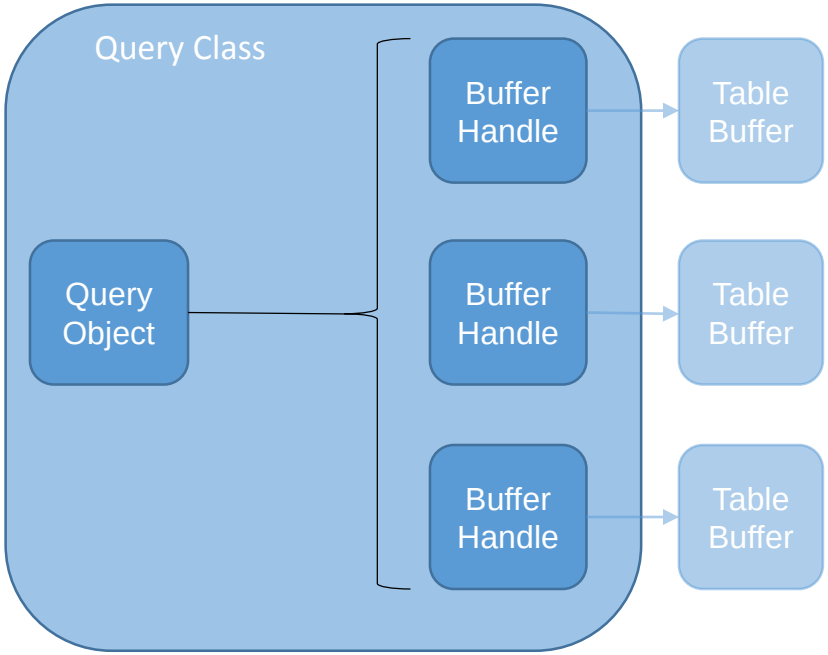
```
METHOD PUBLIC LOGICAL AddBuffer(oGenericTable AS GenericTable):
RETURN(THIS-OBJECT:AddBuffer(oGenericTable:GetGenericRecord())).
END METHOD.

METHOD PUBLIC LOGICAL AddBuffer(oGenericRecord AS GenericRecord):
RETURN(THIS-OBJECT:AddBuffer(oGenericRecord:GetGenericBo())).
END METHOD.

METHOD PUBLIC LOGICAL AddBuffer(oGenericBo AS GenericBufferObject):
RETURN(THIS-OBJECT:AddBuffer(oGenericBo:BufferHandle)).
END METHOD.

METHOD PUBLIC LOGICAL AddBuffer(hBuffer AS HANDLE):
RETURN(THIS-OBJECT:QueryObjectHandle:ADD-BUFFER(hBuffer)).
END METHOD.
```

# Using an OO(Query)



## Modelling Standard Query Operations:

OO(Query) Structure




- Add one or more buffers to the query
- Specify the join and search criteria
- Open the query
- Traverse the result list / Process each record
- Close the Query

# Using an OO(Query)



OO(Query) mimicking a common query structure

```
DEFINE VARIABLE oQryCst  AS QueryObject         NO-UNDO.
DEFINE VARIABLE oCstBuff AS GenericBufferObject NO-UNDO.
```

Create Generic OO Buffer object using a local customer buffer

```
oCstBuff = NEW GenericBufferObject(BUFFER Customer:HANDLE).

oQryCst  = NEW QueryObject().
```

Create an OO Query Object

```
oQryCst:SetBuffers(oCstBuff).
```

Set the Query Object's buffer

```
oQryCst:QueryPrepare("FOR EACH CUSTOMER WHERE CUSTOMER.NAME BEGINS ""All""").

oQryCst:QueryOpen().
oQryCst:GetFirstNoLock().
```

Open the query and get the first record

```
REPEAT WHILE NOT oQryCst:QueryOffEnd:
    DISPLAY Customer.Name          FORMAT "X(20)"
            Customer.City          FORMAT "X(20)"
            Customer.PostalCode    FORMAT "X(6)"
        WITH FRAME f1 DOWN.
    oQryCst:GetNextNoLock().
END.
```

Iterate through the result list and display the corresponding Customer records

# Using an OO(Query)



OO(Query) mimicking a common query structure - results:

| Name | City | Postal Code |
|---|---|---|
| All About Sports | Beavercreek | 45431 |
| All American Pro Sho | Broken Arrow | 74012 |
| All American Sportin | Algonquin | 60102 |
| All American Sports | Logan | 84321 |
| All Area Sports | Albany | 56307 |
| All Dakota Sporting | Aberdeen | 57401 |
| All Good Sports | Somerset | SO8 4G |
| All Out Sports | Bethany Beach | 19930 |
| All Seasons Sportara | Baldwin | 11510 |
| All Sport | Brick | 08724 |
| All Sports | Phoenix | 85021 |
| All Star Sport Ctr | Bayonne | 07002 |
| All Star Sporting Go | Carthage | 37030 |
| All Star Sports | Cody | 82414 |
| All Stick Sports | Sydney | 2098 |
| All Weather wear | Fairbanks | 99709 |
| Allen Sports Ctr | Allen | 75002 |
| Allied Sporting Good | Clarksville | 47129 |
| Allied Sporting Good | Louisville | 40218 |

# OO(Query) + OO(DbRecord)



OO(Query) + OO(CustomerDbRecord), or
Create an OO(Query) using an OO Customer Record object

Design Objectives:

1. Change from using a Generic Buffer Object to a CustomerDdRecord Object
2. Process the full query
3. Show what records were found

Implementation:

1. Create a CustomerDbRecord object
2. Attach the CustomerDbRecord object to an OO Query object
3. Show each customer record found by the query

# OO(Query) + OO(DbRecord)



Create an OO(Query) using an OO Customer DB Record object

```
DEFINE VARIABLE oQryCst   AS QueryObject      NO-UNDO.
DEFINE VARIABLE oCstDbRec AS CustomerDbRecord NO-UNDO.

oCstDbRec = NEW CustomerDbRecord().

oQryCst   = NEW QueryObject().

oQryCst:SetBuffers(oCstDbRec).

oQryCst:QueryPrepare("FOR EACH CUSTOMER " +
                     "WHERE CUSTOMER.NAME BEGINS ""All""").

oQryCst:QueryOpen().
oQryCst:GetFirstNoLock().

REPEAT WHILE NOT oQryCst:QueryOffEnd:
    DISPLAY oCstDbRec:Name           FORMAT "X(20)"
            oCstDbRec:City           FORMAT "X(20)"
            oCstDbRec:PostalCode     FORMAT "X(6)"
        WITH FRAME f1
            DOWN.
    oQryCst:GetNextNoLock().
END.
```

Create Customer Table Record Object

Create Query Object

Set Query Object buffer structure

Open query and
get the first record

Iterate through results and display the corresponding Customer records using the oCstDbRec object

# OO(Query) Design



## OO(Query) using an OO Customer Record object – Results

# Other ABL Objects "in development"



Other OO(ABL) Objects “in development”:

- Data Source Object
- Data Relation Object
- DataSet Table Object (TempTable in a PDS)
- ProDataSet Object
- The rest of the language….

# Questions?

# Partners Wanted



There is much R&D, Q&A, etc. left to do to bring this research to a production-ready state, and TDK Consulting is looking for partners to join in developing this technology.

Contact Tim Kuehn for more information.

# Other Sessions by TDK Consulting



| Session | Title | Description |
|---|---|---|
| 198 | OO ABL Events | A talk on OO "Events" w/bonus material |
| 206 | Migrating to Multi-tenancy | A step-by-step walk-through of merging two Sports2000 databases into a single MultiTenant database. |
| 207 | ABL Buffers and Queries in OO Wrappers | A research report on the development of an OO based Data Access layer |

# Thank you for your time!



Tim Kuehn
TDK Consulting Services Inc.
www.tdkcs.com

Email: tim.kuehn@tdkcs.com
Ph: 519-576-8100
Mobile: 519-781-0081
Twitter: @tdkcs