Table Partitioning Application and Design

Richard Banville
OpenEdge Development
Progress Software

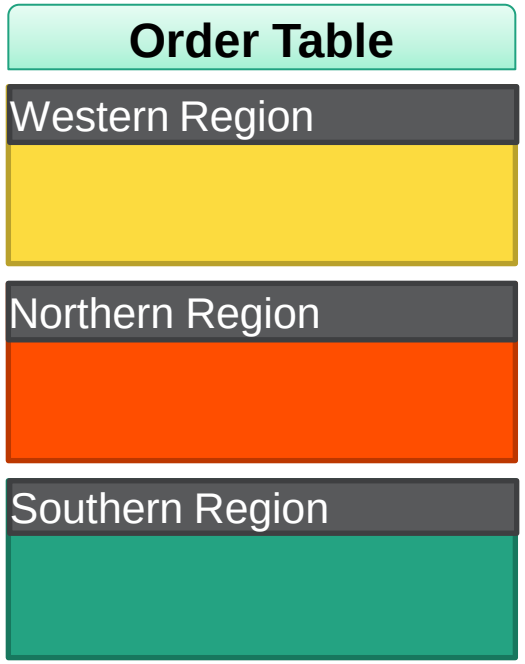
Agenda

- Table Partitioning in OpenEdge
- Partition design considerations
  - Partition definition setup, not physical layout
- Application Impact
  - Transparent to application development (for the most part!)
  - Rowid usage
  - Record operational impact
- New locking construct interactions

Data Access: List Partitioning

List Partitioning

FIND Order WHERE region = Western.

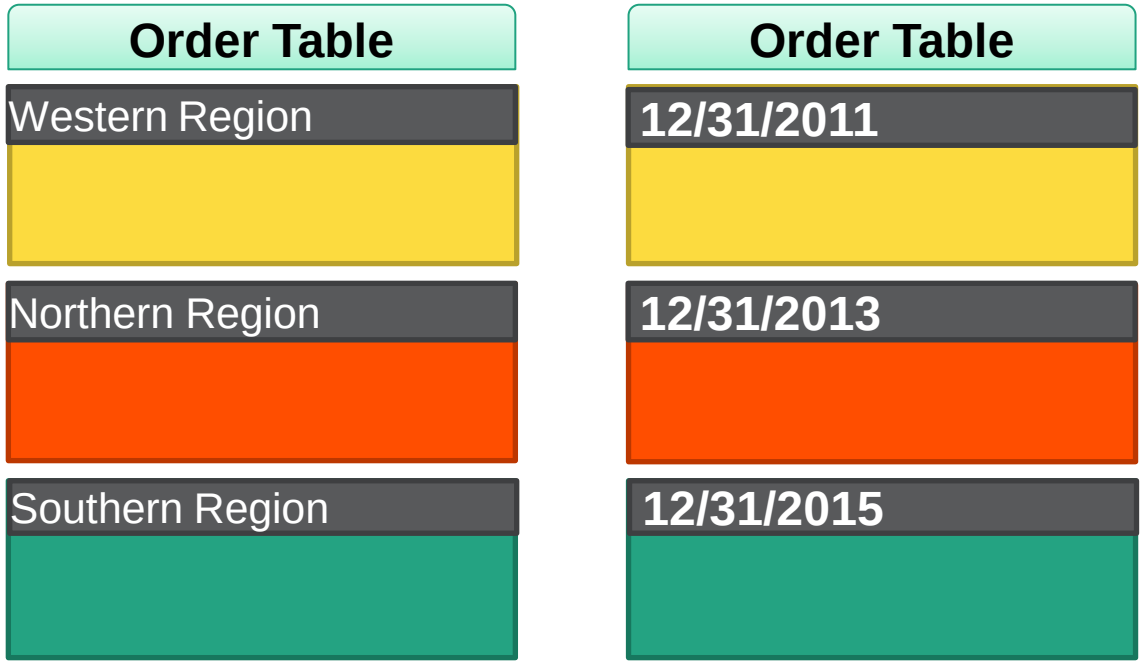


- List Partition
  - Partition based on a single / unique value
    - Data value == partition definition ➔ storage location
  - Must be fully qualified; No “catch all” partition
  - May want to create a “default” partition based on initial value
    - “Other” Region
  - No partitioning on UNKNOWN value

Data Access: Range Partitioning

List Partitioning **OR** **Range Partitioning**

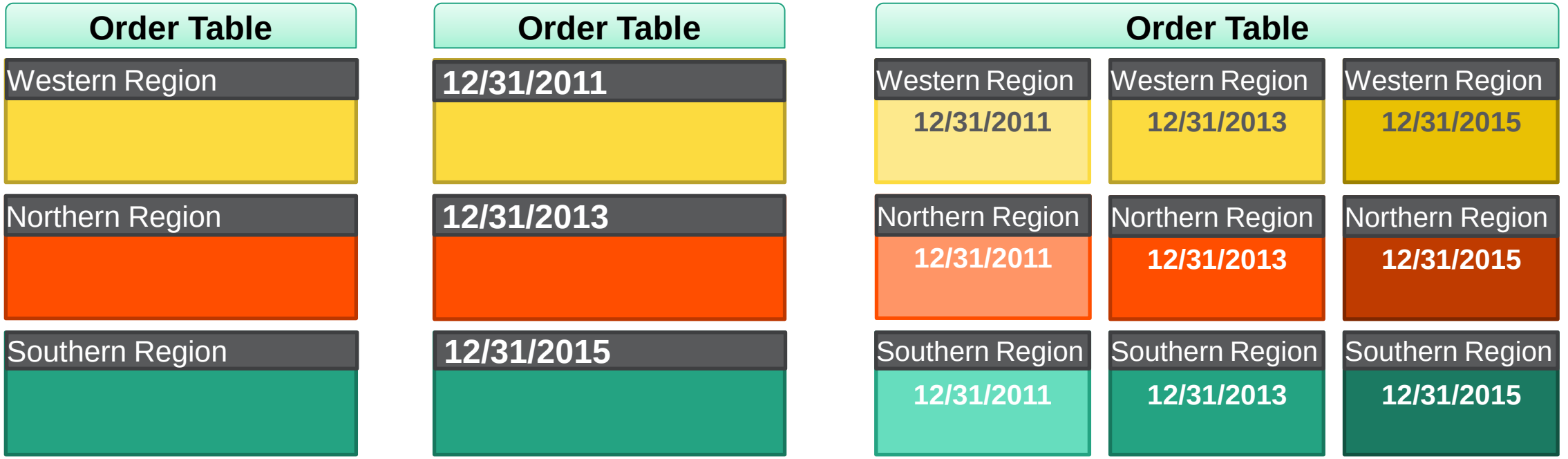
```
FOR EACH Order WHERE
                Order-Date <= 12/012/2013.
```



- Range partition
  - Partition contains values LE specified range
  - No “high-range” value
  - Could just use 99/99/9999 for a date range
    - Use split utility to segregate data
  - Range can be any “indexable” datatype
  - No partitioning on UNKNOWN value

Data Access: Sub-partitioning

List Partitioning **OR** **Range Partitioning** **OR** **Sub-partitioning**



Partitioned Tables:
Design Considerations

Why Are You Partitioning?

Maintenance

- Data re-org / rebuild
- Data purging
- Data archival

Availability

- Data repair
- Data isolation
- Historic data access

Performance

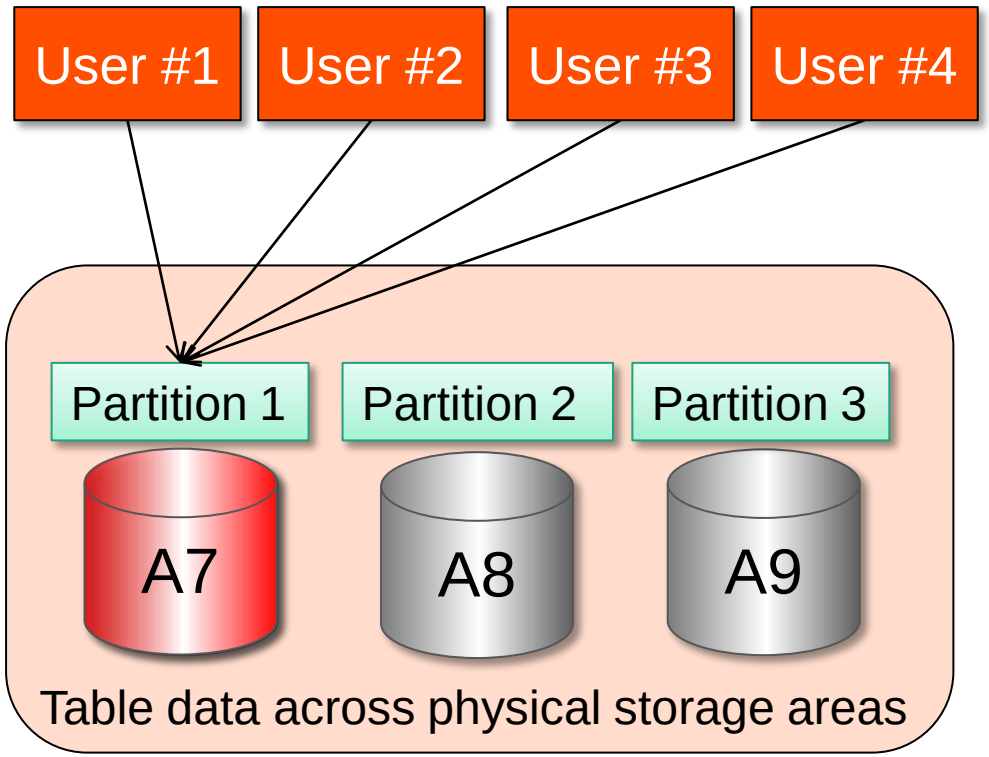
- “Hot” table
- “Hot” index access

Increasing Concurrency With Table Partitioning

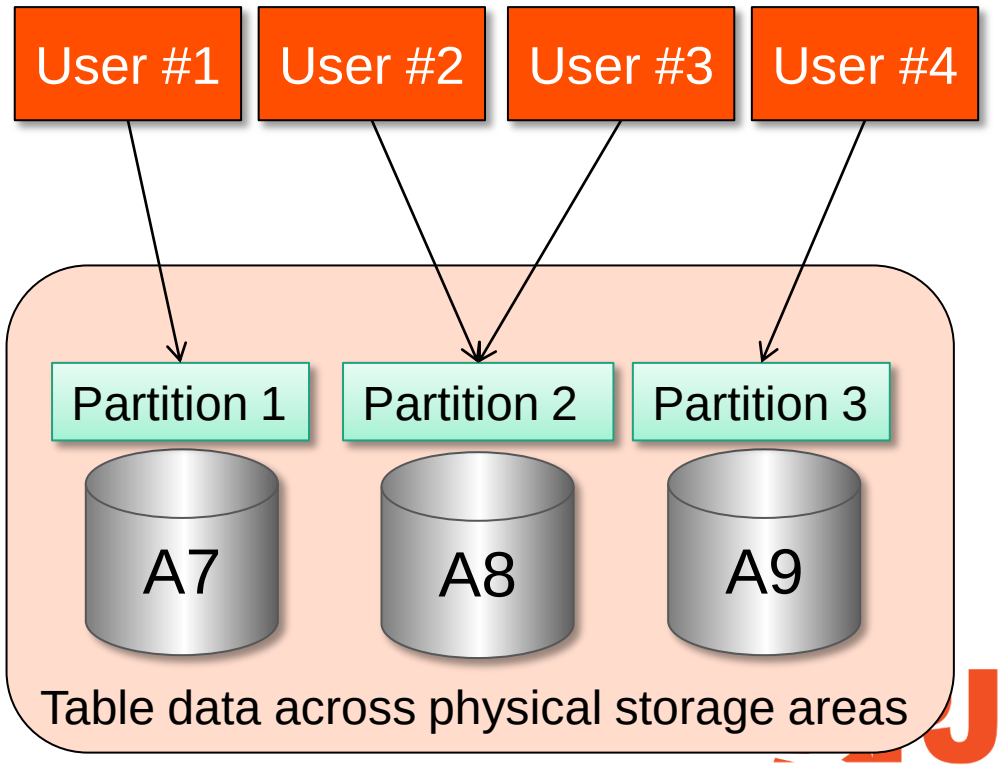
Create order where Order-date = TODAY AND
Sales-Rep = mySalesRep.

Pick me

Range Partitioning by Order-Date



Sub-partitioning by Sales-rep & Order-Date



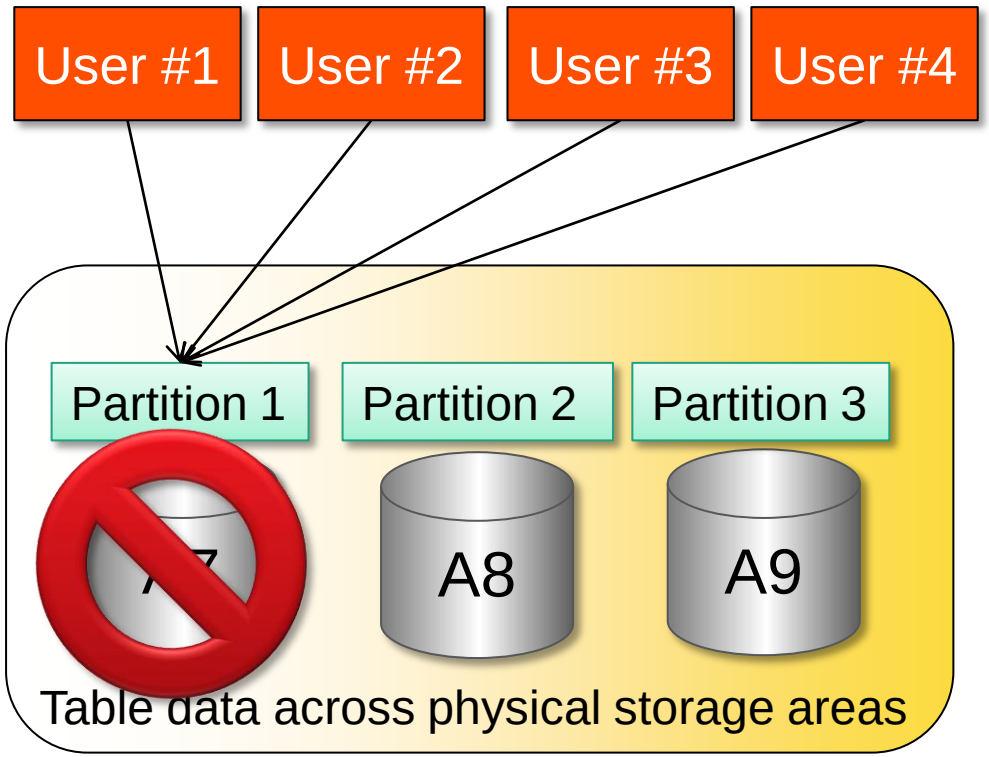
PROGRESS SPARK

Increasing Availability With Table Partitioning

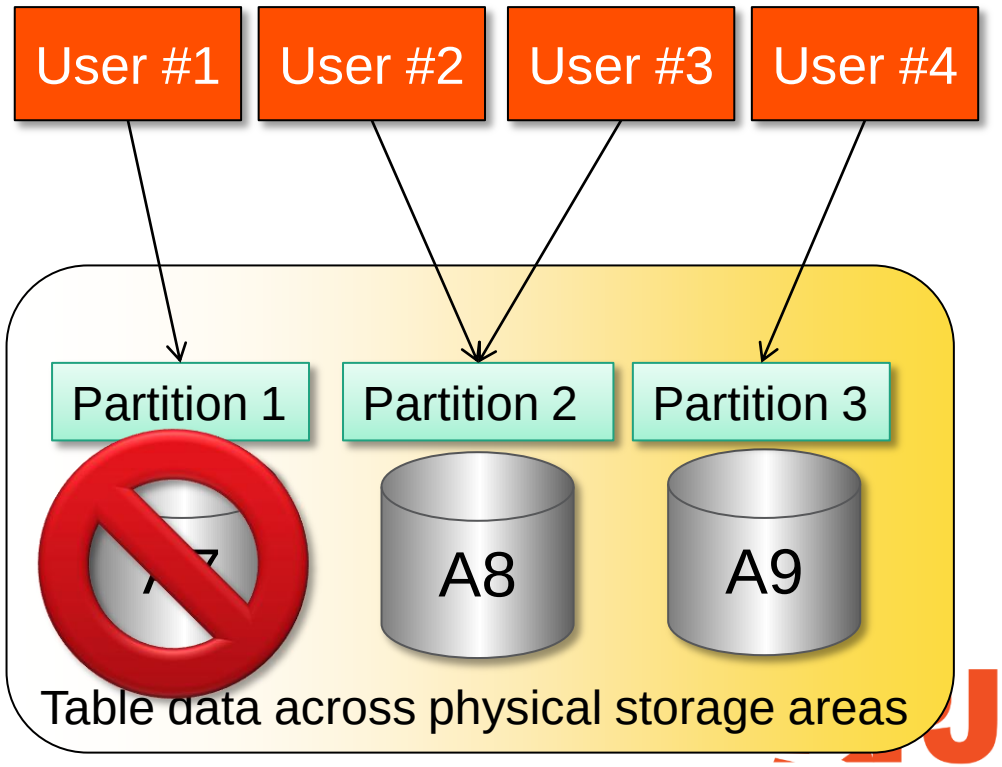
Create order where Order-date = TODAY AND
Sales-Rep = mySalesRep.

Pick me

Range Partitioning by Order-Date



Sub-partitioning by Sales-rep & Order-Date



PROGRESS SPARK

What to Partition

- Get it right the first time
  - Splitting / Merging partitions is straightforward
  - Repartitioning existing definitions requires a dump and load
- Data organization
  - Look for grouping of data “by data value”
  - Organized by sequential data range?
    - Range partitioning
    - Range rather than single value to identify a group of data
    - Date (most typical), product code, alphabetic range
  - Organized geographically or grouped by specific “static” entities
    - List partitioning
    - Country, region, company, division

Consider Data Access Patterns

Range partitions

- By “year” is a typical approach for tables with date field
  - Order-date vs Shipped-date
- Sub-partitioning candidate?
  - Can you include another column (or add one)?
- Determine appropriate date range
  - Maintain / access data
    - Activity patterns: Purge, archive, reorganize, relocate
    - Calendar year, fiscal year, quarter?
- Determine product code or alphabet range grouping
  - Affect on high-availability, archival, etc.
  - Load balance groups of data, not just modulus

PROGRESS SPARK

Consider Data Access Patterns

List partitions

- By geographic region or division are typical approaches
  - Why or why not Sales-Rep?
- Sub-partitioning candidate?
  - Can you include another column (or add one)?
  - By country-code by region
- Consider number of unique data values
  - 32,765 max defined partitions per table

Consider Join Activity

- How is the data typically accessed?
  - Customers have orders
  - Orders refer back to customers by cust-num
- I want to organize customers by region, orders by date (year)?
  - What should my partitioning scheme be?
- Should I de-normalize
  - Customers by region
  - Orders by region and year?
- Should I just rely on global indexes for the child join?
- Should I add a global index on year and join using local index?

Decisions are based on YOUR data access patterns

PROGRESS SPARK

Partitioned Tables: Application Development Considerations

Record Creation: Some “Gotchas”

- Requires partition fields be filled out
  - UNKNOWN values not allowed in partition columns
  - Use mandatory fields
  - Use appropriate initial values
    - Changing assigned initial value may adversely affect performance
- When is a record actually created?
  - RELEASE or VALIDATE
  - Buffer out of scope / reuse (txn end, new record)
  - LOB assignment
  - Assign a field of a unique index

PROGRESS SPARK

Record Creation: Range Partitioning

Range partitioning by **Order-date**

Record creation

Record update

```
CREATE Order.
ASSIGN  Cust-num = Customer.cust-num
        Order-Num = NEXT-VALUE(Ord-Seq).
...
ASSIGN  Order-date = TODAY.
```

Initial record creation will fail

Either:

- Move Order-date assignment
- Make TODAY initial value

Attempt to create/set a value for a data partition column in partitioned table Order where the value is not in one of the defined partitions. (17094)

PROGRESS SPARK

Record Creation: List Partitioning

List partitioning by **Country**

Record creation

Record update

```
CREATE Customer.
ASSIGN  Cust-num = NEXT-VALUE(Cust-Seq)
        name     = "Ritchid".
...
ASSIGN  Country = "Finland".
```

Initial record creation will fail

Either:

- Move Country assignment
- Have valid initial value "Other"
  - Re-assignment causes delete/insert

Attempt to create/set a value for a data partition column in partitioned table Customer where the value is not in one of the defined partitions. (17094)

Rowid and Data Location

Recid

- Unique per area
- NOT unique per partitioned table
- Integer
- Range comparisons (<, >, =)

Rowid

- Unique per area
- Unique per partitioned table
- Raw storage (convert to string for display)
- Variable length of bytes
- Equality comparisons
- Format subject to change

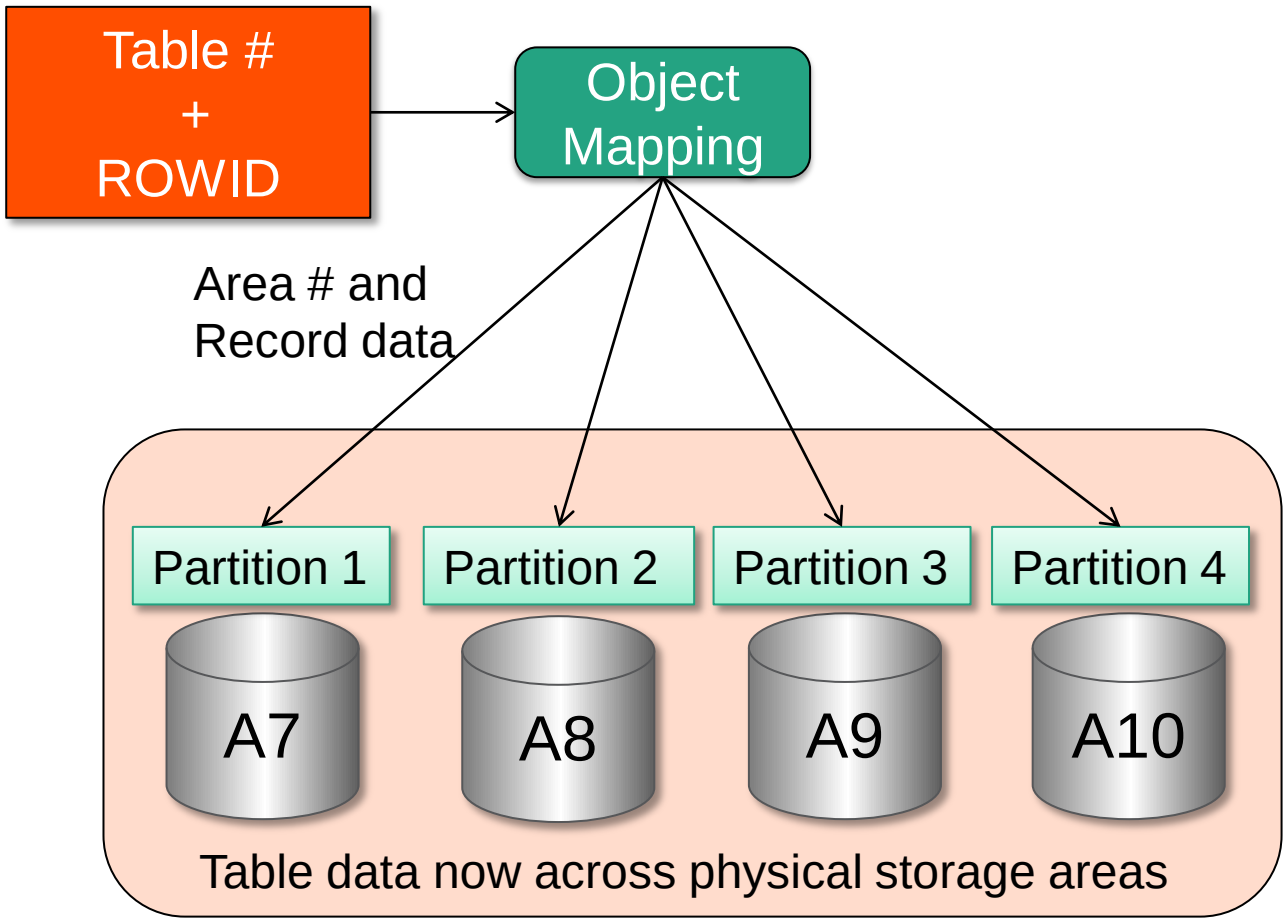
```
Display  recid(Order)  string(rowid(Order)).
```

Version	Recid	Rowid	
11.3:	1951	0x000000000000079f	Table 2, row # 1951
11.4:	1951	0x000000000000079f**0000**	Table 2, row # 1951, **Partition 0** (implied)
11.4:	1951	0x000000000000079f**0002**	Table 2, row # 1951, **Partition 2**

PROGRESS SPARK

Rowids and Data Location Mapping

FIND Cust where rowid(Cust) = myRowid.

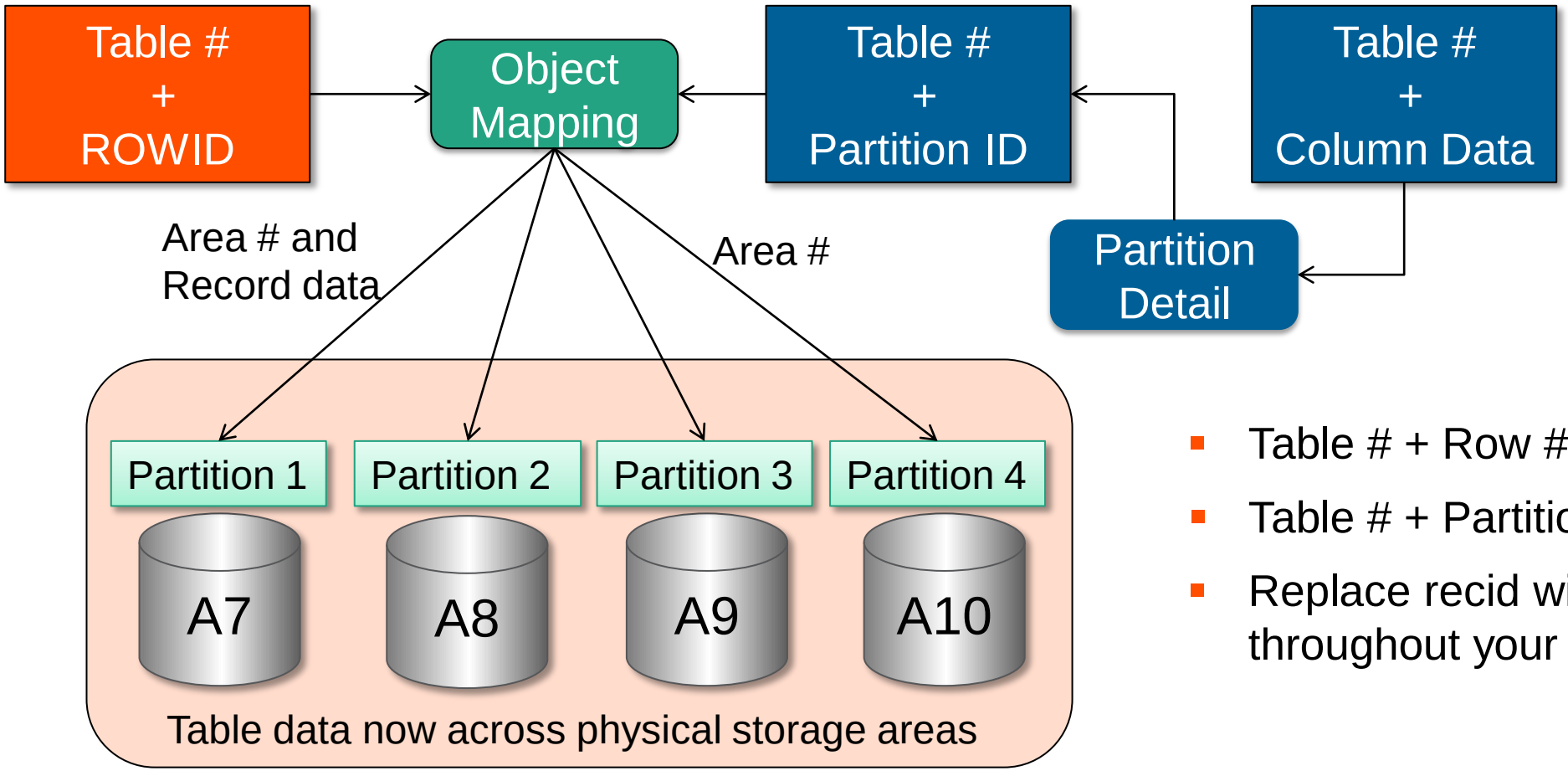


- Table # + Row # => record data
- Table # + Partition => Area
- Replace recid with rowid throughout your code

Rowids and Data Location Mapping

FIND Cust where rowid(Cust) = myRowid.

Create Cust. Assign country = “Austria”.



- Table # + Row # => record data
- Table # + Partition => Area
- Replace recid with rowid throughout your code

Displaying Partition Information

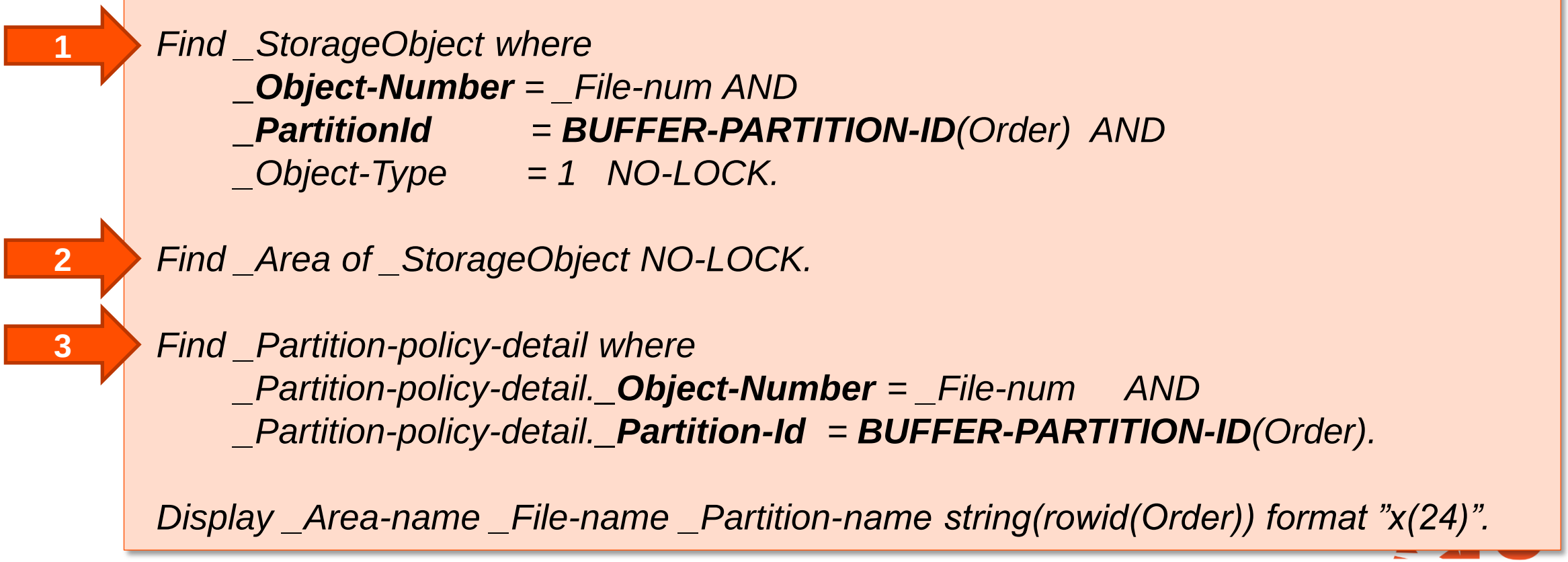
- Partition Ids are NOT needed in your code
- However we've provided access to partition information via the ABL & SQL:
  - Retrieving the partition Id (ABL Example)
    - Built-in function: BUFFER-PARTITION-ID (*buffer-name*)
    - BUFFER-PARTITION-ID attribute
  - IS-PARTITIONED attribute

```
FIND FIRST Order NO-LOCK.
MESSAGE
  "Partition (Attribute):" BUFFER-PARTITION-ID:getByRowid(ROWID(Order))

  "Partition (Attribute):" BUFFER-PARTITION-ID:getByHdl(BUFFER  Order:HANDLE)

  "Partition (Function):" BUFFER-PARTITION-ID (Order)

  "Is-Partitioned?" BUFFER Order:IS-PARTITIONED.
```

PROGRESS SPARK

Where's my row?

Table number + Partition Id => Area location



PROGRESS SPARK

Dumping Data by Recid

- Disaster recovery scenario
- All alternative approaches are not possible
  - No DR, no AI, no backup, no job

```
DO myRecid = 1 to maxId:
   FIND Order where recid(Order) = myRecid   NO-LOCK NO-ERROR.
   IF  AVAILABLE Order then
      EXPORT Order.
END.
```

- This will **not** work for partitioned tables

Dumping Partitioned Data Using Rowids

- New RowidGenerator class with 2 static methods
  - Useful for disaster recovery scenario when alternative approaches are not possible
- Specify partitionId of “?” to indicate ALL partitions
- Ensure maxId is large enough!

```
USING OpenEdge.DataAdmin.Util.RowidGenerator.

METHOD PUBLIC STATIC VOID TableStart
   (tableName  AS CHAR,
    startId    AS INT64,   /* Starting record # */
    maxId      AS INT64,   /* Ending record # */
    partitionId AS INT)


METHOD PUBLIC STATIC ROWID GetNextRowid()
```

Dumping Partitioned Data Using Rowids

```
USING OpenEdge.DataAdmin.Util.RowidGenerator.
DEFINE VAR myRowid AS ROWID.

/* Scan all order partitions */
RowidGenerator:TableStart("Order", 1, 9999999, ?).
myRowid = RowidGenerator:GetNextRowid().

DO WHILE myRowid <> ?:
    FIND Order where rowid(order) = myRowid  NO-LOCK  NO-ERROR.
    IF AVAILABLE(Order) THEN DO:
        EXPORT Order.
    END.

    myRowid = RowidGenerator:GetNextRowid().
END.
```

Scan ALL partitions

Record Update Affect: Can Now Change rowid

- Prefetch / scrolling queries:
  - REPOSITION-TO-ROW
    - Behaves as if record was deleted *
    - Must reopen query to revalidate
  - Current session makes the change
    - The result set is automatically fixed up
    - GET/FINDs will continue to work
  - Different session makes the change
    - CURRENT-CHANGED will return TRUE
    - GET/FINDs will behave as if the record were deleted *
      - Even if the record still satisfies the query!
      - Must reopen query to revalidate

* Indicates change in behavior

PROGRESS SPARK

Record Update: Can Now Change rowid

```
DEFINE QUERY q1 FOR Customer SCROLLING.


QUERY q1:HANDLE:SKIP-DELETED-RECORD = NO.
OPEN QUERY q1 FOR EACH Customer.

GET FIRST q1.
GET NEXT q1.
                    /***** Another session changes partitioned field country */
GET FIRST q1.     /***** GET ERROR if  explicitly NOT skipping deleted records */
GET NEXT q1.
```

** No Customer record is available. (91)

PROGRESS SPARK

Record Update: Can Now Change rowid

```
DEFINE QUERY q1 FOR Customer SCROLLING.

                    /***** Following set to YES (default) will avoid the runtime error */
QUERY q1:HANDLE:SKIP-DELETED-RECORD = YES.
OPEN QUERY q1 FOR EACH Customer.

GET FIRST q1.
GET NEXT q1.
                    /***** Another user changes partitioned field country */
GET FIRST q1.       /***** Changed record is skipped */
GET NEXT q1.
```

NOTE: This is the default behavior when SKIP-DELETED-RECORD is not specified.

PROGRESS SPARK

Data Access Restrictions

- No pre-OpenEdge 11.4 client access to partition enabled database
  - Not single user, self service nor remote access
- Post OpenEdge 11.0: r-code compatible
- Shared schema access (online)
  - Adding new partitioned tables
  - Adding new partitions to existing tables
  - Partition maintenance
- Exclusive schema access (ugh!)
  - Marking existing table partitioned
- Reverting:
  - Must remove all partitioned tables, partition definitions and disable partitioning
  - Dump, reconfigure, load, disable

Data Access Restrictions

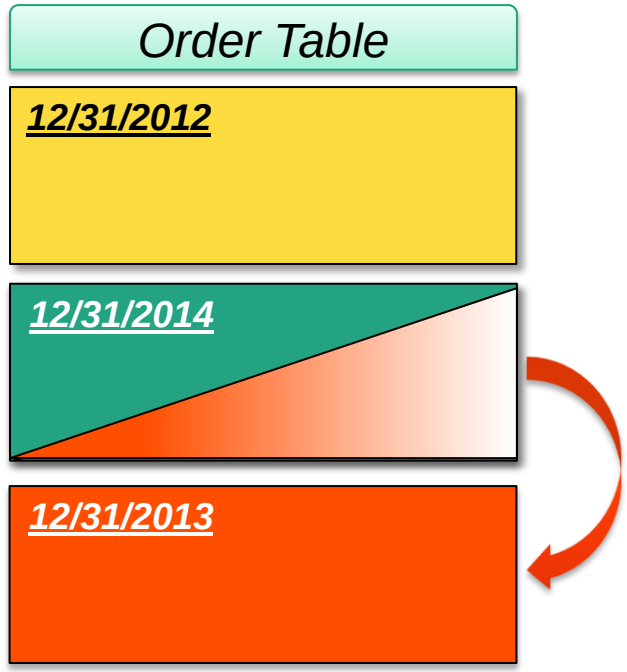
- Cancelled or in progress split/merge operation
  - Data being moved is “generally” inaccessible
  - Inserts to the transitioning partition are also prevented

*FOR EACH Order NO-LOCK by **Order-date**:*
DISPLAY Cust-num Order-num Order-date.

A partition of table “Order“”cannot be accessed pending completion of database utility operation (17604)

- NOTE: This is a STOP condition
  - use ON-STOP vs ON-ERROR

Range Partitioning



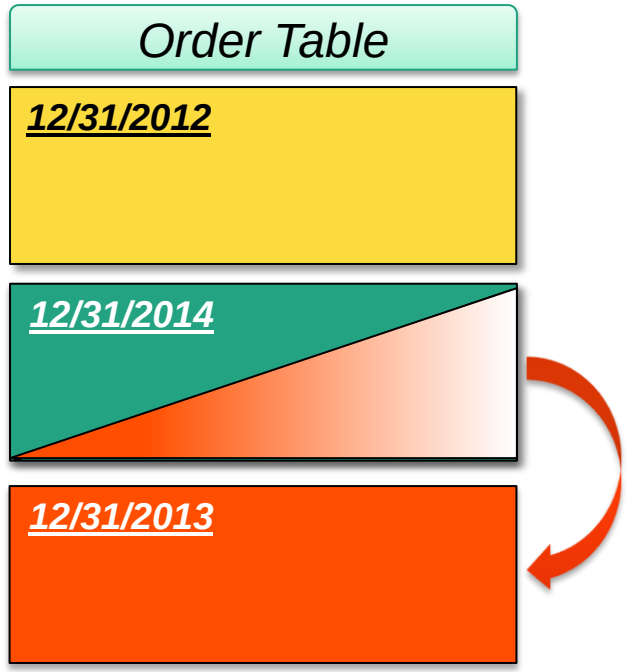
Data Access Restrictions

- In progress or cancelled split/merge operation
  - Data being moved is “generally” inaccessible
  - Partition data **NOT** in transition is **fully** accessible

*FOR EACH Order NO-LOCK where **order-date >= 2014**:*
DISPLAY Cust-num Order-num Order-date.

Cust-Num	Order-Num	Order-Date
1	6	01/05/2014
1	36	01/19/2014
…	…	…

Range Partitioning



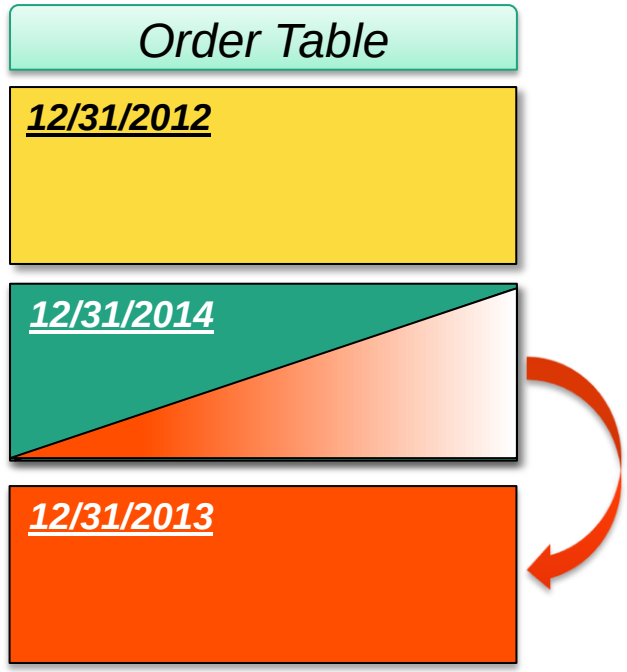
Data Access Restrictions

- In progress or cancelled split/merge operation
  - Data being moved is “generally” inaccessible
  - Global vs local index support

*FOR EACH Order NO-LOCK by **Cust-num**:*
DISPLAY Cust-num Order-num Order-date.

Cust-Num	Order-Num	Order-Date
1	6	09/25/2013
1	36	01/19/2014
…	…	…

Range Partitioning



- Access via global index is allowed
  - Global index always knows where the data is
- Updates to transitioning partition **always** disallowed

Exception Handling

- Some more stop conditions:
  - No partition definition exists

Attempt to create/set a value for a data partition column in partitioned table “Customer“ where the value is not in one of the defined partitions. (17094)

  - Field assignment to Offline/ unallocated partition

Attempt to create in partitioned table “Customer’” where the partition has not been allocated (17698)

- Exception conditions
  - “Lookup” type failures are ERROR conditions
  - “Assignment” type failures are generally STOP conditions

PROGRESS SPARK

New Schema: proutil <*db*> -C enabletablepartitioning

- _Partition-Policy (-352)
  - Describes partition at the “table” level
  - Lookup requires Table #

Column	Name	Type
2	_Partition-Policy-Name	char
3	_Object-Number	Integer
4	_DataArea-default	Integer
5	_IndexArea-default	Integer
6	_LobArea-default	Integer
7	_Allocation-default (None, immediate, delayed)	Char
8	_Num-Columns	Integer
9	_Column-Name	char16]
10	_Has-Range	Logical
11	_Description	char
12	_Misc	char[16]

- _Partition-Policy-Detail (-353)
  - Defines each individual partition
  - Lookup requires Table # AND PartitionId

Column	Name	Type
2	_Object-Number	integer
3	_Partition-Id	integer
4	_Partition-Name	character
5	_Partition-Column-Value	character[16]
6	_Partition-Internal-Value	raw
7	_Attributes	Logical[64] [1] = 1 space allocated [2] = 1 this is a sub-partition [3] = 1 lowest level sub-partition [4-63] unused
8	_Description	character
9	_ianum-Data	Integer
10	_ianum-Index	Integer
11	_ianum-Lob	integer
12	_Misc	character[16]

Useful for self-provisioning partitions

PROGRESS SPARK

Programmatic Partition Definition Support

- ABL API support is also provided
- Used by OpenEdge Explorer and OpenEdge Management
- OpenEdge SQL provides DDL support

Associate table w/partitioning

Define partitions for table

```
define variable policy as IPartitionPolicy no-undo.
define variable detail as IPartitionPolicyDetail no-undo.
policy = Service:NewPartitionPolicy("OrderDate")
    policy:DefaultAllocation = "Immediate"
    policy:Table = tbl…
 detail = Service:NewPartitionPolicyDetail("OldData")
    detail:SetValue(12/31/2012)…
    policy:Detail:Add(detail).
```

Summary

Transparent to application development (for the most part!)

- Must consider record creation and changing rowids
- Partition information accessible but not needed

Online partition maintenance affects

- Potential for new ERROR / STOP conditions
- Improves availability and maintenance scope

Partitioning design scheme is important

- Lots of things to think about
- Getting it right the first time is the goal

PROGRESS SPARK

PROGRESS

Want To Learn More About Openedge 11?

- Role-based learning paths are available for OpenEdge 11
- Each course is available as Instructor-led training or eLearning
- Instructor-led training:
  - $500 per student per day
  - https://www.progress.com/support-and-services/education/instructor-led-training
- eLearning:
  - Via the Progress Education Community (https://wbt.progress.com):
  - OpenEdge Developer Catalog: $1500 per user per year
  - OpenEdge Administrator Catalog: $900 per user per year
- User Assistance videos:
  - https://www.progress.com/products/pacific/help/openedge

New Course: Implementing Progress OpenEdge Table Partitioning

- **Description:** This course teaches the key tasks to partition tables in an OpenEdge RDBMS database. First, you will be introduced to the concepts, types, and tasks of OpenEdge table partitioning. Then, you will learn how to prepare for table partitioning and enable partitioning for a database. Next, you will learn how to create new partitioned tables and partition existing non-partitioned tables. Finally, you will learn how to manage partitions, maintain indexes, and gather statistics for partitioned tables and indexes.
- **Course duration:** Equivalent to 2 days of instructor-led training
- **Audience:** Database Administrators who want to partition Progress OpenEdge RDBMS tables
- **Version compatibility:** This course is compatible with OpenEdge 11.4.
- **After taking this course, you should be able to:**
  - Describe Progress OpenEdge table partitioning.
  - Create new partitioned tables
  - Partition existing tables
  - Manage partitions
  - Maintain indexes
  - Gathering statistics for partitioned tables and indexes

PROGRESS SPARK