SQL intersection

Very Large Tables: Optimizing Performance and Availability through Partitioning

Preconference Workshop #10

SQLintersection Fall 2014

Workshop: Monday, November 10, 2014

written and presented by
Kimberly L. Tripp
http://www.SQLskills.com

SQLskills
immerse yourself in sql server

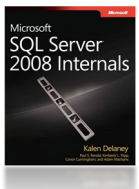
SQLintersection
PRECON10

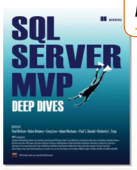
Very Large Tables: Optimizing Performance and Availability through Partitioning

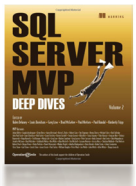
Kimberly L. Tripp
Kimberly@SQLskills.com

SQL intersection

Author/Instructor: Kimberly L. Tripp







MVP Microsoft Most Valuable Professional

Microsoft Regional Director PROGRAM

- **Consultant/Trainer/Speaker/Writer**
- **President/Founder, SYSolutions, Inc.**
  - e-mail: Kimberly@SQLskills.com
  - blog: http://www.SQLskills.com/blogs/Kimberly
  - Twitter: @KimberlyLTripp
- **Author/Instructor for SQL Server Immersion Events**
- **Instructor for multiple rotations of both the SQL MCM & Sharepoint MCM**
- **Author/Manager of SQL Server 2005 & 2008 Launch Content**
- **Author/Speaker at Microsoft TechEd, SQL Connections, SQLPASS, ITForum, TechDays**
- **Author of several SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability, SQLCLR for DBAs**
- **Author/Presenter for more than 25 online webcasts on MSDN and TechNet**
- **Co-author MSPress Title: SQL Server 2008 Internals, the SQL Server MVP Project (1 & 2), and SQL Server 2000 High Availability**
- **Presenter/Technical Manager for SQL Server HOL DVDs**
- **Owner and Manager of the SQLintersection conference**

SQLskills immerse yourself in sql server

SQL intersection

2

Overview

- **Workloads**
  - Hybrid – OLTP with real-time analysis
  - Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated
- **Horizontal partitioning strategies**
  - Partitioned views
  - Partitioned tables
- **Implementing the sliding window scenario**
- **Key partitioning concerns / considerations**
  - Other features and partitioning
  - Partition-aligned indexed views
  - Partitioning vs. filtering
- **Partitioning design techniques combined**

SQL intersection

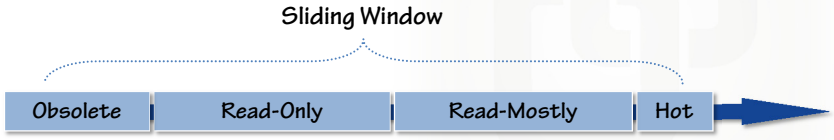
Workloads

- **Hybrid – OLTP with real-time analysis**
  - Larger and larger tables…
  - New data coming in and performance problems with existing / older data
  - Maintenance takes longer and longer to maintain tables (larger and larger amount that's not changing)
  - Management nightmares (backup / restore / downtime)
  - Don't generally consider "partitioning" in OLTP environments (meant more for management of the sliding window scenario) but here it's ideal
- **Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated**
  - Limited to no analysis done on the OLTP server
  - Data is read-only (except for periodic loads) to DSS
  - This is the canonical case for "sliding window" but how you implement this is surprising…

SQL intersection

Time-based Data Patterns

- **Structures designed for "sliding window" (or, "rolling range") scenario**
- **Tables created and data flows on regular / consistent basis (weekly, monthly, yearly, etc.)**
- **Data may be archived / removed to keep only "current" timeframe (year, two years, etc.)**
- **Implementation benefits vary with implementation choice:**
  - Partitioned views
  - Partitioned tables



SQL intersection

5

Horizontal Partitioning: Motivation?

- **Resource contention limitations**
- **Varying access patterns**
- **Maintenance restrictions**
- **Availability requirements**
- **To remove resource blocking or minimize maintenance costs**
- **Important notes:**
  - Partitioning does not automatically mean Distributed Partitioned Views (DPVs)
  - Partitioning is more of a scale-up / manageability enhancement; DPVs are a form of scale-out partitioning

SQL intersection

6

Table's Shared Data/Indexes

- **A single large table…**
  - Presents different types of problems
    - Management
    - Index create / build or rebuilds
    - Backup / restore and recovery
    - Escalation
  - May have different access patterns
    - Some data new – OLTP (inserts / updates for new and current sales)
    - Some data old for historical lookups – small singleton for OTLP lookups or larger for analysis
- **Does NOT have to be one table!**

SQL intersection

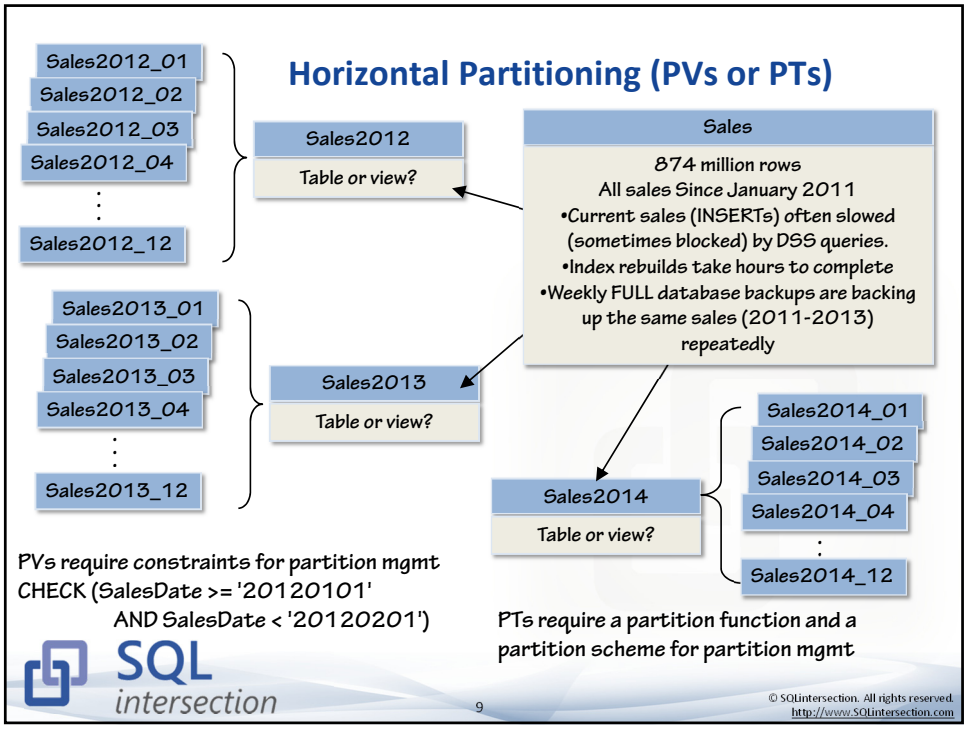
7

Overview

- Workloads
- **Horizontal partitioning strategies**
  - Horizontal partitioning concepts
  - Functionally partitioning data
  - Partitioned views
    - Creation / implementation
    - Concerns
    - Challenges
  - Partitioned tables
    - Creation / implementation
    - Concerns
    - Challenges
- Implementing the sliding window scenario
- Key partitioning concerns / considerations
- Partitioning design techniques combined

SQL intersection

8



Functional Partitioning

- **More granular control**
  - Varying access patterns
  - Varying index defrag patterns
  - Very fast sliding window control
- **More backup/restore choices**
  - File / filegroup backups
  - File / filegroup restores from anything larger
    - Full backups support PAGE, FILE, FILEGROUP or FULL restores
    - Filegroup backups support PAGE, FILE or FILEGROUP restores
    - File backups support PAGE or FILE restores
  - Even support for file / filegroup in simple recovery model (backups separated between RW and RO backups and RO backups can be performed at any frequency!)
- **Significantly better availability**

SQL intersection
10

Functionally Partitioning Data

- **Partitioned tables (requirement: Enterprise Edition)**
  - Can convert an existing table as an ONLINE operation IF the table doesn't have any LOB columns in 2005 / 2008 / R2 (fixed in 2012)
    - Might run into problems around "unique" index requirements for PTs in that the partitioning column must be a member of the key – for all unique indexes
  - Cannot do fast switching in 2005 if Indexed Views
  - Cannot do fast switching if iFTS desired
- **Partitioned views (benefit: available in any edition)**
  - Might be able to replace an existing table with a view (even for DML) if you meet the correct criteria
    - Might not be able to replace all statements, can programmatically direct modifications (for INSERTs)
  - Conversion may require downtime or time where certain data is inaccessible

SQL intersection

11

Partitioned Views

Individual tables are created on filegroups – manually.

FG1

FG2

FG12

Sales201401

ID	SalesDate	...
1	20140101	...
2	20140101	...
3	20140101	...
n	20140131	...

Sales201402

ID	SalesDate	...
1	20140201	...
2	20140201	...
3	20140201	...
n	20140228	...

Sales201412

ID	SalesDate	...
1	20141201	...
2	20141201	...
3	20141201	...
n	20141231	...

The view is logically bringing the data together in a tabular data set.

ID	SalesDate	...
1	20140101	...
2	20140101	...
3	20140101	...
n	20141231	...

```
SELECT * FROM Sales201401
UNION ALL
SELECT * FROM Sales201402
UNION ALL
SELECT * FROM Sales201403
UNION ALL
...
SELECT * FROM Sales201412
```

SQL intersection

12

Partitioned Views

Still Motivation for Creating in SQL Server 2005+

- **Queryable partitioned views added in SQL Server 7.0**
  - Query Optimizer removes irrelevant tables from query plan (partition elimination)
- **Updateable partitioned views added in SQL Server 2000**
  - Inserts / updates / deletes may need to be directed to correct base table (if you can't meet UPV requirements that partitioning key has to be the first column of the primary key)
  - Data modification statements can be directed to appropriate base tables IF the partition key is part of the primary key and it's enforced by a CHECK constraint
- **Requires separately named / created tables**
- **Manual placement on different filegroups**
- **Checked / verified constraints – and tested for logic problems (what will happen on February 29th?)**
- **UNION ALL views**

SQL intersection

13

Creating Partitioned Views

- **Create files / filegroups**
- **Create individual tables on filegroups**
- **Create base table indexes (CL, NCs, views with indexes, and columnstore indexes) as per performance requirements on ALL partitions**
  - It's a pro and a con that each "partition" can have different indexes:
    - PRO: you can reduce indexes for OLTP and increase for DSS
    - CON: it's harder for the optimizer to optimize – each table has to be analyzed individually
- **Create base table constraints to restrict each table's partitioning column**
  - Range must be protected on ALL partitions using a constraint; the constraint MUST be checked and verified (= "trusted" constraints)
  - Can create restrictions on multiple columns and SQL Server can eliminate on any combination of them (you can only do this with partitioned views)
- **Create views (must use UNION ALL (not UNION))**

SQL intersection

14

Check Constraints (Filtering)

- **Defines column's domain of legal values**
- **All data checked on INSERT / UPDATE**
  - Only range constraints supported

```
CHECK (SalesDate >= '20040101'
   AND SalesDate < '20040201')
```

  - Limited subset of range operators supported for partitioning (i.e. filtering) are: BETWEEN, AND, OR, <, <=, >, >=, =
  - Others are supported for data integrity purposes only
- **Added during CREATE TABLE**
- **Add later using ALTER TABLE**

But what about the existing data?

Default is to CHECK the existing data…

Verify Constraint is "Trusted"

- **Check to see if constraint is trusted:**

```
OBJECTPROPERTY(object_id('constraint'),
  'CnstIsNotTrusted')
```

- **Check to see if any data violates constraints:**

```
DBCC CHECKCONSTRAINTS('charge')
DBCC CHECKCONSTRAINTS('constraint')
```

- **However, even if DBCC CHECKCONSTRAINTS returns NO errors or warnings about your data… your constraint is still NOT trusted!**
- **You must DROP / re-CREATE or re-CHECK**

Partitioned View Challenges

- **Administration of separate tables, index creation on many tables**
- **Constraints (and business logic) are not guaranteed with partitioned views: must be managed individually on all tables**
- **Constraints must be trusted: if you disable constraints make sure you not only "enable" but recheck the data**
- **Queries can work well but data modifications may not work through the view**
  - You cannot insert through a PV if the base table has an identity column
    - TIP: Use application-directed inserts
  - Partitioning column must be leading column of PK
    - Some people really hate this but it does have benefits
      - It can help query performance to lead the clustered index with the partitioning column
      - The negative is that you often have to "push-down" a column in related tables that may not have been necessary
- **Harder for the optimizer to optimize as tables could be different (some problems with optimizing certain predicates):**
  - http://blogs.msdn.com/craigfr/archive/2006/11/27/introduction-to-partitioned-tables.aspx (http://bit.ly/Qyu6Zp)
  - http://blogs.msdn.com/sqlcat/archive/2006/02/17/Partition-Elimination-in-SQL-Server-2005.aspx (http://bit.ly/Qyu4Rx)

SQL intersection

17

Stored Procedure Estimates and Recompilation

- **Some features cannot support a single plan or perform well with one that's cached**
  - Filtered objects
    - SQL Server cannot save a single plan based on filtered indexes or statistics
    - SQL Server won't use a filtered object unless recompilation is performed:
      - Using OPTION (RECOMPILE): this is the most safe as it forces the statement to be recompiled and is not affected by the database setting for parameterization
      - Using a dynamically constructed string (unless forced parameterization is on)
  - Partitioned views
    - SQL Server CAN do partition elimination correctly (even if the first execution is only against one partition) so you will get the correct data
    - However, the first execution will set the "estimates" for the plan which could vary across the sets and lead to poor performance
  - CHOOSE
    - This is similar to partitioned views in that you will get the correct data but subsequent executions will have estimation issues

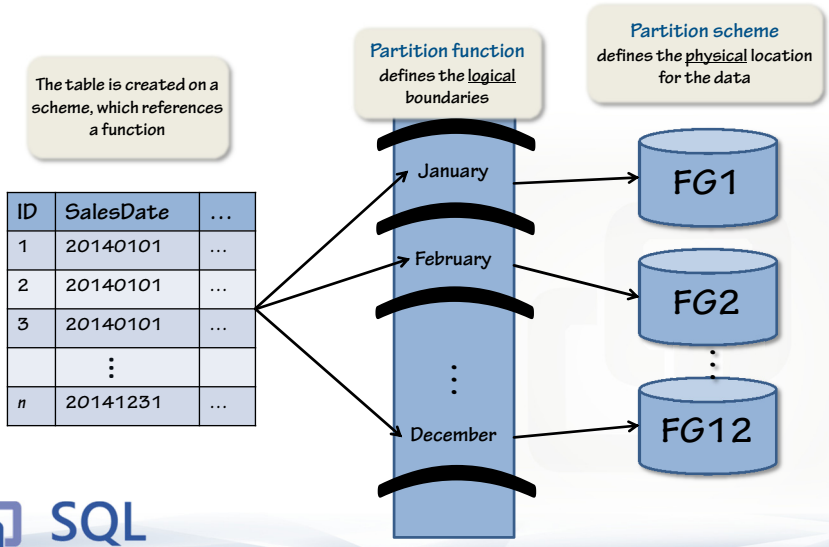
SQL intersection

pluralsight

Slide from Pluralsight course: SQL Server Optimizing Stored Procedure Performance

18

Partitioned Tables



SQL intersection

19

Range Partitioned Tables

- **Scripts from samples, lab and whitepaper (updated) in file:** `PartitioningScripts.ssmssln`
- **Step 1: Create filegroups**
- **Step 2: Create files in filegroups**
- **Step 3: Create partition function**
- **Step 4: Create partition scheme**
- **Step 5: Create table(s) on partition scheme**
- **Step 6: Create index(es) on partition scheme**
- **Optionally, verify data with catalog views and functions**
- **Add data to tables**
  - SQL Server redirects data and queries to appropriate partition

SQL intersection

20

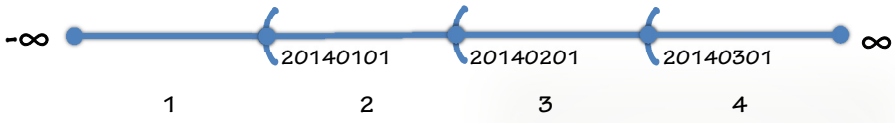
Step 3: Partition Function

- **Not related to scalar / table-valued functions**
- **Must define the data type over which the function will calculate**
  - Partition function can partition over ONLY ONE column of a table
- **Always includes the entire domain of possible values from negative infinity (-∞) to infinity (∞)**
- **Table should use constraints to limit the domain of data values (they are not required as they are with partitioned views)**

SQL intersection

Partition Function

- **Range "right" means the value is contained in the partition to the right**



- **Right-based partitions**
  - Use lower boundaries
- **n boundaries creates n+1 partitions**
- **One table, one set of indexes, simplified management (to a point)**

SQL intersection

Partition Function
Left or Right?

- **Defines whether or not the first boundary point should fall to the LEFT or the RIGHT within the full domain of values (-∞ to ∞)**
- **Values can use functions to calculate boundary point but boundary point will always be stored as a literal value based on calculation of the function at time the partition function is created**
- **Only the boundary point is stored, not the expression used to calculate it**
- **Can see boundary points stored within the catalog view (sys.partition_range_values)**

SQL intersection

Partition Function
Left or Right?

- **LEFT**
  - Defines that the boundary point falls into the FIRST (LEFT, of the first two) partition
  - All other boundary points follow this pattern
  - An "ending point" makes more sense when using LEFT
  - In date-based partitioning, this can be a very frustrating value with which to work
- **Right**
  - Defines that the boundary point falls into the SECOND (RIGHT, of the first two) partition
  - All other boundary points follow this pattern
  - A "beginning point" makes more sense when using RIGHT
  - **In date-based partitioning, this is an easier value to define (recommended)**

SQL intersection

Choosing Upper Boundary (LEFT)

hidden slide w/extra details

Added complexity with datetime data type

- **Upper boundary is inclusive (<=) so must be exact**
- **Datetime value of 23:59:59.999 is not "available" due to the precision with datetime data. If entered it will round to 00:00:00.000 of the next day**
  - 23:59:59.999 rounds up to 00:00:00.000
  - 23:59:59.998 rounds down to 23:59:59.997
  - 23:59:59.997 can be stored and is the last explicit value supported
- **The partition function should use either of these two values for the upper boundary:**
  - date 23:59:59.997
  - DATEADD(ms, -3, date)

SQL intersection

Create the Partition Function

hidden slide w/extra details

Using LEFT

- **Orders and Order Details will include the OrderDate column (yes, duplicated data needed in the Order Details table)**
- **For example, if orders ranged from October 2012 through September 2014**

```
CREATE PARTITION FUNCTION TwoYearDateRangePFN(datetime)
AS
RANGE LEFT FOR VALUES
( '20121031 23:59:59.997',   -- Oct 2012
  '20121130 23:59:59.997',   -- Nov 2012
  '20121231 23:59:59.997',   -- Dec 2012
  '20130131 23:59:59.997',   -- Jan 2013
  '20130228 23:59:59.997',   -- Feb 2013
  ...
  '20140930 23:59:59.997')   -- Sep 2014
```

SQL intersection

Create the Partition Function

hidden slide w/extra details

Using LEFT with DATEADD Function

- Can use a function in place of a literal value
- Function is evaluated at creation and only the boundary point is stored
- See sys.partition_range_values for values

```
CREATE PARTITION FUNCTION TwoYearDateRangePFN(datetime)
AS
RANGE LEFT FOR VALUES
(   dateadd (ms, -3, '20121101') -- Oct 2012
  , dateadd (ms, -3, '20121201') -- Nov 2012
  , dateadd (ms, -3, '20130101') -- Dec 2012
  , dateadd (ms, -3, '20130201') -- Jan 2013
  , dateadd (ms, -3, '20130301') -- Feb 2013
  ...
  , dateadd (ms, -3, '20141001') -- Sep 2014
```

SQL intersection

Create the Partition Function

hidden slide w/extra details

Using RIGHT

- Orders and Order Details will include the OrderDate column (yes, duplicated data needed in the Order Details table)
- For example, if orders ranged from October 2012 through September 2014

```
CREATE PARTITION FUNCTION TwoYearDateRangePFN(datetime)
AS
RANGE RIGHT FOR VALUES
( '20121001',   -- Oct 2012
  '20121101',   -- Nov 2012
  '20121201',   -- Dec 2012
  '20130101',   -- Jan 2013
  '20130201',   -- Feb 2013
...
  '20140901')   -- Sep 2014
```

SQL intersection

hidden slide w/extra details

Partitioning: Date Column

- **SQL Server 2008 added many new date-related data types**
  - date, time, datetime2, etc.
- **Avoids the precision (and therefore, programmability) problem with datetime and partition switching in the sliding window scenario**
- **You no longer needs to write your partition boundaries with '23:59:59.997' to match the last time value in a day**
- **More details about the SQL Server 2005 limitation in the MSDN Whitepaper: Partitioned Tables and Indexes in SQL Server 2005**
  - http://msdn2.microsoft.com/en-us/library/ms345146.aspx
- **Examples using date in the SQL Server 2008 whitepaper:**
  - http://msdn.microsoft.com/en-us/library/dd578580.aspx

SQL intersection

Step 4: Partition Scheme

- **Maps the partitions (number defined by the function) to the filegroups (and therefore files) with the database**
- **Because both the extreme left and extreme right boundary cases are included, a partition function with *n* boundary conditions actually creates *n+1* partitions**
- **This first (in RIGHT) partition will remain empty as data slides / rolls forward**
  - Note: this is not a requirement at the time of definition but it's a good idea if you want your management procedure to be consistent from the first switch procedure

SQL intersection

Step 4: Partition Scheme

- **You can have empty partitions**
  - In RIGHT-based partition functions, the first partition is empty
    - Pro – Your automation routines are the same from the first onward
    - Con – None, except a few extra partitions to define
  - If LEFT-based partition functions, the last partition is empty (similar pros / cons)
- **If using an empty partition:**
  - Use a special "tiny" filegroup
    - Create a file/filegroup that's restricted to 2MB with autogrowth disabled
      - Fast fail if you "MERGE" the wrong boundary point
      - Data should never reside in it
  - Constraints should be used to restrict the table's data.
- **Explicitly name a filegroup for the "active" partitions and then use "tiny" for the empty partition (currently and always to remain empty)**

SQL intersection

31

Step 4: Partition Scheme

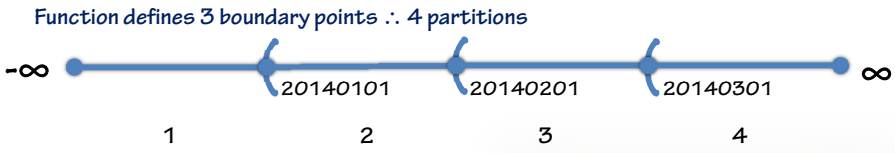
- **If we hold 2 years worth of Order/Order Details data in 24 partitions we will create a partition scheme to handle 25 partitions where the FIRST one will begin/remain empty:**

```
CREATE PARTITION SCHEME [TwoYearDateRangePScheme]
AS PARTITION [TwoYearDateRangePFN] TO
( [Tiny],
  [FG1], [FG2], [FG3], [FG4], [FG5],
  [FG6], [FG7], [FG8], [FG9], [FG10],
  [FG11],[FG12],[FG13],[FG14],[FG15],
  [FG16],[FG17],[FG18],[FG19],[FG20],
  [FG21],[FG22],[FG23],[FG24])
GO
```

SQL intersection

32

Aligned and Storage-Aligned Scheme




```
CREATE PARTITION SCHEME [Scheme1]
AS PARTITION [FNwith3BoundaryPoints]
TO ([Tiny], [FG1], [FG2], [FG3]);
GO
CREATE TABLE [T1] … ON [Scheme1];
CREATE TABLE [T2] … ON [Scheme1];
```

```
CREATE PARTITION SCHEME [Scheme2]
AS PARTITION [FNwith3BoundaryPoints]
TO ([Tiny], [FG4], [FG5], [FG6]);
GO
CREATE TABLE [T3] … ON [Scheme2];
```

- Scheme1 and Scheme2 reference the same function ⇒ Aligned
- If an object uses the same scheme it is said to be Storage Aligned
- T1, T2, and T3 are all aligned
- T1 and T2 are storage aligned

Step 5: Create Table

```
CREATE TABLE [Sales]
  (
   [SalesDate]  [datetime2] NOT NULL
                CONSTRAINT [DF_Sales_SalesDate]
                DEFAULT SYSDATETIME()
  …
  ) ON [PartitionScheme]([SalesDate])
```

- Column has to match data type of partition function
- Table is created on partition scheme
- Can be done with an online operation for an existing table by rebuilding the clustered index ON the partition scheme
  - Nonclustered indexes are un-aligned until manually rebuilt on the same (or aligned) scheme
  - Table is kept online but fast-switching is not allowed until nonclustered are also rebuilt and aligned

Step 6: Create Indexes

```
CREATE UNIQUE CLUSTERED INDEX [SalesPK]
ON [Sales] ([SalesID])
WITH (DROP_EXISTING = ON, ONLINE = ON)
ON [Sales4Partitions_PS]([SalesID])
```

- Can "move" a table to one or more filegroups as an online operation by rebuilding the clustered index on the new scheme WITH ONLINE = ON
  - Note: Online only when there are no LOB columns in the table (fixed in 2012)
- Secondary nonclustered indexes are NOT moved/partitioned
- New indexes are automatically aligned by default but existing indexes (prior to partitioning) must be rebuilt / re-created on the new scheme to be aligned properly
- If clustered index was created through a primary key constraint, you rebuild by using the constraint name and same definition – on new scheme (example above is a primary key)
- Column has to match the exact data type of partition function
- Unique indexes must contain the partitioning column as part of the key (it does not have to be the leading column)

SQL intersection

35

hidden slide w/extra details

Verify Partition Data / Location

- Programmatically determine which partition…

```
SELECT $partition.PFN('date-to-modify')
```

- See all data across all partitions…

```
SELECT $partition.[PFN]([s].[Date]) AS [Partition#]
      , min([s].[Date]) AS [Min Sale Date]
      , max([s].[Date]) AS [Max Sale Date]
      , count(*) AS [Rows In Partition]
FROM [dbo].[Sales] AS [s]
GROUP BY $partition.[PFN]([s].[Date])
ORDER BY [Partition#]
```

- See all data across all partitions…

```
SELECT *
  FROM [sys].[dm_db_index_physical_stats](…)
```

SQL intersection

36

Overview

- Workloads
  - Hybrid – OLTP with real-time analysis
  - Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated
- Horizontal partitioning strategies
  - Partitioned views
  - Partitioned tables
- **Implementing the sliding window scenario**
- Key partitioning concerns / considerations
  - Other features and partitioning
  - Partition-aligned indexed views
  - Partitioning vs. filtering
- Partitioning design techniques combined

SQL intersection

The Sliding Window Scenario

- **New data must come in**
- **Old data must be removed**
- **The table should only change it's overall data set… with simple metadata changes**
- **To create metadata ONLY changes**
  - Create a new table on the same filegroup where the partition resides
  - Structure it IDENTICALLY to that of the partitioned table
  - SWITCH the partition OUT – the data from the partition "moves" to the empty table
  - The empty table to be switched in
    - Is initially created on a staging area
    - Is moved to the final location by creating a clustered index on the appropriate filegroup
    - SWITCH the new table IN so that the newly manipulated data becomes the partition

SQL intersection

Sliding Window Scenario

- **Get stakeholders in a room…**
- **Whiteboard your data flow**
  - Understand your needs first
  - Technology LAST (meaning syntax and actual implementation)
- **I'll "whiteboard" what I mean…**
  - The following few hidden slides are from the SQL Server 2005 Partitioning whitepaper

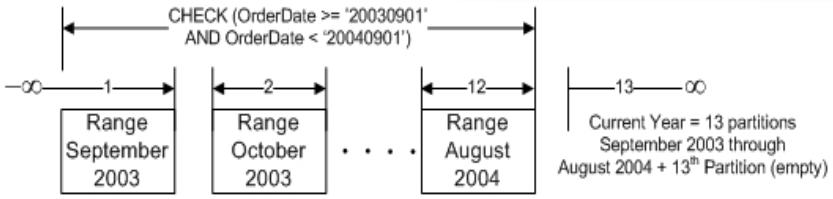
Extra notes:

The Sliding Window: Switch

hidden slide w/extra details

See whitepaper for complete text for these diagrams

- **Whiteboard discussion…**
- **For speed in manipulation, the sliding window must be implemented as a metadata ONLY "switch" (no data should ever move on SPLIT / MERGE)**



- **Most importantly, make sure to "staging" tables on the same <u>physical</u> filegroup as the partition you are switching IN or OUT…**

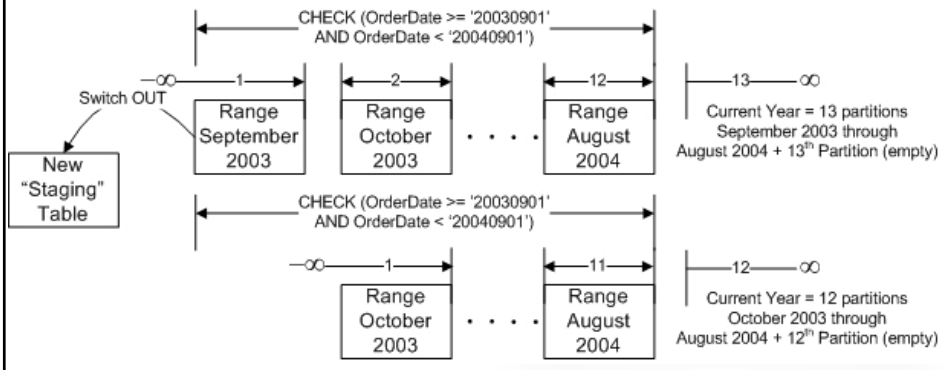
SQL intersection

The Sliding Window: Switch

hidden slide w/extra details

See whitepaper for complete text for these diagrams

- **Create new table (same structure / indexes)**
- **Switch OUT the old data**



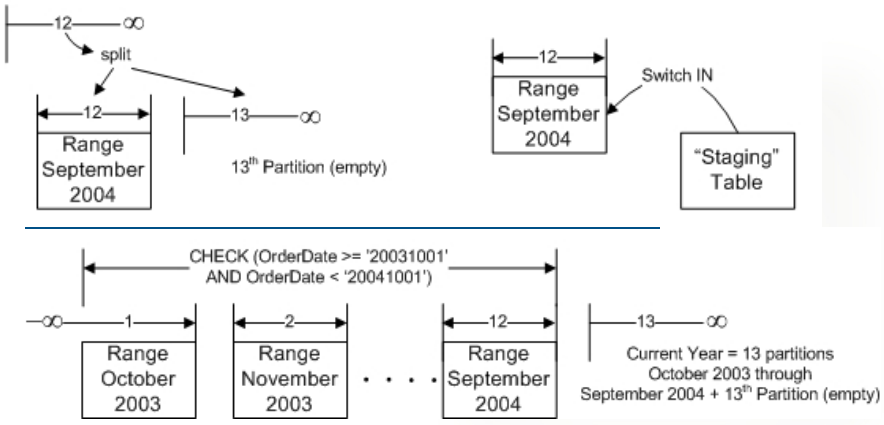
SQL intersection

The Sliding Window: Switch

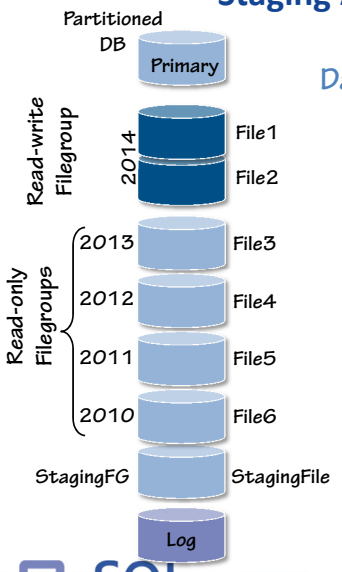
hidden slide w/extra details

See whitepaper for complete text for these diagrams

- **Create new table as a heap, load data, build indexes, then switch IN the current data**



Staging Area/Loading in a Heap



Database *consists of…*
Filegroups *consist of…*
Files *consist of…*
Objects *are placed on a filegroup*

Switch in as a partition! (3)
⇦Move to final FG (2)
Create CL index on FG
Remember – the CL Index IS the data
⇦Data load (1)
Create heap on staging FG

Why Load Into a Staging Area?

- **Log space required depends on recovery model:**
  - 3.25x for FULL recovery model = Log will have size of data
  - 2.25x for BULK_LOGGED or SIMPLE = Minimally logged operation
- **If you load into destination file then you will have a gap at the beginning of the file after the clustered index is created…**
- **How do you get rid of this gap?**
- **Shrink??**
- **NOOOOOOOOOOOOOO! ☺**

SQL intersection

The Sliding Window Summary (1)

1) **If you're removing data, prepare the "staging out" table**
2) **If you're bringing data in**
   - Load / transform / cleanse – in a staging area
   - Move the data by building the clustered index on the destination filegroup
3) **Alter the partition scheme to set NEXT USED (based on the filegroup where you created the clustered index above)**
4) **Split the function to add the new boundary point (this will put the boundary point on the filegroup specified in the scheme's next used setting). IMPORTANT NOTE: be sure that NO data needs to move (if using LEFT, then the partitioning split should be empty)**

Note: none of these first few steps impact the actual table!

SQL intersection

The Sliding Window Summary (2)

5) **Switch "IN" the new data**
6) **Switch "OUT" the old data**

   Note: steps 5 and 6 can be in either order – and are FAST (re: metadata only operations)

7) **Backup / remove the old data (drop the table)**

   Note: file-level backups can be restored with the primary filegroup to another location but not directly into another database [you must INSERT / SELECT])

8) **Merge the boundary point (from the emptied [switched out] partition)**

- **NOTE: If the "merge" process is slow then it's likely that you had data in the partition being merged. Be careful never to split or merge where there's already data; the data will need to be relocated (and the process of moving the data is slow).**

SQL intersection

47

The Sliding Window Key Points

- **The staging tables must match in every way**
  - Same column types
  - Same indexes (and ALL indexes)
  - SQL Server 2008+, all indexed views and their indexes
  - SQL Server 2012+, all columnstore indexes
- **The staging tables must be on the same filegroup at the time of the switch (can build the data in a staging area first and then move to destination)**
- **Be sure that NO data needs to move… test!**
- **One final special consideration for the table that you're switching IN: it must have a "trusted" constraint to guarantee the data matches the partition into which it's switching**

SQL intersection

48

Overview

- Workloads
  - Hybrid – OLTP with real-time analysis
  - Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated
- Horizontal partitioning strategies
  - Partitioned views
  - Partitioned tables
- Implementing the sliding window scenario
- **Key partitioning concerns / considerations**
  - Other features and partitioning
  - Partition-aligned indexed views
  - Partitioning vs. filtering
- Partitioning design techniques combined

SQL intersection

Special Considerations for Partitioned Tables

- **If read-write portion is only current month then that's the only place where fragmentation will occur**
- **If current month is September then only rebuild September:**

```
ALTER INDEX [ChargesPTPK] ON [ChargesPT]
REBUILD PARTITION = 9
WITH (ONLINE = ON)
```

Msg 155, Level 15, State 1, Line 3
'ONLINE' is not a recognized ALTER INDEX REBUILD PARTITION option.

- **Fixed in SQL Server 2014 – you CAN rebuild a partition as an online operation. But, does that actually solve ALL of the problems with partitioning?**

SQL intersection

Online Operations, Indexes, and Partitioning

- **Partitioning and online operations are Enterprise-only (Enterprise, Enterprise Eval, Developer)**
- **Table-level index rebuilds can be done online if there are no LOB columns in the table (fixed in 2012)**
- **Rebuilding/partitioning the clustered index does NOT rebuild any of the nonclustered indexes – these must be rebuilt on the new scheme as well**
  - Until the nonclustered indexes are rebuilt the partitioned table will not support fast-switching; everything else will work!
  - Unique indexes must have the partitioning column as a member of the unique key (to support fast-switching). If you don't need fast-switching then you don't need to do this!

SQL intersection

Other Features and Partitioning

- **iFTS**
  - PTs: cannot do fast-switching if PT has iFTS index(es)
- **Indexed views**
  - 2005 PTs: cannot do fast-switching if PT has IVs – must drop, switch, recreate (nightmare! fixed in 2008)
- **Identity columns**
  - PVs: cannot use UPVs for INSERT if they have an identity column on base tables (must use application directed INSERTs)
  - PTs: can have an identity
- **Multiple partitioning columns**
  - PTs: no, only one
  - PVs: yes, as many as makes sense!

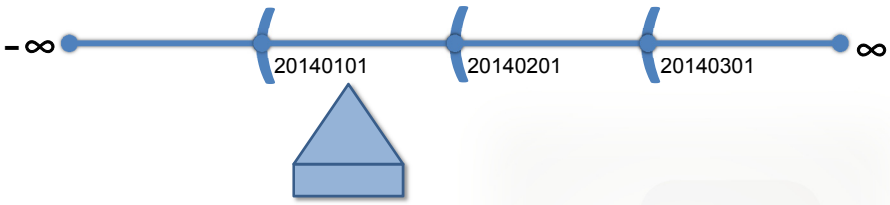
SQL intersection

Partition-Aligned Index Views

- **SQL Server 2005 does not support partition switching while index views are defined**
  - Drop the index(es) to switch partitions ☹
  - Then re-create the index(es) once switched
  - SLOW to do (and those queries using the indexed views are slow during this period too!)
- **SQL Server 2008 has partition-aligned indexed views**
  - Allows the switch to happen with a meta-data operation (including the indexed view partition data)
  - Much better support for Data Warehouse maintenance

SQL intersection

Partitioned Table with Filtered Indexes



Fast-switching is NOT allowed when filtered indexes are created on individual sets (and therefore NOT aligned):

```
CREATE INDEX [SubsetOfItems]
  ON [Table] ([col1], [col2])
  WHERE [Date] >= '20140101'
  AND [Date] < '20140201'
```

SQL intersection

Interval Subsumption

Partitioning / Fast-Switching

- **Undesirable to use filtered indexes over specific sets due to the lack of fast-switching**
- **Can use a filter over the ENTIRE set (and aligned) (e.g. status = 1)**
- **Filtered indexes cannot be used when the query's predicate is not a subset of a particular filter interval:**
  - Filter: January data

```
WHERE date >= '20140101'
   AND date < '20140201'
```

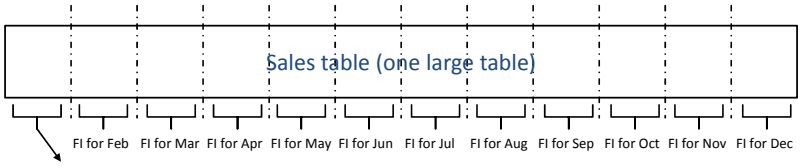
  - Filter: February data

```
WHERE date >= '20140201'
   AND date < '20140301 '
```

  - Predicates and their usage:

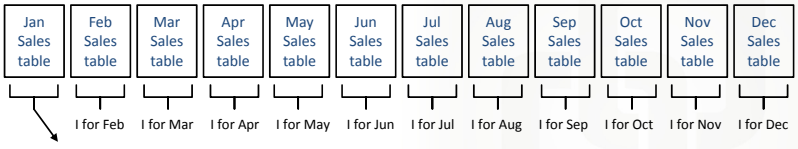
```
WHERE date = '20140115'  OK
WHERE date BETWEEN '20140105'
   AND '20140115'  OK
WHERE date BETWEEN '20140115'
   AND '20140215'  Cannot be used
```

SQL intersection



Filtered Index for January

```
CREATE INDEX JanuaryItems
ON Sales (col1, col2)
WHERE SalesDate >= '20120101'
        AND SalesDate < '20120201'
```



Non-filtered index for January

```
CREATE INDEX JanuaryItems
ON Sales (col1, col2)
```

NOTE: Each table has a constraint

```
ALTER TABLE JanSalesTable
ADD CONSTRAINT JanuaryItems
CHECK (SalesDate >= '20120101'
        AND SalesDate < '20120201')
```

SQL intersection

Overview

- Workloads
  - Hybrid – OLTP with real-time analysis
  - Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated
- Horizontal partitioning strategies
  - Partitioned views
  - Partitioned tables
- Implementing the sliding window scenario
- Key partitioning concerns / considerations
  - Other features and partitioning
  - Partition-aligned indexed views
  - Partitioning vs. filtering
- **Partitioning design techniques combined**

SQL intersection

57

Partitioning: PVs vs. PTs

Partitioned views

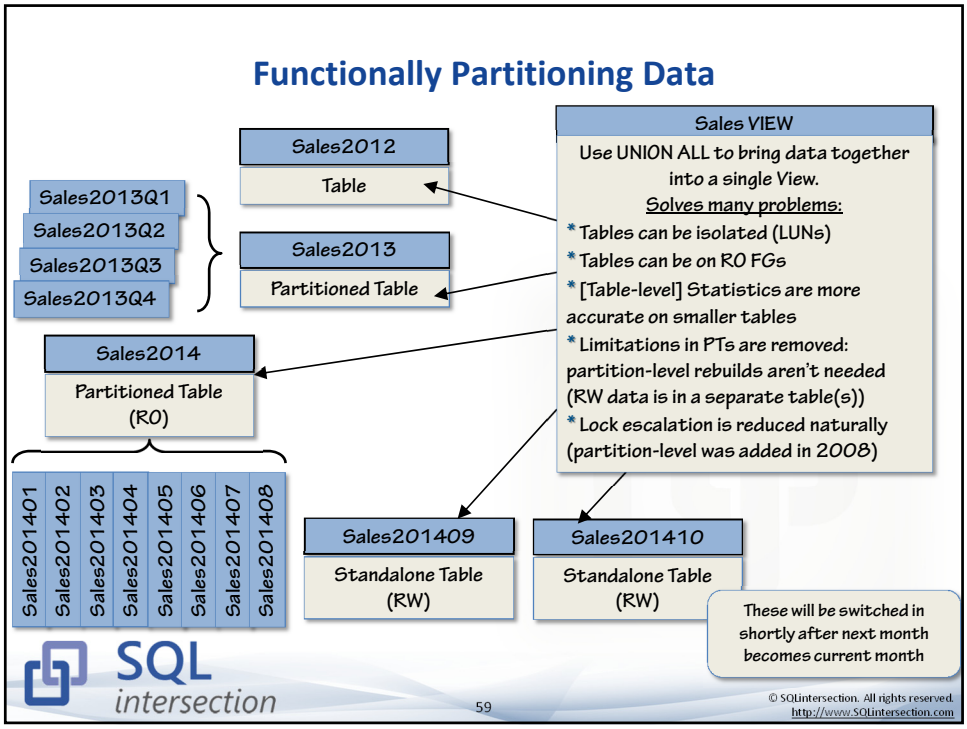
- Any edition
- Lots of tables to administer
  - Must create/drop indexes on all base tables
    - Can have different indexes
    - Harder for the optimizer to optimize with so many indexes
  - Must verify business logic so that there are no gaps or overlapping values
  - Each table has [potentially] better statistics as the tables are smaller
- Can rebuild any of the tables ONLINE (if using EE)
- Can support multiple constraints on one or more columns

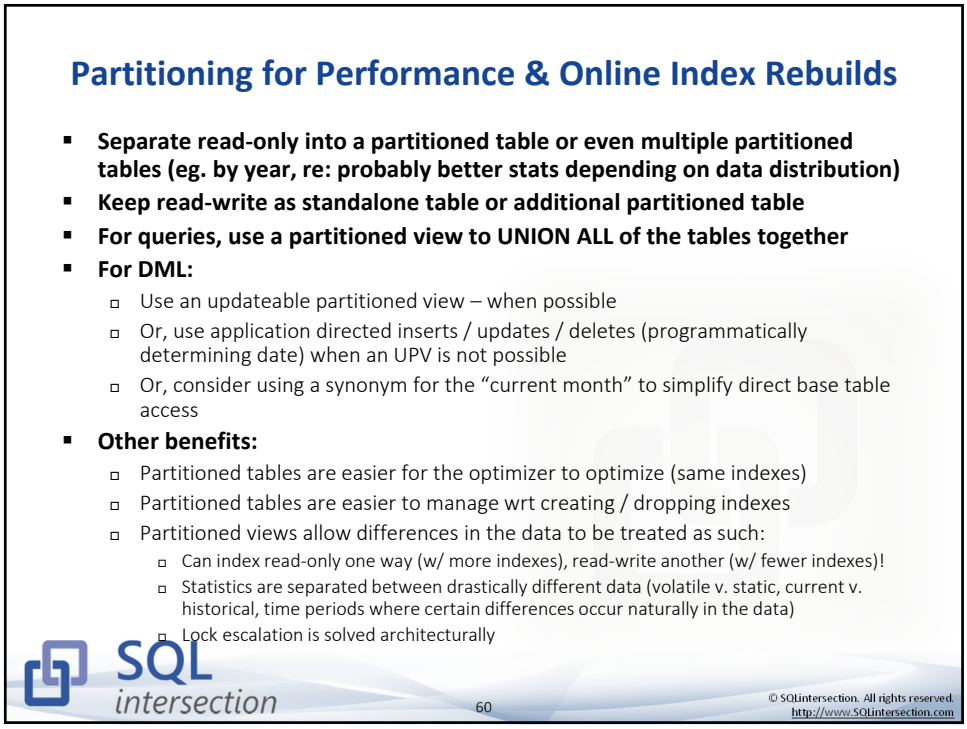
Partitioned tables

- Enterprise Edition only
- Only 1 table to administer
  - Only 1 table to create/drop indexes
    - All partitions have same indexes (which is easier for the optimizer to optimize)
    - Can create different indexes with filtered indexes
  - No possibility of errors (or gaps or overlapping values)
  - Table-level statistics can be less accurate for very large tables when there's a lot of skew to the data
- Partition-level rebuilds are offline but can rebuild the ENTIRE table online (not desirable)
- Can only support partitioning over a single column

SQL intersection

58


Functionally Partitioning Data
Sales2012
Table
Sales2013Q1
Sales2013Q2
Sales2013Q3
Sales2013Q4
Sales2013
Partitioned Table
Sales2014
Partitioned Table (RO)
Sales201401
Sales201402
Sales201403
Sales201404
Sales201405
Sales201406
Sales201407
Sales201408
Sales VIEW
Use UNION ALL to bring data together into a single View.
Solves many problems:
* Tables can be isolated (LUNs)
* Tables can be on RO FGs
* [Table-level] Statistics are more accurate on smaller tables
* Limitations in PTs are removed: partition-level rebuilds aren't needed (RW data is in a separate table(s))
* Lock escalation is reduced naturally (partition-level was added in 2008)
Sales201409
Standalone Table (RW)
Sales201410
Standalone Table (RW)
These will be switched in shortly after next month becomes current month
SQL intersection
59
© SQLintersection. All rights reserved.
http://www.SQLintersection.com


Partitioning for Performance & Online Index Rebuilds
Separate read-only into a partitioned table or even multiple partitioned tables (eg. by year, re: probably better stats depending on data distribution)
Keep read-write as standalone table or additional partitioned table
For queries, use a partitioned view to UNION ALL of the tables together
For DML:
Use an updateable partitioned view – when possible
Or, use application directed inserts / updates / deletes (programmatically determining date) when an UPV is not possible
Or, consider using a synonym for the "current month" to simplify direct base table access
Other benefits:
Partitioned tables are easier for the optimizer to optimize (same indexes)
Partitioned tables are easier to manage wrt creating / dropping indexes
Partitioned views allow differences in the data to be treated as such:
Can index read-only one way (w/ more indexes), read-write another (w/ fewer indexes)!
Statistics are separated between drastically different data (volatile v. static, current v. historical, time periods where certain differences occur naturally in the data)
Lock escalation is solved architecturally
SQL intersection
60
© SQLintersection. All rights reserved.
http://www.SQLintersection.com

Review

- **Workloads**
  - Hybrid – OLTP with real-time analysis
  - Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated
- **Horizontal partitioning strategies**
  - Partitioned views
  - Partitioned tables
- **Implementing the sliding window scenario**
- **Partitioning design techniques combined**
- **Partition-aligned indexed views**
- **Optimizing sets**
  - Other features and partitioning
  - Partitioning vs. filtering

SQL intersection

Self-paced Study and Exercises

- **Review the MCM Online Videos for recap**
  - Partitioning (1 hr, 15min) http://bit.ly/1A00QkS
- **Review the workshop content**
  - Review the zip – posted to: http://www.sqlskills.com/sql-server-resources/sql-server-demos/
  - Review the whiteboard drawings
  - Review the demo scripts
- **Go through the HOL lab from your demo scripts**
- **Read the SQL Server 2005 whitepaper on Partitioning http://bit.ly/1qlD49k**
- **Read the SQL Server 2008 whitepaper on Partitioning (docx link) http://bit.ly/1A01UFq**

SQL intersection


Questions?
SQLskills
immerse yourself in sql server
Don't forget to complete an online evaluation on EventBoard!
PRECON10
Very Large Tables: Optimizing Performance
and Availability through Partitioning
Your evaluation helps organizers build better conferences
and helps speakers improve their sessions.
SQL
intersection
Thank you!