Table Hints (Transact-SQL)

Table hints override the default behavior of the query optimizer for the duration of the data manipulation language (DML) statement by specifying a locking method, one or more indexes, a query-processing operation such as a table scan or index seek, or other options. Table hints are specified in the FROM clause of the DML statement and affect only the table or view referenced in that clause.

Caution

Because the SQL Server query optimizer typically selects the best execution plan for a query, we recommend that hints be used only as a last resort by experienced developers and database administrators.

Applies to:

DELETE[1]

INSERT[2]

SELECT[3]

UPDATE[4]

MERGE[5]

Transact-SQL Syntax Conventions[6]

Syntax

Arguments

WITH **(** <table_hint> **)** [[**,**]...*n*]

With some exceptions, table hints are supported in the FROM clause only when the hints are specified with the WITH keyword. Table hints also must be specified with parentheses.

> **Important**
>
> Omitting the WITH keyword is a deprecated feature: This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

The following table hints are allowed with and without the WITH keyword: NOLOCK, READUNCOMMITTED, UPDLOCK, REPEATABLEREAD, SERIALIZABLE, READCOMMITTED, TABLOCK, TABLOCKX, PAGLOCK, ROWLOCK, NOWAIT, READPAST, XLOCK, and NOEXPAND. When these table hints are specified without the WITH keyword, the hints should be specified alone. For example:

```
FROM t (TABLOCK)
```

When the hint is specified with another option, the hint must be specified with the WITH keyword:

```
FROM t WITH (TABLOCK, INDEX(myindex))
```

We recommend using commas between table hints.

> **Important**
>
> Separating hints by spaces rather than commas is a deprecated feature: This feature will be removed in a future version of Microsoft SQL Server. Do not use this feature in new development work, and modify applications that currently use this feature as soon as possible.

The restrictions apply when the hints are used in queries against databases with the compatibility level of 90 and higher.

NOEXPAND

Specifies that any indexed views are not expanded to access underlying tables when the query optimizer processes the query. The query optimizer treats the view like a table with clustered index. NOEXPAND applies only to indexed views. For more information, see Remarks.

INDEX **(***index_value* [**,**... *n*]) | INDEX = (*index_value***)**

The INDEX() syntax specifies the names or IDs of one or more indexes to be used by the query optimizer when it processes the statement. The alternative INDEX = syntax specifies a single index value. Only one index hint per table can be specified.

If a clustered index exists, INDEX(0) forces a clustered index scan and INDEX(1) forces a clustered index scan or seek. If no clustered index exists, INDEX(0) forces a table scan and INDEX(1) is interpreted as an error.

If multiple indexes are used in a single hint list, the duplicates are ignored and the rest of the listed indexes are used to retrieve the rows of the table. The order of the indexes in the index hint is significant. A multiple index hint also enforces index ANDing, and the query optimizer applies as many conditions as possible on each index accessed. If

the collection of hinted indexes do not include all columns referenced by the query, a fetch is performed to retrieve the remaining columns after the SQL Server Database Engine retrieves all the indexed columns.

> **Note**
>
> When an index hint referring to multiple indexes is used on the fact table in a star join, the optimizer ignores the index hint and returns a warning message. Also, index ORing is not allowed for a table with an index hint specified.

The maximum number of indexes in the table hint is 250 nonclustered indexes.

KEEPIDENTITY

Is applicable only in an INSERT statement when the BULK option is used with OPENROWSET[7].

Specifies that identity value or values in the imported data file are to be used for the identity column. If KEEPIDENTITY is not specified, the identity values for this column are verified but not imported and the query optimizer automatically assigns unique values based on the seed and increment values specified during table creation.

> **Important**
>
> If the data file does not contain values for the identity column in the table or view, and the identity column is not the last column in the table, you must skip the identity column. For more information, see Use a Format File to Skip a Data Field (SQL Server)[8]. If an identity column is skipped successfully, the query optimizer automatically assigns unique values for the identity column into the imported table rows.

For an example that uses this hint in an INSERT ... SELECT * FROM OPENROWSET (BULK...) statement, see Keep Identity Values When Bulk Importing Data (SQL Server)[9].

For information about checking the identity value for a table, see DBCC CHECKIDENT (Transact-SQL)[10].

KEEPDEFAULTS

Is applicable only in an INSERT statement when the BULK option is used with OPENROWSET[7].

Specifies insertion of a table column's default value, if any, instead of NULL when the data record lacks a value for the column.

For an example that uses this hint in an INSERT ... SELECT * FROM OPENROWSET (BULK...) statement, see Keep Nulls or UseDefault Values During Bulk Import (SQL Server)[11].

FORCESEEK [**(***index_value***(***index_column_name* [**,**... *n*] **))**]

Specifies that the query optimizer use only an index seek operation as the access path to the data in the table or view. Starting with SQL Server 2008 R2 SP1, index

parameters can also be specified. In that case, the query optimizer considers only index seek operations through the specified index using at least the specified index columns.

index_value

Is the index name or index ID value. The index ID 0 (heap) cannot be specified. To return the index name or ID, query the **sys.indexes** catalog view.

index_column_name

Is the name of the index column to include in the seek operation. Specifying FORCESEEK with index parameters is similar to using FORCESEEK with an INDEX hint. However, you can achieve greater control over the access path used by the query optimizer by specifying both the index to seek on and the index columns to consider in the seek operation. The optimizer may consider additional columns if needed. For example, if a nonclustered index is specified, the optimizer may choose to use clustered index key columns in addition to the specified columns.

The FORCESEEK hint can be specified in the following ways.

Syntax	Example	Description
Without an index or INDEX hint	`FROM dbo.MyTable WITH (FORCESEEK)`	The query optimizer considers only index seek operations to access the table or view through any relevant index.
Combined with an INDEX hint	`FROM dbo.MyTable WITH (FORCESEEK, INDEX (MyIndex))`	The query optimizer considers only index seek operations to access the table or view through the specified index.
Parameterized by specifying an index and index columns	`FROM dbo.MyTable WITH (FORCESEEK (MyIndex (col1, col2, col3)))`	The query optimizer considers only index seek operations to access the table or view through the specified index using at least the specified index columns.

When using the FORCESEEK hint (with or without index parameters), consider the following guidelines.

- The hint can be specified as a table hint or as a query hint. For more information about query hints, see Query Hints (Transact-SQL)[12].
- To apply FORCESEEK to an indexed view, the NOEXPAND hint must also be specified.
- The hint can be applied at most once per table or view.
- The hint cannot be specified for a remote data source. Error 7377 is returned when FORCESEEK is specified with an index hint and error 8180 is returned when FORCESEEK is used without an index hint.
- If FORCESEEK causes no plan to be found, error 8622 is returned.

When FORCESEEK is specified with index parameters, the following guidelines and restrictions apply.

- The hint cannot be specified in combination with either an INDEX hint or another FORCESEEK hint.
- At least one column must be specified and it must be the leading key column.
- Additional index columns can be specified, however, key columns cannot be skipped. For example, if the specified index contains the key columns `a`, `b`, and `c`, valid syntax would include `FORCESEEK (MyIndex (a))` and `FORCESEEK (MyIndex (a, b)`. Invalid syntax would include `FORCESEEK (MyIndex (c))` and `FORCESEEK (MyIndex (a, c)`.
- The order of column names specified in the hint must match the order of the columns in the referenced index.
- Columns that are not in the index key definition cannot be specified. For example, in a nonclustered index, only the defined index key columns can be specified. Clustered key columns that are automatically included in the index cannot be specified, but may be used by the optimizer.
- An xVelocity memory optimized columnstore index cannot be specified as an index parameter. Error 366 is returned.
- Modifying the index definition (for example, by adding or removing columns) may require modifications to the queries that reference that index.
- The hint prevents the optimizer from considering any spatial or XML indexes on the table.
- The hint cannot be specified in combination with the FORCESCAN hint.
- For partitioned indexes, the partitioning column implicitly added by SQL Server cannot be specified in the FORCESEEK hint.

Caution

Specifying FORCESEEK with parameters limits the number of plans that can be considered by the optimizer more than when specifying FORCESEEK without parameters. This may cause a "Plan cannot be generated" error to occur in more cases. In a future release, internal modifications to the optimizer may allow more plans to be considered.

FORCESCAN

Introduced in SQL Server 2008 R2 SP1, this hint specifies that the query optimizer use only an index scan operation as the access path to the referenced table or view. The FORCESCAN hint can be useful for queries in which the optimizer underestimates the number of affected rows and chooses a seek operation rather than a scan operation. When this occurs, the amount of memory granted for the operation is too small and query performance is impacted.

FORCESCAN can be specified with or without an INDEX hint. When combined with an index hint, (`INDEX = index_name, FORCESCAN`), the query optimizer considers only scan access paths through the specified index when accessing the referenced table. FORCESCAN can be specified with the index hint INDEX(0) to force a table scan operation on the base table.

For partitioned tables and indexes, FORCESCAN is applied after partitions have been

eliminated through query predicate evaluation. This means that the scan is applied only to the remaining partitions and not to the entire table.

The FORCESCAN hint has the following restrictions.

- The hint cannot be specified for a table that is the target of an INSERT, UPDATE, or DELETE statement.
- The hint cannot be used with more than one index hint.
- The hint prevents the optimizer from considering any spatial or XML indexes on the table.
- The hint cannot be specified for a remote data source.
- The hint cannot be specified in combination with the FORCESEEK hint.

HOLDLOCK

Is equivalent to SERIALIZABLE. For more information, see SERIALIZABLE later in this topic. HOLDLOCK applies only to the table or view for which it is specified and only for the duration of the transaction defined by the statement that it is used in. HOLDLOCK cannot be used in a SELECT statement that includes the FOR BROWSE option.

IGNORE_CONSTRAINTS

Is applicable only in an INSERT statement when the BULK option is used with OPENROWSET[7].

Specifies that any constraints on the table are ignored by the bulk-import operation. By default, INSERT checks Unique Constraints and Check Constraints[13] and Primary and Foreign Key Constraints[14]. When IGNORE_CONSTRAINTS is specified for a bulk-import operation, INSERT must ignore these constraints on a target table. Note that you cannot disable UNIQUE, PRIMARY KEY, or NOT NULL constraints.

You might want to disable CHECK and FOREIGN KEY constraints if the input data contains rows that violate constraints. By disabling the CHECK and FOREIGN KEY constraints, you can import the data and then use Transact-SQL statements to clean up the data.

However, when CHECK and FOREIGN KEY constraints are ignored, each ignored constraint on the table is marked as **is_not_trusted** in the sys.check_constraints[15] or sys.foreign_keys[16] catalog view after the operation. At some point, you should check the constraints on the whole table. If the table was not empty before the bulk import operation, the cost of revalidating the constraint may exceed the cost of applying CHECK and FOREIGN KEY constraints to the incremental data.

IGNORE_TRIGGERS

Is applicable only in an INSERT statement when the BULK option is used with OPENROWSET[7].

Specifies that any triggers defined on the table are ignored by the bulk-import operation. By default, INSERT applies triggers.

Use IGNORE_TRIGGERS only if your application does not depend on any triggers and

maximizing performance is important.

NOLOCK

Is equivalent to READUNCOMMITTED. For more information, see READUNCOMMITTED later in this topic.

> **Note**
>
> For UPDATE or DELETE statements: This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

NOWAIT

Instructs the Database Engine to return a message as soon as a lock is encountered on the table. NOWAIT is equivalent to specifying SET LOCK_TIMEOUT 0 for a specific table.

PAGLOCK

Takes page locks either where individual locks are ordinarily taken on rows or keys, or where a single table lock is ordinarily taken. By default, uses the lock mode appropriate for the operation. When specified in transactions operating at the SNAPSHOT isolation level, page locks are not taken unless PAGLOCK is combined with other table hints that require locks, such as UPDLOCK and HOLDLOCK.

READCOMMITTED

Specifies that read operations comply with the rules for the READ COMMITTED isolation level by using either locking or row versioning. If the database option READ_COMMITTED_SNAPSHOT is OFF, the Database Engine acquires shared locks as data is read and releases those locks when the read operation is completed. If the database option READ_COMMITTED_SNAPSHOT is ON, the Database Engine does not acquire locks and uses row versioning. For more information about isolation levels, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL)[17].

> **Note**
>
> For UPDATE or DELETE statements: This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

READCOMMITTEDLOCK

Specifies that read operations comply with the rules for the READ COMMITTED isolation level by using locking. The Database Engine acquires shared locks as data is read and releases those locks when the read operation is completed, regardless of the setting of the READ_COMMITTED_SNAPSHOT database option. For more information about isolation levels, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL)[17]. This hint cannot be specified on the target table of an INSERT statement; error 4140 is returned.

READPAST

Specifies that the Database Engine not read rows that are locked by other transactions. When READPAST is specified, row-level locks are skipped. That is, the Database Engine skips past the rows instead of blocking the current transaction until the locks are

released. For example, assume table T1 contains a single integer column with the values of 1, 2, 3, 4, 5. If transaction A changes the value of 3 to 8 but has not yet committed, a SELECT * FROM T1 (READPAST) yields values 1, 2, 4, 5. READPAST is primarily used to reduce locking contention when implementing a work queue that uses a SQL Server table. A queue reader that uses READPAST skips past queue entries locked by other transactions to the next available queue entry, without having to wait until the other transactions release their locks.

READPAST can be specified for any table referenced in an UPDATE or DELETE statement, and any table referenced in a FROM clause. When specified in an UPDATE statement, READPAST is applied only when reading data to identify which records to update, regardless of where in the statement it is specified. READPAST cannot be specified for tables in the INTO clause of an INSERT statement. Read operations that use READPAST do not block. Update or delete operations that use READPAST may block when reading foreign keys or indexed views, or when modifying secondary indexes.

READPAST can only be specified in transactions operating at the READ COMMITTED or REPEATABLE READ isolation levels. When specified in transactions operating at the SNAPSHOT isolation level, READPAST must be combined with other table hints that require locks, such as UPDLOCK and HOLDLOCK.

The READPAST table hint cannot be specified when the READ_COMMITTED_SNAPSHOT database option is set to ON and either of the following conditions is true.

- The transaction isolation level of the session is READ COMMITTED.
- The READCOMMITTED table hint is also specified in the query.

To specify the READPAST hint in these cases, remove the READCOMMITTED table hint if present, and include the READCOMMITTEDLOCK table hint in the query.

READUNCOMMITTED

Specifies that dirty reads are allowed. No shared locks are issued to prevent other transactions from modifying data read by the current transaction, and exclusive locks set by other transactions do not block the current transaction from reading the locked data. Allowing dirty reads can cause higher concurrency, but at the cost of reading data modifications that then are rolled back by other transactions. This may generate errors for your transaction, present users with data that was never committed, or cause users to see records twice (or not at all).

READUNCOMMITTED and NOLOCK hints apply only to data locks. All queries, including those with READUNCOMMITTED and NOLOCK hints, acquire Sch-S (schema stability) locks during compilation and execution. Because of this, queries are blocked when a concurrent transaction holds a Sch-M (schema modification) lock on the table. For example, a data definition language (DDL) operation acquires a Sch-M lock before it modifies the schema information of the table. Any concurrent queries, including those running with READUNCOMMITTED or NOLOCK hints, are blocked when attempting to acquire a Sch-S lock. Conversely, a query holding a Sch-S lock blocks a concurrent transaction that attempts to acquire a Sch-M lock.

READUNCOMMITTED and NOLOCK cannot be specified for tables modified by insert, update, or delete operations. The SQL Server query optimizer ignores the READUNCOMMITTED and NOLOCK hints in the FROM clause that apply to the target table of an UPDATE or DELETE statement.

Note

> Support for use of the READUNCOMMITTED and NOLOCK hints in the FROM clause that apply to the target table of an UPDATE or DELETE statement will be removed in a future version of SQL Server. Avoid using these hints in this context in new development work, and plan to modify applications that currently use them.

You can minimize locking contention while protecting transactions from dirty reads of uncommitted data modifications by using either of the following:

- The READ COMMITTED isolation level with the READ_COMMITTED_SNAPSHOT database option set ON.
- The SNAPSHOT isolation level.

For more information about isolation levels, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL)[17].

> **Note**
>
> If you receive the error message 601 when READUNCOMMITTED is specified, resolve it as you would a deadlock error (1205), and retry your statement.

REPEATABLEREAD

Specifies that a scan is performed with the same locking semantics as a transaction running at REPEATABLE READ isolation level. For more information about isolation levels, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL)[17].

ROWLOCK

Specifies that row locks are taken when page or table locks are ordinarily taken. When specified in transactions operating at the SNAPSHOT isolation level, row locks are not taken unless ROWLOCK is combined with other table hints that require locks, such as UPDLOCK and HOLDLOCK.

SPATIAL_WINDOW_MAX_CELLS = *integer*

Specifies the maximum number of cells to use for tessellating a geometry or geography object. *number* is a value between 1 and 8192.

This option allows for fine-tuning of query execution time by adjusting the tradeoff between primary and secondary filter execution time. A larger number reduces secondary filter execution time, but increases primary execution filter time and a smaller number decreases primary filter execution time, but increase secondary filter execution. For denser spatial data, a higher number should produce a faster execution time by giving a better approximation with the primary filter and reducing secondary filter execution time. For sparser data, a lower number will decrease the primary filter execution time.

This option works for both manual and automatic grid tessellations.

SERIALIZABLE

Is equivalent to HOLDLOCK. Makes shared locks more restrictive by holding them until a transaction is completed, instead of releasing the shared lock as soon as the required table or data page is no longer needed, whether the transaction has been completed or not. The scan is performed with the same semantics as a transaction running at the

SERIALIZABLE isolation level. For more information about isolation levels, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL)[17].

TABLOCK

Specifies that the acquired lock is applied at the table level. The type of lock that is acquired depends on the statement being executed. For example, a SELECT statement may acquire a shared lock. By specifying TABLOCK, the shared lock is applied to the entire table instead of at the row or page level. If HOLDLOCK is also specified, the table lock is held until the end of the transaction.

When importing data into a heap by using the INSERT INTO <target_table> SELECT <columns> FROM <source_table> statement, you can enable optimized logging and locking for the statement by specifying the TABLOCK hint for the target table. In addition, the recovery model of the database must be set to simple or bulk-logged. For more information, see INSERT (Transact-SQL)[2].

When used with the OPENROWSET[7] bulk rowset provider to import data into a table, TABLOCK enables multiple clients to concurrently load data into the target table with optimized logging and locking. For more information, see Prerequisites for Minimal Logging in Bulk Import[18].

TABLOCKX

Specifies that an exclusive lock is taken on the table.

UPDLOCK

Specifies that update locks are to be taken and held until the transaction completes. UPDLOCK takes update locks for read operations only at the row-level or page-level. If UPDLOCK is combined with TABLOCK, or a table-level lock is taken for some other reason, an exclusive (X) lock will be taken instead.

When UPDLOCK is specified, the READCOMMITTED and READCOMMITTEDLOCK isolation level hints are ignored. For example, if the isolation level of the session is set to SERIALIZABLE and a query specifies (UPDLOCK, READCOMMITTED), the READCOMMITTED hint is ignored and the transaction is run using the SERIALIZABLE isolation level.

XLOCK

Specifies that exclusive locks are to be taken and held until the transaction completes. If specified with ROWLOCK, PAGLOCK, or TABLOCK, the exclusive locks apply to the appropriate level of granularity.

Remarks

The table hints are ignored if the table is not accessed by the query plan. This may be caused by the optimizer choosing not to access the table at all, or because an indexed view is accessed instead. In the latter case, accessing an indexed view can be prevented by using the OPTION (EXPAND VIEWS) query hint.

All lock hints are propagated to all the tables and views that are accessed by the query plan, including tables and views referenced in a view. Also, SQL Server performs the corresponding lock consistency checks.

Lock hints ROWLOCK, UPDLOCK, AND XLOCK that acquire row-level locks may place locks on index keys rather than the actual data rows. For example, if a table has a nonclustered index, and a SELECT statement using a lock hint is handled by a covering index, a lock is acquired on the index key in the covering index rather than on the data row in the base table.

If a table contains computed columns that are computed by expressions or functions accessing columns in other tables, the table hints are not used on those tables and are not propagated. For example, a NOLOCK table hint is specified on a table in the query. This table has computed columns that are computed by a combination of expressions and functions that access columns in another table. The tables referenced by the expressions and functions do not use the NOLOCK table hint when accessed.

SQL Server does not allow for more than one table hint from each of the following groups for each table in the FROM clause:

- Granularity hints: PAGLOCK, NOLOCK, READCOMMITTEDLOCK, ROWLOCK, TABLOCK, or TABLOCKX.
- Isolation level hints: HOLDLOCK, NOLOCK, READCOMMITTED, REPEATABLEREAD, SERIALIZABLE.

Filtered Index Hints

A filtered index can be used as a table hint, but will cause the query optimizer to generate error 8622 if it does not cover all of the rows that the query selects. The following is an example of an invalid filtered index hint. The example creates the filtered index `FIBillOfMaterialsWithComponentID` and then uses it as an index hint for a SELECT statement. The filtered index predicate includes data rows for ComponentIDs 533, 324, and 753. The query predicate also includes data rows for ComponentIDs 533, 324, and 753 but extends the result set to include ComponentIDs 855 and 924, which are not in the filtered index. Therefore, the query optimizer cannot use the filtered index hint and generates error 8622. For more information, see Create Filtered Indexes[19].

```
USE AdventureWorks2012;
GO
IF EXISTS (SELECT name FROM sys.indexes
    WHERE name = N'FIBillOfMaterialsWithComponentID'
    AND object_id = OBJECT_ID(N'Production.BillOfMaterials'))
DROP INDEX FIBillOfMaterialsWithComponentID
    ON Production.BillOfMaterials;
GO
CREATE NONCLUSTERED INDEX "FIBillOfMaterialsWithComponentID"
    ON Production.BillOfMaterials (ComponentID, StartDate, EndDate)
    WHERE ComponentID IN (533, 324, 753);
GO
SELECT StartDate, ComponentID FROM Production.BillOfMaterials
    WITH( INDEX (FIBillOfMaterialsWithComponentID) )
    WHERE ComponentID in (533, 324, 753, 855, 924);
GO
```

The query optimizer will not consider an index hint if the SET options do not have the required values for filtered indexes. For more information, see CREATE INDEX (Transact-SQL)[20].

Using NOEXPAND

NOEXPAND applies only to *indexed views*. An indexed view is a view with a unique clustered index created on it. If a query contains references to columns that are present both in an indexed view and base tables, and the query optimizer determines that using the indexed view provides the best method for executing the query, the query optimizer uses the index on the view. This function is called *indexed view matching*. Automatic use of indexed view by query optimizer is supported only in specific editions of SQL Server. For a list of features that are supported by the editions of SQL Server, see Features Supported by the Editions of SQL Server 2012[21] (http://go.microsoft.com/fwlink/?linkid=232473).

However, for the optimizer to consider indexed views for matching, or use an indexed view that is referenced with the NOEXPAND hint, the following SET options must be set to ON.

ANSI_NULLS	ANSI_WARNINGS	CONCAT_NULL_YIELDS_NULL
ANSI_PADDING	ARITHABORT[1]	QUOTED_IDENTIFIERS

[1] ARITHABORT is implicitly set to ON when ANSI_WARNINGS is set to ON. Therefore, you do not have to manually adjust this setting.

Also, the NUMERIC_ROUNDABORT option must be set to OFF.

To force the optimizer to use an index for an indexed view, specify the NOEXPAND option. This hint can be used only if the view is also named in the query. SQL Server does not provide a hint to force a particular indexed view to be used in a query that does not name the view directly in the FROM clause; however, the query optimizer considers using indexed views, even if they are not referenced directly in the query.

Using a Table Hint as a Query Hint

Table hints can also be specified as a query hint by using the OPTION (TABLE HINT) clause. We recommend using a table hint as a query hint only in the context of a plan guide[22]. For ad-hoc queries, specify these hints only as table hints. For more information, see Query Hints (Transact-SQL)[12].

Permissions

The KEEPIDENTITY, IGNORE_CONSTRAINTS, and IGNORE_TRIGGERS hints require ALTER permissions on the table.

Examples

A. Using the TABLOCK hint to specify a locking method

The following example specifies that a shared lock is taken on the `Production.Product` table and is held until the end of the UPDATE statement.

Transact-SQL

B. Using the FORCESEEK hint to specify an index seek operation

The following example uses the FORCESEEK hint without specifying an index to force the query optimizer to perform an index seek operation on the `Sales.SalesOrderDetail` table.

Transact-SQL

```
USE AdventureWorks2012;
GO
SELECT *
FROM Sales.SalesOrderHeader AS h
INNER JOIN Sales.SalesOrderDetail AS d WITH (FORCESEEK)
    ON h.SalesOrderID = d.SalesOrderID
WHERE h.TotalDue > 100
AND (d.OrderQty > 5 OR d.LineTotal < 1000.00);
GO
```

The following example uses the FORCESEEK hint with an index to force the query optimizer to perform an index seek operation on the specified index and index column.

Transact-SQL

```
USE AdventureWorks2012;
GO
SELECT h.SalesOrderID, h.TotalDue, d.OrderQty
FROM Sales.SalesOrderHeader AS h
    INNER JOIN Sales.SalesOrderDetail AS d
    WITH (FORCESEEK (PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID (SalesOrderI
D)))
    ON h.SalesOrderID = d.SalesOrderID
WHERE h.TotalDue > 100
AND (d.OrderQty > 5 OR d.LineTotal < 1000.00);
GO
```

C. Using the FORCECAN hint to specify an index scan operation

The following example uses the FORCESCAN hint to force the query optimizer to perform a scan operation on the Sales.SalesOrderDetail table.

Transact-SQL

See Also

Reference

OPENROWSET (Transact-SQL)[7]
Hints (Transact-SQL)[23]
Query Hints (Transact-SQL)[12]

Links Table

[1]*http://msdn.microsoft.com/en-us/library/ms189835.aspx*

[2]*http://msdn.microsoft.com/en-us/library/ms174335.aspx*

[3]*http://msdn.microsoft.com/en-us/library/ms189499.aspx*

[4]*http://msdn.microsoft.com/en-us/library/ms177523.aspx*

[5]*http://msdn.microsoft.com/en-us/library/bb510625.aspx*

[6]*http://msdn.microsoft.com/en-us/library/ms177563.aspx*

[7]*http://msdn.microsoft.com/en-us/library/ms190312.aspx*

[8]*http://msdn.microsoft.com/en-us/library/ms187908.aspx*

[9]*http://msdn.microsoft.com/en-us/library/ms186335.aspx*

[10]*http://msdn.microsoft.com/en-us/library/ms176057.aspx*

[11]*http://msdn.microsoft.com/en-us/library/ms187887.aspx*

[12]*http://msdn.microsoft.com/en-us/library/ms181714.aspx*

[13]*http://msdn.microsoft.com/en-us/library/ms187550.aspx*

[14]*http://msdn.microsoft.com/en-us/library/ms179610.aspx*

[15]*http://msdn.microsoft.com/en-us/library/ms187388.aspx*

[16]*http://msdn.microsoft.com/en-us/library/ms189807.aspx*

[17]*http://msdn.microsoft.com/en-us/library/ms173763.aspx*

[18]*http://msdn.microsoft.com/en-us/library/ms190422.aspx*

[19]*http://msdn.microsoft.com/en-us/library/cc280372.aspx*

[20]*http://msdn.microsoft.com/en-us/library/ms188783.aspx*

[21]*http://go.microsoft.com/fwlink/?linkid=232473*

[22] *http://msdn.microsoft.com/en-us/library/ms190417.aspx*

[23] *http://msdn.microsoft.com/en-us/library/ms187713.aspx*

Community Content