<https://docs.microsoft.com/en-us/sql/t-sql/data-types/hierarchyid-data-type-method-reference>

# hierarchyid Data Type Method Reference

2017-3-14 6 min to read Contributors

* [BYHAM](https://github.com/BYHAM)

* [Craig Guyer](https://github.com/craigg-msft)

**THIS TOPIC APPLIES TO:**yesSQL Server (starting with 2008)yesAzure SQL DatabasenoAzure SQL Data Warehouse noParallel Data Warehouse

The **hierarchyid** data type is a variable length, system data type. Use **hierarchyid** to represent position in a hierarchy. A column of type **hierarchyid** does not automatically represent a tree. It is up to the application to generate and assign **hierarchyid** values in such a way that the desired relationship between rows is reflected in the values.

A value of the **hierarchyid** data type represents a position in a tree hierarchy. Values for **hierarchyid** have the following properties:

* Extremely compact

The average number of bits that are required to represent a node in a tree with n nodes depends on the average fanout (the average number of children of a node). For small fanouts (0-7), the size is about 6\*logAn bits, where A is the average fanout. A node in an organizational hierarchy of 100,000 people with an average fanout of 6 levels takes about 38 bits. This is rounded up to 40 bits, or 5 bytes, for storage.

* Comparison is in depth-first order

Given two **hierarchyid** values **a** and **b**, **a<b** means a comes before b in a depth-first traversal of the tree. Indexes on **hierarchyid** data types are in depth-first order, and nodes close to each other in a depth-first traversal are stored near each other. For example, the children of a record are stored adjacent to that record. For more information, see [Hierarchical Data (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/hierarchical-data-sql-server).

* Support for arbitrary insertions and deletions

By using the [GetDescendant](https://docs.microsoft.com/en-us/sql/t-sql/data-types/getdescendant-database-engine) method, it is always possible to generate a sibling to the right of any given node, to the left of any given node, or between any two siblings. The comparison property is maintained when an arbitrary number of nodes is inserted or deleted from the hierarchy. Most insertions and deletions preserve the compactness property. However, insertions between two nodes will produce hierarchyid values with a slightly less compact representation.

* The encoding used in the **hierarchyid** type is limited to 892 bytes. Consequently, nodes which have too many levels in their representation to fit into 892 bytes cannot be represented by the **hierarchyid** type.

The **hierarchyid** type is available to CLR clients as the **SqlHierarchyId** data type.

## Remarks

The **hierarchyid** type logically encodes information about a single node in a hierarchy tree by encoding the path from the root of the tree to the node. Such a path is logically represented as a sequence of node labels of all children visited after the root. A slash starts the representation, and a path that only visits the root is represented by a single slash. For levels underneath the root, each label is encoded as a sequence of integers separated by dots. Comparison between children is performed by comparing the integer sequences separated by dots in dictionary order. Each level is followed by a slash. Therefore a slash separates parents from their children. For example, the following are valid **hierarchyid** paths of lengths 1, 2, 2, 3, and 3 levels respectively:

* /
* /1/
* /0.3.-7/
* /1/3/
* /0.1/0.2/

Nodes can be inserted in any location. Nodes inserted after **/1/2/** but before **/1/3/** can be represented as **/1/2.5/**. Nodes inserted before 0 have the logical representation as a negative number. For example, a node that comes before **/1/1/** can be represented as **/1/-1/**. Nodes cannot have leading zeros. For example, **/1/1.1/** is valid, but **/1/1.01/** is not valid. To prevent errors, insert nodes by using the [GetDescendant](https://docs.microsoft.com/en-us/sql/t-sql/data-types/getdescendant-database-engine) method.

## Data Type Conversion

The **hierarchyid** data type can be converted to other data types as follows:

* Use the [ToString()](https://docs.microsoft.com/en-us/sql/t-sql/data-types/tostring-database-engine) method to convert the **hierarchyid** value to the logical representation as a **nvarchar(4000)** data type.
* Use [Read ()](https://docs.microsoft.com/en-us/sql/t-sql/data-types/read-database-engine) and [Write ()](https://docs.microsoft.com/en-us/sql/t-sql/data-types/write-database-engine) to convert **hierarchyid** to **varbinary**.
* To transmit **hierarchyid** parameters through SOAP first cast them as strings.

## Upgrading Databases

When a database is upgraded to SQL Server 2017, the new assembly and the **hierarchyid** data type will automatically be installed. Upgrade advisor rules detect any user type or assemblies with conflicting names. The upgrade advisor will advise renaming of any conflicting assembly, and either renaming any conflicting type, or using two-part names in the code to refer to that preexisting user type.

If a database upgrade detects a user assembly with conflicting name, it will automatically rename that assembly and put the database into suspect mode.

If a user type with conflicting name exists during the upgrade, no special steps are taken. After the upgrade, both the old user type, and the new system type, will exist. The user type will be available only through two-part names.

## Using hierarchyid Columns in Replicated Tables

Columns of type **hierarchyid** can be used on any replicated table. The requirements for your application depend on whether replication is one directional or bidirectional, and on the versions of SQL Server that are used.

### One-Directional Replication

One-directional replication includes snapshot replication, transactional replication, and merge replication in which changes are not made at the Subscriber. How **hierachyid** columns work with one directional replication depends on the version of SQL Server the Subscriber is running.

* A SQL Server 2017 Publisher can replicate **hierachyid** columns to a SQL Server 2017 Subscriber without any special considerations.
* A SQL Server 2017 Publisher must convert **hierarchyid** columns to replicate them to a Subscriber that is running SQL Server Compact or an earlier version of SQL Server. SQL Server Compact and earlier versions of SQL Server do not support **hierarchyid** columns. If you are using one of these versions, you can still replicate data to a Subscriber. To do this, you must set a schema option or the publication compatibility level (for merge replication) so the column can be converted to a compatible data type.

Column filtering is supported in both of these scenarios. This includes filtering out **hierarchyid** columns. Row filtering is supported as long as the filter does not include a **hierarchyid** column.

### Bi-Directional Replication

Bi-directional replication includes transactional replication with updating subscriptions, peer-to-peer transactional replication, and merge replication in which changes are made at the Subscriber. Replication lets you configure a table with **hierarchyid** columns for bi-directional replication. Note the following requirements and considerations.

* The Publisher and all Subscribers must be running SQL Server 2017.
* Replication replicates the data as bytes and does not perform any validation to assure the integrity of the hierarchy.
* The hierarchy of the changes that were made at the source (Subscriber or Publisher) is not maintained when they replicate to the destination.
* The hash values for **hierarchyid** columns are specific to the database in which they are generated. Therefore, the same value could be generated on the Publisher and Subscriber, but it could be applied to different rows. Replication does not check for this condition, and there is no built-in way to partition **hierarchyid** column values as there is for IDENTITY columns. Applications must use constraints or other mechanisms to avoid such undetected conflicts.
* It is possible that rows that are inserted on the Subscriber will be orphaned. The parent row of the inserted row might have been deleted at the Publisher. This results in an undetected conflict when the row from the Subscriber is inserted at the Publisher.
* Column filters cannot filter out non-nullable **hierarchyid** columns: inserts from the Subscriber will fail because there is no default value for the **hierarchyid** column on the Publisher.
* Row filtering is supported as long as the filter does not include a **hierarchyid** column.

## See Also

[Hierarchical Data (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/hierarchical-data-sql-server)  
[hierarchyid Data Type Method Reference](http://msdn.microsoft.com/library/01a050f5-7580-4d5f-807c-7f11423cbb06)