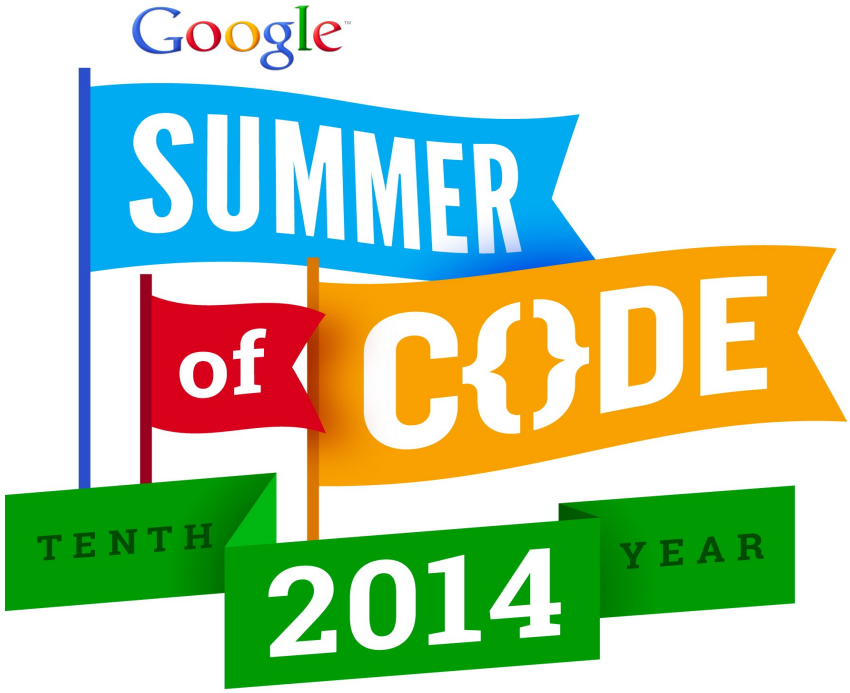

SP-GiST Indexing on PostGIS

Mohit Kumar

Sandro Santilli

Sp-gist indexing

SP-GiST is an abbreviation for space-partitioned GiST. SP-GiST supports partitioned search trees, which facilitate development of a wide range of different non-balanced data structures, such as quad-trees, k-d trees, and radix trees (tries). The common feature of these structures is that they repeatedly divide the search space into partitions that need not be of equal size. Searches that are well matched to the partitioning rule can be very fast.

The challenge addressed by SP-GiST is to map search tree nodes to disk pages in such a way that a search need access only a few disk pages, even if it traverses many nodes.

Like GiST, SP-GiST is meant to allow the development of custom data types with the appropriate access methods, this work extends the spgist quad tree implementation to a geometry data column in a spatially enabled postgresql/postgis database.

SP-GiST offers an interface with a high level of abstraction, requiring the access method developer to implement only methods specific to a given data type which in our case was initially box2df.

Leaf tuples of an SP-GiST tree contain values of the same data type as the indexed column. Leaf tuples at the root level will always contain the original indexed data value, but leaf tuples at lower levels might contain only a compressed representation, such as a suffix. In that case the operator class support functions must be able to reconstruct the original value using information accumulated from the inner tuples that are passed through to reach the leaf level.

Inner tuples are more complex, since they are branching points in the search tree. Each inner tuple contains a set of one or more nodes, which represent groups of similar leaf values. A node contains a downlink that leads to either another, lower-level inner tuple, or a short list of leaf tuples that all lie on the same index page. Each node has a label that describes it; for example, in a radix tree the node label could be the next character of the string value. Optionally, an inner tuple can have a prefix value that describes all its members. In a radix tree this could be the common prefix of the represented strings. The prefix value is not necessarily really a prefix, but can be any data needed by the operator class; for example, in a quad-tree it can store the central point that the four quadrants are measured with respect to. A quad-tree inner tuple would then also contain four nodes corresponding to the quadrants around this central point.

There are five user-defined methods that an index operator class for SP-GiST must provide. All five follow the convention of accepting two internal arguments, the first of which is a pointer to a C struct containing input values for the support method, while the second argument is a pointer to a C struct where output values must be placed

The methods must not modify any fields of their input structs. In all cases, the output struct is initialized to zeroes before calling the user-defined method.

The five user-defined methods are called the support functions and are as follows.

SP-GiST Support Functions

Function	*Description*	*Support No*
config	providebasic information about the operator class	1
choose	determine how to insert a new value into an inner tuple	2
picksplit	determine how to partition a set of values	3
inner_consistent	determine which sub-partitions need to be searched for a query	4
leaf_consistent	determine whether key satisfies the query qualifier	5

Config (geometry_spgist_config_2d)
Returns static information about the index implementation, including the data type OIDs of the prefix and node label data types.

Sql declaration :
CREATE OR REPLACE FUNCTION geometry_spgist_config_2d(internal,internal)
RETURNS void
AS 'MODULE_PATHNAME' ,'geometry_spgist_config_2d'
LANGUAGE 'c';

Arguments :
Input : spgConfigIn
- *Oid attType Data type to be indexed*

Output : spgConfigOut
- *Oid prefixType Data type of inner-tuple prefixes*
- *Oid labelType Data type of inner-tuple node labels*
- *bool canReturnData Opclass can reconstruct original data*
- *bool longValuesOK Opclass can cope with values > 1 page*

choose (geometry_spgist_choose_2d)
Chooses a method for inserting a new value into an inner tuple.

Sql declaration :
CREATE OR REPLACE FUNCTION geometry_spgist_choose_2d(internal,internal)
RETURNS void
AS 'MODULE_PATHNAME' ,'geometry_spgist_choose_2d'
LANGUAGE 'c';

Arguments :
Input : spgChooseIn
- *Datum datum; original datum to be indexed*
- *Datum leafDatum; current datum to be stored at leaf*
- *int level; current level (counting from zero)*
  (Data from current inner tuple)
- *bool allTheSame; tuple is marked all-the-same?*
- *bool hasPrefix; tuple has a prefix?*
- *Datum prefixDatum; if so, the prefix value*
- *int nNodes; number of nodes in the inner tuple*
- *Datum *nodeLabels; node label values (NULL if none)*

Output : spgChooseOut

- *Match node* *descend into existing node*
  - *int nodeN;* *descend to this node (index from 0)*
  - *int levelAdd;* *increment level by this much*
  - *Datum restDatum;* *new leaf datum*
- *spgAddNode* *add a node to the inner tuple*
  - *Datum nodeLabel* *new node's label*
  - *int nodeN* *where to insert it (index from 0) */*
- *spgSplitTuple* *split inner tuple (change its prefix)*
  (Info to form new inner tuple with one node)
  - *bool prefixHasPrefix* *tuple should have a prefix?*
  - *Datum prefixPrefixDatum* *if so, its value*
  - *Datum nodeLabel* *node's label*
  (Info to form new lower-level inner tuple with all old nodes)
  - *bool postfixHasPrefix* *tuple should have a prefix?*
  - *Datum postfixPrefixDatum* *if so, its value*
  -

The choose function can determine either that the new value matches one of the existing child nodes, or that a new child node must be added, or that the new value is inconsistent with the tuple prefix and so the inner tuple must be split to create a less restrictive prefix.

If the new value matches one of the existing child nodes, set resultType to spgMatchNode. Set nodeN to the index (from zero) of that node in the node array. Set levelAdd to the increment in level caused by descending through that node, or leave it as zero if the operator class does not use levels. Set restDatum to equal datum if the operator class does not modify datums from one level to the next, or otherwise set it to the modified value to be used as leafDatum at the next level.

If a new child node must be added, set resultType to spgAddNode. Set nodeLabel to the label to be used for the new node, and set nodeN to the index (from zero) at which to insert the node in the node array. After the node has been added, the choose function will be called again with the modified inner tuple; that call should result in an spgMatchNode result.

If the new value is inconsistent with the tuple prefix, set resultType to spgSplitTuple. This action moves all the existing nodes into a new lower-level inner tuple, and replaces the existing inner tuple with a tuple having a single node that links to the new lower-level inner tuple. Set prefixHasPrefix to indicate whether the new upper tuple should have a prefix, and if so set prefixPrefixDatum to the prefix value. This new prefix value must be sufficiently less restrictive than the original to accept the new value to be indexed, and it should be no longer than the original prefix

picksplit (geometry_spgist_picksplit_2d)
Decides how to create a new inner tuple over a set of leaf tuples.

Sql declaration :

```
CREATE OR REPLACE FUNCTION geometry_spgist_picksplit_2d(internal,internal)
        RETURNS void
        AS 'MODULE_PATHNAME' ,'geometry_spgist_picksplit_2d'
        LANGUAGE 'c';
```

Arguments :
Input : spgPickSplitIn

- *int nTuples;* *number of leaf tuples*

- *Datum *datums; their datums (array of length nTuples)*
- *int level; current level (counting from zero)*

Output : spgPickSplitOut

- *bool hasPrefix; new inner tuple should have a prefix?*
- *Datum prefixDatum; if so, its value*
-
- *int nNodes; number of nodes for new inner tuple*
- *Datum *nodeLabels; their labels (or NULL for no labels)*

- *int *mapTuplesToNodes; node index for each leaf tu*
- *Datum *leafTupleDatums; datum to store in each new leaf tuple*

If more than one leaf tuple is supplied, it is expected that the picksplit function will classify them into more than one node; otherwise it is not possible to split the leaf tuples across multiple pages, which is the ultimate purpose of this operation. Therefore, if the picksplit function ends up placing all the leaf tuples in the same node, the core SP-GiST code will override that decision and generate an inner tuple in which the leaf tuples are assigned at random to several identically-labeled nodes. Such a tuple is marked allTheSame to signify that this has happened. The choose and inner_consistent functions must take suitable care with such inner tuples.

inner_consistent (geometry_spgist_inner_consistent_2d)
Returns set of nodes (branches) to follow during tree search.

SQL declaration:
CREATE OR REPLACE FUNCTION geometry_spgist_inner_consistent_2d(internal,internal)
RETURNS void
AS 'MODULE_PATHNAME' ,'geometry_spgist_inner_consistent_2d'
LANGUAGE 'c';

Arguments:
Input : spgInnerConsistentIn

- *ScanKey scankeys; array of operators and comparison values*
- *int nkeys; length of array*
- *Datum reconstructedValue; value reconstructed at parent*
- *int level; current level (counting from zero)*
- *bool returnData; original data must be returned?*
  (Data from current inner tuple)
- *bool allTheSame; tuple is marked all-the-same?*
- *bool hasPrefix; tuple has a prefix?*
- *Datum prefixDatum; if so, the prefix value*
- *int nNodes; number of nodes in the inner tuple*
- *Datum *nodeLabels; node label values (NULL if none)*

Output : spgInnerConsistentOut

- *int nNodes number of child nodes to be visited*
- *int *nodeNumbers their indexes in the node array*
- *int *levelAdds increment level by this much for each*
- *Datum *reconstructedValues associated reconstructed values*

The array scankeys, of length nkeys, describes the index search condition(s). These conditions are combined with AND — only index entries that satisfy all of them are interesting. (Note that nkeys = 0 implies that all index entries satisfy the query.) reconstructedValue is the value reconstructed for the parent tuple; it is (Datum) 0 at the root level or if the inner_consistent function did not provide a value at the parent level. level is the current inner tuple's level, starting at zero for the root level. returnData is true if reconstructed data is required for this query; this will only be so if the config function asserted canReturnData. allTheSame is true if the current inner tuple is marked "all-the-same"; in this case all the nodes have the same label. hasPrefix is true if the current inner tuple contains a prefix; if so, prefixDatum is its value. nNodes is the number of child nodes contained in the inner tuple, and nodeLabels is an array of their label values, or NULL if the nodes do not have labels.

nNodes must be set to the number of child nodes that need to be visited by the search, and nodeNumbers must be set to an array of their indexes. If the operator class keeps track of levels, set levelAdds to an array of the level increments required when descending to each node to be visited. (Often these increments will be the same for all the nodes, but that's not necessarily so, so an array is used.) If value reconstruction is needed, set reconstructedValues to an array of the values reconstructed for each child node to be visited; otherwise, leave reconstructedValues as NULL. Note that the inner_consistent function is responsible for palloc'ing the nodeNumbers, levelAdds and reconstructedValues arrays.

leaf_consistent (geometry_spgist_leaf_consistent_2d)
Returns true if a leaf tuple satisfies a query.

SQL declaration:
```
CREATE OR REPLACE FUNCTION geometry_spgist_leaf_consistent_2d(internal,internal)
        RETURNS bool
        AS 'MODULE_PATHNAME' ,'geometry_spgist_leaf_consistent_2d'
        LANGUAGE 'c';
```

Arguments:
Input : spgLeafConsistentIn

- *ScanKey scankeys;* — *array of operators and comparison values*
- *int nkeys;* — *length of array*
- *Datum reconstructedValue;* — *value reconstructed at parent*
- *int level;* — *current level (counting from zero)*
- *bool returnData;* — *original data must be returned?*
- *Datum leafDatum;* — *datum in leaf tuple*

Output : spgLeafConsistentOut

- *Datum leafValue;* — *reconstructed original data, if any*
- *bool recheck;* — *set true if operator must be rechecked*

The array scankeys, of length nkeys, describes the index search condition(s). These conditions are combined with AND — only index entries that satisfy all of them satisfy the query. (Note that nkeys = 0 implies that all index entries satisfy the query.) reconstructedValue is the value reconstructed for the parent tuple; it is (Datum) 0 at the root level or if the inner_consistent function did not provide a value at the parent level. level is the current leaf tuple's level, starting at zero for the root level. returnData is true if reconstructed data is required for this query; this will only be so if the config function asserted canReturnData. leafDatum is the key value stored in the current leaf tuple.

The function must return true if the leaf tuple matches the query, or false if not. In the true case, if returnData is true then leafValue must be set to the value originally supplied to be indexed for this leaf tuple. Also, recheck may be set to true if the match is uncertain and so the operator(s) must be re-applied to the actual heap tuple to verify the match.

Index Method Strategies :

The operators associated with an operator class are identified by "strategy numbers", which serve to identify the semantics of each operator within the context of its operator class. Because PostgreSQL allows the user to define operators, PostgreSQL cannot look at the name of an operator (e.g., < or >=) and tell what kind of comparison it is. Instead, the index method defines a set of "strategies", which can be thought of as generalized operators. Each operator class specifies which actual operator corresponds to each strategy for a particular data type and interpretation of the index semantics. The point strategies for Spgist are as follows :

Operation	**Strategy Number**
strictly left of	*1*
strictly right of	*5*
same	*6*
contained by	*8*
strictly below	*10*
strictly above	*11*

Operator Class

As spgist does not support an index storage type different from the target type, in the operator class geometry was used as the storage datatype.

```
CREATE OPERATOR CLASS spgist_geometry_ops_2d
        DEFAULT FOR TYPE geometry USING SPGIST AS
        STORAGE geometry,
        OPERATOR        1       <<  ,
        OPERATOR        5       >>    ,
        OPERATOR        6       ~=    ,
        OPERATOR        8       @     ,
        OPERATOR        10       <<| ,
        OPERATOR        11      |>> ,
        FUNCTION    1       geometry_spgist_config_2d(internal,internal),
        FUNCTION    2       geometry_spgist_choose_2d(internal,internal),
        FUNCTION        5       geometry_spgist_leaf_consistent_2d(internal,geometry,int4),
        FUNCTION        4       geometry_spgist_inner_consistent_2d(internal,geometry,int4),
        FUNCTION        3       geometry_spgist_picksplit_2d(internal,internal);
```

Example

```
postgres=# CREATE TABLE lotsOfPoints AS SELECT ST_MakePoint(x,y) g
from generate_series(1,1000)x,
generate_series(1,1000)y;
SELECT 1000000
postgres=#

postgres=# create index on lotsOfPoints using spgist(point(g));
DEBUG:  ProcessUtility
DEBUG:  building index "lotsofpoints_point_idx" on table "lotsofpoints"
DEBUG:  creating and filling new WAL file
DEBUG:  done creating and filling new WAL file
DEBUG:  creating and filling new WAL file
DEBUG:  done creating and filling new WAL file
DEBUG:  creating and filling new WAL file
DEBUG:  done creating and filling new WAL file
DEBUG:  CommitTransactionCommand
DEBUG:  CommitTransaction
DEBUG:  name: unnamed; blockState:       STARTED; state: INPROGR, xid/subid/cid: 764/1/2, nestlvl: 1, children:
CREATE INDEX
postgres=#

postgres=# SET enable_seqscan to off;
SET
postgres=#

postgres=# EXPLAIN SELECT * FROM lotsOfPoints WHERE g && ST_MakeLine(
postgres(#   ST_MakePoint(0,0),
postgres(#   ST_MakePoint(1,0)
postgres(# );
```

QUERY PLAN

```
-----------------------------------------------------------------------------------------------------------
 Seq Scan on lotsofpoints  (cost=10000000000.00..10000019853.00 rows=1 width=32)
   Filter: (g && '01020000000200000000000000000000000000000000000000000000000000F03F0000000000000000'::geometry)
(2 rows)
postgres=#
```

ERROR

```
postgres=# create index on lotsOfPoints using spgist(g);
The connection to the server was lost. Attempting reset: Failed.
!>
!>
```

Whenever the geometry field is passed directly as a parameter instead of passing it as a point. The current code gives a problem.

This error is coming while using both box2df and geometry datatypes for storage. The problem is arising in the config function geometry_spgist_config_2d where the input argument spgconfigIn is not able to access the geometry.

We need to parse the geometry in some other data type in order to store in the spgist data structures which are different for every function.