8.3.13 Descending Indexes

MySQL supports descending indexes: `DESC` in an index definition is no longer ignored but causes storage of key values in descending order. Previously, indexes could be scanned in reverse order but at a performance penalty. A descending index can be scanned in forward order, which is more efficient. Descending indexes also make it possible for the optimizer to use multiple-column indexes when the most efficient scan order mixes ascending order for some columns and descending order for others.

Consider the following table definition, which contains two columns and four two-column index definitions for the various combinations of ascending and descending indexes on the columns:

```
CREATE TABLE t (
  c1 INT, c2 INT,
  INDEX idx1 (c1 ASC, c2 ASC),
  INDEX idx2 (c1 ASC, c2 DESC),
  INDEX idx3 (c1 DESC, c2 ASC),
  INDEX idx4 (c1 DESC, c2 DESC)
);
```

The table definition results in four distinct indexes. The optimizer can perform a forward index scan for each of the `ORDER BY` clauses and need not use a `filesort` operation:

```
ORDER BY c1 ASC, c2 ASC    -- optimizer can use idx1
ORDER BY c1 DESC, c2 DESC  -- optimizer can use idx4
ORDER BY c1 ASC, c2 DESC   -- optimizer can use idx2
ORDER BY c1 DESC, c2 ASC   -- optimizer can use idx3
```

Use of descending indexes is subject to these conditions:

- Descending indexes are supported only for the `InnoDB` storage engine, with these limitations:
  - Change buffering is not supported for a secondary index if the index contains a descending index key column or if the primary key includes a descending index column.
  - The `InnoDB` SQL parser does not use descending indexes. For `InnoDB` full-text search, this means that the index required on the `FTS_DOC_ID` column of the indexed table cannot be defined as a descending index. For more information, see Section 15.6.2.4, “InnoDB Full-Text Indexes”.
- Descending indexes are supported for all data types for which ascending indexes are available.
- Descending indexes are supported for ordinary (nongenerated) and generated columns (both `VIRTUAL` and `STORED`).
- `DISTINCT` can use any index containing matching columns, including descending key parts.
- Indexes that have descending key parts are not used for `MIN()`/`MAX()` optimization of queries that invoke aggregate functions but do not have a `GROUP BY` clause.
- Descending indexes are supported for `BTREE` but not `HASH` indexes. Descending indexes are not supported for `FULLTEXT` or `SPATIAL` indexes.

  Explicitly specified `ASC` and `DESC` designators for `HASH`, `FULLTEXT`, and `SPATIAL` indexes results in an error.

You can see in the **`Extra`** column of the output of `EXPLAIN` that the optimizer is able to use a descending index, as shown here:

```
mysql> CREATE TABLE t1 (
    -> a INT,
    -> b INT,
    -> INDEX a_desc_b_asc (a DESC, b ASC)
    -> );

mysql> EXPLAIN SELECT * FROM t1 ORDER BY a ASC\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: index
possible_keys: NULL
          key: a_desc_b_asc
      key_len: 10
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Backward index scan; Using index
```

In `EXPLAIN FORMAT=TREE` output, use of a descending index is indicated by the addition of `(reverse)` following the name of the index, like this:

```
mysql> EXPLAIN FORMAT=TREE SELECT * FROM t1 ORDER BY a ASC\G
*************************** 1. row ***************************
EXPLAIN: -> Index scan on t1 using a_desc_b_asc (reverse)  (cost=0.35 rows=1)
```

See also EXPLAIN Extra Information.