- `Using filesort` (JSON property: `using_filesort`)

  MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 8.2.1.16, “ORDER BY Optimization”.

- `Using index` (JSON property: `using_index`)

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

  For `InnoDB` tables that have a user-defined clustered index, that index can be used even when `Using index` is absent from the `Extra` column. This is the case if `type` is `index` and `key` is `PRIMARY`.

  Information about any covering indexes used is shown for `EXPLAIN FORMAT=TRADITIONAL` and `EXPLAIN FORMAT=JSON`. Beginning with MySQL 8.0.27, it is also shown for `EXPLAIN FORMAT=TREE`.

- `Using index condition` (JSON property: `using_index_condition`)

  Tables are read by accessing index tuples and testing them first to determine whether to read full table rows. In this way, index information is used to defer (“push down”) reading full table rows unless it is necessary. See Section 8.2.1.6, “Index Condition Pushdown Optimization”.

- `Using index for group-by` (JSON property: `using_index_for_group_by`)

  Similar to the `Using index` table access method, `Using index for group-by` indicates that MySQL found an index that can be used to retrieve all columns of a `GROUP BY` or `DISTINCT` query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 8.2.1.17, “GROUP BY Optimization”.

- `Using index for skip scan` (JSON property: `using_index_for_skip_scan`)

  Indicates that the Skip Scan access method is used. See Skip Scan Range Access Method.

- `Using join buffer (Block Nested Loop)`, `Using join buffer (Batched Key Access)`, `Using join buffer (hash join)` (JSON property: `using_join_buffer`)

  Tables from earlier joins are read in portions into the join buffer, and then their rows are used from the buffer to perform the join with the current table. `(Block Nested Loop)` indicates use of the Block Nested-Loop algorithm, `(Batched Key Access)` indicates use of the Batched Key Access algorithm, and `(hash join)` indicates use of a hash join. That is, the keys from the table on the preceding line of the `EXPLAIN` output are buffered, and the matching rows are fetched in batches from the table represented by the line in which `Using join buffer` appears.

  In JSON-formatted output, the value of `using_join_buffer` is always one of `Block Nested Loop`, `Batched Key Access`, or `hash join`.

  Hash joins are available beginning with MySQL 8.0.18; the Block Nested-Loop algorithm is not used in MySQL 8.0.20 or later MySQL releases. For more information about these optimizations, see Section 8.2.1.4, “Hash Join Optimization”, and Block Nested-Loop Join Algorithm.

  See Batched Key Access Joins, for information about the Batched Key Access algorithm.

- `Using MRR` (JSON property: `message`)