use a the primary key, add an auto-increment column. Auto-increment column values are unique and are added automatically as new rows are inserted.

- If you do not define a `PRIMARY KEY` for a table, `InnoDB` uses the first `UNIQUE` index with all key columns defined as `NOT NULL` as the clustered index.
- If a table has no `PRIMARY KEY` or suitable `UNIQUE` index, `InnoDB` generates a hidden clustered index named `GEN_CLUST_INDEX` on a synthetic column that contains row ID values. The rows are ordered by the row ID that `InnoDB` assigns. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in order of insertion.

How the Clustered Index Speeds Up Queries

Accessing a row through the clustered index is fast because the index search leads directly to the page that contains the row data. If a table is large, the clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record.

How Secondary Indexes Relate to the Clustered Index

Indexes other than the clustered index are known as secondary indexes. In `InnoDB`, each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. `InnoDB` uses this primary key value to search for the row in the clustered index.

If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

For guidelines to take advantage of `InnoDB` clustered and secondary indexes, see Section 8.3, “Optimization and Indexes”.

15.6.2.2 The Physical Structure of an InnoDB Index

With the exception of spatial indexes, `InnoDB` indexes are B-tree data structures. Spatial indexes use R-trees, which are specialized data structures for indexing multi-dimensional data. Index records are stored in the leaf pages of their B-tree or R-tree data structure. The default size of an index page is 16KB. The page size is determined by the `innodb_page_size` setting when when the MySQL instance is initialized. See Section 15.8.1, “InnoDB Startup Configuration”.

When new records are inserted into an `InnoDB` clustered index, `InnoDB` tries to leave 1/16 of the page free for future insertions and updates of the index records. If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full.

`InnoDB` performs a bulk load when creating or rebuilding B-tree indexes. This method of index creation is known as a sorted index build. The `innodb_fill_factor` variable defines the percentage of space on each B-tree page that is filled during a sorted index build, with the remaining space reserved for future index growth. Sorted index builds are not supported for spatial indexes. For more information, see Section 15.6.2.3, “Sorted Index Builds”. An `innodb_fill_factor` setting of 100 leaves 1/16 of the space in clustered index pages free for future index growth.

If the fill factor of an `InnoDB` index page drops below the `MERGE_THRESHOLD`, which is 50% by default if not specified, `InnoDB` tries to contract the index tree to free the page. The `MERGE_THRESHOLD` setting applies to both B-tree and R-tree indexes. For more information, see Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

15.6.2.3 Sorted Index Builds