MySQL Indexes

OBJECTIVES

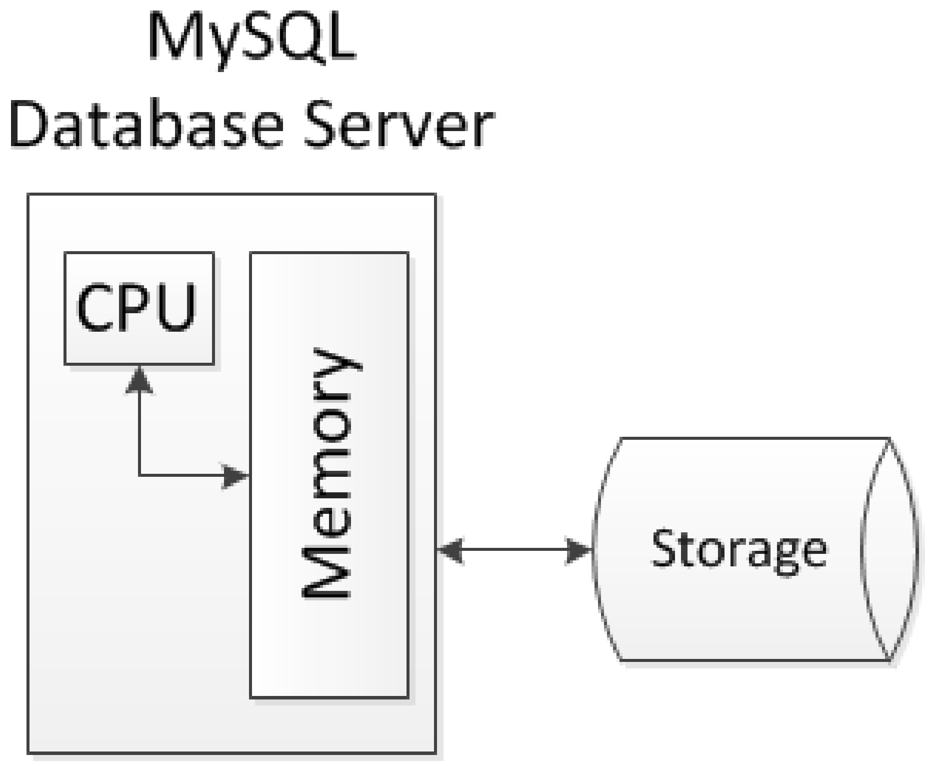
- To understand how indexes work
- Types of MySQL Indexes

MySQL Indexes

The concept of an index

- Think of the index at the back of a book
- Speeds up a search
  - Look up a key word in the index
  - Index points to a page number in the book
- Without the index, you would have to scan every page in the book looking for your key word

Consider database server architecture

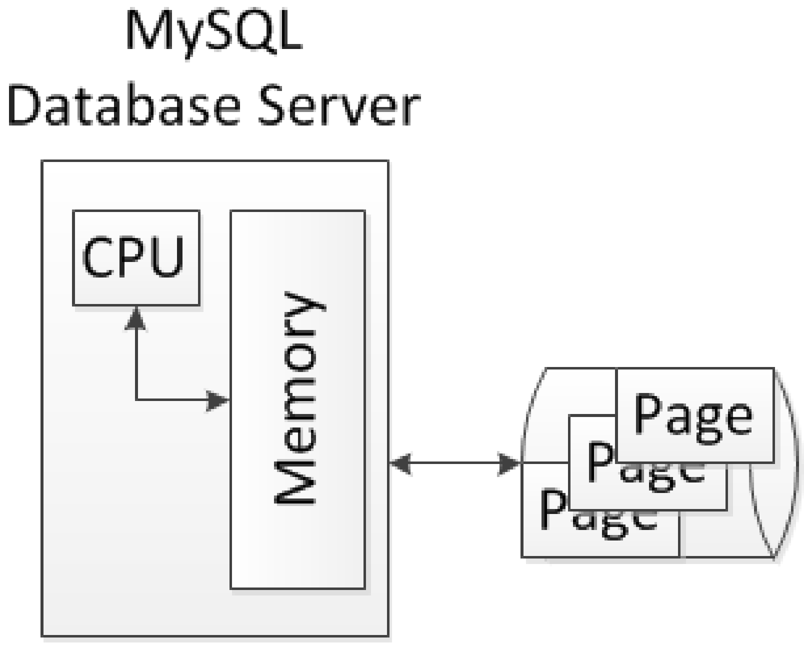


Relative Speeds

nanoseconds
microseconds
milliseconds

MySQL Indexes

Data is moved from disk to memory one page at a time.



MySQL Default page size = 16K

MySQL Indexes

To read an entire table, the table is moved into memory one page at a time.

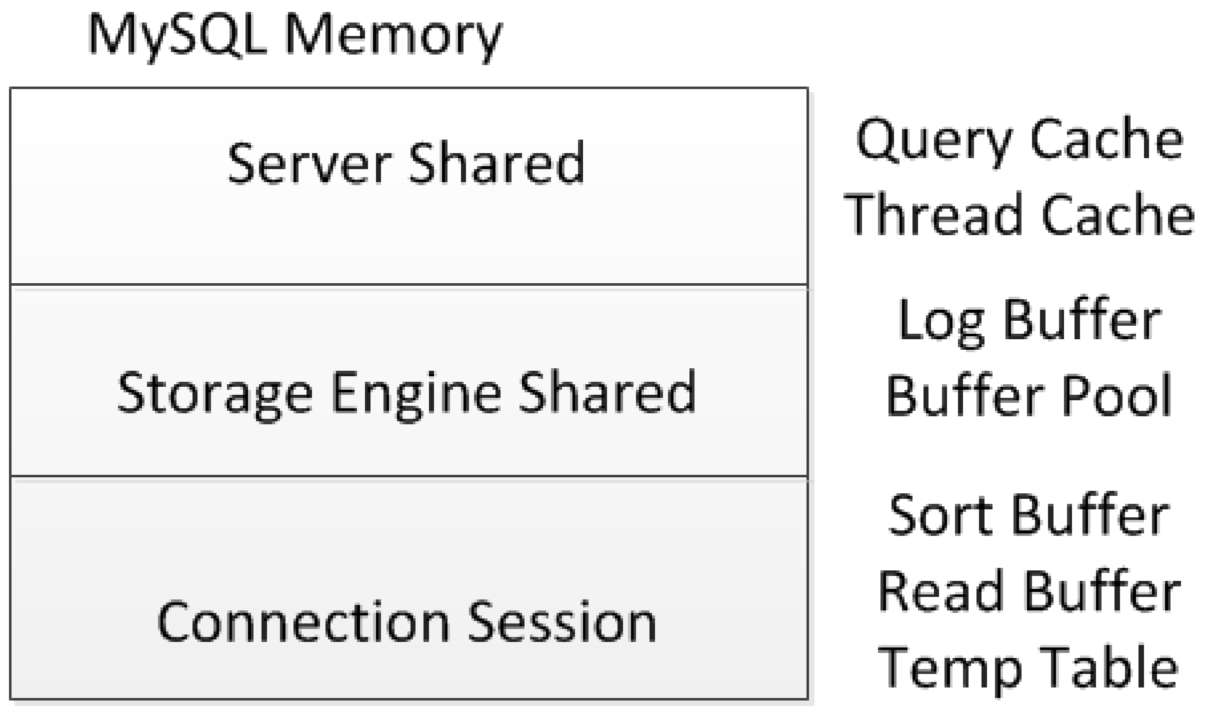
Any select, insert, update, delete requires the affected page where that row resides to be brought into memory (“buffer cache.”)

DML statements modify affected rows on the page in memory. Modified pages are written to disk upon a “commit”.

MySQL Indexes

Moving pages into and out of memory is relatively very time consuming.

MySQL Indexes



MySQL Indexes

MySQL memory allocation is divided into two categories:

- Global (per instance)

Allocated when the instance starts and freed when the instance is shut down. If it fills up, the OS will swap, DB performance will suffer, and eventually the server will crash.

- Session (per connection)

Allocated per session. Released at session end. Used for query results. Read buffer size is per session. 10MB read buffer * 100 concurrent sessions = 1GB of memory.

Query Cache

- Holds the compiled executable query.
- Queries may be run over and over by an application, changing only host variables.
  - Think of a drop-down list created at web page load that reads many rows.
  - Caching the query avoids the time spent validating, parsing, and compiling the query.
- Deprecated in MySQL 5.7
  - Reduces performance predicatability
  - Doesn’t scale with huge workloads on muli-core machines

MySQL Indexes

Thread Cache

- Splits up the execution paths of MySQL into multiple threads, one per user connection, so they run in parallel.

Buffer Pools

- Caching disk I/O
- Can hold objects in memory (smaller tables, indexes)

Session Memory

- Smaller temp tables
- Client-specific memory
  - Table read buffers
  - Sort operations

MySQL Indexes

So why indexes?

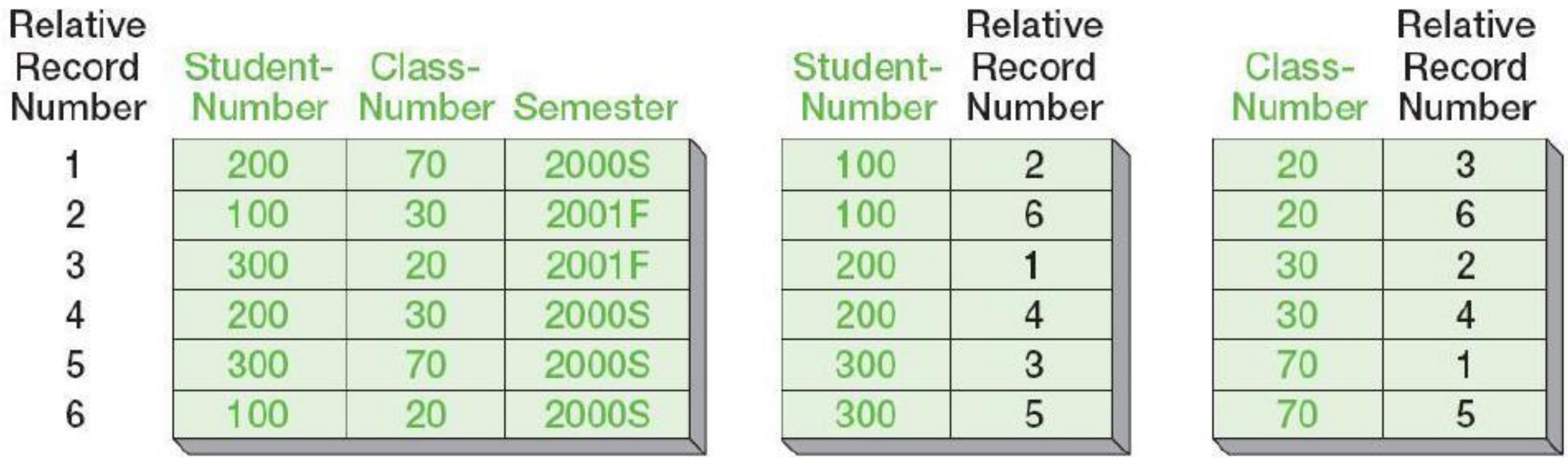
We use indexes to minimize page movement (i.e. Minimize “disk I/O”).

Indexes are usually much smaller than the base tables they serve -- less I/O, less buffer space consumed.

Indexes are often pinned in memory.

MySQL Indexes

Simple index structure.



Relative Record Number	Student-Number	Class-Number	Semester
1	200	70	2000S
2	100	30	2001F
3	300	20	2001F
4	200	30	2000S
5	300	70	2000S
6	100	20	2000S

(a) ENROLLMENT Data

Student-Number	Relative Record Number
100	2
100	6
200	1
200	4
300	3
300	5

(b) Index on StudentNumber

Class-Number	Relative Record Number
20	3
20	6
30	2
30	4
70	1
70	5

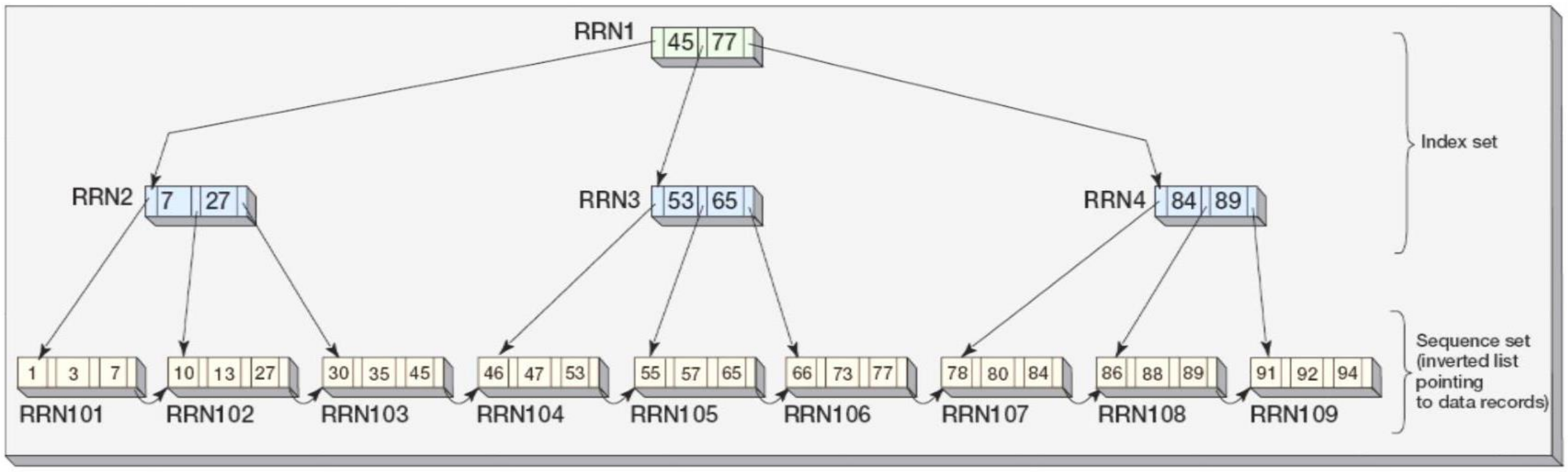
(c) Index on ClassNumber

MySQL Indexes

MySQL indexes are typically B-Tree, the default.

- B-Tree supports both sequential access of rows AND keyed access.
- “Index Set and Sequence Set.”
- “root” and “leaf” nodes.

General Structure of a Simple B-Tree



MySQL Indexes

- The “B” means “balanced”
- B-Tree indexes balance each leaf page node between half-full and full.
- B-Tree indexes balance the levels of leaf pages.
- As pages get full, B-Tree indexes will “split” to create more room for indexes to grow. The index has pre-allocated overflow space.

MySQL Indexes

- As the overflow space gets full, the “split” leaf nodes get spead out.
- Therefore B-Tree indexes occasionally need to be reorganized.
- Drop the index. Recreate it.

MySQL Indexes

- The default DB engine for MySQL is “innodb”.
- A database or table can be created to use the “Memory” DB engine. All objects are memory resident, asynchronously written to disk.
- The Memory engine supports “hash” indexes. The key is run through a hashing algorithm which calculates a shorter key length, which is then associated with a pointer.

MySQL Indexes

- Primary keys are indexed by default.
- Data rows are written to disk in primary key sequence. This is a “clustered” index.
- Allows for binary search.

MySQL Indexes

- Foreign keys may be indexed.
- Other non-key columns may be indexed. (“Secondary” index.)
- The data rows will not be in secondary key sequence.
- MySQL secondary indexes include the primary key values so that access by secondary keys can leverage the clustered index.

MySQL Indexes

- How do we know what secondary indexes are needed?
- Database designers must carefully assess and analyze the full collection of the organization’s queries.
- Secondary indexes are based on query usage patterns.

MySQL Indexes

- Indexes present a trade-off:
  - READs go faster when indexed
  - INSERTs require not only data table updates, but also index updates.
  - For applications that are insert-intensive, indexes can cause significant delays.
  - For small tables, an index can actually hurt performance

MySQL Indexes

- SO –
  - Be very careful about creating indexes
  - Only create a new index when query performance demands it

MySQL Indexes

- **“Ad Hoc”** query access.
  - End-User Reporting tools generate SQL
    - Tableau
    - Business Objects
  - Introduces huge challenges
  - Queries must be monitored for performance
  - Performance bottlenecks must be managed
  - Requires 3rd party tools.
    - WebYog, DataDog for monitoring
    - Queuing - Not in MySQL, but other DBMSs have it

MySQL Indexes

- MySQL allows multi-column indexes.
- MySQL will only use the Nth column in the index if the (N-1)th column is used first.
- If index columns are carefully chosen, a single index can boost performance for many different queries.

MySQL Indexes

- For example:

```
CREATE TABLE test (
     id INT NOT NULL,
     last_name CHAR(30) NOT NULL,
     first_name CHAR(30) NOT NULL,
     PRIMARY KEY (id),
     INDEX IX_Name (last_name,first_name)
);
```

MySQL Indexes

These WILL use `IX_Name (last_name,first_name)`

- `SELECT * FROM test WHERE last_name='Widenius';`
- `SELECT * FROM test WHERE last_name='Widenius' AND first_name='Michael';`
- `SELECT * FROM test WHERE last_name='Widenius' AND (first_name='Michael' OR first_name='Monty');`
- `SELECT * FROM test WHERE last_name='Widenius' AND first_name >='M' AND first_name < 'N';`

MySQL Indexes

- **These WILL NOT use** `IX_Name (last_name,first_name)`

- `SELECT * FROM test WHERE first_name='Michael';`

- `SELECT * FROM test WHERE last_name='Widenius' OR first_name='Michael';`

MySQL Indexes

- Indexes can be included in the TABLE CREATE
- Indexes can be dropped, renamed, added later via a ALTER TABLE

```
ALTER TABLE contacts
    RENAME INDEX contacts_idx
    TO  contacts_new_index;

DROP INDEX contacts_idx
    ON contacts;
```

MySQL Indexes

MySQL keeps statistics on indexes in the “statistics” table in the “information_schema” table. And, innodb keeps table and index stats in the mysql schema.

The query optimizer uses these statistics as it parses a query and builds the execution plan to decide whether or not to use an index.

If the statistics are “stale”, the optimizer might decide to NOT use an important index.

Therefore, statistics need to be refreshed.

MySQL Indexes

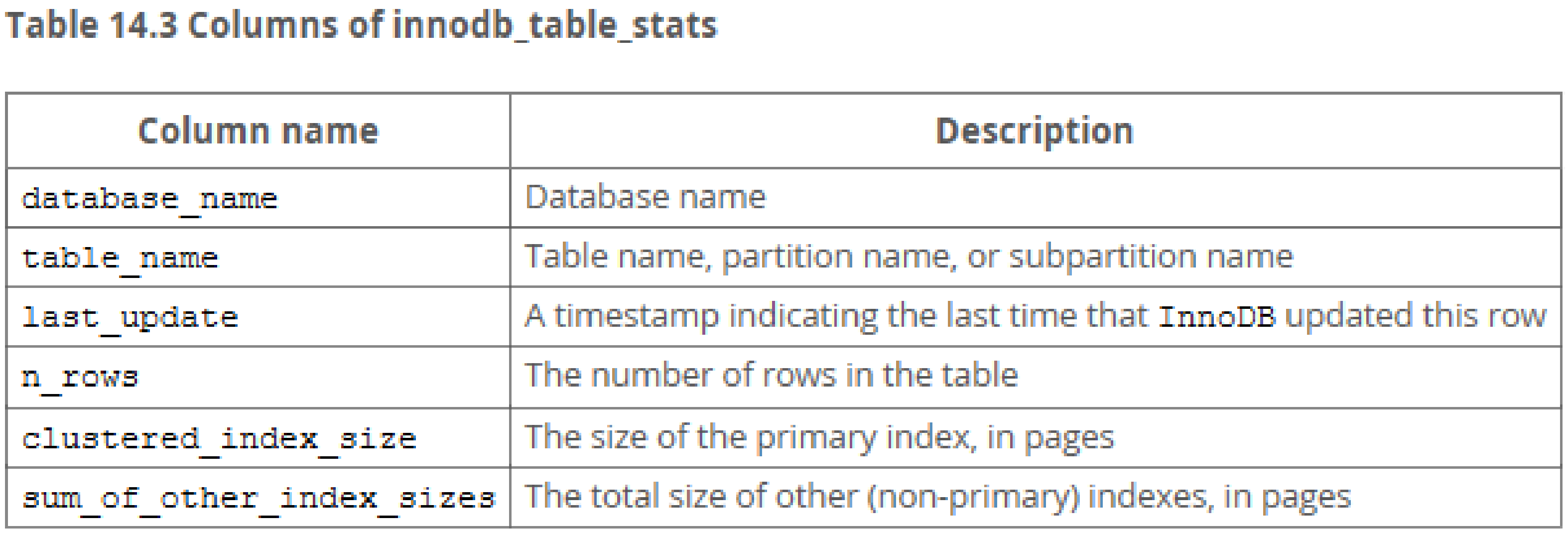


Table 14.3 Columns of innodb_table_stats

Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row
`n_rows`	The number of rows in the table
`clustered_index_size`	The size of the primary index, in pages
`sum_of_other_index_sizes`	The total size of other (non-primary) indexes, in pages

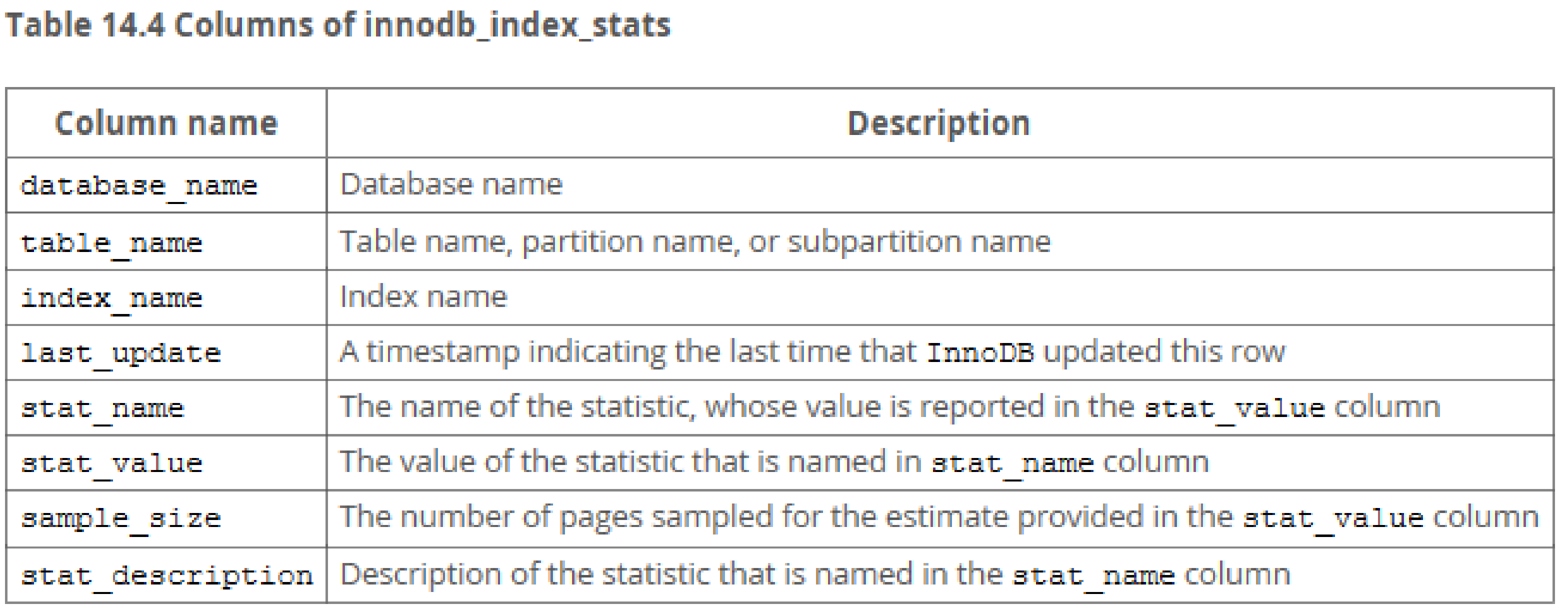


Table 14.4 Columns of innodb_index_stats

Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`index_name`	Index name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row
`stat_name`	The name of the statistic, whose value is reported in the `stat_value` column
`stat_value`	The value of the statistic that is named in `stat_name` column
`sample_size`	The number of pages sampled for the estimate provided in the `stat_value` column
`stat_description`	Description of the statistic that is named in the `stat_name` column

MySQL Indexes

For example, if massive inserts are done that shifts the cardinality (number of unique values in the domain of an indexed column), the optimizer may choose NOT to use the index.

Changes to > 10% of the rows triggers a stats recalc automatically

Solutions:

1. Optimize table – rebuilds the table and alll indexes.
2. Analyze table – forces a stats recalc