



**Instant
ALTER TABLE in
MariaDB 10.3+
Marko Mäkelä
Lead Developer InnoDB**

History of ALTER TABLE in MySQL/MariaDB

- The old way (also known as ALGORITHM=COPY starting with MySQL 5.6)
 - CREATE TABLE ...; INSERT...SELECT; RENAME TABLE ...; DROP TABLE ...;
 - Lots of unnecessary undo and redo logging in InnoDB (“bulk insert” would help)
- “Fast index creation” in InnoDB Plugin for MySQL 5.1 (built-in 5.5 InnoDB)
 - Supports ADD INDEX, ADD UNIQUE INDEX, ADD PRIMARY KEY
- ALGORITHM=INPLACE starting with MySQL 5.6
 - Misleading name; some operations may rebuild the table
 - ADD/DROP COLUMN, ADD PRIMARY KEY, CHANGE...[NOT] NULL
 - Some operations are instant: rename column, change DEFAULT value, ...
 - Should have ALGORITHM=(INSTANT|NOCOPY) to avoid surprises ([MDEV-13134](#))
 - Sloppily called “online” DDL. Online (LOCK=NONE) is sometimes refused:
 - ALTER TABLE...ADD(FULLTEXT|SPATIAL) INDEX, ALGORITHM=INPLACE;
 - Any table rebuild operation when FULLTEXT or SPATIAL indexes are present

Problems with Online Table Rebuild

- MySQL 5.6 includes table-rebuilding `ALTER ((ADD|DROP) COLUMN etc.)`, with `LOCK=NONE`. Why are tools like GH-OST still used?
 - Replication ignores `LOCK=NONE`: Slave will only start after commit→huge lag
 - The `online_log` needs to be buffered (in memory or temporary files).
 - The size depends on the concurrent DML workload; hard to predict!
 - The whole table (including all indexes) will have to be copied.
 - MySQL 5.7 included some performance improvements to this, but huge I/O remains.
- Theoretically, do we really have to rebuild?
 - Only when introducing stricter constraints (shorter columns, add `NOT NULL`).
 - Even that could be done by validating the table and editing metadata.
 - Only `ADD [UNIQUE|PRIMARY|SPATIAL|FULLTEXT] KEY` really require writes.

History of Instant `ADD COLUMN` for InnoDB

- Both Alibaba and Tencent have instant `ADD` in their MySQL 5.6 forks
 - Does not work with old data files; requires a new `ROW_FORMAT`
- MariaDB wants it to work on old (possibly large) files
 - Vin Chen (陈福荣) from Tencent Game DBA Team wrote a prototype that adds an optional record header to identify “afterwards added columns”
 - The `ADD...DEFAULT` values are stored in one place
 - Data dictionary only reflects the latest table definition, including the latest `DEFAULT`
- Marko Mäkelä rewrote the prototype for MariaDB 10.3.2
 - Store a ‘default row’ at the start of the table (we want to remove `SYS_*` tables)
 - Support all but `ROW_FORMAT=COMPRESSED`
 - Crash-safe DDL (a new undo record type); simpler DML rollback; “compression”
 - ensured that online table-rebuild (e.g. `DROP COLUMN`) still works

Basic Usage of Instant ADD COLUMN

- By default, ALTER TABLE...ADD COLUMN is instantaneous
 - Limitation: No hidden FTS_DOC_ID column (for FULLTEXT INDEX) must exist
- Use the FORCE keyword for the old-fashioned ADD COLUMN, with the old-fashioned (additional) limitations:
 - ALGORITHM=INPLACE will not work if multiple FULLTEXT INDEX exist
 - LOCK=NONE will not work if FULLTEXT or SPATIAL INDEX exist
- To monitor the number of avoided table rebuilds:
SELECT variable_value
FROM information_schema.global_status
WHERE variable_name = 'innodb_instant_alter_column';
- See also <https://mariadb.com/resources/blog/instant-add-column-innodb>

Example of Instant ADD COLUMN

```
CREATE TABLE t(id INT PRIMARY KEY, u INT UNIQUE) ENGINE=InnoDB;
INSERT INTO t(id,u) VALUES (1,1), (2,2), (3,3);
ALTER TABLE t ADD COLUMN
(d DATETIME DEFAULT current_timestamp(),
 t TEXT CHARSET utf8 DEFAULT 'The quick brown fox',
 p POINT NOT NULL DEFAULT ST_GeomFromText('POINT(0 0)'));
UPDATE t SET t=NULL WHERE id=3;
```

id	u	d	t	p
1	1	2017-11-10 12:14:00	'The quick brown fox'	POINT(0 0)
2	2	2017-11-10 12:14:00	'The quick brown fox'	POINT(0 0)
3	3	2017-11-10 12:14:00	NULL	POINT(0 0)

Record Changes for Instant `ADD COLUMN`

- An InnoDB table is a collection of indexes:
 - Clustered index (ordered by `PRIMARY KEY` or similar); index-organized table
 - Optional secondary indexes, pointing to clustered index keys
- We primarily need to concentrate on the clustered index leaf page records
 - (`PRIMARY KEY`, `DB_TRX_ID`, `DB_ROLL_PTR`, non-virtual columns)
- For now, we only allow `ADD COLUMN` of the last column(s). We will get:
 - (`PRIMARY KEY`, `DB_TRX_ID`, `DB_ROLL_PTR`, non-virtual columns, **added columns**)
- How to tell if added columns are present?
 - `ROW_FORMAT=REDUNDANT` explicitly stores the number of index fields.
 - `ROW_FORMAT=COMPACT`, `ROW_FORMAT=DYNAMIC` will require bigger changes:
 - Record header flag and optional field for “number of added columns”.
 - Must store the original number of fields or columns somewhere.

Page Changes for Instant ADD COLUMN

- Clustered index root page changes:
 - `FIL_PAGE_TYPE_INSTANT` indicates that instant operation was used
 - `PAGE_INSTANT` stores the original (smaller) number of clustered index fields
- Change the leftmost clustered index leaf page:
 - After the infimum, store a “default” record with `REC_INFO_MIN_REC_FLAG`:
 - Must have the optional “added fields” header
 - The number of fields must match the current table definition
 - Values of “added fields” are the values of “missing fields”
- Clustered index contents from the previous example:
 - (**default**,id,u,d=2017-11-10 12:14:00,t='The quick brown fox',p=POINT(o o)),
 - (1,1), (2,2), (3,3,2017-11-10 12:14:00, **NULL**)
 - We omit trailing fields that are equal to the fields in the “default” record.

MariaDB 10.4: ADD...(FIRST | AFTER), DROP...

- Keep the user record format unchanged.
 - Physically, keep doing `ADD COLUMN` last in the clustered index records
 - `DROP COLUMN` will leave garbage in the records.
 - Changing column order physically becomes a no-op.
 - `ADD COLUMN` will be possible even if hidden `FTS_DOC_ID` exists
- In the “default” record, store a mapping of table columns to index fields
 - Pass the mapping to `row_build()`
 - A new function `row_build_clust_index_entry()` will take the mapping
 - The old `row_build_index_entry()` will remain for secondary indexes

MariaDB 10.4+: Instant CHANGE COLUMN

- MySQL 5.7/MariaDB 10.2: Extend VARCHAR maximum size
 - Only if the physical format allows; not VARCHAR(255) to VARCHAR(256)
- We need something in the user data records to indicate physical format
 - “Format version number” that points to something in the “default” record?
- Format changes can only be instantaneous if they relax constraints:
 - Example: Changing CHAR(1) to CHAR(2), INT to BIGINT or NOT NULL to NULL
 - Less likely: Changing POINT to GEOMETRY, changing latin1 to utf8
- Failure is an option, if we perform table scan to validate the data:
 - Example: Changing BIGINT NULL to INT UNSIGNED NOT NULL
- Affected secondary indexes must be rebuilt if the physical format changes
 - Still much faster than rebuilding the entire table; can be done online



Thank you

