MySQL 8.0 What's new in Data Dictionary and DDL

Bin Su
Oracle, MySQL
April 2018

Safe Harbor Statement

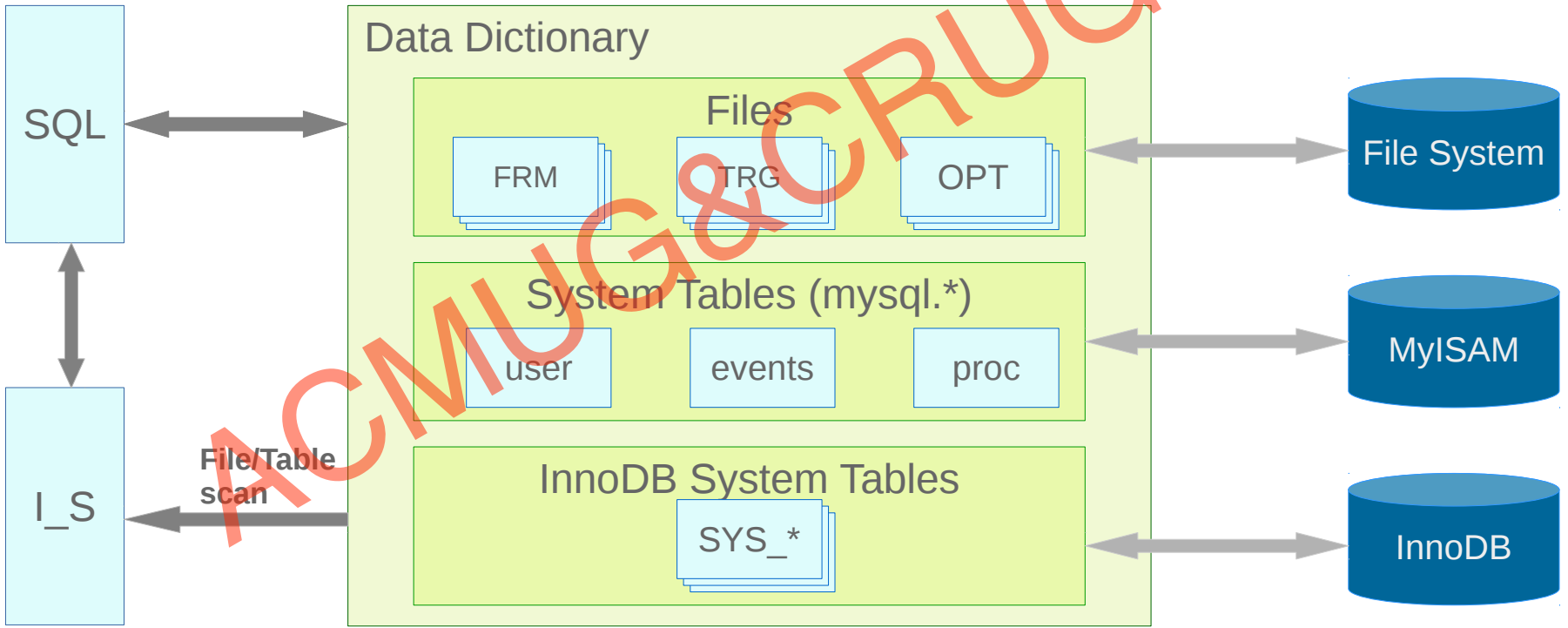
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

1. What is new in Data Dictionary
2. What is new in DDL
3. Summary

What is new in Data Dictionary

Traditional Data Dictionary



New Data Dictionary

Data Dictionary Storage Engine

- **InnoDB plays an important role**
  - MySQL Data Dictionary Storage Engine
  - Single set of persisted metadata for all storage engines
  - Control meta-data access using single locking mechanism
  - Improves table spaces by removing .frm files
  - Optimizations for DD table access
  - All DD tables are put into a dedicated DD tablespace(mysql.ibd)
  - Atomic DDL → Transactional DDL

InnoDB with new DD

- **InnoDB initializes internal tables from DD**
- **InnoDB stores SE specific metadata to DD**

se_private_data (in several tables)	se_private_id	row_format, options etc.
auto_increment, version Index root page, trx_id, Server version ...	Internal table id, etc.	Row format, key_block_size, etc.

Version

- **Server version, such as 8.0.5**
  - Stored in DD table and page 0 of tablespace, at byte 8
  - To identify release with new features
  - Visible in information_schema.innodb_tablespaces
- **Tablespace version, starting from 1...**
  - Stored in DD table and page 0 of tablespace, at byte 12
  - To identify any space format changes
  - Also visible in I_S.innodb_tablespaces

SDI

- **SDI(Serialized Dictionary Information)**
  - Metadata stored in addition to the DD itself
  - To make the tablespace self descriptive
- **The SDI is stored in tablespace**
  - Stored in the form of B-tree
  - Compressed JSON format
- **ibd2sdi to extract SDI from tablespaces**
- **IMPORT/EXPORT**

What is new in DDL

Atomic DDL

- **Prerequisites**
  - Both atomic DD update and data file update
  - Storing DD metadata in transactional SE
  - Writing necessary SE DDL logs
  - Single DD transaction to update for DDL

Atomic DDL - DDL log table

```
innodb_ddl_log    CREATE TABLE `innodb_ddl_log` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `thread_id` bigint(20) unsigned NOT NULL,
  `type` int(10) unsigned NOT NULL,
  `space_id` int(10) unsigned DEFAULT NULL,
  `page_no` int(10) unsigned DEFAULT NULL,
  `index_id` bigint(20) unsigned DEFAULT NULL,
  `table_id` bigint(20) unsigned DEFAULT NULL,
  `old_file_path` varchar(512) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `new_file_path` varchar(512) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `thread_id` (`thread_id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 STATS_PERSISTENT=0
```

- One of the DD tables resides in DD tablespace
- No row locking
- One DDL will generate several logs
- Changes are persisted immediately, exempted from innodb_flush_log_at_trx_commit
- Table size won’t grow infinitely

CREATE TABLE

- **Concurrent CREATE TABLE regression**
  - https://bugs.mysql.com/bug.php?id=87827
  - Due to Atomic DDL implementation
  - DD mutex/lock contention is too hot
- **With new DD, it's fixed by removing the DD mutex/lock - 33-40% saving**

Branch	Time for 20 * 100 loop (200K tables)	Time for 100 * 100 loop (1M tables)
5.7	real 4m30.542s	real 22m52.473s
8.0.1	real 4m59.040s	real 25m32.688s
8.0.5	real 2m48.158s	real 16m56.874s

ALTER TABLE - background

- **ALTER TABLE tbl_name [alter_specification] ...**
  - ALGORITHM [=] { **DEFAULT** | INPLACE | COPY }
  - LOCK [=] { **DEFAULT** | **NONE** | SHARED | EXCLUSIVE }
- **INPLACE**
  - Not always true INPLACE, looks like ONLINE only
  - Three phases
  - Table has to be locked for a period of time
- **COPY**
  - Copy the whole table with table lock

ALTER TABLE - INSTANT

- **ALTER TABLE tbl_name … ALGORITHM = INSTANT;**
  - New (default) algorithm
  - Does not work with LOCK clause
  - Exception: ALGORITHM=INSTANT, LOCK=DEFAULT
- **Internal**
  - This would be basically metadata change only
  - No table lock required
  - INPLACE equals INSTANT in some scenarios

ALTER TABLE - INSTANT

- **Operations which can be INSTANT**
  - RENAME TABLE(ALTER)
  - SET DEFAULT
  - DROP DEFAULT
  - MODIFY COLUMN
  - CHANGE COLUMN(Virtual column generation expression)
  - Change index option
  - ...

ALTER TABLE - INSTANT

- **Operations which can be INSTANT**
  - RENAME TABLE(ALTER)
  - SET DEFAULT
  - DROP DEFAULT
  - MODIFY COLUMN
  - CHANGE COLUMN(Virtual column generation expression)
  - Change index option
  - ADD virtual column, DROP virtual column
  - ADD COLUMN(non-generated)

ALTER TABLE – ADD COLUMN

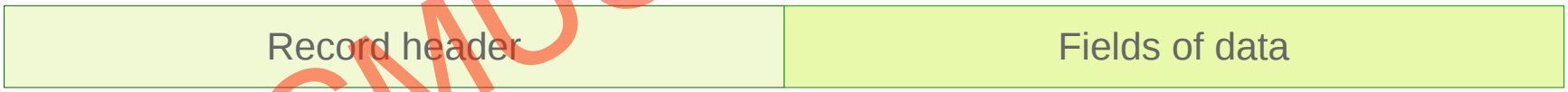
- **It could be the most pain point for users**
  - New columns have to be added (to a big table) from time to time
  - Copy table - time, disk, resource schedule
  - Table lock
  - Replication
  - …
- **But why?**
  - InnoDB doesn’t keep enough metadata in physical record/page

ALTER TABLE – INSTANT ADD COLUMN

- **Contribution from Tencent**
  - Only metadata change
  - No copy data any more
  - No double (or even more) disk space
  - Smaller final data size
  - Forward compatibility with old data file
- **ALTER TABLE … ADD COLUMN c, ALGORITHM = INSTANT**
- **Can be INSTANT along with other instant operations**
- **Support DYNAMIC/COMPACT/REDUNDANT**

INSTANT ADD COLUMN – Phys record

- **New style physical record, ROW_FORMAT=DYNAMIC/COMPACT**
  - Clustered index
  - Leaf pages

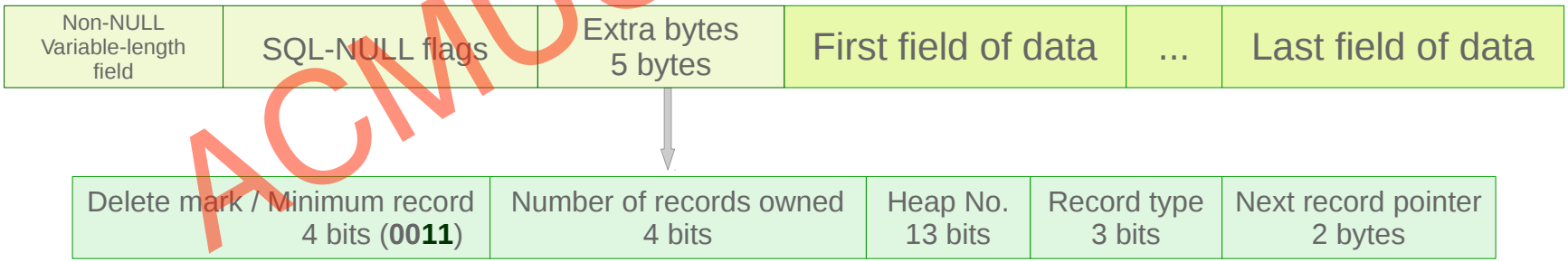


INSTANT ADD COLUMN – Phys record

- **New style physical record, ROW_FORMAT=DYNAMIC/COMPACT**
  - Clustered index
  - Leaf pages

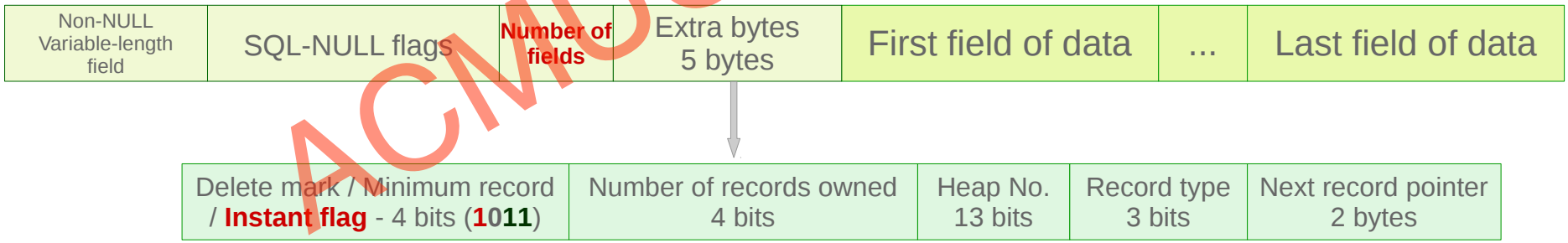
INSTANT ADD COLUMN – Phys record

- **New style physical record, ROW_FORMAT=DYNAMIC/COMPACT**
  - Clustered index
  - Leaf pages



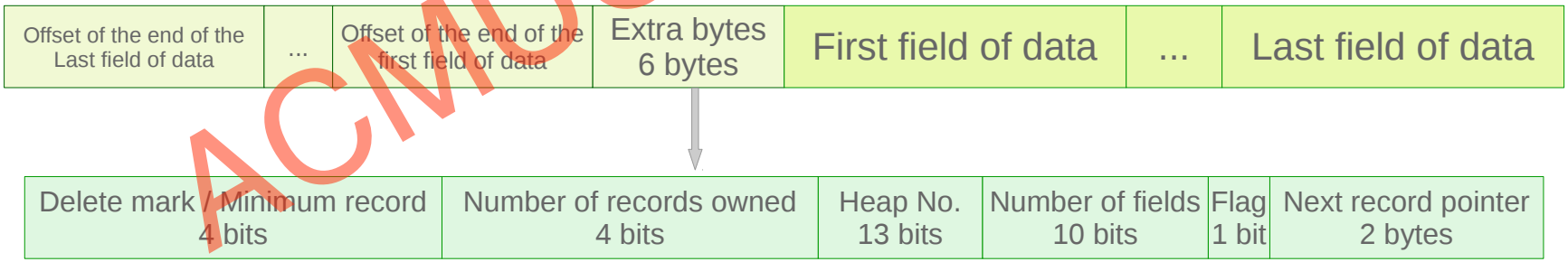
INSTANT ADD COLUMN – Phys record

- **New style physical record, ROW_FORMAT=DYNAMIC/COMPACT**
  - Clustered index
  - Leaf pages



INSTANT ADD COLUMN – Phys record

- **Old style physical record, ROW_FORMAT=REDUNDANT**

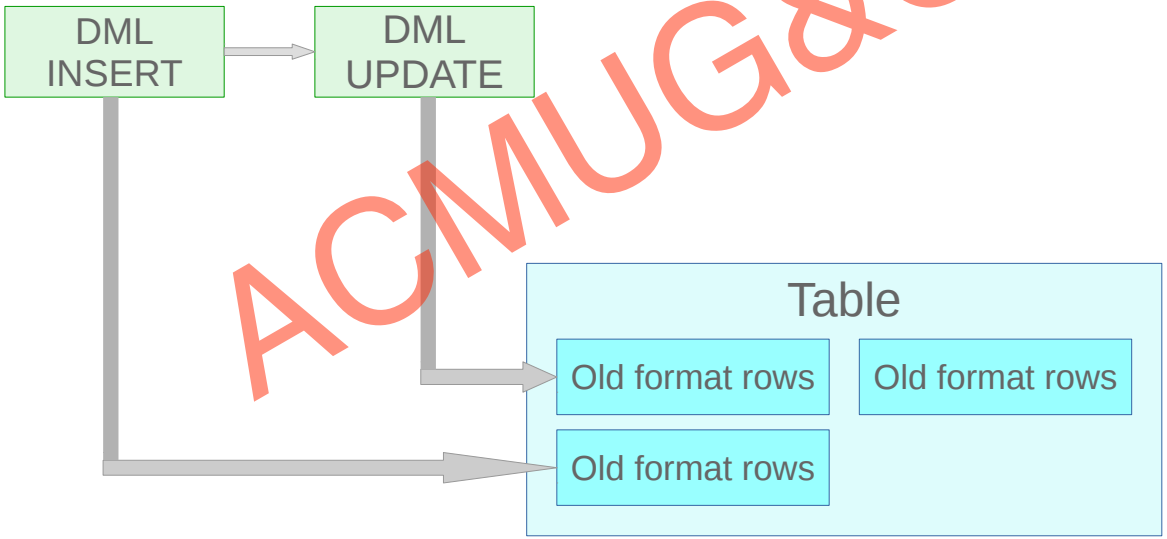


INSTANT ADD COLUMN – Phys record

- **Record created before first INSTANT ADD COLUMN**
  - In old format
  - Number of fields = Instant columns
- **Record created after last INSTANT ADD COLUMN**
  - In new format
  - Number of fields == Latest number of fields
- **Record created between above two**
  - In new format
  - Instant columns < Number of fields < Latest number of fields

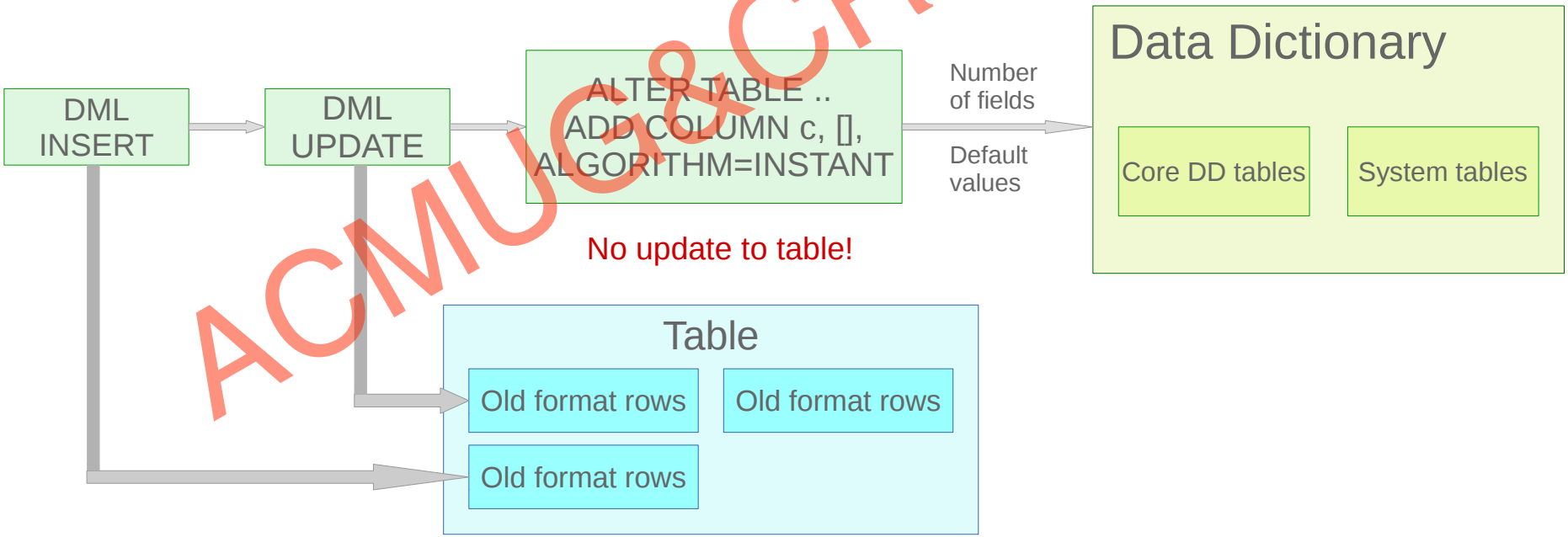
INSTANT ADD COLUMN - Implementation

- **The key is the first INSTANT ADD COLUMN**



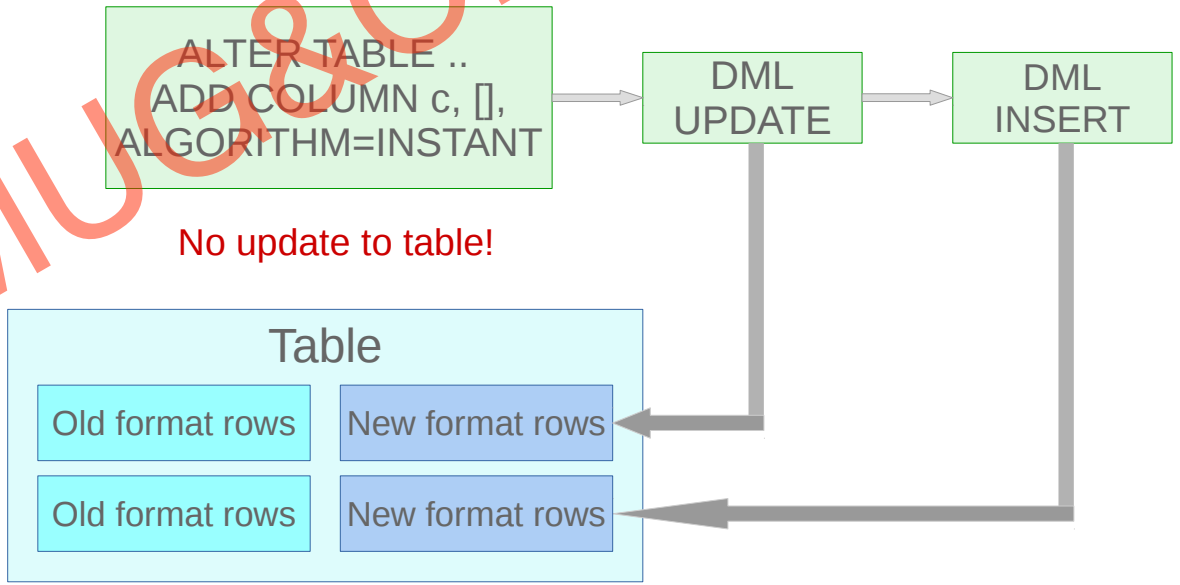
INSTANT ADD COLUMN - Implementation

- **The key is the first INSTANT ADD COLUMN**



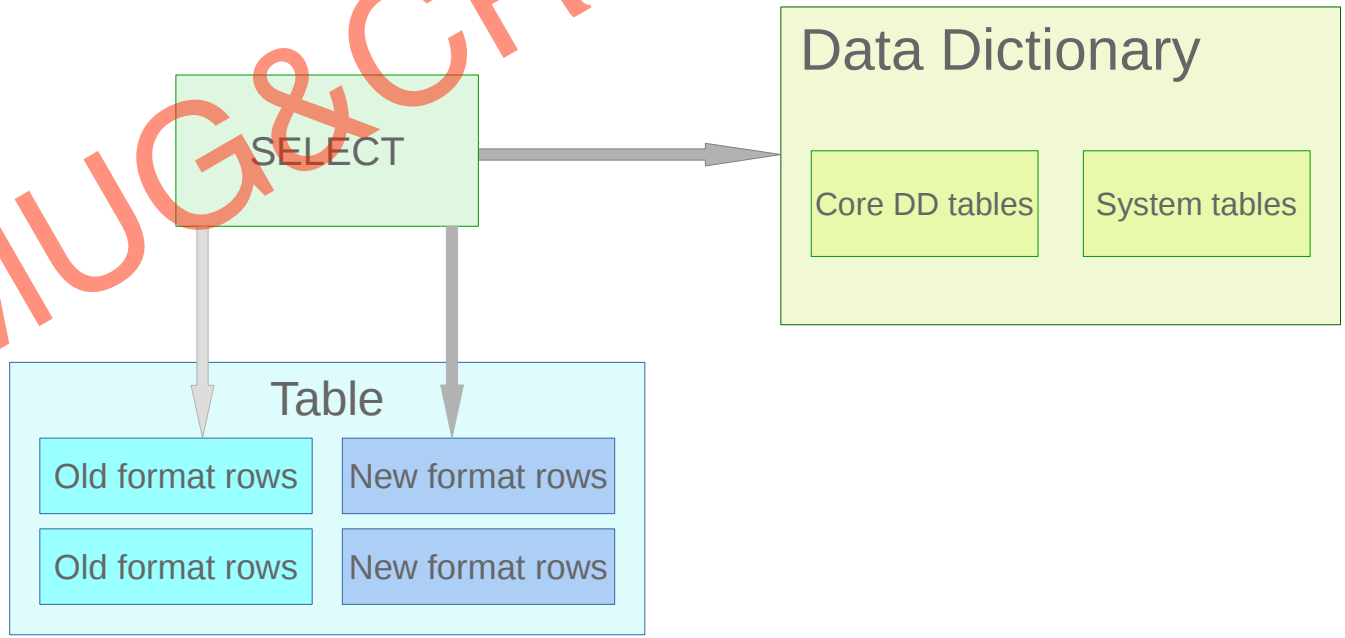
INSTANT ADD COLUMN - Implementation

- **The key is the first INSTANT ADD COLUMN**



INSTANT ADD COLUMN - Implementation

- **The key is the first INSTANT ADD COLUMN**



INSTANT ADD COLUMN – Metadata

- **First INSTANT ADD COLUMN**
  - Remember current number of fields
  - Remember the default value of new columns, either value or NULL
- **Follow-up INSTANT ADD COLUMN**
  - Only remember the default value of new columns

INSTANT ADD COLUMN – Metadata

- **Why store the DEFAULT?**

```
mysql> CREATE TABLE t1(a INT);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO t1 VALUES(0), (1);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE t1 ADD COLUMN b INT DEFAULT 20;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1;
+------+------+
| a    | b    |
+------+------+
|    0 |   20 |
|    1 |   20 |
+------+------+
2 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE t1 ALTER COLUMN b SET DEFAULT 30;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> INSERT INTO t1 VALUES(2, default);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t1;
+------+------+
| a    | b    |
+------+------+
|    0 |   20 |
|    1 |   20 |
|    2 |   30 |
+------+------+
3 rows in set (0.00 sec)

mysql> DROP TABLE t1;
Query OK, 0 rows affected (0.03 sec)
```

INSTANT ADD COLUMN – Metadata

- **Table rebuild, create, truncate, etc.**
  - Will discard the relevant metadata, and keep the table/partitions as before
  - Column default values would be abandoned if useless
- **Partitioned table**
  - Some partition operations will only re-create / truncate some of partitions
  - Some partitions will need the default values, some not

INSTANT ADD COLUMN – Observability

```
mysql> CREATE TABLE t1 (a INT, b INT);
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT table_id, name, instant_cols FROM information_schema.innodb_tables WHERE name LIKE '%t1%';
+----------+---------+--------------+
| table_id | name    | instant_cols |
+----------+---------+--------------+
|     1062 | test/t1 |            0 |
+----------+---------+--------------+
1 row in set (0.03 sec)

mysql> SELECT table_id, name, has_default, default_value FROM information_schema.innodb_columns WHERE table_id = 1062;
+----------+------+-------------+---------------+
| table_id | name | has_default | default_value |
+----------+------+-------------+---------------+
|     1062 | a    |           0 | NULL          |
|     1062 | b    |           0 | NULL          |
+----------+------+-------------+---------------+
2 rows in set (0.34 sec)

mysql> ALTER TABLE t1 ADD COLUMN c INT, ADD COLUMN d INT DEFAULT 1000;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT table_id, name, instant_cols FROM information_schema.innodb_tables WHERE name LIKE '%t1%';
+----------+---------+--------------+
| table_id | name    | instant_cols |
+----------+---------+--------------+
|     1062 | test/t1 |            2 |
+----------+---------+--------------+
1 row in set (0.01 sec)

mysql> SELECT table_id, name, has_default, default_value FROM information_schema.innodb_columns WHERE table_id = 1062;
+----------+------+-------------+---------------+
| table_id | name | has_default | default_value |
+----------+------+-------------+---------------+
|     1062 | a    |           0 | NULL          |
|     1062 | b    |           0 | NULL          |
|     1062 | c    |           1 | NULL          |
|     1062 | d    |           1 | 800003e8      |
+----------+------+-------------+---------------+
4 rows in set (0.36 sec)
```

INSTANT ADD COLUMN – Observability

```
mysql> ALTER TABLE t1 ADD COLUMN e VARCHAR(100) DEFAULT 'Hello world!';
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT table_id, name, instant_cols FROM information_schema.innodb_tables WHERE name LIKE '%t1%';
+----------+---------+--------------+
| table_id | name    | instant_cols |
+----------+---------+--------------+
|     1062 | test/t1 |            2 |
+----------+---------+--------------+
1 row in set (0.03 sec)

mysql> SELECT table_id, name, has_default, default_value FROM information_schema.innodb_columns WHERE table_id = 1062;
+----------+------+-------------+---------------------------+
| table_id | name | has_default | default_value             |
+----------+------+-------------+---------------------------+
|     1062 | a    |           0 | NULL                      |
|     1062 | b    |           0 | NULL                      |
|     1062 | c    |           1 | NULL                      |
|     1062 | d    |           1 | 800003e8                  |
|     1062 | e    |           1 | 48656c6c6f20776f726c6421  |
+----------+------+-------------+---------------------------+
5 rows in set (0.11 sec)
```

INSTANT ADD COLUMN – Row size too large

- **An interesting rollback problem**

```
CREATE TABLE t1(id INT PRIMARY KEY, c1 VARCHAR(4000), c2 VARCHAR(4000), c3
VARCHAR(1000));

INSERT INTO t1 VALUES(1, repeat('a', 4000), repeat('b', 4000), repeat('c', 1));

ALTER TABLE t1 ADD COLUMN c4 VARCHAR(500) NOT NULL DEFAULT repeat('d', 500);

START TRANSACTION;
UPDATE t1 SET c1 = repeat('x', 200) WHERE id = 1;
ROLLBACK;

START TRANSACTION;
UPDATE t1 SET c4 = 'x' WHERE id = 1;
ROLLBACK;
```

INSTANT ADD COLUMN – Others

- **Redo**
  - Number of fields(before first instant ADD COLUMN) should be remembered
- **EXPORT/IMPORT**
  - Both number of fields and default values should be remembered in .cfg
- **Side effects**
  - Could not fix corrupted table/index, etc.
  - Postpone the row size checking

INSTANT ADD COLUMN - Limitations

- **Only support adding columns at last**
- **Not support COMPRESSED, which is seldom used**
- **Not support a table which already has any fulltext index**
- **Not support any table residing in DD tablespace**
- **Not support temporary table(it goes with COPY)**

ALTER TABLE … PARTITION

- **InnoDB supports 'ALTER … PARTITION' natively**
  - ADD / DROP / COALESCE / REORGANIZE / REBUILD / EXCHANGE PARTITION
  - 'ALGORITHM = …, LOCK = …' is also supported now
  - Less logs would be written, so better performance
  - It paves the way for future improvement

Summary

Upgrade steps

- **Upgrade from 5.7 only**
  - Upgrade automatically
  - Make sure no crash and previous innodb_fast_shutdown is not 2
  - Create new DD tables in DD tablespace
  - Update all tables to new DD tables
  - Handle Undo tablespaces
  - Create SDI
  - Finally, InnoDB system tables get dropped
- **Downgrade is not allowed for now**
- **Incompatibility and crash can be handled**

Summary

- **Simplified and unified Data Dictionary**
- **Atomic and crash-safe DDL**
- **Better DDL performance**
- **INSTANT ALTER TABLE, especially ADD COLUMN**
- **Better I_S queries performance**
- **Self descriptive tablespace**
- **Easy upgrade**