



PERCONA  
Performance Consulting Experts

# Recovery of lost or corrupted InnoDB tables

MySQL User Conference 2010, Santa Clara

Aleksandr.Kuzminsky@Percona.com

Percona Inc.

<http://MySQLPerformanceBlog.com>

# Agenda

*Three things are certain:  
Death, taxes and lost data.  
Guess which has occurred?*

1. InnoDB format overview
2. Internal system tables `SYS_INDEXES` and `SYS_TABLES`
3. InnoDB Primary and Secondary keys
4. Typical failure scenarios
5. InnoDB recovery tool

# 1. InnoDB format overview

# How MySQL stores data in InnoDB

## 1. A table space (ibdata1)

- System tablespace(data dictionary, undo, insert buffer, etc.)
- PRIMARY indices (PK + data)
- SECONDARY indices (SK + PK) If the key is (f1, f2) it is stored as (f1, f2, PK)

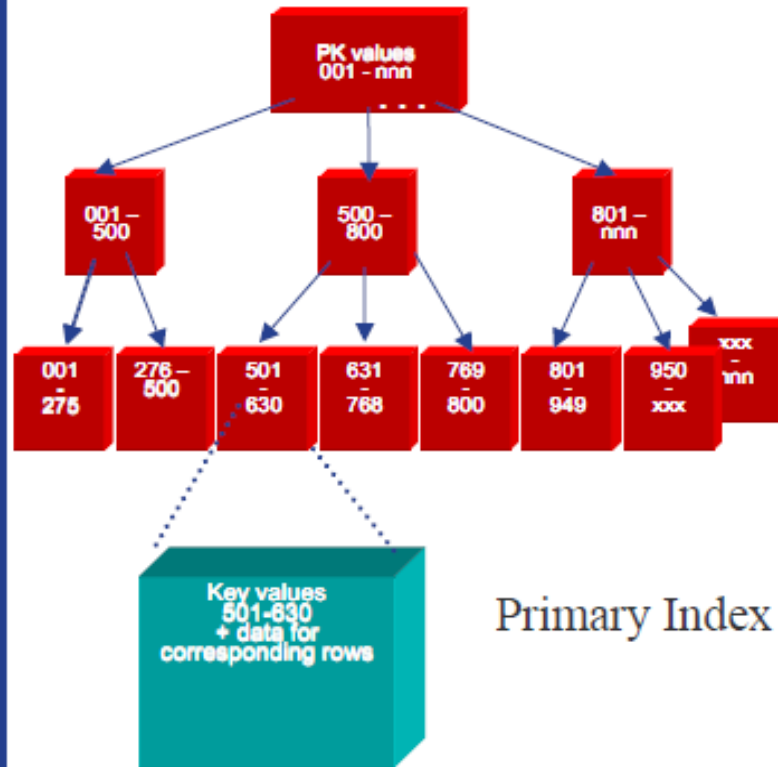
## 1. file per table (.ibd)

- PRIMARY index
- SECONDARY indices

## 1. InnoDB pages size 16k (uncompressed)

## 2. Every index is identified by **index\_id**

# InnoDB Indexes - Primary



- Data rows are stored in the B-tree leaf nodes of a clustered index

- B-tree is organized by primary key or non-null unique key of table, if defined; else, an internal column with 6-byte ROW\_ID is added.

# How MySQL stores data in InnoDB

## Page identifier index\_id

```
mysql> CREATE TABLE innodb_table_monitor(x int)
engine=innodb
```

### Error log:

TABLE: name test/site\_folders, id 0 119, columns 9, indexes 1, appr.rows 1  
COLUMNS: id: DATA\_INT len 4 prec 0; name: type 12 len 765 prec 0; sites\_count: DATA\_INT len 4 prec 0;  
created\_at: DATA\_INT len 8 prec 0; updated\_at: DATA\_INT len 8 prec 0;  
DB\_ROW\_ID: DATA\_SYS prtype 256 len 6 prec 0; DB\_TRX\_ID: DATA\_SYS prtype 257 len 6 prec 0;  
DB\_ROLL\_PTR: DATA\_SYS prtype 258 len 7 prec 0;  
INDEX: name PRIMARY, id **0 254**, fields 1/7, type 3  
root page 271, appr.key vals 1, leaf pages 1, size pages 1  
FIELDS: id DB\_TRX\_ID DB\_ROLL\_PTR name sites\_count created\_at updated\_at

# InnoDB page format

FIL HEADER
PAGE_HEADER
INFINUM+SUPREMUM RECORDS
USER RECORDS
FREE SPACE
Page Directory
Fil Trailer

# InnoDB page format

## Fil Header

Name	Size	Remarks
<b>FIL_PAGE_SPACE</b>	4	4 ID of the space the page is in
<b>FIL_PAGE_OFFSET</b>	4	ordinal page number from start of space
<b>FIL_PAGE_PREV</b>	4	offset of previous page in key order
<b>FIL_PAGE_NEXT</b>	4	offset of next page in key order
<b>FIL_PAGE_LSN</b>	8	log serial number of page's latest log record
<b>FIL_PAGE_TYPE</b>	2	current page type FIL_PAGE_INODE == 0x03
<b>FIL_PAGE_FILE_FLUSH_LSN</b>	8	"the file has been flushed to disk at least up to this lsn" (log serial number), valid only on the first page of the file
<b>FIL_PAGE_ARCH_LOG_NO</b>	4	the latest archived log file number at the time that FIL_PAGE_FILE_FLUSH_LSN was written (in the log)

# InnoDB page format

## Page Header

Name	Size	Remarks
PAGE_N_DIR_SLOTS	2	number of directory slots in the Page Directory part; initial value = 2
PAGE_HEAP_TOP	2	record pointer to the first record in the heap
PAGE_N_HEAP	2	number of records in the heap
PAGE_FREE	2	record pointer to the first free record
PAGE_GARBAGE	2	"number of bytes in deleted records"
PAGE_LAST_INSERT	2	record pointer to the last inserted record
PAGE_DIRECTION	2	either PAGE_LEFT, PAGE_RIGHT, or PAGE_NO_DIRECTION
PAGE_N_DIRECTION	2	number of consecutive inserts in the same direction, e.g. "last 5 were all to the left"
PAGE_N_RECS	2	number of user records
PAGE_MAX_TRX_ID	8	the highest ID of a transaction which might have changed a record on the page (only set for secondary indexes)
PAGE_LEVEL	2	level within the B-tree (0 for a leaf page)
PAGE_INDEX_ID	8	index ID
PAGE_BTR_SEG_LEAF	10	"file segment header for the leaf pages in a B-tree" (this is irrelevant here)
PAGE_BTR_SEG_TOP	10	"file segment header for the non-leaf pages in a B-tree" (this is irrelevant here)

Highest bit is row format(1 - COMPACT, 0 - REDUNDANT )

index\_id

# InnoDB page format (REDUNDANT)

## Extra bytes

<i>Name</i>	<i>Size</i>	<i>Description</i>
<i>record_status</i>	2 bits	_ORDINARY, _NODE_PTR, _INFIMUM, _SUPREMUM
<i>deleted_flag</i>	1 bit	1 if record is deleted
<i>min_rec_flag</i>	1 bit	1 if record is predefined minimum record
<i>n_owned</i>	4 bits	number of records owned by this record
<i>heap_no</i>	13 bits	record's order number in heap of index page
<i>n_fields</i>	10 bits	number of fields in this record, 1 to 1023
<i>lbyte_offs_flag</i>	1 bit	1 if each Field Start Offsets is 1 byte long (this item is also called the "short" flag)
<i>next 16 bits</i>	16 bits	pointer to next record in page

# InnoDB page format (COMPACT)

## Extra bytes

<i>Name</i>	<i>Size, bits</i>	<i>Description</i>
<i>record_status</i> <i>deleted_flag</i> <i>min_rec_flag</i>	4	4 bits used to delete mark a record, and mark a predefined minimum record in alphabetical order
<i>n_owned</i>	4	the number of records owned by this record (this term is explained in page0page.h)
<i>heap_no</i>	13	the order number of this record in the heap of the index page
<i>record type</i>	3	000=conventional, 001=node pointer (inside B-tree), 010=infimum, 011=supremum, 1xx=reserved
<i>next 16 bits</i>	16	a relative pointer to the next record in the page

# How to check row format?

- The highest bit of the *PAGE\_N\_HEAP* from the page header
- 0 stands for version REDUNDANT, 1 - for COMACT
- ```
dc -e "2o `hexdump -C d pagefile | grep 00000020 | awk '{ print $12}' ` p" | sed 's/./& /g' | awk '{ print $1}'
```

# Rows in an InnoDB page

- Rows in a single pages is a linked list
- The first record INFIMUM
- The last record SUPREMUM
- Sorted by Primary key

|      |     |         |
|------|-----|---------|
| next |     | infimum |
| 0    |     | supremu |
| next | 100 | data... |
| next | 101 | data... |
| next | 103 | data... |
| next | 102 | data... |
|      |     |         |

# Records are saved in insert order

```
insert into t1 values(10, 'aaa');
insert into t1 values(30, 'ccc');
insert into t1 values(20, 'bbb');
```

```
JG.....N<E.....
.....2..
...infimum.....supremum.....6.
.....).....2..aaa.....
...*.....2..ccc.....+
...2..bbb.....
.....
```

# Row format

## EXAMPLE:

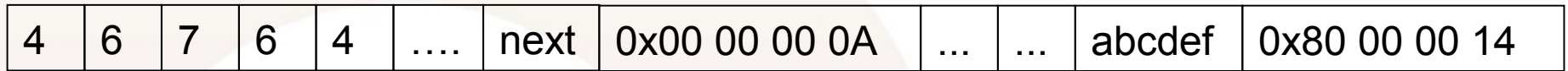
```
CREATE TABLE `t1` (  
  `ID` int(11) unsigned NOT NULL,  
  `NAME` varchar(120),  
  `N_FIELDS` int(10),  
  PRIMARY KEY (`ID`)  
) ENGINE=InnoDB DEFAULT  
CHARSET=latin1
```

| Name                | Size                                |
|---------------------|-------------------------------------|
| Field Start Offsets | (F*1) or (F*2) bytes                |
| Extra Bytes         | 6 bytes (5 bytes if COMPACT format) |
| Field Contents      | depends on content                  |

# REDUNDANT

A row: (10, 'abcdef', 20)

Actually stored as: (10, TRX\_ID, PTR\_ID, 'abcdef', 20)



Field Offsets

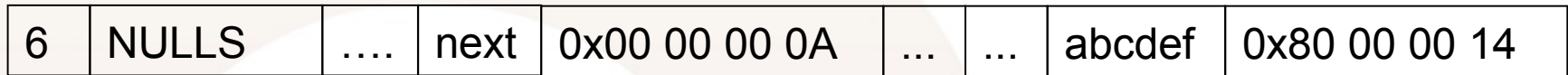
Extra 6 bytes:  
record\_status  
deleted\_flag  
min\_rec\_flag  
n\_owned  
heap\_no  
n\_fields  
1byte\_offs\_flag

Fields

# COMPACT

A row: (10, 'abcdef', 20)

Actually stored as: (10, TRX\_ID, PTR\_ID, 'abcdef', 20)



Field Offsets

Extra 5 bytes:

Fields

A bit per NULL-able field

# Data types

---

INT types (fixed-size)

String types

- VARCHAR(x) – variable-size
- CHAR(x) – fixed-size, variable-size if UTF-8

DECIMAL

- Stored in strings before 5.0.3, variable in size
- Binary format after 5.0.3, fixed-size.

# BLOB and other long fields

- Field length (so called offset) is one or two byte long
- Page size is 16k
- If record size  $< (\text{UNIV\_PAGE\_SIZE}/2 - 200) \approx \sim 7\text{k}$ 
  - the record is stored internally (in a PK page)
- Otherwise – 768 bytes internally, the rest in an external page

---

## 2. Internal system tables SYS\_INDEXES and SYS\_TABLES

# Why are SYS\_\* tables needed?

- Correspondence “table name” -> “index\_id”
- Storage for other internal information

# How MySQL stores data in InnoDB

## SYS\_TABLES and

Always REDUNDANT

Name:  
PRIMARY  
GEN\_CLUSTER\_ID  
or unique index name

```
CREATE TABLE `SYS_INDEXES` (  
  `TABLE_ID` bigint(20) unsigned NOT NULL  
    default '0',  
  `ID` bigint(20) unsigned NOT NULL default  
    '0',  
  `NAME` varchar(120) default NULL,  
  `N_FIELDS` int(10) unsigned default NULL,  
  `TYPE` int(10) unsigned default NULL,  
  `SPACE` int(10) unsigned default NULL,  
  `PAGE_NO` int(10) unsigned default NULL,  
  PRIMARY KEY (`TABLE_ID`, `ID`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```



index\_id = 0-3

```
CREATE TABLE `SYS_TABLES` (  
  `NAME` varchar(255) NOT NULL default '',  
  `ID` bigint(20) unsigned NOT NULL default  
    '0',  
  `N_COLS` int(10) unsigned default NULL,  
  `TYPE` int(10) unsigned default NULL,  
  `MIX_ID` bigint(20) unsigned default NULL,  
  `MIX_LEN` int(10) unsigned default NULL,  
  `CLUSTER_NAME` varchar(255) default NULL,  
  `SPACE` int(10) unsigned default NULL,  
  PRIMARY KEY (`NAME`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```



index\_id = 0-1

# How MySQL stores data in InnoDB

Example:

## SYS\_TABLES

| NAME                | ID        | ...            |
|---------------------|-----------|----------------|
| "archive/msg_store" | <b>40</b> | 8 1 0 0 NULL 0 |
| "archive/msg_store" | 40        | 8 1 0 0 NULL 0 |
| "archive/msg_store" | 40        | 8 1 0 0 NULL 0 |

## SYS\_INDEXES

| TABLE_ID | ID            | NAME       | ...            |
|----------|---------------|------------|----------------|
| 40       | <b>196389</b> | "PRIMARY"  | 2 3 0 21031026 |
| 40       | 196390        | "msg_hash" | 1 0 0 21031028 |

---

# 3. InnoDB Primary and Secondary keys

# Primary key

## The table:

```
CREATE TABLE `t1` (  
  `ID` int(11),  
  `NAME` varchar(120),  
  `N_FIELDS` int(10),  
  PRIMARY KEY (`ID`),  
  KEY `NAME` (`NAME`)  
) ENGINE=InnoDB DEFAULT  
  CHARSET=latin1
```

## Fields in the PK:

1. ID
2. DB\_TRX\_ID
3. DB\_ROLL\_PTR
4. NAME
5. N\_FIELDS

# Secondary key

## The table:

```
CREATE TABLE `t1` (  
  `ID` int(11),  
  `NAME` varchar(120),  
  `N_FIELDS` int(10),  
  PRIMARY KEY (`ID`),  
  KEY `NAME` (`NAME`)  
) ENGINE=InnoDB DEFAULT  
  CHARSET=latin1
```

## Fields in the SK:

1. NAME
2. ID ← Primary key

---

# 4. Typical failure scenarios

# Deleted records

- `DELETE FROM table WHERE id = 5;`
- Forgotten `WHERE` clause

Band-aid:

- Stop/kill `mysqld` ASAP

# How delete is performed?

**"row/row0upd.c":**

*".../\* How is a delete performed?...The delete is performed by setting the delete bit in the record and substituting the id of the deleting transaction for the original trx id, and substituting a new roll ptr for previous roll ptr. The old trx id and roll ptr are saved in the undo log record.*

*Thus, no physical changes occur in the index tree structure at the time of the delete. Only when the undo log is purged, the index records will be physically deleted from the index trees...."*

# Dropped table/database

- DROP TABLE table;
- DROP DATABASE database;
- Often happens when restoring from SQL dump
- Bad because .FRM file goes away
- Especially painful when innodb\_file\_per\_table

## Band-aid:

- Stop/kill mysqld ASAP
- Stop IO on an HDD or mount read-only or take a raw image

# Corrupted InnoDB tablespace

---

- Hardware failures
- OS or filesystem failures
- InnoDB bugs
- Corrupted InnoDB tablespace by other processes

## Band-aid:

- Stop mysqld
- Take a copy of InnoDB files

# Wrong UPDATE statement

- UPDATE user SET Password = PASSWORD('qwerty') WHERE User='root';
- Again forgotten WHERE clause
- Bad because changes are applied in a PRIMARY index immediately
- Old version goes to UNDO segment

Band-aid:

- Stop/kill mysqld ASAP

# 5. InnoDB recovery tool

# Recovery prerequisites

---

## 1. Media

1. ibdata1
2. \*.ibd
3. HDD image

## 2. Tables structure

1. SQL dump
2. \*.FRM files

# table\_defs.h

- generated by **create\_defs.pl**

```
{ /* int(11) unsigned */
  name: "ID",
  type: FT_UINT,
  fixed_length: 4,
  has_limits: TRUE,
  limits: {
    can_be_null: FALSE,
    uint_min_val: 0,
    uint_max_val: 4294967295ULL
  },
  can_be_null: FALSE
},
```

```
{ /* varchar(120) */
  name: "NAME",
  type: FT_CHAR,
  min_length: 0,
  max_length: 120,
  has_limits: TRUE,
  limits: {
    can_be_null: TRUE,
    char_min_len: 0,
    char_max_len: 120,
    char_ascii_only: TRUE
  },
  can_be_null: TRUE
},
```

# How to get CREATE info from .frm files

1. CREATE TABLE t1 (id int) Engine=INNODB;
2. Replace t1.frm with the one's you need to get scheme
3. Run "show create table t1"

If mysqld crashes

1. See the end of `bvi t1.frm`:  
`.ID.NAME.N_FIELDS..`
2. \*.FRM viewer **!TODO**

# InnoDB recovery tool

<http://launchpad.net/percona-innodb-recovery-tool/>

1. Written in Percona
2. Contributed by Percona and community
3. Supported by Percona

Consists of two major tools

- page\_parser – splits InnoDB tablespace into 16k pages
- constraints\_parser – scans a page and finds good records

# InnoDB recovery tool

## page\_parser

```
server# ./page_parser -4 -f /var/lib/mysql/ibdata1
Opening file: /var/lib/mysql/ibdata1
Read data from fn=3...
Read page #0.. saving it to pages-1259793800/0-18219008/0-00000000.page
Read page #1.. saving it to pages-1259793800/0-0/1-00000001.page
Read page #2.. saving it to pages-1259793800/4294967295-65535/2-00000002.page
Read page #3.. saving it to pages-1259793800/0-0/3-00000003.page
```

# Page signature check

```
0{...0...4...4...=..E.  
.....  
.....<..~...A.....|..  
.....  
.....  
.....  
...infimum.....supremumf..  
....qT  
M/T/196001834/XXXXX  
XXXXXXXXXXXXX L  
X X X X X X X X X  
XXXXXXXXXXXXX
```

1. INFIMUM and SUPREMUM records are in fixed positions
2. Works with corrupted pages

# InnoDB recovery tool

## constraints\_parser

```
server# ./constraints_parser -4 -f pages-1259793800/0-16/51-00000051.page
```

Table structure is defined in "include/table\_defs.h"



Filters inside table\_defs.h are very important

See HOWTO for details

<http://code.google.com/p/innodb-tools/wiki/InnodbRecoveryHowto>

# Check InnoDB page before reading recs

```
#!/constraints_parser -5 -U -f pages/0-418/12665-00012665.page -V
```

```
Initializing table definitions...
```

```
Processing table: document_type_fieldsets_link
```

```
- total fields: 5  
- nullable fields: 0  
- minimum header size: 5  
- minimum rec size: 25  
- maximum rec size: 25
```

```
Read data from fn=3...
```

```
Page id: 12665
```

```
Checking a page
```

```
Infimum offset: 0x63
```

```
Supremum offset: 0x70
```

```
Next record at offset: 0x9F (159)
```

```
Next record at offset: 0xB0 (176)
```

```
Next record at offset: 0x3D95 (15765)
```

```
...
```

```
Next record at offset: 0x70 (112)
```

```
Page is good
```

1. Check if the tool can follow all records by addresses
2. If so, find a rec. exactly at the position where the record is.
3. Helps a lot for COMPACT format!

# Import result

|    |   |              |    |
|----|---|--------------|----|
| t1 | 1 | "browse"     | 10 |
| t1 | 2 | "dashboard"  | 20 |
| t1 | 3 | "addFolder"  | 18 |
| t1 | 4 | "editFolder" | 15 |

```
mysql> LOAD DATA INFILE '/path/to/datafile'  
REPLACE INTO TABLE <table_name>  
FIELDS TERMINATED BY '\t' OPTIONALLY ENCLOSED BY  
''''  
LINES STARTING BY '<table_name>\t' ;
```

# Questions ?

Thank you for coming!

- References

- <http://www.mysqlperformanceblog.com/>
- <http://percona.com/>
- <http://www.slideshare.net/guest808c167/recovery-of-lost-or-corrupted-inno-db-tablesmysql-uc-2010>

**Applause 🎉**