



PERCONA
Performance Consulting Experts

Data Recovery for MySQL

Istvan Podor

Percona Webinar

26 / April / 2011

Introduction

- Support Engineer at Percona
- InnoDB recovery walk-through
- Data recovery for beginners :)

About the presentation

- InnoDB recovery
- Not explaining deep innodb architecture ...
- Showing you an example instead
- Feel free to interrupt and ask questions

Data set

- Sample database available at <http://downloads.mysql.com/docs/sakila-db.zip>
- Table: `store`
- 2 rows only

```
mysql> show create table store\G
***** 1. row *****
      Table: store
Create Table: CREATE TABLE `store` (
  `store_id` tinyint(3) unsigned NOT NULL auto_increment,
  `manager_staff_id` tinyint(3) unsigned NOT NULL,
  `address_id` smallint(5) unsigned NOT NULL,
  `last_update` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  PRIMARY KEY (`store_id`),
  UNIQUE KEY `idx_unique_manager` (`manager_staff_id`),
  KEY `idx_fk_address_id` (`address_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Some basic things about InnoDB

- InnoDB not storing tables, but clustered primary indexes
- Tables are identified by `index_id`
- Belonging index ID for tables is in InnoDB data dictionary (`SYS_TABLES`, `SYS_INDEXES`)
- Split common table space file (`ibdata`) in to pages
- Find the belonging index id to your table
- Create a table definition file
- Extract 'lost' data
- Restore data to MySQL

Table content

- As simple as:

```
debianvm:~/recovery# mysql sakila -e 'select * from store';
```

```
+-----+-----+-----+-----+
| store_id | manager_staff_id | address_id | last_update          |
+-----+-----+-----+-----+
|         1 |                 1 |           1 | 2006-02-15 04:57:12 |
|         2 |                 2 |           2 | 2006-02-15 04:57:12 |
+-----+-----+-----+-----+
```

Get the tool

- Download and compile first:
 - <https://code.launchpad.net/~percona-dev/percona-innodb-recovery-tool/trunk>
 - Use the latest version in the trunk
- Just simply run `make` to compile

Let's drop it

- Just drop the table from mysql

```
mysql> select * from store;
+-----+-----+-----+-----+
| store_id | manager_staff_id | address_id | last_update |
+-----+-----+-----+-----+
|          1 |                  1 |           1 | 2006-02-15 04:57:12 |
|          2 |                  2 |           2 | 2006-02-15 04:57:12 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> drop table store;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from store;
ERROR 1146 (42S02): Table 'sakila.store' doesn't exist
mysql> █
```

Grab Ibddata

- Stop MySQL and make a copy of the ibdata file

```
debianvm:~/recovery# mkdir ibdata_file
debianvm:~/recovery# /etc/init.d/mysql stop
Stopping MySQL database server: mysqld.
debianvm:~/recovery# cp /var/lib/mysql/ibdata1 ./ibdata_file/
debianvm:~/recovery# █
```

Where are we at the moment

- Table dropped, MySQL stopped
- Recovery tool is in ~/recovery/
- Ibddata file copied to ~/recovery/ibdata_file

Run page_parser

- Nothing needed to be done, once the tool is compiled
- page_parser splits InnoDB tables space into pages, sorts by index_id (using InnoDB page signatures)
- Our main directory will look like 'pages-\$current_timestamp'
- Innodb pages will be splitted by Primary Key ID like : 'pages-\$current_timestamp/FIL_PAGE_INDEX/0-91'

Run page_parser

```

debianvm:~/recovery# ./page_parser -4 -f ibdata_file/ibdata1
Opening file: ibdata_file/ibdata1:
2049          ID of device containing file
133955       inode number
33184        protection
1            number of hard links
0            user ID of owner
0            group ID of owner
0            device ID (if special file)
48676866     total size, in bytes
4096         blocksize for filesystem I/O
95184        number of blocks allocated
1303542936   time of last access
1303542470   time of last modification
1303542470   time of last status change
48676866     Size to process in bytes
104857600    Disk cache size in bytes
48.01% done. 2011-04-23 09:17:08 ETA(in 00:00 hours). Processing speed: 23368227 B/sec
89.61% done. 2011-04-23 09:17:08 ETA(in 00:00 hours). Processing speed: 20251884 B/sec
debianvm:~/recovery# ls -lF | grep pages
drwxr-xr-x  3 root root    4096 2011-04-23 09:17 pages-1303543026/
debianvm:~/recovery# █

```

- Our workdir is 'pages-1303543025'

Extract basic SYS files

- InnoDB stores indexes and not tables
- To restore the table, we need its index ID
- At first we need to restore InnoDB dictionary (SYS_TABLES, SYS_INDEXES), to identify the index_id
- SYS tables have a fixed structure and it's always in redundant format
- Let's try to locate where is our `store` table ...

Get sys_tables and sys_indexes

- This is the way to find where the belonging pages stored for the `store` table
- There are two files in include/
 - table_defs.h.SYS_INDEXES
 - table_defs.h.SYS_TABLES
- sys_tables are always in pages_xxx/FIL_PAGE_INDEX/0-1
- sys_indexes are always in pages_xxx/FIL_PAGE_INDEX/0-3

(things will be clear soon...)

Get sys_tables list

- Make a symlink and recompile
`cd include/; rm table_defs.h \
ln -s table_defs.h.SYS_TABLES table_defs.h`
- Recompile ...

```
debianvm:~/recovery# cd include/  
debianvm:~/recovery/include# ls -laF  
total 48  
drwxr-xr-x  2 root root 4096 2011-04-23 08:17 ./  
drwxr-xr-x 10 root root 4096 2011-04-23 09:17 ../  
-rw-r--r--  1 root root  392 2011-04-23 08:17 check_data.h  
-rw-r--r--  1 root root  112 2011-04-23 08:17 error.h  
-rw-r--r--  1 root root 4154 2011-04-23 08:17 innochecksum.h  
-rw-r--r--  1 root root  769 2011-04-23 08:17 print_data.h  
lrwxrwxrwx  1 root root   23 2011-04-23 08:17 table_defs.h -> table_defs.h.SYS_TABLES  
-rw-r--r--  1 root root 1497 2011-04-23 08:17 table_defs.h.SYS_COLUMNS  
-rw-r--r--  1 root root 3732 2011-04-23 08:17 table_defs.h.SYS_INDEXES  
-rw-r--r--  1 root root 4412 2011-04-23 08:17 table_defs.h.SYS_TABLES  
-rw-r--r--  1 root root 2145 2011-04-23 08:17 tables_dict.h  
debianvm:~/recovery/include# rm
```

Get sys_tables list

- Run constraint parser to get the table list with index ids
- File format is -4 (redundant) for SYS tables

```
`.constraints_parser -4 -f pages-1303543026/FIL_PAGE_INDEX/0-1/* | awk
'sakila/store"/ && /SYS_TABLES/'`
```

```
debianvm:~/recovery# ./constraints_parser -4 -f pages-1303543026/FIL_PAGE_INDE
LOAD DATA INFILE '/root/recovery/dumps/default/SYS_TABLES' REPLACE INTO TABLE
Y ENCLOSED BY '"' LINES STARTING BY 'SYS_TABLES\t' (NAME, ID, N_COLS, TYPE, MI
SYS_TABLES      "sakila/store" 45      2147483652      1      0      0
debianvm:~/recovery# █
```

- Our DB is `sakila` and we are looking for 'SYS_TABLES' ID
- The tool already return a command to restore the table, don't get disturbed
- Note the number in the 3rd column, it's 45

Get sys_indexes list

- Make a symlink and recompile
`cd include/; rm table_defs.h \
ln -s table_defs.h.SYS_INDEXES table_defs.h`
- Recompile ...

```
debianvm:~/recovery/include# ls -laF
total 48
drwxr-xr-x  2 root root 4096 2011-04-23 09:49 ./
drwxr-xr-x 10 root root 4096 2011-04-23 09:17 ../
-rw-r--r--  1 root root  392 2011-04-23 08:17 check_data.h
-rw-r--r--  1 root root  112 2011-04-23 08:17 error.h
-rw-r--r--  1 root root 4154 2011-04-23 08:17 innochecksum.h
-rw-r--r--  1 root root  769 2011-04-23 08:17 print_data.h
-rw-r--r--  1 root root 1497 2011-04-23 08:17 table_defs.h.SYS_COLUMNS
-rw-r--r--  1 root root 3732 2011-04-23 08:17 table_defs.h.SYS_INDEXES
-rw-r--r--  1 root root 4412 2011-04-23 08:17 table_defs.h.SYS_TABLES
lrwxrwxrwx  1 root root   24 2011-04-23 09:49 table_defs.hu -> table_defs.h.SYS_INDEXES
-rw-r--r--  1 root root 2145 2011-04-23 08:17 tables_dict.h
debianvm:~/recovery/include# █
```

Get sys_indexes list

- Run constraint parser to get the table list with index ids
- File format is -4 (redundant) for SYS indexes

```
`. /constraints_parser -4 -f pages-1303543026/FIL_PAGE_INDEX/0-3/* | awk '/45/ && /PRIMARY/'`
```

```
debianvm:~/recovery# ./constraints_parser -4 -f pages-1303543026/FIL_PAGE_INDEX/0-3/* | awk '/45/ && /PRIMARY/'
LOAD DATA INFILE '/root/recovery/dumps/default/SYS_INDEXES' REPLACE INTO TABLE SYS_INDEXES
LINES ENCLOSED BY '"' LINES STARTING BY 'SYS_INDEXES\t' (TABLE_ID, ID, NAME, N_F
SYS_INDEXES      37      73      "PRIMARY"      2      3      0      452
SYS_INDEXES      44      94      "PRIMARY"      1      3      0      45
SYS_INDEXES      45      97      "PRIMARY"      1      3      0      429496
debianvm:~/recovery#
```

- We are looking for the Primary index ID where the tables ID is 45
- The tool already return a command to restore the table, don't get disturbed
- Note the number in the 2nd column, it's 45, we need the value from the 3rd column
- It's 97! Our magic number here.

Now comes the hard part

- We need to create a `table_defs.h` for our table
- `table_defs.h` defines the structure of the table we want to restore
- And for what the tool should look for

table_defs.h's structure

- Fortunately, we have a tool to create our table definition file
- Delete the table_defs.h symlink in the include dir first
- Re-create the store table in mysql with the exact same structure

Run create_defs.pl

- The tool will connect to mysql and create a definition file

```
debianvm:~/recovery# ./create_defs.pl --db sakila --table store | head -n 20
#ifndef table_defs_h
#define table_defs_h

// Table definitions
table_def_t table_definitions[] = {
    {
        name: "store",
        {
            { /* tinyint(3) unsigned */
                name: "store_id",
                type: FT_UINT,
                fixed_length: 1,

                has_limits: FALSE,
                limits: {
                    can_be_null: FALSE,
                    uint_min_val: 0,
                    uint_max_val: 255
                }
            },
        }
    }
};

debianvm:~/recovery# █
```

Run create_defs.pl

- As we've seen before, it works. Let's redirect the output to `./include/table_defs.h`

```
debianvm:~/recovery# ./create_defs.pl --db sakila --table store > include/table_defs.h
debianvm:~/recovery# █
```

- Recompile the tool

Let's see what we have

- If you are lucky, that's all, constraint_parser will be able to restore your table.
- Remember the magic number of 97 we extracted before
- And keep the table data in mind what we got in the beginning

```
debianvm:~/recovery# mysql sakila -e 'select * from store';
```

store_id	manager_staff_id	address_id	last_update
1	1	1	2006-02-15 04:57:12
2	2	2	2006-02-15 04:57:12

Run constraints_parser

- No we can run the constraint parser against real data
- File format here is compact
 - -4 for redundant
 - -5 is for compact

```
debianvm:~/recovery# ./constraints_parser -5 -f pages-1303543026/FIL_PAGE_INDEX/0-97/*
store  1      1      1      "2006-02-15 03:57:12"
store  2      2      2      "2006-02-15 03:57:12"
LOAD DATA INFILE '/root/recovery/dumps/default/store' REPLACE INTO TABLE `store` FIELDS TERMINATED
BY '\t' OPTIONALLY ENCLOSED BY '"' LINES STARTING BY 'store\t' (store_id, manager_staff_id, address
_id, last_update)
debianvm:~/recovery# █
```

- It's pretty much look like what we need! :)

The dump file

- Redirect the output to a file (~/.recovery/default/store)

```
debianvm:~/recovery# ./constraints_parser -5 -f pages-1303543026/FIL_PAGE_INDEX/0-97/* > /root/recovery/default/store
LOAD DATA INFILE '/root/recovery/dumps/default/store' REPLACE INTO TABLE `store` FIELDS TERMINATED BY '\t' OPTIONALLY ENCLOSED BY '"' LINES STARTING BY 'store\t' (store_id, manager_staff_id, address_id, last_update)
debianvm:~/recovery# █
```

Load data back

- Now you can load your data back
- MySQL needs access to the file (I moved it to /var/lib/mysql)
- As the table have FOREIGN KEYS we are not able to use REPLACE INTO like the tool printed it. So I used just simply `LOAD DATA INFILE '/var/lib/mysql/store' INTO TABLE store ...`
- Let's see what's happened....

The result

```
debianvm:~/recovery# mysql sakila
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 48
Server version: 5.0.51a-24+lenny5-log (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> LOAD DATA INFILE '/var/lib/mysql/store' INTO TABLE `store` FIELDS TERMINATED BY '\t' OPTIONA
LLY ENCLOSED BY '"' LINES STARTING BY 'store\t' (store_id, manager_staff_id, address_id, last_updat
e);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from store;
+-----+-----+-----+-----+
| store_id | manager_staff_id | address_id | last_update          |
+-----+-----+-----+-----+
|          1 |                1 |          1 | 2006-02-15 03:57:12 |
|          2 |                2 |          2 | 2006-02-15 03:57:12 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> █
```

It's not always this easy ...

- If you restore from corrupted table spaces, you might have to tune your table_defs.h created by create_defs.pl

//few hints//

- The table_defs.h file basically describe each column in the table
- You can always see two hidden column in the table defs file:

```
{ /* */
    name: "DB_TRX_ID",
    type: FT_INTERNAL,
    fixed_length: 6,

    can_be_null: FALSE
},
{ /* */
    name: "DB_ROLL_PTR",
    type: FT_INTERNAL,
    fixed_length: 7,

    can_be_null: FALSE
},
```

table_defs.h

- A definition of a smallint(5) and a timestamp column look like this:

```
{ /* smallint(5) unsigned */
    name: "address_id",
    type: FT_UINT,
    fixed_length: 2,

    has_limits: FALSE,
    limits: {
        can_be_null: FALSE,
        uint_min_val: 0,
        uint_max_val: 65535
    },

    can_be_null: FALSE
},
{ /* timestamp */
    name: "last_update",
    type: FT_TIMESTAMP,
    fixed_length: 4,

    can_be_null: FALSE
},
```

- Try on narrowing down the possible values!

Making table_defs.h more precise

- Try on narrowing down the possible values!
- We can remember how our data looked like. This smallint(5) def is too wide! This means the minimum value can be 0, and the largest is around 65k.

```
uint_min_val: 0,  
uint_max_val: 65535
```

- But we remember that the largest value was 2 and the smallest was 1
- We can change it like this:

```
uint_min_val: 1,  
uint_max_val: 10
```
- You can change a timestamp column to HAVE_LIMITS = TRUE
- And so on... This ain't easy

File_per_table

- If you drop the table with File_per_table, InnoDB will delete the file. Remount the partition in READ_ONLY and run the tool against the disk image.

```
./page_parser -f /dev/sda
```

- If you got the IBD file (some rows deleted from the table)
 - Run page_parser against the simple ibd file
 - Look for the smallest index_id inside of the pages directory where page_parser extracts ibd pages. (e.g. pages_xxx/FIL_PAGE_INDEX/0-15)
 - Do the same steps like before

Questions/Comments ?

- It's not easy
- However, you should try!
- Download the slides, try to repeat
- Email me if you have (easy) questions
 - istvan.podor at percona .com
 - // www.percona.com
 - // <https://launchpad.net/percona-innodb-recovery-tool>