



PERCONA
Performance Consulting Experts

InnoDB Recovery Techniques

MySQL Conference and
Expo

April 20-23 2009

Santa Clara, CA

by Peter Zaitsev, Percona Inc

Reasons for Data Loss

- Hardware Failure
- Hardware Misconfiguration (ie excessive caching)
- Software Misconfiguration
 - Sharing tablespace data or MySQL directory
- Maintenance error
 - Copying files while Innodb is still up, removing logs etc.
- MySQL/Innodb, OS Kernel Bugs
- File System Corruption
- User Error (Drop table, Wrong Delete or Update)
- Security breach

Two Types of Data Loss

- “Corruption”
 - Innodb is unable to start or access some data
- “Data Missing”
 - Rows Deleted
 - Tables Dropped
- You can have both of these at the same time at badly damaged system

Sources of Data for Recovery

- InnoDB Tablespace
 - Main InnoDB tablespace contains deleted rows and tables
- Undo Space
 - Contains previous row versions until they are purged
- Free space on the hard drive
 - Dropped tables with multiple tablespaces
- InnoDB Logs (only very recently added/modified)
- Insert Buffer
- Binary logs
- Full Query Logs
- **Backups :)**

Consolidation

- Recovery offer does not recover all data
- Recover by multiple methods or multiple sources when possible
- Consolidate them together to get the most
- Example
 - 3 months old backup
 - Data in binary logs for last 7 days
 - Recovery of deleted data using innodb tools
 - Some data was already overwritten

What to do if you found a data loss

- Further writes destroy data
- Kill -9 mysqld may make sense to prevent writes
- Prevent writes to partition storing data if some files were deleted or filesystem was corrupted
 - take the binary snapshot of all partition/device
- **Have backup of your corrupted data before you attempt any recovery**

Recovery Processes

- Simple Innodb Corruption
- .frm/.idb Inconsistences
- Major corruptions and deleted data

Simple Corruption Recovery

Symptoms of “Simple Corruption”

- InnoDB crashes with page corruption errors while accessing certain data
- InnoDB crashes with page corruption on startup

InnoDB: Database page corruption on disk or a failed

InnoDB: file read of page 50.

InnoDB: You may have to recover from a backup.

InnoDB: Page checksum 2993571300, prior-to-4.0.14-form checksum
264318302

InnoDB: stored checksum 3243635356, prior-to-4.0.14-form stored
checksum 264318302

InnoDB: Page lsn 0 54681, low 4 bytes of lsn at page end 54681

InnoDB: Page number (if stored to page already) 50,

InnoDB: space id (if created with >= MySQL-4.1.1 and stored already) 0

InnoDB: Page may be an update undo log page

InnoDB: Page may be an index page where index id is 0 15

Where corrupted page belongs ?

- **Secondary Index**
 - Very good. Can just rebuild the table
- **Primary Key leaf page**
 - Good. Can skip only data from the page
- **Primary Key “Branch page”**
 - May need to use innodb tools for best recovery
- **Undo Tablespace, Insert Buffer**
 - May need to dump and reload whole tablespace
- **System/Dictionary Tables**
 - May need to use innodb_tools to get anything

innodb_force_recovery magic

- Only use this option for recovery.
 - Always set to 0 in production use
- Make sure application does not access server during recovery
 - **--skip-networking --socket=/tmp/mysecret.sock**
- Stops innodb assertions and background jobs
 - Innodb will still crash on corrupted data
- Values 1-6; larger – more stuff skipped
- Values up to 4 give you relatively consistent data
- Start from lower; increase if Innodb still crashes

Fixing Secondary Key corruption

- Typically no `innodb_force_recovery` is needed
- **ALTER TABLE corrupted ENGINE=INNODB**
 - Scans the table (using primary key) and rebuilds it
- **SELECT INTO OUTFILE / LOAD DATA INFILE** is another option.

Fixing Primary Key Corruption

- Check if corruption does not cause the crash if dumped with `innodb_force_recovery=1`
- If table has PRIMARY KEY:
- **SELECT * FROM TBL WHERE ID BETWEEN A AND B**
 - Going over all table in ranges to see what can be read
 - Can use “adaptive” technique – reading large ranges, retrying them in small ranges if it fails
 - MySQL will often crash and recover many times in the process. May consider reducing logs.
- No PK – may go by secondary key.

Undo Tablespace/Insert Buffer/Log

- Fatal Wound. Will need to recreate tablespace after dumping the data.
- Data may be inconsistent (ie non committed changes seen).
- Do table by table mysqldump
 - Some tables may fail
- Do the “ranges” recovery for tables which can't be dumped successfully
- Recreate database and reload from Dump
- **Setup proper backups**

System Tables Corruption

- Data may become completely inaccessible
- Any SELECTs from table crash; **SHOW TABLE STATUS** crashes etc.
- Dump tables which can be dumped
- Recovery using Innodb tool may be best choice

.frm/.idb Inconsistency

MySQL Meta Data

- MySQL with Innodb holds 2 copies of meta data
- **.frm** file holds general definition
- Innodb Data Dictionary in System Tablespace holds similar data for Innodb Needs
- **.ibd** file does not hold meta data but should match
 - General format
 - Tablespace id
- Multiple cases of inconsistency are possible
 - Look at the most important ones

Clean Up InnoDB Dictionary

- The .ibd and .frm are deleted but Internal dictionary was not updated for some reason.
- Need to clean up data dictionary to reload table
- Create another InnoDB table in the same database
- Copy its frm file to **a.frm**

```
mysql> create table a(i int) type=innodb;  
ERROR 1005 (HY000): Can't create table './pztest/a.frm' (errno: 121)
```

```
090417 17:10:29 InnoDB: Error: table `pztest/a` already exists in InnoDB internal  
InnoDB: data dictionary. Have you deleted the .frm file  
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE  
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
```

Cleaning up InnoDB Dictionary cont

- Now you can drop the table
 - **DROP TABLE a;**
- Clean up remainder
 - (.ibd file may remain from failed CREATE TABLE)
- Confirm you have a message in error log:
 - Ignore other messages about mismatching table space ids, These are expected

InnoDB: We removed now the InnoDB internal data dictionary entry
InnoDB: of table `pztest/a`.

“Connecting” .ibd file

- You should do this only on the installation this .ibd file originated from
 - Other server will have issues with different LSNs
- **create table a(i int) engine=innodb**
 - Create the new table with same structure
- **alter table a discard tablespace**
 - Discard empty tablespace as we'll get the new one.
- Copy **a.ibd** to the proper location
- **mysql> alter table a import tablespace;**
 - ERROR 1030 (HY000): Got error -1 from storage engine

“Connecting” .ibd file

- Check out error logs:
 - InnoDB: Error: tablespace id in file './pztest/a.ibd' is 2857, but in the InnoDB InnoDB: data dictionary it is 2861.
- Let us change a.ibd so it has tablespace number InnoDB expects
 - Use BVI: <http://bvi.sourceforge.net/download.html>
- The tablespace ID is stored at positions x25 and x29
- **select hex(2861)** - get the number in hex
- **mysql> alter table a import tablespace;**
 - Query OK, 0 rows affected (0.00 sec)
- More details: <http://www.chriscalender.com/?p=28>

Using innodb-tools for recovery

About Innodb-Tools

- Tool to do low level data extraction from Innodb
 - <http://code.google.com/p/innodb-tools/>
- GPL; Written and maintained by Percona
- Offers a lot of flexibility recovering the data
- Contains multiple scripts to
 - “Sort” data by “Index_ID”
 - Prepare parser definitions from CREATE TABLE
 - Extract the data from given “Index_ID”

General Recovery Process

- Split Tablespace(s) by INDEX_ID
 - Run **page_parser**
- Create table_defs.h
 - Definitions for the data format
 - **./create_defs.pl --user=u --password=p --db=test --table=recover > table_defs.h**
- Build the constraints_parser
- Run extraction process
 - Run **constraints_parser**
 - Repeat it adjusting definition if you get too little/too much data

Deleted Rows

- Finding INDEX_ID is easy as table still exists
 - **CREATE TABLE innodb_table_monitor(x int) engine=innodb;**
 -
 - INDEX: name PRIMARY, id 0 254, fields 1/7, type 3
 - root page 271, appr.key vals 1, leaf pages 1, size pages 1
 - **./constraints_parser -f /innodb-recovery-0.3/pages-1239037839/0-254/50-00000050.page -5**
 - Run for each page or “cat” them to the file

Dropped Table

- Assume `innodb_file_per_table=0`
 - Otherwise data is not in the innodb tablespace
- We can't find `INDEX_ID` from data dictionary
 - We can run recover process for `SYS_TABLES` and `SYS_INDEXES` to “undelete” deleted dictionary row
 - Or grep parsed Innodb data for some known data which was deleted.
- Consider using **-U** flag for parser to only recover non deleted rows.

Recovery of Secondary Keys

- Secondary key do not contain all data
 - But can be very valuable if PRIMARY KEY data was overwritten
 - Especially good for long “covering” indexes
- Secondary index is similar to PK just lacks some extra system fields
- If you have Index (A,B) and PRIMARY KEY(ID)
 - You need to define fields (A,B,ID) in table_defs.h

Advanced Recovery

- `Innodb_file_per_table=1`
 - Undelete file if possible
 - Parse full partition as innodb data file
 - Warning there could be multiple unwanted copies recovered
 - Fragmentation may cause some 16K pages to be in different locations.
 - May need to modify tool to increment offsets by 4K while looking for proper pages.
- Trashed filesystem
 - Follow same process

Final Notes

- The tool is in constant development
 - Using latest tree may help
 - Expect Bugs
- Data may become quickly overwritten
 - Do not count on 100% recovery.
- Backups, Backups, Backups
 - Do not only have them but test them and monitor