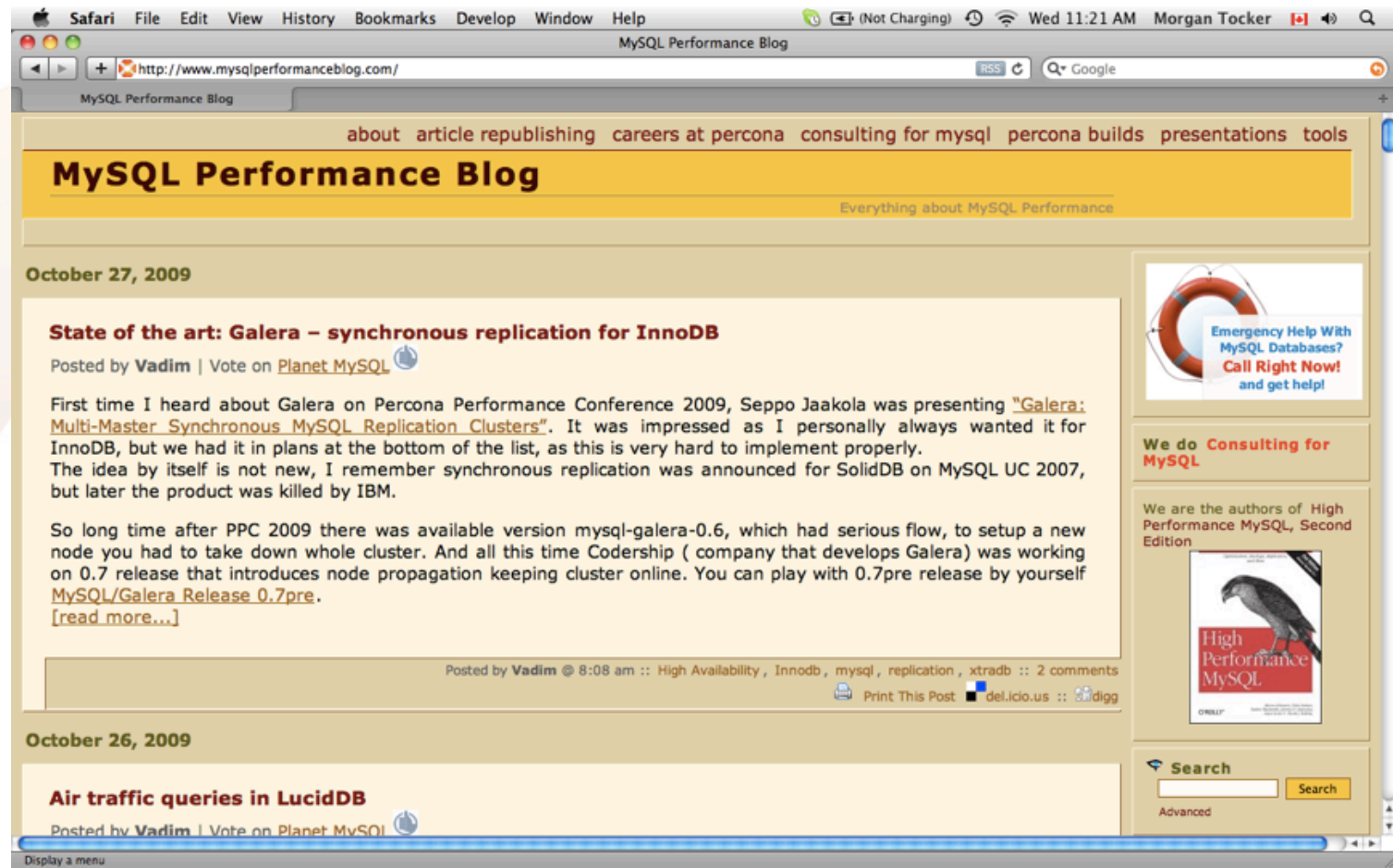

Introduction to InnoDB/XtraDB Tuning

MySQL Meetup Los Angeles
18 November 2009

Morgan Tocker, [firstname at percona dot com](mailto:firstname@percona.com)
Director of Training, Percona Inc.

You probably know Percona as:



The screenshot shows a Safari browser window displaying the MySQL Performance Blog. The browser's address bar shows the URL <http://www.mysqlperformanceblog.com/>. The page features a navigation menu with links: [about](#), [article republishing](#), [careers at percona](#), [consulting for mysql](#), [percona builds](#), [presentations](#), and [tools](#). The main heading is "MySQL Performance Blog" with the tagline "Everything about MySQL Performance".

The main content area is dated "October 27, 2009" and features a post titled "State of the art: Galera – synchronous replication for InnoDB". The post is attributed to Vadim and includes a "Vote on Planet MySQL" link. The text of the post discusses the Galera Multi-Master Synchronous MySQL Replication Clusters presented at the Percona Performance Conference 2009, noting that while the idea is not new, its implementation is challenging. It mentions the availability of version mysql-galera-0.6 and the ongoing work on version 0.7, which includes node propagation. A "[read more...]" link is provided at the end of the post.

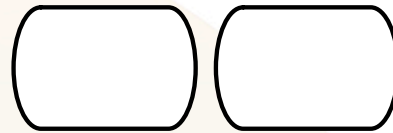
Below the main post, there is a section dated "October 26, 2009" with a post titled "Air traffic queries in LucidDB".

On the right side of the page, there are several promotional elements: a "Emergency Help With MySQL Databases? Call Right Now! and get help!" banner, a "We do Consulting for MySQL" advertisement, and a book cover for "High Performance MySQL, Second Edition" by Percona. A search bar is also visible at the bottom right of the page.

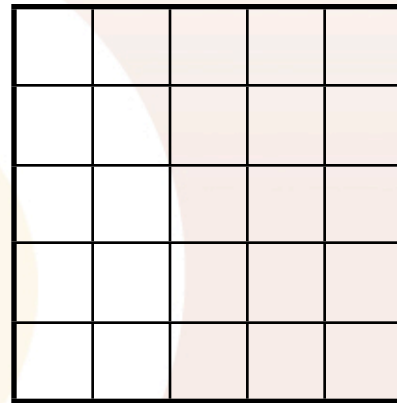
About this talk:

- ★ I think you need to understand how InnoDB works to know how to tune it.
- ★ When we've passed the theory, then we can look at how certain optimizations effect InnoDB:
 - ◆ Hardware changes
 - ◆ Configuration changes, etc.
- ★ Most of this applies to both InnoDB and XtraDB. I'll point out XtraDB specifics.

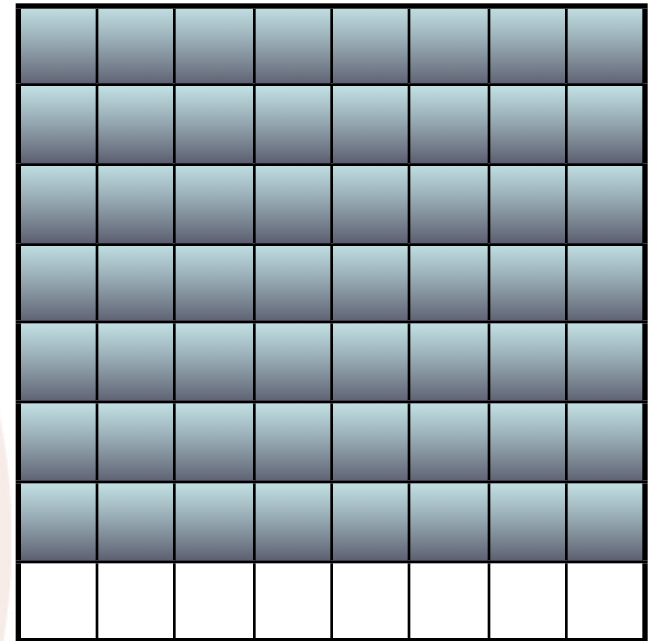
Basic Operation (High Level)



Log Files



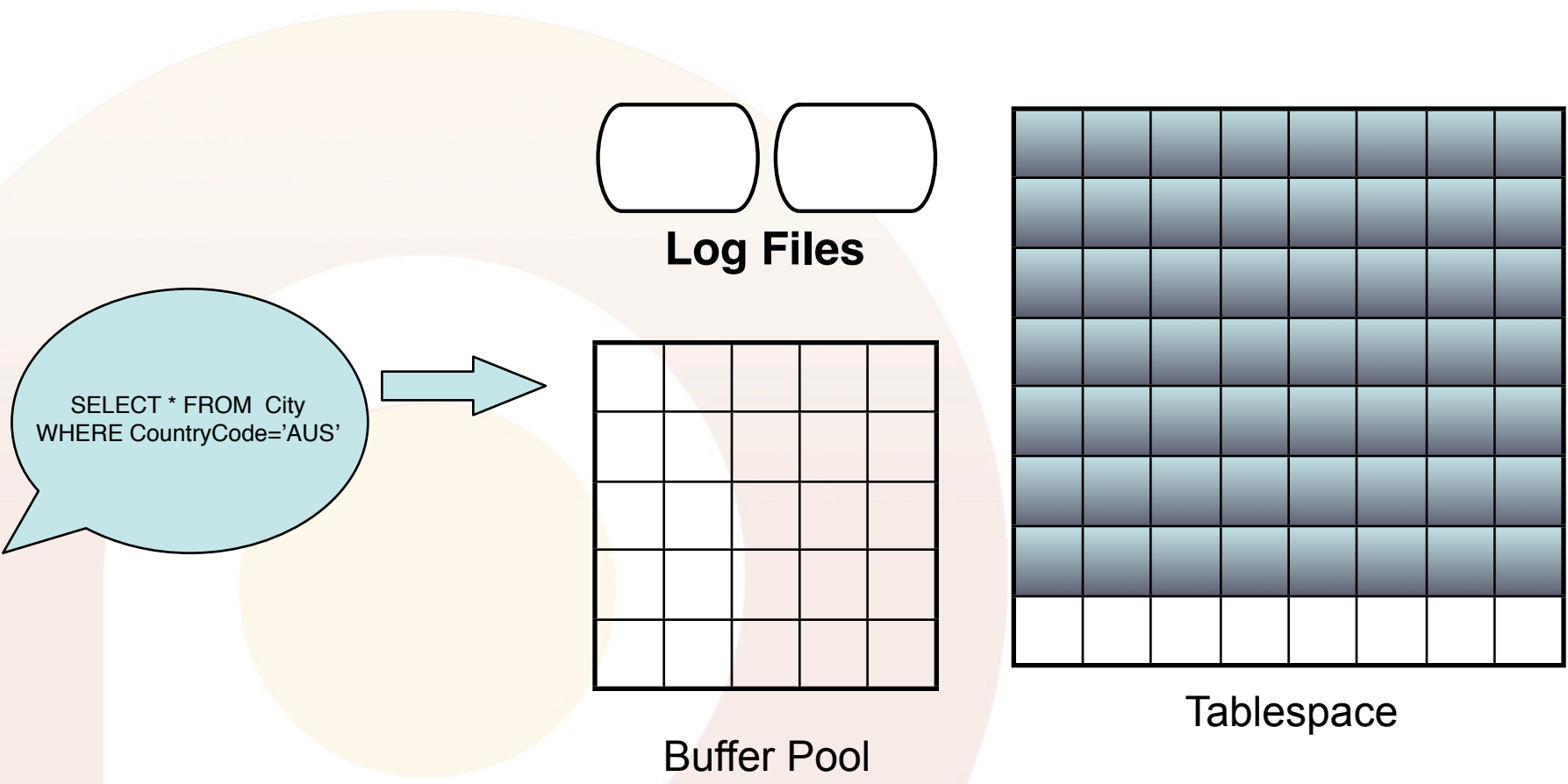
Buffer Pool



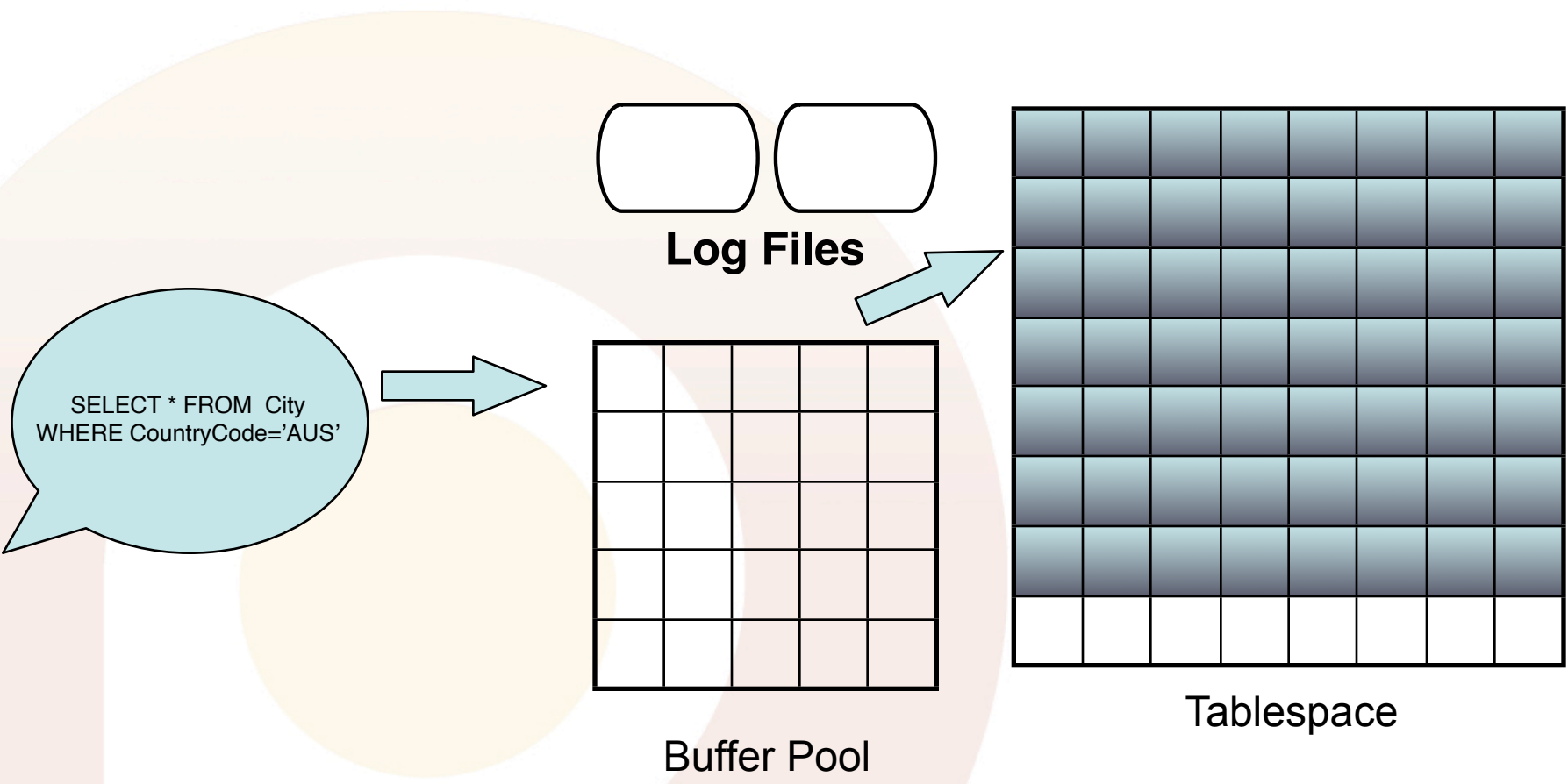
Tablespace

SELECT * FROM City
WHERE CountryCode='AUS'

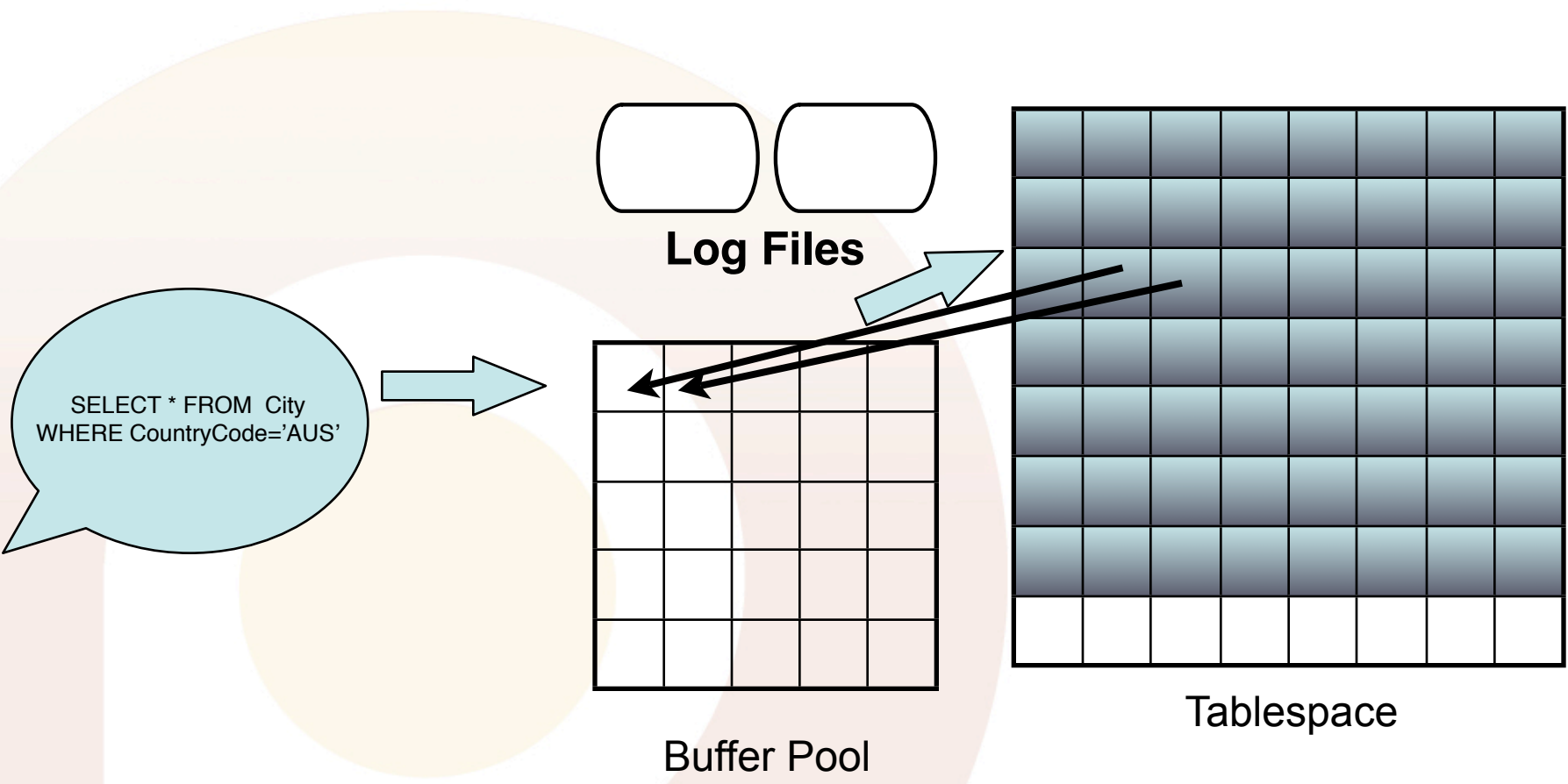
Basic Operation (High Level)



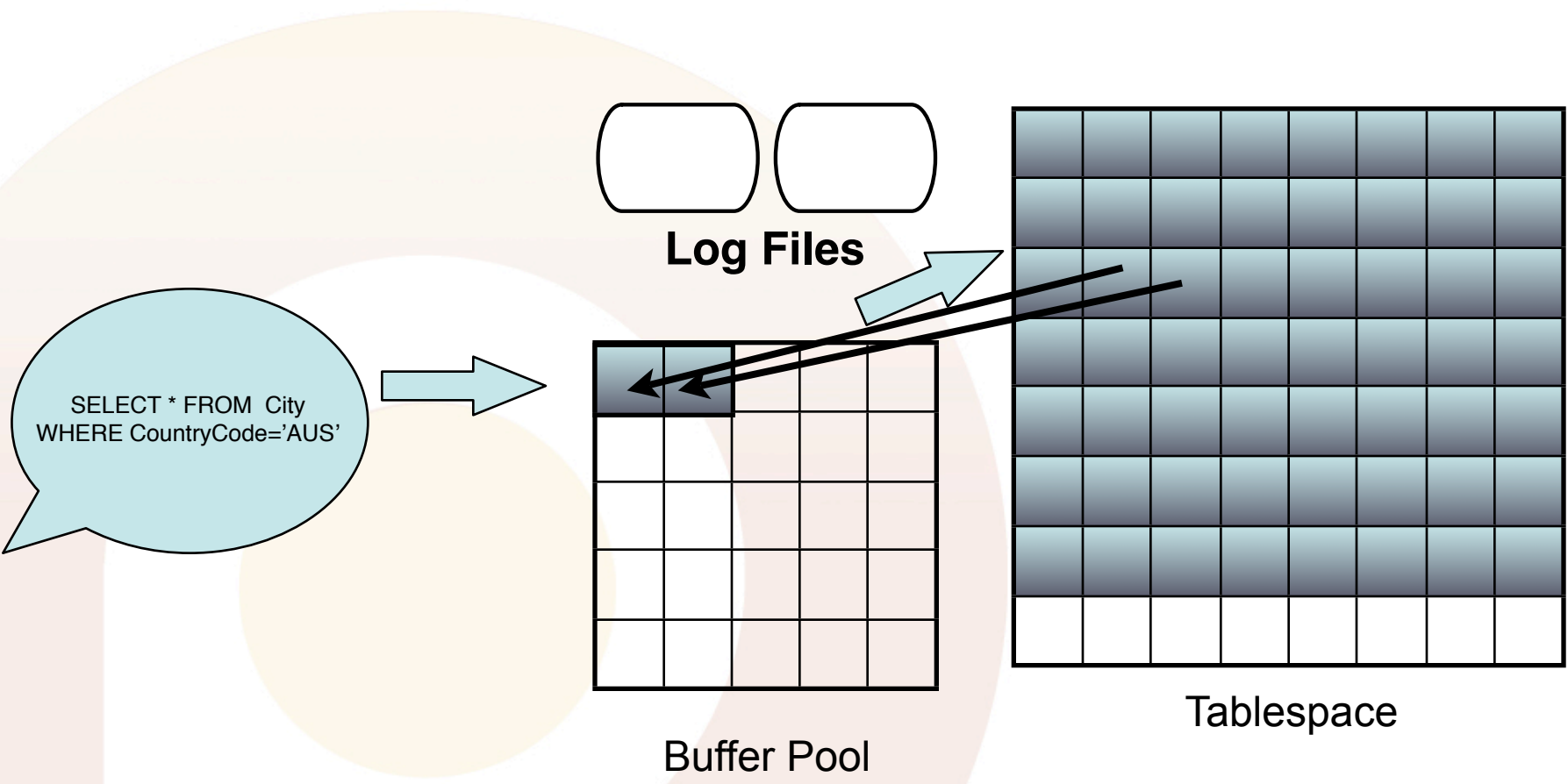
Basic Operation (High Level)



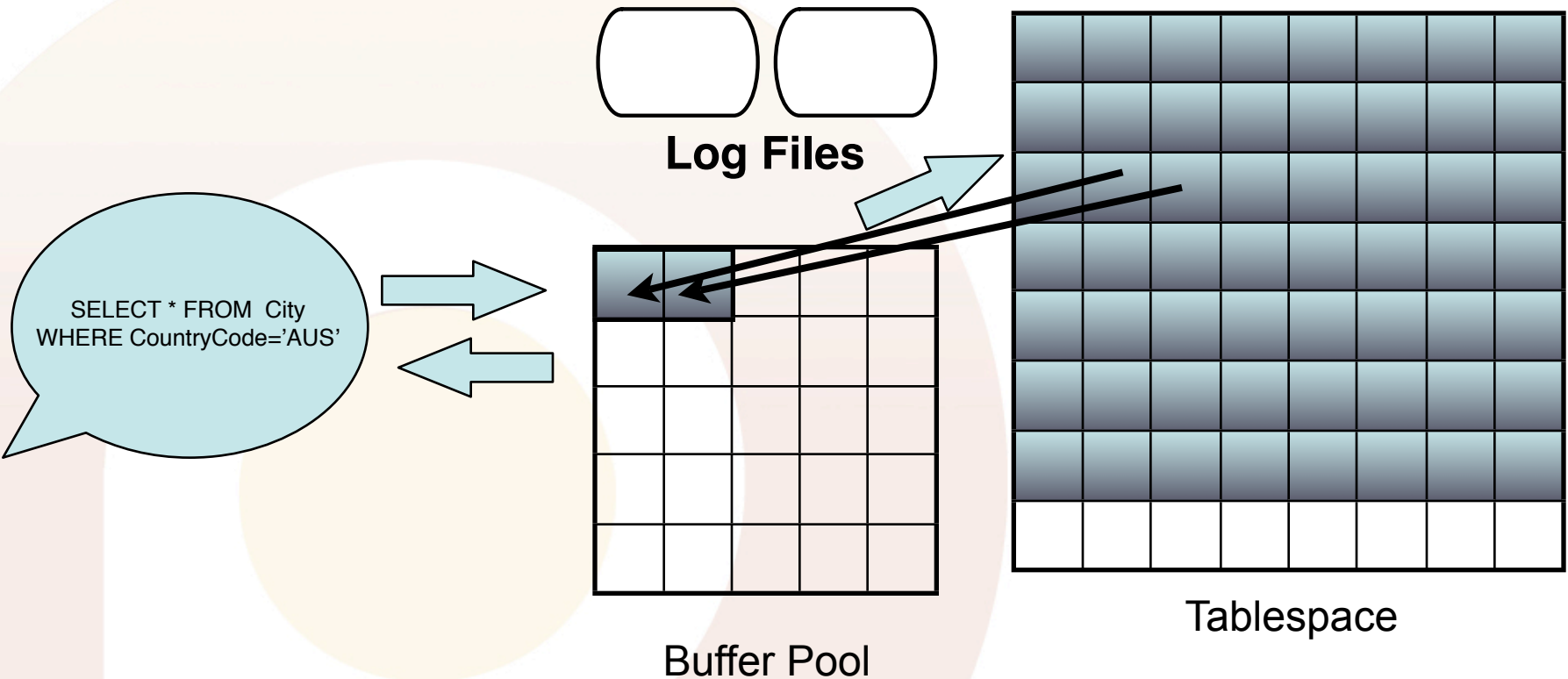
Basic Operation (High Level)



Basic Operation (High Level)



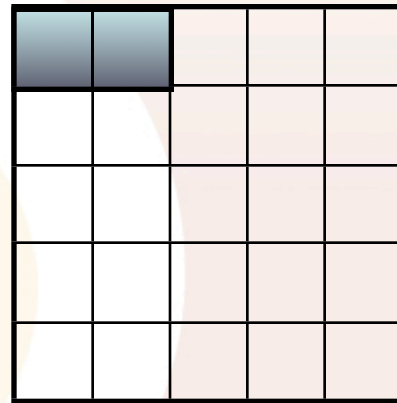
Basic Operation (High Level)



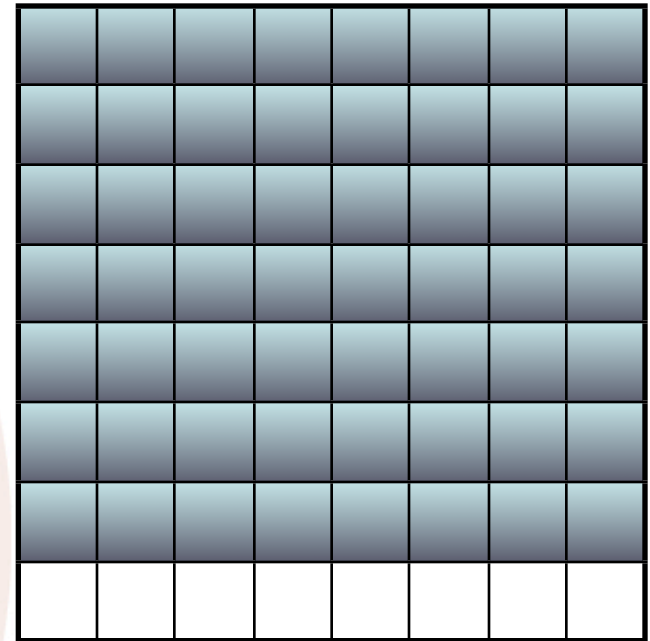
Basic Operation (cont.)



Log Files



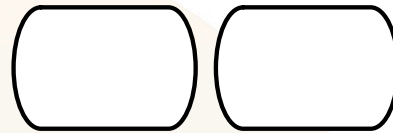
Buffer Pool



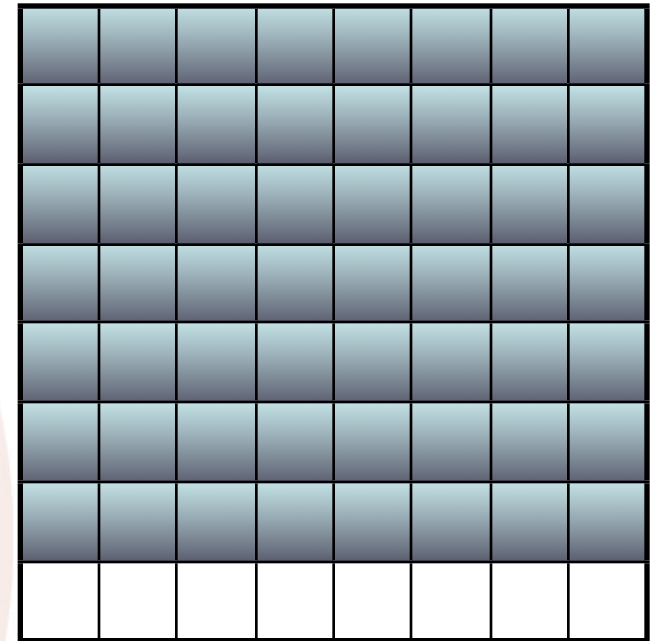
Tablespace

UPDATE City SET
name = 'Morgansville'
WHERE name = 'Brisbane'
AND CountryCode='AUS'

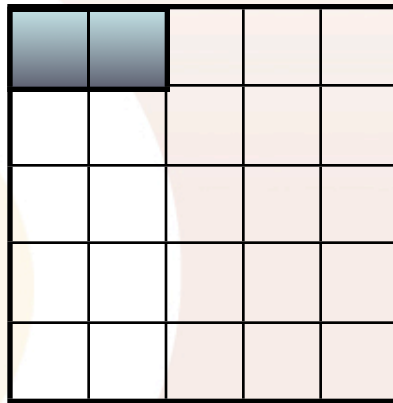
Basic Operation (cont.)



Log Files

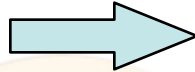


Tablespace

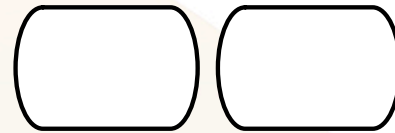


Buffer Pool

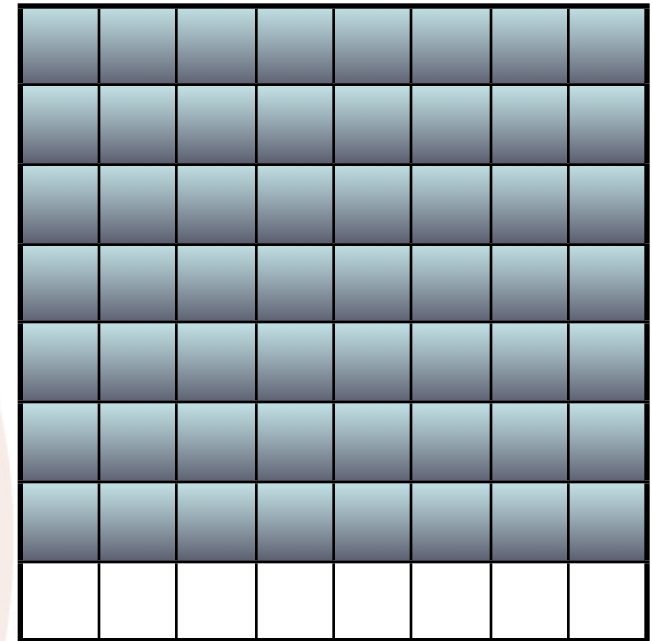
UPDATE City SET
name = 'Morgansville'
WHERE name = 'Brisbane'
AND CountryCode='AUS'



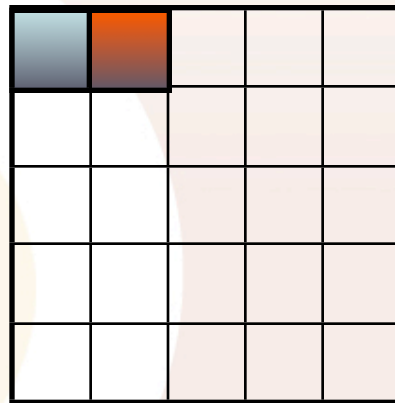
Basic Operation (cont.)



Log Files

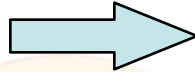


Tablespace

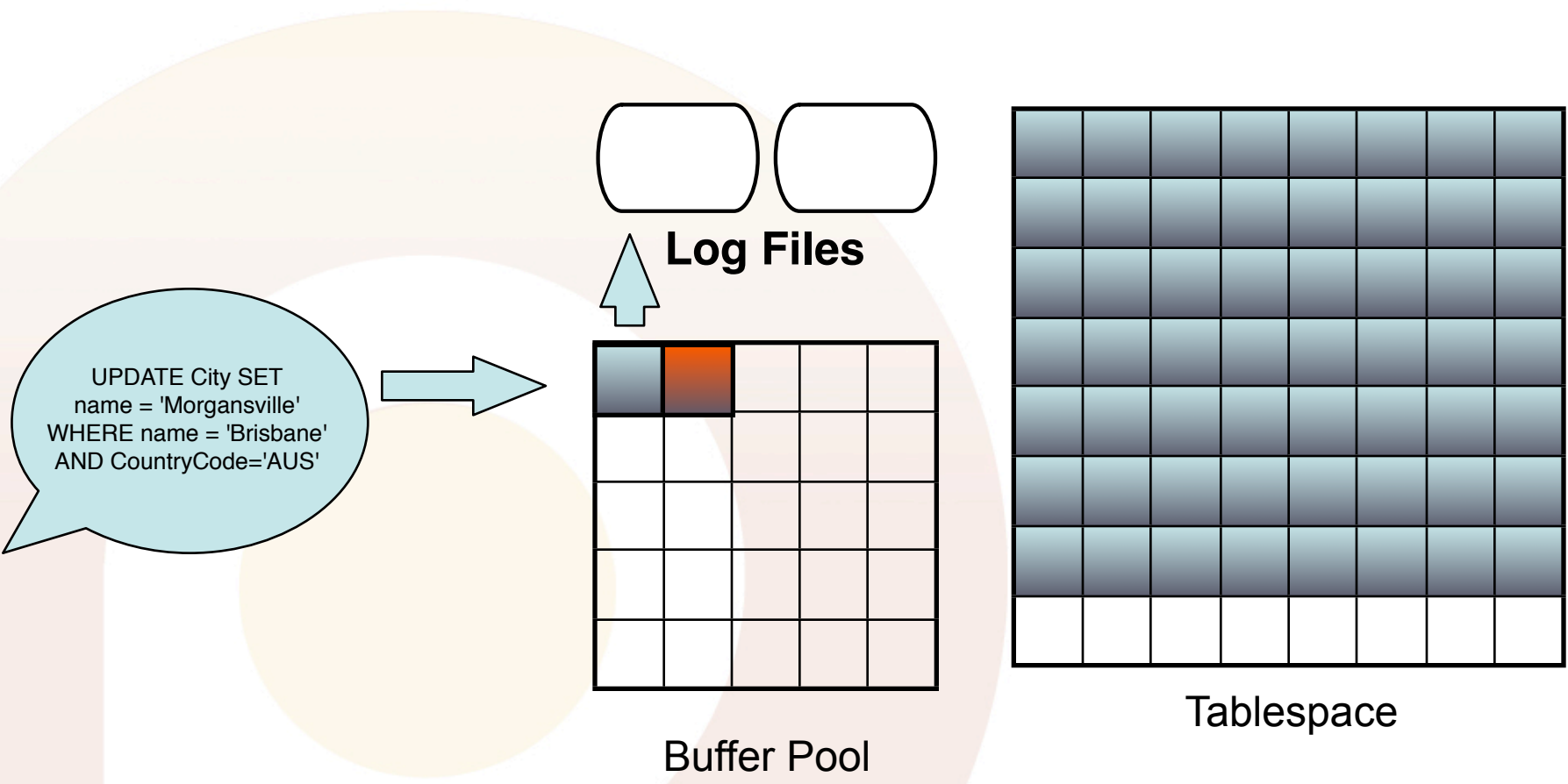


Buffer Pool

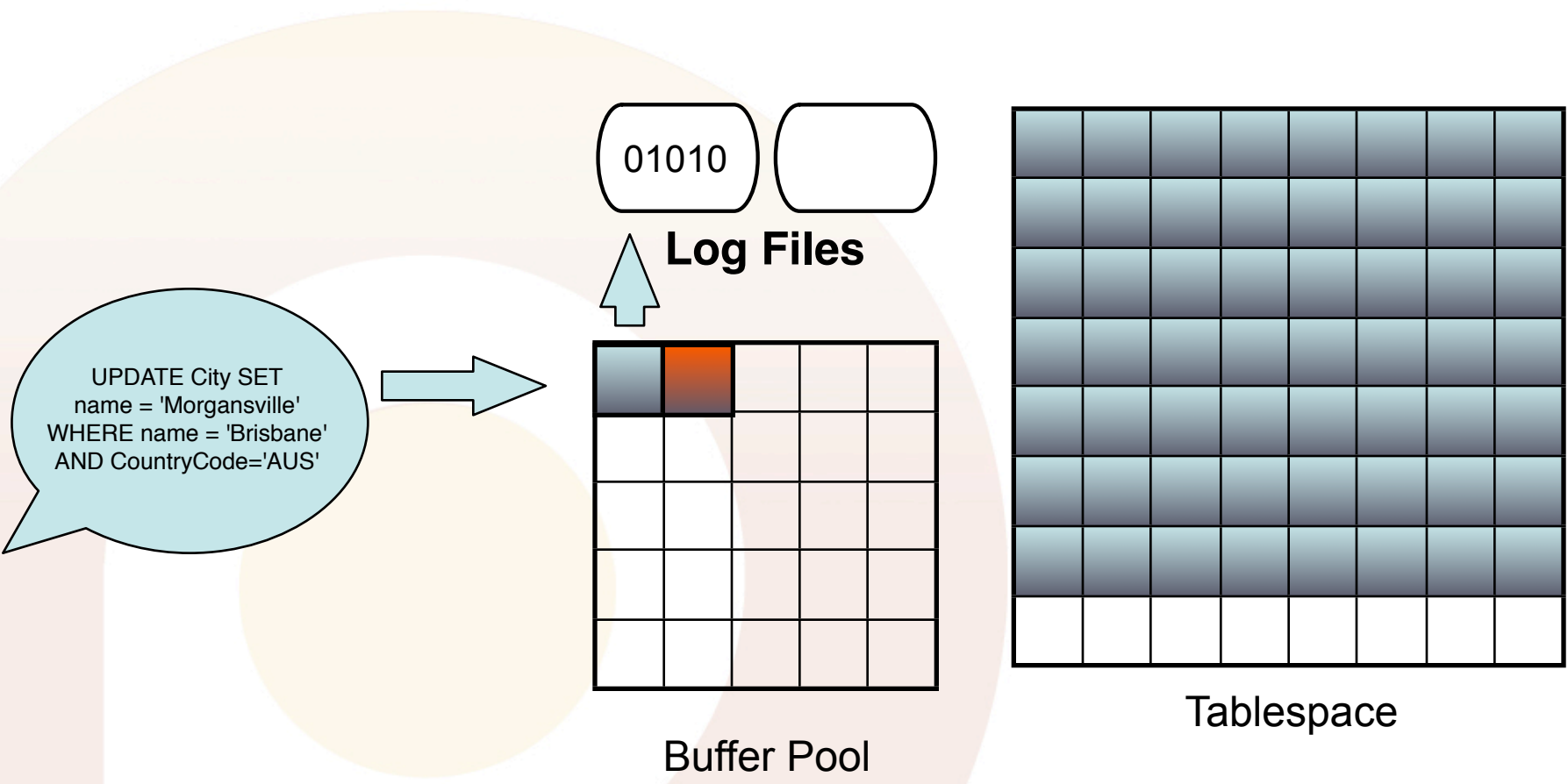
UPDATE City SET
name = 'Morgansville'
WHERE name = 'Brisbane'
AND CountryCode='AUS'



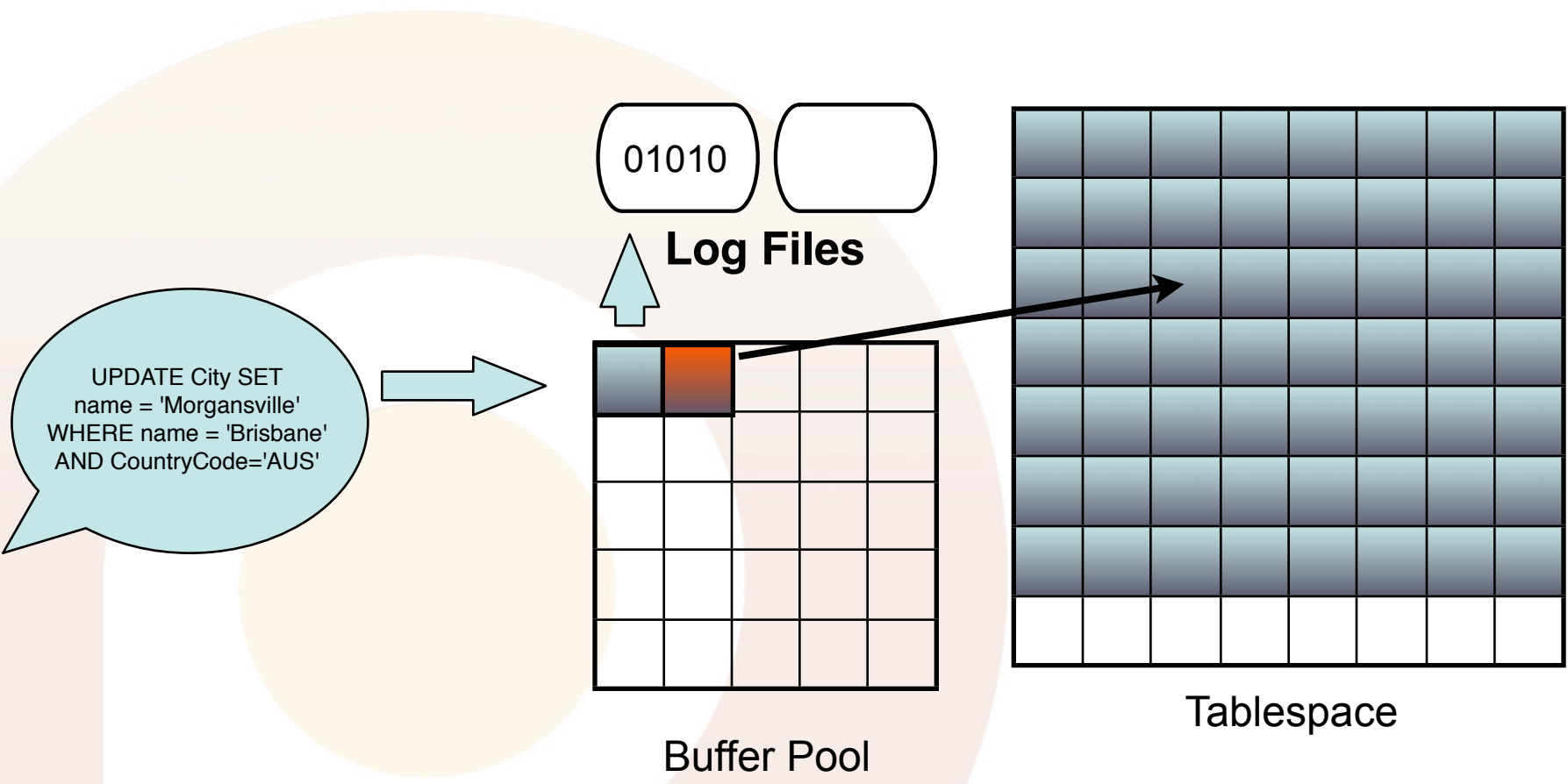
Basic Operation (cont.)



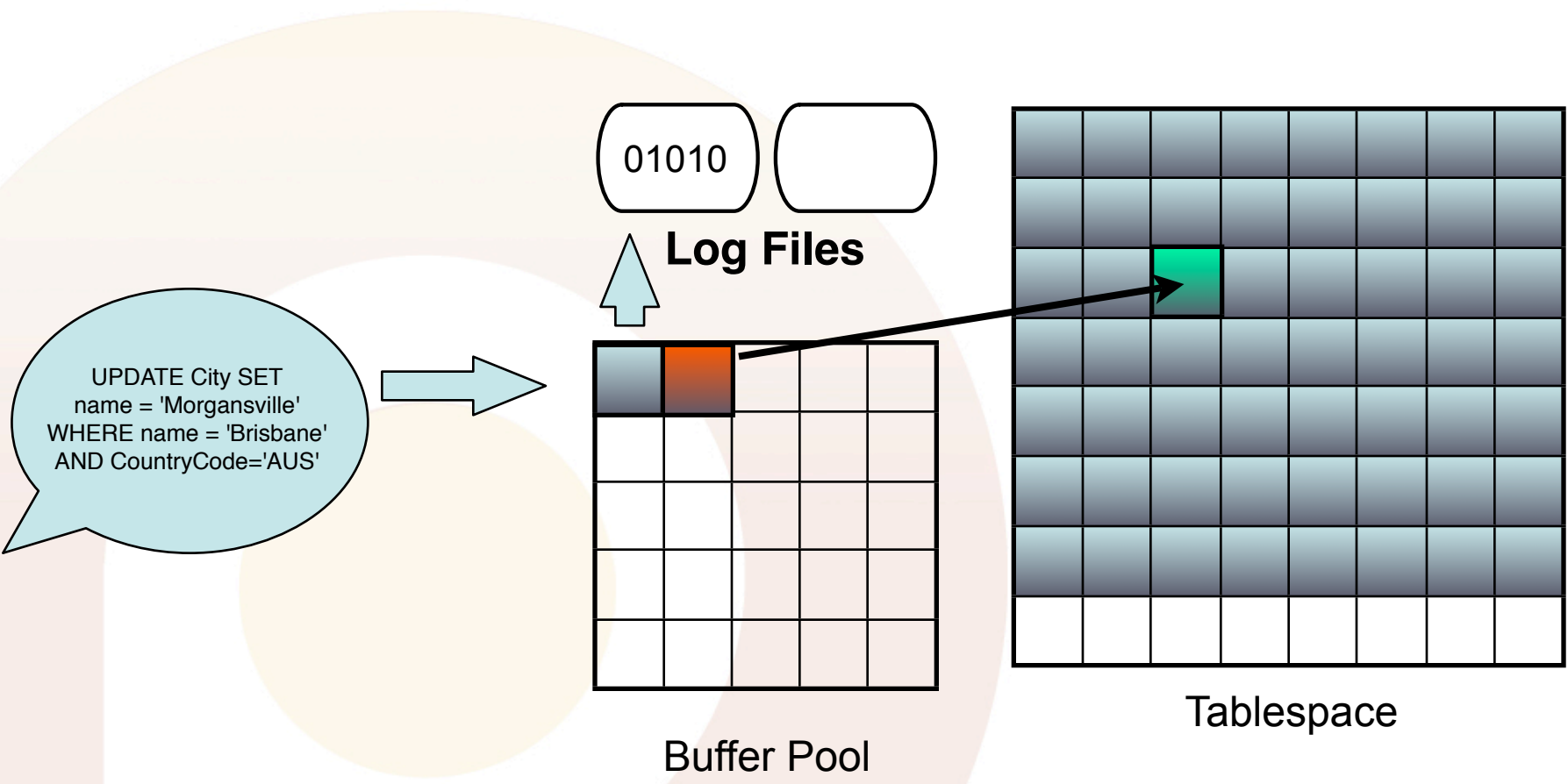
Basic Operation (cont.)



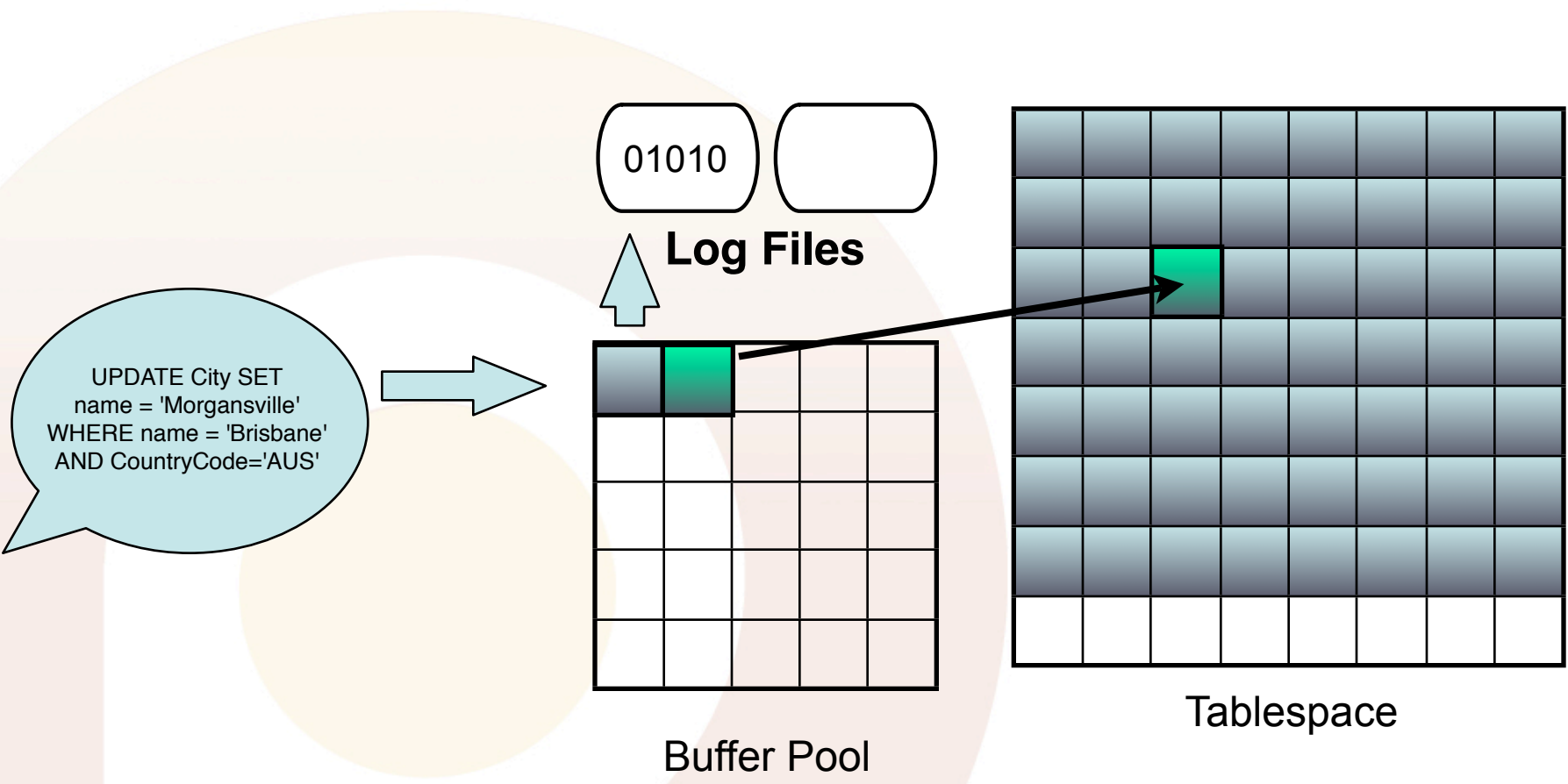
Basic Operation (cont.)



Basic Operation (cont.)



Basic Operation (cont.)

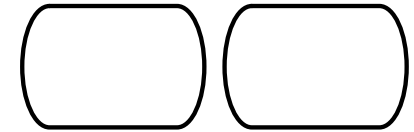


Why don't we update?

- ★ This is an optimization!
 - ✦ The log file IO is sequential and much cheaper than live updates.
 - ✦ The IO for the eventual updates to the tablespace can be optimized as well.
- ★ Provided that you saved enough to recover, this shouldn't matter should it?

More on Logs...

- ★ Logs are only used during recovery.
 - ★ Not even read when we need to write down dirty pages!
- ★ Freeing items from the buffer pool is done by examining at **the flush list** (dirty pages) and an **LRU** list.
- ★ Log activities all have an associated *Log Sequence Number* (LSN).



Log Files

Log Files, Checkpoints, etc.

- ★ Most database systems work this way:
 - ★ In Oracle the transaction logs are called “Redo Logs”.
- ★ The background process of syncing dirty pages is normally referred to as a “Checkpoint”.
- ★ InnoDB has *fuzzy* checkpointing.

Recovery Process

- ★ We replay through the log files, and figure out what changes need to be applied starting from the oldest LSN.
- ★ Since dirty pages were not flushed in order, the first part of recovery is normally very fast -
 - ★ Pages internally store their most up to date LSN, and it may already be up to date.

Okay, Theory is Over.

Log Writing

- ★ A lot of InnoDB performance tuning centers around the log files - the defaults of 5M is *very* conservative:
 - ✦ To change, you need to safely shutdown MySQL, delete the old log files, then increase `innodb_log_file_size`.
 - ✦ **Tip:** Optionally you could change `innodb_log_files_in_group` as well. Be aware that your effective log file is `innodb_log_file_size` * `innodb_log_files_in_group`

Log Writing (cont.)

- ★ You can also change `innodb_flush_log_at_trx_commit` to 0 or 2 to reduce the durability requirements of this write.
 - ✦ Requires less flushing - particularly helpful on systems without writeback caches!
- ★ `innodb_log_buffer_size` can sometimes be increased to batch the activity written to the log file at once (see `SHOW GLOBAL STATUS` like `'innodb_log_waits'`).

Other insane defaults:

Defaults
improve in
XtraDB &
MySQL 5.4!

- ★ The InnoDB buffer pool defaults to 8M.
- ★ These settings should be more like:
 - ♦ 70-80% of system memory for `innodb_buffer_pool_size` and 64M to 256M for `innodb_log_file_size`.
- ★ More information:
 - ♦ <http://www.mysqlperformanceblog.com/2006/09/29/what-to-tune-in-mysql-server-after-installation/>
 - ♦ <http://www.mysqlperformanceblog.com/2007/11/01/innodb-performance-optimization-basics/>

But never follow my advice blindly..

★

```
-----  
BUFFER POOL AND MEMORY  
-----
```

```
Total memory allocated 4648979546; in additional pool  
allocated 16773888
```

```
Buffer pool size      262144
```

```
Free buffers          0
```

```
Database pages       258053
```

```
Modified db pages    37491
```

```
Pending reads        0
```

```
Pending writes: LRU 0, flush list 0, single page 0
```

```
Pages read 57973114, created 251137, written 10761167
```

```
9.79 reads/s, 0.31 creates/s, 6.00 writes/s
```

```
Buffer pool hit rate 999 / 1000
```

Other (usually safe) optimizations:

- ★ Move the log files to separate spindles. Since all log writes are sequential IO, this normally helps quite a bit.
 - ✦ Not safe to do if your backup method is LVM - can't snapshot across volumes.
- ★ Set `innodb_flush_method=O_DIRECT` to use direct (unbuffered) IO.
 - ✦ This may result in less performance when you have no raid controller, or a very cheap one.

What's broken in InnoDB?

InnoDB Limitations (as at 5.0)

Slow Crash Recovery Process ●	Not enough diagnostic information, particularly around threads that write data/sync ●	Only one buffer pool. No QoS of mapping tables to buffer pool or pinning indexes/content to prevent eviction. ●
Poor Multi CPU Scalability ●	Broken group commit support ●	No way to see contents of buffer pool. ●
No way to limit the memory resident data dictionary size. ●	No features for warming big buffer pools on server start. ●	The adaptive hash does not suit all workloads. ●
Not able to take advantage of a more powerful IO system, that can sustain multiple concurrent threads. ●	No real ability to configure tablespaces - just two limited options. ●	Page flushing is not aggressive enough, early enough leading up to checkpoints. ●
Insert buffer shows weakness - can be up to 1/2 the buffer pool size - and doesn't make active attempts to be more aggressive at contracting when reaching limit. ●	IO read ahead assumptions have no configuration options / ability to disable. ●	Limited number of undo segments limits concurrent transactions to 1024. ●

InnoDB Limitations (as at 5.0)

Can't move tables between servers. ●	Slow statistics not available in slow query log. ●	Replication is not transactional. ●
No way to force checkpoint	Can't cluster on an index other than Primary key.	Opening tables is serialized by LOCK_Open mutex.
No way to freeze checkpoint/flushing activity. ●	auto_increment scalability is very bad. ●	No parallel query execution plans.
Adding files to a table space must be done via configuration file not online.	Statistics sampling is done by 10 random dives - limited control over resampling ●	Index statistics don't persist on restart and are recalculated each time.
Can't change page sizes without recompile. Not possible to have multiple page sizes. ●	Diagnostics - Can't see a history of deadlocks.	Can't control page fill factor.

InnoDB Limitations (as at 5.0)

InnoDB pages have checksums - a very helpful feature to detect silent corruption. The problem is there's 2 checksums and there may be benefit from being able to change the algorithm.	Further improvements possible to IO. InnoDB's emulated async IO may not be required. Newer system calls like fallocate/fadvise may lead to improvements.	No memory manager or effective way to limit memory use. This is both true for MySQL and the overhead consumed with InnoDB meta data.
Insert buffer does not assist for delete operations.	Dropping an index recreates the whole table. ●	Indexes can not be added online
InnoDB per page memory/storage overhead could probably be reduced.	There are no features to compress/pack indexes.	There is no support for additional index algorithms (such as hash or bitmap)

IO scalability enhancements

- ★ `--innodb_io_capacity` - Set the number of IO operations per second the server is capable of to influence background thread algorithms (default 100).
 - ★ 100 IOPS is about the capability of a single 7200RPM disk.
- ★ `--innodb_read_io_threads` and `--innodb_write_io_threads` - Using multiple IO threads will often lead to better performance on bigger raid systems.

IO Scalability (cont.)

- ★ Ability to control/disable features which assume physical alignment optimization - `--innodb_read_ahead ('none', 'random', 'linear', 'both')` and `--innodb_flush_neighbor_pages (0,1)`

Fast Crash Recovery.

- ★ Crash recovery in InnoDB can be very slow. From MySQL BUG #29847:

[28 Oct 2008 21:40] James Day

One reported effect of this performance limitation is that a system with 24GB buffer pool size could only recover 10% after 2 hour. With a 4G buffer pool and `innodb_flush_method=O_DIRECT` removed the system recovered completely in 30 minutes.

Partial workarounds.

1. During recovery, temporarily reduce `innodb_buffer_pool_size` to force InnoDB to flush pages from the flush list. A value of 4G is likely to be reasonable.
2. During recovery, temporarily remove `O_DIRECT` so that the operating system can cache changes during recovery.

Crash Recovery (cont.)

- ★ Is Single Threaded.
- ★ Depends on the size of `--innodb_log_file_size`
 - ✦ It's not linear - 256M can be a lot worse than 128M, and most values > 512M mean "take a vacation".
- ★ Depends on the size of the updates in the log file, and how far behind dirty page flushing was at the time of crash.
 - ✦ Seen in `SHOW INNODB STATUS` under "LOG":

```
Log sequence number 84 3000620880
Log flushed up to   84 3000611265
Last checkpoint at  84 2939889199
```

Fast Crash Recovery (cont.)

- ★ In our own tests, was x10 improvement.
- ★ Should be safe, but not enabled by default.
- ★ Can be enabled with:
 - ★ `--innodb_fast_recovery=1`

Adaptive Checkpointing

- ★ The built-in InnoDB can have erratic dips in performance as it approaches the end of a log file and needs to checkpoint.
- ★ The checkpoint process is supposed to be background only, but if it needs to - foreground tasks will be suspended until new log operations can occur.

Unpatched TPC-C benchmark

- ★ Pictures speak louder than words:



Source: <http://www.mysqlperformanceblog.com/2008/11/13/adaptive-checkpointing/>

Adaptive Checkpointing (cont.)

- ★ So the default behaviour can cause a denial of service
- ★ In XtraDB, there is an option to enable adaptive checkpointing `--innodb_adaptive_checkpoint =1` or `2`
- ★ This makes the page flushing become more aggressive before the end of a log file is reached.

The End

★ A few quick public service announcements if I may:

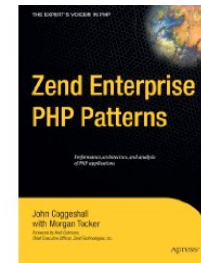
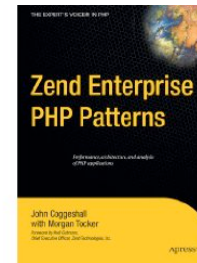
- ★ Many of the examples I use show up on our blog:

- ★ www.mysqlperformanceblog.com

- ★ We're a consulting company. Our most popular service is a Performance Audit. You might be interested:

- ★ www.mysqlperformanceblog.com/2008/11/24/how-percona-does-a-mysql-performance-audit/

- ★ I have two books to give away:



The EXPLAIN Command

```
mysql> EXPLAIN SELECT Name FROM Country WHERE continent =  
'Asia' AND population > 5000000 ORDER BY Name\G
```

```
***** 1. row *****  
      id: 1  
  select_type: SIMPLE  
    table: Country  
     type: ALL  
possible_keys: NULL  
      key: NULL  
   key_len: NULL  
      ref: NULL  
     rows: 239  
  Extra: Using where; Using filesort  
1 row in set (0.00 sec)
```

Explain (cont.)

```
mysql> ALTER TABLE Country ADD INDEX p (Population);
```

```
Query OK, 239 rows affected (0.01 sec)
```

```
Records: 239 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent =  
'Asia' AND population > 5000000 ORDER BY Name\G
```

```
***** 1. row *****  
      id: 1  
  select_type: SIMPLE  
    table: Country  
     type: ALL  
possible_keys: p  
      key: NULL  
  key_len: NULL  
   ref: NULL  
   rows: 239  
  Extra: Using where; Using filesort
```

```
1 row in set (0.06 sec)
```

Now it is...

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent = 'Asia'
AND population > 5000000 ORDER BY Name\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: Country
         type: range
possible_keys: p
          key: p
       key_len: 4
         ref: NULL
        rows: 54
   Extra: Using where; Using filesort
1 row in set (0.00 sec)
```

Another Index..

```
mysql> ALTER TABLE Country ADD INDEX c (Continent);
```

```
Query OK, 239 rows affected (0.01 sec)
```

```
Records: 239 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent = 'Asia'  
AND population > 50000000 ORDER BY Name\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: Country
```

```
type: ref
```

```
possible_keys: p,c
```

```
key: c
```

```
key_len: 1
```

```
ref: const
```

```
rows: 42
```

```
Extra: Using where; Using filesort
```

```
1 row in set (0.01 sec)
```

Changes back to p at 500M!

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent = 'Asia'  
AND population > 500000000 ORDER BY Name\G  
***** 1. row *****  
      id: 1  
  select_type: SIMPLE  
    table: Country  
      type: range  
possible_keys: p,c  
      key: p  
  key_len: 4  
    ref: NULL  
   rows: 4  
  Extra: Using where; Using filesort  
1 row in set (0.00 sec)
```

Try another index...

```
mysql> ALTER TABLE Country ADD INDEX p_c (Population, Continent);
Query OK, 239 rows affected (0.01 sec)
Records: 239 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent = 'Asia'
AND population > 50000000 ORDER BY Name\G
```

```
***** 1. row *****
```

```
    id: 1
  select_type: SIMPLE
    table: Country
    type: ref
possible_keys: p,c,p_c
    key: c
   key_len: 1
    ref: const
    rows: 42
  Extra: Using where; Using filesort
```

```
34 1 row in set (0.01 sec)
```

How about this one?

```
mysql> ALTER TABLE Country ADD INDEX c_p (Continent,Population);  
Query OK, 239 rows affected (0.01 sec)  
Records: 239 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent = 'Asia'  
AND population > 50000000 ORDER BY Name\G  
***** 1. row *****  
      id: 1  
  select_type: SIMPLE  
    table: Country  
     type: range  
possible_keys: p,c,p_c,c_p  
      key: c_p  
   key_len: 5  
     ref: NULL  
    rows: 7  
  Extra: Using where; Using filesort  
1 row in set (0.00 sec)
```

The Best...

```
mysql> ALTER TABLE Country ADD INDEX c_p_n  
      (Continent,Population,Name);  
Query OK, 239 rows affected (0.02 sec)  
Records: 239  Duplicates: 0  Warnings: 0
```

```
mysql> EXPLAIN SELECT Name FROM Country WHERE Continent = 'Asia'  
      AND population > 50000000 ORDER BY Name\G  
***** 1. row *****  
      id: 1  
      select_type: SIMPLE  
      table: Country  
      type: range  
      possible_keys: p,c,p_c,c_p,c_p_n  
      key: c_p_n  
      key_len: 5  
      ref: NULL  
      rows: 7  
      Extra: Using where; Using index; Using filesort  
36 1 row in set (0.00 sec)
```

So what's the end result?

- ★ We're looking at 9 rows, not the whole table.
 - ✦ We're returning those rows from the index - bypassing the table.
- ★ A simple example - but easy to demonstrate how to reduce table scans.
- ★ You wouldn't add all these indexes - I'm just doing it as a demonstration.
 - ✦ Indexes (generally) hurt write performance.

Example 2: Join Analysis

```
mysql> EXPLAIN SELECT * FROM city WHERE countrycode IN
(SELECT code FROM country WHERE name='Australia')\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
        table: city
         type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
         ref: NULL
        rows: 4079
   Extra: Using where
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
        table: country
         type: unique_subquery
possible_keys: PRIMARY
          key: PRIMARY
       key_len: 3
         ref: func
        rows: 1
   Extra: Using where
```

Join analysis (cont.)

```
mysql> EXPLAIN SELECT city.* FROM city, country WHERE
city.countrycode=country.code AND country.name='Australia'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: city
         type: ALL
possible_keys: NULL
          key: NULL
     key_len: NULL
         ref: NULL
        rows: 4079
      Extra:
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: country
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
     key_len: 3
         ref: world.city.CountryCode
        rows: 1
      Extra: Using where
```

Try an index...

- ★ `mysql> ALTER TABLE city ADD INDEX (countrycode);`
Query OK, 4079 rows affected (0.03 sec)
Records: 4079 Duplicates: 0 Warnings: 0

Is that any better?

```
mysql> EXPLAIN SELECT city.* FROM city, country WHERE city.countrycode=country.code  
AND country.name='Australia'\G
```

```
***** 1. row *****
```

```
    id: 1  
  select_type: SIMPLE  
    table: city  
    type: ALL  
possible_keys: CountryCode  
    key: NULL  
  key_len: NULL  
    ref: NULL  
   rows: 4079  
  Extra:
```

```
***** 2. row *****
```

```
    id: 1  
  select_type: SIMPLE  
    table: country  
    type: eq_ref  
possible_keys: PRIMARY  
    key: PRIMARY  
  key_len: 3  
    ref: world.city.CountryCode  
   rows: 1  
  Extra: Using where
```

```
41 2 rows in set (0.01 sec)
```

Try Again

- ★ `mysql> ALTER TABLE country ADD INDEX (name);`
Query OK, 239 rows affected (0.01 sec)
Records: 239 Duplicates: 0 Warnings: 0

Looking good...

```
mysql> EXPLAIN SELECT city.* FROM city, country WHERE city.countrycode=country.code  
AND country.name='Australia'\G
```

```
***** 1. row *****
```

```
    id: 1  
  select_type: SIMPLE  
    table: country  
    type: ref  
possible_keys: PRIMARY,Name  
    key: Name  
   key_len: 52  
    ref: const  
   rows: 1  
  Extra: Using where
```

```
***** 2. row *****
```

```
    id: 1  
  select_type: SIMPLE  
    table: city  
    type: ref  
possible_keys: CountryCode  
    key: CountryCode  
   key_len: 3  
    ref: world.country.Code  
   rows: 18  
  Extra:
```

```
43 2 rows in set (0.00 sec)
```

My Advice

- ★ Focus on components of the WHERE clause.
- ★ The optimizer does cool things - don't make assumptions.
For Example:

- ★ `EXPLAIN SELECT * FROM City WHERE id = 1810;`

- ★ `EXPLAIN SELECT * FROM City WHERE id = 1810
LIMIT 1;`

- ★ `EXPLAIN SELECT * FROM City WHERE id BETWEEN 100
and 200;`

- ★ `EXPLAIN SELECT * FROM City WHERE id >= 100 and
id <= 200;`

The answer...

```
mysql> EXPLAIN SELECT * FROM City WHERE id = 1810;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	const	PRIMARY	PRIMARY	4	const	1	

```
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN SELECT * FROM City WHERE id = 1810 LIMIT 1;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	const	PRIMARY	PRIMARY	4	const	1	

```
1 row in set (0.00 sec)
```

The answer (2)

```
mysql> EXPLAIN SELECT * FROM City WHERE id BETWEEN 100 and 200;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key      | key_len | ref  | rows | Extra      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  1 | SIMPLE      | City  | range | PRIMARY      | PRIMARY  | 4       | NULL | 101 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

```
mysql> EXPLAIN SELECT * FROM City WHERE id >= 100 and id <= 200;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key      | key_len | ref  | rows | Extra      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  1 | SIMPLE      | City  | range | PRIMARY      | PRIMARY  | 4       | NULL | 101 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```