



# MySQL Performance Tuning



























Boros Péter

Rendszermérnök

# Virgo Systems

- Nagy teljesítményű web alapú rendszerek
- A seven24 az üzemeltetési részleg
  - Performance tuning
    - Unix(like) (Linux, Solaris)
    - DB (MySQL, Oracle)
  - Teljes infrastruktúra tervezés, üzemeltetés

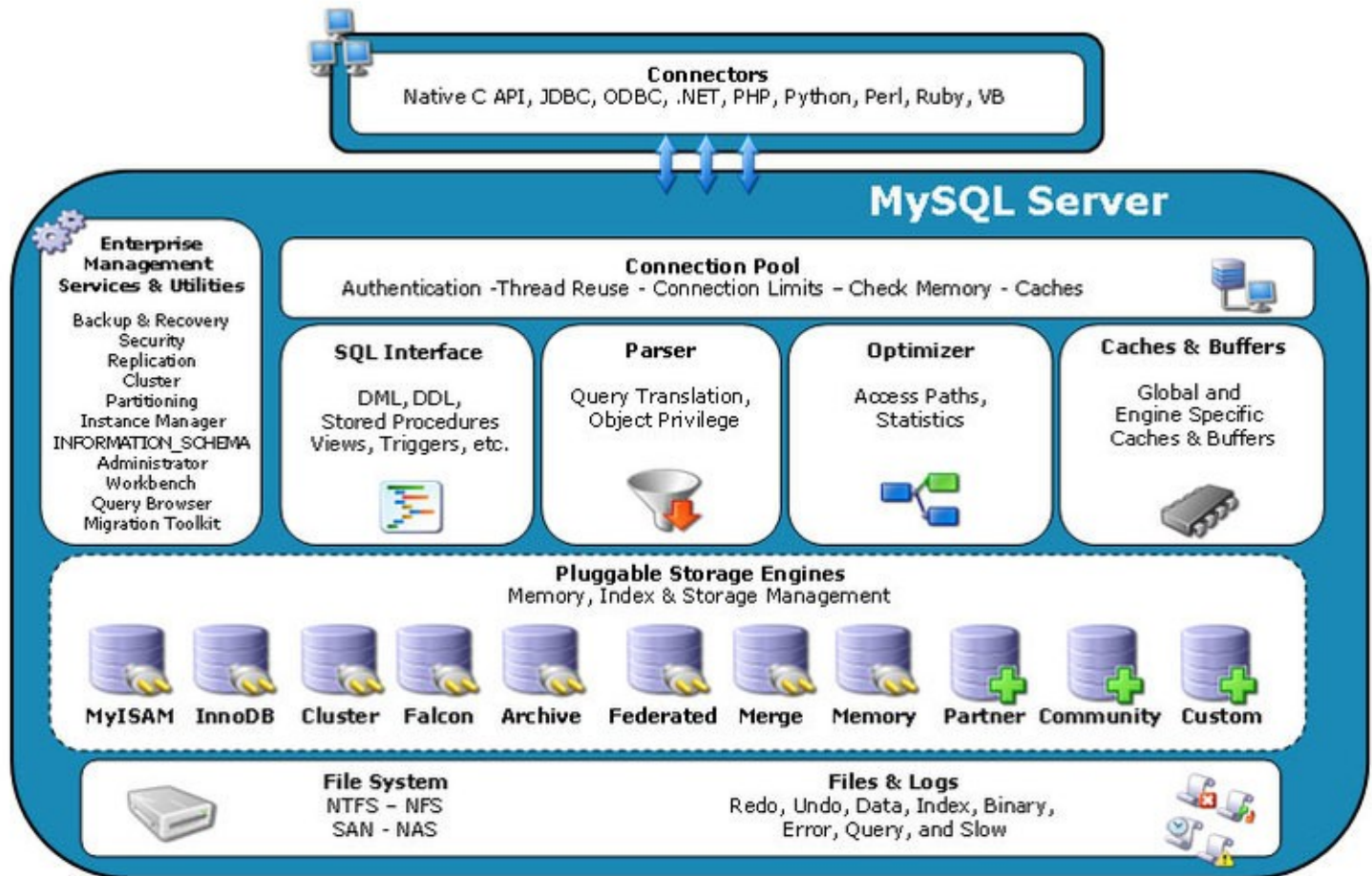
# Sun és Open Source

<b>Database Platform</b>	    
<b>Application Infrastructure</b>	     
<b>Virtualization</b>	 
<b>Operating System</b>	    
<b>Partners</b>	    
<b>Architecture</b>	  

# MySQL ügyfelek



# MySQL architektúra



# Hol lehet/érdeemes?

- MySQL szerver konfiguráció
- Adatbázis séma szint
- Lekérdezések/DML-ek

# MySQL szerver konfiguráció

- Az alapértemezett beállítások fejlesztői gépre jók
- MySQL disztribúciók szállítanak előre gyártott konfigurációkat (jó kiindulási alapok, pl. my.cnf-huge)
- Általában séma/query szintű tuninggal többet lehet nyerni

# Konfiguráció hogyan

- Nem storage engine specifikus
  - Sort buffer méret
  - Query cache
  - Max connections
- MyISAM specifikus
  - Key buffer méret

# Konfiguráció hogyan II.

- InnoDB specifikus
  - InnoDB buffer cache
  - InnoDB log méret
  - InnoDB log buffer méret
  - DirectIO
  - Flush method
  - ...
- InnoDB-t tuningolni kell, tuning után tipikusan 10-50-szeres sebességnövekedés

# Adatbázis séma szint

- A normalizáláson túl storage engine specifikus
- Nagyon nagy téma, InnoDB esetén fogjuk megnézni hogy működik a primary key és hogy lehet optimálisan használni
- Nagyon sokan elrontják és sokat lehet vele nyerni

# InnoDB primary key

- Az adatfile clustering primary key alapján történik
- A primary key alapján rendezve kérhetőek le rekordok
- Tervezési szempontok
  - A primary key legyen narrow index
  - A nagy terhelést jelentő lekérdezések primary key alapján történjenek

# Primary Key bővítés

- Primary key-t mindig érdemes használni, unique indexnek kell lennie
- Érdemes új oszlopot is bevezetni, hogy tudjon unique index lenni
- BTREE indexeknek tetszőleges bal oldali prefixe használható lekérdezéseknél

# Primary Key bővítés II.

- Példa: rossz adatbázis, az `AUTO_INCREMENT` primary key, amit nem használunk

```
CREATE TABLE ki_kit (  
  id int NOT NULL AUTO_INCREMENT,  
  a int NOT NULL,  
  b int NOT NULL,  
  nemerdekes VARCHAR(255),  
  ezsemerdekes VARCHAR(255),  
  PRIMARY KEY (id)  
);
```

# Primary Key bővítés III.

- Pl. a 'b'-re vagyunk kíváncsiak a leggyakrabban

```
CREATE TABLE ki_kit (  
    id int NOT NULL,  
    a int NOT NULL,  
    b int NOT NULL,  
    nemerdekes VARCHAR(255),  
    ezsemerdekes VARCHAR(255),  
    PRIMARY KEY (b,id)  
);
```

# Primary Key szűkítés

- Példa: távközlési szolgáltató adatbázisa a napi lila hívásokról

```
CREATE TABLE nap_telefonszam_lila (  
    nap date NOT NULL,  
    telefonszam int NOT NULL,  
    hivas int NOT NULL,  
    idotaram int NOT NULL,  
    lilae tinyint UNSIGNED NOT NULL,  
    nemerdekes VARCHAR(255),  
    ezsemerdekes VARCHAR(255),  
    PRIMARY KEY (nap, telefonszam, lilae)  
);
```

# Primary Key szűkítés II.

## ■ Módosítás után

```
CREATE TABLE nap_telefonszam (  
    nap date NOT NULL,  
    telefonszam int NOT NULL,  
    hivas int NOT NULL,  
    idotaram int NOT NULL,  
    lilahivas int,  
    lilaidotartam int,  
    nemerdekes VARCHAR(255),  
    ezsemerdekes VARCHAR(255),  
    PRIMARY KEY (nap, telefonszam)  
);
```

# Primary Key szűkítés III.

- Logikailag ugyanazt az adatot tároljuk, fizikailag nem
- Az ugyanarra a telefonszámra érkező lila és nem lila hívásokat ugyanabban a rekordban tároljuk
- A tábla kevesebb rekordot fog tartalmazni, és az index is kisebb lesz

# Query szint

- Explain és explain partitions a barátunk
- A primary key további előnyös tulajdonságai
- BTREE és HASH indexek használhatósága

# Primary Key és LIMIT

- `SELECT * FROM t WHERE a > 3 ORDER BY a LIMIT 100;`
  - Fogja az összes sort, ahol  $a > 3$ , majd az első 100-on kívül a többi eldobja (kivéve ha a primary key)
- `SELECT * FROM t where a > 3 LIMIT 10 OFFSET 1000;`
  - Hasonló a helyzet, helyette BETWEEN-es lekérdezés

# HASH Indexek használhatósága

- Nem használhatók a prefixei
- Főleg in-memory adatbázisoknál használják
- Alacsony kardinalitású indexnél (kevés különböző érték) rossz
  - A különböző elemek mellett tárolja a hash értékeket update/delete lassú lesz

# BTREE Indexek használhatósága

- Bármelyik prefixe használható
- Diszkes adatbázisokra találták ki
  - a lemezen kell seekelni, ahogy egyre beljebb megyünk a fában, az érdekes adatokat majdnem szekvenciálisan lehet olvasni

# BTREE Indexek használhatósága II.

- (A,B) index adott
- `SELECT * FROM t WHERE a=2; --OK`
- `SELECT * FROM t WHERE b=3; --NEM OK`
  - Nem prefix
- `SELECT * FROM t WHERE a=2 AND b=3; --OK`
- `SELECT * FROM t WHERE a=2 OR b=2; --NEM OK`
  - Külön index kell A-ra és B-re
- `SELECT * FROM t WHERE a=2 ORDER BY b;`
  - A rendezéshez nem jó
- `SELECT * FROM t WHERE a=2 ORDER BY a;`
  - A rendezéshez is jó

# Partícionálás

- MySQL 5.1-ben csak horizontális partícionálás (tábla és index)
- Az egyes részeket a MySQL külön tárolja
- A részekre bontást a partícionálási logika bontja, és ez mondja meg, hogy melyik adat melyik partíción van

# Partícionálás II.

- Típusok
  - Key
  - Hash
  - Value
- Akkor előnyös ha egy adott query csak 1 vagy kevés partíciót érint.

# Particionálás III.

```
mysql> show create table  
t;
```

```
| Table | Create Table  
| t     | CREATE TABLE `t`  
(  
  `a` int(11) DEFAULT  
NULL,  
  `b` int(11) DEFAULT  
NULL,  
  `c` int(11) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT  
CHARSET=utf8 |
```

```
mysql> select count(*)  
from t;
```

```
+-----+  
| count(*) |  
+-----+  
| 3000000 |  
+-----+  
1 row in set (1.15 sec)
```

```
mysql> select * from t  
where a=121212;
```

```
| a      | b      | c  
| 121212 | 121212 | 121212  
1 row in set (1.31 sec)
```

# Particionálás III.

```
mysql> alter table t
partition by hash(a)
partitions 100;
```

```
Query OK, 3000000 rows
affected (34.07 sec)
```

```
Records: 3000000
```

```
Duplicates: 0 Warnings: 0
```

```
mysql> select * from t
where a=121212;
```

```
| a          | b          | c
| 121212    | 121212    | 121212
1 row in set (0.00 sec)
```

```
mysql> select * from t
where b=121212;
```

```
| a          | b          | c
| 121212    | 121212    | 121212
1 row in set (1.36 sec)
```

# Hangolás témában olvasnivaló

- [mysqlperformanceblog.com](http://mysqlperformanceblog.com)
- Percona nevű cég írja  
([percona.com](http://percona.com))
- MySQL-hez egyedi patch-ek,  
profiling eszközök



# MySQL újdonosságok

# Falcon Storage Engine

- Eredetileg a Netfrastructure fejlesztette
- Következő generációs tranzakcionális adatbázismotor
- MVCC (MultiVersion Concurrency Control)
- Crash recovery
- Hatékonyabb memóriakezelés
- No tuning (ez máris megoldótni látszik)
- Jó integráltság MySQL-lel
- PERFORMANCE\_SCHEMA

# Új backup engine

- MySQL 6.0-ban
- SQL parancsokkal vezérelhető
- Online, nem blokkoló DML, tranzakcionális storage engine esetén
- Visszaállítás adott tranzakcióra
- Plugin-ek fejleszhetők hozzá (pl. natív online MyISAM backup)

# DTrace támogatás MySQL 6-hoz

- Alkalmazás szintű probe-ok
- A query melyik részével telik el sok idő?
  - A parseolással?
  - A rendezéssel?
  - A sorok kiválasztásával?
- MySQL belső állapotának megfigyelhetősége



```
SELECT  
REVERSE('uh.ogriv@retep.sorob');
```