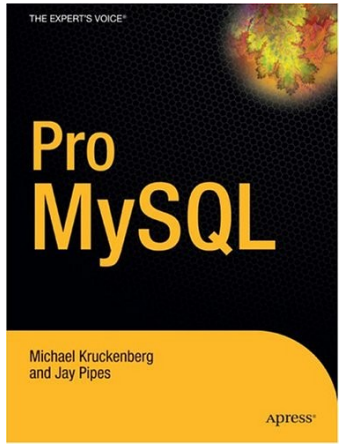
Performance-minded MySQL for the PHP Developer

Jay Pipes
Community Relations Manager, North America, MySQL
jay@mysql.com -- http://jpipes.com

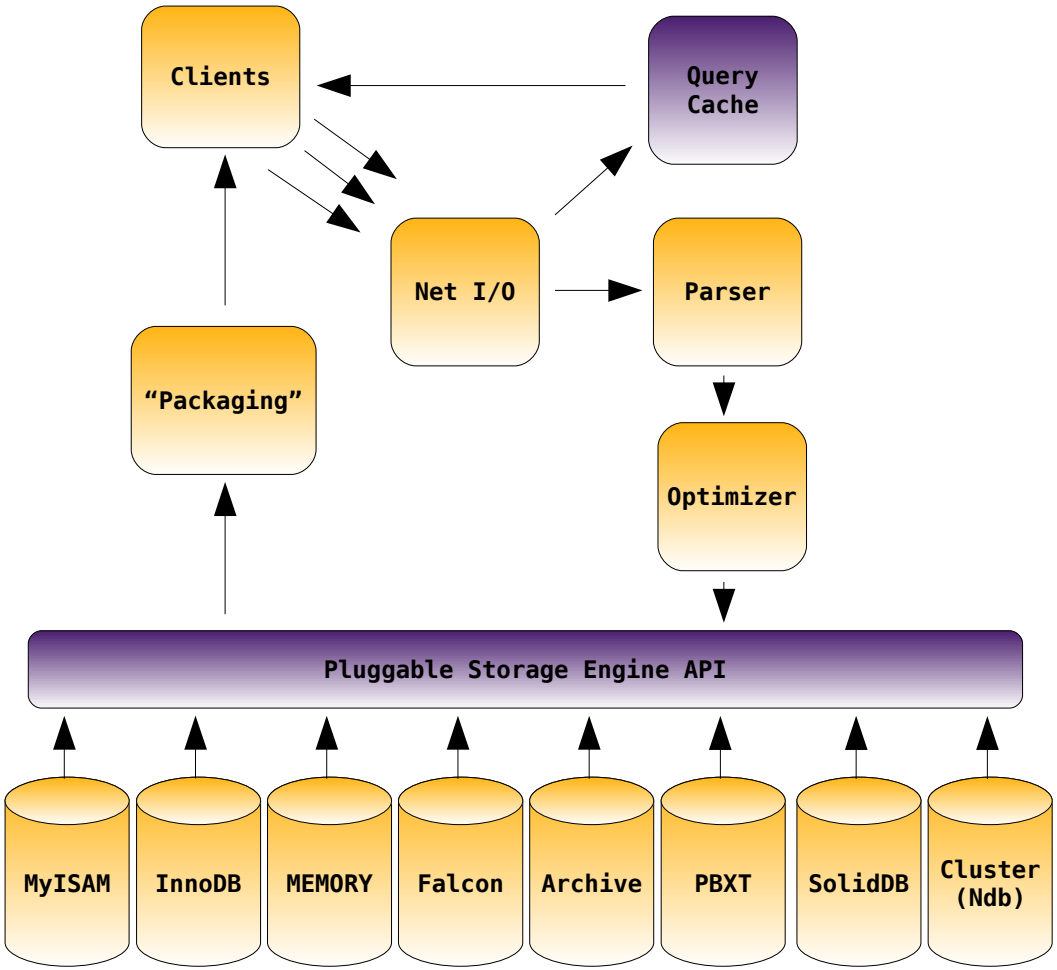
before we start...

- Who am I?
  - Just some dude who works at MySQL (eh...Sun)
  - Oh, I co-wrote a book on MySQL
  - Active PHP/MySQL community member
  - Other than that, semi-normal geek, married, 2 dogs, 2 cats, blah blah



- How many using...
  - PHP4? PHP5? PDO? mysqli? mysqlnd?
  - SQL Server? PostgreSQL? Oracle? Replication?
- This talk is about how to code your app to get the best performance out of your MySQL server

system architecture of MySQL



- Highly coupled subsystems
- Emphasis on *connection*-based memory allocation (as opposed to global)
- Caching on many different levels
- Storage engine layer is both blessing and curse
- Optimizer is cost-based, simplistic, and must be guarded against
- Efforts to modularize ongoing

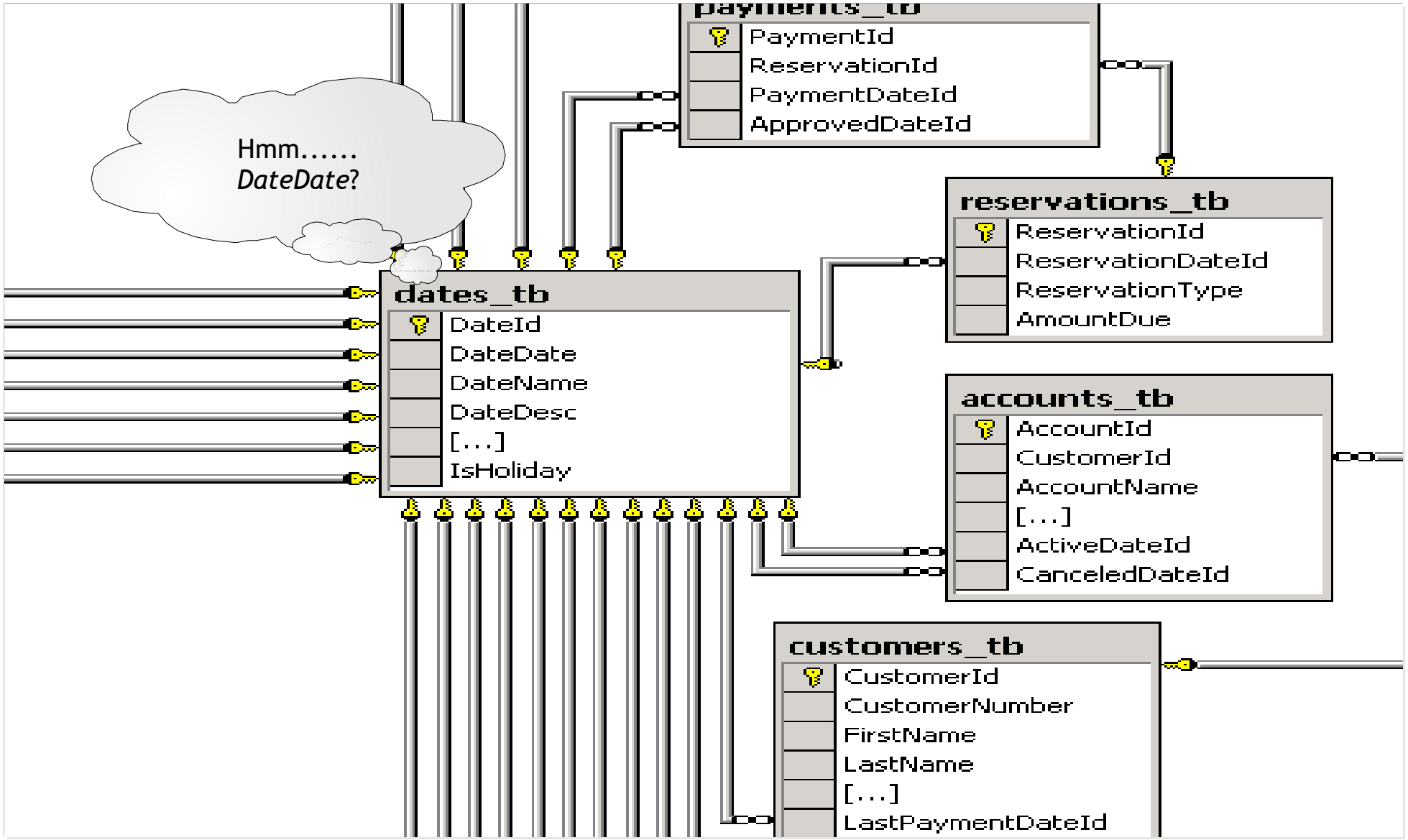
the schema

- Basic foundation of performance
- Normalize first, de-normalize later
- Smaller, smaller, smaller
- Divide and conquer
- Understand benefits *and disadvantages* of different storage engines

The Leaning Tower of Pisa
from Wikipedia:

"Although intended to stand vertically, the tower began leaning to the southeast soon after the onset of construction in 1173 ***due to a poorly laid foundation*** and loose substrate that has allowed the foundation to shift direction."

taking normalization *way* too far



http://thedailywtf.com/forums/thread/75982.aspx

smaller, smaller, smaller

The Pygmy Marmoset
world's smallest monkey

This picture is a cheap stunt intended to induce kind feelings for the presenter.

Oh, and I *totally* want one of these guys for a pet.

The more records you can fit into a single page of memory/disk, the faster your seeks and scans will be

- Do you *really* need that `BIGINT`?
- Use `INT UNSIGNED` for IP addresses
- Use `VARCHAR` carefully
  - Converted to `CHAR` when used in a temporary table
- Use `TEXT` sparingly
  - Consider separate tables
- Use `BLOB`s very sparingly
  - Use the filesystem for what it was intended

handling IPv4 addresses

```
CREATE TABLE Sessions (
  session_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, ip_address INT UNSIGNED NOT NULL // Compare to CHAR(15)...
, session_data TEXT NOT NULL
, PRIMARY KEY (session_id)
, INDEX (ip_address)
) ENGINE=InnoDB;
```

```
// Insert a new dummy record
INSERT INTO Sessions VALUES
(NULL, INET_ATON('192.168.0.2'), 'some session data');
```

```
INSERT INTO Session VALUES (NULL, 3232235522, 'some session data');
```

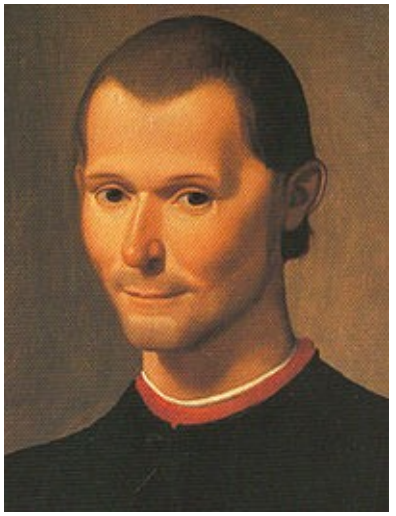
```
// Find all sessions coming from a local subnet
SELECT
  session_id
, ip_address as ip_raw
, INET_NTOA(ip_address) as ip
, session_data
FROM Sessions
WHERE ip_address
BETWEEN INET_ATON('192.168.0.1')
AND INET_ATON('192.168.0.255');
```

```
WHERE ip_address BETWEEN 3232235521 AND 3232235775
```

```
mysql> SELECT session_id, ip_address as ip_raw, INET_NTOA(ip_address) as ip, session_data
    -> FROM Sessions
    -> WHERE ip_address BETWEEN
    -> INET_ATON('192.168.0.1') AND INET_ATON('192.168.0.255');
+------------+------------+-------------+-------------------+
| session_id | ip_raw     | ip          | session_data      |
+------------+------------+-------------+-------------------+
|          1 | 3232235522 | 192.168.0.2 | some session data |
+------------+------------+-------------+-------------------+
```

divide et impera

- Vertical partitioning
  - Split tables with many columns into multiple tables
- Horizontal partitioning
  - Split table with many rows into multiple tables
- Both are important for different reasons
- Partitioning in MySQL 5.1 is transparent horizontal partitioning *within the DB*

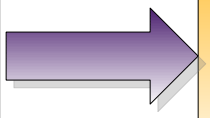


Niccolò Machiavelli
The Art of War, (1519-1520):

"A Captain ought, among all the other actions of his, endeavor **with every art to divide the forces of the enemy**, either by making him suspicious of his men in whom he trusted, or by giving him cause that he has to separate his forces, **and, because of this, become weaker**."

vertical partitioning #1

```
CREATE TABLE Users (
  user_id INT NOT NULL AUTO_INCREMENT
, email VARCHAR(80) NOT NULL
, display_name VARCHAR(50) NOT NULL
, password CHAR(41) NOT NULL
, first_name VARCHAR(25) NOT NULL
, last_name VARCHAR(25) NOT NULL
, address VARCHAR(80) NOT NULL
, city VARCHAR(30) NOT NULL
, province CHAR(2) NOT NULL
, postcode CHAR(7) NOT NULL
, interests TEXT NULL
, bio TEXT NULL
, signature TEXT NULL
, skills TEXT NULL
, PRIMARY KEY (user_id)
, UNIQUE INDEX (email)
) ENGINE=InnoDB;
```

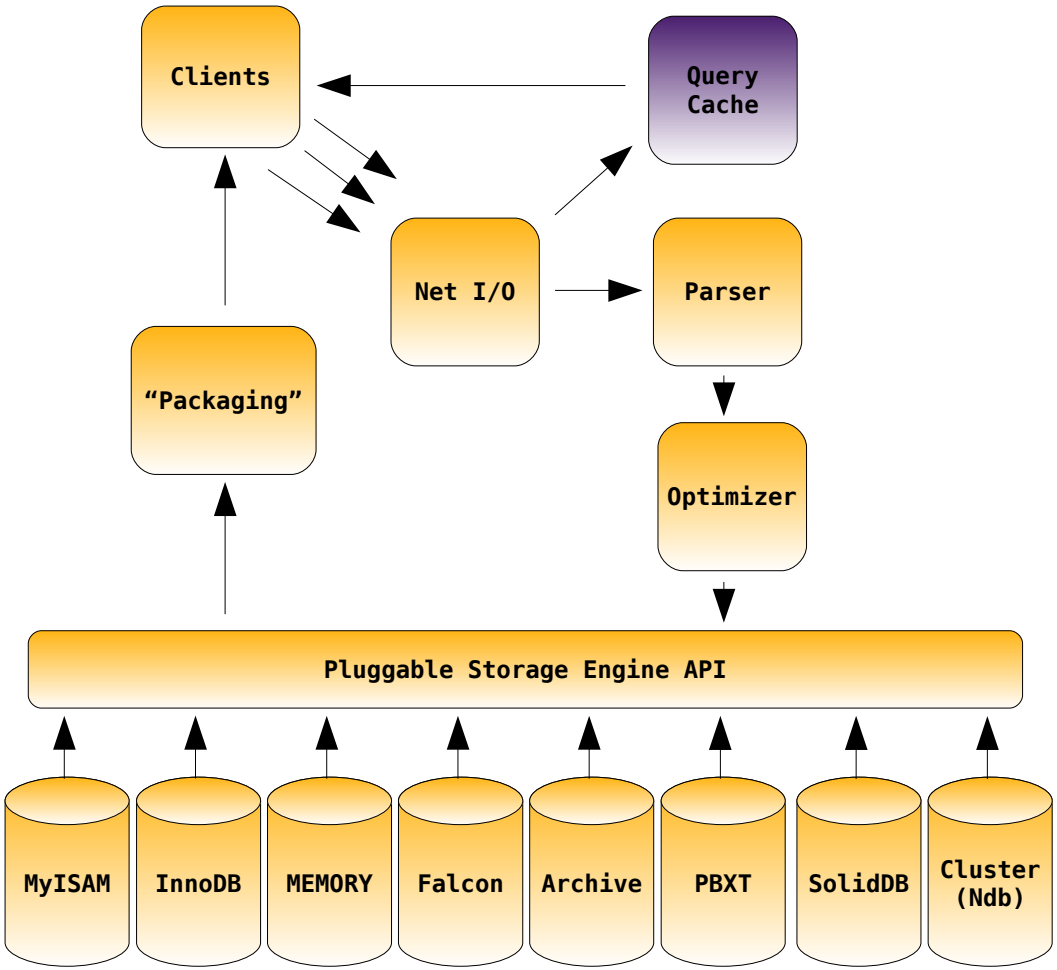


```
CREATE TABLE Users (
  user_id INT NOT NULL AUTO_INCREMENT
, email VARCHAR(80) NOT NULL
, display_name VARCHAR(50) NOT NULL
, password CHAR(41) NOT NULL
, PRIMARY KEY (user_id)
, UNIQUE INDEX (email)
) ENGINE=InnoDB;
```

```
CREATE TABLE UserExtra (
  user_id INT NOT NULL
, first_name VARCHAR(25) NOT NULL
, last_name VARCHAR(25) NOT NULL
, address VARCHAR(80) NOT NULL
, city VARCHAR(30) NOT NULL
, province CHAR(2) NOT NULL
, postcode CHAR(7) NOT NULL
, interests TEXT NULL
, bio TEXT NULL
, signature TEXT NULL
, skills TEXT NULL
, PRIMARY KEY (user_id)
) ENGINE=InnoDB;
```

- "Extra" columns are mostly NULL ***or*** infrequently accessed?
- Space in buffer pool is at a premium?
  - Splitting the table allows main records to consume the buffer pages without the extra data taking up space in memory
- Need FULLTEXT on your text columns?

the MySQL query cache



- *You* must understand application read/write patterns
- Internal query cache design is a compromise between CPU usage and read performance
- Stores the MYSQL_RESULT of a SELECT along with a hash of the SELECT SQL statement
- *Any modification* to *any* table involved in the SELECT invalidates the stored result
- Write applications to be aware of the query cache

vertical partitioning #2

```
CREATE TABLE Products (
  product_id INT NOT NULL
, name VARCHAR(80) NOT NULL
, unit_cost DECIMAL(7,2) NOT NULL
, description TEXT NULL
, image_path TEXT NULL
, num_views INT UNSIGNED NOT NULL
, num_in_stock INT UNSIGNED NOT NULL
, num_on_order INT UNSIGNED NOT NULL
, PRIMARY KEY (product_id)
, INDEX (name(20))
) ENGINE=InnoDB;

// Getting a simple COUNT of products
// easy on MyISAM, terrible on InnoDB
SELECT COUNT(*)
FROM Products;
```

```
CREATE TABLE Products (
  product_id INT NOT NULL
, name VARCHAR(80) NOT NULL
, unit_cost DECIMAL(7,2) NOT NULL
, description TEXT NULL
, image_path TEXT NULL
, PRIMARY KEY (product_id)
, INDEX (name(20))
) ENGINE=InnoDB;
```

```
CREATE TABLE ProductCounts (
  product_id INT NOT NULL
, num_views INT UNSIGNED NOT NULL
, num_in_stock INT UNSIGNED NOT NULL
, num_on_order INT UNSIGNED NOT NULL
, PRIMARY KEY (product_id)
) ENGINE=InnoDB;
```

```
CREATE TABLE TableCounts (
  total_products INT UNSIGNED NOT NULL
) ENGINE=MEMORY;
```

- Mixing static attributes with frequently updated fields in a single table?
  - Thrashing occurs with query cache. Each time an update occurs on any record in the table, all queries referencing the table are invalidated in the query cache
- Doing COUNT(*) with no WHERE on an indexed field on an InnoDB table?
  - Complications with versioning make full table counts very slow

indexes - your schema's phone book

- Speed up SELECTs, but slow down modifications
- Ensure indexes on columns used in `WHERE`, `ON`, `GROUP BY` clauses
- Always ensure `JOIN` conditions are indexed (and have identical data types)
- Be careful of the column order
- Look for covering indexes
  - Occurs when all fields in one table needed by a SELECT are available in an index record

The Yellow Pages
from *Wikipedia*:

"The name and concept of "Yellow Pages" came about in 1883, when a printer in Cheyenne, Wyoming working on a regular telephone directory ran out of white paper and used yellow paper instead"

selectivity - the key to good, erm...keys

- Selectivity
  - % of distinct values in a column
  - *S=d/n*
  - Unique/primary always 1.0
- If column has a low selectivity
  - It may still be put in a multi-column index
    - As a prefix?
    - As a suffix?
    - Depends on the application

remove crappy or redundant indexes

```
SELECT
  t.TABLE_SCHEMA AS `db`, t.TABLE_NAME AS `table`, s.INDEX_NAME AS `index name`
 , s.COLUMN_NAME AS `field name`, s.SEQ_IN_INDEX `seq in index`, s2.max_columns AS `# cols`
 , s.CARDINALITY AS `card`, t.TABLE_ROWS AS `est rows`
 , ROUND(((s.CARDINALITY / IFNULL(t.TABLE_ROWS, 0.01)) * 100), 2) AS `sel %`
FROM INFORMATION_SCHEMA.STATISTICS s
 INNER JOIN INFORMATION_SCHEMA.TABLES t
  ON s.TABLE_SCHEMA = t.TABLE_SCHEMA AND s.TABLE_NAME = t.TABLE_NAME
 INNER JOIN (
  SELECT TABLE_SCHEMA, TABLE_NAME, INDEX_NAME, MAX(SEQ_IN_INDEX) AS max_columns
  FROM INFORMATION_SCHEMA.STATISTICS WHERE TABLE_SCHEMA != 'mysql'
  GROUP BY TABLE_SCHEMA, TABLE_NAME, INDEX_NAME
 ) AS s2
 ON s.TABLE_SCHEMA = s2.TABLE_SCHEMA AND s.TABLE_NAME = s2.TABLE_NAME AND s.INDEX_NAME = s2.INDEX_NAME
WHERE t.TABLE_SCHEMA != 'mysql'       /* Filter out the mysql system DB */
AND t.TABLE_ROWS > 10                 /* Only tables with some rows */
AND s.CARDINALITY IS NOT NULL         /* Need at least one non-NULL value in the field */
AND (s.CARDINALITY / IFNULL(t.TABLE_ROWS, 0.01)) < 1.00 /* unique indexes are perfect anyway */
ORDER BY `sel %`, s.TABLE_SCHEMA, s.TABLE_NAME          /* DESC for best non-unique indexes */
LIMIT 10;
```

TABLE_SCHEMA	TABLE_NAME	INDEX_NAME	COLUMN_NAME	SEQ_IN_INDEX	COLS_IN_INDEX	CARD	ROWS	SEL %
worklog	**amendments**	**text**	**text**	**1**	**1**	**1**	**33794**	**0.00**
planetmysql	entries	categories	categories	1	3	1	4171	0.02
planetmysql	entries	categories	title	2	3	1	4171	0.02
planetmysql	entries	categories	content	3	3	1	4171	0.02
sakila	inventory	idx_store_id_film_id	store_id	1	2	1	4673	0.02
sakila	rental	idx_fk_staff_id	staff_id	1	1	3	16291	0.02
worklog	tasks	title	title	1	2	1	3567	0.03
worklog	tasks	title	description	2	2	1	3567	0.03
sakila	payment	idx_fk_staff_id	staff_id	1	1	6	15422	0.04
mysqlforge	mw_recentchanges	rc_ip	rc_ip	1	1	2	996	0.20

http://forge.mysql.com/snippets/view.php?id=85

effects of column order on indexing

```
CREATE TABLE Tag2Project (
tag INT UNSIGNED NOT NULL
, project INT UNSIGNED NOT NULL
, PRIMARY KEY (tag, project)
) ENGINE=MyISAM;
```

```
mysql> EXPLAIN SELECT project, COUNT(*) as num_tags
    -> FROM Tag2Project
    -> GROUP BY project;
+-------------+-------+---------+---------------------------------------------+
| table       | type  | key     | Extra                                       |
+-------------+-------+---------+---------------------------------------------+
| Tag2Project | index | PRIMARY | Using index; Using temporary; Using filesort |
+-------------+-------+---------+---------------------------------------------+
```

- Filesorts occur when the optimizer does not have a list of values in the order it needs
  - You will see increases in status counters `Created_tmp_tables` and `Created_tmp_disk_tables`

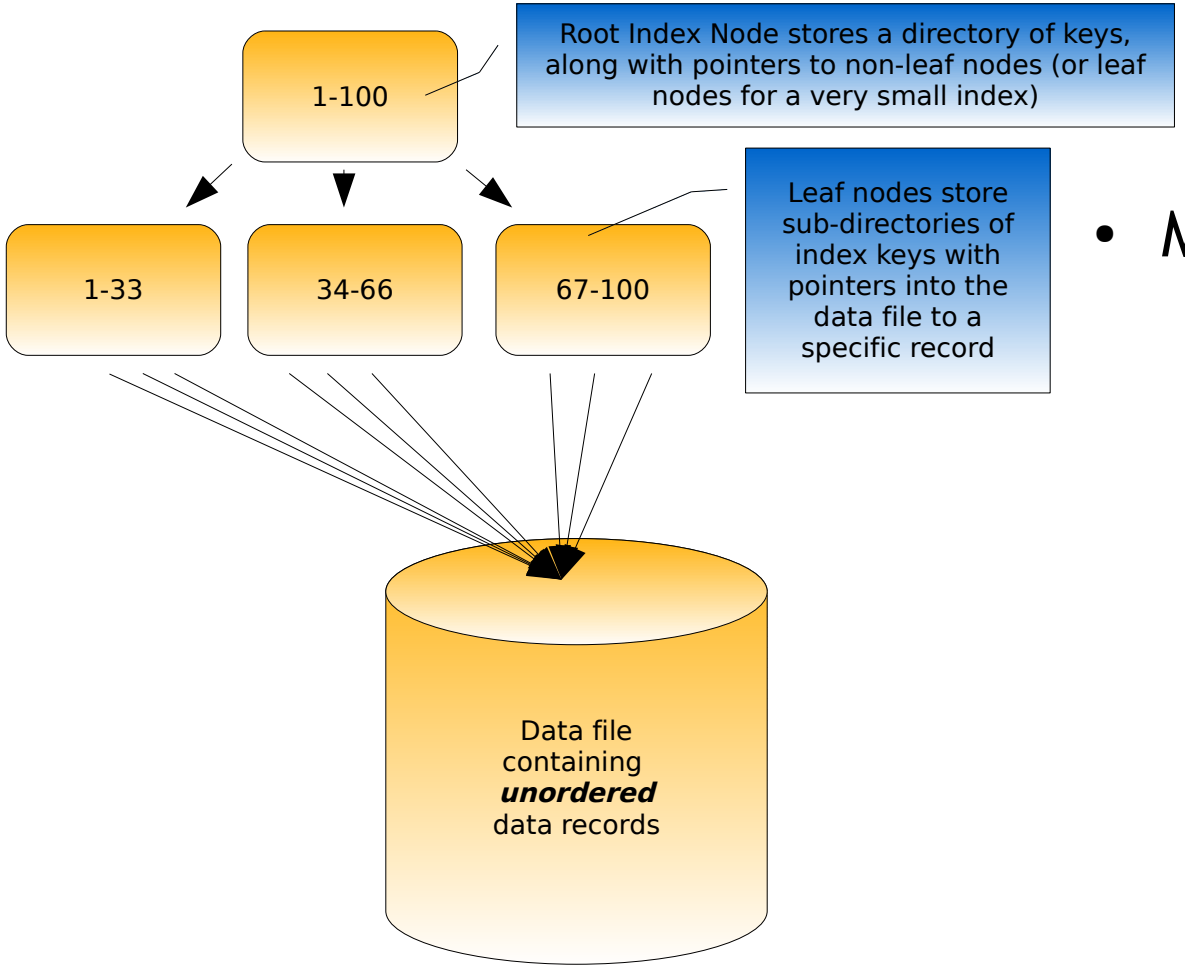
```
mysql> CREATE INDEX project ON Tag2Project (project);
Query OK, 701 rows affected (0.01 sec)
Records: 701  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT project, COUNT(*) as num_tags
    -> FROM Tag2Project
    -> GROUP BY project;
+-------------+-------+---------+-------------+
| table       | type  | key     | Extra       |
+-------------+-------+---------+-------------+
| Tag2Project | index | project | Using index |
+-------------+-------+---------+-------------+
```

- Eliminate the filesort by providing the optimizer with a sorted list of values needed in the query

covering indexes

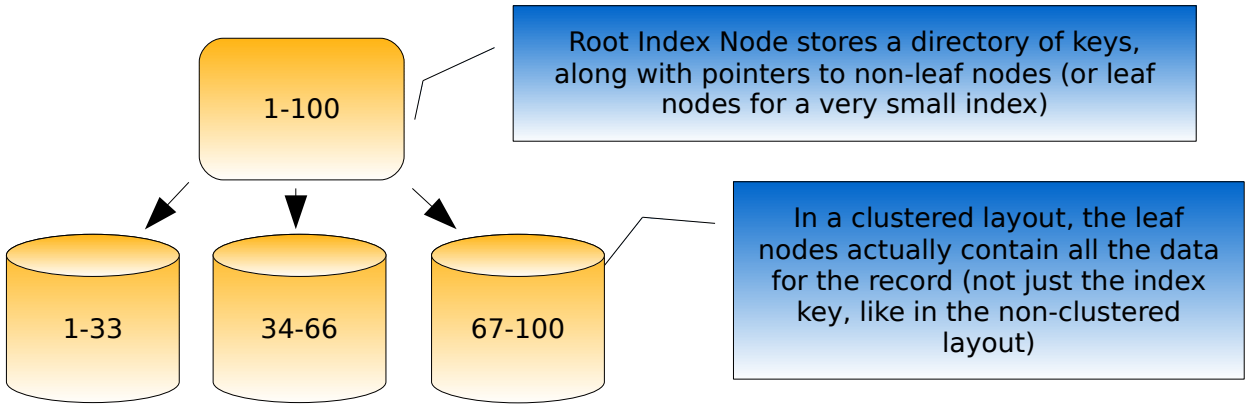
- When all columns needed from a single table for a SELECT are available in the index
  - No need to grab the rest of the columns from the data (file or page)
    - Known as a *bookmark lookup* operation
- Extra column in EXPLAIN SELECT shows "Using index"
- Important to know the data to index organization of the storage engine

non-clustered data and index layout (MyISAM)



- MyISAM's non-clustered layout
  - .MYD file contains records (unordered)
  - .MYI file contains all indexes
    - Index records contain pointer to where rest of data for the record can be found in .MYD file

clustered data and index layout (InnoDB)



- Organized in 16KB data pages
  - Data page contains the leaf node of the B+tree primary key index
  - Each data page has a small dictionary at the end of the page which stores an ordered list of primary key values and pointers into the page
- No bookmark lookup operation is needed for ***primary key lookups***
- Very important to have as small a clustering key (primary key) as possible
  - Every secondary index built on the table will have the primary key appended to each index record
- If you don't pick a primary key (bad idea!), one will be created for you

pop quiz

```
mysql> SHOW INDEX FROM ProjectDailyStats;
+------------------+------------+----------------+--------------+-------------+-----------+-------------+
| Table            | Non_unique | Key_name       | Seq_in_index | Column_name | Collation | Cardinality |
+------------------+------------+----------------+--------------+-------------+-----------+-------------+
| ProjectDailyStats |          0 | PRIMARY        |            1 | stat_date   | A         |        1081 |
| ProjectDailyStats |          0 | PRIMARY        |            2 | project     | A         |       88644 |
| ProjectDailyStats |          1 | ix_project     |            1 | project     | A         |         464 |
| ProjectDailyStats |          1 | ix_total_views |            1 | total_views | A         |          54 |
+------------------+------------+----------------+--------------+-------------+-----------+-------------+
```

```
mysql> EXPLAIN SELECT stat_date, project, AVG(total_views)
    -> FROM ProjectDailyStats
    -> WHERE total_views > 100
    -> GROUP BY stat_date, project\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: ProjectDailyStats
         type: range
possible_keys: ix_total_views
          key: ix_total_views
      key_len: 4
          ref: NULL
         rows: 23
        Extra: Using where; Using index; Using temporary; Using filesort
```

#1: Why is the `ix_total_views` *index a covering index?*

#2: Why is a ***filesort*** *still needed?*

coding like a join-fu master

Did you know?
from *Wikipedia:*

Join-fu is a close cousin to **Jun Fan Gung Fu**, the method of martial arts Bruce Lee began teaching in 1959.

(Slightly altered)

- Building upon the foundation of the schema
- Do not think in terms of iterators, for loops, while loops, etc
- Instead, think in terms of sets
- Break complex SQL statements (or business requests) into smaller, manageable chunks

set-wise thinking

"Show the last payment information for each customer"

Many programmers think:

OK, ***for each*** customer, find the maximum date the payment was made get that payment record(s)

Think instead:

OK, I have 2 sets of data here. One set of last payments dates and another set containing payment information (so, how do I join these sets?)

different execution plans

```
mysql> EXPLAIN SELECT
    ->   p.*
    -> FROM payment p
    -> WHERE p.payment_date =
    -> ( SELECT MAX(payment_date)
    ->   FROM payment
    ->   WHERE customer_id=p.customer_id);
+--------------------+---------+------+---------------------------------+--------------+---------------+-------+-------------+
| select_type        | table   | type | possible_keys                   | key          | ref           | rows  | Extra       |
+--------------------+---------+------+---------------------------------+--------------+---------------+-------+-------------+
| PRIMARY            | p       | ALL  | NULL                            | NULL         | NULL          | 16451 | Using where |
| DEPENDENT SUBQUERY | payment | ref  | idx_fk_customer_id,payment_date | payment_date | p.customer_id |    12 | Using index |
+--------------------+---------+------+---------------------------------+--------------+---------------+-------+-------------+
3 rows in set (0.00 sec)
```

```
mysql> EXPLAIN SELECT
    ->   p.*
    -> FROM (
    ->  SELECT customer_id, MAX(payment_date) as last_order
    ->  FROM payment
    ->  GROUP BY customer_id
    -> ) AS last_orders
    -> INNER JOIN payment p
    -> ON p.customer_id = last_orders.customer_id
    -> AND p.payment_date = last_orders.last_order;
+-------------+------------+---------+---------------------------------+-------------------+-------------------------------+-------+
| select_type | table      | type    | possible_keys                   | key               | ref                           | rows  |
+-------------+------------+---------+---------------------------------+-------------------+-------------------------------+-------+
| PRIMARY     | <derived2> | ALL     | NULL                            | NULL              | NULL                          |   599 |
| PRIMARY     | p          | ref     | idx_fk_customer_id,payment_date | payment_date      | customer_id,last_order        |     1 |
| DERIVED     | payment    | index   | NULL                            | idx_fk_customer_id | NULL                          | 16451 |
+-------------+------------+---------+---------------------------------+-------------------+-------------------------------+-------+
3 rows in set (0.10 sec)
```

- Note difference in times to grab EXPLAIN (0.00 vs. 0.10)
- Which plan looks simpler? looks better? *performs* better?

drastic performance difference

```
mysql> SELECT
    ->   p.*
    -> FROM payment p
    -> WHERE p.payment_date =
    -> ( SELECT MAX(payment_date)
    ->   FROM payment
    ->   WHERE customer_id=p.customer_id);
+------------+-------------+----------+-----------+--------+---------------------+---------------------+
| payment_id | customer_id | staff_id | rental_id | amount | payment_date        | last_update         |
+------------+-------------+----------+-----------+--------+---------------------+---------------------+
<snip>
|      16049 |         599 |        2 |     15725 |   2.99 | 2005-08-23 11:25:00 | 2006-02-15 19:24:13 |
+------------+-------------+----------+-----------+--------+---------------------+---------------------+
623 rows in set (0.49 sec)
```

```
mysql> SELECT
    ->   p.*
    -> FROM (
    ->  SELECT customer_id, MAX(payment_date) as last_order
    ->  FROM payment
    ->  GROUP BY customer_id
    -> ) AS last_orders
    -> INNER JOIN payment p
    -> ON p.customer_id = last_orders.customer_id
    -> AND p.payment_date = last_orders.last_order;
+------------+-------------+----------+-----------+--------+---------------------+---------------------+
| payment_id | customer_id | staff_id | rental_id | amount | payment_date        | last_update         |
+------------+-------------+----------+-----------+--------+---------------------+---------------------+
<snip>
|      16049 |         599 |        2 |     15725 |   2.99 | 2005-08-23 11:25:00 | 2006-02-15 19:24:13 |
+------------+-------------+----------+-----------+--------+---------------------+---------------------+
623 rows in set (0.09 sec)
```

- Only looking at 16K rows – gets *much* worse as size of table increases
- Derived table method retains good performance even as dataset grows to millions of records

operating on an indexed column with a function

```
mysql> EXPLAIN SELECT * FROM film WHERE title LIKE 'Tr%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: range
possible_keys: idx_title
          key: idx_title
      key_len: 767
          ref: NULL
         rows: 15
        Extra: Using where
```

- A fast *range* access strategy is chosen by the optimizer, and the index on title is used to *winnow* the query results down

```
mysql> EXPLAIN SELECT * FROM film WHERE LEFT(title,2) = 'Tr' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 951
        Extra: Using where
```

- A slow full table scan (the ALL access strategy) is used because a function (LEFT) is operating on the title column

correcting multiple faults in a SELECT statement

```
SELECT * FROM Orders WHERE TO_DAYS(CURRENT_DATE()) – TO_DAYS(order_created) <= 7;
```

- First, we are operating on an indexed column (order_created) with a function – let's fix that:

```
SELECT * FROM Orders WHERE order_created >= CURRENT_DATE() - INTERVAL 7 DAY;
```

- Although we rewrote the WHERE expression to remove the operating function, we still have a non-deterministic function in the statement, which eliminates this query from being placed in the query cache – let's fix that:

```
SELECT * FROM Orders WHERE order_created >= '2008-01-11' - INTERVAL 7 DAY;
```

- We replaced the function with a constant (probably using our application programming language). However, we are specifying SELECT * instead of the actual fields we need from the table.
- What if there is a TEXT field in Orders called order_memo that we don't need to see? Well, having it included in the result means a larger result set which may not fit into the query cache

```
SELECT order_id, customer_id, order_total, order_created
FROM Orders WHERE order_created >= '2008-01-11' - INTERVAL 7 DAY;
```

calculated fields

```
CREATE TABLE Customers (
  customer_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, email VARCHAR(80) NOT NULL
// more fields
, PRIMARY KEY (customer_id)
, INDEX (email(40))
);
```

```
mysql> ALTER TABLE Customers
    -> ADD COLUMN rv_email VARCHAR(80) NOT NULL;
...
mysql> UPDATE Customers
    -> SET rv_email = REVERSE(email);
...
mysql> CREATE INDEX ix_rv_email
    -> ON Customers (rv_email);
...
mysql> DELIMITER ;;
mysql> CREATE TRIGGER trg_bi_cust
    -> BEFORE INSERT ON Customers
    -> FOR EACH ROW BEGIN
    ->  SET NEW.rv_email = REVERSE(NEW.email);
    -> END ;;
...
mysql> CREATE TRIGGER trg_bu_cust
    -> BEFORE UPDATE ON Customers
    -> FOR EACH ROW BEGIN
    ->  SET NEW.rv_email = REVERSE(NEW.email);
    -> END ;;
mysql> DELIMITER ;
```

```
mysql> EXPLAIN SELECT COUNT(*) FROM Customers
    -> WHERE email LIKE '%.edu'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: Customers
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 910283
        Extra: Using where
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN SELECT COUNT(*) FROM Customers
    -> WHERE rv_email LIKE CONCAT(REVERSE('.edu'), '%')\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: Customers
         type: range
possible_keys: ix_rv_email
          key: ix_rv_email
      key_len: 242
          ref: NULL
         rows: 4387
        Extra: Using where; Using index
1 row in set (0.00 sec)
```

avoiding DELETEs

- Avoid DELETE, critical for MyISAM
  - Use a deleted_rows table
- Insert rows into the table, then periodically:

```
DELETE from main_table
INNER JOIN deleted_rows
ON main_table.id = deleted_rows.id;
TRUNCATE TABLE deleted_rows;
OPTIMIZE TABLE main_table;
```

- To SELECT from the main_table, do:

```
SELECT .. FROM main_table
LEFT JOIN deleted_rows
ON main_table.id = deleted_rows.id
WHERE deleted_rows.id IS NULL
```

- Will use a very fast NOT EXISTS predicate to satisfy the join condition

Jim Starkey
Senior Architect at MySQL AB

Referring to the insanely cheap amounts of storage available today and the modern enterprise's trend of not throwing any data away, he told me once:

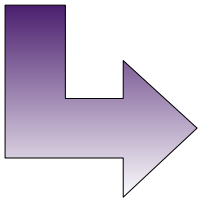
"*If the SQL standards committee removed the DELETE keyword, I don't think anyone would notice.*"

working with “mapping” or N:M tables (#1)

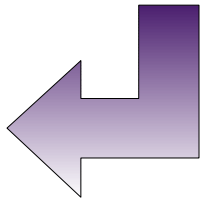
- Assume the following schema which shows a many-to-many relationship between Tags and Projects:

```
CREATE TABLE Project (
  project_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, name VARCHAR(50) NOT NULL
, url TEXT NOT NULL
, PRIMARY KEY (project_id)
) ENGINE=MyISAM;
```

```
CREATE TABLE Tags (
  tag_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, tag_text VARCHAR(50) NOT NULL
, PRIMARY KEY (tag_id)
, INDEX (tag_text)
) ENGINE=MyISAM;
```



```
CREATE TABLE Tag2Project (
  tag INT UNSIGNED NOT NULL
, project INT UNSIGNED NOT NULL
, PRIMARY KEY (tag, project)
, INDEX (project)
) ENGINE=MyISAM;
```



- The next few slides will walk through examples of querying across these relationship mapping tables

dealing with OR conditions

- Starting simple, let's grab the project names which match *either* the “mysql” tag *or* the “php” tag...

```
mysql> SELECT p.name FROM Project p
    -> INNER JOIN Tag2Project t2p
    -> ON p.project_id = t2p.project
    -> INNER JOIN Tag t
    -> ON t2p.tag = t.tag_id
    -> WHERE t.tag_text IN ('mysql','php');
+-------------------------------------------------------+
| name                                                  |
+-------------------------------------------------------+
| phpMyAdmin                                            |
...
| MySQL Stored Procedures Auto Generator                |
+-------------------------------------------------------+
90 rows in set (0.05 sec)
```

```
+----+-------------+-------+--------+----------------------+-------------+---------+-------------+------+-------------+
| id | select_type | table | type   | possible_keys        | key         | key_len | ref         | rows | Extra       |
+----+-------------+-------+--------+----------------------+-------------+---------+-------------+------+-------------+
|  1 | SIMPLE      | t     | range  | PRIMARY,uix_tag_text | uix_tag_text | 52      | NULL        |    2 | Using where |
|  1 | SIMPLE      | t2p   | ref    | PRIMARY,rv_primary   | PRIMARY      | 4       | t.tag_id    |   10 | Using index |
|  1 | SIMPLE      | p     | eq_ref | PRIMARY              | PRIMARY      | 4       | t2p.project |    1 |             |
+----+-------------+-------+--------+----------------------+-------------+---------+-------------+------+-------------+
3 rows in set (0.00 sec)
```

- Note the order in which the optimizer chose to join the tables is exactly the opposite of how we wrote our SELECT

dealing with AND conditions

- A little more complex, let's grab the project names which match ***both*** the “mysql” tag ***and*** the “php” tag
- Here is perhaps the most common method – using a HAVING COUNT(*) against a GROUP BY on the relationship table
- EXPLAIN on next page

```
mysql> SELECT p.name FROM Project p
    -> INNER JOIN (
    -> SELECT t2p.project
    -> FROM Tag2Project t2p
    -> INNER JOIN Tag t
    -> ON t2p.tag = t.tag_id
    -> WHERE t.tag_text IN ('mysql','php')
    -> GROUP BY t2p.project
    -> HAVING COUNT(*) = 2
    -> ) AS projects_having_all_tags
    -> ON p.project_id = projects_having_all_tags.project;
+-------------------------------------------+
| name                                      |
+-------------------------------------------+
| phpMyAdmin                                |
...
| MySQL Stored Procedures Auto Generator    |
+-------------------------------------------+
8 rows in set (0.00 sec)
```

dealing with AND conditions (cont'd)

- The EXPLAIN plan shows the execution plan using a derived table containing the project Ids having records in the Tag2Project table with both the “mysql” and “php” tags
- Note that a filesort is needed on the Tag table rows since we use the index on tag_text but need a sorted list of tag_id values to use in the join

```
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: <derived2>
         type: ALL
         rows: 8
*************************** 2. row ***************************
           id: 1
  select_type: PRIMARY
        table: p
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: projects_having_all_tags.project
         rows: 1
*************************** 3. row ***************************
           id: 2
  select_type: DERIVED
        table: t
         type: range
possible_keys: PRIMARY,uix_tag_text
          key: uix_tag_text
      key_len: 52
         rows: 2
        Extra: Using where; Using temporary; Using filesort
*************************** 4. row ***************************
           id: 2
  select_type: DERIVED
        table: t2p
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: t.tag_id
         rows: 10
        Extra: Using index
4 rows in set (0.00 sec)
```

alternate method for AND conditions

- Do two separate queries – one which grabs tag_id values based on the tag text and another which does a self-join after the application has the tag_id values in memory

Benefit #1

- If we assume the Tag2Project table is updated 10X more than the Tag table is updated, the first query on Tag will be cached more effectively in the query cache

Benefit #2

- The EXPLAIN plan on the self-join query is ***much*** better than the HAVING COUNT(*) derived table solution

```
mysql> SELECT t.tag_id FROM Tag t
     > WHERE tag_text IN ("mysql","php");
+--------+
| tag_id |
+--------+
|      3 |
|      2 |
+--------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT p.name FROM Project p
    -> INNER JOIN Tag2Project t2p
    -> ON p.project_id = t2p.project
    -> AND t2p.tag = 2
    -> INNER JOIN Tag2Project t2p2
    -> ON t2p.project = t2p2.project
    -> AND t2p2.tag = 3;
+-------------------------------------------+
| name                                      |
+-------------------------------------------+
| phpMyAdmin                                |
...
| MySQL Stored Procedures Auto Generator    |
+-------------------------------------------+
8 rows in set (0.00 sec)
```

alternate method for AND conditions (cont'd)

```
mysql> EXPLAIN SELECT t.tag_id FROM Tag t
    -> WHERE tag_text IN ("mysql","php");
+----+-------------+-------+-------+---------------+--------------+---------+------+------+-------------+
| id | select_type | table | type  | possible_keys | key          | key_len | ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+-------------+
|  1 | SIMPLE      | t     | range | uix_tag_text  | uix_tag_text | 52      | NULL |    2 | Using where |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

- Removed the filesort which occurred with the derived table
- Top query will be cached in query cache

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t2p2
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 26
        Extra: Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: t2p
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: const,t2p2.project
         rows: 1
        Extra: Using index
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: p
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: t2p2.project
         rows: 1
        Extra:
3 rows in set (0.00 sec)
```

getting random records

- Never do ORDER BY RAND() LIMIT *x*
  - MySQL will perform a full table scan, executing the RAND() function *for each record in the table*

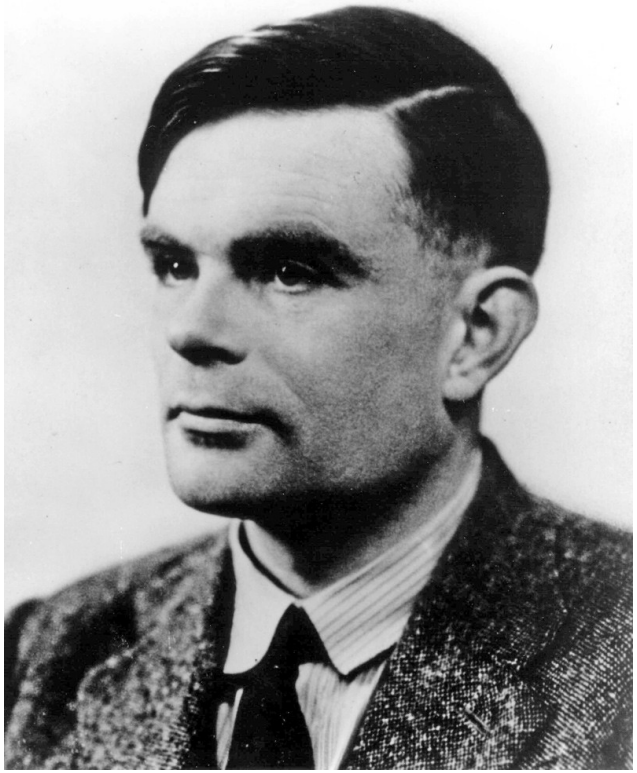
```
mysql> EXPLAIN SELECT * FROM ForgeUser ORDER BY RAND() LIMIT 1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: ForgeUser
         type: ALL
         rows: 910
        Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

- Instead, if you have a sequential primary key, you can use a user variable trick...

```
mysql> SELECT MAX(user_id) INTO @max_id FROM ForgeUser;
Query OK, 1 row affected (0.01 sec)

mysql> SET @rand_id = CEIL(RAND() * @max_id);
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN SELECT * FROM ForgeUser WHERE user_id = @rand_id;
+----+-------------+-----------+-------+---------------+---------+---------+-------+------+-------+
| id | select_type | table     | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+----+-------------+-----------+-------+---------------+---------+---------+-------+------+-------+
|  1 | SIMPLE      | ForgeUser | const | PRIMARY       | PRIMARY | 4       | const |    1 |       |
+----+-------------+-----------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.00 sec)
```

a word on stored procedures

- Question: where does the stored procedure compile cache live?
  - Answer: on the *connection thread*
- Don't use stored procedures for simple SELECTs
- In shared-nothing architectures like PHP, this is a recipe for disaster
- Use for:
  - ETL or complex collections of SQL
  - Repeated execution of statement
  - Batch operations



Alan Turing
1912-1954

The idea behind a “program as data” can be traced to Turing's work on computing machines and the archetypal Universal Turing Machine, sometimes called the “stored-program computer”

server status counters and variables

- Can be a temporary fix
- Be aware of what is *global* vs. *per connection thread*
  - e.g. key_buffer_size is global buffer for MyISAM key cache
  - but... sort_buffer_size is per thread (for scans)
- Use SHOW STATUS
  - Can take a LIKE clause, like so:

```
mysql> SHOW STATUS LIKE 'Created_tmp%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 499   |
| Created_tmp_files       | 5     |
| Created_tmp_tables      | 1933  |
+-------------------------+-------+
```

getting hit ratios

- Hit ratios are a measure of the effectiveness of the database server
- Query cache hit ratio:

```
Qcache_hits / (Com_select + Qcache_hits)
```

- MyISAM key buffer hit ratio:

```
Key_read_requests / Key_reads
```

- InnoDB buffer pool hit ratio:

```
Innodb_buffer_pool_read_requests /
    Innodb_buffer_pool_reads
```

```
mysql> SHOW STATUS LIKE 'Qc%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Qcache_free_blocks      | 1     |
| Qcache_hits             | 6     |
| Qcache_inserts          | 12    |
| Qcache_not_cached       | 41    |
| Qcache_lowmem_prunes    | 0     |
+-------------------------+-------+
```

```
mysql> SHOW STATUS LIKE 'Key_read%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Key_read_requests | 10063 |
| Key_reads         | 98    |
+-------------------+-------+
```

```
mysql> SHOW STATUS LIKE 'Innodb_buffer_pool_read
%';
+---------------------------------+---------+
| Variable_name                   | Value   |
+---------------------------------+---------+
| Innodb_buffer_pool_read_requests | 5415365 |
| Innodb_buffer_pool_reads         | 34260   |
+---------------------------------+---------+
```

designing for scale-out

Gene Amdahl
Amdahl's Law

Amdahl's Law is used in parallel computing to determine the expected improvement in a system's performance when only a certain part of the system is improved.

- Think about scaling out early in your design
- Think about serving reads from multiple slaves
  - Architect your data access layer to be as seamless as possible
  - Use modular code to allow application code to be blissfully unaware of which (set of) server(s) the reads should come from
- Put as little business logic inside the database as you can
  - Put logic in the application tiers which can be scaled separately

cache everywhere

- No major website achieves scale without multiple caching layers
- Cache from the database
  - Query cache, key buffer, innodb buffers
- Cache session and frequently updated data tier
  - Memcached or even local file storage
  - MySQL memcached storage engine
- Cache web pages or page pieces
  - Templating solutions, proxies

favorite resources

- **Blogs**
  - *Peter Zaitsev*
    - http://mysqlperformanceblog.com
  - *Baron Schwartz*
    - http://xaprb.com
  - *All of us...*
    - http://planetmysql.org
- **MySQL forge**
  - http://forge.mysql.com
  - Projects, code snippets, tools, wiki, more
  - New 2.0 forge coming out in next two weeks
- **MySQL developer zone**
  - http://dev.mysql.com

Baron Schwartz
http://xaprb.com

"xaprb" is Baron spelled on a Dvorak keyboard.

yo, we're hiring

- Dozens of positions open
- Work from anywhere
  - Yep, even at Sun. :)
- Great bennies
- Travel if you want to
- A Swedish company culture (but, sorry, no free massages or hot babes.)
- Come to the job fair tonight!

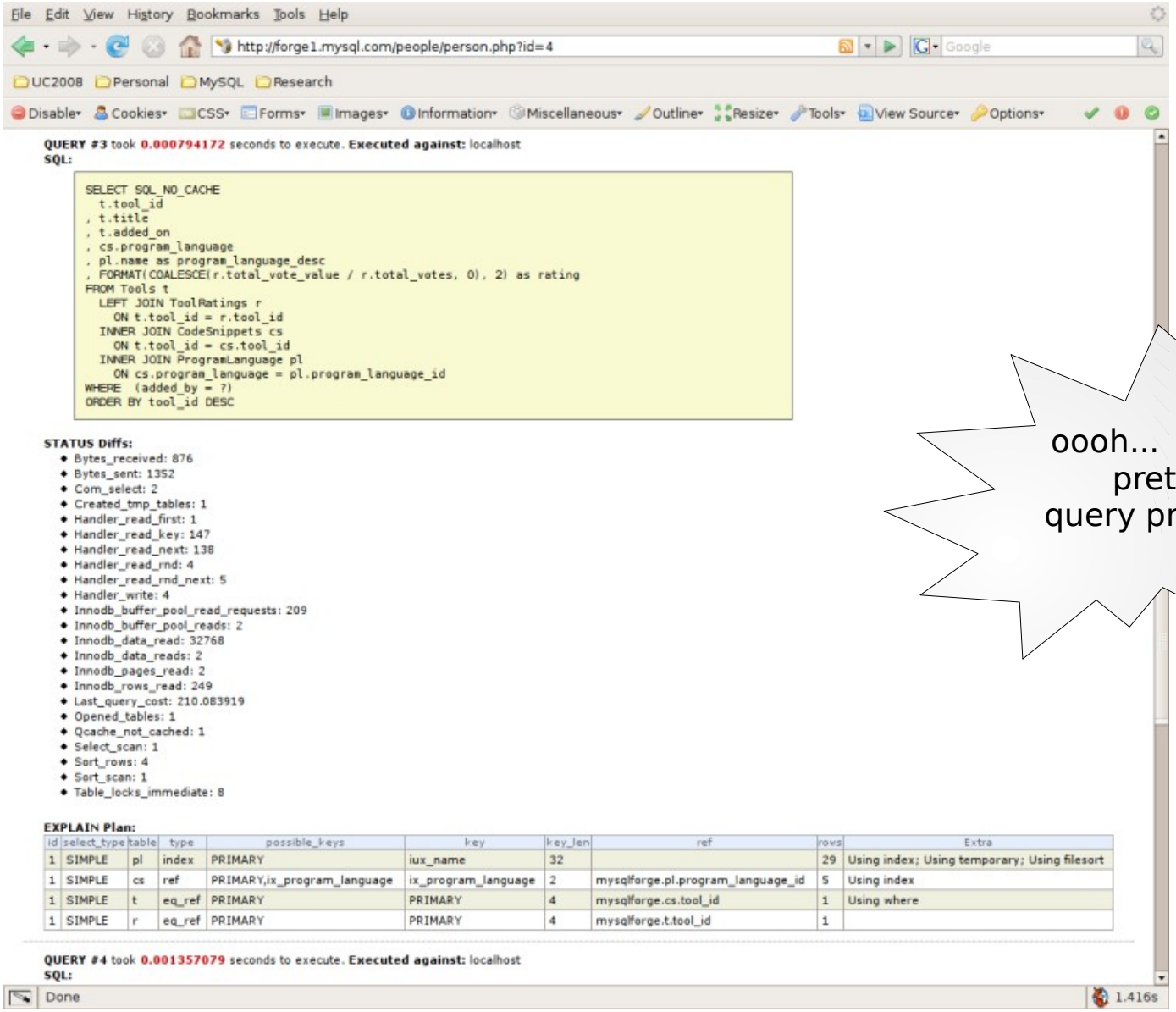
yo, we're conferencing, too

- http://mysqlconf.com
  - 14 tutorials, 115+ sessions
  - April 14-17, 2008, Santa Clara
  - Early Registration through February 26th

putting it all together

- Rest of slides show a possible solution for
  - Flexible, replication-aware data access layer
  - Cache-aware framework
  - Profiling of MySQL database queries
- Written in PHP5.2 but concepts should translate
- This is real code, not fake stuff
  - Used in the next generation of MySQL Forge
- Take what you want, leave the rest
  - The point is to generate ideas and get you thinking

what you will probably like best



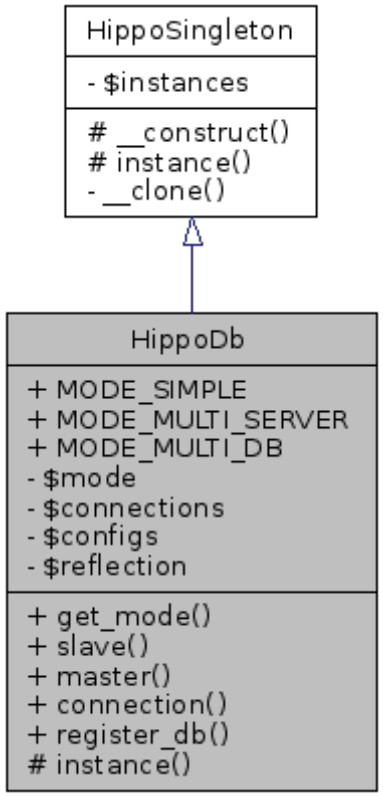
the battle against spaghetti

- How is this related to performance?
  - If you have easier to maintain code, you can spend more timing tuning it! :)
- If it's reusable, it will get reused
- If it's not, it will get rewritten (over and over and over again)
- Use design patterns to overcome obstacles
- Try to avoid tight coupling of objects together
  - Loosely coupled objects are more re-usable
- Number one principle: *a class/object should do **one** thing and do it well*

After looking at my old MySQL Forge code base...

I knew exactly how this boy felt.

a declarative setup for scale out



- A factory class (HippoDB) registers any database server that can be used by the application
  - Connections to MySQL *are not made until a query is actually issued*
- Request a slave or master connection in the application and the library takes care of polling for slaves, falling back to master, etc

```
/* in config.php */
/* Register servers for the MySQL forge */
HippoDb::register_db('forge'
      , 'mysql:host=localhost;dbname=mysqlforge'
      , 'mysqlforge'
      , 'XXXXXXXXX'
      , $is_master=TRUE);

/* Register servers for the MySQL single sign-on */
HippoDb::register_db('single_signon'
      , 'mysql:host=master.web;dbname=web'
      , 'mysqlforge'
      , 'XXXXXXXXX'
      , $is_master=TRUE);

HippoDb::register_db('single_signon'
      , 'mysql:host=slave.web;dbname=web'
      , 'mysqlforge'
      , 'XXXXXXXXX'
      , $is_master=FALSE);

HippoDb::register_db('single_signon'
      , 'mysql:host=slave2.web;dbname=web'
      , 'mysqlforge'
      , 'XXXXXXXXX'
      , $is_master=FALSE);
```

using connections

```
// In ProjectHandler::display_add_project()
$reader= new ProgramLanguageReader(HippoDb::slave(), $this->cache);
$this->handler_vars['program_languages']= $reader->get_hash();


// In WorklogHandler::on_after_worklog_add_comment()
public function on_after_worklog_add_comment() {
      /* Grab a list of email addresses to notify of feedback */
      $sql=<<<ENDOFSQL
SELECT email
FROM user u
INNER JOIN observers o
ON u.id = o.dev_id
WHERE o.task_id = ?
ENDOFSQL;
      if ($connection= HippoDb::slave(array('db'=>'worklog_remote'))) {
            $connection->add_param($this->handler_vars['task_id']);
            if (! $emails= $connection->get_column($sql))
                  $emails= array();

            /* Grab the list of commenter's emails for this task */
            if ($connection= HippoDb::master(array('db'=>'forge'))) {
                  $sql=<<<ENDOFSQL
SELECT fu.email
FROM ForgeUser fu
INNER JOIN WLTaskComment c
ON fu.user_id = c.added_by
WHERE c.added_by != ?
GROUP BY c.added_by
ENDOFSQL;
                  $connection->add_param($this->session->user_id);
                  if ($more_emails= $connection->get_column($sql))
                        $emails= array_unique(array_merge($emails, $more_emails));
...
```

Note that our main handler code **doesn't care how many slaves there are or where the slaves are located**. It just asks for a slave and is supplied a connection, with the master acting as fallback

master() because we just added a new comment, so slave might have a lag

Add a parameter to replace the ? in the SQL statement (prepared statement) and then get just the column (email) from the DB

using transactions

```
// In ProjectHandler::edit_project()
$connection= HippoDb::master();
$project= new Project($connection);

$connection->start_transaction();
if (! $project->save()) {
      $this->errors= $project->errors;
      $connection->rollback();
}
else {
      $project_text= new ProjectText($connection);
      $project_text->project= $project->project_id;
      $project_text->name= $this->handler_vars['name'];
      $project_text->description= $this->handler_vars['description'];
      if (! $project_text->save()) {
            $this->errors= $project_text->errors;
            $connection->rollback();
    }
      else {
            $connection->commit();
      }
}
```

Project and ProjectText are ActiveRecord-type classes which handles saving of data to the database

reading data with HippoReader

```
class EntryReader extends HippoReader {...}

// Use the reader like so:
$reader= new EntryReader(HippoDb::slave());
// Grab all entries in January 2007 and
// which have the entered_by user ID of 34
$reader->entry_date->between('2007-01-01', '2007-01-31');
$reader->entered_by= 34;
// Next, simply read the data into an associative array
$entries= $reader->get();

// A more complex example
$reader= new ProjectReader(HippoDb::slave());
// We can override the ORDER BY
$reader->order('added_on', 'desc');
// We can also override the default SELECT expression
// to optimize or customize
$reader->select_sql =<<<ENDOFSQL
SELECT SQL_NO_CACHE
  p.project_id
, p.name
, p.added_on
, p.added_by
, ab.display_name as added_by_display
FROM Project p
INNER JOIN ForgeUser ab
  ON p.added_by = ab.user_id
ENDOFSQL;
// We can pass an integer to reader->get() to limit the number of results
if (! $newest_projects= $reader->get(15))
  $newest_projects= array();
```

- Base class for all data reading classes
- Purpose is to just read data, not modify it
- Designed to make reading simple to medium-complexity data sets intuitive for the programmer
  - But easy to optimize for performance and specifically for MySQL

built-in SQL query profiling

- We wanted a way to see what exactly was going on inside MySQL for each query issued against the DB servers
- Wanted a modular profiler that could be *entirely* decoupled from the HippoDb and HippoSqlConnection code and be initiated separately
- Wanted the profiler to just “listen” to SQL query events and record them whenever they happen

```
// in config.php
$profiler= new HippoSqlProfiler();
$profiling_options= (
  HippoSqlProfiler::HIPPO_PROFILE_TIMER
| HippoSqlProfiler::HIPPO_PROFILE_STATUS_DIFF
| HippoSqlProfiler::HIPPO_PROFILE_EXPLAIN
);
$profiler->profiling_options= $profiling_options;
HippoEvents::listen('on_before_statement_execute', array($profiler, 'start'));
HippoEvents::listen('on_after_statement_execute', array($profiler, 'stop'));

function add_profiles() {
      if (isset($GLOBALS['handler']))
            $GLOBALS['handler']->handler_vars['profiles']= $GLOBALS['profiler']->get_profiles();
            $GLOBALS['handler']->handler_vars['profiler']= $GLOBALS['profiler'];
}
HippoEvents::listen('on_before_display_template', 'add_profiles');
?>
```

event hooks

```
class HippoEvents extends HippoSingleton {
  /** Internal collection of event hooks */
  private $events= array();
...
  /**
   * Registers a listener callback for a named event
   ...
   * @param event_name  Name of the event to listen for
   * @param callback    Action ... to take
   * @param args      (optional) arguments for the callback
   */
  public static function listen($event_name, $callback, $args=array()) {
    $instance= HippoEvents::instance();
    if (isset($instance->events[$event_name]))
      $instance->events[$event_name][]=
        array('callback'=>$callback, 'args'=>$args);
    else
      $instance->events[$event_name]=
        array(array('callback'=>$callback, 'args'=>$args));
  }

  public static function notify($event_name, $state_instance=null) {
    $instance= HippoEvents::instance();
    if (isset($instance->events[$event_name])) {
      foreach ($instance->events[$event_name] as $observer) {
        if (is_array($observer['callback'])) {
          $caller_instance= $observer['callback'][0];
          $caller_method= $observer['callback'][1];
          $caller_instance->$caller_method($state_instance);
        }
        else
          $observer['callback']($state_instance);
      }
    }
  }
}
```

- The final piece of the puzzle
- Asynchronous event registration/notification
- Allows our classes to react to events which occur in plugins, global space, classes, anything...