Indexierung mit MySQL

DOAG SIG MySQL – Performance
13. März 2012, Wiesbaden

Oli Sennhauser

Senior MySQL Consultant, FromDual GmbH

oli.sennhauser@fromdual.com

FromDual GmbH

- **FromDual bietet neutral und unabhängig:**
  - **Beratung für MySQL**
  - **Support für MySQL und Galera Cluster**
  - **Remote-DBA / MySQL Betrieb**
  - **Schulung für MySQL**
- **Oracle Silber Partner (OPN)**

www.fromdual.com

Kunden

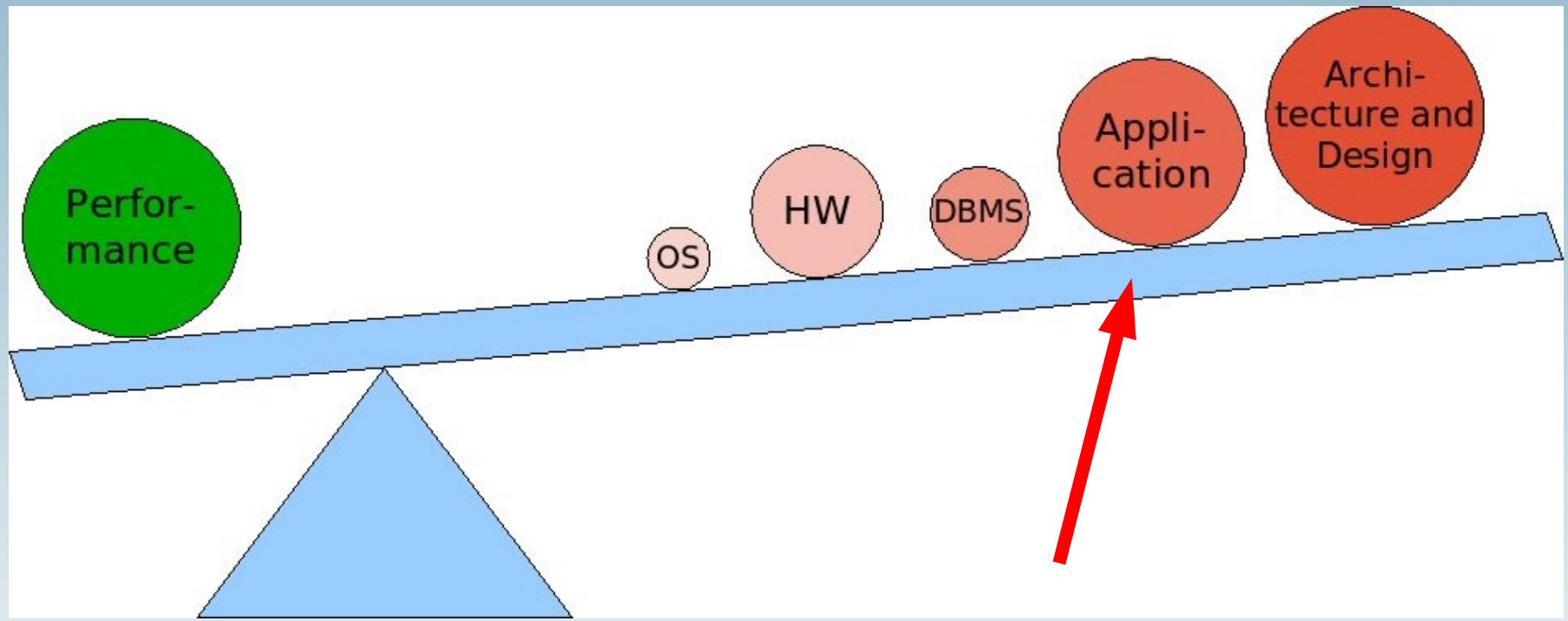

HolidayCheck.com
linuxhotel
Schulungen und Universitäten machen Sinn.
Hostloco
mespas
OX
open-xchange
RYGOL
DÄMMSTOFFE
Boerse Stuttgart
sedo
TEAM NETSOL
careerJET.com
DZ
INFOnline
verified by swiss sign GOLD
STRATO
fashion days
EXCLUSIVE SHOPPING
Flat Planet Phone Company
billiger-mietwagen.de
ERICSSON
BOSCH
Invented for life
CoreTelecom
INNOVATIONS INC.
The Future of Telephony
nexellent
communications. delivered.
Diogenes
integrics
Integrated Telecommunication Solutions
GfK TELECONTROL
University of Zurich UZH

Inhalt

Indexierung mit MySQL

- Grundlagen
- Indizes anzeigen in MySQL
- Indizes und Storage Engines
- Indizes in InnoDB
- Indizes optimieren
- Der Query Execution Plan (QEP)

FromDual Performance Waage



Grundlagen

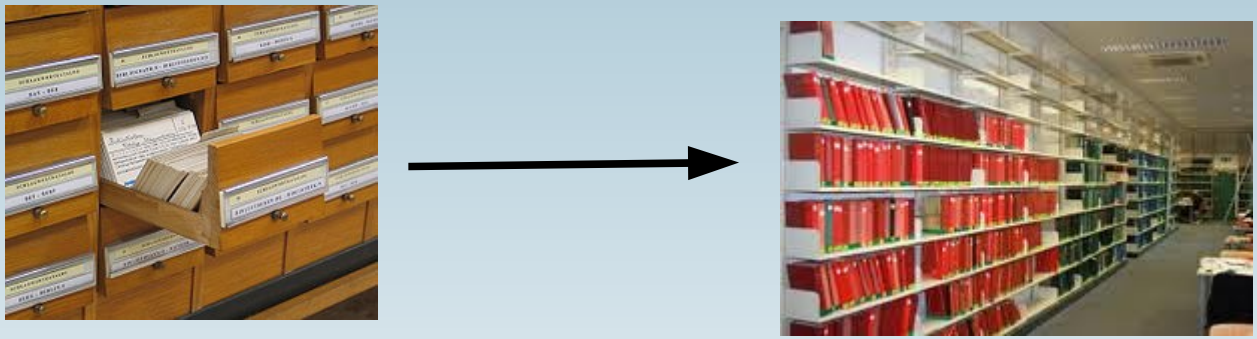
- **Was ist ein Index?**

  "Ein Index ist eine von den Daten getrennte Struktur, welche die Suche und das Sortieren nach bestimmten Feldern beschleunigt."

- **Warum brauchen wir Indizes?**
  - **Sortieren**
  - **Suchen**
- **Vorteil / Nachteil**
  - **Performance**
  - **Pflege, Platz**

Grundlagen

- **Beispiel:**
  - **Karteikarten in Bibliothek → Stockwerk, Raum, Gestell, Reihe**
  - **Simmel, Johannes M. → 3. St., links, 5. Gest. 2. R.**



- **Alternativen zum Index?**
  - **Full Library Scan → „Full Table Scan“**

Vor- und Nachteile von Indizes

- Performance, beschleunigt:

  `SELECT` (und `UPDATE`, `DELETE`)

- Performance, bremst DML-Statements

  `INSERT`, `UPDATE`, `DELETE`, ...

- Indizes brauchen Platz (RAM, Disk)
- Wartung, Pflege, Unterhalt (`OPTIMIZE`, `ANALYZE`)?
- Indizes können Optimizer verwirren!

  → falsche Query Execution Pläne → langsame Abfragen

→ So viele wie nötig aber so wenig wie möglich!

Wo sollten wir Indizes setzen?

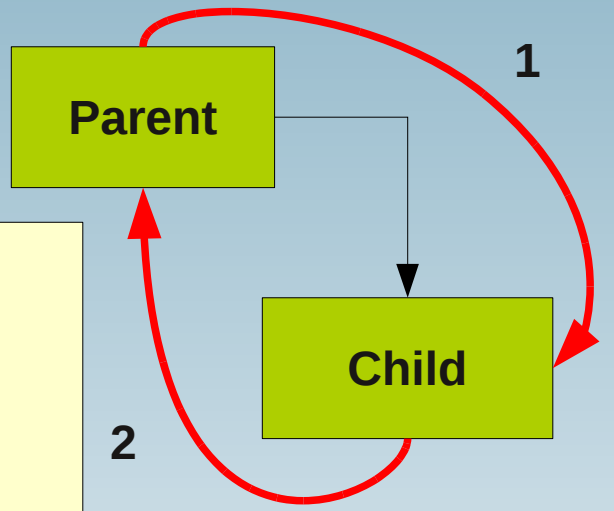
„... die Suche und das Sortieren nach bestimmten Feldern beschleunigt."

- Suche: `WHERE`-Klausel
- Sortieren: `DISTINC`, `ORDER BY`, `GROUP BY`
- `JOIN`-Klausel: ... `a JOIN b ON a.id = b.a_id`

  → und zwar in beide Richtungen!
- Spezialfall: Covering Index

Indexieren bei JOINs

- Je nach Query: 2 Möglichkeiten:
  - Parent → Child
  - Child → Parent




```
EXPLAIN SELECT *
   FROM parent AS p
   JOIN child AS c ON c.p_id = p.id
  WHERE p.id = 69999;
+-------+-------+---------------+---------+---------+-------+------+
| table | type  | possible_keys | key     | key_len | ref   | rows |
+-------+-------+---------------+---------+---------+-------+------+
| p     | const | PRIMARY       | PRIMARY | 4       | const |    1 |
| c     | ref   | p_id          | p_id    | 4       | const |    4 |
+-------+-------+---------------+---------+---------+-------+------+

EXPLAIN SELECT *
   FROM parent AS p
   JOIN child AS c ON p.id = c.p_id
  WHERE c.f = 69999;
+-------+--------+---------------+---------+---------+------------+------+
| table | type   | possible_keys | key     | key_len | ref        | rows |
+-------+--------+---------------+---------+---------+------------+------+
| c     | ref    | p_id,f        | f       | 4       | const      |    1 |
| p     | eq_ref | PRIMARY       | PRIMARY | 4       | test.c.p_id |    1 |
+-------+--------+---------------+---------+---------+------------+------+
```

Arten von Indizes

- B+-Tree Index (Balanced plus tree)
  - Geeignete Struktur für Disks
  - Geeignet für viele unterschiedliche Werte (hohe Kardinalität)
  - InnoDB, MyISAM, (MEMORY, NDB)
- Hash-Index
  - Geeignete Struktur für Speicher
  - MEMORY, NDB, (InnoDB)
- R-Tree Index
  - Mehrdimensionale Informationen (Geo-Indizes)
  - MyISAM
- Fulltext Index
- `UNIQUE` Index (Spezialfall von B+-Tree und Hash)
- Bitmap Index :(

Indizes anzeigen mit MySQL

```
SHOW CREATE TABLE test\G

CREATE TABLE `test` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `data` varchar(64) DEFAULT NULL,
  `ts` timestamp NOT NULL
  PRIMARY KEY (`id`),
  KEY `data` (`data`)
);
```

mysqldump --no-data

```
SHOW INDEX FROM test;

+-------+------------+----------+--------------+-------------+-------------+------+------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Cardinality | Null | Index_type |
+-------+------------+----------+--------------+-------------+-------------+------+------------+
| test  |          0 | PRIMARY  |            1 | id          |       18469 |      | BTREE      |
| test  |          1 | data     |            1 | data        |          26 | YES  | BTREE      |
+-------+------------+----------+--------------+-------------+-------------+------+------------+
```

Indizes anzeigen mit MySQL

```
SELECT table_schema, table_name, column_name, ordinal_position
     , column_key
  FROM information_schema.columns
 WHERE table_name = 'test'
   AND table_schema = 'test';
+--------------+------------+-------------+------------------+------------+
| table_schema | table_name | column_name | ordinal_position | column_key |
+--------------+------------+-------------+------------------+------------+
| test         | test       | id          |                1 | PRI        |
| test         | test       | data        |                2 | MUL        |
| test         | test       | ts          |                3 |            |
+--------------+------------+-------------+------------------+------------+
```

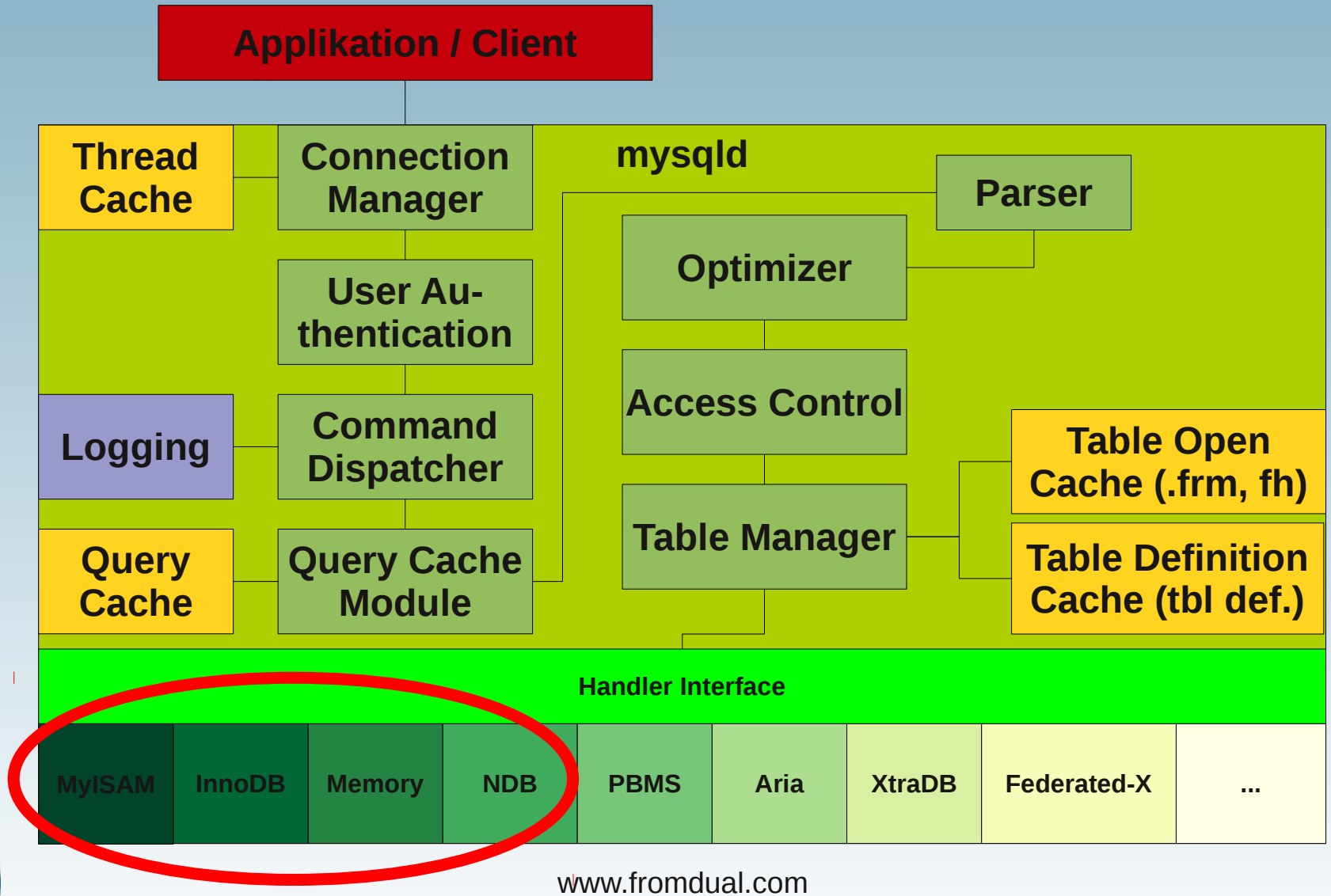
```
CREATE TABLE innodb_table_monitor (id INT) ENGINE=InnoDB;
TABLE: name test/test, id 0 1333, columns 6, indexes 2, appr.rows 18469
  COLUMNS: id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
  ...
  INDEX: name PRIMARY, id 0 1007, fields 1/5, uniq 1, type 3
   root page 3, appr.key vals 18469, leaf pages 54, size pages 97
   FIELDS:  id DB_TRX_ID DB_ROLL_PTR data ts
  INDEX: name data, id 0 1008, fields 1/2, uniq 2, type 0
   root page 4, appr.key vals 1, leaf pages 32, size pages 33
   FIELDS:  data id
```

Indizes anzeigen mit MySQL

- **Neu mit MySQL 5.6**

```
use information_schema;

SELECT t.name, t.n_cols, i.index_id, i.name, i.n_fields
  FROM innodb_sys_tables AS t
  JOIN innodb_sys_indexes AS i ON t.table_id = i.table_id
 WHERE t.name = 'test/test';
+-----------+--------+----------+---------+----------+
| name      | n_cols | index_id | name    | n_fields |
+-----------+--------+----------+---------+----------+
| test/test |      6 |      644 | PRIMARY |        1 |
| test/test |      6 |      650 | data    |        1 |
+-----------+--------+----------+---------+----------+
```

MySQL Architektur



Indizes mit MyISAM

- MyISAM kennt:
  - B+-Tree
  - Volltext
  - R-Tree
- Variable:
  - `key_buffer_size`
  - ~ 25 – 33% vom RAM

```
CREATE TABLE `test` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `data` varchar(64) DEFAULT NULL,
  `ts` timestamp NOT NULL,
  `geo` geometry NOT NULL,
  PRIMARY KEY (`id`),
  SPATIAL KEY `geo` (`geo`),
  FULLTEXT KEY `data` (`data`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO test
VALUES (NULL, 'Uster ist ein kleines Städtchen.'
      , NULL, GeomFromText('POINT(42 24)'));

SELECT *
  FROM test
 WHERE MATCH (data)
AGAINST ('Uster' IN BOOLEAN MODE);
```

Indizes mit MEMORY (HEAP)

- MEMORY (alt HEAP)
  - Hash Index (default)
  - B+-Tree Index
- Hash langsam bei kleiner Kardinalität!
  - → B+-Tree!
- Variable: `max_heap_table_size`

```
CREATE TABLE `test` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `data` varchar(64) DEFAULT NULL,
  `ts` timestamp NOT NULL
  PRIMARY KEY (`id`),
  KEY `data` (`data`) USING BTREE
) ENGINE=MEMORY;

SHOW INDEX FROM test;
```

```
+-------+------------+----------+-------------+-------------+------+------------+
| Table | Non_unique | Key_name | Column_name | Cardinality | Null | Index_type |
+-------+------------+----------+-------------+-------------+------+------------+
| test  |          0 | PRIMARY  | id          |         128 |      | HASH       |
| test  |          1 | data     | data        |        NULL | YES  | BTREE      |
+-------+------------+----------+-------------+-------------+------+------------+
```

Indizes mit InnoDB

- InnoDB (`innodb_buffer_pool_size`)
  - B+-Tree
  - FULLTEXT (ab MySQL 5.6)
  - Hash (Adaptive Hash Index, selbständig)
- Unterscheidung in
  - Primary Key (PK)
  - Secondary Key (SK) → alles != PK

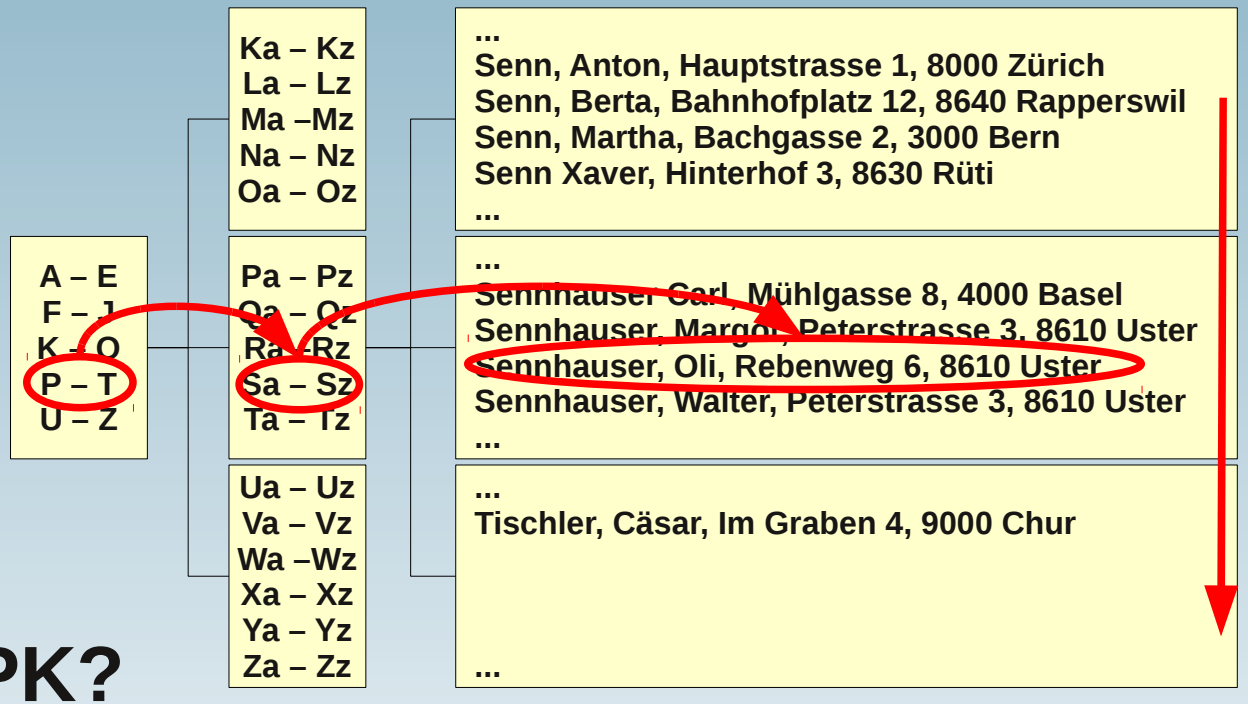
InnoDB Primary Key

- Table = geclusterter Index = Primary Key
  - Oracle: Index Organized Table (IOT)
  - MS SQL Server: Clustered Index
  - PostgreSQL: Cluster
- Beeinflusst:
  - Sortier-Reihenfolge
  - Lokalität der Daten (Locality of data)
  - Länge der Secondary Keys
  - → InnoDB ist sehr schnell für PK Zugriffe/Scans!

InnoDB Primary Key

- **Geclusterter PK (Nachname, Vorname):**

```
SELECT *
  FROM people
 WHERE lastname = 'Sennhauser'
   AND firstname = 'Oli';
```



- **Was wenn kein PK?**
  - **UNIQUE INDEX**
  - **Autogenerated Index**

Fehlender InnoDB PK

```
TABLE: name test/test, id 0 1339, columns 6, indexes 2, appr.rows 0

COLUMNS: id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
        data: DATA_VARCHAR prtype 524303 len 64;
          ts: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
  DB_ROW_ID: DATA_SYS prtype 256 len 6;
  DB_TRX_ID: DATA_SYS prtype 257 len 6;
DB_ROLL_PTR: DATA_SYS prtype 258 len 7;

INDEX: name GEN_CLUST_INDEX, id 0 1014, fields 0/6, uniq 1, type 1
  root page 3, appr.key vals 0, leaf pages 1, size pages 1
  FIELDS:  DB_ROW_ID DB_TRX_ID DB_ROLL_PTR id data ts

INDEX: name data, id 0 1015, fields 1/2, uniq 2, type 0
  root page 4, appr.key vals 0, leaf pages 1, size pages 1
  FIELDS:  data DB_ROW_ID

----

TABLE: name test/test2, id 0 1340, columns 6, indexes 2, appr.rows 0
COLUMNS: id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
       data: DATA_VARCHAR prtype 524303 len 64;
         ts: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
  DB_ROW_ID: DATA_SYS prtype 256 len 6;
  DB_TRX_ID: DATA_SYS prtype 257 len 6;
DB_ROLL_PTR: DATA_SYS prtype 258 len 7;

INDEX: name PRIMARY, id 0 1016, fields 1/5, uniq 1, type 3
  root page 3, appr.key vals 0, leaf pages 1, size pages 1
  FIELDS:  id DB_TRX_ID DB_ROLL_PTR data ts

INDEX: name data, id 0 1017, fields 1/2, uniq 2, type 0
  root page 4, appr.key vals 0, leaf pages 1, size pages 1
  FIELDS:  data id
```

InnoDB Secondary Key

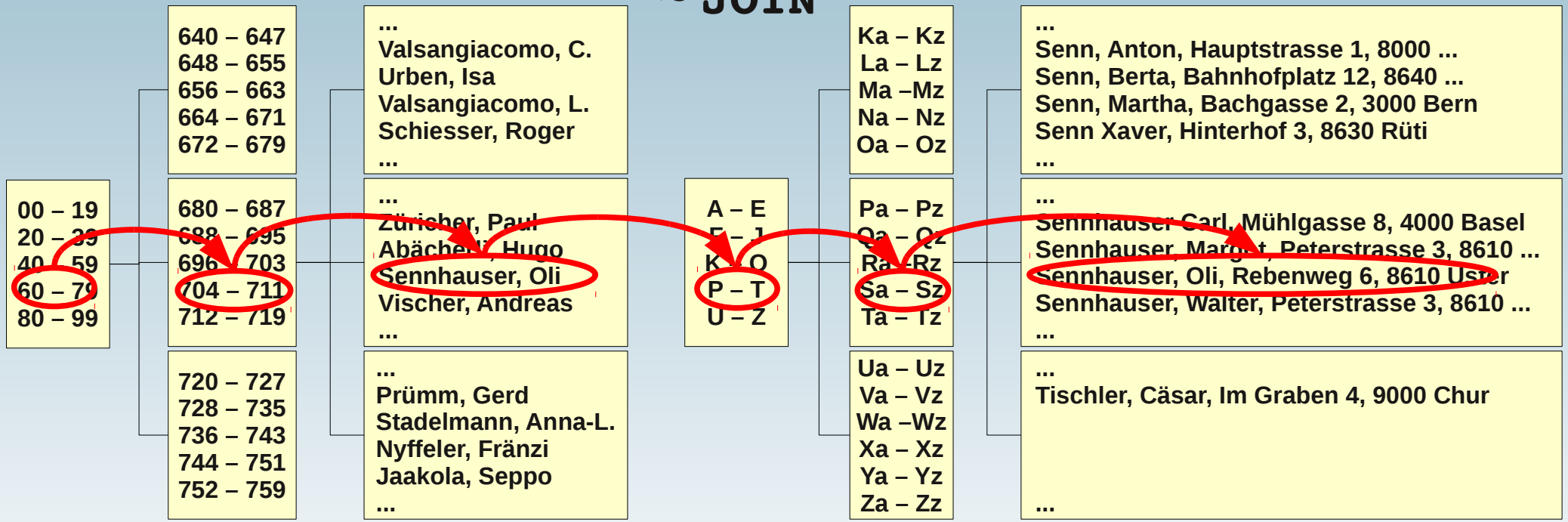
- Grösse des PK spielt eine Rolle
- SK Zugriff ist teurer als PK Zugriff
- Zugriff auf SK ist ähnlich wie `JOIN`
- Ab MySQL 5.5: Fast Index Create/Drop

InnoDB Secondary Key

- SK (Personal-Nummer):

```
SELECT *
  FROM people
 WHERE personal_id = '70720230344';
```

~ JOIN



InnoDB Secondary Key

```
TABLE: name test/crap, id 0 1342, columns 10, indexes 3, appr.rows 0

  COLUMNS: a: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           b: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           c: DATA_VARCHAR prtype 524559 len 32;
           d: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 8;
           e: DATA_VARCHAR prtype 524559 len 32;
           f: DATA_INT DATA_BINARY_TYPE len 4;
           g: DATA_INT DATA_BINARY_TYPE len 4;
   DB_ROW_ID: DATA_SYS prtype 256 len 6;
   DB_TRX_ID: DATA_SYS prtype 257 len 6;
 DB_ROLL_PTR: DATA_SYS prtype 258 len 7;

  INDEX: name PRIMARY, id 0 1019, fields 5/9, uniq 5, type 3
   root page 3, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  a b c d e DB_TRX_ID DB_ROLL_PTR f g

  INDEX: name f, id 0 1020, fields 1/6, uniq 6, type 0
   root page 4, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  f a b c d e

  INDEX: name g, id 0 1021, fields 1/6, uniq 6, type 0
   root page 5, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  g a b c d e
```

Index-Arten

- **Single Column Index**

```
ALTER TABLE test ADD COLUMN (data);
```

- **Multi Column Index**

```
ALTER TABLE test ADD COLUMN (a, b);
```

- **Covering Index**

```
ALTER TABLE test ADD COLUMN (a, b, c, data);
```

- **Prefixed Index**

```
ALTER TABLE test ADD COLUMN (a, data(16));
```

Indizes optimieren?

- **Fehlende Indizes → einfach! :-)**

```
# my.cnf
slow_query_log                = 1
slow_query_log_file           = slow.log
log_queries_not_using_indexes = 1
```

→ **EXPLAIN …**

- **Zu viele Indizes → etwas schwieriger! :-(**
  - **„Userstat“ (Percona Server)**
- **Warum Indizes Optimieren?**
  - **Performance**
  - **Zu kleiner Speicher → random I/O (sehr langsam!)**

Userstat (Percona Sever)

```
CREATE TABLE `test` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `data` varchar(64) DEFAULT NULL,
  `ts` timestamp NOT NULL,
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `c` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `data` (`data`),
  KEY `a` (`a`),
  KEY `b` (`b`),
  KEY `a_2` (`a`,`b`),
  KEY `b_2` (`b`,`a`),
  KEY `a_3` (`a`,`data`)
);

SET GLOBAL userstat = 1;

SELECT t.schema, t.name AS table_name
     , i.name AS index_name
  FROM innodb_sys_tables AS t
  JOIN innodb_sys_indexes AS i
    ON t.table_id = i.table_id
 WHERE t.name = 'test'
   AND t.schema = 'test'
 ORDER BY index_name;
```

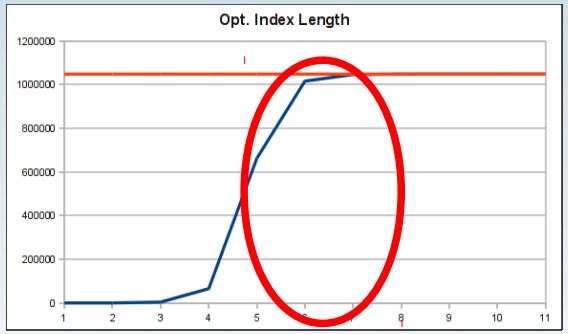
```
+--------+------------+------------+
| schema | table_name | index_name |
+--------+------------+------------+
| test   | test       | a          |
| test   | test       | a_2        |
| test   | test       | a_3        |
| test   | test       | b          |
| test   | test       | b_2        |
| test   | test       | data       |
| test   | test       | PRIMARY    |
+--------+------------+------------+

SELECT table_schema, table_name, index_name
  FROM information_schema.index_statistics
 WHERE table_schema = 'test'
   AND table_name = 'test'
 ORDER BY index_name;

+--------------+------------+------------+
| table_schema | table_name | index_name |
+--------------+------------+------------+
| test         | test       | a          |
| test         | test       | b          |
| test         | test       | b_2        |
| test         | test       | PRIMARY    |
+--------------+------------+------------+
```

- → `a_2`, `a_3` und `data` wurden in der beobachteten Periode nie benutzt!
- Ob der MySQL Optimizer damit recht hatte, ist eine andere Frage!

Indizes optimieren ohne userstat

- Voll redundante Indizes (a, b) + ~~(a, b)~~
- Partiell redundante Indizes (a, b, c) + ~~(a, b)~~
- Achtung! (c, a, b) + ~~(a, b)~~ → (a, b, c)?
- Möglicherweise unnütze Indizes: (gender)
- Überspezifizierter Index (a, b, c, d, ~~e, f, g, h~~)
  - Achtung: Covering Index!
- Index kürzen: (`hash(7)`)

```
SELECT COUNT(DISTINCT LEFT(hash, <n>))
  FROM test;
```



Unnützer Index?

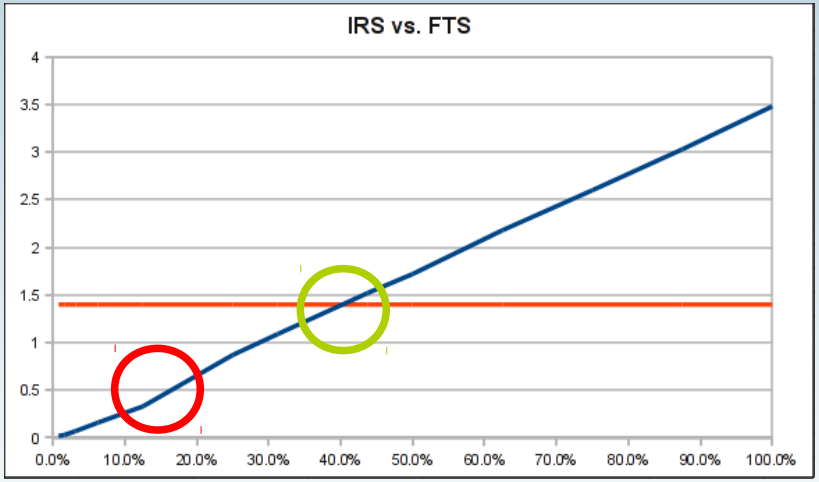
```
EXPLAIN
 SELECT SQL_NO_CACHE *
   FROM test
  WHERE gender = 1;
+----+-------------+-------+------+---------------+--------+---------+-------+--------+-------------+
| id | select_type | table | type | possible_keys | key    | key_len | ref   | rows   | Extra       |
+----+-------------+-------+------+---------------+--------+---------+-------+--------+-------------+
|  1 | SIMPLE      | test  | ref  | gender        | gender | 2       | const | 524465 | Using where |
+----+-------------+-------+------+---------------+--------+---------+-------+--------+-------------+

Laufzeit: 1.0 s

EXPLAIN
 SELECT SQL_NO_CACHE *
   FROM test IGNORE INDEX (gender)
  WHERE gender = 1;

+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE      | test  | ALL  | NULL          | NULL | NULL    | NULL | 1048930 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+

Laufzeit: 0.8 s
```

IRS vs. FTS

- **Index Range Scan vs. Full Table Scan?**

```
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE      | test  | ALL  | id2           | NULL | NULL    | NULL | 1048773 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+

+----+-------------+-------+-------+---------------+------+---------+------+-------+-------------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref  | rows  | Extra       |
+----+-------------+-------+-------+---------------+------+---------+------+-------+-------------+
|  1 | SIMPLE      | test  | range | id2           | id2  | 5       | NULL | 49828 | Using where |
+----+-------------+-------+-------+---------------+------+---------+------+-------+-------------+
```

- **Wo liegt der Break-even?**
  - **MySQL flippt bei <15% :-(**
  - **Zu Untersuchen!**



Warum wird mein Index nicht genutzt?

- Sie fehlen? :-)
- Index: `(a, b, c)` → `WHERE b = ... AND c = ...`
- Index: `(data)` → `WHERE data LIKE '%bla';`
- Index: `(`date`)` → `WHERE YEAR(date) = 2012;`
  - MySQL kennt Function Based Index (FBI) noch nicht (MariaDB Virtual Columns!) → selber machen
- Spalten stimmen nicht überein (Character Set!)
- MySQL Optimizer verschätzt sich!
- Hints wurden verwendet?
  - `use index ()`
  - `force index ()`
  - `ignore index()`

Der Query Execution Plan (QEP)

- **Wie gehen wir sicher, dass/ob der Index verwendet wird?**
  → Optimizer fragen (`EXPLAIN`)

```
EXPLAIN
 SELECT SQL_NO_CACHE *
   FROM test
  WHERE id2 BETWEEN  0 AND ROUND(0 + (14 / 100 * 1310693), 0);

+----+-------------+-------+-------+---------------+------+---------+------+--------+-------------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref  | rows   | Extra       |
+----+-------------+-------+-------+---------------+------+---------+------+--------+-------------+
|  1 | SIMPLE      | test  | range | id2           | id2  | 5       | NULL | 176400 | Using where |
+----+-------------+-------+-------+---------------+------+---------+------+--------+-------------+

EXPLAIN
 SELECT SQL_NO_CACHE *
   FROM test
  WHERE id2 BETWEEN  0 AND ROUND(0 + (15 / 100 * 1310693), 0);

+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE      | test  | ALL  | id2           | NULL | NULL    | NULL | 1048671 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
```

Lesen eines QEP

- Der QEP zeigt uns, wie MySQL plant, die Abfrage auszuführen
  - → `EXPLAIN`
  - Optimizer Trace (neu in 5.6)
- Wie liest man QEP?
  - Meist von oben nach unten
- Type (von oben nach unten langsamer):
  - `eq_ef`
  - `ref`
  - `range` (Index Range Scan)
  - `index_merge` (langsam)
  - `index` (Full Index Scan)

Lesen eines QEP

```
+----+-------------+-------+--------+---------------+---------+---------+----------------------+-------+----------------------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref                  | rows  | Extra                      |
+----+-------------+-------+--------+---------------+---------+---------+----------------------+-------+----------------------------+
|  1 | SIMPLE      | pa    | range  | idx1,idx2     | idx2    | 13      | NULL                 | 40404 | Using where                |
|  1 | SIMPLE      | kd    | ref    | idx4,idx5     | idx4    | 520     | const,const,pa.k_id  |     1 | Using where; Using index |
|  1 | SIMPLE      | k     | eq_ref | PRIMARY       | PRIMARY | 4       | pa.k_id              |     1 | Using where; Using index |
+----+-------------+-------+--------+---------------+---------+---------+----------------------+-------+----------------------------+

+----+-------------+-------+--------+---------------+---------+---------+----------------------+------+----------------------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref                  | rows | Extra                      |
+----+-------------+-------+--------+---------------+---------+---------+----------------------+------+----------------------------+
|  1 | SIMPLE      | pa    | index  | pa_on_k_id    | PRIMARY | 4       | NULL                 |   10 |                            |
|  1 | SIMPLE      | kd    | ref    | idx2,idx1     | idx2    | 520     | const,const,kd_k.p_id |    1 | Using where; Using index |
|  1 | SIMPLE      | k     | eq_ref | PRIMARY       | PRIMARY | 4       | kd_k.p_id            |    1 | Using index                |
+----+-------------+-------+--------+---------------+---------+---------+----------------------+------+----------------------------+

+----+-------------+-------+-------+---------------+-----------+---------+-------------+------+----------------------------+
| id | select_type | table | type  | possible_keys | key       | key_len | ref         | rows | Extra                      |
+----+-------------+-------+-------+---------------+-----------+---------+-------------+------+----------------------------+
|  1 | SIMPLE      | a     | ref   | UN1_a,a_S20   | a_S20     | 19      | const,const |    1 | Using where                |
|  1 | SIMPLE      | b     | merge | b_1,b_S2,b_AC | b_AC1,b_1 | 97,19   | NULL        |  373 | Using intersect(b_AC,b_1) |
+----+-------------+-------+-------+---------------+-----------+---------+-------------+------+----------------------------+
```

Optimizer braucht Statistiken

- Optimizer am besten mit:
  - MyISAM / MEMORY
  - InnoDB :-| (besser mit 5.5/5.6)
  - NDB :-(
- InnoDB: 8 (10, 13?) „random dives“ in (jeden?) InnoDB B+-Tree (für jedes Query?)
- Besser kontrollierbar mit MySQL 5.5
- Persistente Statistiken mit MySQL 5.6
- Ab 5.6 macht `ANALYZE TABLE` wieder Sinn!

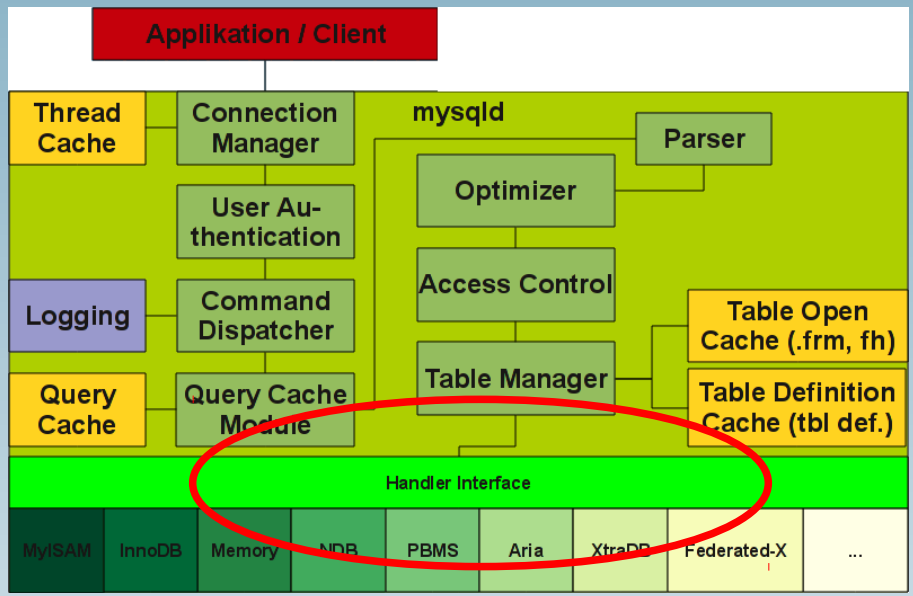
Optimizer Trace

- **Neues Spielzeug mit MySQL 5.6 :-)**

```
SET optimizer_trace = enabled=on;

SELECT SQL_NO_CACHE *
  FROM test
 WHERE id2 BETWEEN 0 AND ROUND(0 + (15 / 100 * 1310693), 0);

SELECT * FROM information_schema.optimizer_trace\G

join_optimization:
  condition_processing:
    original_condition:
      transformation: equality_propagation,
      transformation: constant_propagation,
      transformation: trivial_condition_removal,
  rows_estimation:
    range_analysis:
      table_scan: rows: 1048671, cost: 216624
      potential_range_indices:
        index: PRIMARY, usable: false, cause: not_applicable
        index: gender, usable: false, cause: not_applicable
        index: id2, usable: true, key_parts: [id2]
      analyzing_range_alternatives:
        range_scan_alternatives: rows: 262804, cost: 315366, chosen: false, cause: cost
  considered_execution_plans:
    best_access_path:
      considered_access_paths: access_type: scan, rows: 262804, cost: 164061
      cost_for_plan: 216622, rows_for_plan: 262804, chosen: true
  refine_plan:
    access_type: table_scan
```

Das Handler Interface lesen

```
SHOW GLOBAL STATUS LIKE 'handler_read%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_read_first         | 4549    |
| Handler_read_key           | 3058193 |
| Handler_read_last          | 0       |
| Handler_read_next          | 5108721 |
| Handler_read_prev          | 50      |
| Handler_read_rnd           | 57408   |
| Handler_read_rnd_next      | 3498842 |
+----------------------------+---------+
```



- **The handler_read_* status variables:**

  http://www.fromdual.com/mysql-handler-read-status-variables

Q & A

Fragen ?

Diskussion?

Wir haben noch Zeit für persönliche und indviduelle Beratung

und bieten

Support, Schulung und Betrieb für MySQL