MySQL查询调优实践

David Jiang
Weibo: insidemysql
Gtalk: jiangchengyao@gmail.com

DTCC2012

主题

- B+树索引
- 简单查询优化
  - OLTP
- 复杂查询优化
  - OLAP

关于我

- 近10年MySQL数据库使用经验
  - MySQL 3.23 ~ MySQL 5.6
- InnoSQL分支版本创始人
  - www.innomysql.org
- 独立数据库咨询顾问
  - www.innosql.com
- 《MySQL技术内幕》系列作者
  - 《MySQL技术内幕: InnoDB存储引擎》（已出版）
  - 《MySQL技术内幕: SQL编程》（已出版）
  - 《MySQL技术内幕: 性能调优》（待出版）

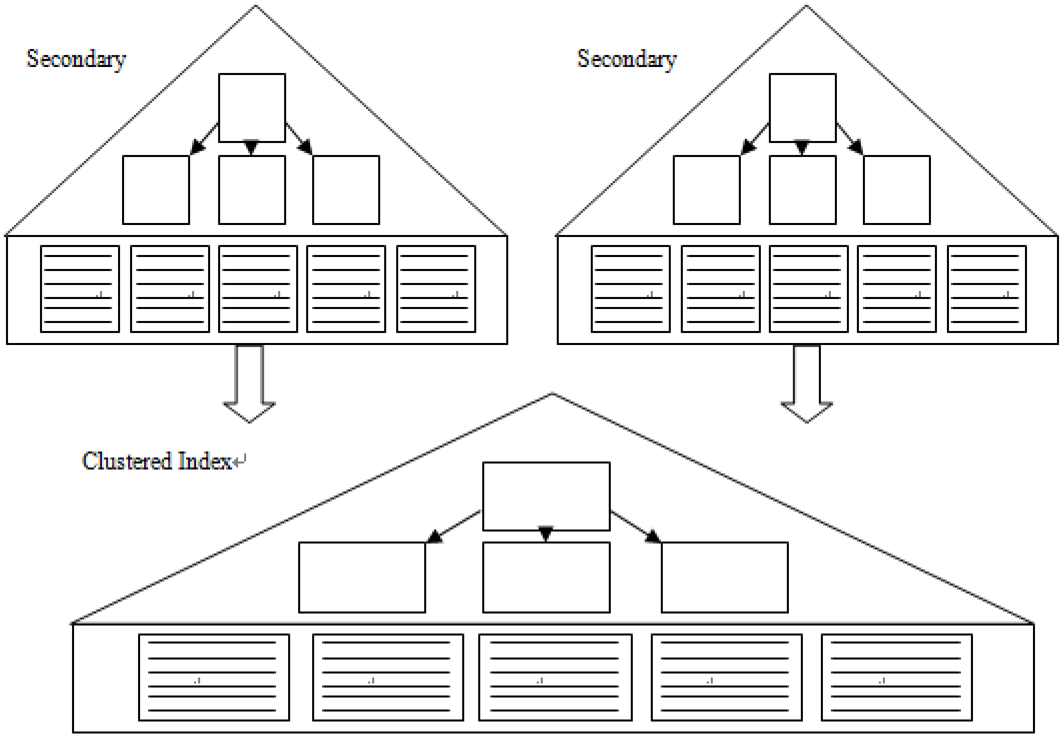
B+树索引

- 常用的索引
  - B+ Tree Index
  - T Tree Index
  - Hash Index
- 什么是索引？
  - 提高查询速度？？？
  - It depends
  - 减少IO次数

B+树索引

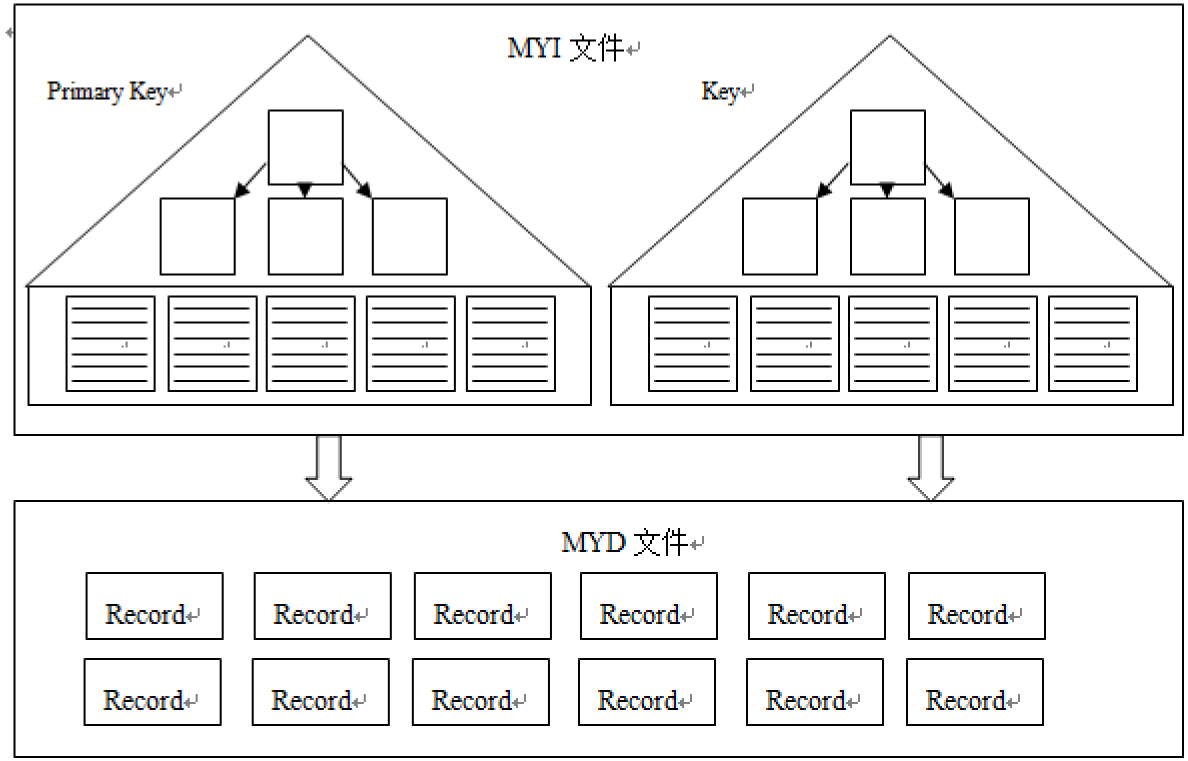
- 聚集索引（Clustered Index）
  - 叶子节点存放整行记录
- 辅助索引（Secondary Index）
  - 叶子节点存放row identifier
    - InnoDB：primary key
      - 书签查找（bookmark lookup）
        - » 查找代价大
    - MyISAM：物理位置（偏移量）
      - 更新代价大
- B+树的高度=> IO次数=>随机IO
  - 3~4层

B+树索引-InnoDB存储引擎



DTCC2012

B+树索引-MyISAM存储引擎



B+树索引

- 聚集索引 VS 辅助索引
  - Clustered index key = 4 bytes
  - Secondary index key = 4 bytes
  - Key pointer = 6 bytes
  - Average row length = 300 bytes
  - Page size = 16K = 16384 bytes
  - Average node occupancy = 70% (both for leaf and index page)
  - Fan-out for clustered index = 16384 * 70% / (4+6) = 1000
  - Fan-out for secondary index = 16384 * 70% / (4+ 6) = 1000
  - Average row per page for clustered index = 16384 * 70% / 300 = 35
  - Average row per page for clustered index = 16384 * 70% / (4 + 6) = 1000

H	Clustered Index	Secondary Index
2	1000*35 = 35,000	1000 * 1000 = 1,000,000
3	$(1000)^2$*35 = 35,000,000	$(1000)^2$*1000 = 1,000,000,000
4	$(1000)^3$*35 = 35,000,000,000	$(1000)^3$*1000 = 1,000,000,000,000

B+树索引

- 辅助索引的优势
  - 树的高度较小=>需要的IO次数少
  - 树的大小较小=>scan需要扫描的页较少
  - 优化器倾向于使用辅助索引
- 辅助索引的劣势
  - 查找完整记录还需查询
    - InnoDB：查询聚集索引
    - MyISAM：直接查找MYD物理位置

B+树索引-InnoDB索引

```
CREATE TABLE UserInfo (
userid INT NOT NULL AUTO_INCREMENT,
username VARCHAR(30),
registdate DATETIME,
email       VARCHAR(50),
PRIMARY KEY (userid),
UNIQUE KEY idx_username (username),
KEY idx_registdate (registdate)
)Engine=InnoDB;
```

```
CREATE TABLE
idx_username_constraint (
username VARCHAR(30),
PRIMARY KEY (username)
);
```

```
CREATE TABLE UserInfo (
userid INT NOT NULL AUTO_INCREMENT,
username VARCHAR(30),
registdate DATETIME,
email       VARCHAR(50),
PRIMARY KEY (userid)
);
```

```
CREATE TABLE idx_username (
userid INT NOT NULL,
username VARCHAR(30),
PRIMARY KEY (username,userid)
);
```

```
CREATE TABLE idx_registdate (
userid INT NOT NULL,
registdate DATETIME),
PRIMARY KEY (registdate,userid)
);
```

B+树索引-InnoDB索引

```
START TRANSACTION;
INSERT INTO UserInfo values (aaa,bbb,ccc);
INSERT INTO idx_username_constraint (bbb);
INSERT INTO idx_username(bbb,aaa);
INSERT INTO idx_registdate(ccc,aaa);
COMMIT;
```

B+树索引-插入顺序问题

- 聚集索引插入
  - 主键是自增长的
  - 插入是顺序的
  - 每页中的填充率高（15/16）
  - 顺序扫描（Scan）可以达到磁盘顺序读的速率
  - 一般不推荐使用UUID
    - 插入非顺序

B+树索引-插入顺序问题

- 辅助索引插入
  - 插入的顺序是乱序的
    - 插入（’David’, ‘Monty’, ‘Jimmy’, ‘Amy’, ‘Michael’ ）
  - 插入的顺序是顺序的
    - 插入时间
  - 需要产生B+树的分裂
    - 需要较大的开销
  - 每页的填充率较低（60%～70%）
  - 顺序扫描不能达到磁盘顺序读的速率
    - 若插入是乱序的

DTCC2012

B+树索引-插入顺序问题

- 辅助索引顺序扫描速度
  - select count(1) from stock
    - KEY `fkey_stock_2` (`s_i_id`), INT
    - Avg row length: 355

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	stock	index	NULL	fkey_stock_2	4	NULL	99833763	Using index

辅助索引：6分42秒

•～4M/秒

•Logical_reads: 12700001 Physical_reads: 100057

强制聚集索引：4分38秒

•～120～130M/秒

•Logical_reads: 14670405 Physical_reads: 2170333

DTCC2012

简单查询优化

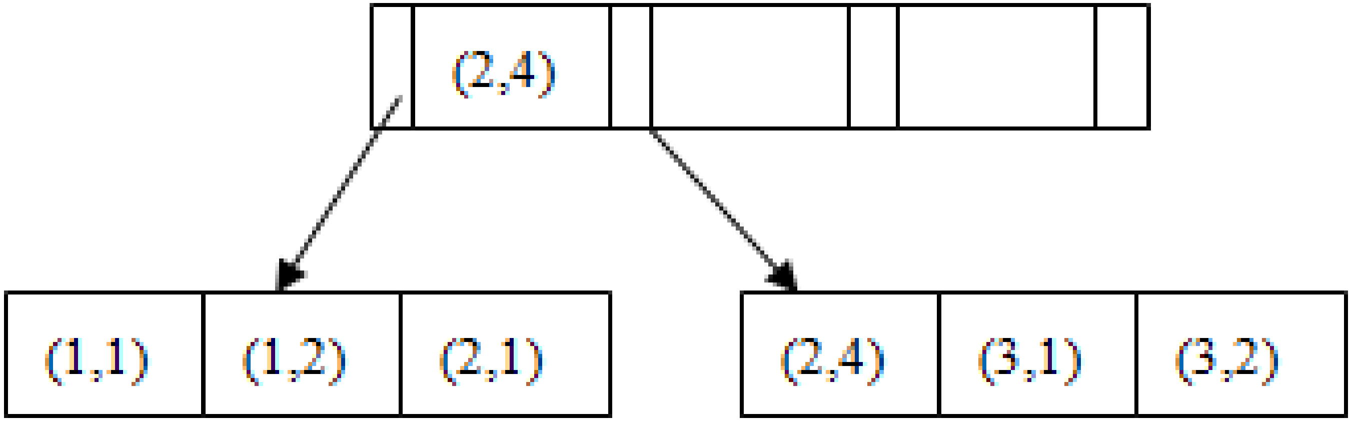
- 简单查询
  - OLTP
- 简单查询特点
  - SQL语句较为简单
  - 返回少部分数据
  - 并发量大
- 优化原则
  - 减少随机读=> 使用索引
    - High Cardinality

简单查询优化

- SELECT … FROM table where **primary_key** = ???
- SELECT … FROM table where **key** = ???

简单查询优化-复合索引

- 索引键值为多个列
  - （a，b）



a : 1,1,2,2,3,3

(a, b): (1,1),(1,2),(2,1),(2,4),(3,1),(3,2)

b: 1,2,1,4,1,2 ✖

简单查询优化-复合索引

- 复合索引(a,b)可被使用于：
  - SELECT * FROM t WHERE a = ?
  - SELECT * FROM t WHERE a = ? AND b = ？
  - SELECT * FROM t WHERE a = ? ORDER BY b
  - 索引覆盖
    - 查询b也可以使用该索引
    - WHERE b = ???

简单查询优化-索引覆盖

- 从辅助索引直接得到结果
  - 不需要书签查找
- （primary key1，primary key2，…，key1，key2，…）
  - SELECT key2 FROM table WHERE key1=xxx；
  - SELECT primary key2,key2 FROM table WHERE key1=xxx；
  - SELECT primary key1,key2 FROM table WHERE key1=xxx；
  - SELECT primary key1,primary key2，key2 FROM table WHERE key1=xxx；

简单查询优化-索引覆盖

```
CREATE TABLE ItemLog(
logId INT NOT NULL AUTO_INCREMENT,
userId VARCHAR(100) NOT NULL,
itemId INT NOT NULL,
date DATETIME NOT NULL,
……
PRIMARY KEY(logId),
KEY idx_userId_date (userId,date)
)ENGINE=INNODB;
```

SELECT COUNT(1) FROM ItemLog WHERE date>='2012-04-01' AND date<'2012-05-01';

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	ItemLog	index	NULL	idx_userId_date	310	NULL	1	Using where; Using index

DTCC2012

简单查询优化-书签查找优化

```
CREATE TABLE UserInfo (
userid INT NOT NULL AUTO_INCREMENT,
username VARCHAR(30),
registdate DATETIME,
email       VARCHAR(50),
PRIMARY KEY (userid),
UNIQUE KEY idx_username (username),
KEY idx_registdate (registdate)
)Engine=InnoDB;
```

SELECT email FROM UserInfo WHERE **username = ‘David’**

If：聚集索引高度：4 && 辅助索引高度：3
Then：　一共需要7次逻辑IO

简单查询优化-书签查找优化

- 分表
  - Like Index Coverage

CREATE TABLE UserInfo (	CREATE TABLE UserInfoDetail (
userid INT NOT NULL AUTO_INCREMENT,	userid INT NOT NULL AUTO_INCREMENT,
username VARCHAR(30),	username VARCHAR(30),
registdate DATETIME,	email VARCHAR(50),
PRIMARY KEY (userid),	PRIMARY KEY (userid),
UNIQUE KEY **idx_username (username)**,	UNIQUE KEY **idx_username (username)**
KEY idex_registdate (registdate)	);
);	

SELECT email FROM UserInfoDetail WHERE **username = ‘David’**

If：辅助索引高度：3

Then：逻辑IO减少为3

DTCC2012

简单查询优化-总结

- 每个页填充率高
  – 包含的记录多
- 减少IO次数
  – IO => 性能
  – OLTP only
- 利用索引覆盖技术避免书签查找
- 利用分表技术避免书签查找

复杂查询优化

- 复杂查询
  - OLAP
    - JOIN
    - Subquery
- 复杂查询特点
  - 数据量大
  - 并发少
  - 需访问较多的数据
  - 索引并不再是唯一的优化方向
  - **调优工作复杂**

复杂查询优化-JOIN

- MySQL JOIN 类型
  - Simple Nested Loops Join
  - Block Nested Loops Join
    - MySQL 5.5+
  - Classic Hash Join
    - MariaDB 5.3+
    - Block Hash Nested Loops Join

复杂查询优化-SNLJ

```
For each tuple r in R do
    For each tuple s in S do
        If r and s satisfy the join condition
            Then output the tuple <r,s>
```

```
For each tuple r in R do
    lookup r join condition in S index
        if found s == r
            Then output the tuple <r,s>
```

Scan cost (no index) = $R_n+R_n \times S_n = O(R_n \times S_n)$

Scan cost (with index) = $R_n + R_n \times S_{BH\,=} = O(R_n)$

复杂查询优化-SNLJ

- INNER JOIN with Index
  - INNER JOIN联接顺序可更改
  - 优化器喜欢选择较小的表作为外部表
  - Scan cost (with index) = $R_n + R_n \times S_{BH} = O(R_n)$

```
SELECT b.emp_no,a.title,a.from_date,a.to_date
FROM titles a
INNER JOIN employees b
on a.emp_no = b.emp_no;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	b	index	PRIMARY	PRIMARY	4	NULL	300363	Using index
1	SIMPLE	a	ref	PRIMARY,emp_no	PRIMARY	4	employees.b.emp_no	1	

```
mysql> SELECT COUNT(1) FROM employees\G;
*************************** 1. row ***************************
count(1): 300024

mysql> SELECT COUNT(1) FROM titles\G;
*************************** 1. row ***************************
count(1): 443308
```

复杂查询优化-SNLJ

```
SELECT b.emp_no,a.title,a.from_date,a.to_date
FROM titles a
STRAIGHT_JOIN employees b
on a.emp_no = b.emp_no;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	ALL	PRIMARY,emp_no	NULL	NULL	NULL	443803	
1	SIMPLE	b	eq_ref	PRIMARY	PRIMARY	4	employees.a.emp_no	1	Using index

复杂查询优化-BNLJ

```
For each tuple r in R do
    store used columns as p from R in join buffer
    For each tuple s in S do
            If p and s satisfy the join condition
                    Then output the tuple <p,s>
```

•The *join_buffer_size* system variable determines the size of each join buffer
•Join buffering can be used when the join is of type ALL or index or range.
•One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
•Only columns of interest to the join are stored in the join buffer, not whole rows.

- For join with no index
- Reduce inner table scan times

复杂查询优化-Example

- Outer table R
  - (1,’a’),(2,’b’),(3,’c’),(4,’d’)
- Inner table S:
  - (1,’2010-01-01’),(2, ’2010-01-01’),(3, ’2010-01-01’)

	NLJ	BNLJ
Outer table scan	1	1
Inner table scan	4	1
Compare times	12	12

复杂查询优化-BNLJ

```
SELECT b.emp_no,a.title,a.from_date,a.to_date
FROM employees_noindex b
INNER JOIN titles_noindex a ON a.emp_no = b.emp_no
WHERE b.birth_date >= '1965-01-01';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	b	ALL	NULL	NULL	NULL	NULL	300629	Using where
1	SIMPLE	a	ALL	NULL	NULL	NULL	NULL	443463	Using where; Using join buffer

复杂查询优化-BNLJ

```
SELECT b.emp_no,a.title,a.from_date,a.to_date
FROM titles_noindex a
LEFT OUTER JOIN employees_noindex b ON a.emp_no = b.emp_no
WHERE b.birth_date >= '1965-01-01';
```

MySQL 5.5 709.645 sec (**MySQL 5.5不支持OUTER JOIN**）

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	b	ALL	NULL	NULL	NULL	NULL	300629	Using where
1	SIMPLE	a	ALL	NULL	NULL	NULL	NULL	443463	

MySQL 5.6 57.483 sec ~12x

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	b	ALL	NULL	NULL	NULL	NULL	300504	Using where
1	SIMPLE	a	ALL	NULL	NULL	NULL	NULL	444828	Using where; Using join buffer (Block Nested Loop)

DTCC2012

复杂查询优化-BHNJ

```
For each tuple r in R do
      store used columns as p from R in join buffer
      build hash table according join buffer
       for each tuple s in S do
                 probe hash table
                 if  find
                              Then output the tuple <r,s>
```

- 通过哈希表减少内部表的比较次数
- Join Buffer Size的大小决定了Classic Hash Join的效率
- 只能用于等值联接

Scan cost $= R_n + S_{n} = O(R_n + S_n)$

复杂查询优化-Example

- Outer table
  - (1,’a’),(2,’b’),(3,’c’),(4,’d’)
- Inner table:
  - (1,’2010-01-01’),(2, ’2010-01-01’),(3, ’2010-01-01’)

	NLJ	BNLJ	BNLJH
Outer table scan	1	1	1
Inner table scan	4	1	1
Compare times	12	12	3

复杂查询优化-BHNJ

```
SELECT MAX(l_extendedprice)
FROM orders, lineitem WHERE
o_orderdate BETWEEN '1995-01-01' AND '1995-01-31' AND
l_orderkey=o_orderkey;
```

MySQL 5.5 ***125.3sec***

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	orders	range	PRIMARY,i_o_orderdate	i_o_orderdate	4	NULL	42008	Using where; Using index
1	SIMPLE	lineitem	ref	PRIMARY,i_l_orderkey,i_l_orderkey_quantity	PRIMARY	4	dbt3.orders.o_orderkey	1	

MariaDB 5.3 ***23.104sec*** ~5x

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	orders	range	PRIMARY,i_o_orderdate	i_o_orderdate	4	NULL	42008	Using where; Using index
1	SIMPLE	lineitem	hash_ALL	PRIMARY,i_l_orderkey,i_l_orderkey_quantity	#hash#PRIMARY	4	dbt3.orders.o_orderkey	5994679	Using join buffer (flat, BNLH join)

DTCC2012

复杂查询优化-BHNJ

	执行速度（秒）	逻辑IO
MySQL 5.5	125.3	157061
MySQL 5.5 （强制使用i_l_orderkey索引）	153.021	406879
MariaDB 5.3 （BNLH）	23.104	6077083

复杂查询优化-JOIN总结

- SNLJ
  - 适合较小表之间的联接
  - 联接的列需包含有索引
- BHNLJ
  - 大表之间的等值联接操作
  - 大表与小表之间的等值联接操作
  - Join Buffer Size的大小决定了内部表的扫描次数
  - 推荐MariaDB作为数据集市或者数据仓库

查询优化-子查询

- 子查询
  - 独立子查询
  - 相关子查询
- MySQL子查询
  - 独立子查询转换为相关子查询（SQL重写 **LAZY**）
    - SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
    - SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
    - Scan cost: O(A+A*B)
  - 优化视情况而定

查询优化-子查询

```
SELECT orderid,customerid,employeeid,orderdate
FROM orders
WHERE orderdate IN
        ( SELECT MAX(orderdate)
                FROM orders
                GROUP BY (DATE_FORMAT(orderdate,'%Y%m'))
        )
```

```
# Time: 111227 23:49:16
# User@Host: root[root] @ localhost [127.0.0.1]
# Query_time: 6.081214  Lock_time: 0.046800 Rows_sent: 42  Rows_examined: 727558 Logical_reads: 91584 Physical_reads: 19
use tpcc;
SET timestamp=1325000956;
SELECT orderid,customerid,employeeid,orderdate
FROM orders
WHERE orderdate IN
        ( SELECT MAX(orderdate)
                FROM orders
                GROUP BY (DATE_FORMAT(orderdate,'%Y%M'))
        );
```

DTCC2012

查询优化-子查询

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	PRIMARY	orders	ALL	NULL	NULL	NULL	NULL	867	Using where
	2	DEPENDENT SUBQUERY	orders	index	NULL	OrderDate	9	NULL	867	Using index; Using temporary; Using filesort

```
SELECT orderid,customerid,employeeid,orderdate
FROM orders AS A
WHERE EXISTS
        ( SELECT *
                FROM orders
                GROUP BY(DATE_FORMAT(orderdate,'%Y%M'))
                HAVING MAX(orderdate)= A.OrderDate
        );
```

查询优化-子查询

```
SELECT orderid,customerid,employeeid,orderdate
FROM orders A
WHERE EXISTS
        ( SELECT * FROM (SELECT MAX(orderdate) AS orderdate
                FROM orders
                GROUP BY (DATE_FORMAT(orderdate,'%Y%M')) ) B
                WHERE A.orderdate = B.orderdate
        );
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	A	ALL	NULL	NULL	NULL	NULL	867	Using where
2	DEPENDENT SUBQUERY	<derived3>	ALL	NULL	NULL	NULL	NULL	23	Using where
3	DERIVED	orders	index	NULL	OrderDate	9	NULL	867	Using index; Using temporary; Using filesort

Time: 111227 23:45:49
User@Host: root[root] @ localhost [127.0.0.1]
Query_time: 0.251133 Lock_time: 0.052001 Rows_sent: 42 Rows_examined: 1729 *Logical_reads: 1923*
Physical_reads: 25

查询优化-子查询

```
SELECT orderid,customerid,employeeid,A.orderdate
FROM orders AS A,
( SELECT MAX(orderdate) AS orderdate
                    FROM orders
                    GROUP BY (DATE_FORMAT(orderdate,'%Y%m'))
) AS B
WHERE A.orderdate = B.orderdate;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	key0	NULL	NULL	NULL	795	Using where
1	PRIMARY	A	ref	OrderDate	OrderDate	9	B.orderdate	1	
2	DERIVED	orders	index	NULL	OrderDate	9	NULL	795	Using index; Using temporary; Using filesort

```
# User@Host: root[root] @ localhost [127.0.0.1]
# Thread_id: 1  Schema: tpcc  QC_hit: No
# Query_time: 0.296897  Lock_time: 0.212167  Rows_sent: 42  Rows_examined: 941 Logical_reads: 1258,
Physical_reads: 28.
```

查询优化-子查询

- MySQL 5.6/MariaDB对子查询的优化
  - 支持对于独立子查询的优化
    - SEMI JOIN
  - 支持各种类型的子查询优化
  - SET optimizer_switch='semijoin=on, materialization=on';

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<subquery2>	ALL	distinct_key	NULL	NULL	NULL	795	100.00	Using where
1	PRIMARY	orders	ref	OrderDate	OrderDate	9	<subquery2>.MAX(orderdate)	1	100.00	
2	SUBQUERY	orders	index	NULL	OrderDate	9	NULL	795	100.00	Using index; Using temporary

User@Host: root[root] @ localhost [127.0.0.1]
Thread_id: 1 Schema: tpcc QC_hit: No
Query_time: 0.176819 Lock_time: 0.147888 Rows_sent: 42 Rows_examined: 1729 Logical_IO: 1927, Physical_IO: 29.

查询优化-子查询总结

- 通过EXPLAIN分析执行计划
- 避免IN => EXISTS转换带来的高额开销
- 避免多次的关联查询操作
- 使用MySQL 5.6/MariaDB 5.3处理复杂子查询操作
  - SQL优化器自动优化

参考资料

- http://dev.mysql.com/doc/
- http://kb.askmonty.org/en/
- 《MySQL技术内幕》
  - 《MySQL技术内幕：InnoDB存储引擎》
  - 《MySQL技术内幕：SQL编程》

Q&A