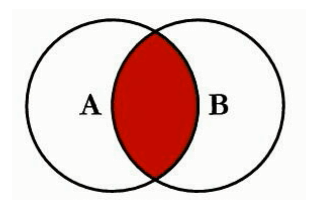
## Join

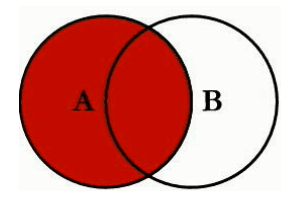
1. inner join

A、B共有，也就是交集。



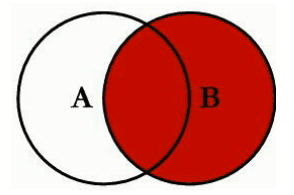
1. left jion

A独有+AB共有（交集）

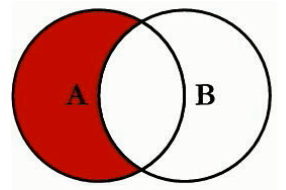


1. right jion

B独有+AB共有（交集）

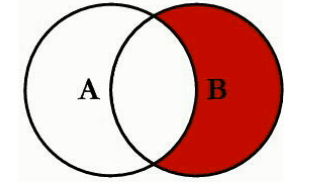


1. A独有



注：参照left join，A独有只是将AB交集部分去掉。

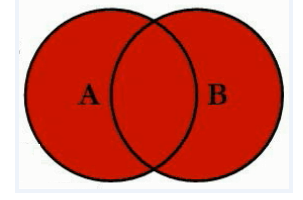
1. B独有

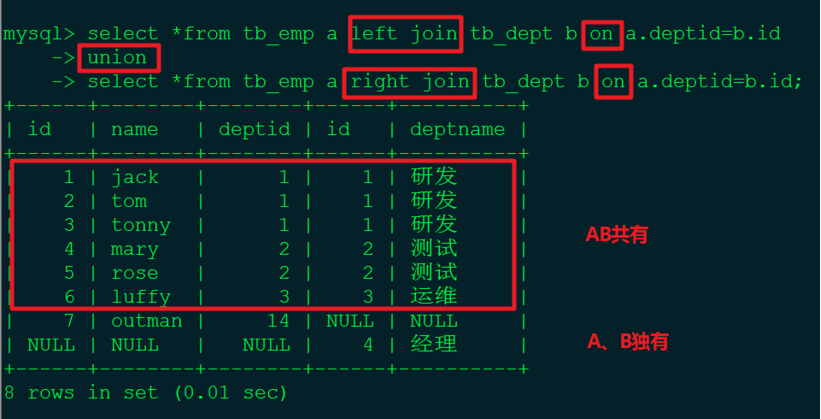


注：参照right join，B独有只是将AB交集部分去掉。

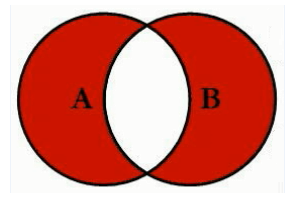
1. AB全有（并集）

由于mysql中不支持full outer join，所以这里通过union进行转换。AB并集：AB交集+A独有+B独有

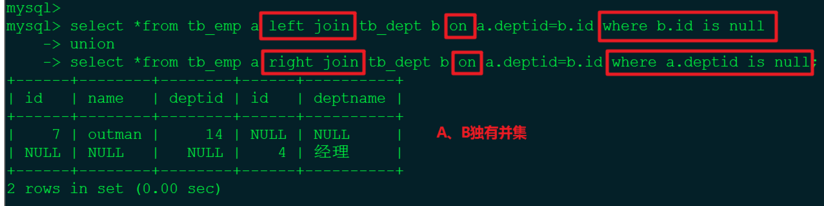




1. A、B独有并集



A、B独有并集，相当于A、B全有去掉AB的共有（交集）。



总结：这里主要对MySQL中join语句的7中用法进行了总结，主要注意MySQL不支持full outer join，所以需要对其进行转换变形，最终达到效果。

# 索引

1. 索引优缺点

优点

1. 类似大学图书馆的书目索引，提高数据的检索效率，降低数据库的IO成本。
2. 通过索引列对数据进行排序，降低数据的排序成本，从而降低CPU的消耗。

缺点

1. 索引实际上也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也要占用空间。
2. 虽然索引大大提高了查询效率，但是降低了更新表的速度，如insert、update和delete操作。因为更新表时，MySQL不仅要保存数据，还要保存索引文件每次更新的索引列字段，并且在更新操作后，会更新相应字段索引的信息。
3. 索引只是提高查询效率的一个因素，如果你的MySQL有大量的数据表，就需要花时间研究建立最优秀的索引或优化查询语句。
4. 索引分类
5. 单值索引：一个索引只包含单个列，一个表可以有多个单值索引。
6. 唯一索引：索引列的值必须唯一，但允许有空值，主键就是唯一索引。
7. 复合索引：一个索引包含多个列。
8. 索引结构
9. BTREE索引；
10. Hash索引；
11. Full-Text索引；
12. R-Tree索引。
13. 基本语法
14. 创建索引

create [unique] index indexname on tablename(columnname(length));

alter table tablename add index indexname (columnname(length));

注：如果是char、varchar类型的字段，length可以小于字段实际长度；如果是blob、text类型，必须指定length。

1. 删除索引

drop index indexname on tablename;

1. 查看索引

show index from tablename;

1. 其他创建索引的方式
2. 添加主键索引

ALTER TABLE `table\_name` ADD PRIMARY KEY (`column`)

1. 添加唯一索引

ALTER TABLE `table\_name` ADD UNIQUE (`column`)

1. 添加全文索引

ALTER TABLE `table\_name` ADD FULLTEXT (`column`)

1. 添加普通索引

ALTER TABLE `table\_name` ADD INDEX index\_name (`column` )

1. 添加组合索引

ALTER TABLE `table\_name` ADD INDEX index\_name (`column1`, `column2`, `column3`)

1. 建立索引的一般情景
2. 需建立索引的情况
3. 主键自动建立唯一索引；
4. 频繁作为查询条件的字段；
5. 查询中与其他表关联的字段，外键关系建立索引；
6. 高并发下趋向创建组合索引；
7. 查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度；
8. 查询中统计或分组字段；
9. 不需要创建索引的情况
10. 表记录太少。（数据量太少MySQL自己就可以搞定了）；
11. 经常增删改的表；
12. 数据重复且平均分配的字段，如国籍、性别，不适合创建索引；
13. 频繁更新的字段不适合建立索引；
14. Where条件里用不到的字段不创建索引；

# 执行计划（Explain）

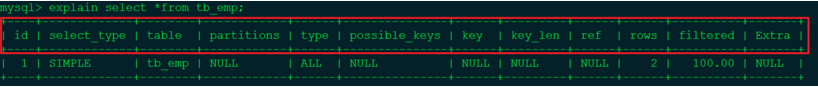
explain（执行计划），使用explain关键字可以模拟优化器执行sql查询语句，从而知道MySQL是如何处理sql语句。explain主要用于分析查询语句或表结构的性能瓶颈。

1. Explain的作用

通过explain+sql语句可以知道如下内容：

1. 表的读取顺序。（对应id）
2. 数据读取操作的操作类型。（对应select\_type）
3. 哪些索引可以使用。（对应possible\_keys）
4. 哪些索引被实际使用。（对应key）
5. 表直接的引用。（对应ref）
6. 每张表有多少行被优化器查询。（对应rows）
7. Explain包含的信息

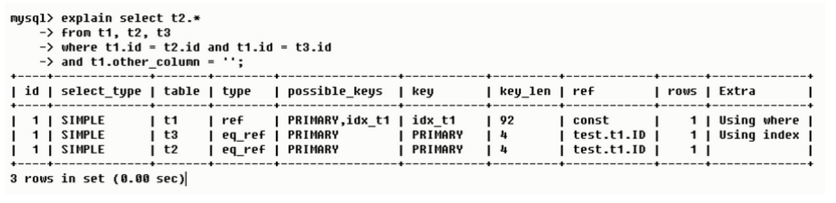
explain使用：explain+sql语句，通过执行explain可以获得sql语句执行的相关信息。



1. id

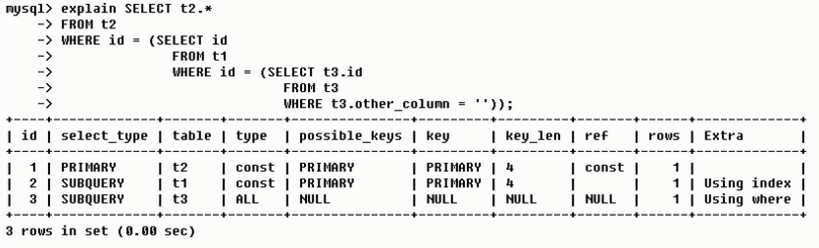
select查询的序列号，包含一组数字，表示查询中执行select子句或操作表的顺序，该字段通常与table字段搭配来分析。

1. id相同，执行顺序从上到下



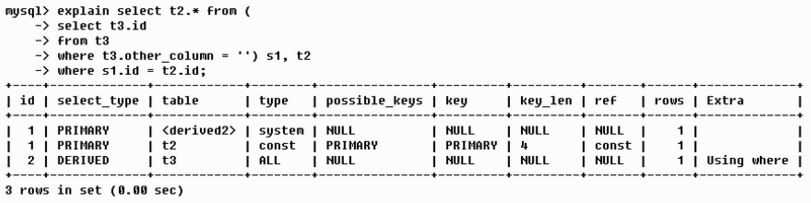
id相同，执行顺序从上到下，搭配table列进行观察可知，执行顺序为t1->t3->t2。

1. id不同，如果是子查询，id的序号会递增，id值越大执行优先级越高



如果是子查询id的序号会递增，id值越大执行优先级越高，搭配table列可知，执行顺序为t3->t1->t2。

1. id相同和不同的情况同时存在



id如果相同，可认为是同一组，执行顺序从上到下。在所有组中，id值越大执行优先级越高。所以执行顺序为t3->derived2(衍生表，也可以说临时表)->t2。

**总结**：id的值表示select子句或表的执行顺序，id相同，执行顺序从上到下，id不同，值越大的执行优先级越高。

1. select\_type

查询的类型，主要用于区别普通查询、联合查询、子查询等复杂的查询。其值主要有六个：

1. SIMPLE

简单的select查询，查询中不包含子查询或union查询。

1. PRIMARY

查询中若包含任何复杂的子部分，最外层查询为PRIMARY，也就是最后加载的就是PRIMARY。

1. SUBQUERY

在select或where列表中包含了子查询，就为被标记为SUBQUERY。

1. DERIVED

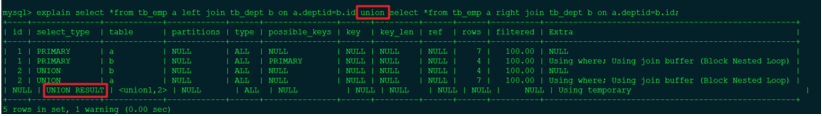
在from列表中包含的子查询会被标记为DERIVED(衍生)，MySQL会递归执行这些子查询，将结果放在临时表中。

1. UNION

若第二个select出现在union后，则被标记为UNION，若union包含在from子句的子查询中，外层select将被标记为DERIVED。

1. UNION RESULT

从union表获取结果的select。



1. table

显示sql操作属于哪张表的。

1. partitions

官方定义为The matching partitions（匹配的分区），该字段应该是看table所在的分区吧（不晓得理解错误没）。值为NULL表示表未被分区。

1. type

表示查询所使用的访问类型，type的值主要有八种，该值表示查询的sql语句好坏，从最好到最差依次为：system>const>eq\_ref>ref>range>index>ALL。

要详细了解type取值的作用，需要用数据说话。创建tb\_emp（员工表）和tb\_dept（部门表）。

1. tb\_emp表

DROP TABLE IF EXISTS `tb\_emp`;

CREATE TABLE `tb\_emp` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(20) NOT NULL,

`deptid` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `idx\_tb\_emp\_name` (`name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `tb\_emp`(name,deptid) VALUES ('jack', '1');

INSERT INTO `tb\_emp`(name,deptid) VALUES ('tom', '1');

INSERT INTO `tb\_emp`(name,deptid) VALUES ('tonny', '1');

INSERT INTO `tb\_emp`(name,deptid) VALUES ('mary', '2');

INSERT INTO `tb\_emp`(name,deptid) VALUES ('rose', '2');

INSERT INTO `tb\_emp`(name,deptid) VALUES ('luffy', '3');

INSERT INTO `tb\_emp`(name,deptid) VALUES ('outman', '4');

1. tb\_dept表

DROP TABLE IF EXISTS `tb\_dept`;

CREATE TABLE `tb\_dept` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`deptname` varchar(20) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `tb\_dept`(deptname) VALUES ('研发');

INSERT INTO `tb\_dept`(deptname) VALUES ('测试');

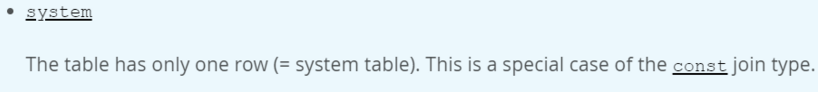
INSERT INTO `tb\_dept`(deptname) VALUES ('运维');

INSERT INTO `tb\_dept`(deptname) VALUES ('经理');

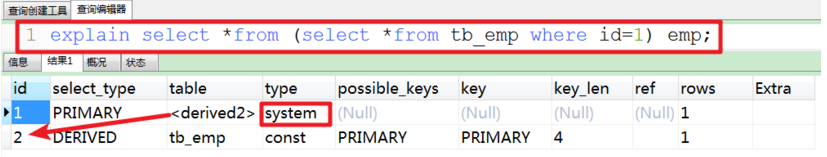
1. system

表只有一行记录（等于系统表），是const的特例类型，平时不会出现，可以忽略不计。

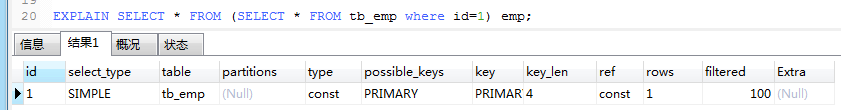
但是笔者发现在MySQL5.7.22时，不会出现该字段值，只能出现const，但是在MySQL5.7版本以下可以出现该情况。猜测MySQL5.7版本是不是进行了优化，因为system官网的解释：



5.5.48：



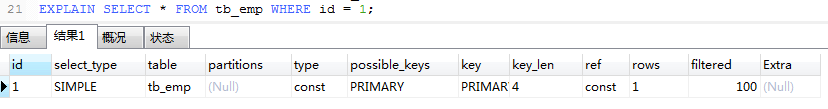
5.7.19：



注：两个引擎的执行信息不一样，5.5.48执行过程中产生了临时表（DERIVED），5.7.19为简单查询。

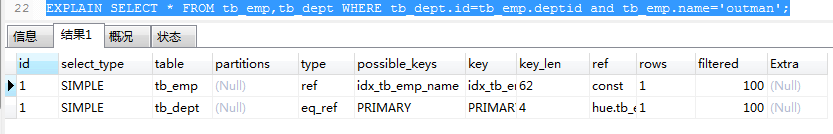
1. const

表示通过一次索引就找到了结果，常出现于primary key或unique索引。因为只匹配一行数据，所以查询非常快。如将主键置于where条件中，MySQL就能将查询转换为一个常量。



1. eq\_ref

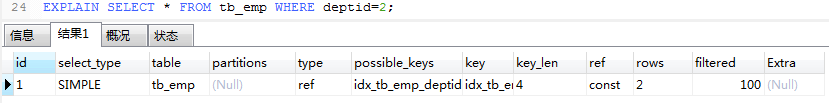
唯一索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见主键或唯一索引扫描。



1. ref

非唯一性索引扫描，返回匹配某个单独值的所有行。本质上也是一种索引访问，返回匹配某值（某条件）的多行值，属于查找和扫描的混合体。由于是非唯一性索引扫描，所以对tb\_emp表的deptid字段创建索引：

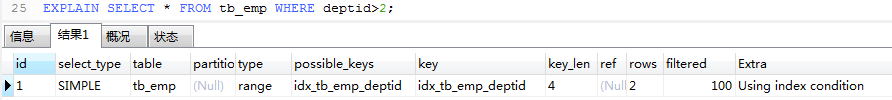
create index idx\_tb\_emp\_deptid on tb\_emp(deptid);



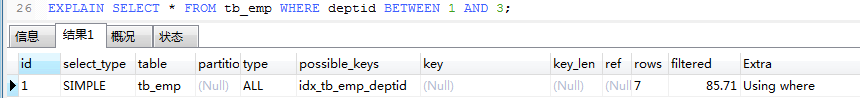
1. range

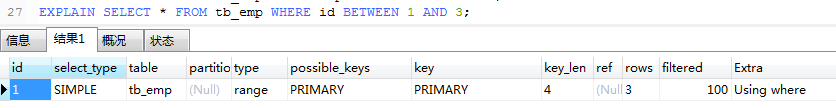
只检索给定范围的行，使用一个索引来检索行，可以在key列中查看使用的索引，一般出现在where语句的条件中，如使用between、>、<、in等查询。

这种索引的范围扫描比全表扫描要好，因为索引的开始点和结束点都固定，不用扫描全索引。



虽然我们为deptid字段创建了索引并在where中使用了between等，但在如下情况type仍为ALL。



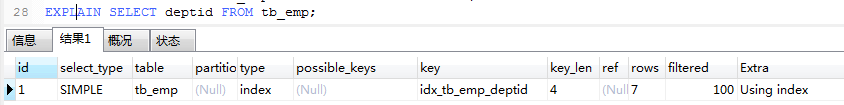


对比两图，可以看到使用deptid和id进行操作，其type的值一个是ALL也就是进行了全表扫描，一个是range进行了指定索引范围值检索。可能原因deptid并不是唯一索引。

对于以上问题，需要具体问题具体分析，并不能一概而论。

1. index

全索引扫描，index和ALL的区别：index只遍历索引树，通常比ALL快，因为索引文件通常比数据文件小。虽说index和ALL都是全表扫描，但是index是从索引中读取，ALL是从磁盘中读取。



1. all

全表扫描

注：一般来说，需保证查询至少达到range级别，最好能达到ref。

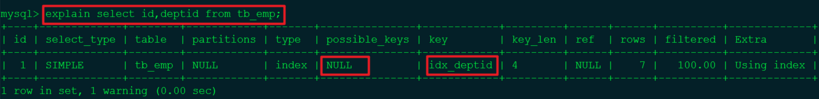
1. possible\_keys和key、key\_len

possible\_keys：显示可能应用在表中的索引，可能一个或多个。查询涉及到的字段若存在索引，则该索引将被列出，但不一定被查询实际使用。

key：实际中使用的索引，如为NULL，则表示未使用索引。若查询中使用了覆盖索引，则该索引和查询的select字段重叠。

key\_len：表示索引中所使用的字节数，可通过该列计算查询中使用的索引长度。在不损失精确性的情况下，长度越短越好。key\_len显示的值为索引字段的最大可能长度，并非实际使用长度，即key\_len是根据表定义计算而得，并不是通过表内检索出的。

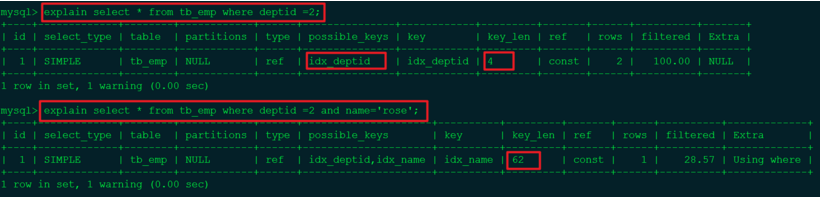
简单理解：possible\_keys表示理论上可能用到的索引，key表示实际中使用的索引。



possible\_keys为NULL表示可能未用到索引，但key=idx\_deptid表示在实际查询的过程中进行了索引的全扫描。

通过下面的例子来理解key\_len，首先为name字段创建索引：

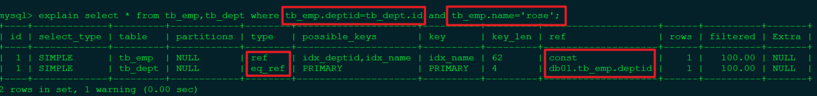
create index idx\_name on tb\_emp(name);



注：在使用索引查询时，当条件越精确，key\_len的长度可能会越长，所以在不影响结果的情况下，key\_len的值越短越好。

1. ref

显示关联的字段。如果使用常数等值查询，则显示const，如果是连接查询，则会显示关联的字段。



注：由于id相同，因此从上到下执行：

#1.tb\_emp表为非唯一性索引扫描，实际使用的索引列为idx\_name，由于tb\_emp.name='rose'为一个常量，所以ref=const。

#2.tb\_dept为唯一索引扫描，从sql语句可以看出，实际使用了PRIMARY主键索引，ref=db01.tb\_emp.deptid表示关联了db01数据库中tb\_emp表的deptid字段。

1. rows

根据表统计信息及索引选用情况大致估算出找到所需记录所要读取的行数。当然该值越小越好。

1. filtered

百分比值，表示存储引擎返回的数据经过滤后，剩下多少满足查询条件记录数量的比例。

1. extra

显示十分重要的额外信息。其取值有以下几个：

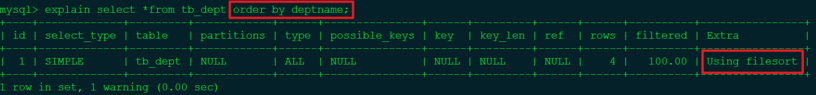
1. Using filesort

Using filesort表明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。

mysql中无法利用索引完成的排序操作称为“文件排序”。

出现Using filesort就非常危险了，在数据量非常大的时候几乎“九死一生”。出现Using filesort尽快优化sql语句。

deptname字段未建索引的情况。

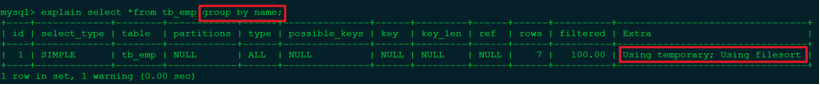


为deptname字段创建索引后。

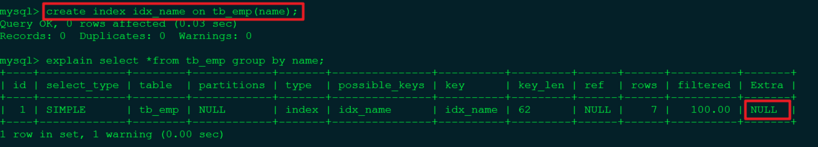
1. Using temporary

使用了临时表保存中间结果，常见于排序order by和分组查询group by。非常危险，“十死无生”，急需优化。

将tb\_emp中name的索引先删除，出现如下图结果，非常烂，Using filesort和Using temporary，“十死无生”。



为name字段创建索引后。



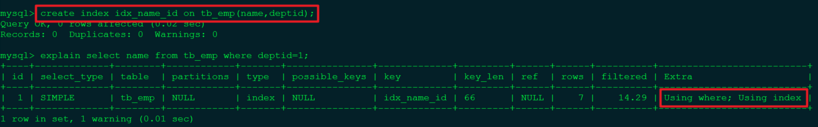
1. Using index

表明相应的select操作中使用了覆盖索引，避免访问表的额外数据行，效率不错。

如果同时出现了Using where，表明索引被用来执行索引键值的查找。（where deptid=1）

如果没有同时出现Using where，表明索引用来读取数据而非执行查找动作。

删除tb\_emp表中name和deptid字段的单独索引，创建复合索引。

****

从这里给出覆盖索引的定义：select的数据列只从索引中就能取得数据，不必读取数据行。通过上面的例子理解：创建了（name，deptid）的复合索引，查询的时候也使用复合索引或部分，这就形成了覆盖索引。简记：查询使用复合索引，并且查询的列就是索引列，不能多，个数需对应。

使用优先级Using index>Using filesort（九死一生）>Using temporary（十死无生）。也就说出现后面两项表明sql语句是非常烂的，急需优化！！！

1. 总结

explain（执行计划）包含的信息十分的丰富，着重关注以下几个字段信息。

1. id，select子句或表执行顺序，id相同，从上到下执行，id不同，id值越大，执行优先级越高。
2. type，type主要取值及其表示sql的好坏程度（由好到差排序）：

**system** > **const** >  **eq\_ref**  > **ref** > **range** > **index** > **ALL** 。

保证range，最好到ref。

1. key，实际被使用的索引列。
2. ref，关联的字段，常量等值查询，显示为const，如果为连接查询，显示关联的字段。
3. Extra，额外信息，使用优先级Using index>Using filesort（九死一生）>Using temporary（十死无生）。

着重关注上述五个字段信息，对日常生产过程中调优十分有用。