# MYSQL5.7新特性

写在前面  
   MySQL 5.7版本于2015年10月份左右 GA，至今已经半年多了，但自己一直没有时间来follow MySQL 5.7 新的特性，作为MySQL DBA 实在汗颜，以后会花时间来研究5.7 版本的特性并针对部分优化功能做出压力测试。本系列基于5.7.12 版本来讲述MySQL的新特性,同时也建议大家跟踪官方blog和[**文档**](http://dev.mysql.com/doc/refman/5.7/en/)，以尽快知悉其新的变化。 **一 安全性**

   MySQL 5.7 的目标是成为发布以来最安全的 MySQL 服务器，其在 SSL/TLS 和全面安全开发方面有一些重要的改变。

**1.1** **5.7版本的用户表mysql.user要求plugin字段非空**，且默认值是mysql\_native\_password，并且不再支持mysql\_old\_password。

**1.2 增加密码过期功能**，dba可以设置任何用户的密码过期时间，具体详见官方文档《[密码过期策略](http://dev.mysql.com/doc/refman/5.7/en/password-expiration-policy.html)》

1. 设置 用户的过期时间为 90天
2. ALTER USER yangyi@'127.0.0.1' PASSWORD EXPIRE INTERVAL 90 DAY;
3. 设置 用户密码永远不过期
4. ALTER USER yangyi@'127.0.0.1' PASSWORD EXPIRE NEVER;

可能大家都有如果业务系统的账号某一天突然过期了，业务受到影响 怎么办？ 别担心那么可以设置密码永不过期。

1. default\_password\_lifetime=0

**1.3 DBA可以通过对用户加锁/ 解锁进一步控制其访问db 。**

例子

1. mysql> alter user yang@'%' account lock;
2. Query OK, 0 rows affected (0.00 sec)
3. root@rac4:/srv/my3306/data# >mysql -uyang -h127.0.0.1 -p
4. Enter password:
5. ERROR 3118 (HY000): Access denied for user 'yang'@'localhost'. Account is locked.
6. mysql> alter user yang@'%' account unlock;
7. Query OK, 0 rows affected (0.00 sec)

**1.4 SSL 特性** MySQL 5.7版本提供了更为简单SSL安全访问配置，并且默认连接就采用SSL的加密方式。细心的朋友在看官方文档的时候  会注意到 安装步骤中多了一安装SSL的步骤，具体关于SSL是什么，请参考[**《图解SSL和加密》**](http://www.linuxidc.com/Linux/2016-04/130034.htm)

1. shell> bin/mysql\_ssl\_rsa\_setup # MySQL 5.7.6 and up

推荐两篇延伸阅读，关于SSL 特性我相信绝大多数数据库都没有开启，姜承尧文章中的测试案例显示弱开启SSL 性能开销在25% 左右。大家在尝试新的特性的时候 ，根据自己的业务做评估。  
《[MySQL的SSL加密连接与性能开销](https://mp.weixin.qq.com/s?__biz=MjM5MjIxNDA4NA==&mid=400683438&idx=1&sn=7637f79054f80f5c911b6c7faf6e89da&scene=1&srcid=0511n2vBrgcBlr2ZA3jyxOBG&key=b28b03434249256b1540382bc1000e1ce256eb698f88c5abe17704930be107e65093b9f4c5a9ac49790ee4479f827d0f&ascene=0&uin=MTYwMzMzNTIwMA%3D%3D&devicetype=iMac+MacBookPro12%2C1+OSX+OSX+10.11.2+build(15C50)&version=11020201&pass_ticket=WAiKeXr2YKf%2BOCuNjtbwfGvDz8mnMS0gWjLzvw2BcZ10%2B9FCIbpuweJTB%2F3pGN0A)》

《[SSL/TLS in MySQL 5.7](http://mysqlblog.fivefarmers.com/2015/04/09/ssltls-in-mysql-5-7/)》

**1.5 使用更安全的初始化方式**  
逐步废弃mysql\_install\_db的安装方式使用 initialize代替(mysql\_install\_db <5.7.6<= mysqld  —initialize) ,使用  initialize 参数初始化数据库有如下特性

a 只创建一个 root账号，并且生成一个临时的标记为过期密码

b 不创建其他账号

c 不创建test 数据库

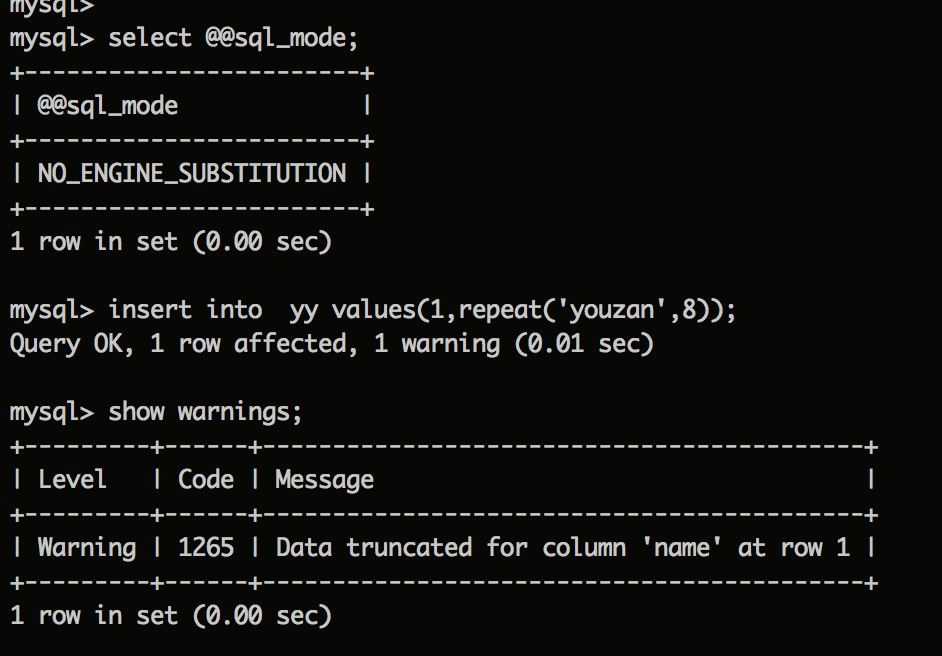
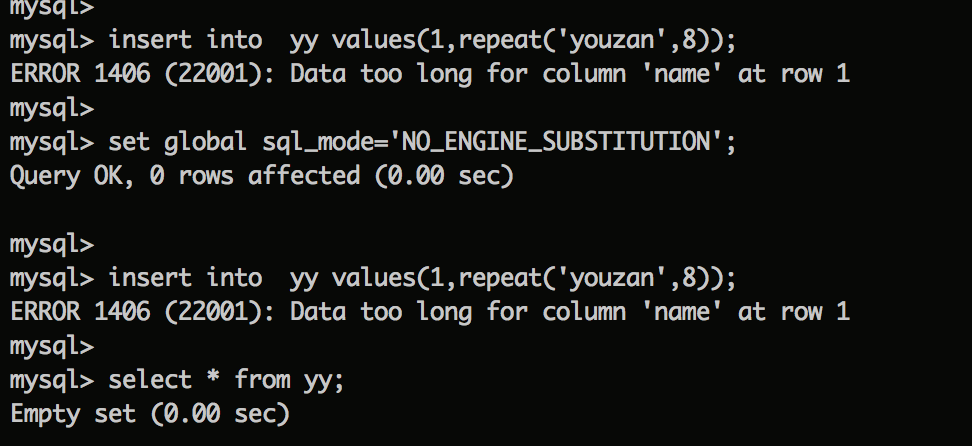
特别强调与—initialize 不同 ,初始化的时候 使用参数

shell> **bin/mysqld --initialize-insecure --user=mysql**

则会生成一个无密码的root 账号。

**1 SQL\_MODE的变化**  
官方文档上表述, 5.7 版本默认的SQL\_MODE="ONLY\_FULL\_GROUP\_BY, STRICT\_TRANS\_TABLES, NO\_ZERO\_IN\_DATE, NO\_ZERO\_DATE, ERROR\_FOR\_DIVISION\_BY\_ZERO, NO\_AUTO\_CREATE\_USER, and NO\_ENGINE\_SUBSTITUTION" 。而实际上我自己的测试环境中默认添加了 STRICT\_TRANS\_TABLES 属性。

1. mysql> select @@sql\_mode;
2. +--------------------------------------------+
3. | @@sql\_mode |
4. +--------------------------------------------+
5. | STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION |
6. +--------------------------------------------+
7. 1 row in set (0.00 sec)

 STRICT\_TRANS\_TABLES 意思是说要存储的字段的长度大于字段定义的大小，直接报错而非像5.6版本以及之前，截断数据进行存储，同时抛出一个warning。注意同一个会话调整 sql\_mode 必须退出之后在进入sql\_mode才会生效。详细了解SQL\_MODE 请移步《[sql\_mode官方文档](http://dev.mysql.com/doc/refman/5.7/en/sql-mode.html)》 注意5.7 对于null，'00000000 00:00:00' 这类default值的影响.  
  
  
  
**2 online ddl支持 rename index name** .个人感觉这个功能比较鸡肋，通常改变索引名称的时候 本身索引的结构需要增删字段，新建的索引名称也要修改合适的名称。

1. mysql> alter table yy rename index idx\_name to idx\_nm;
2. Query OK, 0 rows affected (0.01 sec)
3. Records: 0 Duplicates: 0 Warnings: 0

**3 新增内置的full text 插件**，支持中文 ，韩文，日文全文索引。  
  之前的版本 只能依赖单词之间空格进行分词，对于依赖于语义分词而非空格分词的其他语言种类，5.7 版本的引入支持解析中文，韩文，日文的全文索引--ngram full-text parser解决了该问题。具体请移步《[ngram Full-Text Parser](http://dev.mysql.com/doc/refman/5.7/en/fulltext-search-ngram.html)》  
**3.1 动态修改varchar 长度大小。**  
  可以可以通过ALTER TABLE 语句以in place方式修改varchar的大小且无需table-copy。但存在限制:表示varchar 长度的字节数不能变化(如果变更前使用1个字节表示长度，变更后也必须使用1个字节表示)，即只支持0~255内的或者255以上的范围变更(增大)，如果字段的长度从254增到256时就不能使用in-place算法，必须使用copy算法，否侧报错.需要注意的是 减小 varchar(N)长度的大小必须使用copy类型，如：

1. mysql> alter table yy change column name name varchar(256) ,algorithm=inplace;
2. ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.
3. mysql> alter table yy change column name name varchar(255) ,algorithm=inplace;
4. Query OK, 0 rows affected (0.05 sec)
5. Records: 0 Duplicates: 0 Warnings: 0
6. mysql> alter table yy change column name name varchar(256) ,algorithm=copy;
7. Query OK, 1 row affected (0.10 sec)
8. Records: 1 Duplicates: 0 Warnings: 0
9. mysql> alter table yy change column name name varchar(100) ,algorithm=inplace;
10. ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.

**3.2 临时表性能优化**  
MySQL 5.7对临时表做了极大的改动，提升性能。  
通过优化 CREATE TABLE, DROP TABLE, TRUNCATE TABLE,和ALTER TABLE 语句的执行逻辑，提升临时表的性能。(这个是从官网翻译的，还没找到除了alter之外的其他资料说明详细的优化过程)  
InnoDB临时表元数据不再存储于InnoDB系统表而是存储在INNODB\_TEMP\_TABLE\_INFO，包含所有用户和系统创建的临时表信息。该表在第一次在其上运行select时被创建。

1. mysql> desc information\_schema.INNODB\_TEMP\_TABLE\_INFO;
2. +----------------------+---------------------+------+-----+---------+-------+
3. | Field                | Type                | Null | Key | Default | Extra |
4. +----------------------+---------------------+------+-----+---------+-------+
5. | TABLE\_ID             | bigint(21) unsigned | NO   |     | 0       |       |
6. | NAME                 | varchar(202)        | YES  |     | NULL    |       |
7. | N\_COLS               | int(11) unsigned    | NO   |     | 0       |       |
8. | SPACE                | int(11) unsigned    | NO   |     | 0       |       |
9. | PER\_TABLE\_TABLESPACE | varchar(64)         | YES  |     | NULL    |       |
10. | IS\_COMPRESSED        | varchar(64)         | YES  |     | NULL    |       |
11. +----------------------+---------------------+------+-----+---------+-------+
12. 6 rows in set (0.00 sec)

**3.3 新增**[**临时表空间**](https://dev.mysql.com/doc/refman/5.7/en/internal-temporary-tables.html)   
 为所有 非压缩的innodb临时表提供一个独立的表空间，默认的临时表空间文件为ibtmp1，位于数据目录。我们可通过innodb\_temp\_data\_file\_path参数指定临时表空间的路径和大小。

1. mysql> show global variables like 'innodb\_temp\_data\_file\_path';
2. +----------------------------+-----------------------+
3. | Variable\_name              | Value                 |
4. +----------------------------+-----------------------+
5. | innodb\_temp\_data\_file\_path | ibtmp1:12M:autoextend |
6. +----------------------------+-----------------------+
7. 1 row in set (0.00 sec)

MySQL每次重新启动时，会重新创建临时表空间。  
**注意**从5.7.5开始，新增一个系统选项 internal\_tmp\_disk\_storage\_engine 可定义**磁盘临时表的引擎类型为 InnoDB**，而在这以前，只能使用 MyISAM。而在5.6.3以后新增的参数default\_tmp\_storage\_engine是控制create temporary table创建的**临时表的存储引擎**，在以前默认是MEMORY，不要把这二者混淆了。

1. mysql> show global variables like '%storage\_engine%';
2. +----------------------------------+--------+
3. | Variable\_name                    | Value |
4. +----------------------------------+--------+
5. | default\_storage\_engine           | InnoDB |
6. | default\_tmp\_storage\_engine       | InnoDB |
7. | disabled\_storage\_engines         |        |
8. | internal\_tmp\_disk\_storage\_engine | InnoDB |
9. +----------------------------------+--------+

**3.4 引入新的"non-redo" undo log** 详见《[innodb-temporary-table-undo-logs](https://dev.mysql.com/doc/refman/5.7/en/innodb-temporary-table-undo-logs.html)》  
从MySQL 5.7.2 针对临时表及相关对象引入新的"non-redo" undo log，存放于临时表空间。该类型的undo log非 redolog， 因为临时表在数据库崩溃后不需要恢复，也就无需redo logs，避免了写relog相关的io，从而提高了性能。必须指出出操作临时表需要 undo log用于MySQL运行时的回滚、MVCC等。  
**3.5 支持新的DATA\_GEOMETRY空间类型的数据**  
InnoDB现在支持MySQL-supported空间数据类型。也即，之前的空间数据是以binary BLOB数据存储的，现在空间数据类型被映射到了一个InnoDB内部数据类型DATA\_GEOMETRY.  
**3.6 升级innochecksum**  
    innochecksum--离线的InnoDB文件校验工具，新增新的选择项或扩展的功能，如可指定特定的校验算法、可以只重写校验值而不进行验证、可指定允许的校验和不匹配量、显示各类页的个数、导出页类型信息、输出至日志、从标准输入读取数据等。从 5.7.2 起可支持校验超过2G的文件。详细的用法参考《[innochecksum 官方文档](http://dev.mysql.com/doc/refman/5.7/en/innochecksum.html)》  
**3.7 online DDL语句重建普通表和分区表**OPTIMIZE TABLE、ALTER TABLE … FORCE、ALTER TABLE … ENGINE=INNODB等操作支持支持使用inplace算法。减少了重建时间和对应用的影响。  
**3.8 针对Fusion-io NVM 文件系统的优化**  
   Linux系统中Fusion-io Non-Volatile Memory (NVM)文件系统提供了原子写能力，使InnoDB的doublewrite变得冗余。因此，MySQL5.7.4以后，如果Fusion-io设备支持原子写， MySQL系统会自动关闭doublewrite，减少IO，提升性能。

**4.1 innodb buffer dump 功能增强**       
      5.7.5 新增加innodb\_buffer\_pool\_dump\_pct参数，来控制每个innodb buffer中转储活跃使用的innodb buffer pages的比例。之前的版本默认值是100%，当触发转储的时候 会全量dump innodb buffer pool中的pages。如果启用新的参数比如40 ，每个innodb buffer pool instance中有100个 ，每次转储每个innodb buffer 实例中的40个pages。  
    注意:当innodb发现innodb 后台io资源紧张时，会主动降低该参数设置的比例。  
  
**4.2 支持多线程刷脏页**     
      MySQL 5.6.2版本中，MySQL将刷脏页的线程从master线程独立出来，5.7.4版本之后，MySQL系统支持多线程刷脏页，进程的数量由innodb\_page\_cleaners参数控制，该参数不能动态修改，最小值为1 ，最大值支持64，5.7.7以及之前默认值是1 ，5.7.8版本之后修改默认参数为4。当启用多线程刷脏也，系统将刷新innodb buffer instance脏页分配到各个空闲的刷脏页的线程上，如果设置的innodb\_page\_cleaners>innodb\_buffer\_pool\_instances,系统会自动重置为innodb\_buffer\_pool\_instances大小。  
  
**4.3 动态调整 innodb buffer size**  
    从5.7.5版本, MySQL支持在不重启系统的情况下动态调整innodb\_buffer\_pool\_size。resize的过程是以chunk(每个chunk的大小默认为128M)的为单位迁移pages到新的内存空间，迁移进度可以通过Innodb\_buffer\_pool\_resize\_status 查看。记住整个resize的大小是以chunk为单位的。innodb\_buffer\_pool\_chunk\_size的大小,计算公式是innodb\_buffer\_pool\_size / innodb\_buffer\_pool\_instances,新调整的值必须是 innodb\_buffer\_pool\_chunk\_size\*innodb\_buffer\_pool\_instances的整数倍。如果不是整数倍，则系统则会调整值为大于两者乘积的最大值。   
例子

1. mysql> SHOW STATUS WHERE Variable\_name='InnoDB\_buffer\_pool\_resize\_status';
2. +----------------------------------+----------------------------------------------------------------------+
3. | Variable\_name | Value |
4. +----------------------------------+----------------------------------------------------------------------+
5. | Innodb\_buffer\_pool\_resize\_status | Size did not change (old size = new size = 268435456. Nothing to do. |
6. +----------------------------------+----------------------------------------------------------------------+
7. 1 row in set (0.00 sec)
8. mysql> set global innodb\_buffer\_pool\_size=128\*1024\*1024;
9. Query OK, 0 rows affected (0.00 sec)
10. mysql> SHOW STATUS WHERE Variable\_name='InnoDB\_buffer\_pool\_resize\_status';
11. +----------------------------------+----------------------------------------------------+
12. | Variable\_name | Value |
13. +----------------------------------+----------------------------------------------------+
14. | Innodb\_buffer\_pool\_resize\_status | Completed resizing buffer pool at 160702 23:53:51. |
15. +----------------------------------+----------------------------------------------------+
16. 1 row in set (0.00 sec)
17. mysql> set global innodb\_buffer\_pool\_size=256\*1024\*1024;
18. Query OK, 0 rows affected (0.00 sec)
19. mysql> SHOW STATUS WHERE Variable\_name='InnoDB\_buffer\_pool\_resize\_status';
20. +----------------------------------+----------------------------------------------------+
21. | Variable\_name | Value |
22. +----------------------------------+----------------------------------------------------+
23. | Innodb\_buffer\_pool\_resize\_status | Completed resizing buffer pool at 160702 23:54:19. |
24. +----------------------------------+----------------------------------------------------+
25. 1 row in set (0.00 sec)

online调整bp size的log 记录大致过程   
a 计算要调整的bpsize  
b 禁止AHI，清理所有的索引缓存  
c Withdrawing block是遍历freelist 确定可以使用的空闲block  
d 锁住整个buffer pool   
e 迁移重新分配chunk/删除可以释放的chunk   
f 设置innodb\_buffer\_pool\_size为新的值  
g 重新开启AHI

1. 2016-07-02T15:40:44.724495Z 0 [Note] InnoDB: Resizing buffer pool from 134217728 to 268435456 (unit=134217728).
2. 2016-07-02T15:40:44.724546Z 2 [Note] InnoDB: Resizing buffer pool from 134217 (new size: 268435456 bytes)
3. 2016-07-02T15:40:44.724559Z 0 [Note] InnoDB: Disabling adaptive hash index.
4. 2016-07-02T15:40:44.724979Z 0 [Note] InnoDB: disabled adaptive hash index.
5. 2016-07-02T15:40:44.725029Z 0 [Note] InnoDB: Withdrawing blocks to be shrunken.
6. 2016-07-02T15:40:44.725040Z 0 [Note] InnoDB: Latching whole of buffer pool.
7. 2016-07-02T15:40:44.725210Z 0 [Note] InnoDB: buffer pool 0 : resizing with chunks 1 to 2.
8. 2016-07-02T15:40:44.735439Z 0 [Note] InnoDB: buffer pool 0 : 1 chunks (8192 blocks) were added.
9. 2016-07-02T15:40:44.735511Z 0 [Note] InnoDB: Completed to resize buffer pool from 134217728 to 268435456.
10. 2016-07-02T15:40:44.735561Z 0 [Note] InnoDB: Re-enabled adaptive hash index.
11. 2016-07-02T15:40:44.735586Z 0 [Note] InnoDB: Completed resizing buffer pool at 160702 23:40:44.

这个特性是最令众多MySQL DBA 期待的特性之一。以后线上动态扩容，缩容就无需做数据库切换了，间接增强了系统的稳定性和DBA的生活幸福感。当然本文中介绍的略显粗略。详细内容请参考《[官方文档](http://dev.mysql.com/doc/refman/5.7/en/innodb-buffer-pool-resize.html)》   
  
**4.4 支持**[**全局表空间**](http://dev.mysql.com/doc/refman/5.7/en/general-tablespaces.html)  
     全局表空间可以被所有的数据库的表共享,而且相比于 file-per-table tablespaces. 使用共享表空间可以节约元数据方面的内存。(需要更深入的了解共享表空间 主要是大小 收缩问题)

1. mysql> CREATE TABLESPACE `youzan\_com`
2. -> ADD DATAFILE 'youzan\_com.ibd' FILE\_BLOCK\_SIZE = 16k;
3. Query OK, 0 rows affected (0.02 sec)
4. mysql> use yang
5. Database changed
6. mysql> create table yztb(id int primary key not null ,val char(10)) engine=innodb default charset=utf8 TABLESPACE youzan\_com ;
7. Query OK, 0 rows affected (0.04 sec)
8. mysql> create database youzan default charset utf8;
9. Query OK, 1 row affected (0.02 sec)
10. mysql> use youzan
11. Database changed
12. mysql>
13. mysql> create table yztb(id int primary key not null ,val char(10)) engine=innodb default charset=utf8 TABLESPACE youzan\_com ;
14. Query OK, 0 rows affected (0.03 sec)

**4.5 行格式默认为DYNAMIC**  
     从MySQL 5.7.9 开始,行格式DYNAMIC 取代COMPACT 成为innodb存储引擎默认的行格式，MySQL提供了新的参数innodb\_default\_row\_format来控制Innodb 行格式，详细的信息请参考《[Specifying the Row Format for a Table](http://dev.mysql.com/doc/refman/5.7/en/innodb-row-format-specification.html)》  
  
**4.6 支持原生的分区表**  
      在MySQL 5.7.6之前的版本中,创建分区表时MySQL为每个分区创建一个ha\_partition handler，自MySQL 5.7.6之后，MySQL支持原生的分区表并且只会为分区表创建一个partition-aware handler，这样的分区表功能增强节约分区表使用的内存。对于老版本创建的分区表在升级到新的版本之后怎么处理呢？莫慌，5.7.9之后，MySQL提供了如下升级方式解决这个问题：

1. ALTER TABLE ... UPGRADE PARTITIONING.

当然友情提示:从我个人的理解来看，在没有合适的自动化维护分区表系统的基础上，不推荐使用分区表。四年的工作经历已经数次在分区表上掉坑里了。  
  
**4.7 支持truncate undo logs**  
     MySQL 5.7.5版本开始支持truncate undo 表空间中的undo log。启用该特性必须设置innodb\_undo\_log\_truncate=[ON|1]。大致原理是系统必须设置至少两个undo 表空间(初始化的时候设置 innodb\_undo\_tablespaces=2 ) 用于清理undo logs的切换。该特性的好处是 解决了 ibdata 文件一直增大的问题，减轻系统的空间使用。 详细信息参考《[官方文档](http://dev.mysql.com/doc/refman/5.7/en/truncate-undo-tablespace.html)》

**5.1  支持JSON**  
  从MySQL 5.7.8 开始，MySQL支持原生的JSON格式，即有独立的json类型，用于存放 json格式的数据。JSON 格式的数据并不是以string格式存储于数据库而是以内部的binary 格式，以便于快速的定位到json 格式中值。  
在插入和更新操作时MySQL会对JSON 类型做校验，已检查数据是否符合json格式，如果不符合则报错。同时5.7.8 版本提供了四种JSON相关的函数，从而不用遍历全部数据。  
a 创建: JSON\_ARRAY(), JSON\_MERGE(), JSON\_OBJECT()  
b 修改: JSON\_APPEND(), JSON\_ARRAY\_APPEND(), JSON\_ARRAY\_INSERT(), JSON\_INSERT(), JSON\_QUOTE(), JSON\_REMOVE(), JSON\_REPLACE(), JSON\_SET(), and JSON\_UNQUOTE()  
c 查询: JSON\_CONTAINS(), JSON\_CONTAINS\_PATH(), JSON\_EXTRACT(), JSON\_KEYS(),JSON\_SEARCH().  
d 属性: JSON\_DEPTH(), JSON\_LENGTH(), JSON\_TYPE() JSON\_VALID().   
我们通过简单的例子来对json有一定的认识。  
创建

1. mysql> SELECT JSON\_ARRAY('id', 1, 'name', 'dba@youzan');
2. +-------------------------------------------+
3. | JSON\_ARRAY('id', 1, 'name', 'dba@youzan') |
4. +-------------------------------------------+
5. | ["id", 1, "name", "dba@youzan"] |
6. +-------------------------------------------+
7. 1 row in set (0.00 sec)
8. mysql> SELECT JSON\_OBJECT('id', 1, 'name', 'dba@youzan');
9. +--------------------------------------------+
10. | JSON\_OBJECT('id', 1, 'name', 'dba@youzan') |
11. +--------------------------------------------+
12. | {"id": 1, "name": "dba@youzan"} |
13. +--------------------------------------------+
14. 1 row in set (0.00 sec)

初始化

1. create table json\_test (
2. id int(11) PRIMARY KEY NOT NULL auto\_increment,
3. data json
4. ) engine=innodb default charset=utf8;
5. insert into json\_test values (1,'{ "DBA": [ { "firstName": "yi", "lastName":"yang", "email": "dba@youzan.com" }],
6. "SA": [{ "firstName": "you", "lastName": "zan", "email": "sa@youzan.com" }],
7. "PE": [{ "firstName": "xiao", "lastName": "xiao", "email": "pe@youzan.com" }] }')

修改

1. mysql> select \* from json\_test \G
2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
3. id: 1
4. data: {"PE": [{"email": "pe@youzan.com", "lastName": "xiao", "firstName": "xiao"}], "SA": [{"email": "sa@youzan.com", "lastName": "zan", "firstName": "you"}], "DBA": [{"email": "dba@youzan.com", "lastName": "yang", "firstName": "yi"}]}
5. 1 row in set (0.00 sec)
6. mysql> update json\_test set data=json\_array\_append(data,'$.DBA','{"email": "dba@youzan.com", "lastName": "yang", "firstName": "qilong"}') where id=1;
7. Query OK, 1 row affected (0.01 sec)
8. Rows matched: 1 Changed: 1 Warnings: 0
9. mysql> select \* from json\_test \G
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
11. id: 1
12. data: {"PE": [{"email": "pe@youzan.com", "lastName": "xiao", "firstName": "xiao"}], "SA": [{"email": "sa@youzan.com", "lastName": "zan", "firstName": "you"}], "DBA": [{"email": "dba@youzan.com", "lastName": "yang", "firstName": "yi"}, "{\"email\": \"dba@youzan.com\", \"lastName\": \"yang\", \"firstName\": \"qilong\"}"]}
13. 1 row in set (0.00 sec)

删除

1. mysql> update json\_test set data=json\_remove(data,'$.DBA[1]') where id=1;
2. Query OK, 1 row affected (0.01 sec)
3. Rows matched: 1 Changed: 1 Warnings: 0
4. mysql> select \* from json\_test \G
5. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
6. id: 1
7. data: {"PE": [{"email": "pe@youzan.com", "lastName": "xiao", "firstName": "xiao"}], "SA": [{"email": "sa@youzan.com", "lastName": "zan", "firstName": "you"}], "DBA": [{"email": "dba@youzan.com", "lastName": "yang", "firstName": "yi"}]}
8. 1 row in set (0.00 sec)

查看 json的key

1. mysql> SELECT id,json\_keys(data) as "keys" FROM json\_test;
2. +----+---------------------+
3. | id | keys |
4. +----+---------------------+
5. | 1 | ["PE", "SA", "DBA"] |
6. +----+---------------------+
7. 1 row in set (0.00 sec)
8. 查看DBA对应的值
9. mysql> SELECT id,json\_extract(data,'$.DBA[0]') from json\_test;
10. +----+--------------------------------------------------------------------+
11. | id | json\_extract(data,'$.DBA[0]')                                      |
12. +----+--------------------------------------------------------------------+
13. | 1 | {"email": "dba@youzan.com", "lastName": "yang", "firstName": "yi"}  |
14. +----+--------------------------------------------------------------------+
15. 1 row in set (0.00 sec)

其他函数的用法请感兴趣的读者朋友自行参考《[官方文档](http://dev.mysql.com/doc/refman/5.7/en/json-functions.html)》  
MySQL 5.7 版本提供的json格式以及对应的操作函数极丰富了MySQL的存储格式,可以在一定程度上和Mongodb和pg竞争，对于经常使用MySQL varchar 存储json的业务是一个福音。同时再强调一下对于OLTP业务的表结构设计 尽可能的避免大字段存储。一来是减少不必要的查询带来的IO，带宽，内存方面的影响 二来是 避免因为表大小太大导致的ddl 时间成本增加系统风险。  
**5.2 sys schema**  
  MySQL 5.7 版本新增了sys 数据库,该库通过视图的形式把information\_schema 和performance\_schema结合起来，查询出更加令人容易理解的数据，帮助DBA快速获取数据库系统的各种纬度的元数据信息，帮助DBA和开发快速定位性能瓶颈。详细的信息请参考《[官方文档](http://dev.mysql.com/doc/refman/5.7/en/sys-schema.html)》，这里给两个例子能直观的了解sys 功能的强大。

1. mysql> select \* from sys.schema\_table\_statistics limit 2\G
2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
3. table\_schema: yang
4. table\_name: json\_test
5. total\_latency: 1.81 ms
6. rows\_fetched: 21
7. fetch\_latency: 1.45 ms
8. rows\_inserted: 2
9. insert\_latency: 192.67 us
10. rows\_updated: 2
11. update\_latency: 166.94 us
12. rows\_deleted: 0
13. delete\_latency: 0 ps
14. io\_read\_requests: 54
15. io\_read: 4.21 KiB
16. io\_read\_latency: 289.37 us
17. io\_write\_requests: 43
18. io\_write: 388.53 KiB
19. io\_write\_latency: 703.51 us
20. io\_misc\_requests: 75
21. io\_misc\_latency: 40.02 ms
22. ##直接查看未使用过的索引 ，方便吗？
23. mysql> SELECT \* FROM schema\_unused\_indexes;
24. +---------------+-------------+------------+
25. | object\_schema | object\_name | index\_name |
26. +---------------+-------------+------------+
27. | yang          | t           | idx\_a |
28. | yang          | yy          | idx\_nm |
29. +---------------+-------------+------------+
30. 2 rows in set (0.00 sec)

**6.1  优化(工具方面)增强**  
  5.7 版本中如果一个会话正在执行sql，且该sql 是支持explain的，那么我们可以通过指定会话id，查看该sql的执行计划。

1. EXPLAIN [options] FOR CONNECTION connection\_id

该功能可以在一个会话里面查看另外一个会话中正在执行的长查询。

1. mysql> show processlist;
2. +----+-------------+-----------------+------+---------+------+--------------------------------------------------------+------------------+
3. | Id | User        | Host            | db   | Command | Time | State                                                  | Info             |
4. +----+-------------+-----------------+------+---------+------+--------------------------------------------------------+------------------+
5. | 1  | system user |                 | NULL | Connect | 78   | Connecting to master                                   | NULL             |
6. | 2  | system user |                 | NULL | Connect | 78   | Slave has read all relay log; waiting for more updates | NULL             |
7. | 3  | system user |                 | NULL | Connect | 78   | Waiting for an event from Coordinator                  | NULL             |
8. | 4  | system user |                 | NULL | Connect | 78   | Waiting for an event from Coordinator                  | NULL             |
9. | 5  | system user |                 | NULL | Connect | 78   | Waiting for an event from Coordinator                  | NULL             |
10. | 6  | system user |                 | NULL | Connect | 78   | Waiting for an event from Coordinator                  | NULL             |
11. | 8  | root        | localhost:47896 | NULL | Query   | 0    | starting                                               | show processlist |
12. | 9  | root        | localhost:47897 | NULL | Query   | 3    | User sleep                                             | select sleep(10) |
13. +----+-------------+-----------------+------+---------+------+--------------------------------------------------------+------------------+
14. 8 rows in set (0.00 sec)
15. mysql> explain FOR CONNECTION 9;
16. +----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
17. | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | filtered | Extra |
18. +----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
19. | 1 | SIMPLE | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | No tables used |
20. +----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
21. 1 row in set (0.00 sec)

可以参考官方文档 和 [**work log**](http://dev.mysql.com/worklog/task/?id=6369)  
  
**6.2 hint 功能增强**  
 相比于MySQL5.6版本的hint 主要是index 级别的hint和控制表join 顺序的hint,5.7.7之后，MySQL增加了优化器hint,来控制sql执行的方式，因为目前MySQL支持nest loop join，故暂时无hint来修改sql 的join方式。熟悉Oracle 的朋友是否会发现MySQL 和Oracle 在功能上越来越近了。话说回来5.7的hint (先别和 index hint 比较)的用法 ，和oracle 的类似:

1. SELECT /\*+ NO\_RANGE\_OPTIMIZATION(t3 PRIMARY, f2\_idx) \*/ f1 FROM t3 WHERE f1 > 30 AND f1 < 33;
2. SELECT /\*+ BKA(t1) NO\_BKA(t2) \*/ \* FROM t1 INNER JOIN t2 WHERE ...;
3. SELECT /\*+ NO\_ICP(t1, t2) \*/ \* FROM t1 INNER JOIN t2 WHERE ...;
4. SELECT /\*+ SEMIJOIN(FIRSTMATCH, LOOSESCAN) \*/ \* FROM t1 ...;
5. EXPLAIN SELECT /\*+ NO\_ICP(t1) \*/ \* FROM t1 WHERE ...

优化器级别的hint分四种类型

1. Global: The hint affects the entire statement
2. Query block: The hint affects a particular query block within a statement ，什么是query block
3. (SELECT ... ) UNION (SELECT /\*+ ... \*/ ... ) --后面的括号里面的称为 query block 。
4. Table-level: The hint affects a particular table within a query block
5. Index-level: The hint affects a particular index within a table

其他更加详细的信息请参考  [**官方文档**](http://dev.mysql.com/doc/refman/5.7/en/optimizer-hints.html)   
  
**6.3 触发器功能增强**  
 5.7版本之前一个表 对于每种action(INSERT,UPDATE, DELETE)和时机(BEFORE or AFTER) 只能支持一种类型的触发器。新版本可以针对同一个action支持多个触发器。  
  
**6.4 syslog 功能**  
  之前的版本，\*nix系统上的MySQL支持将错误日志发送到syslog是通过mysqld\_safe捕获错误输出然后传递到syslog来实现的。新的版本原生支持将错误日志输出到syslog,且适用于windows系统，只需要通过简单的参数(log\_syslog等)配置即可。参考 [**官方文档**](http://dev.mysql.com/doc/refman/5.7/en/error-log.html)  
  MySQL支持–syslog选项，可将在交互式模式下执行过的命令输出到syslog中（\*nix系统下一般是.mysql\_history）。对于匹配“ignore”过滤规则（可通过 –histignore选项或者 MYSQL\_HISTIGNORE环境变量进行设置）的语句不会被记入。关于mysql客户端的日志使用参见：[**官方文档**](http://dev.mysql.com/doc/refman/5.7/en/mysql-logging.html)  
**6.5 虚拟列**  
在MySQL 5.7中，支持两种Generated Column，  
1 Virtual Generated Column :只将Generated Column保存在数据字典中表的元数据，每次读取该列时进行计算，并不会将这一列数据持久化到磁盘上；  
注意:MySQL 5.7.8 以前 虚拟列字段不支持创建索引。5.7.8之后Innodb支持在虚拟列创建辅助索引。  
2 Stored Generated Column : 将Column持久化到存储，会占用一定的存储空间。与Virtual Column相比并没有明显的优势，因此，MySQL 5.7中，不指定Generated Column的类型，默认是Virtual Column。  
创建虚拟列语法：

1. col\_name data\_type [GENERATED ALWAYS] AS (expression)
2. [VIRTUAL | STORED] [UNIQUE [KEY]] [COMMENT comment]
3. [[NOT] NULL] [[PRIMARY] KEY]

具体的例子

1. CREATE TABLE triangle (
2. id int(10) not null primary key auto\_increment,
3. sidea DOUBLE,
4. sideb DOUBLE,
5. sidec DOUBLE AS (SQRT(sidea \* sidea + sideb \* sideb))
6. );
7. INSERT INTO triangle (sidea, sideb) VALUES(1,1),(3,4),(6,8),(12,16);
8. mysql> select \* from triangle;
9. +----+-------+-------+--------------------+
10. | id | sidea | sideb | sidec              |
11. +----+-------+-------+--------------------+
12. | 1  | 1     | 1     | 1.4142135623730951 |
13. | 2  | 3     | 4     | 5                  |
14. | 3  | 6     | 8     | 10                 |
15. | 4  | 12    | 16    | 20                 |
16. +----+-------+-------+--------------------+
17. 4 rows in set (0.00 sec)
18. mysql> explain select \* from triangle where sidec > 10;
19. +----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------------+
20. | id | select\_type | table    | partitions | type | possible\_keys | key  | key\_len | ref  | rows | filtered | Extra       |
21. +----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------------+
22. | 1  | SIMPLE      | triangle | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 4    | 33.33    | Using where |
23. +----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------------+
24. 1 row in set, 1 warning (0.00 sec)
25. mysql> alter table triangle add key idx\_sidec(sidec);
26. Query OK, 0 rows affected (0.03 sec)
27. Records: 0 Duplicates: 0 Warnings: 0
28. mysql> explain select \* from triangle where sidec > 10;
29. +----+-------------+----------+------------+-------+---------------+-----------+---------+------+------+----------+-------------+
30. | id | select\_type | table    | partitions | type  | possible\_keys | key       | key\_len | ref  | rows | filtered | Extra       |
31. +----+-------------+----------+------------+-------+---------------+-----------+---------+------+------+----------+-------------+
32. | 1  | SIMPLE      | triangle | NULL       | range | idx\_sidec     | idx\_sidec | 9       | NULL | 1    | 100.00   | Using where |
33. +----+-------------+----------+------------+-------+---------------+-----------+---------+------+------+----------+-------------+
34. 1 row in set, 1 warning (0.00 sec)

看到这个例子,熟悉oracle的朋友可能会和函数索引作比较，两者比较类似.使用虚拟列达到函数索引或者解决业务上的设计缺陷，但是个人不建议使用类似的功能，因为虚拟列在一定程度上也会给后期运维带来潜在的风险和复杂度。网络上的例子基本都是使用虚拟列解决业务逻辑上的问题，违背了数据库只存储数据的初衷，思考一下MVC 框架的基本逻辑,业务逻辑要放到C 层或者V层，M层只存放数据即可。  
各位看官自己要了解更详细的虚拟列资料请移步[**官方文档**](http://dev.mysql.com/doc/refman/5.7/en/create-table-generated-columns.html)