

MySql笔记-innodb无特殊说明，基于5.7版本

[零 参考资料 4](#_Toc43661137)

[一 概述 4](#_Toc43661138)

[二 表的逻辑存储 4](#_Toc43661139)

[1 概述 4](#_Toc43661140)

[2 IO和文件管理 5](#_Toc43661141)

[3 文件格式 5](#_Toc43661142)

[3.1 概述 5](#_Toc43661143)

[4 系统表空间(System tablespace) 5](#_Toc43661144)

[5 file-per-table表空间 6](#_Toc43661145)

[5.1 概述 6](#_Toc43661146)

[5.2 优势 6](#_Toc43661147)

[5.3 劣势 6](#_Toc43661148)

[6 一般表空间(General Tablespaces) 6](#_Toc43661149)

[6.1 概述 6](#_Toc43661150)

[6.2 限制 7](#_Toc43661151)

[7 Undo Tablespaces 8](#_Toc43661152)

[8 临时表空间(The Temporary Tablespace) 8](#_Toc43661153)

[9 Segment 8](#_Toc43661154)

[10 Extent 8](#_Toc43661155)

[11 Page 8](#_Toc43661156)

[11.1 概述 8](#_Toc43661157)

[11.2 File Header 9](#_Toc43661158)

[11.3 Page Header 9](#_Toc43661159)

[11.4 Infimun+Supremum Records 9](#_Toc43661160)

[11.5 User Records 9](#_Toc43661161)

[11.6 Free Space 9](#_Toc43661162)

[11.7 Page Directory 9](#_Toc43661163)

[11.8 File Trailer 9](#_Toc43661164)

[11.9 页文件分析 9](#_Toc43661165)

[12 Row 10](#_Toc43661166)

[12.1 概述 10](#_Toc43661167)

[12.2 REDUNDANT 10](#_Toc43661168)

[12.3 COMPACT 11](#_Toc43661169)

[12.4 DYNAMIC 12](#_Toc43661170)

[12.5 COMPRESSED 12](#_Toc43661171)

[12.6 COMPACT 行格式文件解析 12](#_Toc43661172)

[二 索引 13](#_Toc43661173)

[1 概述 13](#_Toc43661174)

[2 B树 13](#_Toc43661175)

[3 B+树 13](#_Toc43661176)

[4 Clustered index 13](#_Toc43661177)

[5 Secondary index 13](#_Toc43661178)

[6 哈希索引(Hash index) 13](#_Toc43661179)

[7 覆盖索引(Covering Index) 13](#_Toc43661180)

[8 全文检索(Full-Text Search)和倒排索引(Inverted Index) 13](#_Toc43661181)

[9 空间索引(Spatial Index)和R树(R-tree) 13](#_Toc43661182)

[三 InnoDB In-Memory Structures 13](#_Toc43661183)

[1 LUR(the least recently used) 算法 13](#_Toc43661184)

[2 Buffer Pool 13](#_Toc43661185)

[2.1 概述 13](#_Toc43661186)

[2.2 缓冲池配置 14](#_Toc43661187)

[2.3 控缓冲池监控参数 14](#_Toc43661188)

[3 Change Buffer 14](#_Toc43661189)

[3.1 概述 14](#_Toc43661190)

[3.2 相关配置 15](#_Toc43661191)

[3.3 相关监控 15](#_Toc43661192)

[4 Log Buffer 15](#_Toc43661193)

[5 自适应哈希索引(Adaptive Hash Index) 15](#_Toc43661194)

[四 Lock 15](#_Toc43661195)

[1 概述 15](#_Toc43661196)

[2 Lock类型 16](#_Toc43661197)

[2.1 Shared and Exclusive Locks 16](#_Toc43661198)

[2.2 Intention Locks 16](#_Toc43661199)

[2.3 Record Locks 16](#_Toc43661200)

[2.4 Gap Locks 16](#_Toc43661201)

[2.5 Next-Key Locks 17](#_Toc43661202)

[2.6 Insert Intention Locks 17](#_Toc43661203)

[2.7 AUTO-INC Locks 17](#_Toc43661204)

[2.8 Predicate Locks 17](#_Toc43661205)

[3 Phantom Row 18](#_Toc43661206)

[4 Deadlock 18](#_Toc43661207)

[五 日志(Log) 18](#_Toc43661208)

[六 事务(Transaction) 18](#_Toc43661209)

[1 概述 18](#_Toc43661210)

[2 ACID 18](#_Toc43661211)

[https://en.wikipedia.org/wiki/ACID 18](#_Toc43661212)

[2.1 A(atomicity) 18](#_Toc43661213)

[2.2 C(cosistency) 18](#_Toc43661214)

[2.3 I(isolation) 18](#_Toc43661215)

[2.4 D(durability) 18](#_Toc43661216)

[3 事务隔离(Transaction Isolation) 19](#_Toc43661217)

[https://en.wikipedia.org/wiki/Isolation\_(database\_systems) 19](#_Toc43661218)

[3.1 概述 19](#_Toc43661219)

[3.2 READ UNCOMMITED 19](#_Toc43661220)

[3.3 READ COMMITTED 19](#_Toc43661221)

[3.4 REPEATABLE READ 19](#_Toc43661222)

[3.5 SERIALIZABLE 20](#_Toc43661223)

[4 redo log 20](#_Toc43661224)

[5 undo log 20](#_Toc43661225)

[6 purge 20](#_Toc43661226)

[7 事务类型 20](#_Toc43661227)

[8 XA事务 20](#_Toc43661228)

[七 其他 20](#_Toc43661229)

[1 、配置 20](#_Toc43661230)

[2 、优化 20](#_Toc43661231)

[3 、限制 20](#_Toc43661232)

# 零 参考资料

《Mysql技术内幕 InnoDB存储引擎》

《高性能Mysql 第三版》

《高可用Mysql》

《算法导论 第三版》

《漫画算法：小灰的算法之旅》

《算法 第四版》

<https://dev.mysql.com/doc/refman/5.7/en/>

<https://dev.mysql.com/doc/refman/5.7/en/innodb-on-disk-structures.html>

# 概述

* + - 2. 整体架构图



# 表的逻辑存储

## 概述

* + - 1. 索引组织表(index organized table)：表都是根据主键顺序存放
      2. 如果没有显式指定主键，则：
         1. 判断表中是否由唯一非空索引，有则指定其为主键，如果有多个，则根据定义索引的顺序，指定第一个为主键
         2. 如果b.1不符合，则自建一个6字节大小的指针键
      3. 单列主键情况下，可以用\_rowid查看主键情况。
      4. 所有数据都被逻辑地放在表空间中(tablespace),表空间又由段(segment),区(Extent),页(page,或者称为块(block))
      5. 逻辑存储结构



## IO和文件管理

## 文件格式

### 概述

* + - 1. Innodb目前支持两种文件格式：Antelope、Barracuda
         1. Antelope

是最初的文件格式，之前甚至没有名字。。。

仅支持COMPACT和REDUNDANT行格式

## 系统表空间(System tablespace)

<https://dev.mysql.com/doc/refman/5.7/en/innodb-system-tablespace.html>

* + - 1. 可以有多个表空间数据文件，默认情况下，为一个名为ibdata1的在数据目录下的初始大小为10M的单文件，可以通过innodb\_data\_file\_path参数配置存储了Innodb的数据字典(the InnoDB data dictionary)，双写缓冲(the doublewrite buffer)，改变缓冲(the change buffer)，undo日志(undo logs)以及一些创建在此空间的表的
      2. 当innodb\_file\_per\_table参数禁用且未指定表空间时，默认在此空间创建表
      3. 当操作引起表空间数据减少时，不会收缩表空间大小，只是将释放出来的空间标记为可用

## file-per-table表空间

<https://dev.mysql.com/doc/refman/5.7/en/innodb-file-per-table-tablespaces.html>

### 概述

* + - 1. 当innodb\_file\_per\_table启用且未指定表空间时，会为每个创建表动作在数据目录下独立创建一个对应的文件名为${表名}.idb的表空间文件

### 

### 优势

* + - 1. 在truncating 或 dropping操作后，可以将存储空间还给系统
      2. 表复制ALTER TABLE操作，不会增加存储空间占用
      3. TRUNCATE TABLE操作性能相对更好
      4. 此表空间文件可以创建在不同的存储空间中
      5. 可以从其他Mysql直接迁移指定的表
      6. 可以使用Barracuda文件格式
      7. 当数据异常时可以更快速修复以及提高恢复概率
      8. 在不打断其他表使用的情况下，可以迅速恢复/备份表
      9. 可以通过监视表空间文件大小，监视文件系统上的表大小
      10. 当设置innodb\_flush\_method值为O\_DIRECT时，可以提高性能
      11. 每个表都是64TB大小极限(Tables in a shared tablespace are limited in size by the 64TB tablespace size limit. By comparison, each file-per-table tablespace has a 64TB size limit, which provides plenty of room for individual tables to grow in size.)

### 劣势

* + - 1. 此表空间只能被指定表使用，导致资源浪费
      2. 会导致有很多fsync操作
      3. mysqld必须保存很多文件句柄，导致影响性能
      4. 每个表需要更多文件描述符
      5. 如果没管理好，会产生很多碎片，影响dropTable和表扫描操作。反之会提高性能
         1. 因为缓冲池扫描会用到一个内部锁，所以在此模式下，会延迟其他操作执行
         2. 配置innodb\_autoextend\_increment只应用于共享表空间文件，这个空间固定每次扩展4MB

## 一般表空间(General Tablespaces)

https://dev.mysql.com/doc/refman/5.7/en/general-tablespaces.html

### 概述

* + - 1. 此空间是个通过CREATE TABLESPACE创建，可共享的表空间，文件后缀为.isl
      2. 存储表空间元数据消耗内存比file-per-table tablespaces少
      3. 表空间文件可以存在相对目录或独立的数据目录
      4. 支持Antelop和Barracuda文件格式，因此支持所有行格式和相关功能
      5. 参数innodb\_file\_format 和 innodb\_file\_per\_table不影响此表空间
      6. 在执行Create table时，可以指定tablespace
      7. 在执行Alter table时，可以通过Tablespace，对表做表空间迁移
      8. 由于物理页不同，压缩和非压缩的表不能在同一个表空间共存
      9. 表空间如果包含压缩表，FILE\_BLOCK\_SIZE必须指定具体值，同时innodb\_page\_size不能为32k或64k,而且KEY\_BLOCK\_SIZE必须等于FILE\_BLOCK\_SIZE/1024
      10. 当表空间中最后一张表被删除时，不会自动删除表空间文件，必须执行drop tablespace
      11. 此表空间不属于特定database，drop database操作能删掉此空间的表，但删不掉此空间
      12. 和系统空间一样，修改表数据不会改变空间大小，但会在表空间内部释放资源
      13. 表空间创建语句

1. **CREATE** TABLESPACE tablespace\_name
2. **ADD** DATAFILE 'file\_name'
3. [FILE\_BLOCK\_SIZE = value]
4. [ENGINE [=] engine\_name]

### 限制

* + - 1. 不能将其他已存在的表空间改成General tablespace
      2. 不支持创建临时General tablespace
      3. Generaltable不支持临时表
      4. 此空间中的表不支持DISCARD TABLESPACE 和IMPORT TABLESPACE操作
      5. MySQL 5.7.24以后会不支持此表空间中的表分区
      6. 不支持同主机主从复制

## Undo Tablespaces

https://dev.mysql.com/doc/refman/5.7/en/innodb-undo-tablespaces.html

## 临时表空间(The Temporary Tablespace)

https://dev.mysql.com/doc/refman/5.7/en/innodb-temporary-tablespace.html

## Segment

* + - 1. 在Innodb中，段的管理由引擎自动完成
      2. 这里的段跟回滚段不同
      3. 如果表空间是目录，段就是目录中文件
      4. 段每次扩容都一次1个区，每次释放也是一次一个区
      5. file-per-table 表空间，数据都在一个段中，对应的索引在各自的段中
      6. system tablespace表空间则包含各种不同的段
      7. 在Innodb中每个索引分配两个段，一个用于Btree的非叶子节点，一个用于叶子节点
      8. 个别Extent会分配在多个Segment中

## Extent

* + - 1. 由连续页组成的空间，
         1. 当页大小为4KB、8KB、16KB时，区大小固定为1MB；
         2. 当页大小为32KB，对应的区大小为2MB，
         3. 页大小为64KB,对应的区大小为4MB
      2. 默认情况下，一个区包含64个大小为16KB的page

## Page

### 概述

* + - 1. 结构：File Header，Page Header，Infimun和Supremum Records，User Records，Free Space，Page Directory，File Trailer



### File Header

### Page Header

### Infimun+Supremum Records



### User Records

### Free Space

### Page Directory

### File Trailer

### 页文件分析

## Row

### 概述

* + - 1. 行格式(row format): 决定了表的每行物理存储格式，直接影响到查询和DML操作的性能
      2. 硬盘上单页中存储的行更多，性能越好
      3. 行记录数据格式有四种

| **格式** | **紧凑存储特征** | **Enhanced Variable-Length Column Storage** | **Large Index Key Prefix Support** | **支持压缩** | **表空间类型** | **文件格式** |
| --- | --- | --- | --- | --- | --- | --- |
| REDUNDANT | No | No | No | No | system  file-per-table, general | Antelope  Barracuda |
| COMPACT | Yes | No | No | No | system  file-per-table  general | Antelope  Barracuda |
| DYNAMIC | Yes | Yes | Yes | No | system  file-per-table general | Barracuda |
| COMPRESSED | Yes | Yes | Yes | Yes | file-per-table general | Barracuda |

* + - 1. 可变长列(Variable-length columns):
         1. 一种存储在B树索引节点的特例
         2. 被存储在overflow pages上(Variable-length columns that are too long to fit on a B-tree page are stored on separately allocated disk pages called overflow pages.)
         3. 这些列也称为off-page
         4. 列上的数据存储在overflow page的单链表上
         5. 根据列长度不同，为避免浪费存储，列上的值的全部或部分前缀存储在B树上(Depending on column length, all or a prefix of variable-length column values are stored in the B-tree to avoid wasting storage and having to read a separate page)

### REDUNDANT

* + - 1. 老版本Mysql采用的数据结构，5.7版本为兼容老版本而依旧存在
      2. 支持Antelope和Barracuda文件
      3. 对于可变长列，值的前768 bytes的内容存储在B树节点的索引中，其余的存储在overflow page中
      4. 对于固定长度的列，如果值实际长度大于等于768 bytes ，会被转为可变长列存储
      5. 当列的值实际长度小于768 bytes，则为节省I/O开支，不会存储在overflow page，都存储在B树节点上
      6. 特征
         1. 每个索引记录包含6 byte的头，用于将连续的记录连在一起，以及行级锁
         2. 聚集索引的记录包含所有用户定义的列，一个6 byte的事务ID字段，一个7 byte的滚动指针(roll pointer)字段
         3. 如果用户没有定义表的主键，聚集索引还包括一个6 byte的row id字段
         4. 每个辅助索引(Secondary Index)包含聚集索引中的所有不在辅助索引中的主键字段
         5. 每个记录都包含一个指向所有字段的指针。

如果所有字段总长度小于128byte，则指针大小为1byte

如果所有字段总长度大于等于128byte，则指针大小为2byte

这类指针组被称为记录目录(The array of pointers is called the record directory)

指针指向的空间，是记录的数据部分

* + - * 1. 对于数据NULL值

无论可变长度列或固定长度列，都会在记录目录(record directory)预留1byte或2byte

对于可变长度列，NULL值不会在数据区预留空间

对于固定长度列，NULL值也会预留固定长度的空间，目的是为了字段改为非NULL时不产生碎片

* + - * 1. 对于采用可变长度字符集的char(N)列，占用空间=最大字符长度\*N

### COMPACT

* + - 1. 相对于REDUNDANT格式，以增加CPU消耗为代价，减少了20%存储空间
      2. 支持Antelope和Barracuda文件
      3. 在I/O密集型环境中，性能会有一定提升；在计算密集型，性能反而下降
      4. 对于可变长列，值的前768 bytes的内容存储在B树节点的索引中，其余的存储在overflow page中
      5. 对于固定长度的列，如果值实际长度大于等于768 bytes ，会被转为可变长列存储
      6. 当列的值实际长度小于768 bytes，则为节省I/O开支，不会存储在overflow page，都存储在B树节点上
      7. 特征
         1. 每个索引记录包含5 byte头，用于将连续的记录连在一起，以及行级锁，在其前面可能还有个可变长度的头部
         2. 记录头的可变长部分(The variable-length part of the record header)

包含 1个 位矢量(bit vector)用于表明有NULL列，

如果NULL列数量为N，则这个位矢量占CEILING(N/8) bytes

除了这个位矢量, NULL值不占用其他空间

也包含可变列的长度，根据可变长列的最大长度，占1byte或2byte

如果所有列都是固定长度且NOT NULL，记录头就没有可变长部分

* + - * 1. 对于每个 非NULL的可变长度字段

记录头用1byte或2byte表示列长度

仅当当前记录需要存储在overflow page或字段最大长度超过255byte且实际长度超过127byte时，占用2byte（Two bytes are only needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes）

对于大字段，2 byte 表示了内部存储部分长度加上20 byte 指向外部存储的部分的指针 的长度；例如：内部数据长度时768 bytes，所以 2byte表示 768 + 20

* + - 1. 记录头后面是非NULL列的数据
      2. 聚集索引的记录包含所有用户定义的列，一个6 byte的事务ID字段，一个7 byte的滚动指针(roll pointer)字段
      3. 如果用户没有定义表的主键，聚集索引还包括一个6 byte的row id字段
      4. 每个辅助索引(Secondary Index)包含聚集索引中的所有不在辅助索引中的主键字段。如果部分主键列是可变长度的，每个辅助索引的头部都会有一个记录主键列长度的可变长度部分，即使辅助索引字列是固定长度
      5. 在内部，对于非可变长度字符集(nonvariable-length character sets)
         1. 固定长度字符列采用固定长度格式存储
         2. 尾部的空格不会从varchar列截掉
      6. 在内部，对于可变长度字符集(utf8mb3、utf8mb4)
         1. 针对char(N)

先尝试通过截断尾部空格，达到用N bytes空间存储的目的

如果实际值超过了N bytes，尾部空格将被截取到列值的最小字节长度

char(N)最大长度 = 单字符最大长度 \* N

至少要预留Nbytes空间，用于保证列完成更新并不产生碎片

### DYNAMIC

* + - 1. 整体特征跟COMPACT一样
      2. 增加了对超长可变长度列(long variable-length columns)的支持
      3. 增加了对大索引key前缀(large index key prefixes)的支持
      4. 支持Barracuda文件
      5. 当建表时指定ROW\_FORMAT=DYNAMIC，超长可变长度列的数据将完全存储在overflow page，而聚簇索引记录仅包含20byte的指向overflow page的指针。
      6. 列是否存储在off-page 依赖于页大小和行的总大小，当行太长时，最大的列将存在off-page，直到聚簇索引记录适合B树页
      7. 当TEXT 和 BLOB 列长度小于等于40byte时，存储在行中
      8. DYNAMIC格式规避了大数据量的列填满B树节点的问题
      9. 短的列将被保留在B树节点，从而最小化每行的 overflow page数量
      10. 支持索引key前缀3072 byte，需要通过innodb\_large\_prefix配置

### COMPRESSED

* + - 1. 整体特征跟COMPACT和 DYNAMIC一样
      2. 增加了支持对表和索引数据的压缩
      3. 支持Barracuda文件
      4. 采用了于DYNAMIC类似的内部细节存储off-page，同时也考虑到了因为表和索引数据被压缩以及用更小的页所带来的额外存储和性能消耗
      5. 不支持系统表空间
      6. 不支持同个表空间中同时存在压缩和不压缩的表

### COMPACT 行格式文件解析

# 索引

## 概述

## B树

## B+树

## Clustered index

## Secondary index

## 哈希索引(Hash index)

## 覆盖索引(Covering Index)

## 全文检索(Full-Text Search)和倒排索引(Inverted Index)

## 空间索引(Spatial Index)和R树(R-tree)

# InnoDB In-Memory Structures

## LUR(the least recently used) 算法

## Buffer Pool

<https://dev.mysql.com/doc/refman/5.7/en/innodb-buffer-pool.html>

### 概述

* + - 1. 实现算法是LUR算法的一个变种，默认操作流程：
         1. 将缓冲池分为两部分：
         2. 年轻子链表(new/young sublist)：占5/8缓存池容量，最新访问的页
         3. 年老子链表(old sublist)：占3/8缓存池容量，非最近访问的页，淘汰页时候选页
         4. 设置一个边界midpoint：在年轻子链表的表尾链接年老子链的表头位置
         5. 当因为用户查询操作或作为innodb自动执行的预读操作的一部分，需要读取一个新页进入缓冲池时，将新页插入到midpoint位置
         6. 当缓冲池中的一个页被存取
         7. 如果是用户操作引起，在第一次存取时，立即将此页放入到年轻子链表的头部
         8. 如果是因为innodb自动执行的预读操作引起，在第一次存取时，不会将此页放入到年轻子链表的头部，可能直到此页被淘汰，都不会被放入到年轻子链表的头部
         9. 随着数据库操作，缓冲池中的页会被逐步移向缓冲池尾部(年老子链表尾部)
         10. 每个页随着其他页被移到年轻子链表的头部而逐渐老化
         11. 年老子链中的页，同时也因为新页插入到midpoint而加速老化
         12. 最后，缓冲池中始终未被使用的页因为到达了年老子链表的尾部而被淘汰



### 缓冲池配置

### 控缓冲池监控参数

* + - 1. 执行SHOW ENGINE INNODB STATUS 命令查看

## Change Buffer

<https://dev.mysql.com/doc/refman/5.7/en/innodb-change-buffer.html>

<https://dev.mysql.com/doc/refman/5.7/en/faqs-innodb-change-buffer.html>

### 概述

* + - 1. 这部分用于缓存不在缓冲池中的secondary index 页的变更
      2. 这部分缓存会根据相关的读操作，定期合并到Buffer Pool中去，避免频繁的磁盘随机I/O
      3. 当系统有大量空闲时间或处于slow shutdown时，会对这部分缓冲执行清理操作，将数据索引页更新序列化地写入到磁盘
      4. 当涉及很多行数据和secondary index更新，合并操作可能持续几小时。
      5. 当事务提交后也可能会触发合并操作
      6. 在memory中，Change Buffer占用buffer pool一部分空间；在disk中，Change Buffer是系统表空间的一部分
      7. Change Buffer不支持包含降序索引列的索引



### 相关配置



### 相关监控

## Log Buffer

<https://dev.mysql.com/doc/refman/5.7/en/innodb-redo-log-buffer.html>

* + - 1. 用于缓冲日志数据，并定期写入到disk
      2. 通过innodb\_log\_buffer\_size 配置，默认16MB
      3. 大的Log Buffer可以使执行大事务而不需要在事务提交前将redo日志数据写入到disk中，减少disk I/O
      4. 参数innodb\_flush\_log\_at\_trx\_commit 控制log buffer 写disk方式，参数innodb\_flush\_log\_at\_timeout控制写频率

## 自适应哈希索引(Adaptive Hash Index)

<https://dev.mysql.com/doc/refman/5.7/en/innodb-adaptive-hash.html>



# Lock

## 概述

* + - 1. Innodb支持行locks和表 locks并存的多粒度锁
      2. Innodb 支持row-level locks，实际上可视为index-record locks
      3. 事务请求lock
         1. 如果跟现有lock不冲突，则可以获取，
         2. 反之则不能获取，并阻塞等待，直到现有的锁被释放
         3. 如果现有lock 一直不被释放，会产生deadlock

## Lock类型

### Shared and Exclusive Locks

* + - 1. shared lock：允许事务读取一行数据， 简称S
      2. exclusive lock：允许事务更新/删除一行，简称 X
      3. 当事务T1获取R行一个shared lock后，事务T2也想操作R行：
         1. 如果T2也是请求一个shared lock，则同时获取，即T1和T2同时持有R行的shared lock
         2. 如果T2请求一个exclusive lock，请求将被阻塞，直到T1释放shared lock
      4. 如果事务T1 获取到R行的exclusive lock后，事务T2的任何请求都将被阻塞，直到T1释放exclusive lock

### Intention Locks

* + - 1. 是一种table-level locks，具体有两种：
         1. intention shared locks：当一个事务想在一张表中获取某行的shared locks，需要先获取此锁或更强的锁，简称IS
         2. intention exclusive lock：当一个事务想在一张表中获取某行的exclusive locks，需要先获取此锁，简称IX
      2. Intention Locks主要目的是用于哪些人锁定了某行，除了全表扫描请求，不会阻塞请求
      3. lock冲突简表：

|  | X | IX | S | IS |
| --- | --- | --- | --- | --- |
| X | 不兼容 | 不兼容 | 不兼容 | 不兼容 |
| IX | 不兼容 | 兼容 | 不兼容 | 兼容 |
| S | 不兼容 | 不兼容 | 兼容 | 兼容 |
| IS | 不兼容 | 兼容 | 兼容 | 兼容 |

### Record Locks

* + - 1. 锁住索引记录，阻止其他事务对锁定的行执行inserting、updating、deleting操作；比如(SELECT \* FROM t WHERE a=1 FOR UPDATE)
      2. Record Locks 一定是锁索引记录，如果表没有建立任何索引，默认锁建表时默认创建的clustered index

### Gap Locks

* + - 1. 锁在两个索引记录之间或者在第一个索引记录之前或最后个索引记录之后，用于阻止其他事务插入数据到Gap中
      2. 使用场景例子
         1. SELECT c1 FROM t WHERE c1 BETWEEN 10 AND 20 FOR UPDATE;
         2. 会产生一个Gap Lock，会锁住10-20之间所有记录
         3. 即阻止其他事务插入 t.c1=15 的记录，即使15值已经存在
      3. Gap lock 的跨度可以是N个索引值(N >= 0)
      4. 但搜索条件是单列时，
         1. 如果通过唯一索引搜索行，则不会产生Gap Lock，
         2. 如果通过非索引或非唯一索引搜索行，则会在行前加个Gap Lock
      5. 如果搜索条件是多列的组合唯一索引，则会产生Gap Lock
      6. 相同Gap位置，允许多个事务的Gap Locks共存，如果从索引中清除一条记录时，则必须合并记录上由不同事务持有的Gap Lock
      7. 可以通过将隔离级别设置为READ\_COMMITTED或者启用 innodb\_locks\_unsafe\_for\_binlog 配置，在搜索和索引扫描操作中禁用Gap lock，使Gap Lock仅在外键约束检查和唯一键检查时产生

### Next-Key Locks

https://dev.mysql.com/doc/refman/5.7/en/innodb-locking.html#innodb-next-key-locks

* + - 1. 是一种Record Locks和在索引记录之前的Gap Lock的组合Locks
      2. 如果一个请求持有索引中记录R的shared或exclusive lock，其他请求就不能在索引序列中的记录R前插入新的索引记录，即在最大索引值之后的Gap位置设置lock
      3. 在innodb的默认事务隔离级别(REPEATABLE\_READ)下，next-key lock可以防止phantom rows出现

### Insert Intention Locks

https://dev.mysql.com/doc/refman/5.7/en/innodb-locking.html#innodb-insert-intention-locks

* + - 1. 是一种Gap lock，在insert操作执行前设置
      2. 当多个事务插入数据到同一个Gap的不同位置的时，这个锁用于表明事务插入操作不需要阻塞
         1. 假设索引记录值4和7，两个事务分别尝试插入5和6，这时都会在获得各自对应的exclusive lock前锁住4和7之间的Insert intention Locks, 但因为插入行不冲突，所以，两个事务不会互相阻塞
         2. 还有个例子建议看官方文档

### AUTO-INC Locks

https://dev.mysql.com/doc/refman/5.7/en/innodb-locking.html#innodb-auto-inc-locks

* + - 1. 是一种特别的table-level lock，用于事务将数据插入到有自增字段的表中
      2. 当一个事务插入数据时，其他事务必须等待，以便当前事务获得连续的字段值
      3. 可用通过配置innodb\_autoinc\_lock\_mode参数指定此锁的实现算法

### Predicate Locks

https://dev.mysql.com/doc/refman/5.7/en/innodb-locking.html#innodb-predicate-locks

* + - 1. 用于支持对spatial 字段索引
      2. 对于spatial index，由于在多维度数据中，没有绝对的序列观念，所以next-key lock无法很好支持REPEATABLE READ 或者 SERIALIZABLE 事务隔离级别，因为无法清楚获取”下一个”key
      3. 每个spatial index 包含一组minimum bounding rectangle(MBR)值，innodb强制对匹配查询条件的记录设置predicate lock，阻塞非当前事务修改/插入查询条件匹配的行

## Phantom Row

## Deadlock

# 日志(Log)

# 事务(Transaction)

https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-model.html

## 概述

* + - 1. ACID 模型是一组强调数据库可靠性的设计原则, innodb 严格遵守 此模型
      2. Innodb的事务模型，目标是结合多版本数据库的最佳特性与传统的两阶段锁
      3. 所有用户行为都在一个事务中，
      4. 默认情况下，autocommit是启用的：
         1. 每个SQL statement都各自形成自己的事务，默认情况下，针对每个新链接：

如果SQL statement执行没有抛出错误 ，在执行之后自动执行一次commit

如果SQL statement执行抛出错误，根据错误决定提交当前事务或回滚

* + - * 1. 也可以通过在逻辑开始处指定START\_TRANSACTION 或 BEGIN statement，并在逻辑结束时执行 COMMIT或ROLLBACK statement ，来实现同个事务中执行多个statement
      1. 如果autocommit禁用时：
         1. 当前session的事务将一直开放，直到执行COMMIT或ROLLBACK statement 结束当前事务
         2. 如果当前session结束前没有以commit statement 结束最后的事务，将会回滚最后一个事务
      2. 有些statement会在执行前，产生一个隐式的提交操作以结束当前事务：
      3. COMMIT 和 ROLLBACK statement：
         1. COMMIT：表明将当前事务对数据的修改持久化并对其他session可见
         2. ROLLBACK：表明要取消当前事务所作的所有修改
         3. 两者都会释放所有当前事务所获得的Locks
      4. Consistent read

## ACID

### https://en.wikipedia.org/wiki/ACID

https://dev.mysql.com/doc/refman/5.7/en/mysql-acid.html

### A(atomicity)

* + - 1. 事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用
      2. 涉及功能点：自动提交、commit声明、rollback声明、操作INFORMATION\_SCHEMA 表的数据

### C(cosistency)

* + - 1. 在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。
      2. 涉及功能点：doublewrite buffer、crash recovery

### I(isolation)

* + - 1. 并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据是独立的
      2. 涉及功能点：自动提交、配置隔离等级、lock的low-level detail

### D(durability)

* + - 1. 一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。
      2. 涉及功能：
         1. doublewrite buffer开关
         2. 配置项：innodb\_flush\_log\_at\_trx\_commit、sync\_binlog、innodb\_file\_per\_table
         3. write buffer
         4. Battery-backed cache
         5. 操作系统是否支持fsync()
         6. UPS
         7. 备份策略

## 事务隔离(Transaction Isolation)

### https://en.wikipedia.org/wiki/Isolation\_(database\_systems)

### 概述

* + - 1. ISO和ANIS SQL 标准制定了四种事务隔离级别(SLQ:1992)从低到高：READ\_UNCOMMITTED、READ\_COMMITTED、REPEATABLE\_READ(2.9999。)、SERIALIZABLE(3。)
         1. 标准的REPEATABLE\_READ没有幻读保护，所以是2.9999。隔离
         2. SQL和SQL2标准的默认事务隔离级别是SERIALIZABLE
      2. 不是所有数据库都遵循这个标准
      3. Innodb通过不同的lock策略支持了所有4种事务隔离级别，默认隔离级别是REPEATABLE\_READ

### READ UNCOMMITED

https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html#isolevel\_read-uncommitted

* + - 1. SELECT statement 采用无锁方式执行，很可能获取记录的早期版本
      2. 无法保证读一致性，会产生 dirty read
      3. 其他方面，跟READ\_COMMITED一致

### READ COMMITTED

https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html#isolevel\_read-committed

* + - 1. 同个事务中，通过第一次读取数据时建立的快照，实现读一致性
      2. 对于 Locking read(SELECT \* FROM t FOR UPDATE 或者 LOCK IN SHARE MODE)、UPDATE以及DELETE statements，只锁索引记录，不锁gap，因此允许在被锁定的记录旁边插入新记录，可能会导致phantom problems
      3. 仅在需要检查外键约束和唯一键约束时，才使用Gap locks
      4. 此隔离级别只支持基于行的二进制日志，如果配置binlog\_format=MIXED，系统会自动的转为基于行的日志
      5. 对于UPDATE 或 DETELE statements，只会保持锁定条件匹配的行，不匹配的行会被释放，会大程度的减少deadlock(但仍可能发生deadlock)
      6. 对于 UPDATE statement：
         1. 如果指定的行R已经被锁定，Innodb执行一个”semi-consistent” read，即返回R最后一次提交的版本，用于判断R是否匹配WHERE条件
         2. 如果R匹配，则重新读取R最新记录，并尝试锁定R或者阻塞等待

### REPEATABLE READ

https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html#isolevel\_repeatable-read

* + - 1. Innodb 默认事务隔离等级，与SQL标准不同的是，Innodb在此隔离级别下，使用了next-key lock 避免了幻读的产生，达到了SQL标准的SERIALIZABLE隔离级别
      2. 同个事务中，通过第一次读取数据时建立的快照，实现读一致性
      3. 对于 Locking read(SELECT \* FROM t FOR UPDATE 或者 LOCK IN SHARE MODE)、UPDATE以及DELETE statements的lock策略 取决于：
         1. 对于带唯一索引字段的搜索条件，仅锁定发现索引记录，不锁记录前的gap
         2. 对于其他搜索条件，锁定扫描到的索引范围，通过gap lock或 next-key locks 阻塞其他事务在范围内的插入数据请求

### SERIALIZABLE

https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html#isolevel\_serializable

* + - 1. 在innodb中，类似于REPEATABLE\_READ
      2. 如果禁用了autocommit，Innodb会自动在每个SELECT statements后面加上LOCK IN SHARE MODE
      3. 如果启用了autocommit，则SELECT会有自己的事务
      4. 不支持一致性的非锁定读

## redo log

## undo log

## purge

## 事务类型

## XA事务

# 其他

## 、配置

## 、优化

## 、限制