# 数据库知识点

## 索引

索引是帮助数据库（MySQL）高效获取数据的排好序的数据结构

索引是关系型数据库中给数据库表中一列或多列的值排序后的存储结构，SQL的主流索引结构有B+树以及Hash结构，聚集索引以及非聚集索引用的是B+树索引。

### 1、索引类型以及各种类型的解释

MySQL里的索引类型主要有以下几种。

### 从数据结构角度

1、**B+树索引(O(log(n)))：关于B+树索引，**可以参考 MySQL索引背后的数据结构及算法原理

2、**hash索引**：（只能=值查询，检索效率高，只有memory引擎支持）

3、**FULLTEXT**索引（现在MyISAM和InnoDB引擎都支持了）

4、**R-Tree索引**（用于对GIS数据类型创建SPATIAL索引）

### 从物理存储角度

1、**聚集索引**（clustered index）

2、**非聚集索引**（non-clustered index）

聚集索引和非聚集索引的根本区别是表中记录的物理顺序和索引的排列顺序是否一致

**聚集索引的表中记录的物理顺序与索引的排列顺序一致**，一个表中只能拥有一个聚集索引。

优点是查询速度快，因为一旦具有第一个索引值的记录被找到，具有连续索 引值的记录也一定物理的紧跟其后。

缺点是对表进行修改速度较慢，这是为了保持表中的记录的物理顺序与索引 的顺序一致，而把记录插入到数据页的相应位置，必须在数据页中进行数据重排，降低了执行速度。在插入新记录时数据文件为了维持 B+Tree 的特性而频繁的分裂调整，十分低效。

**非聚集索引的记录的物理顺序和索引的顺序不一致。**一个表中可以拥有多个非聚集索引。

其他方面的区别：

1.聚集索引和非聚集索引都采用了 B+树的结构，但非聚集索引的叶子层并 不与实际的数据页相重叠，而采用叶子层包含一个指向表中的记录在数据页中 的指针的方式。聚集索引的叶节点就是数据节点，而非聚集索引的叶节点仍然 是索引节点。

2.非聚集索引添加记录时，不会引起数据顺序的重组。

## 从逻辑角度

1、**主键索引**：主键索引是一种特殊的唯一索引，不允许有空值

2、**普通索引或者单列索引**

3、**多列索引（复合索引）**：复合索引指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用复合索引时遵循最左前缀集合

4、**唯一索引**或者非唯一索引

5、**空间索引**：空间索引是对空间数据类型的字段建立的索引，MYSQL中的空间数据类型有4种，分别是GEOMETRY、POINT、LINESTRING、POLYGON。MYSQL使用SPATIAL关键字进行扩展，使得能够用于创建正规索引类型的语法创建空间索引。创建空间索引的列，必须将其声明为NOT NULL，空间索引只能在存储引擎为MYISAM的表中创建

```
CREATE TABLE table_name[col_name data type]
[unique|fulltext|spatial][index|key][index_name](col_name[length])
[asc|desc]
```

1、unique|fulltext|spatial为可选参数，分别表示唯一索引、全文索引和空间索引；

2、index和key为同义词，两者作用相同，用来指定创建索引

3、col_name为需要创建索引的字段列，该列必须从数据表中该定义的多个列中选择；

4、index_name指定索引的名称，为可选参数，如果不指定，MYSQL默认col_name为索引值；

5、length为可选参数，表示索引的长度，只有字符串类型的字段才能指定索引长度；

6、asc或desc指定升序或降序的索引值存储

# 2、索引的底层数据结构

B+TREE索引

在B-Tree中按key检索数据的算法非常直观：首先从根节点进行二分查找，如果找到则返回对应节点的data，否则对相应区间的指针指向的节点递归进行查找，直到找到节点或找到null指针，前者查找成功，后者查找失败。

B+Tree是B-Tree的变种，在B+Tree的每个叶子节点增加一个指向相邻叶子节点的指针，就形成了带有顺序访问指针的B+Tree。做这个优化的目的是为了提高区间访问的性能

HASH索引

哈希索引基于hash表实现，类似于Java中的HashMap，通过计算key的hash值映射对应的value，在不发生hash冲突的情况下时间复杂度为常数级别，MySQL的hash索引会对所有的索引列计算一个hash码，由于hash的索引的特点，它的缺点也显而易见，只有精确匹配索引所有列的查询才有效，hash索引数据也并不是按照索引值顺序存储的，所以也无无法用于排序，只支持等值查询，不支持范围查询。他是Memory引擎的默认索引类型，也是Memory引擎速度快的原因之一。

在InnoDB有一个特殊的功能叫做自适应哈希索引，当它发现某些索引值被使用的非常频繁时，它会在内存中基于B+树索引之上再创建一个hash索引，加快数据的查找速度。

全文索引

这是一种特殊类型的索引，它查找的是文本中的关键词，而不是直接比较索引中的值，全文索引更类似于搜索引擎做的事情，实际生产中我们一般不会使用MySQL来做类似搜索引擎的工作。

## 3、索引的优缺点

**索引的优点**

1. 通过创建唯一索引，可以保证数据库每一行数据的唯一性
2. 可以大大提高查询速度
3. 可以加速表与表的连接
4. 可以显著的减少查询中分组和排序的时间。

**索引的缺点**

1. 创建索引和维护索引需要时间，而且数据量越大时间越长
2. 创建索引需要占据磁盘的空间，如果有大量的索引，可能比数据文件更快达到最大文件尺寸
3. 当对表中的数据进行增加，修改，删除的时候，索引也要同时进行维护，降低了数据的维护速度

## 4、复合索引使用的注意点

复合索引遵守“最左前缀”原则，即在查询条件中使用了复合索引的第一个字段，索引才会被使用。因此，在复合索引中索引列的顺序至关重要。如果不是按照索引的最左列开始查找，则无法使用索引。

# 事务

## 1、什么是事务?

一个Session中所进行所有的操作，要么同时成功，要么同时失败

## 2、ACID的解释

ACID — 数据库事务正确执行的四个基本要素

原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。

原子性是指事务是一个不可再分割的工作单元，事务中的操作要么都发生，要么都不发生。

一致性是指在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。这是说数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。

多个事务并发访问时，事务之间是隔离的，一个事务不应该影响其它事务运行效果。

持久性，意味着在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

## 4、数据库的四大隔离，解决的问题

四大隔离级别:

Read uncommitted（未提交读） -- 不防止任何隔离性问题,具有脏读/不可重复度/虚读(幻读)问题

Read committed（提交读） -- 可以防止脏读问题,但是不能防止不可重复度/虚读(幻读)问题

Repeatable read（可重复读） -- 可以防止脏读/不可重复读问题,但是不能防止虚读(幻读)问题

Serializable（可串行化） -- 数据库被设计为单线程数据库,可以防止上述所有问题

# 5、不同数据库和引擎的默认隔离级别

Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别

MySQL实现了这4种隔离级别，默认的隔离级别在Innodb中是Repeatable Read。而Oracle数据库只实现了Read Committed和Serializable两种，它的默认隔离级别是Read Committed。

Innodb在Repeatable Read隔离级别中，通过`Next-Key Lock`锁的算法来避免幻读的产生，这与其他关系型数据库不同。所以说，InnoDB存储引擎在默认的`Repeatable Read`隔离级别下已经能完全保证事务的隔离性要求，即达到SQL标准的`Serializable`隔离级别。

MyISAM不支持事务

# 6、事务操作

1. 开启事务： start transaction;
2. 回滚：rollback;
3. 提交：commit;

# 7、事务提交的两种方式

自动提交：

- mysql就是自动提交的
- 一条DML(增删改)语句会自动提交一次事务。

手动提交：

- Oracle 数据库默认是手动提交事务
- 需要先开启事务，再提交

# 8、事务回滚原理

InnoDB存储引擎还提供了两种事务日志：redo log(重做日志)和undo log(回滚日志)。其中redo log用于保证事务持久性；undo log则是事务原子性和隔离性实现的基础。

下面说回undo log。实现原子性的关键，是当事务回滚时能够撤销所有已经成功执行的sql语句。InnoDB实现回滚，靠的是undo log：当事务对数据库进行修改时，InnoDB会生成对应的undo log；如果事务执行失败或调用了rollback，导致事务需要回滚，便可以利用undo log中的信息将数据回滚到修改之前的样子。

undo log属于逻辑日志，它记录的是sql执行相关的信息。当发生回滚时，InnoDB会根据undo log的内容做与之前相反的工作：对于每个insert，回滚时会执行delete；对于每个delete，回滚时会执行insert；对于每个update，回滚时会执行一个相反的update，把数据改回去。

以update操作为例：当事务执行update时，其生成的undo log中会包含被修改行的主键(以便知道修改了哪些行)、修改了哪些列、这些列在修改前后的值等信息，回滚时便可以使用这些信息将数据还原到update之前的状态。

## 9、事务的传播行为

说说Spring中的事务传播行为，事务传播行为是Spring框架提供的一种事务管理方式，它不是数据库提供的。

事务的传播行为，默认值为 Propagation.REQUIRED。可以手动指定其他的事务传播行为，如下：

- Propagation.REQUIRED

如果当前存在事务，则加入该事务，如果当前不存在事务，则创建一个新的事务。

- Propagation.SUPPORTS

如果当前存在事务，则加入该事务；如果当前不存在事务，则以非事务的方式继续运行。

- Propagation.MANDATORY

如果当前存在事务，则加入该事务；如果当前不存在事务，则抛出异常。

- Propagation.REQUIRES_NEW

重新创建一个新的事务，如果当前存在事务，延缓当前的事务。

- Propagation.NOT_SUPPORTED

以非事务的方式运行，如果当前存在事务，暂停当前的事务。

- Propagation.NEVER

以非事务的方式运行，如果当前存在事务，则抛出异常。

- Propagation.NESTED

如果没有，就新建一个事务；如果有，就在当前事务中嵌套其他事务。

## 10、什么是脏读，不可重复读，幻读

1、脏读：事务A读到了事务B未提交的数据。

2、不可重复读：事务A第一次查询得到一行记录row1，事务B提交修改后，事务A第二次查询得到row1，但列内容发生了变化。

3、幻读：事务A第一次查询得到一行记录row1，事务B提交修改后，事务A第二次查询得到两行记录row1和row2。

# 锁

## 1、有哪些锁，及实现方式

数据库有乐观锁和悲观锁

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

- **在查询完数据的时候就把事务锁起来，直到提交事务**
- 实现方式：使用数据库中的锁机制

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

- **在修改数据的时候把事务锁起来，通过version的方式来进行锁定**
- 实现方式：使用version版本或者时间戳

# 其他

## 1、数据库有哪些引擎?

**ISAM**：它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。

**MyISAM**：MyISAM是MySQL的ISAM扩展格式和缺省的数据库引擎。插入数据快，空间和内存使用比较低。如果表主要是用于插入新记录和读出记录，那么选择MyISAM能实现处理高效率。如果应用的完整性、并发性要求比较低，也可以使用。

**InnoDB**：支持事务处理，支持外键，支持崩溃修复能力和并发控制。如果需要对事务的完整性要求比较高（比如银行），要求实现并发控制（比如售票），那选择InnoDB有很大的优势。如果需要频繁的更新、删除操作的数据库，也可以选择InnoDB，因为支持事务的提交（commit）和回滚（rollback）。

**MEMORY:** MEMORY是MySQL中一类特殊的存储引擎。所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，可以选择MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

注意，同一个数据库也可以使用多种存储引擎的表。如果一个表要求比较高的事务处理，可以选择InnoDB。这个数据库中可以将查询要求比较高的表选择MyISAM存储。如果该数据库需要一个用于查询的临时表，可以选择MEMORY存储引擎。

## 2、怎么备份表，删除表的几个关键字的区别

### 备份表：

将建表结构与插入数据的sql语句生成并保存，下次如果需要改结构和数据，直接将数据语句执行即可。

1. 备份student_1数据库：mysqldump -uroot -p student_1 > E:/back.sql

   则在E盘新形成了一个back.sql的文件，文件中的内容为一系列sql语句！

2. 将备份的数据还原

   所谓备份数据的还原，就是将刚刚生成的sql语句，执行即可！

   首先创建一个新的数据库：create database student_3

   然后选中该数据库：use student_3;

   然后在该数据库中还原数据，则原先的数据库中的表都会备份到该数据库中：source E:/back.sql;

- 备份整个数据内的表：Mysqldump -uroot -p db_name > bak.sql

- 备份数据库内的某张表：mysqldump -uroot -p student_1 teacher_class > e:/teacher_class.sql

### 删除表的几个关键字的区别:

1. delete

```
delete from Student where [条件]
```

- 执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中保存以便进行进行回滚操作。

2. truncate

```
truncate table Student
```

- 一次性地从表中删除所有的数据并不把单独的删除操作记录记入日志保存，删除行是不能恢复的。并且在删除的过程中不会激活与表有关的删除触发器。执行速度快。TRUNCATE 只能对TABLE；DELETE可以是table和view

3. drop

```
drop table Student1
```

- 清除关于当前表的所有信息，表不再存在

表和索引所占空间。当表被TRUNCATE 后，这个表和索引所占用的空间会恢复到初始大小，而DELETE操作不会减少表或索引所占用的空间。drop语句将表所占用的空间全释放掉。

## 3、数据库优化

1、选取最适用的字段属性

在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

另外一个提高效率的方法是在可能的情况下，应该尽量把字段设置为NOTNULL，这样在将来执行查询的时候，数据库不用去比较NULL值。 对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为ENUM类型。因为在MySQL中，ENUM类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

2、使用连接（JOIN）来代替子查询(Sub-Queries)

有些情况下，子查询可以被更有效率的连接（JOIN）..替代。

如果使用连接（JOIN）..来完成这个查询工作，速度将会快很多。

连接（JOIN）..之所以更有效率一些，是因为MySQL不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

3、使用联合(UNION)来代替手动创建的临时表

MySQL从4.0的版本开始支持union查询，它可以把需要使用临时表的两条或更多的select查询合并的一个查询中。在客户端的查询会话结束的时候，临时表会被自动删除，从而保证数据库整齐、高效。使用union来创建查询的时候，我们只需要用UNION作为关键字把多个select语句连接起来就可以了，要注意的是所有select语句中的字段数目要想同。下面的例子就演示了一个使用UNION的查询。

```
SELECT Name,Phone FROM client UNION
SELECT Name,BirthDate FROM author UNION
SELECT Name,Supplier FROM product
```

4、事务

可以保持数据库中数据的一致性和完整性。事务以BEGIN关键字开始，COMMIT关键字结束。在这之间的一条SQL操作失败，那么，ROLLBACK命令就可以把数据库恢复到BEGIN开始之前的状态。


```
SELECT Name,Phone FROM client UNION
SELECT Name,BirthDate FROM author UNION
SELECT Name,Supplier FROM product
```


事务的另一个重要作用是当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰。

5、锁定表

尽管事务是维护数据库完整性的一个非常好的方法，但却因为它的独占性，有时会影响数据库的性能，尤其是在很大的应用系统中。由于在事务执行的过程中，数据库将会被锁定，因此其它的用户请求只能暂时等待直到该事务结束。如果一个数据库系统只有少数几个用户来使用，事务造成的影响不会成为一个太大的问题；但假设有成千上万的用户同时访问一个数据库系统，例如访问一个电子商务网站，就会产生比较严重的响应延迟。

其实，有些情况下我们可以通过锁定表的方法来获得更好的性能。下面的例子就用锁定表的方法来完成前面一个例子中事务的功能。

```
LOCK TABLE inventory WRITE SELECT Quantity FROM inventory WHERE
Item='book';

...

UPDATE inventory SET Quantity=11 WHERE Item='book'; UNLOCKTABLES
```

这里，我们用一个select语句取出初始数据，通过一些计算，用update语句将新值更新到表中。包含有WRITE关键字的LOCKTABLE语句可以保证在UNLOCKTABLES命令被执行之前，不会有其它的访问来对inventory进行插入、更新或者删除的操作。

6、使用外键

**锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键。**

例如，外键可以保证每一条销售记录都指向某一个存在的客户。在这里，外键可以把customerinfo表中的CustomerID映射到salesinfo表中CustomerID，任何一条没有合法CustomerID的记录都不会被更新或插入到salesinfo中。

```
CREATE TABLE customerinfo( CustomerIDINT NOT NULL,PRIMARYKEY(CustomerID))TYPE=INNODB;

CREATE TABLE salesinfo( SalesIDNT NOT NULL,CustomerIDINT NOT NULL,

PRIMARYKEY(CustomerID,SalesID),

FOREIGNKEY(CustomerID) REFERENCES customerinfo(CustomerID) ON DELETE CASCADE)TYPE=INNODB;
```

注意例子中的参数“ON DELETE CASCADE”。该参数保证当customerinfo表中的一条客户记录被删除的时候，salesinfo表中所有与该客户相关的记录也会被自动删除。如果要在MySQL中使用外键，一定要记住在创建表的时候将表的类型定义为事务安全表InnoDB类型。该类型不是MySQL表的默认类型。定义的方法是在CREATETABLE语句中加上TYPE=INNODB。如例中所示。

7、使用索引

索引是提高数据库性能的常用方法，它可以令数据库服务器以比没有索引快得多的速度检索特定的行，尤其是在查询语句当中包含有MAX(),MIN()和ORDERBY这些命令的时候，性能提高更为明显。

8、优化的查询语句

绝大多数情况下，使用索引可以提高查询的速度，但如果SQL语句使用不恰当的话，索引将无法发挥它应有的作用。

下面是应该注意的几个方面。

- 首先，最好是在相同类型的字段间进行比较的操作。

在MySQL3.23版之前，这甚至是一个必须的条件。例如不能将一个建有索引的INT字段和BIGINT字段进行比较；但是作为特殊的情况，在CHAR类型的字段和VARCHAR类型字段的字段大小相同的时候，可以将它们进行比较。

- 其次，在建有索引的字段上尽量不要使用函数进行操作。

> 例如，在一个DATE类型的字段上使用YEAE()函数时，将会使索引不能发挥应有的作用。所以，下面的两个查询虽然返回的结果一样，但后者要比前者快得多。

- 第三，在搜索字符型字段时，我们有时会使用LIKE关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

例如下面的查询将会比较表中的每一条记录。

```
SELECT * FROM books

WHERE name like"MySQL%"
```

但是如果换用下面的查询，返回的结果一样，但速度就要快上很多：

```
SELECT * FROM books

WHERE name>="MySQL" andname <"MySQM"
```

至此，应该注意避免在查询中让MySQL进行自动类型转换，因为转换过程也会使索引变得不起作用。

# 4、数据库范式

**第一范式（1NF）无重复的列**

所谓第一范式（1NF）是**指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性**。

**第二范式（2NF）属性完全依赖于主键**

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。

**第三范式（3NF）**

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，**第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息**。

# 5、分表

关系型数据库本身比较容易成为系统瓶颈，单机存储容量、连接数、处理能力都有限。当单表的数据量达到1000W或100G以后，由于查询维度较多，即使添加从库、优化索引，做很多操作时性能仍下降严重。此时就要考虑对其进行切分了，切分的目的就在于减少数据库的负担，缩短查询时间。

数据切分根据其切分类型，可以分为两种方式：垂直（纵向）切分和水平（横向）切分

垂直切分常见有垂直分库和垂直分表两种。

垂直切分的优点：

- 解决业务系统层面的耦合，业务清晰
- 与微服务的治理类似，也能对不同业务的数据进行分级管理、维护、监控、扩展等
- 高并发场景下，垂直切分一定程度的提升IO、数据库连接数、单机硬件资源的瓶颈

缺点：

- 部分表无法join，只能通过接口聚合方式解决，提升了开发的复杂度
- 分布式事务处理复杂
- 依然存在单表数据量过大的问题（需要水平切分）

水平切分分为库内分表和分库分表，是根据表内数据内在的逻辑关系，将同一个表按不同的条件分散到多个数据库或多个表中，每个表中只包含一部分数据，从而使得单个表的数据量变小，达到分布式的效果。

水平切分的优点：

- 不存在单库数据量过大、高并发的性能瓶颈，提升系统稳定性和负载能力
- 应用端改造较小，不需要拆分业务模块

缺点：

- 跨分片的事务一致性难以保证
- 跨库的join关联查询性能较差
- 数据多次扩展难度和维护量极大