第三版：MySQL 80 道

为什么要使用数据库

数据保存在内存

优点：

存取速度快

缺点：

数据不能永久保存

数据保存在文件

优点：

数据永久保存

缺点：

1、 速度比内存操作慢，频繁的 IO 操作。

2、 查询数据不方便

数据保存在数据库

1、 数据永久保存

2、 使用 SQL 语句，查询方便效率高。

3、 管理数据方便

什么是 SQL?

结构化查询语言(Structured Query Language)

为什么要使用数据库

数据保存在内存

优点：

存取速度快

缺点：

数据不能永久保存

数据保存在文件

优点：数据永久保存

缺点：

1、 速度比内存操作慢，频繁的 IO 操作。

2、 查询数据不方便

数据保存在数据库

1、 数据永久保存

2、 使用 SQL 语句，查询方便效率高。

3、 管理数据方便

什么是 SQL?

结构化查询语言(Structured Query Language)简称 SQL，是一种数据库查询语言。

作用：

用于存取数据、查询、更新和管理关系数据库系统。

什么是 MySQL?

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL 是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件之一。在 Java 企业级开发中非常常用，因为 MySQL 是开源免费的，并且方便扩展。

MySql, Oracle，Sql Service 的区别

1、 Sql Service 只能在 Windows 上使用，而 MySql 和 Oracle 可以在其他系统上使用， 而且可以支持数据库不同系统之间的移植

2、 MySql 开源免费的，Sql Service 和 Oracle 要钱。

3、 我从小到大排序哈，MySql 很小，Sql Service 居中，Oracle 最大

4、 Oracle 支持大并发量，大访问量，Sql Service 还行，而 MySql 的话压力没这么大，因此现在的 MySql 的话最好是要使用集群或者缓存来搭配使用

5、 Oracle 支持多用户不同权限来进行操作，而 MySql 只要有登录权限就可操作全部数据库

6、 安装所用的空间差别也是很大的，MySQL 安装完后才几百 M 而 Oracle 有几 G 左右，且使用的时候 Oracle 占用特别大的内存空间和其他机器性能。

7、 做分页的话，MySql 使用 Limit，Sql Service 使用 top，Oracle 使用 row

8、 Oracle 没有自动增长类型，MySQL 和 Sql Service 一般使用自动增长类型

数据库三大范式是什么

第一范式：每个列都不可以再拆分。
第二范式：在第一范式的基础上，非主键列完全依赖于主键，而不能是依赖于主键的一部分。
第三范式：在第二范式的基础上，非主键列只依赖于主键，不依赖于其他非主键。

在设计数据库结构的时候，要尽量遵守三范式，如果不遵守，必须有足够的理由。比如性能。事实上我们经常会为了性能而妥协数据库的设计。

MySQL 有关权限的表都有哪几个

MySQL 服务器通过权限表来控制用户对数据库的访问，权限表存放在 MySQL 数据库里，由 MySQL_install_db 脚本初始化。这些权限表分别 user，db，table_priv，columns_priv 和 host。

下面分别介绍一下这些表的结构和内容：

1、 user 权限表：记录允许连接到服务器的用户帐号信息，里面的权限是全局级的。

2、 db 权限表：记录各个帐号在各个数据库上的操作权限。

3、 table_priv 权限表：记录数据表级的操作权限。

4、 columns_priv 权限表：记录数据列级的操作权限。

5、 host 权限表：配合 db 权限表对给定主机上数据库级操作权限作更细致的控制。这个权限表不受 GRANT 和 REVOKE 语句的影响。

MySQL 的 binlog 有有几种录入格式？分别有什么区别？

有三种格式，statement，row 和 mixed。

1、 statement 模式下，每一条会修改数据的 sql 都会记录在 binlog 中。不需要记录每一行的变化，减少了 binlog 日志量，节约了 IO，提高性能。由于 sql 的执行是有上下文的，因此在保存的时候需要保存相关的信息，同时还有一些使用了函数之类的语句无法被记录复制。

2、 row 级别下，不记录 sql 语句上下文相关信息，仅保存哪条记录被修改。记录单元为每一行的改动，基本是可以全部记下来但是由于很多操作，会导致大量行的改动(比如 alter table)，因此这种模式的文件保存的信息太多，日志量太大。

3、 mixed，一种折中的方案，普通操作使用 statement 记录，当无法使用 statement 的时候使用 row。

此外，新版的 MySQL 中对 row 级别也做了一些优化，当表结构发生变化的时候，会记录语句而不是逐行记录

数据库经常使用的函数

1、 count(*/column)：返回行数

2、 sum(column)：返回指定列中唯一值的和

3、 max(column)：返回指定列或表达式中的数值最大值

4、 min(column)：返回指定列或表达式中的数值最小值

5、 avg(column)：返回指定列或表达式中的数值平均值

6、 date（Expression）：返回指定表达式代表的日期值

MySQL 有哪些数据类型

分类	类型名称	说明
整数类型	tinyInt	很小的整数(8 位二进制)
整数类型	smallint	小的整数(16 位二进制)

整数类型	mediumint	中等大小的整数(24 位二进制)
整数类型	int(integer)	普通大小的整数(32 位二进制)
小数类型	float	单精度浮点数
小数类型	double	双精度浮点数
小数类型	decimal(m,d)	压缩严格的定点数
日期类型	year	YYYY 1901~2155
日期类型	time	HH:MM:SS -838:59:59~838:59:59
日期类型	date	YYYY-MM-DD 1000-01-01~9999-12-3
日期类型	datetime	YYYY-MM-DD HH:MM:SS 1000-01-01 00:00:00~9999-12-31 23:59:59
日期类型	timestamp	YYYY-MM-DD HH:MM:SS 19700101 00:00:01 UTC~2038-01-19 03:14:07UTC
文本、二进制类型	CHAR(M)	M 为 0~255 之间的整数
文本、二进制类型	VARCHAR(M)	M 为 0~65535 之间的整数
文本、二进制类型	TINYBLOB	允许长度 0~255 字节
文本、二进制类型	BLOB	允许长度 0~65535 字节
文本、二进制类型	MEDIUMBLOB	允许长度 0~167772150 字节
文本、二进制类型	LONGBLOB	允许长度 0~4294967295 字节
文本、二进	TINYTEXT	允许长度 0~255 字节

制类型		
文本、二进制类型	TEXT	允许长度 0~65535 字节
文本、二进制类型	MEDIUMTEXT	允许长度 0~167772150 字节
文本、二进制类型	LONGTEXT	允许长度 0~4294967295 字节
文本、二进制类型	VARBINARY(M)	允许长度 0~M 个字节的变长字节字符串
文本、二进制类型	BINARY(M)	允许长度 0~M 个字节的定长字节字符串

整数类型

包括 TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT，分别表示 1 字节、2 字节、3 字节、4 字节、8 字节整数。任何整数类型都可以加上 UNSIGNED 属性，表示数据是无符号的，即非负整数。

长度：整数类型可以被指定长度，例如：INT(11)表示长度为 11 的 INT 类型。长度在大多数场景是没有意义的，它不会限制值的合法范围，只会影响显示字符的个数，而且需要和 UNSIGNED ZEROFILL 属性配合使用才有意义。

例子：

假定类型设定为 INT(5)，属性为 UNSIGNED ZEROFILL，如果用户插入的数据为 12 的话，那么数据库实际存储数据为 00012。

实数类型

1、 包括 FLOAT、DOUBLE、DECIMAL。

2、 DECIMAL 可以用于存储比 BIGINT 还大的整型，能存储精确的小数。

3、 而 FLOAT 和 DOUBLE 是有取值范围的，并支持使用标准的浮点进行近似计算。

4、 计算时 FLOAT 和 DOUBLE 相比 DECIMAL 效率更高一些，DECIMAL 你可以理解成是用字符串进行处理。

字符串类型

包括 VARCHAR、CHAR、TEXT、BLOB

1、 VARCHAR 用于存储可变长字符串，它比定长类型更节省空间。

2、 VARCHAR 使用额外 1 或 2 个字节存储字符串长度。列长度小于 255 字节时，使用 1 字节表示，否则使用 2 字节表示。

3、 VARCHAR 存储的内容超出设置的长度时，内容会被截断。

4、 CHAR 是定长的，根据定义的字符串长度分配足够的空间。

5、 CHAR 会根据需要使用空格进行填充方便比较。

6、 CHAR 适合存储很短的字符串，或者所有值都接近同一个长度。

7、 CHAR 存储的内容超出设置的长度时，内容同样会被截断。

使用策略：

1、 对于经常变更的数据来说，CHAR 比 VARCHAR 更好，因为 CHAR 不容易产生碎片。

2、 对于非常短的列，CHAR 比 VARCHAR 在存储空间上更有效率。

3、 使用时要注意只分配需要的空间，更长的列排序时会消耗更多内存。

4、 尽量避免使用 TEXT/BLOB 类型，查询时会使用临时表，导致严重的性能开销。

枚举类型（ENUM）

1、 把不重复的数据存储为一个预定义的集合。

2、 有时可以使用 ENUM 代替常用的字符串类型。

3、 ENUM 存储非常紧凑，会把列表值压缩到一个或两个字节。

4、 ENUM 在内部存储时，其实存的是整数。

5、 尽量避免使用数字作为 ENUM 枚举的常量，因为容易混乱。

6、 排序是按照内部存储的整数

日期和时间类型

1、 尽量使用 timestamp，空间效率高于 datetime，

2、 用整数保存时间戳通常不方便处理。

3、 如果需要存储微妙，可以使用 bigint 存储。

4、 看到这里，这道真题是不是就比较容易回答了。

MySQL 存储引擎 MyISAM 与 InnoDB 区别

存储引擎 Storage engine：MySQL 中的数据、索引以及其他对象是如何存储的，是一套文件系统的实现。

常用的存储引擎有以下：

Innodb 引擎：Innodb 引擎提供了对数据库 ACID 事务的支持。并且还提供了行级锁和外键的约束。它的设计的目标就是处理大数据容量的数据库系统。

MyIASM 引擎(原本 MySQL 的默认引擎)：不提供事务的支持，也不支持行级锁和外键。

MEMORY 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。

MyISAM 与 InnoDB 区别

比较	MyISAM	Innodb
存储结构	每张表被存放在三个文件：frm-表格定义、MYD(MYData)-数据文件、MYI(MYIndex)-索引文件	所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB 表的大小只受限于操作系统文件的大小，一般为 2GB
存储空间	MyISAM 可被压缩，存储空间较小	InnoDB 的表需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引
可移植性、备份及恢复	由于 MyISAM 的数据是以文件的形式存储，所以在跨平台的数	免费的方案可以是拷贝数据文件、备份 binlog，或者用

	据转移中会很方便。在备份和恢复时可单独针对某个表进行操作	MySQLdump，在数据量达到几十G 的时候就相对痛苦了
文件格式	数据和索引是分别存储的，数据.MYD，索引.MYI	数据和索引是集中存储的，.ibd
记录存储顺序	按记录插入顺序保存	按主键大小有序插入
外键	不支持	支持
事务	不支持	支持
锁支持（锁是避免资源争用的一个机制，MySQL锁对用户几乎是透明的）	表级锁定	行级锁定、表级锁定，锁定力度小并发能力高
SELECT	MyISAM 更优	--
INSERT、UPDATE、DELETE	--	InnoDB 更优
select count(*)	myisam 更快，因为 myisam 内部维护了一个计数器，可以直接调取。	
索引的实现方式	B+树索引，myisam 是堆表	B+树索引，Innodb 是索引组织表
哈希索引	不支持	支持
全文索引	支持	不支持

MyISAM 索引与 InnoDB 索引的区别?

1、 InnoDB 索引是聚簇索引，MyISAM 索引是非聚簇索引。

2、 InnoDB 的主键索引的叶子节点存储着行数据，因此主键索引非常高效。

3、 MyISAM 索引的叶子节点存储的是行数据地址，需要再寻址一次才能得到数据。

4、 InnoDB 非主键索引的叶子节点存储的是主键和其他带索引的列数据，因此查询时做到覆盖索引会非常高效。

InnoDB 引擎的 4 大特性

1、 插入缓冲（insert buffer)

2、 二次写(double write)

3、 自适应哈希索引(ahi)

4、 预读(read ahead)

存储引擎选择

1、 如果没有特别的需求，使用默认的 Innodb 即可。

2、 MyISAM：以读写插入为主的应用程序，比如博客系统、新闻门户网站。

3、 Innodb：更新（删除）操作频率也高，或者要保证数据的完整性；并发量高，支持事务和外键。比如 OA 自动化办公系统。

什么是索引?

1、 索引是一种特殊的文件(InnoDB 数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

2、 索引是一种数据结构。数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B 树及其变种 B+树。

3、 更通俗的说，索引就相当于目录。为了方便查找书中的内容，通过对内容建立索引形成目录。索引是一个文件，它是要占据物理空间的。

索引有哪些优缺点?

索引的优点

1、 可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

2、 通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

索引的缺点

1、 时间方面：创建索引和维护索引要耗费时间，具体地，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，会降低增/改/删的执行效率；

2、 空间方面：索引需要占物理空间。

怎么创建索引的，有什么好处，有哪些分类

1、 创建索引的语法：create index depe_unique_ide on depe(dept_no) tablespace idx_

2、 创建索引可以增加查询速度，唯一索引可以保证数据库列的一致性，可以确定表与表之间的连接

3、 索引的分类： 逻辑分类：单列索引，复合索引，唯一索引，非唯一索引，函数索引 物理分类：B 数索引，反向键索引，位图索引

简述有哪些索引和作用

索引的作用：通过索引可以大大的提高数据库的检索速度，改善数据库性能

1、 唯一索引：不允许有俩行具有相同的值

2、 主键索引：为了保持数据库表与表之间的关系

3、 聚集索引：表中行的物理顺序与键值的逻辑（索引）顺序相同。

4、 非聚集索引：聚集索引和非聚集索引的根本区别是表记录的排列顺序和与索引的排列顺序是否一致

5、 复合索引：在创建索引时，并不是只能对一列进行创建索引，可以与主键一样，讲多个组合为索引

6、 全文索引： 全文索引为在字符串数据中进行复杂的词搜索提供有效支持

索引使用场景

1、 当数据多且字段值有相同的值得时候用普通索引。

2、 当字段多且字段值没有重复的时候用唯一索引。

3、 当有多个字段名都经常被查询的话用复合索引。

4、 普通索引不支持空值，唯一索引支持空值。

5、 但是，若是这张表增删改多而查询较少的话，就不要创建索引了，因为如果你给一列创建了索引，那么对该列进行增删改的时候，都会先访问这一列的索引，

6、 若是增，则在这一列的索引内以新填入的这个字段名的值为名创建索引的子集，

7、 若是改，则会把原来的删掉，再添入一个以这个字段名的新值为名创建索引的子集，

8、 若是删，则会把索引中以这个字段为名的索引的子集删掉。

9、 所以，会对增删改的执行减缓速度，

10、 所以，若是这张表增删改多而查询较少的话，就不要创建索引了。

11、 更新太频繁地字段不适合创建索引。

12、 不会出现在 where 条件中的字段不该建立索引。

主键索引与唯一索引的区别

1、 主键是一种约束，唯一索引是一种索引，两者在本质上是不同的。

2、 主键创建后一定包含一个唯一性索引，唯一性索引并不一定就是主键。

3、 唯一性索引列允许空值，而主键列不允许为空值。

4、 主键列在创建时，已经默认为空值 ++ 唯一索引了。

5、 一个表最多只能创建一个主键，但可以创建多个唯一索引。

6、 主键更适合那些不容易更改的唯一标识，如自动递增列、身份证号等。

7、 主键可以被其他表引用为外键，而唯一索引不能。 ？

索引有哪几种类型?

主键索引: 数据列不允许重复，不允许为 NULL，一个表只能有一个主键。

唯一索引: 数据列不允许重复，允许为 NULL 值，一个表允许多个列创建唯一索引。

1、 可以通过 ALTER TABLE table_name ADD UNIQUE (column); 创建唯一索引

2、 可以通过 ALTER TABLE table_name ADD UNIQUE (column1,column2); 创建唯一组合索引

普通索引: 基本的索引类型，没有唯一性的限制，允许为 NULL 值。

1、 可以通过 ALTER TABLE table_name ADD INDEX index_name (column);创建普通索引

2、 可以通过 ALTER TABLE table_name ADD INDEX index_name(column1, column2, column3);创建组合索引

全文索引： 是目前搜索引擎使用的一种关键技术。

可以通过 ALTER TABLE table_name ADD FULLTEXT (column);创建全文索引

索引的数据结构（b 树，hash）

索引的数据结构和具体存储引擎的实现有关，在 MySQL 中使用较多的索引有 **Hash 索引，B+树索引**等，而我们经常使用的 InnoDB 存储引擎的默认索引实现为：B+树索引。对于哈希索引来说，底层的数据结构就是哈希表，因此在绝大多数需求为单条记录查询的时候，可以选择哈希索引，查询性能最快；其余大部分场景，建议选择 BTree 索引。

B 树索引

MySQL 通过存储引擎取数据，基本上 90%的人用的就是 InnoDB 了，按照实现方式分，InnoDB 的索引类型目前只有两种：BTREE（B 树）索引和 HASH 索引。B 树索引是 MySQL 数据库中使用最频繁的索引类型，基本所有存储引擎都支持 BTree 索引。通常我们说的索引不出意外指的就是（B 树）索引（实际是用 B+树实现的，因为在查看表索引时，MySQL 一律打印 BTREE，所以简称为 B 树索引）

![99_1.png][99_1.png]

查询方式：

1、 主键索引区:PI(关联保存的时数据的地址)按主键查询，

2、 普通索引区:si(关联的 id 的地址,然后再到达上面的地址)。所以按主键查询,速度最快

B+tree 性质：

1、 n 棵子 tree 的节点包含 n 个关键字，不用来保存数据而是保存数据的索引。

2、 所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。

3、 所有的非终端结点可以看成是索引部分，结点中仅含其子树中的最大（或最小）关键字。

4、 B+ 树中，数据对象的插入和删除仅在叶节点上进行。

5、 B+树有 2 个头指针，一个是树的根节点，一个是最小关键码的叶节点。

哈希索引

简要说下,类似于数据结构中简单实现的 HASH 表(散列表)一样,当我们在 MySQL 中用哈希索引时，主要就是通过 Hash 算法（常见的 Hash 算法有直接定址法、平方取中法、折叠法、除数取余法、随机数法），将数据库字段数据转换成定长的 Hash 值，与这条数据的行指针一并存入 Hash 表的对应位置；如果发生 Hash 碰撞（两个不同关键字的 Hash 值相同），则在对应 Hash 键下以链表形式存储。当然这只是简略模拟图。

![99_2.png][99_2.png]

索引的基本原理

索引用来快速地寻找那些具有特定值的记录。如果没有索引，一般来说执行查询时遍历整张表。

索引的原理很简单，就是把无序的数据变成有序的查询

1、 把创建了索引的列的内容进行排序

2、 对排序结果生成倒排表

3、 在倒排表内容上拼上数据地址链

4、 在查询的时候，先拿到倒排表内容，再取出数据地址链，从而拿到具体数据

索引算法有哪些?

索引算法有 BTree 算法和 Hash 算法

BTree 算法

BTree 是最常用的 MySQL 数据库索引算法，也是 MySQL 默认的算法。因为它不仅可以被用在=,>,>=,<,<=和 between 这些比较操作符上，而且还可以用于 like 操作符，只要它的查询条件是一个不以通配符开头的常量， 例如：

```
-- 只要它的查询条件是一个不以通配符开头的常量
select * from user where name like 'jack%';
-- 如果一通配符开头，或者没有使用常量，则不会使用索引，例如：
select * from user where name like '%jack';
```

Hash 算法

Hash Hash 索引只能用于对等比较，例如=,<=>（相当于=）操作符。由于是一次定位数据，不像 BTree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次 IO 访问，所以检索效率远高于 BTree 索引。

索引设计的原则?

1、 适合索引的列是出现在 where 子句中的列，或者连接子句中指定的列

2、 基数较小的类，索引效果较差，没有必要在此列建立索引

3、 使用短索引，如果对长字符串列进行索引，应该指定一个前缀长度，这样能够节省大量索引空间

4、 不要过度索引。索引需要额外的磁盘空间，并降低写操作的性能。在修改表内容的时候，索引会进行更新甚至重构，索引列越多，这个时间就会越长。所以只保持需要的索引有利于查询即可。

创建索引的原则

索引虽好，但也不是无限制的使用，最好符合一下几个原则

1、 最左前缀匹配原则，组合索引非常重要的原则，MySQL 会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如 a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d 是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到， a,b,d 的顺序可以任意调整。

2、 较频繁作为查询条件的字段才去创建索引

3、 更新频繁字段不适合创建索引

4、 若是不能有效区分数据的列不适合做索引列(如性别，男女未知，最多也就三种，区分度实在太低)

5、 尽量的扩展索引，不要新建索引。比如表中已经有 a 的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可。

6、 定义有外键的数据列一定要建立索引。

7、 对于那些查询中很少涉及的列，重复值比较多的列不要建立索引。

8、 对于定义为 text、image 和 bit 的数据类型的列不要建立索引。

创建索引的三种方式

第一种方式： 在执行 CREATE TABLE 时创建索引

```
CREATE TABLE user\_index2 ( id INT auto\_increment PRIMARY KEY, first\_name VARCHAR (16), last\_name VARCHAR (16), id\_card VARCHAR (18), information text, KEY name (first\_name, last\_name), FULLTEXT KEY (information), UNIQUE KEY (id\_card) );
```

第二种方式： 使用 ALTER TABLE 命令去增加索引

```
ALTER TABLE table_name ADD INDEX index_name (column_list);
```

1、 ALTER TABLE 用来创建普通索引、UNIQUE 索引或 PRIMARY KEY 索引。

2、 其中 table_name 是要增加索引的表名，column_list 指出对哪些列进行索引，多列时各列之间用逗号分隔。

3、 索引名 index_name 可自己命名，缺省时，MySQL 将根据第一个索引列赋一个名称。另外，ALTER TABLE 允许在单个语句中更改多个表，因此可以在同时创建多个索引。

4、 第三种方式：使用 CREATE INDEX 命令创建

```
CREATE INDEX index_name ON table_name (column_list);
```

CREATE INDEX 可对表增加普通索引或 UNIQUE 索引。(但是，不能创建 PRIMARY KEY 索引)

如何删除索引

根据索引名删除普通索引、唯一索引、全文索引：alter table 表名 drop KEY 索引名

```
alter table user_index drop KEY name;
alter table user_index drop KEY id_card;
alter table user_index drop KEY information;
```

删除主键索引：alter table 表名 drop primary key（因为主键只有一个）。这里值得注意的是，如果主键自增长，那么不能直接执行此操作（自增长依赖于主键索引）：

![99_3.png][99_3.png]

需要取消自增长再行删除：

```
alter table user_index
```

```
-- 重新定义字段
MODIFY id int,
drop PRIMARY KEY
```

但通常不会删除主键，因为设计主键一定与业务逻辑无关。

创建索引时需要注意什么？

1、 非空字段：应该指定列为 NOT NULL，除非你想存储 NULL。在 MySQL 中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。你应该用 0、一个特殊的值或者一个空串代替空值；

2、 取值离散大的字段：（变量各个取值之间的差异程度）的列放到联合索引的前面，可以通过 count()函数查看字段的差异值，返回值越大说明字段的唯一值越多字段的离散程度高；

3、 索引字段越小越好：数据库的数据存储以页为单位一页存储的数据越多一次 IO 操作获取的数据越大效率越高。

使用索引查询一定能提高查询的性能吗？为什么

通常，通过索引查询数据比全表扫描要快。但是我们也必须注意到它的代价。

索引需要空间来存储，也需要定期维护， 每当有记录在表中增减或索引列被修改时，索引本身也会被修改。 这意味着每条记录的 INSERT，DELETE，UPDATE 将为此多付出 4，5 次的磁盘 I/O。 因为索引需要额外的存储空间和处理，那些不必要的索引反而会使查询反应时间变慢。使用索引查询不一定能提高查询性能，索引范围查询(INDEX RANGE SCAN)适用于两种情况:

1、 基于一个范围的检索，一般查询返回结果集小于表中记录数的 30%

2、 基于非唯一性索引的检索

百万级别或以上的数据如何删除

关于索引：由于索引需要额外的维护成本，因为索引文件是单独存在的文件,所以当我们对数据的增加,修改,删除,都会产生额外的对索引文件的操作,这些操作需要消耗额外的 IO,会降低增/改/删的执行效率。所以，在我们删除数据库百万级别数据的时候，查询 MySQL 官方手册得知删除数据的速度和创建的索引数量是成正比的。

1、 所以我们想要删除百万数据的时候可以先删除索引（此时大概耗时三分多钟）

2、 然后删除其中无用数据（此过程需要不到两分钟）

3、 删除完成后重新创建索引(此时数据较少了)创建索引也非常快，约十分钟左右。

4、 与之前的直接删除绝对是要快速很多，更别说万一删除中断,一切删除会回滚。那更是坑了。

前缀索引

1、 语法：index(field(10))，使用字段值的前 10 个字符建立索引，默认是使用字段的全部内容建立索引。

2、 前提：前缀的标识度高。比如密码就适合建立前缀索引，因为密码几乎各不相同。

3、 实操的难度：在于前缀截取的长度。

4、 我们可以利用 select count(*)/count(distinct left(password,prefixLen));，通过从调整 prefixLen 的值（从 1 自增）查看不同前缀长度的一个平均匹配度，接近 1 时就可以了（表示一个密码的前 prefixLen 个字符几乎能确定唯一一条记录）

什么是最左前缀原则？什么是最左匹配原则

1、 顾名思义，就是最左优先，在创建多列索引时，要根据业务需求，where 子句中使用最频繁的一列放在最左边。

2、 最左前缀匹配原则，非常重要的原则，MySQL 会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如 a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d 是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d 的顺序可以任意调整。

3、 =和 in 可以乱序，比如 a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，MySQL 的查询优化器会帮你优化成索引可以识别的形式

B 树和 B+树的区别

1、 在 B 树中，你可以将键和值存放在内部节点和叶子节点；但在 B+树中，内部节点都是键，没有值，叶子节点同时存放键和值。

2、 B+树的叶子节点有一条链相连，而 B 树的叶子节点各自独立。

![99_4.png][99_4.png]

使用 B 树的好处

B 树可以在内部节点同时存储键和值，因此，把频繁访问的数据放在靠近根节点的地方将会大大提高热点数据的查询效率。这种特性使得 B 树在特定数据重复多次查询的场景中更加高效。

使用 B+树的好处

由于 B+树的内部节点只存放键，不存放值，因此，一次读取，可以在内存页中获取更多的键，有利于更快地缩小查找范围。 B+树的叶节点由一条链相连，因此，当需要进行一次全数据遍历的时候，B+树只需要使用 O(logN)时间找到最小的一个节点，然后通过链进行 O(N)的顺序遍历即可。而 B 树则需要对树的每一层进行遍历，这会需要更多的内存置换次数，因此也就需要花费更多的时间

Hash 索引和 B+树所有有什么区别或者说优劣呢?

1、 首先要知道 Hash 索引和 B+树索引的底层实现原理：

2、 hash 索引底层就是 hash 表，进行查找时，调用一次 hash 函数就可以获取到相应的键值，之后进行回表查询获得实际数据。B+树底层实现是多路平衡查找树。对于每一次的查询都是从根节点出发，查找到叶子节点方可以获得所查键值，然后根据查询判断是否需要回表查询数据。

那么可以看出他们有以下的不同：

1、 hash 索引进行等值查询更快(一般情况下)，但是却无法进行范围查询。

2、 因为在 hash 索引中经过 hash 函数建立索引之后，索引的顺序与原顺序无法保持一致，不能支持范围查询。而 B+树的的所有节点皆遵循(左节点小于父节点，右节点大于父节点，多叉树也类似)，天然支持范围。

3、 hash 索引不支持使用索引进行排序，原理同上。

4、 hash 索引不支持模糊查询以及多列索引的最左前缀匹配。原理也是因为 hash 函数的不可预测。AAAA 和 AAAAB 的索引没有相关性。

5、 hash 索引任何时候都避免不了回表查询数据，而 B+树在符合某些条件(聚簇索引，覆盖索引等)的时候可以只通过索引完成查询。

6、 hash 索引虽然在等值查询上较快，但是不稳定。性能不可预测，当某个键值存在大量重复的时候，发生 hash 碰撞，此时效率可能极差。而 B+树的查询效率比较稳定，对于所有的查询都是从根节点到叶子节点，且树的高度较低。

7、 因此，在大多数情况下，直接选择 B+树索引可以获得稳定且较好的查询速度。而不需要使用 hash 索引。

数据库为什么使用 B+树而不是 B 树

1、 B 树只适合随机检索，而 B+树同时支持随机检索和顺序检索；

2、 B+树空间利用率更高，可减少 I/O 次数，磁盘读写代价更低。一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘 I/O 消耗。B+树的内部结点并没有指向关键字具体信息的指针，只是作为索引使用，其内部结点比 B 树小，盘块能容纳的结点中关键字数量更多，一次性读入内存中可以查找的关键字也就越多，相对的，IO 读写次数也就降低了。而 IO 读写次数是影响索引检索效率的最大因素；

3、 B+树的查询效率更加稳定。B 树搜索有可能会在非叶子结点结束，越靠近根节点的记录查找时间越短，只要找到关键字即可确定记录的存在，其性能等价于在关键字全集内做一次二分查找。而在 B+树中，顺序检索比较明显，随机检索时，任何关键字的查找都必须走一条从根节点到叶节点的路，所有关键字的查找路径长度相同，导致每一个关键字的查询效率相当。

4、 B-树在提高了磁盘 IO 性能的同时并没有解决元素遍历的效率低下的问题。B+树的叶子节点使用指针顺序连接在一起，只要遍历叶子节点就可以实现整棵树的遍历。而且在数据库中基于范围的查询是非常频繁的，而 B 树不支持这样的操作。

5、 增删文件（节点）时，效率更高。因为 B+树的叶子节点包含所有关键字，并以有序的链表结构存储，这样可很好提高增删效率。

B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据，

在 B+树的索引中，叶子节点可能存储了当前的 key 值，也可能存储了当前的 key 值以及整行的数据，这就是聚簇索引和非聚簇索引。 在 InnoDB 中，只有主键索引是聚簇索引，如果没有主键，则挑选一个唯一键建立聚簇索引。如果没有唯一键，则隐式的生成一个键来建立聚簇索引。

当查询使用聚簇索引时，在对应的叶子节点，可以获取到整行数据，因此不用再次进行回表查询。

什么是聚簇索引？何时使用聚簇索引与非聚簇索引

聚簇索引：

将数据存储与索引放到了一块，找到索引也就找到了数据

非聚簇索引：

将数据存储于索引分开结构，索引结构的叶子节点指向了数据的对应行，myisam 通过 key_buffer 把索引先缓存到内存中，当需要访问数据时（通过索引访问数据），在内存中直接搜索索引，然后通过索引找到磁盘相应数据，这也就是为什么索引不在 key buffer 命中时，速度慢的原因

澄清一个概念：

innodb 中，在聚簇索引之上创建的索引称之为辅助索引，辅助索引访问数据总是需要二次查找，非聚簇索引都是辅助索引，像复合索引、前缀索引、唯一索引，辅助索引叶子节点存储的不再是行的物理位置，而是主键值

何时使用聚簇索引与非聚簇索引

![99_5.png][99_5.png]

非聚簇索引一定会回表查询吗?

不一定，这涉及到查询语句所要求的字段是否全部命中了索引，如果全部命中了索引，那么就不必再进行回表查询。

举个简单的例子

假设我们在员工表的年龄上建立了索引，那么当进行 select age from employee where age < 20 的查询时，在索引的叶子节点上，已经包含了 age 信息，不会再次进行回表查询。

联合索引是什么？为什么需要注意联合索引中的顺序?

MySQL 可以使用多个字段同时建立一个索引，叫做联合索引。在联合索引中，如果想要命中索引，需要按照建立索引时的字段顺序挨个使用，否则无法命中索引。

具体原因为：

MySQL 使用索引时需要索引有序，假设现在建立了"name，age，school"的联合索引，那么索引的排序为：先按照 name 排序，如果 name 相同，则按照 age 排序，如果 age 的值也相等，则按照 school 进行排序。

当进行查询时，此时索引仅仅按照 name 严格有序，因此必须首先使用 name 字段进行等值查询，之后对于匹配到的列而言，其按照 age 字段严格有序，此时可以使用 age 字段用做索引查找，以此类推。因此在建立联合索引的时候应该注意索引列的顺序，一般情况下，将查询需求频繁或者字段选择性高的列放在前面。此外可以根据特例的查询或者表结构进行单独的调整。

什么是数据库事务?

事务是一个不可分割的数据库操作序列，也是数据库并发控制的基本单位，其执行的结果必须使数据库从一种一致性状态变到另一种一致性状态。事务是逻辑上的一组操作，要么都执行，要么都不执行。

事务最经典也经常被拿出来说例子就是转账了

假如小明要给小红转账 1000 元，这个转账会涉及到两个关键操作就是：将小明的余额减少 1000 元，将小红的余额增加 1000 元。万一在这两个操作之间突然出现错误比如银行系统崩溃，导致小明余额减少而小红的余额没有增加，这样就不对了。事务就是保证这两个关键操作要么都成功，要么都要失败。

事物的四大特性(ACID)介绍一下?

关系性数据库需要遵循 ACID 规则，具体内容如下：

![99_6.png][99_6.png]

1、 原子性：

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、 一致性：

执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；

3、 隔离性：

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库是独立的；

4、 持久性：

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。

什么是脏读？幻读？不可重复读?

1、 脏读(Drity Read)：

某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个 RollBack 了操作，则后一个事务所读取的数据就会是不正确的。

2、 不可重复读(Non-repeatable read):

在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

3、 幻读(Phantom Read):

在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

什么是事务的隔离级别？MySQL 的默认隔离级别是什么？

为了达到事务的四大特性，数据库定义了 4 种不同的事务隔离级别，由低到高依次为 Read uncommitted、Read committed、Repeatable read、Serializable，这四个级别可以逐个解决脏读、不可重复读、幻读这几类问题。

隔离级别	脏读	不可重复读	幻影读
READ-UNCOMMITTED	√	√	√
READ-COMMITTED	×	√	√
REPEATABLE-READ	×	×	√
SERIALIZABLE			

SQL 标准定义了四个隔离级别：

1、 READ-UNCOMMITTED(读取未提交):

最低的隔离级别，允许读取尚未提交的数据变更，**可能会导致脏读、幻读或不可重复读。**

2、 READ-COMMITTED(读取已提交)： 允许读取并发事务已经提交的数据，**可以阻止脏读，但是幻读或不可重复读仍有可能发生。**

3、 REPEATABLE-READ(可重复读):

对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，**可以阻止脏读和不可重复读，但幻读仍有可能发生。**

4、 SERIALIZABLE(可串行化):

最高的隔离级别，完全服从 ACID 的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，**该级别可以防止脏读、不可重复读以及幻读。**

注意：

1、 这里需要注意的是：MySQL 默认采用的 REPEATABLE_READ 隔离级别 Oracle 默认采用的 READ_COMMITTED 隔离级别

2、 事务隔离机制的实现基于锁机制和并发调度。其中并发调度使用的是 MVVC（多版本并发控制），通过保存修改的旧版本信息来支持并发一致性读和回滚等特性。

3、 因为隔离级别越低，事务请求的锁越少，所以大部分数据库系统的隔离级别都是 **READ-COMMITTED(读取提交内容):**，但是你要知道的是 InnoDB 存储引擎默认使用 **REPEATABLE-READ（可重读）**并不会有任何性能损失。

4、 InnoDB 存储引擎在 **分布式事务** 的情况下一般会用到**SERIALIZABLE(可串行化)**隔离级别。

对 MySQL 的锁了解吗

当数据库有并发事务的时候，可能会产生数据的不一致，这时候需要一些机制来保证访问的次序，锁机制就是这样的一个机制。

就像酒店的房间，如果大家随意进出，就会出现多人抢夺同一个房间的情况，而在房间上装上锁，申请到钥匙的人才可以入住并且将房间锁起来，其他人只有等他使用完毕才可以再次使用。

从锁的类别上分 MySQL 都有哪些锁呢?

从锁的类别上来讲，有共享锁和排他锁。

共享锁：

又叫做读锁。 当用户要进行数据的读取时，对数据加上共享锁。共享锁就是让多个线程同时获取一个锁。

排他锁：

又叫做写锁。 当用户要进行数据的写入时，对数据加上排他锁。排它锁也称作独占锁，一个锁在某一时刻只能被一个线程占有，其它线程必须等待锁被释放之后才可能获取到锁。排他锁只可以加一个，他和其他的排他锁，共享锁都相斥。

隔离级别与锁的关系

1、 在 Read Uncommitted 级别下，读取数据不需要加共享锁，这样就不会跟被修改的数据上的排他锁冲突

2、 在 Read Committed 级别下，读操作需要加共享锁，但是在语句执行完以后释放共享锁；

3、 在 Repeatable Read 级别下，读操作需要加共享锁，但是在事务提交之前并不释放共享锁，也就是必须等待事务执行完毕以后才释放共享锁。

4、 SERIALIZABLE 是限制性最强的隔离级别，因为该级别**锁定整个范围的键**，并一直持有锁，直到事务完成。

按照锁的粒度分数据库锁有哪些？锁机制与 InnoDB 锁算法

在关系型数据库中，可以**按照锁的粒度把数据库锁分**为行级锁(INNODB 引擎)、表级锁(MYISAM 引擎)和页级锁(BDB 引擎)。

MyISAM 和 InnoDB 存储引擎使用的锁：

1、 MyISAM 采用表级锁(table-level locking)。

2、 InnoDB 支持行级锁(row-level locking)和表级锁，默认为行级锁

行级锁，表级锁和页级锁对比

行级锁

行级锁是 MySQL 中锁定粒度最细的一种锁，表示只针对当前操作的行进行加锁。行级锁能大大减少数据库操作的冲突。其加锁粒度最小，但加锁的开销也最大。行级锁分为共享锁 和 排他锁。

特点：

开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

表级锁

表级锁是 MySQL 中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单，资源消耗较少，被大部分 MySQL 引擎支持。最常使用的 MYISAM 与 INNODB 都支持表级锁定。表级锁定分为表共享读锁（共享锁）与表独占写锁（排他锁）

特点：

开销小，加锁快；不会出现死锁；锁定粒度大，发出锁冲突的概率最高，并发度最低。

页级锁

页级锁是 MySQL 中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折衷的页级，一次锁定相邻的一组记录。

特点：

开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般

MySQL 中 InnoDB 引擎的行锁是怎么实现的?

InnoDB 是基于索引来完成行锁

例:

select * from tab_with_index where id = 1 for update;

for update 可以根据条件来完成行锁锁定，并且 id 是有索引键的列，如果 id 不是索引键那么 InnoDB 将完成表锁，并发将无从谈起

InnoDB 存储引擎的锁的算法有三种

1、 Record lock：单个行记录上的锁

2、 Gap lock：间隙锁，锁定一个范围，不包括记录本身

3、 Next-key lock：record+gap 锁定一个范围，包含记录本身

相关知识点：

1、 innodb 对于行的查询使用 next-key lock

2、 Next-locking keying 为了解决 Phantom Problem 幻读问题

3、 当查询的索引含有唯一属性时，将 next-key lock 降级为 record key

4、 Gap 锁设计的目的是为了阻止多个事务将记录插入到同一范围内，而这会导致幻读问题的产生

5、 有两种方式显式关闭 gap 锁：（除了外键约束和唯一性检查外，其余情况仅使用 record lock） A、将事务隔离级别设置为 RC B、将参数 innodb_locks_unsafe_for_binlog 设置为 1

什么是死锁？怎么解决？

死锁是指两个或多个事务在同一资源上相互占用，并请求锁定对方的资源，从而导致恶性循环的现象。

常见的解决死锁的方法

1、 如果不同程序会并发存取多个表，尽量约定以相同的顺序访问表，可以大大降低死锁机会。

2、 在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁产生概率；

3、 对于非常容易产生死锁的业务部分，可以尝试使用升级锁定颗粒度，通过表级锁定来减少死锁产生的概率；

如果业务不好处理,可以用分布式事务锁或者使用乐观锁

数据库的乐观锁和悲观锁是什么？怎么实现的？

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。乐观并发控制（乐观锁）和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

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

```
//核心 SQL,主要靠 for update
select status from t_goods where id=1 for update;
```

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：乐一般会使用版本号机制或 CAS 算法实现。

```
//核心 SQL
update table set x=x+1, version=version+1 where id=#{id} and
version=#{version};
```

两种锁的使用场景

从上面对两种锁的介绍，我们知道两种锁各有优缺点，不可认为一种好于另一种，像**乐观锁适用于写比较少的情况下（多读场景）**，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。

但如果是多写的情况，一般会经常产生冲突，这就会导致上层应用会不断的进行 retry，这样反倒是降低了性能，所以一般多写的场景下用悲观锁就比较合适。

为什么要使用视图？什么是视图？

为了提高复杂 SQL 语句的复用性和表操作的安全性，MySQL 数据库管理系统提供了视图特性。所谓视图，本质上是一种虚拟表，在物理上是不存在的，其内容与真实的表相似，包含一系列带有名称的列和行数据。但是，视图并不在数据库中以储存的数据值形式存在。行和列数据来自定义视图的查询所引用基本表，并且在具体引用视图时动态生成。

视图使开发者只关心感兴趣的某些特定数据和所负责的特定任务，只能看到视图中所定义的数据，而不是视图所引用表中的数据，从而提高了数据库中数据的安全性。

视图有哪些特点？

视图的特点如下：

1、 视图的列可以来自不同的表，是表的抽象和在逻辑意义上建立的新关系。

2、 视图是由基本表(实表)产生的表(虚表)。

3、 视图的建立和删除不影响基本表。

4、 对视图内容的更新(添加，删除和修改)直接影响基本表。

5、 当视图来自多个基本表时，不允许添加和删除数据。

视图的操作包括创建视图，查看视图，删除视图和修改视图。

视图的使用场景有哪些?

视图根本用途：简化 sql 查询，提高开发效率。如果说还有另外一个用途那就是兼容老的表结构

下面是视图的常见使用场景：

1、 重用 SQL 语句；

2、 简化复杂的 SQL 操作。在编写查询后，可以方便的重用它而不必知道它的基本查询细节；

3、 使用表的组成部分而不是整个表；

4、 保护数据。可以给用户授予表的特定部分的访问权限而不是整个表的访问权限；

5、 更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据。

视图的优点

1、 查询简单化。视图能简化用户的操作

2、 数据安全性。视图使用户能以多种角度看待同一数据，能够对机密数据提供安全保护

3、 逻辑数据独立性。视图对重构数据库提供了一定程度的逻辑独立性

视图的缺点

1、 性能。数据库必须把视图的查询转化成对基本表的查询，如果这个视图是由一个复杂的多表查询所定义，那么，即使是视图的一个简单查询，数据库也把它变成一个复杂的结合体，需要花费一定的时间。

2、 修改限制。当用户试图修改视图的某些行时，数据库必须把它转化为对基本表的某些行的修改。事实上，当从视图中插入或者删除时，情况也是这样。对于简单视图来说，这是很方便的，但是，对于比较复杂的视图，可能是不可修改的

这些视图有如下特征：

1、 有 UNIQUE 等集合操作符的视图。

2、 有 GROUP BY 子句的视图。

3、 有诸如 AVG\SUM\MAX 等聚合函数的视图。

4、 使用 DISTINCT 关键字的视图。

5、 连接表的视图（其中有些例外）

什么是游标?

游标是系统为用户开设的一个数据缓冲区，存放 SQL 语句的执行结果，每个游标区都有一个名字。用户可以通过游标逐一获取记录并赋给主变量，交由主语言进一步处理。

什么是存储过程？有哪些优缺点？

存储过程是一个预编译的 SQL 语句，优点是允许模块化的设计，就是说只需要创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次 SQL，便用存储过程比单纯 SQL 语句执行要快。

优点

1、 存储过程是预编译过的，执行效率高。

2、 存储过程的代码直接存放于数据库中，通过存储过程名直接调用，减少网络通讯。

3、 安全性高，执行存储过程需要有一定权限的用户。

4、 存储过程可以重复使用，减少数据库开发人员的工作量。

缺点

1、 调试麻烦，但是用 PL/SQL Developer 调试很方便！弥补这个缺点。

2、 移植问题，数据库端代码当然是与数据库相关的。但是如果是做工程型项目，基本不存在移植问题。

3、 重新编译问题，因为后端代码是运行前编译的，如果带有引用关系的对象发生改变时，受影响的存储过程、包将需要重新编译（不过也可以设置成运行时刻自动编译）。

4、 如果在一个程序系统中大量的使用存储过程，到程序交付使用的时候随着用户需求的增加会导致数据结构的变化，接着就是系统的相关问题了，最后如果用户想维护该系统可以说是很难很难、而且代价是空前的，维护起来更麻烦。

什么是触发器？触发器的使用场景有哪些？

触发器是用户定义在关系表上的一类由事件驱动的特殊的存储过程。触发器是指一段代码，当触发某个事件时，自动执行这些代码。

使用场景

1、 可以通过数据库中的相关表实现级联更改。

2、 实时监控某张表中的某个字段的更改而需要做出相应的处理。

3、 例如可以生成某些业务的编号。

4、 注意不要滥用，否则会造成数据库及应用程序的维护困难。

5、 大家需要牢记以上基础知识点，重点是理解数据类型 CHAR 和 VARCHAR 的差异，表存储引擎 InnoDB 和 MyISAM 的区别。

MySQL 中都有哪些触发器？

在 MySQL 数据库中有如下六种触发器：

1、 Before Insert

2、 After Insert

3、 Before Update

4、 After Update

5、 Before Delete

6、 After Delete

SQL 语句主要分为哪几类

1、 数据定义语言 DDL（Data Ddefinition Language）CREATE，DROP，ALTER

主要为以上操作 即对逻辑结构等有操作的，其中包括表结构，视图和索引。

2、 数据查询语言 DQL（Data Query Language）SELECT

这个较为好理解 即查询操作，以 select 关键字。各种简单查询，连接查询等 都属于 DQL。

3、 数据操纵语言 DML（Data Manipulation Language）INSERT，UPDATE，DELETE

主要为以上操作 即对数据进行操作的，对应上面所说的查询操作 DQL 与 DML 共同构建了多数初级程序员常用的增删改查操作。而查询是较为特殊的一种 被划分到 DQL 中。

4、 数据控制功能 DCL（Data Control Language）GRANT，REVOKE，COMMIT，ROLLBACK

主要为以上操作 即对数据库安全性完整性等有操作的，可以简单的理解为权限控制等。

SQL 语句的语法顺序：

1、 SELECT

2、 FROM

3、 JOIN

4、ON

5、 WHERE

6、 GROUP BY

7、 HAVING

8、 UNION

9、 ORDER BY

10、 LIMIT

超键、候选键、主键、外键分别是什么？

1、 超键：

在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。

2、 候选键：

是最小超键，即没有冗余元素的超键。

3、 主键：

数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。

4、 外键：

在一个表中存在的另一个表的主键称此表的外键。

SQL 约束有哪几种?

SQL 约束有哪几种？

1、 NOT NULL：用于控制字段的内容一定不能为空（NULL）。

2、 UNIQUE：控件字段内容不能重复，一个表允许有多个 Unique 约束。

3、 PRIMARY KEY：也是用于控件字段内容不能重复，但它在一个表只允许出现一个。

4、 FOREIGN KEY：用于预防破坏表之间连接的动作，也能防止非法数据插入外键列，因为它必须是它指向的那个表中的值之一。

5、 CHECK：用于控制字段的值范围。

六种关联查询

1、 交叉连接（CROSS JOIN）

2、 内连接（INNER JOIN）

3、 外连接（LEFT JOIN/RIGHT JOIN）

4、 联合查询（UNION 与 UNION ALL）

5、 全连接（FULL JOIN）

6、 交叉连接（CROSS JOIN）

SELECT * FROM A,B(,C)或者 SELECT * FROM A CROSS JOIN B (CROSS JOIN C)#没有任何关联条件，结果是笛卡尔积，结果集会很大，没有意义，很少使用内连接（INNER JOIN）SELECT * FROM A,B WHERE A.id=B.id 或者 SELECT * FROM A INNER JOIN B ON A.id=B.id 多表中同时符合某种条件的数据记录的集合，INNER JOIN 可以缩写为 JOIN

内连接分为三类

1、 等值连接：ON A.id=B.id

2、 不等值连接：ON A.id > B.id

3、 自连接：SELECT * FROM A T1 INNER JOIN A T2 ON T1.id=T2.pid

外连接（LEFT JOIN/RIGHT JOIN）

左外连接：

LEFT OUTER JOIN，以左表为主，先查询出左表，按照 ON 后的关联条件匹配右表，没有匹配到的用 NULL 填充，可以简写成 LEFT JOIN

右外连接：

RIGHT OUTER JOIN，以右表为主，先查询出右表，按照 ON 后的关联条件匹配左表，没有匹配到的用 NULL 填充，可以简写成 RIGHT JOIN

联合查询（UNION 与 UNION ALL）

```
SELECT * FROM A UNION SELECT * FROM B UNION ...
```

1、 就是把多个结果集集中在一起，UNION 前的结果为基准，需要注意的是联合查询的列数要相等，相同的记录行会合并

2、 如果使用 UNION ALL，不会合并重复的记录行

3、 效率 UNION 高于 UNION ALL

全连接（FULL JOIN）

```
SELECT * FROM A LEFT JOIN B ON A.id=B.id UNIONSELECT * FROM A RIGHT JOIN B ON A.id=B.id
```

MySQL 不支持全连接

可以使用 LEFT JOIN 和 UNION 和 RIGHT JOIN 联合使用

有 2 张表

1 张 R、1 张 S，R 表有 ABC 三列，S 表有 CD 两列，表中各有三条记录

R 表

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

S 表

C	D
c1	d1
c2	d2
c4	d3

交叉连接(笛卡尔积)

SQL

```
select r.*,s.* from r,s
```

结果

A	B	C	C	D
a1	b1	c1	c1	d1
a2	b2	c2	c1	d1

a3	b3	c3	c1	d1
a1	b1	c1	c2	d2
a2	b2	c2	c2	d2
a3	b3	c3	c2	d2
a1	b1	c1	c4	d3
a2	b2	c2	c4	d3
a3	b3	c3	c4	d3

内连接结果

SQL

```
select r.*,s.* from r inner join s on r.c=s.c
```

结果

A	B	C	C	D
a1	b1	c1	c1	d1
a2	b2	c2	c2	d2

左连接结果

SQL

```
select r.*,s.* from r left join s on r.c=s.c
```

结果

A	B	C	C	D

a1	b1	c1	c1	d1
a2	b2	c2	c2	d2
a3	b3	c3		

右连接结果

SQL

```
select r.*,s.* from r right join s on r.c=s.c
```

结果

A	B	C	C	D
a1	b1	c1	c1	d1
a2	b2	c2	c2	d2
			c4	d3

全表连接的结果（MySql 不支持，Oracle 支持）

SQL

```
select r.*,s.* from r full join s on r.c=s.c
```

结果

A	B	C	C	D
a1	b1	c1	c1	d1
a2	b2	c2	c2	d2
a3	b3	c3		

			c4	d3

什么是子查询

1、 条件：一条 SQL 语句的查询结果做为另一条查询语句的条件或查询结果

2、 嵌套：多条 SQL 语句嵌套使用，内部的 SQL 查询语句称为子查询。

MySQL 中 in 和 exists 区别

MySQL 中的 in 语句是把外表和内表作 hash 连接，而 exists 语句是对外表作 loop 循环，每次 loop 循环再对内表进行查询。一直大家都认为 exists 比 in 语句的效率要高，这种说法其实是不准确的。这个是要区分环境的。

1、 如果查询的两个表大小相当，那么用 in 和 exists 差别不大。

2、 如果两个表中一个较小，一个是大表，则子查询表大的用 exists，子查询表小的用 in。

3、 not in 和 not exists：如果查询语句使用了 not in，那么内外表都进行全表扫描，没有用到索引；而 not extsts 的子查询依然能用到表上的索引。所以无论那个表大，用 not exists 都比 not in 要快。

varchar 与 char 的区别

char 的特点

1、 char 表示定长字符串，长度是固定的；

2、 如果插入数据的长度小于 char 的固定长度时，则用空格填充；

3、 因为长度固定，所以存取速度要比 varchar 快很多，甚至能快 50%，但正因为其长度固定，所以会占据多余的空间，是空间换时间的做法；

4、 对于 char 来说，最多能存放的字符个数为 255，和编码无关

varchar 的特点

1、 varchar 表示可变长字符串，长度是可变的；

2、 插入的数据是多长，就按照多长来存储；

3、 varchar 在存取方面与 char 相反，它存取慢，因为长度不固定，但正因如此，不占据多余的空间，是时间换空间的做法；

4、 对于 varchar 来说，最多能存放的字符个数为 65532

总之

结合性能角度（char 更快）和节省磁盘空间角度（varchar 更小），具体情况还需具体来设计数据库才是妥当的做法

varchar(50)中 50 的涵义

最多存放 50 个字符，varchar(50)和(200)存储 hello 所占空间一样，但后者在排序时会消耗更多内存,因为 order by col 采用 fixed_length 计算 col 长度(memory 引擎也一样)。在早期 MySQL 版本中， 50 代表字节数，现在代表字符数。

int(20)中 20 的涵义

1、 是指显示字符的长度。20 表示最大显示宽度为 20，但仍占 4 字节存储，存储范围不变；

2、 不影响内部存储，只是影响带 zerofill 定义的 int 时，前面补多少个 0，易于报表展示

MySQL 为什么这么设计

对大多数应用没有意义，只是规定一些工具用来显示字符的个数；int(1)和 int(20)存储和计算均一样；