SQL语句分类
1)Data Definition Language (DDL数据定义语言) 如：建库，建表
2)Data Manipulation Language(DML数据操纵语言)，如：对表中的记录操作增删改
3)Data Query Language(DQL 数据查询语言)，如：对表中的查询操作
4)Data Control Language(DCL 数据控制语言)，如：对用户权限的设置

DataBase

创建数据库

- 创建数据库

```
CREATE DATABASE 数据库名;
```

- 判断数据库是否已经存在，不存在则创建数据库

```
CREATE DATABASE IF NOT EXISTS 数据库名;
```

- 创建数据库并指定字符集

```
CREATE DATABASE 数据库名 CHARACTER SET 字符集;
```

查看所有的数据库

show databases;

-- 查看某个数据库的定义信息
show create database db3;

将db3数据库的字符集改成utf8
alter database db3 character set utf8;

删除数据库的语法
DROP DATABASE 数据库名;

-- 查看正在使用的数据库
select database();

Table

创建表

7.1.1 创建表的格式

```
CREATE TABLE 表名 (
  字段名 1 字段类型 1,
  字段名 2 字段类型 2
);
```

查看某个数据库中的所有表
SHOW TABLES;

查看表结构
DESC 表名;

查看创建表的SQL语句

SHOW CREATE TABLE 表名;

快速创建一个表结构相同的表
CREATE TABLE 新表名 LIKE 旧表名;

判断表是否存在，如果存在则删除表
DROP TABLE IF EXISTS 表名;

添加表列ADD
ALTER TABLE 表名 ADD 列名 类型;

修改列类型MODIFY
ALTER TABLE 表名 MODIFY列名 新的类型;

修改列名 CHANGE
ALTER TABLE 表名 CHANGE 旧列名 新列名 类型;

删除列 DROP
ALTER TABLE 表名 DROP 列名;

修改表名
RENAME TABLE 表名 TO 新表名;

修改字符集character set
ALTER TABLE 表名 character set 字符集;

向表中插入数据

- 所有的字段名都写出来

```
INSERT INTO 表名 (字段名 1, 字段名 2, 字段名 3…) VALUES (值 1, 值 2, 值 3);
```

- 不写字段名

```
INSERT INTO 表名 VALUES (值 1, 值 2, 值 3…);
```

8.1.2 插入部分数据

```
INSERT INTO 表名 (字段名 1, 字段名 2, ...) VALUES (值 1, 值 2, ...);
```

将表名2中的所有的列复制到表名1中
INSERT INTO 表名1 SELECT * FROM 表名2;
只复制部分列
INSERT INTO 表名1(列1, 列2) SELECT 列1, 列2 FROM student;

更新表中数据

```
-- 不带条件修改数据，将所有的性别改成女
update student set sex = '女';

-- 带条件修改数据，将 id 号为 2 的学生性别改成男
update student set sex='男' where id=2;

-- 一次修改多个列，把 id 为 3 的学生，年龄改成 26 岁，address 改成北京
update student set age=26, address='北京' where id=3;
```

删除表记录

```
-- 带条件删除数据，删除 id 为 1 的记录
delete from student where id=1;

-- 不带条件删除数据,删除表中的所有数据
delete from student;
```

表的查询

查询指定列并且结果不出现重复数据
SELECT DISTINCT 字段名 FROM 表名;

某列数据和固定值运算
SELECT 列名1 + 固定值 FROM 表名;

某列数据和其他列数据参与运算
SELECT 列名1 + 列名2 FROM 表名;

- 运算符

比较运算符	说明
>、<、<=、>=、=、<>	<>在 SQL 中表示不等于，在 mysql 中也可以使用!= 没有==
BETWEEN...AND	在一个范围之内，如：between 100 and 200 相当于条件在 100 到 200 之间，包头又包尾
IN(集合)	集合表示多个值，使用逗号分隔
LIKE '张%'	模糊查询
IS NULL	查询某一列为 NULL 的值，注：不能写=NULL

- 逻辑运算符

逻辑运算符	说明
and 或 &&	与，SQL 中建议使用前者，后者并不通用。
or 或 ||	或
not 或 ！	非

- in 关键字

SELECT 字段名 FROM 表名 WHERE 字段 in (数据 1, 数据 2...);
in 里面的每个数据都会作为一次条件，只要满足条件的就会显示

- 范围查询

BETWEEN 值 1 AND 值 2
表示从值 1 到值 2 范围，包头又包尾
比如：age BETWEEN 80 AND 100 相当于： age>=80 && age<=100

查询 english 成绩大于等于 75，且小于等于 90 的学生

```
select * from student3 where english between 75 and 90;
```

- like 关键字

LIKE 表示模糊查询
SELECT * FROM 表名 WHERE 字段名 LIKE '通配符字符串';

- MySQL 通配符

通配符	说明
%	匹配任意多个字符串
_	匹配一个字符

表的约束

约束名	约束关键字
主键	primary key
唯一	unique
非空	not null
外键	foreign key
检查约束	check 注：mysql 不支持

- 创建主键方式：

1. 在创建表的时候给字段添加主键

```
字段名 字段类型 PRIMARY KEY
```

2. 在已有表中添加主键

```
ALTER TABLE 表名 ADD PRIMARY KEY(字段名);
```

```
-- 创建表学生表 st5, 包含字段(id, name, age)将 id 做为主键
create table st5 (
  id int primary key,  -- id 为主键
  name varchar(20),
  age int
)

desc st5;
```

自增

创建表时指定起始值
CREATE TABLE 表名(
列名 int primary key AUTO_INCREMENT
) AUTO_INCREMENT=起始值;

创建好以后修改起始值
ALTER TABLE 表名 AUTO_INCREMENT=起始值;
alter table st4 auto_increment = 2000;

- 新建表时增加外键：

```
[CONSTRAINT] [外键约束名称] FOREIGN KEY(外键字段名) REFERENCES 主表名(主键字段名)
```

- 已有表增加外键：

```
ALTER TABLE 从表 ADD [CONSTRAINT] [外键约束名称] FOREIGN KEY (外键字段名) REFERENCES 主表(主键字段名);
```

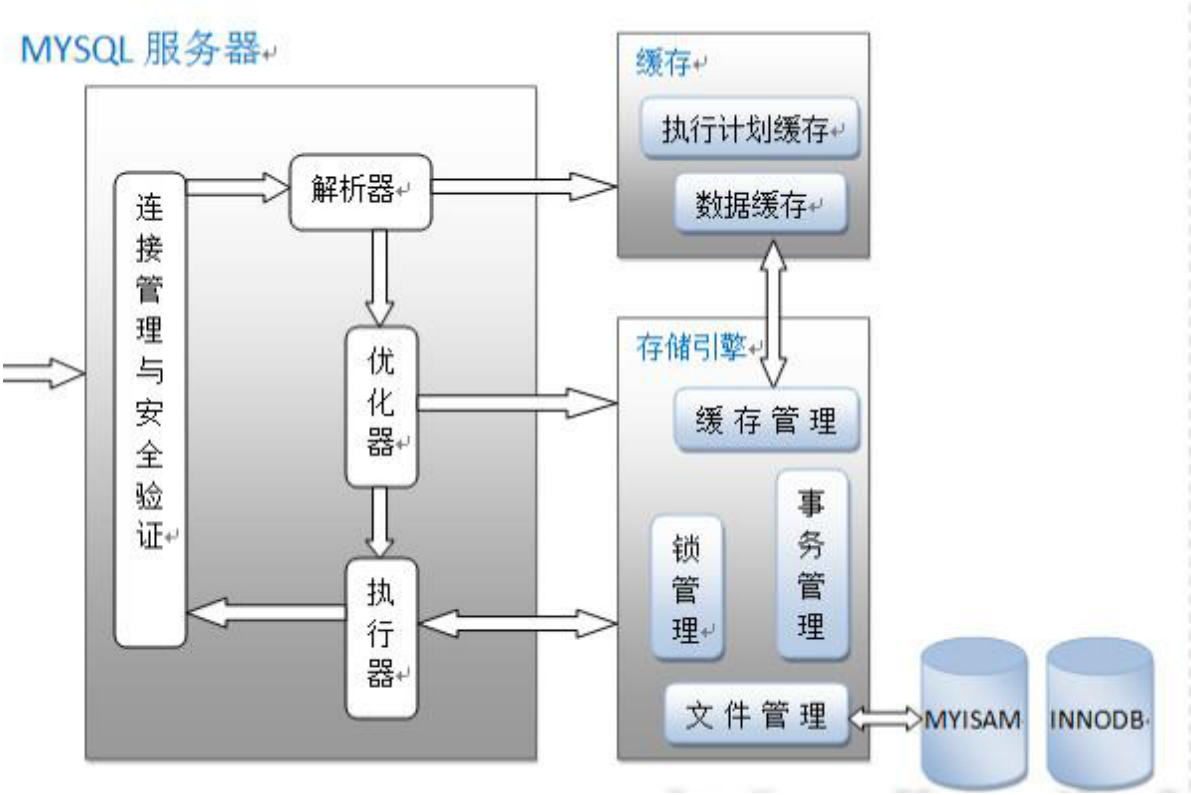
```
create table employee(
id int primary key auto_increment,
name varchar(20),
age int,
dep_id int, -- 外键对应主表的主键
-- 创建外键约束
constraint emp_depid_fk foreign key (dep_id) references department(id)
)

-- 删除employee表的emp_depid_fk外键
alter table employee drop foreign key emp_depid_fk;

-- 在employee表情存在的情况下添加外键
alter table employee add constraint emp_depid_fk
foreign key (dep_id) references department(id);
```

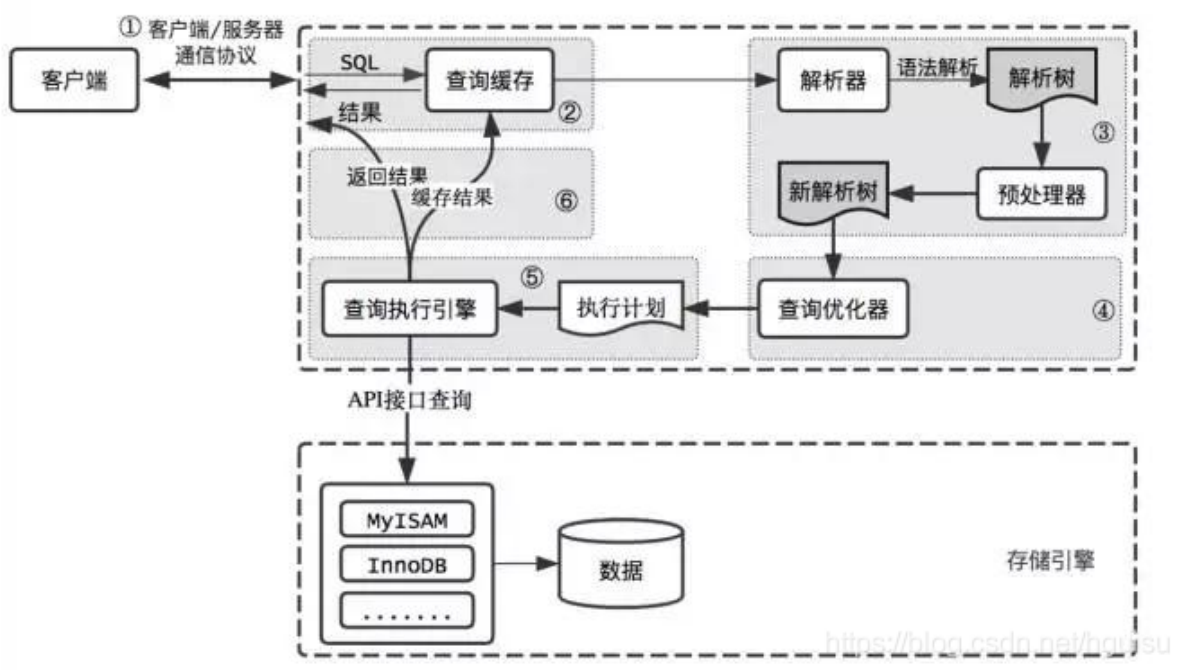
存储引擎

功能	MyISAM	MEMORY	InnoDB
存储限制	256TB	RAM	64TB
支持事务	No	No	Yes
支持全文索引	Yes	No	No
支持B树索引	Yes	Yes	Yes
支持哈希索引	No	Yes	No
支持集群索引	No	No	Yes
支持数据索引	No	Yes	Yes
支持数据压缩	Yes	No	No
空间使用率	低	N/A	高
支持外键	No	No	Yes

面试问题

MySQL架构





MySQL 整个查询执行过程，总的来说分为 6 个步骤：

SQL执行步骤：请求、缓存、SQL解析、优化SQL查询、调用引擎执行，返回结果

1、连接：客户端向 MySQL 服务器发送一条查询请求，与connectors交互：连接池认证相关处理。

2、缓存：服务器首先检查查询缓存，如果命中缓存，则立刻返回存储在缓存中的结果，否则进入下一阶段

3、解析：服务器进行SQL解析(词法语法)、预处理。

4、优化：再由优化器生成对应的执行计划。

5、执行：MySQL 根据执行计划，调用存储引擎的 API来执行查询。

6、结果：将结果返回给客户端，同时缓存查询结果。

1、MySQL语句的执行顺序

1、from 子句组装来自不同数据源的数据；
2、where 子句基于指定的条件对记录行进行筛选；
3、group by 子句将数据划分为多个分组；
4、使用聚集函数进行计算；
5、使用 having 子句筛选分组；
6、计算所有的表达式；
7、select 的字段；
8、使用 order by 对结果集进行排序。

2、MySQL聚合函数

[AVG(distinct] expr)	**求平均值**
[COUNT({*|distinct] } expr)	统计行的数量
[MAX(distinct] expr)	求最大值
[MIN(distinct] expr)	求最小值
[SUM(distinct] expr)	求累加和

3、左连接、右连接、内连接

```
-- 左外链接 left
SELECT * FROM students st LEFT JOIN score sc ON st.sid=sc.stu_id;
-- 右外链接 right
SELECT * FROM students st RIGHT JOIN score sc ON st.sid=sc.stu_id;
-- 显示内连接所有数据:
SELECT * FROM students st INNER JOIN score sc ON st.sid=sc.stu_id;
```

左连接（左外连接）：以左表作为基准进行查询，左表数据会全部显示出来，右表如果和左表匹配的数据则显示相应字段的数据，如果不匹配则显示为 null。
右连接（右外连接）：以右表作为基准进行查询，右表数据会全部显示出来，左表如果和右表匹配的数据则显示相应字段的数据，如果不匹配则显示为 null。
全连接：先以左表进行左外连接，再以右表进行右外连接

4、数据库的三范式

第一范式(1NF)：每个列都不可以再拆分。
第二范式(2NF)：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(3NF)：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。

1 第一范式(确保每列保持原子性)

第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。

第一范式的合理遵循需要根据系统的实际需求来定。比如某些数据库系统中需要用到“地址”这个属性，本来直接将“地址”属性设计成一个数据库表的字段就行。但是如果系统经常会访问“地址”属性中的“城市”部分，那么就非要将“地址”这个属性重新拆分为省份、城市、详细地址等多个部分进行存储，这样在对地址中某一部分操作的时候将非常方便。这样设计才算满足了数据库的第一范式，如下表所示。

用户信息表

编号	姓名	性别	年龄	联系电话	省份	城市	详细地址
1	张红欣	男	26	0378-23459876	河南	开封	朝阳区新华路23号
2	李四平	女	32	0751-65432584	广州	广东	白云区天明路148号
3	刘志国	男	21	0371-87659852	河南	郑州	二七区大学路198号
4	郭小明	女	27	0371-62556789	河南	郑州	新郑市薛店北街218号

上表所示的用户信息遵循了第一范式的要求，这样在对用户使用城市进行分类的时候就非常方便，也提高了数据库的性能。

2．第二范式(确保表中的每列都和主键相关)

第二范式在第一范式的基础之上更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

比如要设计一个订单信息表，因为订单中可能会有多种商品，所以要将订单编号和商品编号作为数据库表的联合主键，如下表所示。

订单信息表

订单编号	商品编号	商品名称	数量	单位	价格	客户	所属单位	联系方式
001	1	挖掘机	1	台	1200000￥	张三	上海玖智	020-1234567
001	2	冲击钻	8	把	230￥	张三	上海玖智	020-1234567
002	3	铲车	2	辆	980000￥	李四	北京公司	010-1234567

这样就产生一个问题：这个表中是以订单编号和商品编号作为联合主键。这样在该表中商品名称、单位、商品价格等信息不与该表的主键相关，而仅仅是与商品编号相关。所以在这里违反了第二范式的设计原则。

而如果把这个订单信息表进行拆分，把商品信息分离到另一个表中，把订单项目表也分离到另一个表中，就非常完美了。如下所示。

订单信息表

订单编号	客户	所属单位	联系方式
001	张三	上海玖智	020-1234567
002	李四	北京公司	010-1234567

订单项目表

订单编号	商品编号	数量
001	1	1
001	2	8
002	3	2

商品信息表

商品编号	商品名称	单位	商品价格
1	挖掘机	台	1200000￥
2	冲击钻	个	230￥
3	铲车	辆	980000￥

这样设计，在很大程度上减小了数据库的冗余。如果要获取订单的商品信息，使用商品编号到商品信息表中查询即可。

3．第三范式(确保每列都和主键列直接相关,而不是间接相关)

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

举例说明：

学号	姓名	性别	家庭人口	班主任姓名	班主任性别	班主任年龄
20150001	李白	男	3口人	陈洁	女	35
20150002	杜甫	男	2口人	陈洁	女	35
20150003	王维	男	4口人	陈洁	女	35
20150004	白居易	男	3口人	李丽	女	32
20150005	刘禹锡	男	4口人	李丽	女	32
20150006	李清照	女	5口人	王安	男	29
20150007	苏轼	男	2口人	南林	男	34
20150008	屈原	男	4口人	南林	男	34
20150009	陶渊明	男	1口人	王安	男	29

上表中，所有属性都完全依赖于学号，所以满足第二范式，但是“班主任性别”和“班主任年龄”直接依赖的是“班主任姓名”，

而不是主键“学号”，所以需做如下调整：

学号	姓名	性别	家庭人口	班主任姓名
20150001	李白	男	3口人	陈洁
20150002	杜甫	男	2口人	陈洁
20150003	王维	男	4口人	陈洁
20150004	白居易	男	3口人	李丽
20150005	刘禹锡	男	4口人	李丽
20150006	李清照	女	5口人	王安
20150007	苏轼	男	2口人	南林
20150008	屈原	男	4口人	南林
20150009	陶渊明	男	1口人	王安

班主任姓名	班主任性别	班主任年龄
陈洁	女	35
陈洁	女	35
陈洁	女	35
李丽	女	32
李丽	女	32
王安	男	29
南林	男	34
南林	男	34
王安	男	29

这样以来，就满足了第三范式的要求。

5、数据库的存储引擎

InnoDB存储引擎

InnoDB 是事务型数据库的首选引擎，支持事务安全表（ACID），支持行锁定和外键。MySQL5.5.5之后，InnoDB 作为默认的存储引擎，InnoDB 主要特性有：

支持事务
灾难恢复性好
为处理巨大数据量的最大性能设计
实现了缓冲管理，不仅能缓冲索引也能缓冲数据，并且会自动创建散列索引以加快数据的获取
支持外键完整性约束。存储表中的数据时，每张表的存储都按逐渐顺序存放，如果没有显示在表定义时指定主键，InnoDB会为每一行生成一个6B的ROWID,并以此作为主键。
被用在众多需要高性能的大型数据库站点上

MyISAM存储引擎

MyISAM 基于 ISAM 的存储引擎，并对其进行扩展。它是在Web、数据存储和其他应用环境下最常使用的存储引擎之一。MyISAM 拥有较高的插入、查询速度，但不支持事务。在 MySQL5.5.5 之前的版本中，MyISAM 是默认的存储引擎。MyISAM 主要特性有：

不支持事务
使用表级锁，并发性差
主机宕机后，MyISAM表易损坏，灾难恢复性不佳
可以配合锁，实现操作系统下的复制备份、迁移
只缓存索引，数据的缓存是利用操作系统缓冲区来实现的。可能引发过多的系统调用且效率不佳
数据紧凑存储，因此可获得更小的索引和更快的全表扫描性能
可以把数据文件和索引文件放在不同目录
使用 MyISAM 引擎创建数据库，将产生3个文件。文件的名字以表的名字开始，扩展名指出文件类型：frm 文件存储表定义，数据文件的扩展名为 .MYD（MYData），索引文件的扩展名是 .MYI（MYIndex）。

MEMORY存储引擎

MEMORY 存储引擎将表中的数据存储在内存中，为查询和引用其他表数据提供快速访问。MEMORY 主要特性有：

使用表级锁，虽然内存访问快，但如果频繁的读写，表级锁会成为瓶颈
只支持固定大小的行。Varchar类型的字段会存储为固定长度的Char类型，浪费空间
不支持TEXT、BLOB字段。当有些查询需要使用到临时表（使用的也是MEMORY存储引擎）时，如果表中有TEXT、BLOB字段，那么会转换为基于磁盘的MyISAM表，严重降低性能
由于内存资源成本昂贵，一般不建议设置过大的内存表，如果内存表满了，可通过清除数据或调整内存表参数来避免报错
服务器重启后数据会丢失，复制维护时需要小心

6、MyISAM和InnoDB的区别

1. myisam是默认表类型不是事物安全的；innodb支持事物。
2. myisam不支持外键；Innodb支持外键。
3. myisam支持表级锁（不支持高并发，以读为主）；innodb支持行锁（共享锁，排它锁，意向锁），粒度更小，但是在执行不能确定扫描范围的sql语句时，innodb同样会锁全表。
4. 执行大量select，myisam是最好的选择；执行大量的update和insert最好用innodb。
5. myisam在磁盘上存储上有三个文件.frm(存储表定义）.myd（存储表数据） .myi（存储表索引）；innodb磁盘上存储的是表空间数据文件和日志文件，innodb表大小只受限于操作系统大小。
6. myisam使用非聚集索引，索引和数据分开，只缓存索引；innodb使用聚集索引，索引和数据存在一个文件。
7. myisam保存表具体行数；innodb不保存。
8. delete from table时，innodb不会重新简历表，而会一行一行的删除。

如何选择：

```
1. **是否要支持事务，如果要请选择innodb，如果不需要可以考虑MyISAM；**
```

2. **如果表中绝大多数都只是读查询，可以考虑MyISAM，如果既有读也有写，请使用InnoDB。**
3. **系统奔溃后，MyISAM恢复起来更困难，能否接受；**
4. **MySQL5.5版本开始Innodb已经成为Mysql的默认引擎(之前是MyISAM)，说明其优势是有目共睹的，如果你不知道用什么，那就用InnoDB，至少不会差。**

7、MySQL中常见索引

MySQL中常见索引有：

- 普通索引
- 唯一索引
- 主键索引
- 组合索引

一、普通索引(index)

普通所以只有一个功能，就是加快查找速度。操作如下

1、先创建一个表

```
create table tab1(
    nid int not null auto_increment primary key,
    name varchar(32) not null,
    email varchar(64) not null,
    extra text,
    index ix_name (name)
)
```

2、创建索引

```
create index 索引名称 on 表名(列名)
```

3、删除索引

```
drop 索引名称 on 表名;
```

4、查看索引

```
show index from 表名;
```

5、注意事项(对于创建索引时如果是BLOB 和 TEXT 类型，必须指定length。)

```
create index index_name on tab1(extra(32));
```

二、唯一索引(unique)

唯一性索引unique index和一般索引normal index最大的差异就是在索引列上增加了一层唯一约束。添加唯一性索引的数据列可以为空，但是只要存在数据值，就必须是唯一的。

1、创建表+唯一索引

```
create table tab2(
    nid int not null auto_increment primary key,
    name varchar(32) not null,
    email varchar(64) not null,
    extra text,
    unique ix_name (name)  -- 重点在这里
)
```

2、创建索引

```
create unique index 索引名 on 表名(列名)
```

3、删除索引

```
drop unique index 索引名 on 表名
```

三、主键索引

在数据库关系图中为表定义一个主键将自动创建主键索引，主键索引是唯一索引的特殊类型。主键索引要求主键中的每个值是唯一的。当在查询中使用主键索引时，它还允许快速访问数据。数据不能为空

1、创建表+主键索引

```
create table in1(
    nid int not null auto_increment,
    name varchar(32) not null,
    email varchar(64) not null,
    extra text,
    primary key(nid),
    index zhang (name)
)
```

2、创建主键

```
alter table 表名 add primary key(列名);
```

3、删除主键

```
alter table 表名 drop primary key;
alter table 表名  modify  列名 int, drop primary key;
```

四、组合索引

组合索引，就是组合查询的意思嘛嘻嘻，将两列或者多列组合成一个索引进行查询

其应用场景为：频繁的同时使用n列来进行查询，如：where name = '张岩林' and email = 666。

1、创建表

```
create table in3(
    nid int not null auto_increment primary key,
    name varchar(32) not null,
    email varchar(64) not null,
    extra text
)
```

2、创建组合索引

```
create index ix_name_email on in3(name,email);
```

如上创建组合索引之后，查询有的会使用索引，有的不会：

- name and email -- 使用索引
- name -- 使用索引
- email -- 不使用索引

其他注意事项

- 避免使用``select *
- \`count(1)或count(列) 代替 count(*)`
- 创建表时尽量时 ``char 代替 ``varchar
- 表的字段顺序固定长度的字段优先
- 组合索引代替多个单列索引（经常使用多个条件查询时）
- 尽量使用短索引
- 使用连接（``JOIN``）来代替子查询(Sub-Queries)
- 连表时注意条件类型需一致
- 索引散列值（重复少）不适合建索引，例：性别不适合

8、count（*）、count（1）和count（列名）的区别

1、执行效果上：

l count(*)包括了所有的列，相当于行数，在统计结果的时候，不会忽略列值为NULL

l count(1)包括了忽略所有列，用1代表代码行，在统计结果的时候，不会忽略列值为NULL

l count(列名)只包括列名那一列，在统计结果的时候，会忽略列值为空（这里的空不是只空字符串或者0，而是表示null）的计数，即某个字段值为NULL时，不统计。

2、执行效率上：

l 列名为主键，count(列名)会比count(1)快

l 列名不为主键，count(1)会比count(列名)快

l 如果表多个列并且没有主键，则 count（1） 的执行效率优于 count（*）

l 如果有主键，则 select count（主键）的执行效率是最优的

l 如果表只有一个字段，则 select count（*）最优。

9、什么是存储过程？有哪些优缺点？

什么是存储过程：存储过程可以说是一个记录集吧，它是由一些T-SQL语句组成的代码块，这些T-SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。

存储过程的优点：

- **能够将代码封装起来**
- **保存在数据库之中**
- **让编程语言进行调用**
- **存储过程是一个预编译的代码块，执行效率比较高**
- **一个存储过程替代大量T_SQL语句 ，可以降低网络通信量，提高通信速率**

存储过程的缺点：

- **每个数据库的存储过程语法几乎都不一样，十分难以维护（不通用）**
- **业务逻辑放在数据库上，难以迭代**

10、drop、delete与truncate分别在什么场景之下使用?

我们来对比一下他们的区别：

drop table

- 1)属于DDL
- 2)不可回滚
- 3)不可带where
- 4)表内容和结构删除
- 5)删除速度快

truncate table

- 1)属于DDL
- 2)不可回滚
- 3)不可带where
- 4)表内容删除
- 5)删除速度快

delete from

- 1)属于DML
- 2)可回滚
- 3)可带where
- 4)表结构在，表内容要看where执行的情况
- 5)删除速度慢,需要逐行删除
- **不再需要一张表的时候，用drop**
- **想删除部分数据行时候，用delete，并且带上where子句**
- **保留表而删除所有数据的时候用truncate**

11、什么是事务?

事务简单来说：**一个Session中所进行所有的操作，要么同时成功，要么同时失败**

ACID — 数据库事务正确执行的四个基本要素

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

一个支持事务（Transaction）中的数据库系统，必需要具有这四种特性，否则在事务过程（Transaction processing）当中无法保证数据的正确性，交易过程极可能达不到交易。

举个例子:**A向B转账，转账这个流程中如果出现问题，事务可以让数据恢复成原来一样【A账户的钱没变，B账户的钱也没变】。**

12、数据库的乐观锁和悲观锁是什么?

悲观锁：当要对数据库中的一条数据进行修改的时候，为了避免同时被其他人修改，最好的办法就是直接对该数据进行加锁以防止并发。这种借助数据库锁机制，在修改数据之前先锁定，再修改的方式被称之为悲观并发控制【Pessimistic Concurrency Control，缩写“PCC”，又名“悲观锁”】。

乐观锁：乐观锁是相对悲观锁而言的，乐观锁假设数据一般情况不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果冲突，则返回给用户异常信息，让用户决定如何去做。乐观锁适用于读多写少的场景，这样可以提高程序的吞吐量。

- 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

悲观锁：

1.悲观锁

select * from eb_sku where sku_id = 1001 for udpate

update eb_sku set stock = 100 - #{quantity} where sku_id = 1001

缺点：性能低

乐观锁：

1.乐观锁

	sku_id	stock	version
a(查询的数据)	1001	100	1
	1001	98	2
b(查询的数据)	1001	100	1

select * from eb_sku where sku_id = 1001

update eb_sku set stock = 100 - #{quantity}, version = #{version} +1 where sku_id = #{skuId} and version = #{version} and stock >= #{quantity}

13、超键、候选键、主键、外键分别是什么？

- 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。**
- **候选键(候选码)：是最小超键，即没有冗余元素的超键。**

- **主键(主码)：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，也叫做选中的候选键。**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）。
- **外键：在一个表中存在的另一个表的主键称此表的外键。**

> 比如一个小范围的所有人，没有重名的，考虑以下属性
>
> 身份证 姓名 性别 年龄
>
> 身份证唯一，所以是一个超键
> 姓名唯一，所以是一个超键
> （姓名，性别）唯一，所以是一个超键
> （姓名，性别，年龄）唯一，所以是一个超键
> --这里可以看出，超键的组合是唯一的，但可能不是最小唯一的
>
> --主键是选中的一个候选键

候选码和主码：

例子：邮寄地址（城市名，街道名，邮政编码，单位名，收件人）

- **它有两个候选键:{城市名，街道名} 和 {街道名，邮政编码}**
- **如果我选取{城市名，街道名}作为唯一标识实体的属性，那么{城市名，街道名} 就是主码(主键)**

14、SQL 约束有哪几种

- NOT NULL: 用于控制字段的内容一定不能为空（NULL）。
- UNIQUE: 控件字段内容不能重复，一个表允许有多个 Unique 约束。
- 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。
- 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。
- CHECK: 用于控制字段的值范围。

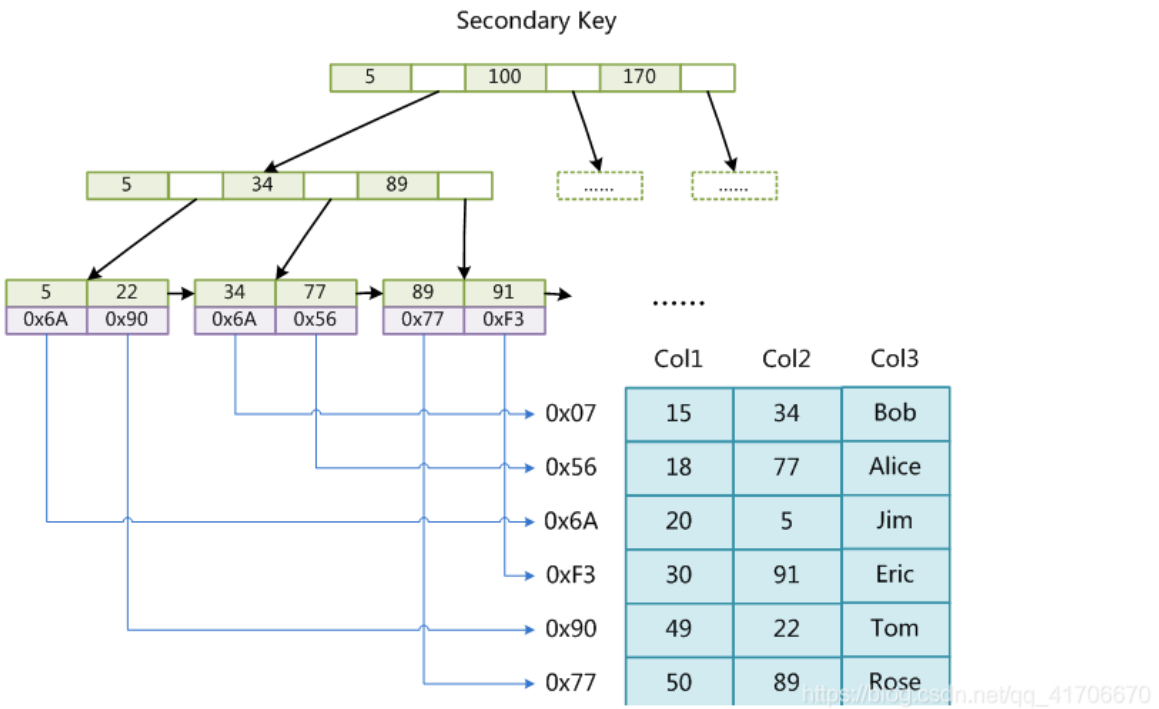
15、MyIASM和Innodb两种引擎所使用的索引的数据结构是什么

答案:都是B+树!

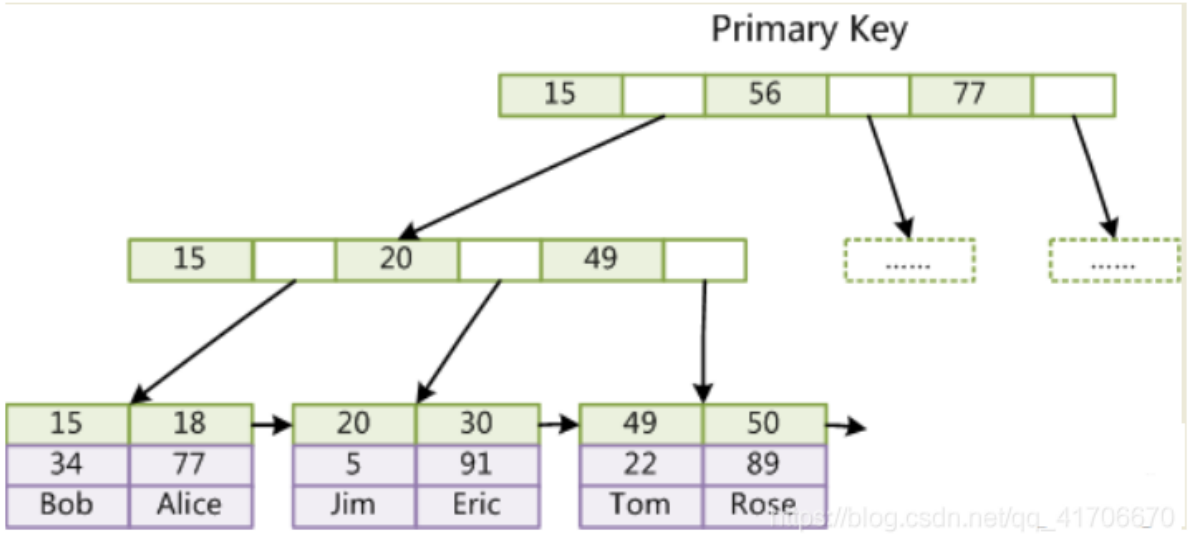
MyIASM引擎，B+树的数据结构中存储的内容实际上是实际数据的地址值。也就是说它的索引和实际数据是分开的，**只不过使用索引指向了实际数据。这种索引的模式被称为非聚集索引。**

Innodb引擎的索引的数据结构也是B+树，**只不过数据结构中存储的都是实际的数据，这种索引有被称为聚集索引。**

myisam:



innodb:



16、varchar和char的区别

Char是一种固定长度的类型，varchar是一种可变长度的类型

17、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：

- 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。
- db权限表：记录各个帐号在各个数据库上的操作权限。
- table_priv权限表：记录数据表级的操作权限。
- columns_priv权限表：记录数据列级的操作权限。
- 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。

18、视图和表的区别

视图是已经编译好的sql语句，是基于sql语句的结果集的可视化的表，而表不是
视图是窗口，表示内容
视图没有实际的物理记录，而表有
视图的建立和删除只影响视图本身，不影响对应的表

两者的联系：视图是基于基本表上建立的表，它的结构和内容都来自基本表，它依据基本表存在而存在。一个视图可以对应一个基本表，也可以对应多个基本表。

19、存储过程

- 存储过程就是具有名字的一段代码，用来完成一个特定的功能。

创建“pro_add”的存储过程来求两数和:

```
CREATE PROCEDURE pr_add( a INT, b INT ) BEGIN
    DECLARE c INT;
    IF a IS NULL THEN SET a = 0;
    END IF;
    IF b IS NULL THEN SET b = 0;
    END IF;
    SET c = a + b;
SELECT c as sum;
```

使用时：

```
call pr_add(1, 2);
```

20、触发器

触发器是一种特殊类型的存储过程，它不同于存储过程，主要是通过事件触发而被执行的，即不是主动调用而执行的；而存储过程则需要主动调用其名字执行

触发器：trigger，是指事先为某张表绑定一段代码，当表中的某些内容发生改变（增、删、改）的时候，系统会自动触发代码并执行。

创建触发器：

```
delimiter 自定义结束符号
create trigger 触发器名字 触发时间 触发事件 on 表 for each row
begin
    -- 触发器内容主体，每行用分号结尾
end
自定义的结束符合

delimiter ;
```

`on 表 for each`：触发对象，触发器绑定的实质是表中的所有行，因此当每一行发生指定改变时，触发器就会发生

- **触发时间**
  当 SQL 指令发生时，会令行中数据发生变化，而每张表中对应的行有两种状态：数据操作前和操作后
  - before：表中数据发生改变前的状态

    - after：表中数据发生改变后的状态
    - PS：如果 before 触发器失败或者语句本身失败，将不执行 after 触发器(如果有的话)
- **触发事件**

  触发器是针对数据发送改变才会被触发，对应的操作只有

    - INSERT
    - DELETE
    - UPDATE

example：

如果**订单表**发生数据插入，对应的商品库存应该减少。因此这里对订单表创建触发器

```
delimiter ##
-- 创建触发器
create trigger after_insert_order after insert on orders for each row
begin
    -- 更新商品表的库存，这里只指定了更新第一件商品的库存
    update goods set goods_num = goods_num - 1 where id = 1;
end
##

delimiter ;
```