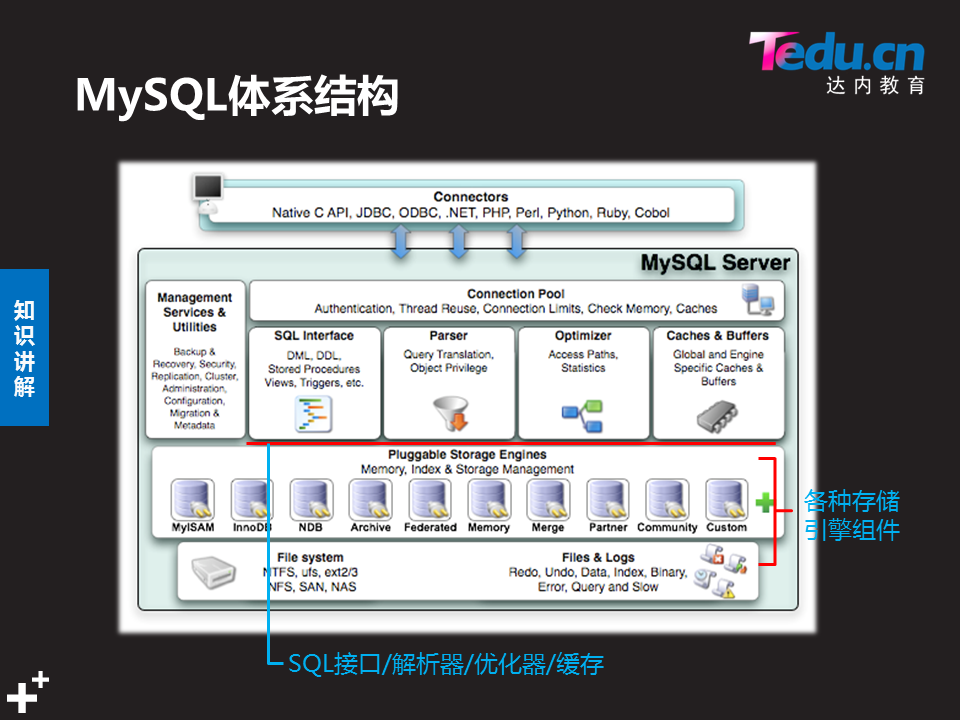
**Morning：**

1. MySQL体系结构



|  |  |
| --- | --- |
| Connectors | 客户端访问(通过SQL语言、各种开发语言) |
| 管理工具 | 软件安装后提供的命令 |
| 连接池 | 服务端收到客户端请求后，连接池检查空闲进程+硬件资源(内存、CPU)，再执行连接 |
| SQL接口 | 将SQL命令传递给mysqld服务进程处理 |
| Parser(分析器) | 检查SQL命令语法、错误 |
| Optimizer(优化器) | 优化要执行的SQL命令，节省系统资源、加快处理速度 |
| 查询缓存 | 缓存查找过的数据  -服务启动时，系统读取缓存配置文件，从物理内存中划分一部分(默认8M)作查询缓存。  -客户端发送查询请求，先从缓存查找；若无缓存，再从硬盘中查找，查询结果先加入缓存，再返回给客户端。 |
| 存储引擎 | 提供表功能、决定表存储方式 |
| 文件系统 | 硬盘存储数据的方式 |

2、存储引擎

**-What’s存储引擎**

存储引擎是MySQL服务软件自带的功能程序，处理表的处理器；

决定表的功能和数据存储方式。

MySQL 5.0/5.1默认为MyISAM；MySQL 5.5/5.6默认Innodb。

**-修改存储引擎**

mysql> show engines;

#列出可用的存储引擎类型

mysql> create table t1( id int, name char(5))engine=innodb;

#建表时指定存储引擎

mysql> alter table 表名 engine=innodb;

#已有表修改存储引擎（若表中有数据，最好不要修改存储引擎）

[root@mysql50 ~]# sed -i '4a default-storage-engine=myisam' /etc/my.cnf

[root@mysql50 ~]# systemctl restart mysqld

#永久修改数据库默认存储引擎

**-存储引擎特点、表文件**

|  |  |  |
| --- | --- | --- |
| 存储引擎 | 主要特点 | 相关表文件 |
| MyISAM | 支持表级锁  不支持事务、事务回滚、外键 | .frm 存放表结构  .MYI 存放索引信息  .MYD 存放数据 |
| Innodb | 支持行级锁  支持事务、事务回滚、外键 | .frm 存放表结构  .idb存放表数据及索引信息  ibdata1、ib\_logfile0、ib\_logfile1  事务日志文件 |

**-存储引擎的选择**

Myisam：适合查询访问次数多的表，节省系统资源(表级锁)。

Innodb：适合写访问次数多的表，并发访问量大(行级锁)。

3、事务、锁机制

**-事务、锁概念解释**

1）事务

事务：连接数据库～断开连接的整个过程

事务回滚：在访问过程中，在执行提交之前，都可以恢复到操作前的状态

事务日志文件：记录所有innodb的表执行过的操作，放在/var/lib/mysql/

2）MySQL锁机制

表级锁：一次直接对整张表加锁

页级锁：一次锁定相邻的一组数据

行级锁：只锁定某一行 **锁粒度**

读锁(共享锁)：对表作读操作时上锁，其他人只能读，无法写

写锁(互斥锁、排它锁)：对表作写操作时上锁，其他人无法读写 **锁类型**

锁的释放：客户端读、写完成后，自动释放锁机制

mysql> show status like ‘Table\_lock%’; #查看表锁状态

+-------------------------------+--------+

| Variable\_name | Value |

+-------------------------------+--------+

| Table\_locks\_immediate | 103 |

| Table\_locks\_waited | 0 | #等待释放锁

+-------------------------------+--------+

**-事务特性**

1）事务特性(ACID)：原子性A、一致性C、隔离性I、持久性D

Atomic：所有操作成功，才算成功

Consistency：提交前数据保持不变、提交后才执行更新

Isolation：不同用户的事务操作互不影响

Durability：数据一旦提交，不可改变

2）验证事务特性

mysql> show variables like "autocommit"; #查看自动提交状态

mysql> set autocommit=off; #关闭当前会话自动提交功能

mysql> rollback; #数据回滚

mysql> commit; #提交数据

**Afternoon：**

1、数据导入导出

**-数据导入(系统文件-->表)**

语法： Load data infile “文件名(**绝对路径**)” into table 表名

Fields terminated by “**列分隔符**”

Lines terminated by “\n”;

注意：-表的字段类型要与文件的每一列匹配

-禁用Selinux

1）导入文件存放位置

mysql> show variables like "secure\_file\_priv";

#查看存放位置

+---------------------+---------------------------+

| Variable\_name | Value |

+---------------------+---------------------------+

| secure\_file\_priv | /var/lib/mysql-files/ |

+---------------------+---------------------------+

[root@mysql50 ~]# mkdir /myload && chown mysql /myload/

[root@mysql50 ~]# sed -i '4a secure\_file\_priv=”/myload”' /etc/my.cnf && systemctl restart mysqld

#修改存放位置(无法通过mysql命令行改)

#注意：mysql用户对目录必须要有w权限

2）准备导入文件

mysql> system cp /etc/passwd /var/lib/mysql-files/

#在mysql服务中执行系统命令

mysql> system ls /var/lib/mysql-files/

1. 创建表

mysql> create table user(

-> name char(30),

-> password char(1),

-> uid int,

-> gid int,

-> comment char(30),

-> home char(30),

-> shell char(15) );

1. 导入数据

mysql> load data infile "/var/lib/mysql-files/passwd"

-> into table db3.user

-> fields terminated by ":"

-> lines terminated by "\n";

5）添加行号字段

mysql> alter table db3.user add

-> id int primary key auto\_increment first;

**-数据导出(表记录-->系统文件)**

语法： SQL查询 into outfile “文件名(**绝对路径**)”

Fields terminated by “**列分隔符**” #默认为tab

Lines terminated by “\n”; #默认为\n

注意：-导出的内容由SQL查询语句决定

-导出的是表中的记录，不包括字段名

-只能导出到secure\_file\_priv目录当中

-禁用Selinux

案例：

mysql> select name,uid,shell from user into outfile "/myload/db3.user";

[root@mysql50 ~]# cat /myload/db3.user

root 0 /bin/bash

bin 1 /sbin/nologin

daemon 2 /sbin/nologin

2、管理表记录

**-插入表记录**

1. 插入格式

|  |  |
| --- | --- |
| 1. 记录   所有字段 | insert into 表名 values (字段值列表); |
| N条记录  所有字段 | insert into 表名 values  (字段值列表),(字段值列表)...; |
| 1. 记录   指定字段 | insert into 表名(字段名列表) values (字段值列表); |
| N条记录  指定字段 | insert into 表名(字段名列表) values  (字段值列表),(字段值列表)...; |

例：mysql> insert into db3.user(name,uid) values

-> ("bob",2000),

-> ("gjq",2001);

2）注意事项

-对于字符类型的字段，要用双引号括起来

-字段名列表、字段值列表中以逗号分隔

**-查询表记录**

1）查询格式

Select 字段1,...,字段N From 表名;

Select 字段1,...,字段N From 表名 where 条件表达式;

例：mysql> select id,name,uid from db3.user where uid>=2000;

2）注意事项

-使用 \* 可匹配所有字段

-字段1,...,字段N控制显示哪些列，where控制显示哪些行

**-更新表记录**

1）更新格式

Update 表名 set 字段1=值1,...,字段N=值N;

Update 表名 set 字段1=值1,...,字段N=值N where 条件表达式;

例：mysql> update db3.user set name="gaojinquan"

-> where uid=2001;

2）注意事项

-若不使用where限定条件，会更新所有记录(整列更新)

-限定条件时，只更新匹配条件的行

**-删除表记录**

1. 删除格式

Delete from 表名 Where 条件表达式;

Delete from 表名;

例：mysql> delete from db3.user where uid>=2000;

3、匹配条件

匹配条件常用于查询、更新、删除

**1）基本匹配条件**

|  |  |  |
| --- | --- | --- |
| 匹配方式 | 类型 | 用途 |
| 数值比较 | = | 等于 |
| >、>= | 大于、大于或等于 |
| <、<= | 小于、小于或等于 |
| != | 不等于 |
| 字符比较  匹配空/非空 | = | 相等 |
| != | 不相等 |
| IS NULL | 匹配空 |
| IS NOT NULL | 非空 |
| 逻辑匹配 | OR | 逻辑或 |
| AND | 逻辑与 |
| ! 或 not | 逻辑非 |
| ( ) | 提高优先级 |
| 范围内匹配  去重显示 | in (值列表) | 在...里 |
| not in (值列表) | 不在...里 |
| between n1 and n2 | 在n1~n2...之间 |
| distinct 字段名 | 去重显示 |

Tips：

-关于逻辑非，以下两种效果一致

mysql> select \* from db3.user where not uid=5;

mysql> select \* from db3.user where uid!=5;

-关于between，边界值也会匹配

mysql> select \* from db3.user where uid between 0 and 2;

-关于去重显示，相当于显示该字段已有的值

mysql> select distinct shell from db3.user;

-关于优先级

逻辑/算术运算符优先级：! > 负号 > \* / % > +- > AND > OR

**2）高级匹配条件**

|  |  |  |
| --- | --- | --- |
| 匹配方式 | 语法 | 备注 |
| 模糊查询 | where 字段名 like ‘通配符’ | \_匹配单个字符  %匹配0~N个字符 |
| 正则表达式 | where 字段名 regexp ‘正则’ | 支持所有基本/扩展正则  常用^ $ . \* [] | |
| 四则运算 | 字段必须是数值类型 | + - \* / % |

Tips：

-关于模糊查询

mysql> select \* from user where shell like "%bash%";

#模糊查询shell字段包含”bash”的行

-关于正则表达式

mysql> select \* from user where uid regexp '^....$';

#查询uid字段值为4位数的行

-关于四则运算

mysql> update user set uid=uid+1;

#对所有uid字段自加1

mysql> select L1,L2,(L1+L2)/2 as avg from user where L1=10;

#对L1、L2取平均值avg显示

+-------+-------+-------------+

| Len1 | Len2 | avg |

+-------+-------+-------------+

| 10 | 20 | 15.0000 |

+-------+-------+-------------+

4、操作查询结果

**-聚集函数(MySQL内置函数)**

|  |  |
| --- | --- |
| 函数 | 功能 |
| avg(字段名) | 统计字段平均值 |
| sum(字段名) | 统计字段之和 |
| min(字段名) | 统计字段最小值 |
| max(字段名) | 统计字段最大值 |
| count(字段名) | 统计字段值个数 |

注意：字段类型必须为数值类型

mysql> select avg(uid),min(uid),max(uid),count(uid) from user ;

+----------------+-------------+------------+-----------------+

| avg(uid) | min(uid) | max(uid) | count(uid) |

+----------------+-------------+------------+-----------------+

| 336.5000 | 1 | 1005 | 44 |

+----------------+-------------+------------+-----------------+

**-查询结果排序**

语法：**SQL查询 order by 字段名 [asc | desc];**

注意：字段类型一般为数值类型；

asc|desc为升序|降序，不写默认为升序

mysql> select uid,name from user where uid>=1000 order by uid desc;

#对uid>=1000的记录按uid值降序排序

**-查询结果分组**

语法：**SQL查询 group by 字段名;**

注意：分组相当于根据查询结果，把字段的所有值列出来

mysql> select shell from user where uid>10 and uid<100 group by shell;

+---------------------+

| shell |

+---------------------+

| /bin/false |

| /sbin/nologin |

+---------------------+

**-查询结果过滤**

语法：**SQL查询 having 条件表达式;**

mysql> select name from user where shell!="%bin/bash%" having name="mysql";

#比使用and判断更节省系统资源

**-限制查询结果显示行数**

|  |  |
| --- | --- |
| 语法 | 备注 |
| SQL查询 limit N; | 显示查询结果前N条记录 |
| SQL查询 limit N,M; | 从查询结果的第N+1条开始，共显示M行(从0开始) |

mysql> select id,name from user where id<10 limit 2,2;

+-------+------------+

| id | name |

+-------+------------+

| 3 | daemon |

| 4 | adm |

+-------+------------+