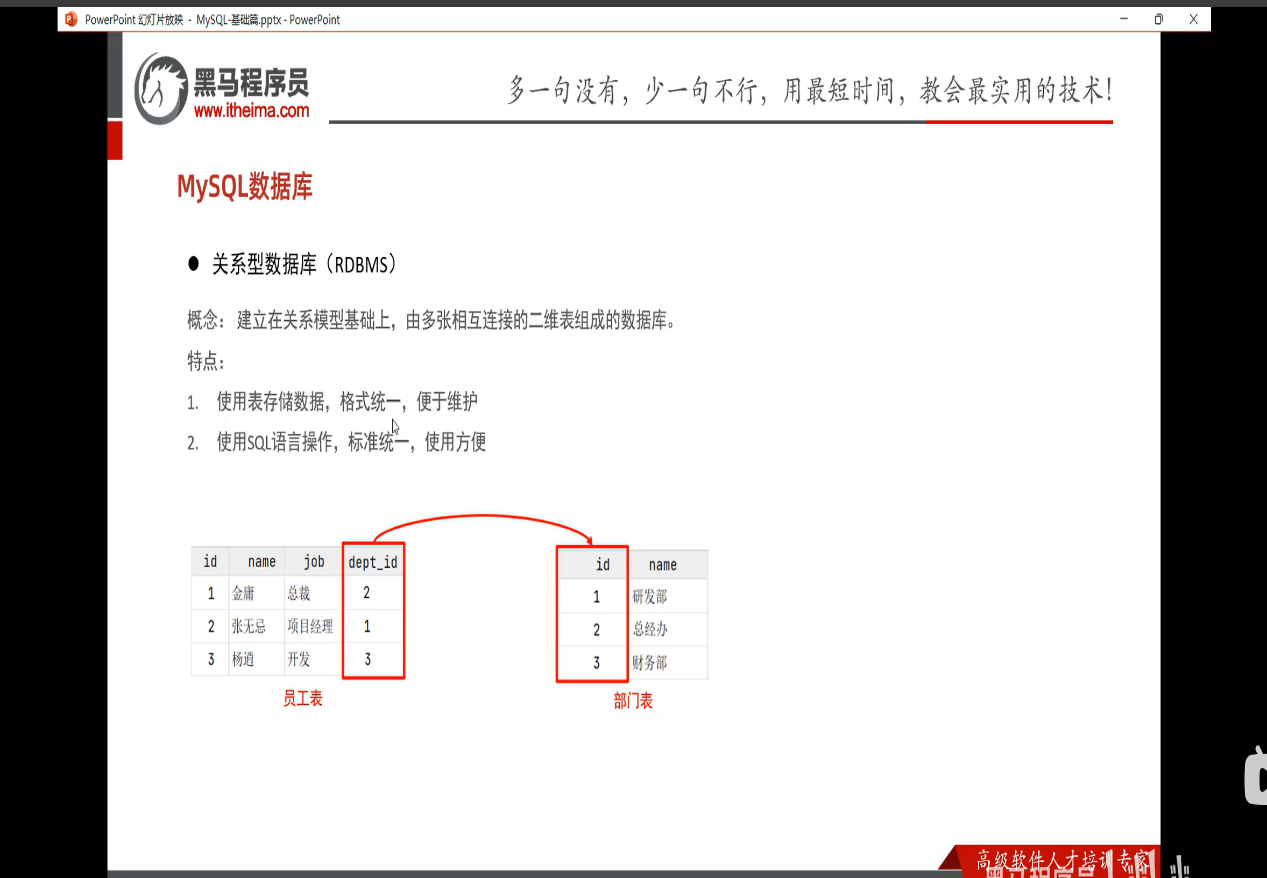
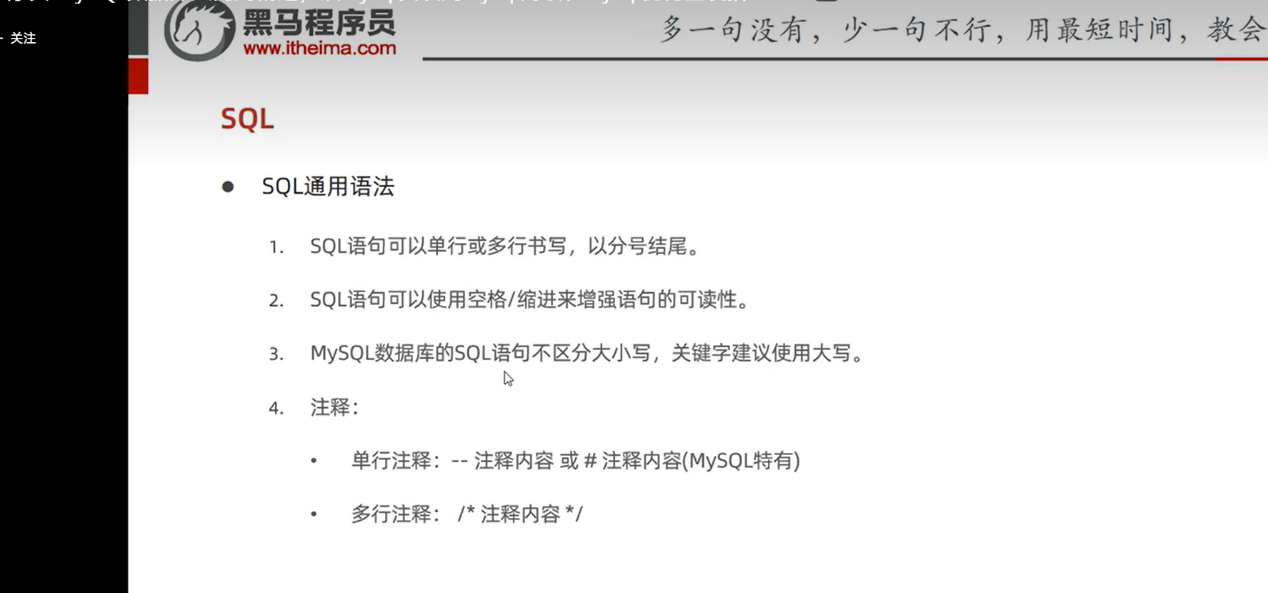
MySQL

1. 数据模拟：客户端将数据存储到MySQL中，DBMS对于数据进行管理，分配到不同的数据库中，不同数据库中还有多张表格，存储数据
2. 





1. DDL
2. 常见的操作：

查询所有数据库：show databases

查询当前数据库：select database

创建：create database [if not exists] 数据库名称 [default charset 字符集] [collate 排序规则]

删除：drop database[if exists]数据库名

使用：use 数据库名

注意：在创建时，如果要使用utf-8，则最好在后面加上mb4，这样它就支持一些占4个字节的字符；默认是这个字符集

在创建数据库时，只要同名，即使改变不同的字符集也不能够创建

1. DDL-表操作-查询

查询当前数据库所有的表：show tables;

查询表结构：desc 表名；

查询指定表的建表语句：show create table 表名；

1. DDL-表操作-创建

Create table 表名（

字段1 字段1类型[comment 字段1注释]，

字段2 字段2类型[comment 字段2注释],

…

字段n 字段n类型[comment 字段n注释]）[comment 表注释]；

字符类型中字符串类型在MySQL中是varchar（字符长度）；

1. DML（数据操作语言，完成数据的增改删）
2. 常见操作：

添加数据（insert）：

给指定字段添加数据：Insert into 表名（字段名1，字段名2……）values(值1，值2);

给全部字段添加数据：Insert into 表名 values（值1，值2）

3、批量添加数据：Insert into 表名（字段名1，字段名2……）values(值1，值2)，(值3，值4)

Insert into 表名 values （值1，值2），（值3，值4）

修改数据（update）：

删除数据（delete）：

学习大方向：

图形用户界面, 文本

中度可信度描述已自动生成

二、

1、RDBMS（关系型数据库）

特点：通过行和列的形式进行存储，形成表格。

将复杂的数据结构归结为简单的二维关系

非RDBMS：相当于RDBMS舍弃一部分功能，以实现更高的功能

种类：键值型数据库、文档型数据库、搜索引擎（倒排索引）、列式数据库、图形数据库

2、关系型数据库设计规则

（1）表、记录、字段：

E-R（实体-联系）：实体集、属性、联系集

一个实体集在数据库中对应一张表，表中的一行代表一个实体（或记录），一个属性对应数据库中的一列（或一个字段）

表格

描述已自动生成

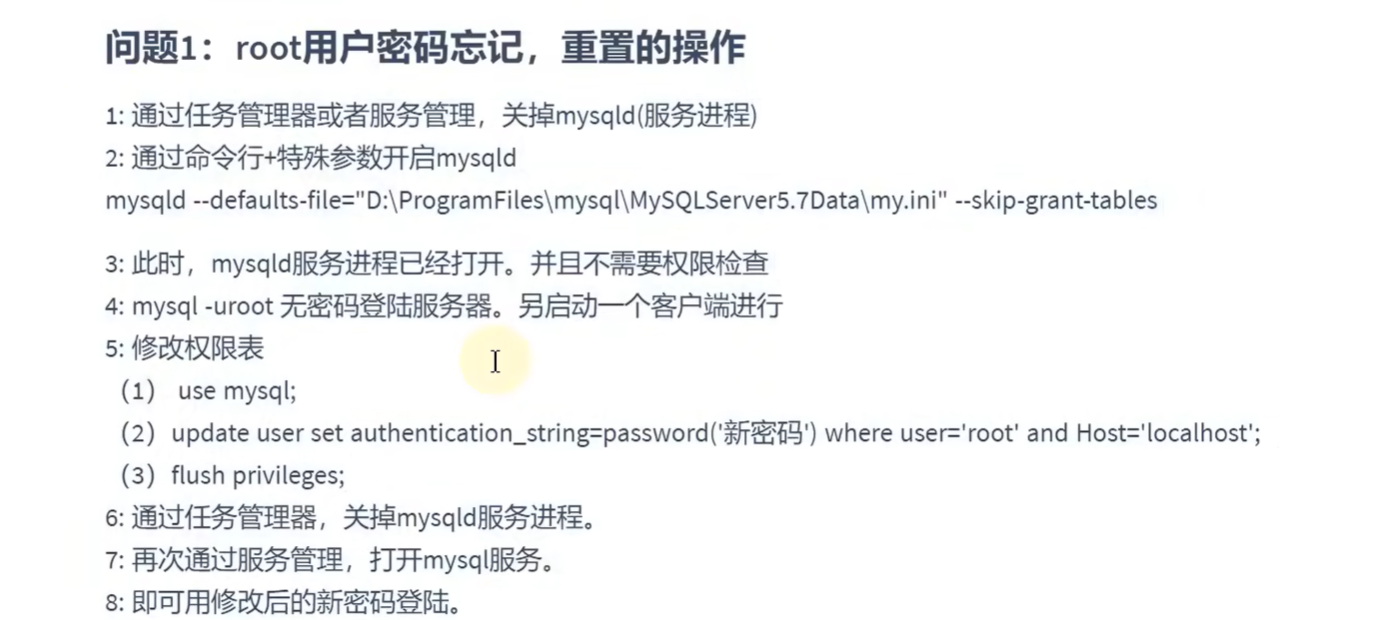
与java的对应关系：表：类； 对象：记录（一行）； 列：字段或属性

1. 表的关联关系：一对一关联、一对多关联、多对关联、自我引用
2. 一对一关联：当一张表格中的属性过多，可以拆分成为两张表格，将不常用的信息放到另一张表中，提高访问效率
3. 一对多关系：例如，部门表和员工表
4. 多对多：创建一个第三个表，链接表，将两个一对多的表格关联起来

链接表：两个表与对方存在不同的关系，用链接表将这样不同的关系存储起来

3、常见的配置问题

（1）重置密码



（2）没有选择数据库就操作表格和数据



修改数据库和表的字符编码





1. 注释：单行注释：#注释文字 或者 – 注释文字（--后面必须带空格）

多行注释：/\*注释文字\*/

DML

1. 更新数据

语法：update 表名 set 列1=值1,列2=值2 where 条件

注意：一般都加条件，否则会进行批量的添加

1. 删除数据

语法：delete from 表名 where 条件

1. 计算列

作用：有一些列的值使用过其它列计算的到，如果数据比较多，需要先找到相关计算的列，才能去输出要找的列的值，比较慢

语法：

1. 创建表时：列n generated always AS （条件）virtual，

字符集设置

1. 整型：tinint、smallint、mediumint、int、bigint
2. 浮点型：float、double
3. 定点型：decimal
4. 位类型：bit
5. 日期时间类型：year、time、date、datetime、timestamp
6. 文本字符串类型：char、varchar、tinytxt、text、mediumtext、longtext
7. 枚举类型：enum
8. 集合类型：set
9. 二进制字符串类型：binary、varbinary、tinybinary、blob、mediumblob、longblob
10. JSON：JSON对象、JSON数组

Mysql关键字

1. null、not null、default、primary key、auto\_increment

unsigned（无符号）、Character set name（指定一个字符集）

排序操作

-- 默认情况下按照升序排列

-- DESC：降序进行排列

-- ASC：进行升序排列（默认）

-- 列的别名只能在order by中进行使用，不能再where中使用

SELECT \*

FROM student

ORDER BY (chinese+Math+English) DESC;

SELECT Chinese chinese,ID,`Name`

FROM student

where chinese >90

ORDER BY chinese DESC,ID DESC;

分页

语法：limit 起始位置（从0开始），条目数

Limit 条目数 offset 起始位置

显示页数：显示第一页，例如，limit 0，20

显示第二页，例如，limit 19 20

-- 多表查询

-- 误用笛卡尔积,会产生大量无用数据

-- 错误原因：缺少连接的条件

SELECT `Name`,department\_id

FROM student,employees;

-- 正确写法

-- 两表应该有关联字段，否则默认使用笛卡尔积

SELECT Chinese,department\_id

FROM student,employees

WHERE student.ID=employees.id;

-- 注意：当两张表有相同的属性名，进行查询的时候要声明到底是哪张表的

-- 建议：当进行多表查询的时候，尽量在属性名前面加上所在的表名。

-- 可以的话最好给要查询的表起别名，在where和select中使用

-- 推荐：多表查询中连接条件一般放在join in上

-- 不推荐：连接条件放在where上，避免连接的条件不够明确，尤其是复杂查询时

SELECT student.ID,salary

FROM student

JOIN employees ON employees.id=student.ID;

-- 内连接和外连接

-- 内连接：合并具有同一列的两个以上的表的行，结果只显示包含匹配列相同的数据

-- 外连接：：合并具有同一行的两个以上的表的行，结果显示包括匹配列不相同的所有数据

-- 92和99语法---外、内连接

-- 92（mysql不支持）：根据是左连接还是右连接，在等值条件的其所在位置使用（+）

-- 99：使用JOIN ON，左外在JOIN前加 left，右外同理；满外连接加full(mysql不支持)

SELECT Math,annual\_salary

FROM student

LEFT JOIN employees

ON employees.id=student.ID;

图示

描述已自动生成

Json的七种实现方式

-- Union查询合并

-- 要求：合并的两个查询必须列数相同，数据类型对应相同

-- 解决的问题：mysql不支持全连接的问题

-- 书写方式

-- 第一种:union:会对于重复的数据进行去重处理

-- 第二种:union all:不会对于重复的数据进行去重

-- 选择:如果不存在重复数据或不需要去除重复数据,则推荐第二种

-- 七种实现

-- 1、交集（内连接）

SELECT student.ID,Chinese,Math,English,salary

FROM student

JOIN employees ON student.ID=employees.id;

-- 2、左外连接

SELECT student.ID,Chinese,English,Math

FROM student

LEFT JOIN employees ON student.ID=employees.id;

-- 3、右外连接

SELECT employees.department\_id,Chinese,Math,English

FROM student

RIGHT JOIN employees ON student.ID=employees.id;

-- 4、左差集

SELECT student.ID,Chinese,Math,employees.salary

FROM student

LEFT JOIN employees ON student.ID=employees.id

WHERE employees.id IS NULL;

-- 5、右差集

SELECT employees.department\_id,Chinese,Math,English

FROM student

RIGHT JOIN employees ON student.ID=employees.id

WHERE student.ID IS NULL;

-- 6、满外连接

SELECT student.ID,Chinese,English,Math,salary

FROM student

LEFT JOIN employees ON student.ID=employees.id

UNION ALL

SELECT employees.department\_id,Chinese,Math,English,salary

FROM student

RIGHT JOIN employees ON student.ID=employees.id

WHERE student.ID IS NULL;

-- 7、两个差集取并集

SELECT student.ID,Chinese,Math,employees.salary

FROM student

LEFT JOIN employees ON student.ID=employees.id

WHERE employees.id IS NULL

UNION ALL

SELECT employees.department\_id,Chinese,Math,English

FROM student

RIGHT JOIN employees ON student.ID=employees.id

WHERE student.ID IS NULL;

-- SQL99的新特性

-- 1、自然连接

-- 作用：自动寻找两张表格中存在相同字段的列并进行自动匹配

-- 优缺点:更简洁,灵活性变差

-- 注意：列名也必须保持一致

SELECT Chinese,Math,English,salary

FROM student

NATURAL JOIN employees;

-- 2、using关键字

-- 作用：对于连接条件的简化

-- 注意:列名要一致，大小写不区分

SELECT Chinese,Math,English,salary

FROM student

JOIN employees

USING (id);

-- 书写注意：进行多表查询的时候，禁止连续使用3次以上join，数据类型保持绝对一致，保证被关联的字段都要有索引

-- 函数

-- 种类：单行函数、多行函数

-- 单行函数

-- 1、数值函数

-- 种类：

-- abs：绝对值；sign：取符号，正1，负-1,0取0；

-- pi:圆周率；ceil/ceiling：向上取整；floor：向下取整

-- mod：取余

SELECT ABS(-25),SIGN(-10),SIGN(10),SIGN(0),PI(),CEIL(-30.5),

CEILING(-3.14),FLOOR(15.5),MOD(3,5),MOD(6,2)

FROM DUAL;

-- least：取最小值；greatest：取最大值

-- rand:取随机数，默认是0到1，可以用四则运算控制范围

-- 注意：rand（x）:相同的x值，产生的随机数是相同的

SELECT LEAST(15,17,abs(-20)),GREATEST(18,26,77),RAND(),

rand()\*100,rand(1),rand(1)

FROM DUAL;

-- 四舍五入，截断操作(不会四舍五入)

-- 注：第二个参数值可以取负值，例如，round（14,-1），就是只保留十位

-- 如果负数超过整数位时，会显示为0

SELECT ROUND(14.234),ROUND(15.734),ROUND(15.165,-1),

TRUNCATE(14.356,0),TRUNCATE(17.25,1)

FROM DUAL;

-- 指数和对数

-- 指数：pow/power(底数，次幂)、exp（次幂）：自然数（e）的次幂

SELECT pow(2,5),POWER(2,4),EXP(2)

FROM DUAL;

-- 对数

SELECT LN(EXP(2)),log2(8),log10(10)

FROM DUAL;

-- 进制的转换

-- bin：10转2；hex：10转16；oct：10转8；

-- conv（数，原进制，转后进制）

SELECT BIN(10),HEX(10),OCT(10),CONV(10,10,16)

FROM DUAL;

-- 字符串函数

-- ASCll：返回首字母的ASCll值，char\_length:返回字符长度

-- length:返回字母长度，汉字占3字节，字母占1字节

-- concat：连接字符串

-- concat\_WS:第一个参数为连接符

SELECT ASCII("啊这"),CHAR\_LENGTH("你好世界！"),LENGTH("你好世界！"),

CONCAT("你好",",","世界"),CONCAT\_WS("-",1,"你好吗？")

FROM DUAL;

-- insert:插入到指定位置，如果位置有元素，则进行取代，尾部就添加，

-- 不过取代数量要写0

-- replace：取代给定元素，字符串中没有该元素，会报错

SELECT INSERT("你好，世界",6,0"我的"),

REPLACE("你好，世界","世界","我的朋友")

FROM DUAL;

-- uper：全大写；lower：全小写

-- left:从左数的几个；right：从右数的几个

-- 注：超过总长度则全部显示

-- lpad（列名，占位长度，长度不够时占位元素）：右对齐

-- rpad：左对齐

-- trim：去除首尾空格，trim（元素 from 字符串）：去除首尾该元素

-- space（数量）：添加空格

-- strcmp（str1，str2）：比较两字符串从首字母开始的ASCll大小

-- substr（开始位置，截的长度）

-- locate（寻找的字符串，母字符串）：返回子字符串首次出现首字母的位置

-- elt（位置，字符1，字符2，...）：返回位置上的字符串

-- field（目标字符，字符1，字符2，...）：返回目标字符所在的位置

-- nullif(字符1，字符2)：如果两字符相等返回null，否则返回字符

-- 时间函数

-- curdate/curtime：获取当前时期、时间

-- now：获取当前时期和时间

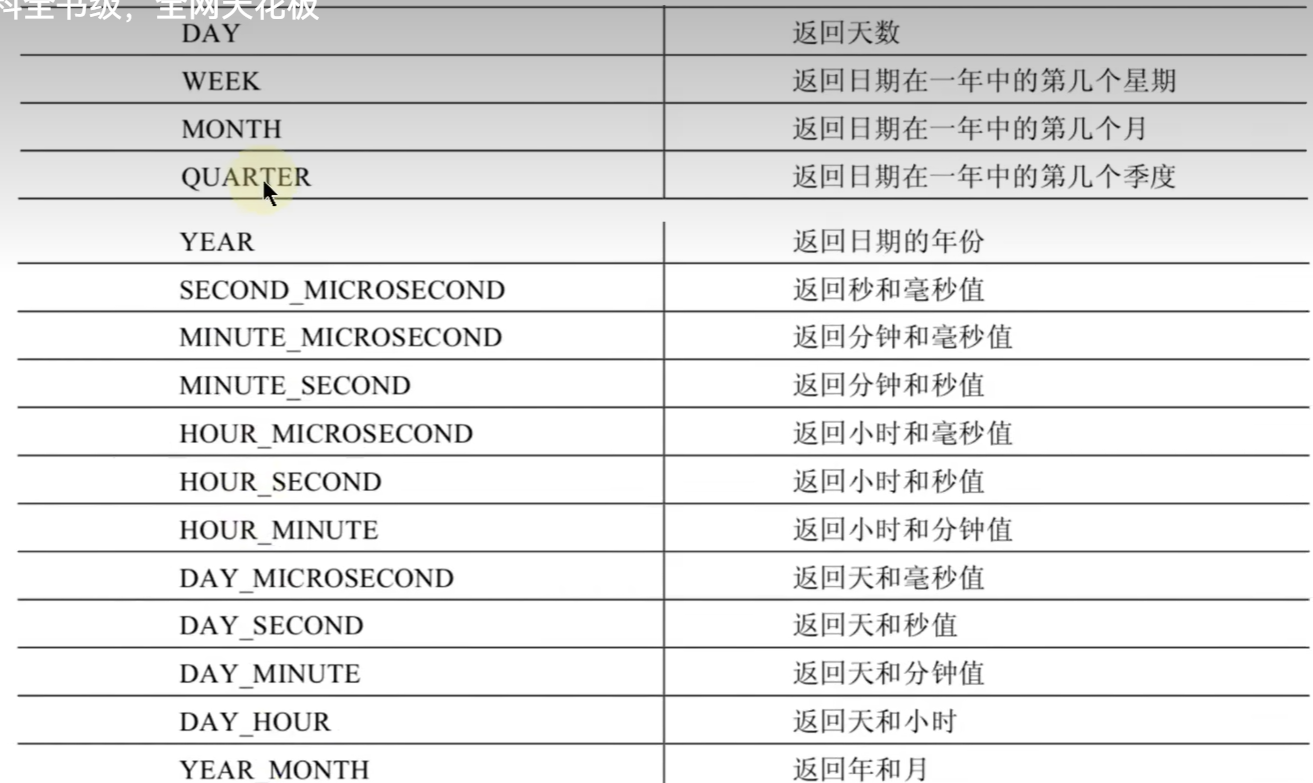
-- utc\_date/utc\_time:世界标准时间、日期，即英国时间、日期

SELECT CURDATE(),CURTIME(),NOW(),

UTC\_DATE(),UTC\_TIME()

FROM DUAL;







-- 时间和时间戳的转换

SELECT UNIX\_TIMESTAMP(),FROM\_UNIXTIME(1728266964),

UNIX\_TIMESTAMP("2021-11-17 10:12:32")

FROM DUAL;

-- 获取月份、星期、星期数、天数等函数

SELECT YEAR(CURTIME()),MONTH(CURTIME()),WEEK(CURTIME()),

DAY(CURTIME()),HOUR(NOW()),MINUTE(NOW()),SECOND(NOW())

FROM DUAL;

SELECT MONTHNAME(CURTIME()),DAYNAME(CURTIME()),WEEKDAY(CURTIME()),

QUARTER(CURTIME()),DAYOFYEAR(CURTIME()),DAYOFMONTH(CURTIME()),

DAYOFWEEK(CURTIME())

FROM DUAL;

-- 日期的操作函数

SELECT EXTRACT(YEAR\_MONTH FROM CURTIME())

FROM DUAL;

-- 时间与秒钟进行转化

SELECT TIME\_TO\_SEC(CURTIME()),SEC\_TO\_TIME(37860)

FROM DUAL

-- 计算日期和时间的函数（重点）

SELECT CURTIME(),DATE\_ADD(NOW(),INTERVAL 1 YEAR),

DATE\_SUB(CURTIME(),INTERVAL '1\_1' YEAR\_MONTH)

FROM DUAL;

-- 日期和时间间隔（重点）

-- addtime:添加的时间默认单位为秒

-- timediff:两段时间相差多少小时

SELECT ADDTIME(NOW(),30),DATEDIFF(NOW(),"2021-11-17"),

TIMEDIFF(NOW(),DATE\_SUB(NOW(),INTERVAL 2 MONTH))

FROM DUAL;

-- FROM\_DAYS():返回从0000-01-01开始后多少天后的日期

-- 注：不能小于366，因为在负数或0到365之间的整数，不被认为是一个有效数

SELECT FROM\_DAYS(354)

FROM DUAL;

-- TO\_DAYS(x)：返回x距离0000-01-01多长时间

SELECT TO\_DAYS('0000-1-25')

FROM DUAL;

-- LAST\_DAY(x)：返回当前时间所在月份的最后一天的日期

-- MAKEDATE(year,dayofyear)：使用年份和天数，返回日期显示

-- maketime(hour,minute,second):组合时间

-- period\_add(time,n)：返回time加上n秒后的时间,时间格式是没有‘-’的

SELECT LAST\_DAY(NOW()),PERIOD\_ADD(20240101010101,20)

FROM DUAL;

-- 日期的格式化与解析

-- 格式化：日期-->字符串

-- 解析：字符串-->日期

-- 方法：（1）DATE\_FORMAT(date,format):日期格式化

-- （2）TIME\_FORMAT(time,format):时间格式化

-- （3）STR\_TO\_DATE(str,format):字符串解析

SELECT DATE\_FORMAT(NOW(),'%Y-%M-%D'),DATE\_FORMAT(NOW(),'%Y-%m-%d'),

TIME\_FORMAT(CURTIME(),'%h:%i:%s'),

STR\_TO\_DATE('2024-10-07','%Y-%m-%d')

FROM DUAL;

-- 流程控制函数

-- if(value,value1,value2)：value为真，返回value1，否则返回value2

-- IFNULL(expr1,expr2):如果expr1为空，则返回expr2

-- 注：0不算null

SELECT if(15<10,15,25),IFNULL(0,15)

FROM DUAL;

-- case when then:类似java中的else if（）{}，最后的end是显示的标题

SELECT CASE WHEN 3>PI() THEN '不合理'

WHEN 4>PI() THEN '合理'

ELSE '无法分析' END 'if判断'

FROM DUAL;

如果case和when中间没有东西，相当于java中的if……else

有东西，相当于Switch case

-- 加密和解密函数

-- 种类

-- password（str）：41长的字符串，加密的结果是不可逆的，用于用户密码的保存

-- 8.0版本不支持

-- MD5（str）：一种加密方式，如果为null，放回null

-- SHA（str）：比MD5更加安全

-- 注意：上边的三种都是不可逆的加密，是不能再解密成明文

SELECT MD5('你好，世界！'),SHA('Hello World')

FROM DUAL;

-- ENCODE(str,pass\_str)：值和密钥

-- DECODE(crypt\_str,pass\_str)：值和密钥

-- 注：这两个在8.0也不支持了

SELECT ENCODE('你好',123456)

FROM DUAL;

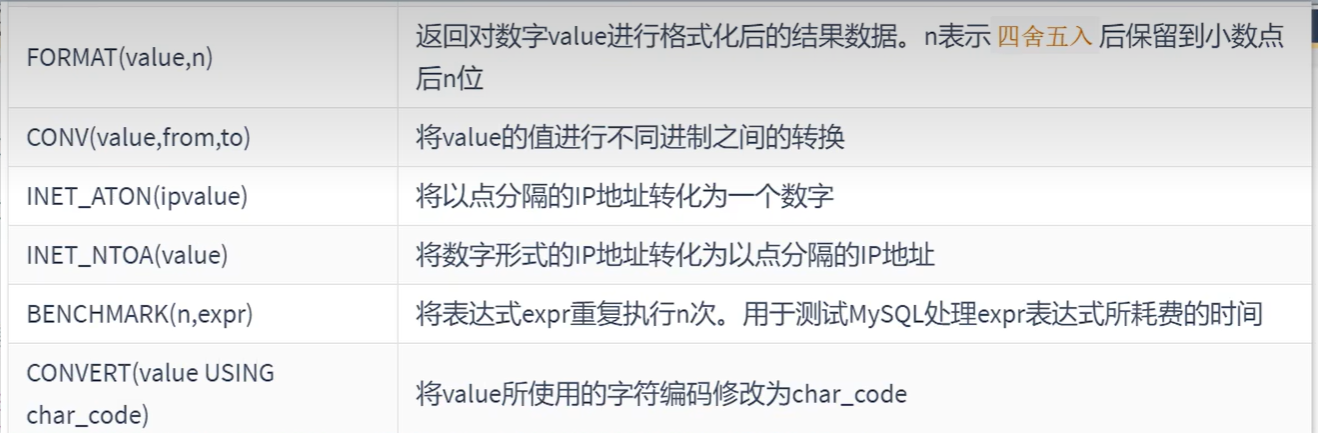
表格

描述已自动生成

-- mysql的信息函数

-- 在其他平台进行开发的时候要和mysql进行连接

其它函数：



-- 约束

-- 约束的作用范围：

-- （1）列级约束（2）表级约束：在所有列级声明完，再声明

-- 自增列

-- 限制:只能够在主键或唯一键中声明,数据类型必须为整型

-- 注:添加数据的时候,不需要再声明自增列,也可以,当没有必要

-- 删除和添加：使用modify进行修改的时候，添加和不添加

-- auto\_increment;

-- 注：在mysql中，如果进行删除数据，并不会影响自增列的自增

-- 例如，1,2,3；此时删掉3，如果再进行自增，4会占据3的位置

-- 外键约束（foreign key）

-- 作用：确保对于一个表中的数据进行增删改的时候，另一张表存在这个属性

-- 注：增删改的时候，要对主表操作，而不是从表

-- 子表的外键和主表对应的位置的名字可以不同，但是数据类型必须保持一致

-- 在创建表的时候进行添加

-- 先创建主表

CREATE TABLE arr1(

a INT PRIMARY KEY AUTO\_INCREMENT UNIQUE,

b VARCHAR(10));

-- 从表的主键也可以作为外键

CREATE TABLE arr2(

c INT PRIMARY KEY AUTO\_INCREMENT,

d VARCHAR(20),

CONSTRAINT fk\_arr1\_a FOREIGN KEY (c) REFERENCES arr1(a)

);

-- 在创建好后添加外键

ADD CONSTRAINT fk\_arr1\_a

FOREIGN KEY (c) REFERENCES arr1(a)

-- 约束等级

-- cascade：主表更新update或delete时，同步更新从表

-- SET NULL:主表更新update或delete时，从表外键改为null

-- 注：外键不能为not null

-- no action:如果从表有匹配的数据，则不允许对主表对应的键进行update或delete

-- restrict:和action效果一致

-- 存储过程与函数

-- 作用：在5.0后，mysql支持使用存储过程与函数将复杂的SQL

-- 逻辑封装起来，应用程序可以直接进行调用

-- 区别：函数必须有返回值，存储过程可以没有

-- 存储过程

-- 与视图的区别：视图是一张虚拟表，不能对于数据进行修改

-- 而其是程序化的SQL语句，可以进行更复杂操作

-- 语法：create procedure 存储过程名（in|out|inout,参数名）

-- [not] determinstic 执行结果是否确定，不写默认为这个，确定返回值不写not

-- begin

-- 存储过程体

-- end

-- in:传入参数，不标明默认为in

-- out：输出参数

-- inout:即使传入参数，也是传出参数

-- 缺点：里面一般非常复杂，所以当其中的某行出现问题，排查起来非常难

-- 可移植性差 注意：mysql默认程序的结束符号为分号，在过程体内部如果带分号，导致无法全部执行，因此，可以用delimiter 符号，来改变执行符号，用delimiter结束

-- 一般为，//

-- 创建存储过程

-- （1）类型一：不带参数

DELIMITER //

CREATE PROCEDURE FindCountry()

BEGIN

SELECT \* FROM country;

END //

DEIMITER;

-- 调用存储过程

CALL FindCountry();

CALL FindCity();

-- （2）类型二：带参数

-- 带输出参数的

delimiter //

CREATE PROCEDURE findMostPopulation(OUT Max INT)

DETERMINISTIC

BEGIN

SELECT MAX(Population) INTO Max

FROM country;

END //

delimiter;

-- 这里的@是必须带的，表示用户定义的变量，可以随便起名字，最好和原来一致

CALL findMostPopulation(@MAX);

SELECT @MAX;

delimiter //

-- 带输入参数的

CREATE PROCEDURE FindOneCountry(IN o\_country VARCHAR(52))

BEGIN

SELECT country.Capital FROM country WHERE country.`Name`=o\_country;

END //

delimiter;

-- 方式一：直接在参数中创建

CALL FindOneCountry('Egypt');

-- 方式二：在外部创建变量，然后放入参数

SET @Capital :='Egypt';

CALL FindOneCountry(@Capital);

-- 带in和out

delimiter //

CREATE PROCEDURE FindPopulation(IN o\_country VARCHAR(52),OUT Population INT)

BEGIN

SELECT country.Population INTO Population

FROM country WHERE country.`Name`=o\_country;

END //

delimiter;

CALL FindPopulation('Egypt',@Population);

SELECT @Population;

-- 使用inout

delimiter //

CREATE PROCEDURE FindWantCountry(INOUT o\_country VARCHAR(52))

BEGIN

SELECT country.Capital INTO o\_country

FROM country WHERE country.`Name`=o\_country;

END //

delimiter;

SET @outcome:='Egypt';

CALL FindWantCountry(@outcome);

SELECT @outcome;

-- 存储函数(自定义)

-- 规则：参数只有in类型的，默认为in

-- 返回值只能有一个

-- 必须有返回值

-- 创建存储函数

DELIMITER //

CREATE FUNCTION cityPopulation(id INT)

RETURNS INT

DETERMINISTIC

BEGIN

RETURN(SELECT city.Population FROM city WHERE city.ID=id);

END //

DELIMITER;

-- 调用存储函数

SELECT cityPopulation();

-- 注意：

/\* 由于必然有返回值，因此必须明确参数信息

至少要声明返回值是否确定

或者，set global log\_bin\_trust\_function\_creators=1

\*/

SET GLOBAL log\_bin\_trust\_function\_creators=1;

-- 错误的例子，不能够直接将常数放到存储函数中

-- SET @q='Tilburg';

-- CELIMITER //

-- CREATE FUNCTION('Tilburg')

-- RETURNS INT

-- BEGIN

-- RETURN(SELECT city.ID FROM city WHERE city.`Name`=@q);

-- END//

-- DELIMITER;

-- 查看存储过程和存储函数的创建信息

-- 注意：括号和参数不用带

SHOW CREATE PROCEDURE DisPlay;

SHOW CREATE FUNCTION cityPopulation;

-- 查看状态信息(所有数据库)

SHOW FUNCTION STATUS;

-- 查看特定的

-- 注意：like后面只能跟存储函数名或存储过程名

SHOW FUNCTION STATUS LIKE 'cityPopulation';

-- 存储过程和存储函数的修改

-- 注意：修改的只是特性，例如，deterministic

-- 语法：使用alter function/procedure 名 修改操作

-- 存储过程与存储函数

-- 优点：存储一次多次使用，减少开发工作量，安全性强，减少网络传输，良好的封装性

-- 缺点：可移植性差，调试困难，版本管理难，不适合高并发场景

-- 变量

-- 种类：系统变量、自定义变量

-- 系统变量

-- global（全局系统变量）、session（会话系统变量）

-- 称谓：全局系统变量称为全局变量，会话系统变量称为local变量

-- 注：如果不写，默认为会话级别

-- 全局变量可以在所有会话连接中有效，但是不能够跨重启

-- 1、查看全部全局变量和会话变量

SHOW GLOBAL VARIABLES;

SHOW SESSION VARIABLES;

-- 查看部分的全局和会话变量

SHOW GLOBAL VARIABLES LIKE '%z%';

-- 2、查看指定的全局或会话变量

-- 注：@@变量名 表示的是系统变量，后边的变量名如果没有标明是

-- global还是session，则会先找会话，再找全局

SELECT @@global.admin\_address;

-- 或者

SELECT @@admin\_address;

-- 修改系统变量的值

-- 作用：可以控制与客户端的连接数量

-- 方式一：直接修改配置文件，但是需要重启数据库（不推荐）

-- 方式二：使用set方法进行修改

-- 注：如果是只读变量，是不可以修改的

SET @@global.admin\_port =33061;

SET @@global.admin\_address=4624;

SET @@admin\_address=4624;

SET GLOBAL admin\_address=4624;

-- 自定义变量（用户变量）

-- 种类：会话用户变量和局部变量

-- 局部变量：在Begin和End之间是有效的

-- 会话变量：@开头，作用于当前会话

-- 创建会话变量

-- 方式一：

SET @m1=1;

SET @m2:=1;

SET @m3:=@m1+@m2;

-- 注：起别名不能用数字

SELECT @m3 AS "第三个";

-- 方式二：

SELECT @count:=COUNT(\*) FROM city;

SELECT @count;

SELECT COUNT(country.`Name`) INTO @count2 FROM country;

SELECT @count2;

-- 创建局部变量

-- 注意：（1）必须在begin和end之间，（2）使用declare前缀

-- （3）declare的方式声明必须放在begin和end中间的首行位置

-- （4）局部变量需要声明类型

DELIMITER //

CREATE PROCEDURE test()

BEGIN

-- 声明局部变量

DECLARE a INT DEFAULT(15);

DECLARE b INT;

DECLARE c INT;

-- 给局部变量赋值

SET a=11;

SET b=25;

SELECT city.Population INTO c FROM city WHERE city.ID=10;

END//

DELIMITER;

-- 调用

CALL test();

-- 索引的数据结构

-- 作用：快速找到数据的一种数据结构，减少磁盘I/O的次数，提升效率

-- 类比：相当于书的目录

-- 磁盘和内存读取I/O的区别：先从磁盘中读取，放到内存中，然后再从内存进行读取，因此速度主要受到磁盘读取速度的影响

-- 索引的本质：可以理解为排好序的快速查找

-- 区别：不同存储引擎的索引不一定完全相等，也不一定支持所有的索引类型，引擎可以定义最大索引数和最大索引长度，每个索引

-- 最少支持每个表的16个索引,总索引长度至少为256字节

-- 注意:添加了唯一约束的的字段,自动会创建一个唯一索引

-- 优点:可以提高表与表之间的连接速度,尤其是子表和父表之间

-- 提高了排序和分组的速度,降低CPU消耗

-- 缺点:创建和维护索引需要消耗时间,占用磁盘空间,增加更新表消耗的时间

-- 在一个页面中查找（16KB）

/\* 如果数据量比较少,仅在一页中,则有以下两种情况

(1)利用主键为搜索条件:可以使用二分法快速进行查找

(2)非主键查找:只能从最小的进行依次遍历每个链表

\*/

-- 在多页中进行查找:确定页数,遵循一页中的方法

-- 索引的简单构成

/\*

日程表

描述已自动生成

record\_type：这个表示记录的类型，0是普通，1是目录项（用于表示此为目录记录），2是最小，3是最大

next\_record：表示记录的下一条记录

图表, 图示, 箱线图

描述已自动生成

注意：在表中，必须保证主键值应该大于上一条记录，因此如果出现后边的主键大于前边的要进行移动，叫做页的移动

由于可能有多个页，下一个页的主键值也应该大于上一个页的主键值

目录项：存在多个页面的时候，为了提升查找的速度，因此可以提取每一个页的最小值，作为key，加上页号；组成一个目录项

\*/

-- 索引的迭代

-- 目录页：目录也使用链表存储，因此也可以创建一个目录组成的页

-- 目录项记录和用户记录的相同点：

/\*

两者用的是相同的数据页，都会为主键值生成Page directory（页目录）

\*/

-- 迭代：会不断进行大目录套小目录

图示

描述已自动生成

-- B+树

/\*

原理：叶子结点存的是真实的数据，数据之间使用单项链表进行相连，

每个叶子结点之间使用双向链表进行相连，每一层都是这样进行连接

大小：一般不会超过4层，因为

假设，一个用户记录能存100条数据，1层是100条，而两层时，上边的目录向记录是1000条，则能存放100000条，以此类推

能存非常大的记录。且树层数越少，查找速度越快

\*/

-- 聚簇索引

-- 分类：索引分为聚簇索引和非聚簇索引，区别在于有没有使用主键作为索引条件

-- 本质：不是索引类型，而是一种数据存储形式

-- 构成：上边讲的这种B+树就是一种聚簇索引

-- 注意：这种形式的索引不需要我们显示的去使用index语句去进行创建，引擎会自动帮我们创建好聚簇索引

-- MySQL中一个表一般只有一个聚簇索引

-- 如果没有定义主键，引擎会选择唯一的非空索引代替，如果没有引擎就隐式的定义一个主键

-- 缺点：过于依赖插入的顺序，按照主键的顺序插入是最好的方式，否则会出现页分裂，严重影响性能

-- 更新主键的成本过高，一般主键不会进行更新

-- 二级索引访问需要两次索引查找，第一次找到主键值，第二次找到对应的行

-- 二级索引、辅助索引

-- 如果想要以其他索引作为条件，则可以再创建一个B+树

-- 二级索引的叶子结点：存储的不是真实的数据，而是构建二级索引的字段+主键的值

-- 回表：如果在二级索引中找不到完整的数据，会回到聚簇索引中寻找

-- 优点：进行增删改查的时候要比聚簇索引快

-- 联合索引

-- 特点：例如c2和c3，先试用c2进行排序，相同时再考虑c3

-- 注：在创建目录页的时候，应该把两个字段都带上

-- InnoDB的B+索引的注意事项

-- （1）根页面的位置万年不动

-- 表中B+树的创建过程：

/\*

首先，当数据不多的时候，只有一个页的是后，这一个表就相当于一个叶子节点

然后，当数据较多的时候， 不是直接创建一个新的表，而是在想拷贝一份到一张新的表中，

原表中，删除所有数据，充当内节点，然后再创建新的数据页，存放超出的数据，然后将进行移项。

最后，将每个叶子页面中的最小点和页面编号，存放到已经变为内节点的原表中

因此，不难发现，第一个页一直作为表中B+树的根节点

\*/

-- （2）内节点中目录项记录的唯一性

-- 作用：如果当插入数据的时候，如果发现目录项中，有两个信息是相同的，此时，会产生争议，

-- 到底插入到哪边。

-- 思路：在目录项的对应位置插入主键，因为主键是唯一的，此时可以进行区分

-- 一个页面至少存2条数据