[SQL基础 2](#_Toc271632201)

[**3范式(NF)** 2](#_Toc271632202)

[分隔符 3](#_Toc271632203)

[SQL语句分类 3](#_Toc271632204)

[常量 4](#_Toc271632205)

[变量 4](#_Toc271632206)

[运算符 5](#_Toc271632207)

[数据类型 5](#_Toc271632208)

[表达式 8](#_Toc271632209)

[内置函数 8](#_Toc271632210)

[聚合函数(多行函数 / 分组函数) 8](#_Toc271632211)

[单行函数 8](#_Toc271632212)

[日期时间函数 9](#_Toc271632213)

[数学(算术)函数 10](#_Toc271632214)

[字符串函数 12](#_Toc271632215)

[转换函数 14](#_Toc271632216)

[其它重要的系统函数 18](#_Toc271632217)

[数据库 19](#_Toc271632218)

[创建 数据库 19](#_Toc271632219)

[修改 数据库 19](#_Toc271632220)

[删除 数据库 19](#_Toc271632221)

[数据定义语言 20](#_Toc271632222)

[表 20](#_Toc271632223)

[创建 表 21](#_Toc271632224)

[修改 表 22](#_Toc271632225)

[删除 表 23](#_Toc271632226)

[重命名 表 23](#_Toc271632227)

[查看表结构 23](#_Toc271632228)

[索引 23](#_Toc271632229)

[视图 27](#_Toc271632230)

[存储过程 28](#_Toc271632231)

[触发器 28](#_Toc271632232)

[模式 28](#_Toc271632233)

[域 28](#_Toc271632234)

[建立联合 28](#_Toc271632235)

[数据操作语言 29](#_Toc271632236)

[记录 29](#_Toc271632237)

[插入 记录 29](#_Toc271632238)

[修改 记录 30](#_Toc271632239)

[删除 记录 30](#_Toc271632240)

[查询 30](#_Toc271632241)

[一般查询 30](#_Toc271632242)

[过滤数据 31](#_Toc271632243)

[多表连接查询 34](#_Toc271632244)

[子查询 40](#_Toc271632245)

[实例 44](#_Toc271632246)

[连接符 46](#_Toc271632247)

[数值运算 46](#_Toc271632248)

[函数 46](#_Toc271632249)

[数据完整性 46](#_Toc271632250)

[事务控制 47](#_Toc271632251)

[并发处理 48](#_Toc271632252)

[安全性控制 48](#_Toc271632253)

[SQL安全模式 48](#_Toc271632254)

[角色管理 48](#_Toc271632255)

[权限管理 48](#_Toc271632256)

[SQL Server安全管理 48](#_Toc271632257)

[Oracle安全管理 48](#_Toc271632258)

[嵌入式SQL 48](#_Toc271632259)

[SQL游标 48](#_Toc271632260)

[SQL优化 49](#_Toc271632261)

[实例 49](#_Toc271632262)

[Sql Server,Oracle和MySQL的分页语句 49](#_Toc271632263)

[**返回前10条数据** 50](#_Toc271632264)

[使用JDBC批量更新 50](#_Toc271632265)

[MySQL / SQL Server / Oracle 特性 50](#_Toc271632266)

[MySQL特性 50](#_Toc271632267)

[SQL Server特性 50](#_Toc271632268)

[Oracle特性 50](#_Toc271632269)

[启动 和 停止SQL Server命令 50](#_Toc271632270)

[My SQL / SQL Server / Oracle 创建表 51](#_Toc271632271)

[--1群类型树表MicBlogGroupTypeTree 51](#_Toc271632272)

# SQL基础

**3范式(NF)**

|  |  |
| --- | --- |
| 第一范式 | 每个关系r的属性值都是不可分的原子值 |
| 第二范式 | 消除非主属性对主键的部分函数依赖. |
| 第三范式 | 不存在"非主属性"对主键的传递函数依赖. |

## 分隔符

|  |  |  |
| --- | --- | --- |
| 单词或花括号 | 分隔符 | 可选个数 |
| , | >=1 |
| | | =1 |
| 方括号 | | | <=1 |
| , | >=0 |

## SQL语句分类

|  |  |  |
| --- | --- | --- |
| 模块 | 语句 | 功能 |
| 数据定义语句DDL | create table | 创建 表 |
| drop table | 删除 表 |
| alert table | 修改 表 |
| rename |  |
| create view | 创建 视图 |
| drop view | 删除 视图 |
| create index | 创建 索引 |
| drop index | 删除 索引 |
| create procedure | 创建 存储过程 |
| drop procedure | 删除 存储过程 |
| create trigger | 创建 触发器 |
| drop trigger | 删除 触发器 |
| create schema | 添加 模式 |
| drop schema | 删除 模式 |
| create domain | 创建 域 |
| alter domain | 修改 域 |
| drop domain | 删除 域 |
| truncate | 截断表的内容, 比如切断一列 |
| comments |  |
| 数据查询 | select | 检索数据 |
| 数据操纵语句DML | insert | 插入 记录 |
| delete | 删除 记录 |
| update | 更新 记录 |
| merge | 表的合并 |
| 数据控制语句DCL | grant | 授权用户访问权限 |
| deny | 拒绝用于访问 |
| revoke | 解除用户访问权限（收回权限） |
| 事务控制语句 | commit | 结束当前事务 |
| rollback | 终止当前事务 |
| avepoint | 存储点 |
| set transaction | 定义当前事务数据访问特征 |
| 程序化SQL | declare | 为查询设定游标 |
| explan | 为查询描述数据访问计划 |
| open | 检索查询结果, 打开一个游标 |
| fetch | 检错一行查询结果 |
| close | 关闭游标 |
| prepare | 为动态执行准备SQL |
| execute | 动态地执行SQL |
| describe | 描述准备好的查询 |

## 常量

|  |  |
| --- | --- |
| 数字常量 | 整数和浮点数都可以作为常量使用, 常量前面可以加正号或负号.  整型常量, 由没有引号括起来且不含小数点的一串数字表示.  浮点型常量, 通常规定使用符号E, 其中E代表"乘以10的几次方", 如6.24E5, +0.25524E, 1.05E-7 |
| 字符串常量 | 由单引号括起来的一串字符, 可以包含: 大小写字母, 数字, 特殊符号(如!, @, #等)  如果字符串常量中包含单引号, 则需要在其前面再加上一个单引号, 以表示起是常量中的一个字符. 如: 'I don''t konw' |
| 日期 时间 常量 | 用单引号括起来. 例如:   |  |  | | --- | --- | | 日期常量 | 'April 15, 2006' | | 时间常量 | '14:30:24' |   日期转换函数 to\_date('April 15 2006', MONTH DD YYYY) |
| 符号常量 | current\_date /\*此符号常量能生成当前日期的值\*/  current\_time  user  system\_user  session\_user等 |

## 变量

|  |  |  |
| --- | --- | --- |
|  | 作用域 | 注意事项 |
| 用户自定义的局部变量 | 仅限制在程序内部 |  |
| 系统提供的全局变量 | 任何程序均可调用 | 不是用户定义的, 是服务器级定义的.  用于只能使用预先定义的全局变量  应用全局变量时, 必须以标记"@@"开头  局部变量的名称不能与全局变量的名称相同, 否则会在应用程序中出现不可预测的结果. |

## 运算符

|  |  |
| --- | --- |
| 算术运算 |  |
| 赋值运算 |  |
| 位运算符 |  |
| 比较运算符 | |  |  | | --- | --- | | > |  | | < |  | | = |  | | >= |  | | <= |  | | <> |  | |
| 逻辑运算符 | |  |  | | --- | --- | | asd |  | | or |  | | not |  | |
| 几乎喘串联运算符 |  |

运算符优先级

|  |  |
| --- | --- |
| 优先级 | 运算 |
| 1 | \* / |
| 2 | + - |
| 3 | || |
| 4 | = > >= < <= <> |
| 5 | is [not] null like [not] in |
| 6 | [not] between ... and |
| 7 | not |
| 8 | and |
| 9 | or |

## 数据类型

|  |  |  |
| --- | --- | --- |
| 数字数据类型 | 整数 | integer  int  number等 |
| 浮点数 | 精确的浮点数, 可以定义他们的精度(小数点前后数字的总个数, 必须小于32, 缺省是5)和位数(小数点后数字的个数, 缺省为0, 表示该数为整数)decimal, numeric  一般的浮点数float, real等 |
| 字符串数据类型 | 定长字符串(char) | 1<=长度<=254, (在排序或处理字符时, 效率远远高于可变长度的字符串变量) |
| 变长字符串(varchar) | 不同DB系统中有很大区别 (许多DBMS不允许为可变长度字符变量创建索引, 而只允许对固定长度字符变量创建索引.) |
| 大型文本 | 通常称为load 或 text |
| 二进制字符串类型 |  |
| 图形字符串类型 |  |
| 日期 时间 数据类型 | date  time  timestamp(时间戳, 日期和时间的组合) | YYYY-MM-DD  HH:MM:SS.nn...  YYYY-MM-DD HH:MM:SS.nn... |
| 自定义 数据类型 |  |  |
| 特殊数据类型 |  |  |

|  |  |  |
| --- | --- | --- |
| 归类 | 数据类型 | 说明 |
| 字符型 | Char(size) |  |
| Varchar(size) |  |
| NChar(size) | Char类型的拓展, 支持多字节和Unicode字符 |
| NVarchar(size) | Varchar类型的拓展, 支持多字节和Unicode字符 |
| 数字型 | Bit |  |
| Int 或 Interger |  |
| Real |  |
| Double Precision |  |
| Float |  |
| SmallInt |  |
| TinyInt |  |
| Number(Decimal 或 Numeric) |  |
| 特殊数字类型(货币) | money |  |
| smallmoney |  |
| 日期 时间 型 | data |  |
| time |  |
| timestamp |  |
| time with time zone |  |
| timestamp with time zone |  |
| 二进制 数据类型 | binary |  |
| varbinary |  |
| raw |  |
| varraw |  |
| 文本 和 图形 数据类型 | text |  |
| ntext |  |
| image |  |
| 自定义 数据类型 |  |  |

日期输入格式

时间输入格式

**Oracle中的数据类型**

|  |
| --- |
| blog替代raw  clob替代long |

## 表达式

|  |  |
| --- | --- |
| 算术运算符 |  |
| 逻辑运算符 | true, false, unknown |

## 内置函数

### 聚合函数(多行函数 / 分组函数)

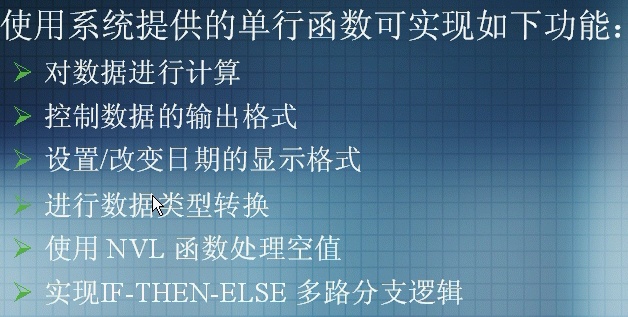
对一组数据进行运算, 一组数据只返回一个结果, 也称多行函数.

分组函数最多可嵌套两层.

|  |  |
| --- | --- |
| count(\*) | 返回组中总记录数目 |
| count(exp) | 返回表达式exp值非空的记录数目 |
| count(distinct(exp)) | 返回表达式exp值不重复的, 非空的记录数目 |
|  |  |

|  |  |
| --- | --- |
| 给予多字段分组 | group by 字段1, 字段2; --先对字段1进行分组, 再对字段2进行分组. |

## 单行函数



|  |  |
| --- | --- |
| avg() | 返回组中的平均值，空值将被忽略。 |
| count() | 返回的行数, 或者是非空值的行数. eg: select count(\*) from user where name like '%罗%' |
| max() | 返回表达式的最大值 |
| min() | 返回表达式的最小值 |
| sum() | 返回表达式中所有值的的和，或只返回DISTINCT值。SUM只能用数字列。 |
| stdev() | 返回给定表达式中所有值的统计标准偏差。 |
| stdevp() | 返回给定表达式中所有值的填充统计标准偏差。 |
| var() | 返回给定表达式中所有值的统计方差。 |
| varp() | 返回给定表达式中所有值的填充的统计方差 |
| binaby\_checksunm) | 可用于检测表中行的更改返回值由表达式的运算结果类型决定 |
| checksum() | 返回在表的行上或在表达式上计算的校验值 CHECKSUM 用于生成哈希索引 |
| checksum\_agg() | 返回组中值的校验值。空值冽被忽略。 |
| count\_big() | 返回组中项目的数量。在使用上和count 基本上是一样的，只是在返回值上有一点区别，count\_big 返回是bigint的数据类型值，count返回的是int数据类型值。 |
| grouping() | 返回一个聚合函数，它产生一个附加列，当用CUBE或ROLLUP运算符添加行时，附加 的列输出值为1，当所添加的行不是由CUBE或ROLLUP产生时，附加列值为0 |
| 备注 | 只有在where子句中使用子查询的时候, 才可以在where子句中使用聚合函数. |

### 日期时间函数

|  |  |
| --- | --- |
| dateadd | 在向指定日期加上一段时间的基础上返回新datetime值。 |
| datediff | 返回跨两个指定日期和时间边界数。 |
| datename | 返回代表指定日期的指定日期部分的字符串。 |
| datepart | 返回代表指定日期的指定日期部分的整数。 |
| day month year | 返回指定日期的天 月 年 的日期部分的整数 |
| getdate | 按datetime值的标准内部格式返回当前系统时间和日期 |
| month | 作为整数返回日期表达式关联的月份. |
| year | 作为整数返回与日期表达式关联的年份. |
|  | 可以直接加减一个数值(其实就是多少天), 结果仍为日期型.  两个日期型数据可以相减, 结果为二者相差多少天. |

### 数学(算术)函数

|  |  |
| --- | --- |
| abs | 返回给定数字的绝对值 |
| acos | 返回以弧度表示的角度值，该角度值的余弦为给定的float表达式，也叫反余弦 |
| asin | 返回以弧度表示的角度值，也叫反正弦。 |
| atan | 返回以弧度表示的角度值，该角度值的正切为给定的float表达式，也叫反正切。 |
| atn2 | 返回以弧度表示的角度值，该角度值的正切介于两个给定的float表达式之间 |
| ceiling | 返回或等于所给数字表达式的最小整数。 |
| cos | 返回给定表达式中给定角度的三角余弦值 |
| cot | 返回给定float表达式指定角度的三角余切值 |
| degrees | 当给出弧度为单位的角度时，返回相应的以度数为单位的角度。 |
| exp | 返回所给的float表达式的指数值。 |
| floor | 返回小于或等于所给数字表达式的最大整数。 |
| log | 返回给定float表达式的自然对数。 |
| log10 | 返回给定float表达式的以10为底的对数。 |
| PI | 返回PI的常量值。 |
| power | 返回给定表达式乘指定次方的值。 |
| radians | 对于数字表达式中输入的度数值返回弧度值。 |
| rand | 返回0到1之间的随机float值。 |
| round | 返回数字表达式并四舍五入为指定的长度或精度。 |
| sign | 返回给定表达式的正 零 或负号 |
| sin | 以近似数字表达式返回给定角度的三角正弦值。 |
| square | 返回给定表达式的平方值。 |
| sqrt | 返回给定表达式的平方根。 |
| tan | 返回给定表达式的正切。 |
|  | 参数可以使字符串常量, 也可以是变量(也就是列名). |

### 字符串函数

|  |  |
| --- | --- |
| ascii | 返回字符串表达式中最左边的字符(第一个字符)的ASCII代码值 |
| char | 返回参数提供的ASCII代码的字符 |
| charindex | 返回指定字符串的起始位置, 然后在将要被搜索的字符串中进行搜索. |
| difference | 计算两个字符串的soundex值, 并返回这两者之间的差值. |
| left | 返回第一个参数中从左开始指定数目的字符, 这个数目是由第二个参数指定的. |
| len | 返回通过参数指定的字符串中的字符数. |
| lower | 返回参数指定的字符串, 将所有大写字符转换成小写字符. |
| ltrim | 返回删除了前导空格的由参数所指定的字符串. |
| nchar | 返回unicode字符, 其代码通过参数指定. |
| patindex | 在该函数中, 第一个参数各处支付模式, 本函数返回在第二个参数指定的字符串中首次出现支付模式的起始位置. |
| quotename | 给unicode字符串的开始和末尾加引号, 使其成为一个有效的SQL引号分界的标识符. |
| replace | 在第一个参数指定的字符串中, 搜索第二个参数指定的模式, 并使用第三个参数指定的字符来替代该模式. |
| replicate | 返回重复第二个参数指定的次数的, 第一个参数给定的字符串. |
| reverse | 返回该参数指定的字符串反转值. |
| right | 在第一个参数所指定的字符串中, 从右边开始返回第二个参数指定的字符数. |
| rtrim | 返回该参数指定的删除了所有尾随空格的给定字符串. |
| soundex | 当你查找Smith姓名, 并希望定位所有的拼写变体时, soundex用于衡量两个字符串是否具有相似性. |
| space | 返回具有多个空格字符的字符串, 空格的数目由参数指定. |
| str | 返回浮点数, 把它作为表示数字的字符串. |
| stuff | 通过删除在给定的起始点上的指定字符数, 然后在给定的起始点上插入指定的字符集, 来对字符串进行修改. |
| substr | 返回由起始位置和长度参数指定的字符串, 文本, 二进制活着是图像表达式的一部分. |
| unicode | 返回字符串表达式中的一个字符, 即最左边的字符的unicode代码. |
| upper | 返回参数指定的字符串, 把字符串的小写字符转换为大写字符. |

**Oracle中字符串函数**

|  |
| --- |
| 参数可以使字符串常量, 也可以是变量(也就是列名). |

### 转换函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 转换函数  没有改变原来数据类型, 只是复制了一份来做转换. | 日期-->字符串   |  |  | | --- | --- | | to\_char(date) | 默认显示格式 | | to\_char(date, 'format\_model') | to\_char(sysdate, 'yyyy/mm/dd hh24:mi:ss') 24小时制  to\_char(sysdate, 'yyyy/mm/dd hh:mi:ss am') 上下午制度  to\_char(sysdate, '"公元"yyyy"年"mm"月"dd"日" hh:mi:ss am')  to\_char(sysdate, 'fm yyyy/mm/dd hh:mi:ss am') 上下午制度 |   常用日期格式    字符串-->日期  to\_date(char)  to\_date(char, 'format\_model')  数字-->字符串   |  |  | | --- | --- | | to\_char(number) |  | | to\_char(number, 'format\_model') | to\_char(123456.78901, '$999,999.000') |     字符串-->数字   |  |  | | --- | --- | | to\_number (char) |  | | to\_number(char, 'format\_model') | to\_char('$123,456.78901', '$999,999.000') | |
| 通用函数(使用与任何类型数据) | |  |  | | --- | --- | | nvl(exp1, exp2) | 如果exp1的值为null, 则返回exp2的值, 否则返回exp1的值. 常用语将空值null替换成指定的缺省值. | | nvl2(exp1, exp2, exp3) | 如果exp1的值不为null, 则返回exp2的值, 否则返回exp3的值. | | nullif(exp1, exp2) | 如果exp1==exp2则返回null, 否则返回exp1的值. | | coalesce(exp1, exp2, ...) | 依次考察各参数表达式, 遇到非null值即停止并返回该值. (一般最后放一个非空值做缺省值) | | case表达式 |  | | decode() |  | |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

### 其它重要的系统函数

|  |  |
| --- | --- |
| case | 检查条件列表, 并返回条件是真的情况下的值. |
| cast | 把参数转换成指定的数据类型. |
| convert | 把表达式转换为在指定的格式中指定的数据类型. |
| isdate | 如果表达式的值是数据库的有效日期, 则返回真. |
| isnull | 如果表达式的值是null, 则返回真. |
| isnumeric | 如果表达式的值是数字, 则返回真. |

# 数据库

## 创建 数据库

|  |  |
| --- | --- |
| 创建数据库 | create database database\_name |
| 使用指定的数据库 | use database 或 connect |
| 查看所有的表 | show tables; |
| 查看表结构 |  |

## 修改 数据库

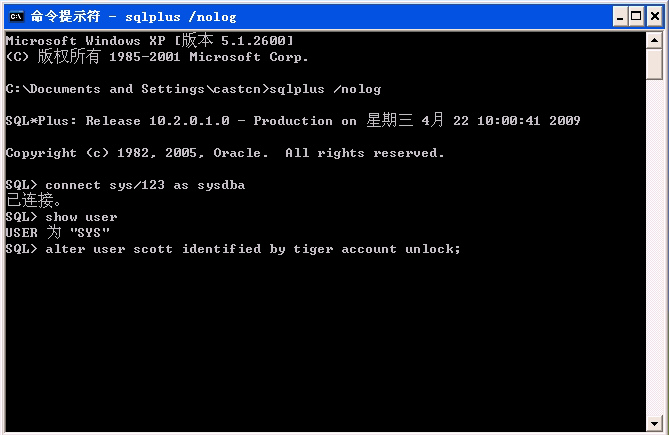
|  |
| --- |
| 在SQL标准中, 并没有提供修改数据库的命令. 但在一些数据库系统中, 提供了修改数据库的命令alter database, 允许用户修改数据库的大小, 名称, 以及其它信息 |

## 删除 数据库

|  |  |
| --- | --- |
| 删除单个数据库 | drop database dabase\_name |
| 删除多个数据库 | drop database dabase\_name1, dabase\_name2 |

|  |  |
| --- | --- |
| 查看当前用户 | show user; |
| 显示所有数据库 | show databases; |
| 指定要使用的数据库 | use mydata; |
| 显示当前数据库中的所有表 | select \* from tab; |
| 设置显示的列数目 | set linesize 100; |
| 设置显示的行数目 | set pagesize 100; |
| sys登陆 | su - oracle //linux中切换用户 [可选]  sqlplus /nolog  connect / as sysdba |
| sys登陆后, 解锁scott用户 | alter user scott identified by tiger account unlock; |
| sys登陆后, 修改语言 | alter system set nls\_language=american scope=spfile; |

**解决Oracle中 SQL plus无法登陆**



# 数据定义语言

## 表

**表的基本结构**

|  |
| --- |
| 每列称为一个字段, 每列的标题称为字段名,  一行数据成为一条记录,  一个表有一条或多条记录组成, 没有记录的表称为空表. |
| |  |  | | --- | --- | | 永久表 | 如果灭有特别的说明, 通常所说的表就是永久表. | | 全局临时表 | 只有在SQL会话的上下文引用该表定义时, 实际的表才会存在, 对会话结束后, 表就不再存在.  不能从一个SQL会话访问在另一个会话中创建的表 (SQL会话是指用户与SQL之间的连接)  全局表的创建语句为: create global temporary table | | 局部临时表 | 只有在SQL会话的过程中才能被引用, 并不能从另一个SQL会话对其进行访问.  与全局临时表的不同之处在于: 在SQL会话的任何地方都可以访问全局临时表, 而局部临时表只有在相关的SQL模块内才能被访问.  局部临时表的创建语句为create local temporary table | |

**非空约束**

**丢失数据(NULL值)**

|  |
| --- |
| null表示一个不可用, 未分配, 未知或不适用等意义的值, 与0和空格严格分开(而不是表示一个值为null的值), 所以不能将其与其他值比较(即使其他值也是null),所以SQL允许除了在true 和 false之外还有第三种类型的值"不确定"(unknown)  null值使用is null 和 is not null来检查.  null并不是0, 而是一个丢失, 不知道, 或 不可用的值.  如果数学表达式中有一个为空值, 则结果为空值. |

**指定缺省值**

**别名 和 属性**

|  |
| --- |
| 字段别名 和 数据表别名  别名都直接放在各自对应名称的后面, 中间用空格分开. |

### 创建 表

|  |
| --- |
| create table table\_name(  column1 datatype1 [null | not null]  , column1 datatype1 [null | not null]  , column1 datatype1 [null | not null]  ...  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  ALTER TABLE `user`  ADD FOREIGN KEY (`fk\_roles\_id`) REFERENCES `roles` (`id`); |
| 使用外部文件创建表: \. c:\\mysql\_script\\mydata.sql | \. c:\\mysql\_script\\mydata.sql |
| 注意:  最后一行结尾的逗号需要删除.  在Oracle中创建表, 表名不得超过30个字符, 由于Oracle对大小写不敏感, 所以在写名字时可以根据需要采用大写或小写方式.  表的第一个字符必须是字母, 其余的字符则可以是下划线, #, $, @等.  字段类型默认为null, 在一个字段上只能使用一次not null参数. |

**创建约束 删除约束**

|  |  |
| --- | --- |
| 创建约束 |  |
| 删除约束 | alter table 表名 drop constraint 列名; |
| 添加外键约束 | CREATE TABLE `bill` (  `fk\_user\_id` int(11) default NULL COMMENT '销售员id',  PRIMARY KEY (`id`),  KEY `cons\_1` (`fk\_user\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  ALTER TABLE `bill`  ADD FOREIGN KEY (`fk\_user\_id`) REFERENCES `phone`.`user` (`id`); |
| 删除外键约束 | ALTER TABLE `bill`  DROP FOREIGN KEY `cons\_1`; |

### 修改 表

|  |  |
| --- | --- |
| 1. 在已有表中添加列  (添加列不能设置为not null) | alter table table\_name  add colum\_name datatype [default expression] [references table\_name(column\_name)]  [check constraint] |
| 2. 在表中修改现有的列  (可修改列的类型 和 列具有的各种约束条件  修改后的数据长度不能少于已有数据中长度最大值  修改数据类型时, 必须保证列中的已有数据不会和新的数据类型产生冲突.) | alter table table\_name  modify column\_name [datatype]  [default expression]  [references table\_name(column\_name)]  [check constraint]  修改MySql列类型：  ALTER TABLE t\_msg\_temp MODIFY COLUMN content longtext; |
| 3. 删除列会导致数据丢失, 最好不  可以通过创建视图来屏蔽不需要的列, 并通过视图来检索数据  对于需要进行大量改动的表, 可以创建一个包含所需列的表, 然后结合使用insert和select语句, 将数据转换为新表中的数据. | alter table orders drop column profit\_sum; |
| 添加外键 | ALTER TABLE cdc\_mapreduce\_instance  ADD FOREIGN KEY (mapreduce\_id)  REFERENCES cdc\_mapreduces(id); |
| 删除外键 | 第一步查找外键  SHOW CREATE TABLE cdc\_mapreduce\_instance;  | cdc\_mapreduce\_instance | CREATE TABLE `cdc\_mapreduce\_instance` (  `mapreduce\_id` int(11) NOT NULL,  `yarn\_application\_id` varchar(255) DEFAULT NULL,  `yarn\_mapreduce\_status` varchar(32) DEFAULT NULL,  `id` bigint(20) NOT NULL AUTO\_INCREMENT,  PRIMARY KEY (`id`),  KEY `FKB50E39AFFBD0F7ED` (`mapreduce\_id`),  CONSTRAINT `FKB50E39AFFBD0F7ED` FOREIGN KEY (`mapreduce\_id`) REFERENCES `cdc\_mapreduces` (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8 |  第二部删除外键  ALTER TABLE cdc\_mapreduce\_instance DROP FOREIGN key FKB50E39AFFBD0F7ED; |

### 删除 表

|  |
| --- |
| 将删除表的定义, 全部数据, 索引, 触发器, 约束, 指定表的许可 等  当删除表时, 绑定在该表上的规则或者默认值得解除绑定, 与表关联的约束或者触发器将自动删除. 如果重新创建表, 必须重新绑定相应的规则和默认值, 重新创建触发器和增加必要的约束.  删除表时, 基于表的试图不会自动删除, 必须使用drop view语句删除视图.  不能使用drop table语句删除正在被约束参考的表, 要么先删除外键约束, 要么删除参考表.  系统表不能删除.  删除表的许可属于表的所有者, 然而, 数据所有者(DBO), 系统管理员(SA) 和 DLL管理员可以删除数据库中的任何对象. |
| drop table table\_name |
| 由于删除表语句的不可更改性, 错误的使用可能会产生灾难性的后果, 因此在多数据库中, 删除表是, 应该使用表的所有者名(owner)或模式名(schema). 例如: drop table Owner.Employee |

### 重命名 表

### 查看表结构

|  |
| --- |
| desc 表名; |

## 索引

|  |
| --- |
| 索引按存储结构的不同, 分为: 簇索引 / 非簇索引  **簇索引:**  对表的物理数据页中的数据按列进行排序, 然后再重新存储到磁盘上. 即簇索引与数据是混为一体的, 它的叶子节点存储的是实际的数据. 对表中的数据一一进行了排序, 因此查找数据很快.  但由于簇索引将表的所有数据完全重新排列了, 所需空间也就特别大, 大概相当于表中数据的120%.  一个表只能有一个簇索引, 因为表行必须以簇索引的顺序排列, 而一个表在磁盘上只能有一个物理记录排列方式;  使用簇索引的好处:  1. 表将占用最小的磁盘空间, 因为DBMS在插入新行时, 会自动地重用以前分配给删除行的空间.  2. 对基于簇索引的列值进行查询时, 会有更快的执行速度, 因为所有值在物理磁盘上相符靠近.  3. 基于簇索引的列以升序显示数据查询, 不再需要order by子句, 因为表的数据本身已经以所要求的输出顺序排列.  **非簇索引:**  非簇索引不必将物理数据页中的数据按列排序, 具有与表的数据分离的结构.  非簇索引的叶节点中存储了关键字的值和行定位器.  行定位器的结构和存储内容取决于数据的存储方式.  如果数据是以簇索引方式存储的, 则行定位器中存储的是簇索引的索引键;  如果数据不是一簇索引方式存储的, 则行定位器存储的是指向数据行的指针, 这种方式又称为堆存储方式.  非簇索引将行定位器按关键字进行排序, 这个顺序与表的行在数据页中的顺序是不匹配的.  由于非簇索引使用索引页存储, 因此比簇索引需要更多的存储空间,且检索效率较低. 但一个表一个表只能建立一个簇索引, 当要使用多个索引时, 就需要使用非簇索引了.  理论上一个表最多建立249个非簇索引 |
| **适合使用索引的情况:**  1. 经常使用查询  2. 查询返回记录相对较少  3. 表中记录在该列上含有大量不同的值  4. 用于两个表连接操作 |
| **使用索引的技巧:**  1. 不要对经常需要更新或修改的字段创建索引, 因为更新索引的开销很大.  2. 当查询要返回的数据很少时, 索引可以优化该查询(比较好的情况是少于全部数据的25%), 否则索引的开销很大.  3. 因为对字段的索引已经对两个表进行了归并操作, 所以可以极大地提高归并的速度.  4. 如果要进行大量的插入或更新操作, 在执行更新操作之前不要忘记先删除索引, 等更新完毕再创建索引.  5. 对于数据比较少的表来说, 使用索引对于性能不会有任何提高, 所以尽量不要在数据比较少的表中创建索引.  6. 当用户要检索的字段的数据包含有很多数值或很多空值(null)时, 为该字段创建索引, 会大大提高检索效率.  7. 索引列在where子句中应频繁使用, 例如, 在学生姓名字段上建立了索引, 但实际查询中根本不是经常用姓名作为查询条件, 该索引就没有起到作用.  8. 先装数据, 再建索引.  9. 尽量要把表和它的索引存放在不同的磁盘上, 这样会提高查询速度. |
| **创建索引**   |  | | --- | | **标准SQL中创建索引的语法格式:**  create [unique] index index\_name /\* unique 用于创建唯一索引时使用, 唯一索引: 索引的键值不重复, 在列中包含重复值 或 null时, 不能创建唯一索引.\*/  on table\_name (  columnl\_name  , columnl\_name /\*如果在多个列上创建复合索引,可以给出多个列名\*/  ....  ) | | **SQL Server 中创建索引的语法格式:**  **create** [unique] [clustered | nonclustered] /\* clustered 指明创建簇索引, nonclustered 指明为非簇索引\*/  index index\_name on{table | view} column [ASC | DESC] [,...n]) /\*视图必须是使用schemabinding选项定义过的; 排序方式: 默认ASC; column 为指定索引的列, 最多可以有16个列, 列的数据类型长度和不能超过900个字节.\*/  [with  [pad\_index] /\*指定填充索引的内部节点的行数, 至少应大于等于两行. pad\_index选项只有在fillfactor选项指定后才起作用, 因为pad\_index 会使用与fillfactor 相同的百分比\*/  [[,] fillfactor=fillfactor] /\* fillfactor 称为填充因子, 值在1~100之间, 对于经常插入或删除的表, 通常设置fillfactor =10; 而对于数据不更改的, 高并发的, 只读的表, fillfactor 可以设置到95以上.\*/  [[,] ignore\_dup\_key] /\*用于控制当往包含于一个唯一约束中的列中插入重复数据时, SQL Server说做出的响应.\*/  [[,] drop\_existing] /\*指定要删除并重新创建簇索引, 删除簇索引会导致所有的非簇索引被重建, 因此需要用行指针来替换簇索引键. 如果再重建簇索引, 那么非簇索引又会重新建一次, 以便用簇索引键来替换行指针. 使用了drop\_existing选项可以使非簇索引只重建一次.\*/  [[,] statistics\_norecompute] /\*指定分布统计, 不自动更新.\*/  [[,] sort\_in\_tempdb] /\*指定用于创建索引的分类排序结果将被存储到Tempdb数据库中 \*/  ]  [on filegroup] /\*指定存放索引的文件组\*/ | | **My SQL中创建索引的语法格式:**  **create** [unique] index index\_name on table\_name(  columnl\_name[(length)] /\*length是指自用一个字段的部分来创建\*/  , ...  ) |   **其中用到的几个关键字**   |  | | --- | | **unique(distinct)**  唯一性索引, 不允许表中不同的行在索引列上取相同的值. 若已有相同值存在, 则系统给出相关信息, 不建此索引.  **clustered / nonclustered**  簇索引 和 非簇索引, 若为clustered, 则为簇索引, 即表中元组按索引项的值排序, 并聚集在一起, 一个基于表上只能建一个簇索引.  nonclustered表示创建的索引为非簇索引, 缺省时, 创建创建的为非簇索引.  **asc / desc**  索引表中索引值的排序次序, 缺省为asc(正序排列) | |
| **删除索引:**  如果需要改变索引的类型, 必须删除原来的索引并重新创建一个.  无论是簇索引还是非簇索引, 都可以用标准删除索引的SQL语句: drop index tablename.indexname |
| **注意:**  最好在创建表的时候就创建索引, 特别是要保证唯一性的情况. 不过SQL夜支持添加了记录之后再为表创建索引, 特定情况下这样可能更好.  数据类型为text, ntext, image或bit的列不能作为索引的列,  由于索引的宽度不能超过900个字节, 因此数据类为char, varchar, binary和varbinary的列的总宽度不能超过900字节; 数据类型为nchar, nvarchar的列的总宽度也不能超过450个字节.  对于查询表中的所有信息, 使用索引毫无意义. |
| **复合索引: 对多个字段进行索引**  **特征:**  1. 一个表中, 把两列或更多列指定为索引  2. 对复合列作为一个单元进行索引  3. 创建复合索引中的列顺序不一定与表定义时列的顺序顺序相同.  在创建复合索引时, 一般都是在查询中有用的列上创建.  创建复合索引只是创建了一个实际的索引, 而不是多个索引, 所以复合索引的性能与单个字段的索引相比是无法断定的. |
| **唯一索引: 可以避免相同值的多个记录重复出现. 常常用于主键列上创建, 以保证可以唯一的标识记录.**  **特征:**  1. 不允许两行出现相同的索引值  2. 实施实体完整性. 是确保数据库完整性的方法之一, 可以防止有相同数据的多个记录多次出现.  3. 在创建主键约束和唯一约束时自动创建.   |  | | --- | | create unique index id\_index  on Stu(id) |   4. 如果用户试图使用insert 或 update语句, 在拥有惟一索引的数据中生成一个重复的值, 那么insert或update就会被终止, SQL服务器会生成一个错误信息.  5. 创建惟一索引时, 应确保被索引的列不允许null值. 例如SQL Server将null也看作一个值, 因而如果有第二个null就失败.  6. 创建惟一索引的前提: 已有的记录本身没有重复的值. |
| **群集索引:**  前面讲的索引, 索引的创建不会修改记录的实际物理存储地址. 在删除索引后就不会对数据库影响了.  群集索引, 不仅创建表的索引, 而且为表中记录的存储地址进行重新排列, 使其物理存储顺序和表中记录的顺序相符. 当向创建了群集索引的表添加记录时, 表自身也会按照索引的顺序自动重新排列. |

## 视图

|  |
| --- |
| 视图的意思: 视图就像一个表, 不同的是只是它在数据库中没有物理存在. 视图是当用户访问它的时候根据要求创建的临时表. |

视图的处理过程

视图的**优点**和**缺点**

|  |
| --- |
| 视图的优点: 简化操作, 定制数据, 合并分隔数据, 安全性等方面 |
| 视图的缺点:  性能:  更新限制: 不是所有的视图都可以更新的, 目前SQL将可更新的视图限制为基于单个表的, 并且没有group by或者having子句的查询, 除此之外, 为了使视图是可更新的, 视图不能使用聚集函数, 计算的列或select distinct子句. |

**创建视图**

创建简单视图

|  |
| --- |
| 语法:  create view view\_name [(column, column2 ...)] as select column\_name from table\_name;  例子  create view aa as select student.student\_name t1, class.class\_id t2 from class, student where class.class\_id = student.class\_id  create view aa as select class.class\_name, student.student\_id, student.student\_name where class.class\_id = student.class\_id |

创建列的**别名**

使用**连接**的视图

使用**子查询**的视图

**修改**视图

**删除**视图

|  |
| --- |
| drop view view\_name |

通过视图管理数据

通过视图插入数据

通过视图更新数据

通过视图删除数据

查询视图信息

## 存储过程

编辑存储过程

使用存储过程

调用存储过程

在存储过程中编程

使用变量

使用流控制

返回值

产生出错信息

使用参数

创建参数

传递值

使用结果

## 触发器

创建SQL触发器

创建Insert触发器

创建Update触发器

创建Delete触发器

嵌套触发器 和 递归触发器

## 模式

## 域

## 建立联合

使用union运算符

遵循联合的约束

建立实用的联合

|  |
| --- |
| 使用union运算符可以将两个或两个以上select语句的查询结果集合合并成一个结果集合显示.  语法:  select\_statement  union [all] select\_statement  [union [all] select\_statement][...n]  例子:  select \* from student union select \* from elective |
| 例子: 查询总数  select count(\*) from (  select ID from T\_USER\_TRADE\_LOG\_201108 where 1=1  union all  select ID from T\_USER\_TRADE\_LOG\_201109 where 1=1  ) as t |
| 例子: 查询列表  select \* from T\_USER\_TRADE\_LOG\_201108 where 1=1 and TRADE\_TIME > 2011-08-01  union  select \* from T\_USER\_TRADE\_LOG\_201109 where 1=1 and TRADE\_TIME < 2011-09-01 |

# 数据操作语言

## 记录

### c插入 记录

|  |  |  |
| --- | --- | --- |
| 插入方式 | 语法 | 例子 |
| 使用insert...values语句  一次只能向表中添加一条记录  如果插入全部字段, 可以省略表名后面的字段列表;  如果插入部分字段, 则必须在表名后面列出字段列表 | insert into tablename[(columnlname1, [columnlname2, ...])]  values (value1, [value2, ...]) |  |
| 使用insert...select语句 | insert into tablename [(column1name, [column2name, ...])]  select columnlist  from tablelist  where conditions | insert into table2(name, age)  (select name, age from table\_1) |
| 一次能向表中添加一条或多条记录 |  | insert into roles(name, remark) values ('1', '1'),('1', '1'),('1', '1') |
| 可以用来: 表间数据复制 |  | create table table\_3  as(select name, age from table\_1); |

### 修改 记录

|  |  |  |
| --- | --- | --- |
| 插入方式 | 语法 | 例子 |
| 指定更新的列和新值 | update tablename  set column1name = value1, [set column2name = value2, ...]  [where conditions] | update authors set contract = 1, phone=''; |
| 指定更新的条件(嵌套更新) | update tablename  set column1name = value1  from (select ...) as t1  where t1.column1name = table2.column2name | update scott.emp  set sal=  (select sal+1000 from scott.emp where empno=7666)  where empno=7666; |

### 删除 记录

|  |  |  |
| --- | --- | --- |
| 插入方式 | 语法 | 例子 |
| 删除指定条件的记录  (仅删除记录, 不会删除表) | delete from tablename  [where conditions] |  |
| 删除指定表的所有记录  保留与表有关的所有结构 | 第一种: delete from tablename; --还允许对指定行的数据进行删除  第二种: truncate table tablename; --不通用, 且仅仅对表的全部删除有用, 不能用来选择性地删除数据) |  |

# 查询

## 一般查询

|  |
| --- |
| select [distinct | all] select\_list --distinct: 删除查询结果完全相同的行,  from table\_name  [where search\_condition]  [group by group\_by\_expression]  [having search\_condition][ --根据group by子句产生的结果进行筛选  [order by order\_expression [asc | desc]] --对查询结果进行排序 |

**select语句的执行步骤**

|  |
| --- |
| 1. 执行from子句, 根据from子句中的一个或多个表创建工作表, 如果在from子句中有两个或多个表, DBMS将对表进行交叉连接, 作为工作表.  2. 如果有where子句, DBMS将where子句将where子句列出的搜索条件作用于步骤1生成的工作表. DBMS将保留那些满足搜索条件的行, 删除那些不满足搜索条件的行.  3. 如果有group by子句, DBMS将步骤2生成的结果表中的行分成多个组, 每个组所有行的group\_by\_expression字段具有相同的值, DBMS将每组减少到单行, 而后将其添加到新的结果表中.  4. 如果有having子句, DBMS将having子句列出的搜索条件作用于步骤3生成的"组合"表中的每一行, DBMS将保留那些满足搜索条件的行, 删除那些不满足搜索条件的行.  5. 将select子句作用于结果表. 删除结果表中不包含在select\_list中的列, 如果select子句包含distinct关键字, DBMS将从结果中删除重复的行.  6. 如果有order by子句, 按指定的排序规则对结果进行排序.  7. 对于交互式的select语句, 在屏幕上显示结果; 对于嵌入式SQL, 使用游标将结果传递给宿主程序. |

**select子句**

|  |  |
| --- | --- |
| 选择所有列 |  |
| 选择所需列 |  |
| 选择某个列中相异值(过滤掉某列相同的记录) | select distinct column from table |
| 别名的使用 |  |
| 使用数字和表达式 | select column1, column2, column3, 1+2 from tabl3 |
| 使用函数进行查询 |  |

## 运算符

|  |  |  |
| --- | --- | --- |
| 比较运算符 | = 、<> 、!= 、 > 、 < 、 >= 、 <= |  |
| 逻辑运算符 | and or not |  |
| in 或 not in 子句 | 在逗号分隔的值列表中查找, 代替多个or组合语句 | where age in(20, 21, 22); |
|  |  |  |
|  |  |  |
|  |  |  |
| between ... and ... | 值在value1~value2之间  必须把小的值放在and前面, 大的值放在and后面  如果是between 30 and 50, 则结果集中包含30和50在内. | where comumn between 50 and 100; |
| like子句匹配字符串 | \_ : 单个字符;  % : 0个或多个字符   |  |  | | --- | --- | | \_ | 任意单个字符 | | % | 任意多个字符 | | \\_ | 代表下划线这个字符, 需要用escape '\'是声明\作为为转义字符 | | \% | 代表百分号这个字符, 需要用escape '\'是声明\作为为转义字符 | | '' | 用两个单引号表示一个单引号, 不需要用转义字符来表示单引号, 插入数据前需要把数据中的单引号替换成两个单引号 |   转义字符可以是任何符号, 但一般都用\ | where name like '罗\_';  where name like '2%%'; 两个百分号转义为一个百分号, 代表包含2%的所有字符串  where name like '2$%' escape $; //指定转义字符$  where name like '[Vv]%'; 忽略大小写 Oracle 10g中不能使用 |
| 等号, in, like三者的比较 |  |  |
| 空值的处理 | is null  is not null  注意:`  不可写成 sal <> null  也不可写成 sal != null | select name, age  from students  where sal is not null |

|  |  |  |
| --- | --- | --- |
| all |  |  |
| some |  | 在功能上any和some是相当的 |
| any | select au\_lname, city from authors where city = any(select city from publishers) | 允许我们匹配任何从子查询返回的值. |

|  |  |  |
| --- | --- | --- |
| and |  |  |
| or |  |  |
| and 和 or组合 | and运算符的优先级高于or  condition1 or condition2 and condition3 等价于 condition1 or (condition2 and condition3) |  |

**order by 排序子句**

|  |  |
| --- | --- |
| 1. order by子句中可以包含多个元素, 元素之间用逗号隔开.  2. order by子句中的列名可以是select中的列名或别名或位置编码数字  3. order by一定要放在整条sql语句最后  4. select name, age from students order by age, name; --如果使用了别名, 那么这里将用别名 | |
| 升序排序 和 降序排序 | order by columnname [asc | desc] |
| 使用表达式排序 |  |
| 多级排序 | order by column1name, column2name |
|  |  |

**group by 和 having子句**

|  |  |
| --- | --- |
| group by子句 | 根据列的内容, 对查询结果按某个字段进行分组.  出现在select字段列表中的字段, 如果一个字段没有被包含在组函数中, 那么这个字段必须出现在group by子句中;  出现在group by子句中的字段, 可以是表中的任意字段, 这个字段可以不是select字段列表中的字段.    执行过程是: 先按某字段分组, 然后对同一组的其它某个字段执行求平均值, 或求和等多行函数.  不可在where子句中出现group by子句 |
| 多次分组 | group by column1name, column2name |
| having子句 | 对形成的组设置条件, 以返回需要的组(用于过滤分组)  为了使用having, 必须先对数据分组, 所以说having在group by短语的后面出现. 除非每个其他列名称已经包括在聚合函数中.  having子句中的每一个元素必须出现在select列中, 但是有些数据库系统也允许在having子句中使用select列表之外的元素  having子句中也可以使用and, or或not运算符来包含多个限制条件. eg: having sum(column\_3) > 60 or avg(column\_3) <30  eg:  having avg(sal) > 1200 |
| having和where的区别 | 1. where子句用来设置返回行的条件, having子句用来设置返回组条件  2. where子句不能使用聚合函数, having子句可以使用聚合函数. |

## 多表连接查询

|  |
| --- |
| **基本概念**  要将两个表按照一定的方式连接在一起, 两个表必须有公共的数据列.  当用来连接的列是主键及对应的外键时, 会保持主键列与外键列在值上是一致的, 可以很好的保持数据库完整性, 可以获得更好的连接. |
| **较好的连接应该符合以下语法:**  1. 用于连接的列必须有相同的数据类型. 或可以自动进行类型转换. 不同类型的数据是不能进行连接的.  2. 用于连接的列必须具有相同的含义, 没有相同的含义的列的连接是毫无意义的.  3. 进行连接的列名可以不同, 当需要使用相同名称的列时必须为其加上表名作为限制. 不然会产生混淆.  4. fron子句中必须包含用于连接的所有表.  5. 连接时, 还需指定连接条件, 如果没有使用条件连接, 查询结果将是两个表所有记录的笛卡尔积. |
| **连接表的处理步骤:**  1. 为连接表生成笛卡尔积, 这是连接操作引起性能下降的主要原因.  2. 使用where子句从笛卡尔积中除去所有不符合限制条件的行.  3. 使用where子句进行删除后, 如果语句包含group by子句, 则按照group by子句中指定的列对剩下的行进行分组. 虽然这时select子句还没有执行, 但是如果select列表中包含了聚集函数将会对分好的组进行计算. 如果使用了having子句, 将对分组结果进行删除.  4. 选出在select列表中的列,删除其他列. |

**传统的 连接语句**

|  |
| --- |
| **语法:**  select select\_list  from tablename, [tablename ...]  where [table1name.]column operator [table2name.]column;  **例子:**  连接两个表  select thesis\_user.username, classes.name  from thesis\_user, classes  where thesis\_user.fk\_classes\_id=classes.pk\_classes\_id;  连接多个表  select name, id, sal  from emp, emp1, emp2  where emp.id=emp1.id and emp.id=emp2.id; |

**现代的 连接语句**

|  |
| --- |
| **语法**  select select\_list  from table1name [cross | natural] join table2name --交叉连接 | 自然连接, 如果使用了这两个  [on [table1name.]column operator [table2name.]column] --字段关系  [where conditions]  **关键字解释**  如果选择了cross | natural中的一个, 就不能使用on关键字了,  使用cross join时, 不能使用关键字on, 因此必须使用where子句来设定连接的条件. 如果不设定条件, 将返回笛卡尔积.  natural join, 只有在两个表有相同名称列的时候才能使用, 并且将在同名列上进行连接.  **实例**  连接两个表  select thesis\_user.username , classes.name  from thesis\_user join classes  on thesis\_user.fk\_classes\_id=classes.pk\_classes\_id;  连接多个表  select thesis\_user.username , classes.name, professional.name  from thesis\_user join classes  on thesis\_user.fk\_classes\_id=classes.pk\_classes\_id  join professional  on professional.pk\_professional\_id=classes.fk\_professional\_id |

|  |  |
| --- | --- |
| join\_table | 参与连接操作的表名 |
| join\_type | 连接的类型 |
| on (join\_condition) | 指出连接条件, 它由被连接表中的列和比较运算符, 逻辑运算符等构成. |

**on子句**

|  |
| --- |
| select empno, ename, deptno, dname  from emp join dept  on (emp.deptno = dept.deptno); --指定连接条件 |

**内连接**(inner join)

|  |  |
| --- | --- |
| 内连接的概念 | 使用比较运算符进行表间某些列数据的比较操作, 并列出这些表中的链接条件相匹配的数据行.  可以说: 内连接是笛卡尔积的子集.  eg:  select \*  from roles r inner join user u  where u.fk\_roles\_id=r.pk\_roles\_id;  eg:  select \* from table1 inner join table2 on table1.job\_id = table2.job\_id; |
| **等值连接** | 解释: 在连接中使用等号(=)运算符比较被连接列的值.  语法:  select select\_list  from table1 as t1 inner join table2 as t2  on t1.colunm = t2.colunm; |
| **不等值连接** | 解释: 在连接中使用除等于运算符以外的其他运算符来比较连接列的值. 包括 >, >=, <, <=, !>, !<, <>  语法:  select select\_list  from table1 as t1 inner join table2 as t2  on t1.colunm > t2.colunm; |
| **自然连接** | 基于两个表中的所有同名列, 选出同名列的值对应相等的所有行.  如果两个表中相同列的数据类型不同, 则出错.  不允许在参照列上使用表名或者别名作为前缀.  select empno, ename, sal, deptno, dname  from emp natural join dept; |
| **备注** | inner可以省略不写, where也可以用on代替 |

**外联接**

|  |  |
| --- | --- |
| 与内连接不同的是: | 内连接的查询结果: 符合查询条件(where或having)和连接条件的行  外连接的查询结果: 是内连接查询结果的扩展. 不仅包含符合连接条件的行, 还包括左表(左外连接时), 右表(右外连接时) 或两个连接表(全外连接时)中的所有不符合搜索条件的行.  外连接不只列出与连接条件向匹配的行, 而是列出左表(左外连接时),  右表(右外连接时) 或两个表(全外连接时)中所有符合搜索条件的数据行. |
| 左外联接(left outer join 或 left join) | 返回 左边表所有行 和 右边表相匹配的行  保证左表的所有记录都返回, 与之相应的右表没有对应记录的字段就返回null.   |  | | --- | | select \* from table1 left join table2 on table1.au\_id = table2.au\_id; | | select emplogee\_id, first\_name, salary, a.department\_id, department\_name  from employees a, departments b  where a.departmentid = b.department\_id (+)  order by employee\_id;  a表的结果会全部显示出来, b表只显示符合条件的行. | |
| 右外联接(right outer join 或 right join) | 返回 左边表相配配的行 和 右边表所有行.   |  | | --- | | select \* from table1 right join table2 on table1.au\_id = table2.au\_id; | | select emplogee\_id, first\_name, salary, a.department\_id, department\_name  from employees a, departments b  where a.departmentid(+) = b.department\_id  order by employee\_id;  b表的结果会全部显示出来. | |
| 全外连接(full outer join 或 full join) | 将左表的所有数据分别与右表的每条记录进行连接组合, 返回的结果除内连接的数据外, 还有两个表中不符合条件的数据, 并在左或右表的相应列中填上null值.   |  | | --- | | select a.username , b.name  from thesis\_user a full join classes b  on a.fk\_classes\_id=b.pk\_classes\_id; | |

**自连接**

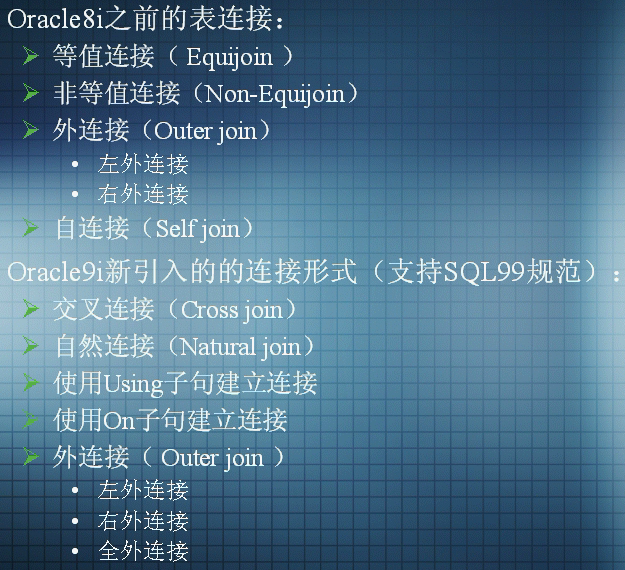
|  |
| --- |
| 对自关联的表  1. 一个表的两个副本之间的内连接.  1. 使用连接时, 必须为表指定两个不同的别名, 使之在逻辑上成为两个表.  2. 必须设定条件 |

**交叉连接**

|  |  |
| --- | --- |
| 没有条件的联接, 称为交叉联接(cross join) (没什么意义) | 它给出了一个结果集, 这个结果集是一行对于表中各行的可能组合, 结果集也称为笛卡尔积.  没有where子句来限定条件  select 字段列表from table1 cross join table2; --这是交叉连接  返回连接表中所有数据行的笛卡尔积.  select s.id, s.name, e.id, e.course\_name, e.course\_id, e.stu\_id  from elective e cross join student s  order by s.id  select empno, ename, deptno, dname  from emp join dept  using (字段1, 字段2); --只比较字段和字段2这两个同名连接. |

**集合查询**

|  |
| --- |
| 如果想把多个查询结果组合起来, 可以使用集合查询功能. |
| **unoin运算**  是将多个查询的结果组合到一个查询结果中, 可以从多个表中组合数据进行显示. 默认情况下, union将删除多个查询结果中的重复行.  但是与连接不同, union不是在from子句中添加多个表, 并指定连接条件, 而是通过将多个查询的结果结合到一起实现的.  使用union应注意以下几点:  1. 要求两个查询语句中列的数量相等和列的数据类型兼容.  2. 最后结果集中的列名来自第一个select语句的列名.  3. 在集合查询中, 在没有使用了all关键字的情况下. 默认将从最后的结果集中删除重复的行,  4. 要对集合进行查询排序时, 必须使用第一个查询语句中的列名或列序号.  5. 查询结果将对select列表中的列按照从左到右的顺序自动进行排序, 与查询相对于union运算符的位置无关.  6. 集合运算的每个查询都可以有自己的where子句, 但是整个集合只能使用一个order by子句, 并且order by子句只能放在集合运算的最后一个查询中.   |  | | --- | | SQL Server 2000中, 使用union语句的语法:  select\_statement  union [all]  select\_statement  [union [all]  select\_statement  [...]  例子  select pub\_id, pub\_name  from publicshers  where state='ca'  union  select stor\_id, stor\_name  from stores  where state='ca' | | Oracle中, 使用union语句的语法:  select select\_list  from table\_list / vies\_list  [where conditions]  union [all]  select select\_list  from table\_list / vies\_list  [where conditions]  例子 | |
| **intersect运算**  返回多个查询中相同的数据行, 即多个查询结果的交集. (没有交集, 则不会有行返回)   |  | | --- | | 例子  select classes.id  from classes  intersect  select address.id  from address | |
| **minus运算**  返回在一个查询结果中存在, 而在另一个查询结果中不存在的行, 即两个查询的补集. (在上面, 但不在下面)或者说(下面的集合是上面集合的子集)   |  | | --- | | 例子:  select classes.id  from classes  minus  select address.id  from address | |
| 注意:  SQL Server不支持intersect和minus两种运算, 但Oracle支持. |



## 子查询

**建立子查询**

|  |  |
| --- | --- |
|  | 子查询是在查询中的查询(嵌套在另一个select, insert, update等语句中的select语句.) 那个外部的查询语句相应地成为该子查询的外层查询. 这个子查询则称为被嵌套查询. 子查询的结果将作为外层查询的参数,  当子查询的结果必须出现在记录集合中时, 最好使用一个AS来命名他们的查询结果.  在主查询中引用表中的列的子查询, 叫做相关子查询. |
| 子查询的执行过程 | 外层查询语句执行一行, 子查询语句执行一遍, 然后判断外层查询语句的条件是否满足. 如果条件满足, 则外层查询语句得到的数据行就是结果集中的行; 如果条件不满足, 则外层查询语句继续执行下一行. |
| 子查询应该遵循以的原则 | 1. 作为子查询的select语句可以包含选择列表, fron子句(不能省略) 和 where子句, having子句, group by子句等.  2. 子查询必须用括号括起来, 否则无法判断子查询的开始和结束.  3. 由比较运算符 或 关键字in引入的内层子查询select列表中只允许有一项内容. 即只能是一个列名 或 表达式. 而且在子查询的select列表中的列必须能与在外围语句的where子句中的列连接兼容.  4. 子查询中不能使用order by子句, 因为子查询不需要在内部处理他们的查询结果的顺序.  5. 子查询允许嵌套多层, 但最多只能嵌套255层.  6. 如果一个数据表只包含在子查询中, 而没有出现在其外层查询中, 则外层查询的选择列表中不能包含该数据的列.  7. 由不跟关键字any或all的比较运算符所引入的子查询除非预先确定了它只能成组返回单个的值, 否则不允许使用group by 或 having子句.  8. 在使用关键字exists引入的子查询的select列表中不需要指定具体的列名, 可以都用星号"\*"组成, 因为这种情况下只需要测试任何一个符合标准的行存在. |
| 替代联接 | 子查询和联接可以相互代替.  连接查询的效率远远高于子查询.  连接查询的复杂程度远远高于子查询语句. |
| 各有所长 | 子查询可以计算一个变化的聚合函数值, 并返回到外层查询进行比较, 而连接查询则不能.  但, 使用连接查询的结果可以包含两个表的数据, 而子查询不能. 只有在输出结果都来自于一个表时, 子查询才可以替换连接.  子查询与自连接查询可以相互替换使用,  使用子查询: select \* from type where id=(select fk\_type\_id from type where id=14)  使用连接: select t1.\* from type t1 , type t2 where t1.id=t2.fk\_type\_id and t2.id=14  当需要使用聚集函数, 并将它的值和其他值进行比较时, 要使用子查询; 当查询的结果要显示多个表的内容时, 使用连接. |

|  |  |
| --- | --- |
| select ... where id=(select...  子查询只能返回一行结果. | select age, name form user where id=(select p\_id from where id=3) |
| select ... where sal in(select ...  子查询可以返回多行结果 | select \* from type where id in(select id from type where fk\_type\_id=7) |

子查询 与 连接

|  |
| --- |
| 子查询都可以使用连接查询来代替, 实际上, 连接查询的效率远远高于子查询., 但是连接查询的复杂程度也远远高于子查询.  子查询可以计算一个变化的聚集函数值, 并返回到外层查询进行比较, 而连接查询则不能.  使用连接查询的结果可以包含两个表的数据, 而子查询不能.  只有在输出结果都来自于一个表时, 子查询才可以替换连接(自连接). |
| 当需要使用聚集函数, 并将它的值和其他值进行比较时, 要使用子查询;  当查询的结果要显示多个表的内容时, 使用连接查询. |

子查询 的 类型

|  |  |
| --- | --- |
| 相关子查询 | 相关子查询的运行和它外围查询关系密切.  是一种子查询和外层查询相互交叉的数据检索方法. 在子查询执行时要使用到外围查询的数据, 子查询结束后再将查询结果返回到外围查询中, 供外围查询比较使用.  相关子查询有点像连接, 在相关子查询中, 表的内容与外围查询中表的内容进行比较, 就好像是在使用连接时将两个表的内容进行比较一样, 只是相关子查询不适用连接条件而已.  相关子查询还可以用于外部查询的having子句. |
| 无关子查询 | 无关子查询的运行是不需要和外围查询发生联系的.  无关子查询在外围查询之前执行, 然后将返回数据供外围查询使用.  如果嵌套子查询语句中, 被嵌套的查询不包含对外围查询的任何引用, 就可以使用无关子查询.  相关子查询中的子查询引用外层查询中的from子句中的表中的列, 但不在子查询的from子句中列出. 如果在子查询的from子句中列出外层查询的from子句中列出的表, 则该子查询就变成了无关子查询. |

|  |
| --- |
| update Information  set 学号=b.学号,姓名=b.姓名  From Information a,学生信息表 b  Where a.某个字段 =b.某个字段  --注意一定要有where条件，不然Information表都更新为学生信息表中最后一条记录值 |
| **实例**  String sql = "update GoodsType set "+countName+"=(select "+countName+""+plusOrMinus+" from GoodsType where goodsTypeId="+goodsTypeId+") where goodsTypeId="+goodsTypeId; |

单行子查询

|  |
| --- |
| 在单行子查询中, 内查询只返回单行单列值. 因此可以把这种子查询作为一个常量来对待. 由于这种子查询只能返回一个值, 因此可以在where子句和子查询之间插入一个比较运算符(如等号), 如果子查询可能返回多个值, 那么就不能使用比较运算符. |
| **where子句中的单行子查询**  在where子查询中可以使用单行比较运算符来比较某个表达式与子查询的结果. 可以使用的比较运算符有"=, >, >=, <, <=, <> 或 != "等6个.  这些比较运算符都可以连接一个子查询, 且在未经使用all或any修饰的比较运算符连接子查询时, 必须保证子查询所返回的结果集合中只有单行数据. 否则将引起错误. |
| **select子句中的单行子查询**  只要不使用group by子句, 在select清单中使用了聚集函数的子查询就总是返回单个值的. 因此, 可以使用比较运算符来直接引入这种使用了聚集函数的子查询. |
| **having子句中的单行子查询** |

多列子查询

|  |
| --- |
|  |

多行子查询

|  |
| --- |
|  |

嵌套子查询

|  |
| --- |
|  |

子查询中的空值问题

|  |
| --- |
|  |

TopN查询分析

|  |
| --- |
|  |
|  |

# 实例

**组合字段输出**

|  |  |
| --- | --- |
| 注意: | 组合后的字段需要使用AS来重命名字段名 |
| 字符串类型字段, 使用 字符串相加 | select au\_lname + ',' + au\_fname as full\_name from authors |
| 对数值类型字段, 使用 加, 减, 乘, 除 | select qty\*price as total\_sale from sales |
| select 作算术运算 | select 3+2 from dual; |

|  |  |  |
| --- | --- | --- |
| 去除重复行  备注: 这是通过排序实现的去除重复行, | select distinct 列名, 列名 from 表名; | 多个字段出现的组合不重复 |
| 连接运算符 | select ename || ' is a ' || job from emp; | 实现列的合成, 类似于java中字符串相加, 空值在连接表达式中表示空字符串. |
| 字段别名 | select empno 员工编号, sal\*12 年薪 from emp; | 如果别名中使用特殊字符或空格, 或者强制使用大小写, 就把别名用双引号引起来. 注意:不能是单引号哦. |

**报表查询**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| select column1[, column2]  from table1 [, table2]  where conditions  group by column1 [, column2]  having ...  order by column1[, column2]   |  |  | | --- | --- | | group by | 指定汇总查询, 即不是对每行产生一个查询结果, 而是将显示的行进行分组, 再对每组长舌妇两个一个汇总结果, 它通常需要和统计函数sun(), count()等联合使用. | | having | 告诉sql只产生有group by得到的某些组的结果, 和where字句一样, 所需要的组也用一个搜索条件指定. | | order by | 将查询结果按一列或多列中的数据排序, 如果省略此字句, 则查询结果将是无序的. | |
| 在报表查询中经常使用的sql聚合函数如下:  count()  min()  max()  sum()  avg() |

|  |  |
| --- | --- |
| 查看系统当前日期时间 | select now(); |
|  |  |

# 连接符

# 数值运算

# 函数

|  |  |
| --- | --- |
| 行集函数 |  |
| 聚合函数 |  |
| 标量函数 |  |

# 数据完整性

|  |
| --- |
| 约束定义关于列中允许值的规则, 是强制完整性的标准机制.  使用约束优先于使用触发器, 规则 和 默认值. |
| 数据完整性分为4类:  1. 实体完整性    2. 域完整性    3. 参照完整性    4. 用户定义的完整性:    列约束   |  |  | | --- | --- | | notnull | 指定该列不能为空值 | | default | 指定当没有输入值时该列的默认值. | | unique | 指定该列不能有重复值, 可对定义为非关键字的任何列使用. | | check | 对列指定规则, 如值的定义域, 可限制输入到受约束列中的值. |   表约束   |  | | --- | | primarykey为对表指定主键, 用来保证列的完整性和引用完整性. | |

规则

|  |
| --- |
|  |

check约束

|  |
| --- |
|  |

unique约束

|  |
| --- |
|  |

not null约束

|  |
| --- |
|  |

primary key约束

|  |
| --- |
|  |

默认值default

|  |
| --- |
|  |

# 事务控制

事务模式

事务的commit和rollback

事务隔离级别

开始事务

使用事务保存点

终止事务

提交事务

回滚(撤销)事务

锁定

锁定模式

处理死锁

# 并发处理

# 安全性控制

## SQL安全模式

## 角色管理

## 权限管理

## SQL Server安全管理

## Oracle安全管理

使用grant 和 revoke

授予权限 或 角色

撤销授权

视图 与 SQL安全

# 嵌入式SQL

# SQL游标

声明游标

更新 和 搜索游标

打开游标

关闭 和 释放游标

检索游标 返回的数据(提取数据)

使用定位update和delete语句

# SQL优化

提高SQL语句的可读性

优化索引

查询优化

动态SQL

# 实例

## Sql Server,Oracle和MySQL的分页语句

|  |
| --- |
| **SQL Server**  SELECT id, title FROM (  SELECT TOP 2 \* FROM ( --倒序后, 查出前2条  SELECT TOP 3 \* FROM project Order by id asc --查出前3条  ) t1 Order by id desc  ) t2 Order by id asc |
| **Oralce数据库**  　从数据库表中第M条记录开始检索N条记录  　SELECT \* FROM (SELECT ROWNUM r,t1.\* From 表名称 t1 where rownum < 到第几条) t2 where t2.r >=从第几条考试  　　例如从表Sys\_option(主键为sys\_id)中从10条记录还是检索20条记录，语句如下：  SELECT \* FROM (SELECT ROWNUM R, r1.\* From TABLE r1 where rownum < 30 ) r2 　Where r2.R >= 10  或  SELECT \* FROM (SELECT ROWNUM R, r1.\* From (select ...) r1 where rownum < 30 ) r2 　Where r2.R >= 10  说明: 把黄色的table换成表或子查询 |
| **MySQL数据库**  　　My sql数据库最简单，是利用mySQL的LIMIT函数,LIMIT [offset,] rows从数据库表中M条记录开始检索N条记录的语句为：  　　SELECT \* FROM 表名称 LIMIT M,N  　　例如从表Sys\_option(主键为sys\_id)中从10条记录开始, 检索20条记录，语句如下：  select \* from table\_name limit 3, 2; //从第三条开始, 返回两条. |

**返回前10条数据**

|  |
| --- |
| select top 10 au\_lname, au\_fname from authors; |

### 使用JDBC批量更新

|  |
| --- |
| Connecion conn = session.connection();  String sql ="update student s set s.cardId=concat(s.cardid, 'beijing')"; //使用concat()方法连接字符串  PreparedStatement stmt = conn.prepareStatement(sql);  stmt.executeUpdate(); |

## MySQL / SQL Server / Oracle 特性

### MySQL特性

### SQL Server特性

## Oracle特性

Oracle实现自动递增: 需要创建一个序列.插入值的时候调用nextValue

## 启动 和 停止SQL Server命令

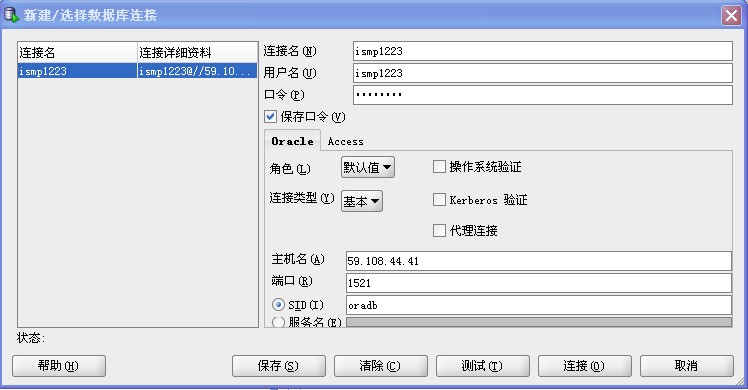
|  |  |
| --- | --- |
| 停止SQL Server | net stop mssqlserver |
| 启动SQL Server | net start mssqlserver |
|  |  |

## My SQL / SQL Server / Oracle 创建表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **My SQL(弄好咯)**  ##代理商表(proxy)  drop TABLE IF EXISTS `proxy`;  CREATE TABLE `proxy` (   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **##英文字段名** | **数据类型** | **缺省值** | **是否为空** | **是否自增** | **备注** | **备注值** | | id | int(11) |  | NOT NULL | auto\_increment, |  |  | | proxy\_name | varchar(50) | default | null |  | COMMENT | '代理商名称', | | person\_name | varchar(20) | default | null |  | COMMENT | '联系人', | | phone\_num | varchar(55) | default | null |  | COMMENT | '联系电话', | | address | varchar(100) | default | null |  | COMMENT | '代理商地址', | | remark | varchar(255) | default | null |  | COMMENT | '备注', |   PRIMARY KEY (id)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  INSERT INTO `proxy` VALUES(1,'强音通讯', '张三', '13320202020', '古镇开元', NULL); |
| **SQL Server** --1群类型树表MicBlogGroupTypeTree CREATE TABLE MicBlogGroupTypeTree (   |  |  |  |  | | --- | --- | --- | --- | | --字段名(英文) | 类型 | 是否为允许为空 | 默认值 | | micBlogGroupTypeTreeId | int IDENTITY(1,1) | NOT NULL**,** |  | | name | varchar(100) | NULL | DEFAULT (''), | | pId | int | NULL | DEFAULT ((0)), | | groupType | varchar(100) | NULL | DEFAULT (''), |   ) |
| **Oracle** |

|  |
| --- |
| **SQL Server导出表结构** |
| **SQL Server导出表结构**  1. 选择数据源(默认, 不作修改)    2. 选择数据源(目标选择好"平面文件")    3. 浏览, 选择保存地址和文件名.    4. 选择要导出得到表 |

## Oracle SQL Developer连接配置



## linux服务器, 修改mysql编码

|  |
| --- |
| 用root用户登录  [root@hp.xjgz /etc]# cat my.cnf  [mysqld]  default-character-set=utf8  datadir=/var/lib/mysql  socket=/var/lib/mysql/mysql.sock  user=mysql  # Disabling symbolic-links is recommended to prevent assorted security risks  symbolic-links=0  [mysqld\_safe]  log-error=/var/log/mysqld.log  pid-file=/var/run/mysqld/mysqld.pid  [client]  default-character-set=utf8 |

# 停止和启动oracle