SQL学习

SQL是什么

* 结构化查询语言
* 可以访问和处理数据库
* 一种ANSI标准的计算机语言

SQL可以做什么

* 面向数据库执行查询
* 可从数据库取回数据
* 可在数据库中插入新的记录
* 可更新数据库的数据
* 可从数据库删除记录
* 可创建新数据库
* 可在数据库中创建新表
* 可在数据库中创建存储过程
* 可在数据库中创建视图
* 可以设置表，存储过程和视图的权限

RDBMS

RDBMS指关系型数据库管理系统，是SQL的基础，同样也是所有现代数据库系统的基础，RDBMS中的数据存储在被称为表的数据库对象中，表是相关数据项的集合，由列和行组成。

SQL语法

数据库表

一个数据库通常包括一个或多个表。每个表有一个名字标识，表包含带有数据的记录（行）

SQL语句

需要在数据库上执行的大部分工作都由SQL语句完成

下面的SQL语句从”Websites”表中选取所有记录

实例：

SELECT \* FROM Websites;

SQL对大小写不敏感，select与SELECT是相同的

SQL语句后面的分号

某些数据库系统要求在每条SQL语句的末端使用分号

分号是在数据库系统中分隔每条SQL语句的标准方法，这样就可以在对服务器的相同请求中执行一条以上的SQL语句

一些最重要的SQL命令

* SELECT 从数据库中提取数据
* UPDATE 更新数据库中的数据
* DELETE 数据库中删除数据
* INSERT INTO 从数据库中插入新数据
* CREATE DATABASE 创建新数据库
* ALTER DATABASE 修改数据库
* CREATE TABLE 创建新表
* ALTER TABEL 变更（改变）数据库表
* DROP TABLE 删除表
* CREATE INDEX 创建索引（搜索键）
* DROP INDEX 删除索引

SQL SELECT 语句

SELECT语句用于从数据库中选取数据，结果被存储在一个结果表中，被称为结果集。

SQL SELECT语法

SELECT *column\_name*,*column\_name*  
FROM *table\_name*;

与

SELECT \* FROM *table\_name*;

SELECT Column实例

下面的SQL语句从“Websites”表中选取“name”和“country”列：

实例：

SELECT name,country FROM Websites;

SELECT \*实例

下面的SQL语句从“Websites”表中选取所有列

实例：

SELECT \* FROM Websites;

SQL SELECT DISTINCT语句

SELECT DISTINCT语句用于返回唯一不同的值

SQL SELECT DISTINCY语句

在表中，一个趔可能会包含多个重复值，有时也许希望仅仅列出不同的值。

DISTINCT关键词用于返回唯一不同的值

SQL SELECT DISTINCT语法

SELECT DISTINCT *column\_name*,*column\_name*  
FROM *table\_name*;

SQL WHERE子句

WHERE子句用于提取那些满足指定条件的记录

SQL WHERE语法

SELECT *column\_name*,*column\_name*  
FROM *table\_name*  
WHERE *column\_name operator value*;

WHERE子句实例

下面的SQL语句从”Websites”表中选取国家为“CN”所有网站

实例：

SELECT \* FROM Websites WHERE country='CN';

文本字段 vs 数值字段

SQL使用单引号来环绕文本值（大部分数据库系统也接受双引号）

如果是数值字段，不用使用引号

实例：

SELECT \* FROM Websites WHERE id=1;

WHERE子句中的运算符

下面的运算符可以在WHERE子句中使用：

|  |  |
| --- | --- |
| 运算符 | 描述 |
| = | 等于 |
| <> | 不等于。注释：在SQL的一些版本中，该操作符可被写成！= |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| BETWEEN | 在某个范围内 |
| LIKE | 搜索某种模式 |
| IN | 指定针对某个列的多个可能值 |

WHERE子句

搜索empno等于7900的数据：

Select \* from emp where empno=7900;

WHERE+条件（筛选行）

条件：列，比较运算符，值

比较运算符包涵：=><>=,<=,!=,<>表示（不等于）

Select \* from emp where ename='SMITH';

例子中的SMITH用单引号引起来，表示是字符串，字符串要区分大小写

逻辑运算

And:与 同时满足两个条件的值

Select \* from emp where sal > 2000 and sal < 3000;

查询EMP表中SAL列中大于2000小于3000的值

Or:或 满足其中一个条件的值

Select \* from emp where sal > 2000 or comm > 500;

查询EMP表中SAL大于2000或COMM大于500的值

Not:非 满足不包含该条件的值

select \* from emp where not sal > 1500;

查询EMP表中sal小于等于1500的值

逻辑运算的优先级：

（） not and or

特殊条件

1. 空值判断： is null

Select \* from emp where comm is null;

查询emp表中comm列中的空值

1. Between and(在 之间的值）

Select \* from emp where sal between 1500 and 3000;

查询emp表中SAL列中大于1500的小于3000的值

注意：大于等于1500且小于等于3000,1500为下限，3000为上限，下限在前，上限在后，查询的范围包含有上下限的值。

1. IN

Select \* from emp where sal in (5000,3000,1500);

查询EMP表SAL列中等于5000,3000,1500的值

1. LIKE

LIKE模糊查询

Select \* from emp where ename like 'M%';

查询EMP表中Ename列中有M的值，M为要查询内容中的模糊信息

* %表示多个字值，\_下划线表示一个字符；
* M%:为能配符，正则表达式，表示的意思为模糊查询信息为M开头的
* %M%:表示查询包含M的所有内容
* %M\_：表示查询以M在倒数第二位的所有内容

不带比较运算符的WHERE子句：

Where子句并不一定带比较运算符，当不带运算符时，会执行一个隐式转换。当0时转化为false，1时转化为true。例如：

SELECT studentNO FROM student WHERE 0

则会返回一个空集，因为每一行记录where都返回false。

SELECT studentNO FROM student WHERE 1

返回student表所有行中studentNO列的值。因为每一行记录where都返回true。

SQL AND & OR 运算符

如果第一个条件和第二个条件都成立，则AND运算符显示一条记录

如果第一个条件和第二个条件中只要有一个成立，则OR运算符显示一条记录

AND运算符实例

下面的SQL语句从”Websites”表中选取国家为”CN”且alexa排名大于“50”的所有网站：

实例：

SELECT \* FROM Websites

WHERE country='CN'

AND alexa > 50;

OR运算符实例

下面的SQL语句从“Websites”表中选取国家为“USA”或者“CN”的所有客户

实例：

SELECT \* FROM Websites

WHERE country='USA'

OR country='CN';

结合AND&OR

可以把AND和OR结合起来（使用圆括号来组成复杂的表达式）

下面的SQL语句从“Websites”表中选取alexa排名大于“15”且国家为“CN”或“USA”的所有网站

实例：

SELECT \* FROM Websites

WHERE alexa > 15

AND (country='CN' OR country='USA');

SQL ORDER BY关键字

ORDER BY关键字用于对结果集按照一个列或者多个列进行排序

ORDER BY关键字默认按照升序对记录进行排序，如果需要按照对记录进行排序，可以使用DESC关键字

SQL ORDER BY语法

SELECT *column\_name*,*column\_name*  
FROM *table\_name*  
ORDER BY *column\_name*,*column\_name* ASC|DESC;

ORDER BY实例

下面的SQL语句从“Websites”表中选取所有网站，并按照“alexa”列排序：

实例：

SELECT \* FROM Websites

ORDER BY alexa;

ORDER BY DESC实例

下面的SQL语句从“Websites”表中选取所有网站，并按照“alexa”列降序排序

实例：

SELECT \* FROM Websites

ORDER BY alexa DESC;

ORDER BY多列

下面的SQL语句从“Wensites”表中选取所有网站，并按照“country”和“alexa”列排序

实例：

SELECT \* FROM Websites

ORDER BY country,alexa;

ORDER BY多列的时候，先按照第一个column name排序，在按照第二个column name排序；如上述教程最后一个例子：

* 先将country值这一列排序，同为CN的排前面，同属USA的排后面
* 然后在同属CN的这些多行数据中，再根据alexa值的大小排列。
* ORDER BY排列时，不写明ASC DESC的时候，默认是ASC。

ORDER BY多列的时候，eg:

Order by A,B 这时候都是默认按升序排列

Order by A desc,B 这时候A降序，B升序排列

Order by A,B DESC 这时候A升序，B降序排列

即desc或者asc只对它紧跟着的第一个列名有效，其他不受影响，任然是默认的升序。

SQL INSERT INTO语句

INSERT INTO语句用于向表中插入新纪录

SQL INSERT INTO语法

INSERT INTO语句可以有两种编写形式

第一种形式无需指定要插入数据的列名，只需提供被插入的值即可：

INSERT INTO *table\_name*  
VALUES (*value1*,*value2*,*value3*,...);

第二种形式需要指定列名及被插入的值：

INSERT INTO *table\_name* (*column1*,*column2*,*column3*,...)  
VALUES (*value1*,*value2*,*value3*,...);

INSERT INTO实例

假设我们要向“Websites”表中插入一个新行

可以使用下面的SQL语句：

实例：

INSERT INTO Websites (name, url, alexa, country)

VALUES ('百度','https://www.baidu.com/','4','CN');

Id列是自动更新的，表中的每条记录都有一个唯一的数字。

在指定的列插入数据

下面的SQL语句将插入一个新行，但是只在“name”，“url”和“country”列插入数据（id字段会自动更新）：

实例：

INSERT INTO Websites (name, url, country)

VALUES ('stackoverflow', 'http://stackoverflow.com/', 'IND');

没有指定要插入数据的列名的形式需要列出插入行的每一列数据：

INSERT INTO table\_name

VALUES (value1,value2,value3,...);

Insert into select和select into from的区别：

insert into scorebak select \* from socre where neza='neza' --插入一行,要求表scorebak 必须存在

select \* into scorebak from score where neza='neza' --也是插入一行,要求表scorebak 不存在

SQL UPDATE语句

UPDATE语句用于更新表中已存在的记录

SQL UPDATE语法

UPDATE *table\_name*  
SET *column1*=*value1*,*column2*=*value2*,...  
WHERE *some\_column*=*some\_value*;

WHERE子句规定那条记录或者哪些记录需要更新，如果省略了where子句，所有的记录都将被更新！

SQL UPDATE实例

假设要将“菜鸟教程”的alexa排名更新为5000，country改为USA

使用下面的SQL语句：

实例：

UPDATE Websites

SET alexa='5000', country='USA'

WHERE name='菜鸟教程';

UPDATE警告！

在更新记录时要格外小心！在上面的实例中，如果省略了WHERE子句，如下所示：

UPDATE Websites  
SET alexa='5000', country='USA'

执行以上代码会将Websites表中所有数据的alexa改为5000，country改为USA。执行没有where子句的UPDATE要慎重，在慎重。

SQL DELETE 语句

DELETE语句用于删除表中的行

SQL DELETE语法

DELETE FROM table\_name  
WHERE some\_column=some\_value;

WHERE子句规定哪条记录或者哪些记录需要删除，如果忽略了WHERE子句，所有记录都将被删除！

SQL DELETE实例

假设要从“Websites”表中删除网站名为“百度”且国家为CN的网站。

使用下面的SQL语句：

实例：

DELETE FROM Websites

WHERE name='百度' AND country='CN';

删除所有数据

可以在不删除表的情况下，删除表中所有的行，这意味着表结构，属性，索引将保持不变：

DELETE FROM *table\_name*;  
或  
DELETE \* FROM *table\_name*;

注释：再删除记录时要格外小心，因为不能重来！

SQL关于删除的三个语句：DROP,TRUNCATE,DELETE的区别

DROP:

Drop test;

删除表test，并释放空间，将test删除的一干二净

TRUNCATE

TRUNCATE TEST;

删除表test里的内容，并释放空间，但不删除表的定义，表的结构还在

DELETE

1. 删除指定数据

删除表test中年龄等于30的且国家为US的数据

DELETE FROM test WHERE age=30 AND country='US';

1. 删除整个表

仅删除表test内的所有内容，保留表的定义，不释放空间

DELETE FROM test 或者 DELETE FROM test;

DELETE \* FROM test 或者 DELETE \* FROM test;

Truncate table命令将快速删除数据表中的所有记录，但保留数据表结构。这种快速删除与delete from数据表的删除全部数据表记录不一样，delete命令删除的数据将存储在系统回滚段中，需要的时候，数据可以回滚回复，而truncate命令删除的数据是不可以恢复的

相同点：

Truncate和不带where子句的delete，以及drop都会删除表内的数据。

不同点：

1. truncate和delete只删除数据不删除表的结构（定义），drop语句将删除表的结构被依赖的约束（constrain），触发器（trigger），索引（index）；

依赖于该表的存储过程/函数将保留，但是变为invalid状态。

1. delete语句是dml,这个操作会放到rollback segment中，实物提交之后才生效，如果有相应的trigger，执行的时候将被触发，truncate，drop是ddl,操作立即生效，原数据不放到rollback segment中，不能回滚。操作不触发trigger
2. 速度：一般来说：drop>truncate>delete
3. 安全性：小心使用drop和truncate，尤其没有备份的时候，使用上，想删除部分数据行用delete，注意带上where子句。回滚段要足够大，想删除表，当然要用drop。想保留表而降所有数据删除，如果和事务无关，用truncate即可，如果和事务有关，或想触发trigger，还是用delete。如果是整理表内部的碎片，可以用truncate跟上reuse stroage，再重新导入/插入数据

SQL SELETE TOP,LIMIT,ROWNUM子句

SELECT TOP子句用于规定要但会的记录的数目

SELECT TOP子句对于拥有数千条记录的大型表来说，是非常有用的。

注意：并非所有的数据库系统都支持SELECT TOP语句

SQL语法

SELECT TOP *number*|*percent* *column\_name(s)*  
FROM *table\_name*;

SQL SELECT TOP PERCENT实例

下面的SQL语句从websites表中选取前表分之50的记录：

实例：

SELECT TOP 50 PERCENT \* FROM Websites;

变相返回后N行：

前五行：

select top 5 \* from table

后五行：

select top 5 \* from table order by id desc --desc 表示降序排列 asc 表示升序

SQL LIKE操作符

LIKE操作符用于在WHERE子句中搜索列中的指定模式

SQL LIKE语法

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name* LIKE *pattern*;

SQL LIKE操作符实例

下面的SQL语句选取name以字母“G”开始的所有客户：

实例：

SELECT \* FROM Websites  
WHERE name LIKE 'G%';

提示：“%”符号用于在模式的前后定义通配符（默认字母）

下面的SQL语句先去name以字母“k”结尾的所有客户：

实例：

SELECT \* FROM Websites  
WHERE name LIKE '%k';

下面的SQL语句选取name包含模式“oo”的所有客户：

实例：

SELECT \* FROM Websites  
WHERE name LIKE '%oo%';

通过使用NOT关键字，可以选取不匹配模式的记录

下面的SQL语句选取name不包含模式“oo”的所有客户：

实例：

SELECT \* FROM Websites  
WHERE name NOT LIKE '%oo%';

总结：“%a” --以a结尾的数据

“a%” --以a开头的数据

“%a%” --含有a的数据

“\_a\_” --三位且中间字母是a的

“\_a” --两位且结尾字母是a的

“a\_” --两位且开头字母是a的

SQL通配符

在SQL中，通配符与SQL LIKE操作符一起使用

SQL通配符用于搜索表中的数据

在SQL中，可使用以下通配符

|  |  |
| --- | --- |
| 通配符 | 描述 |
| % | 替代0个或多个字符 |
| \_ | 替代一个字符 |
| [charlist] | 字符列中的任何单一字符 |
| [^charlist]  或  [!charlist] | 不在字符列中的任何单一字符 |

使用SQL % 通配符

下面的SQL语句选取url以字母“https”开始的所有网站：

实例：

SELECT \* FROM Websites  
WHERE url LIKE 'https%';

下面的SQL语句选取url包含模式“oo”的所有网站：

实例：

SELECT \* FROM Websites  
WHERE url LIKE '%oo%';

使用SQL \_ 通配符

下面的SQL语句选取name以一个任意字符开始，然后是“oogle”的所有客户：

实例：

SELECT \* FROM Websites  
WHERE name LIKE '\_oogle';

下面的SQL语句选取name以“G”开始，然后是任意一个字符，然后是“o”,然后是一个任意字符，然后是“le”的所有网站：

实例：

SELECT \* FROM Websites  
WHERE name LIKE 'G\_o\_le';

使用SQL[charlist]通配符

下面的SQL语句选取name以“G”,“F”或“s”开始的所有网站：

实例：

SELECT \* FROM Websites  
WHERE name REGEXP '^[GFs]';

下面的SQL语句选取name不以A到H字母开头的网站：

实例：

SELECT \* FROM Websites  
WHERE name REGEXP '^[^A-H]';

SQL IN操作符

IN操作符允许在WHERE子句中规定多个值

SQL IN语法

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name* IN (*value1*,*value2*,...);

IN操作符实例

下面的SQL语句选取name为“Google”或“菜鸟教程”的所有网站：

实例：

SELECT \* FROM Websites  
WHERE name IN ('Google','菜鸟教程');

IN与 = 的异同

相同点：均在WHERE中使用作为筛选条件之一，均是等于的含义

不同点：IN可以规定多个值，等于规定一个值

IN

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (value1,value2,...);

=

SELECT column\_name(s)

FROM table\_name

WHERE column\_name=value1;

In与 = 的转换

select \* from Websites where name in ('Google','菜鸟教程');

可以转换成 = 的表达：

select \* from Websites where name='Google' or name='菜鸟教程'

SQL BETWEEN操作符

BETWEEN操作符选取介于两个值之间的数据范围内的值。这些值可以是数值，文本或者日期。

SQL BETWEEN语法

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name* BETWEEN *value1* AND *value2;*

BETWEEN操作符实例

下面的SQL语句选取alexa介于1和20之间的所有网站：

实例：

SELECT \* FROM Websites  
WHERE alexa BETWEEN 1 AND 20;

NOT BETWEEN操作符实例

如需显示不在上面实例范围内的网站，使用NOT BETWEEN:

实例：

SELECT \* FROM Websites  
WHERE alexa NOT BETWEEN 1 AND 20;

带有IN的BETWEEN操作符实例

下面的SQL语句选取alexa介于1和20之间但country不为USA和IND的所有网站：

实例：

SELECT \* FROM Websites  
WHERE (alexa BETWEEN 1 AND 20)  
AND country NOT IN ('USA', 'IND');

带有文本值的BETWEEN操作符实例

下面的SQL语句选取name以介于“A”和“H”之间字母来时的所有网站：

实例：

SELECT \* FROM Websites  
WHERE name BETWEEN 'A' AND 'H';

带有日期值的BETWEEN操作符实例

下面的SQL语句选取date介于“2016-05-10”和“2016-05-14”之间的所有访问记录：

实例：

SELECT \* FROM access\_log  
WHERE date BETWEEN '2016-05-10' AND '2016-05-14';

注意：在不同的数据库中，BETWEEN操作符会产生不同的结果

在某些数据库中，BETWEEN选取介于两个值之间但不包括两个测试值的字段

在某些数据库中，BETWEEN选取介于两个值之间包括两个测试值的字段。

在某些数据库中，BETWEEN选取介于两个值之间且包括第一个测试值但不包括最后一个测试值的字段

因此，检查数据库是如何处理BETWEEN操作符！

SQL别名

通过使用SQL，可以为表名称或列名称指定别名

基本上，创建别名是为了让列名称的可读性更强。

列的SQL别名语法

SELECT *column\_name* AS *alias\_name*  
FROM *table\_name;*

表的SQL别名语法

SELECT *column\_name(s)*  
FROM *table\_name* AS *alias\_name;*

列的别名实例

下面的SQL语句制定了两个别名，一个是name列的别名，一个是country列的别名。

提示：如果列名称包含空格，要求使用双引号或者方括号：

实例：

SELECT name AS n, country AS c  
FROM Websites;

在下面的SQL语句中，把三个列（url,alexa和country）结合在一起，并创建一个名为“site\_info”的别名：

实例：

SELECT name, CONCAT(url, ', ', alexa, ', ', country) AS site\_info  
FROM Websites;

表的别名实例

下面的SQL语句选取“菜鸟教程”的所访问记录，使用“Websites”和“access\_log”表，并分别为他们指定表明“w”和“a”：

实例：

SELECT w.name, w.url, a.count, a.date   
FROM Websites AS w, access\_log AS a   
WHERE a.site\_id=w.id and w.name="菜鸟教程";

不带别名的相同的SQL语句：

SELECT Websites.name, Websites.url, access\_log.count, access\_log.date   
FROM Websites, access\_log   
WHERE Websites.id=access\_log.site\_id and Websites.name="菜鸟教程";

在下面的情况下，使用别名很有用：

* 在查询中涉及超过一个表
* 在查询中使用了函数
* 列名称很长或者可读性差
* 需要把两个列或者多个列结合在一起

SQL连接（JOIN）

SQL JOIN子句用于把来自两个或多个表的行结合起来，基于这些表之间的共同字段。

最常见的JOIN类型：SQL INNER JOIN。 SQL INNER JOIN从多个表中返回满足JOIN条件的所有行。

实例：

SELECT Websites.id, Websites.name, access\_log.count, access\_log.date  
FROM Websites  
INNER JOIN access\_log  
ON Websites.id=access\_log.site\_id;

不同的SQL JOIN

可以累出不同的SQL JOIN类型：

* INNER JOIN:如果表中有至少一个匹配，则返回行
* LEFT JOIN:即使右表中没有匹配，也从左表返回所有的行
* RIGHT JOIN:即使左表中没有匹配，也从右表返回所有的行
* FULL JOIN:只要其中一个表中存在匹配，则返回行

SQL INNER JOIN关键字

INNER JOIN关键字在表中存在至少一个匹配时返回行

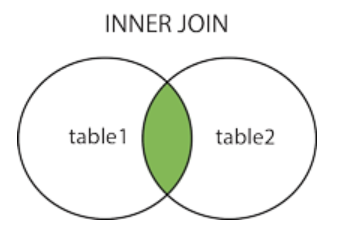
SQL INNER JOIN语法

SELECT *column\_name(s)*  
FROM *table1*  
INNER JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;

或

SELECT *column\_name(s)*  
FROM *table1*  
JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;

注释：INNER JOIN与JOIN是相同的



SQL INNER JOIN实例

下面的SQL语句将返回所有网站的访问记录：

实例：

SELECT Websites.name, access\_log.count, access\_log.date  
FROM Websites  
INNER JOIN access\_log  
ON Websites.id=access\_log.site\_id  
ORDER BY access\_log.count;

注释：INNER JOIN关键字在表中存在至少一个匹配时返回行，如果“Websites”表中的行在“access\_log”中没有匹配，则不会列出这些行。

在使用join时，on和where条件的区别如下：

* On条件是在生产临时时使用的条件，不管on中的条件是否为真，都会返回左边表中的记录。
* Where条件是在临时表生成好后，再对临时表进行过滤的条件。这是已经没有left join的含义（必须返回左边表的记录）了，条件不为真的就全部过滤掉。

SQL LEFT JOIN关键字

LEFT JOIN关键字从左表（table1）返回所有的行，即使右表（table2）中没有匹配，如果右表中没有匹配，则结果为NULL.

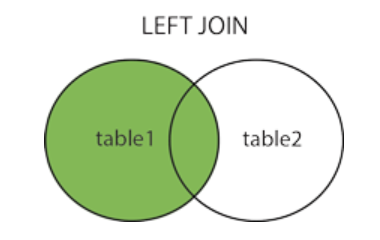
SQL LEFT JOIN语法

SELECT *column\_name(s)*  
FROM *table1*  
LEFT JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;

或

SELECT *column\_name(s)*  
FROM *table1*  
LEFT OUTER JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;

注释：在某些数据库中，LEFT JOIN称为LEFT OUTER JOIN



SQL LEFT JOIN实例

下面的SQL语句将返回所有网站及他们的访问量

以下实例中把Websites作为左表，access\_log作为右表：

实例：

SELECT Websites.name, access\_log.count, access\_log.date  
FROM Websites  
LEFT JOIN access\_log  
ON Websites.id=access\_log.site\_id  
ORDER BY access\_log.count DESC;

注释：LEFT JOIN 关键字从左表返回所有的行，即使右表中没有匹配

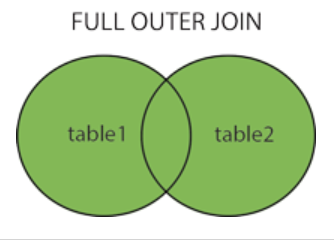
SQL FULL OUTER JOIN关键字

FULL OUTER JOIN关键字只要左表（table1）和右表（table2）其中一个表中存在匹配，则返回行。

FULL OUTER JOIN关键字结合了LEFT JOIN和RIGHT JOIN的结果

SQL FULL OUTER JOIN语法

SELECT *column\_name(s)*  
FROM *table1*  
FULL OUTER JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;



SQL FULL OUTER JOIN实例

下面的SQL语句选取所有网站访问记录

实例：

SELECT Websites.name, access\_log.count, access\_log.date  
FROM Websites  
FULL OUTER JOIN access\_log  
ON Websites.id=access\_log.site\_id  
ORDER BY access\_log.count DESC;

注释：FULL OUTER JOIN关键字返回左表和右表的所有的行。如果“Websites”表中的行在“access\_log”中没有匹配或者“access\_log”表中的行在“Websites”表中没有匹配，也会列出这些行。

SQL UNION操作符

UNION操作符用于合并两个或多个SELECT语句的结果集

注意，UNION内部的每个SELECT语句必须拥有相同数量的列，列也必须拥有相似的数据类型。同时，每个SELECT语句中的列的顺序必须相同。

SQL UNION语法

SELECT *column\_name(s)* FROM *table1*  
UNION  
SELECT *column\_name(s)* FROM *table2*;

注释：默认的，UNION操作符选取不同的值，如果允许重复的值，使用UNION ALL

SQL UNION ALL语法

SELECT *column\_name(s)* FROM *table1*  
UNION ALL  
SELECT *column\_name(s)* FROM *table2*;

注释：UNION结果集中的列名总是等于UNION中第一个SELECT语句中的列名

SQL UNION实例

下面的SQL语句从“Websites”和“apps”表中选取所有不同的country（只有不同的值）：

实例：

SELECT country FROM Websites  
UNION  
SELECT country FROM apps  
ORDER BY country;

注释：UNION不能用于列出两个表中所有的country。如果一些网站和APP来自同一个国家，每个国家只会列出一次。UNION只会选取不同的值。使用UNION ALL来选取重复的值

SQL UNION ALL实例

下面的SQL语句使用UNION ALL从“Websites”和“apps”表中选取所有的country（也有重复的值）：

实例：

SELECT country FROM Websites  
UNION ALL  
SELECT country FROM apps  
ORDER BY country;

带有WHERE的SQL UNION ALL

下面的SQL语句使用UNION ALL从“Websites”和“apps”表中选取所有的中国（CN）的数据（也有重复的值）：

实例：

SELECT country, name FROM Websites  
WHERE country='CN'  
UNION ALL  
SELECT country, app\_name FROM apps  
WHERE country='CN'  
ORDER BY country;

select country from websites union select country from apps;

连接两个表的查询结果集，重复度不显示

select country from websites union all select country from apps order by country;

连接两个个表的查询结果集，显示重复

select country,name from websites where country = 'CN' union all

select country,app\_name from apps where country='CN' order by name;

通过where条件查询结果，连接两个表的结果集，并根据名字排序

使用UNION命令时需要注意，只能在最后使用一个ORDER BY命令，是将两个查询结果合在一起之后，在进行排序！绝对不能写两个ORDER BY命令

另外，在使用ORDER BY排序时，注意两个结果的别名保持一致，使用别名排序很方便。当然也可以使用列数

ORDER BY除了可以对指定的字段进行排序，还可以使用函数进行排序：

order by abs(a);

ORDER BY只能当前SQL查询结果进行排序，如要对UNION ALL出来的结果进行排序，需要先做集合：

select aa.\* from

(select country,name from websites where country = 'CN'

union all select country,app\_name from apps where country='CN' ) aa

order by aa.name;

SQL SELECT INTO语句

SELECT INTO语句从一个表复制数据，然后把数据插入到另一个新表中

SQL SELECT INTO语法

可以复制所有的列插入到新表中：

SELECT \*  
INTO *newtable* [IN *externaldb*]  
FROM *table1;*

或者只复制希望的列插入到新表中：

SELECT *column\_name(s)*  
INTO *newtable* [IN *externaldb*]  
FROM *table1;*

提示：新表将会使用SELECT语句中定义的列名称和类型进行创建，可以使用AS子句来应用新名词

SQL SELECT INTO实例

创建Websites的备份复件：

SELECT \*  
INTO WebsitesBackup2016  
FROM Websites;

只复制一些列插入新表中：

SELECT name, url  
INTO WebsitesBackup2016  
FROM Websites;

只复制中国的网站插入到新表中：

SELECT \*  
INTO WebsitesBackup2016  
FROM Websites  
WHERE country='CN';

复制多个表中的数据插入到新表中：

SELECT Websites.name, access\_log.count, access\_log.date  
INTO WebsitesBackup2016  
FROM Websites  
LEFT JOIN access\_log  
ON Websites.id=access\_log.site\_id;

提示：SELECT INTO语句可用于通过另一种模式创建一个新的空表。只需要添加促使查询没有数据返回的WHERE子句即可：

SELECT \*  
INTO *newtable*  
FROM *table1*  
WHERE 1=0;

SQL INSERT INTO SELECT语句

INSERT INTO SELECT语句从一个表复制数据，然后把数据插入到一个已存在的表中，目标表中任何已存在的行都不会受影响。

SQL INSERT INTO SELECT语法

可以从一个表中复制所有的列插入到另一个已存在的表中：

INSERT INTO *table2*  
SELECT \* FROM *table1;*

或者可以只复制希望的列插入到另一个已存在的表中：

INSERT INTO *table2*  
*(column\_name(s))*  
SELECT *column\_name(s)*  
FROM *table1;*

SQL INSERT INTO SELECT实例

复制“apps”中的数据插入到“Websites”中：

实例：

INSERT INTO Websites (name, country)  
SELECT app\_name, country FROM apps;

只复制QQ的APP到“Websites”中：

实例：

INSERT INTO Websites (name, country)  
SELECT app\_name, country FROM apps  
WHERE id=1;

Select into from和insert into select都是用来复制表

两者的主要区别为：select into from要求目标表不存在，因为在插入时会自动创建，insert into select from要求目标表存在

1. 复制表结构及其数据：

create table table\_name\_new as select \* from table\_name\_old

1. 只复制表结构：

create table table\_name\_new as select \* from table\_name\_old where 1=2;

或者：

create table table\_name\_new like table\_name\_old

1. 只复制表数据：

如果两个表结构一样：

insert into table\_name\_new select \* from table\_name\_old

如果两个表结构不一样：

insert into table\_name\_new(column1,column2...) select column1,column2... from table\_name\_old

SQL CREATE DATABASE语句

CREATE DATABASE语句用于创建数据库

SQL CREATE DATABASE语法

CREATE DATABASE *dbname*;

*SQL CREATE DATABASE实例：*

*下面的SQL语句创建一个名为“my\_db”的数据库：*

CREATE DATABASE my\_db;

数据库表可以通过CREATE TABLE语句来添加

SQL CREATE TABLE语句

CREATE TABLE语句用于创建数据库的表

标有行和列组成，每个表都必须有个表名。

SQL CREATE TABLE语法

CREATE TABLE *table\_name*  
(  
*column\_name1 data\_type*(*size*),  
*column\_name2 data\_type*(*size*),  
*column\_name3 data\_type*(*size*),  
....  
);

Column\_name参数规定表中列的名称

Data\_type参数规定列的数据类型（例如varchar,integer,decimal,date等等）

Size参数规定表中列的最大长度

SQL CREATE TABLE实例

创建一个名为“Persons”的表，包含五列：PersonID,LastName,FirstName,Address和City

使用下面的CREATE TABLE语句：

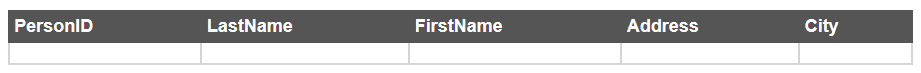
实例：

CREATE TABLE Persons  
(  
PersonID int,  
LastName varchar(255),  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
);

PersonID列的数据类型是int,包含整数

LastName,FirstName,Address和City列的数据类型是varchar，包含字符，且这些字段的最大长度为255个字符。

空的“Persons”表如下所示：



提示：可使用INSERT INTO语句向空表写入数据

SQL约束

用于规定表中的数据规则

如果存在违反约束的数据行为，行为会被约束中止。

约束可以在创建表时规定（通过CREATE TABLE语句），或者在创建表之后规定（通过ALTER TABLE语句）

SQL CREATE TABLE+CONSTRAINT语法

CREATE TABLE *table\_name*  
(  
*column\_name1 data\_type*(*size*) *constraint\_name*,  
*column\_name2 data\_type*(*size*) *constraint\_name*,  
*column\_name3 data\_type*(*size*) *constraint\_name*,  
....  
);

在SQL中，有如下约束：

* NOT NULL -指示某列不能存储NULL值
* UNIQUE -保证某列的每行必须有唯一的值
* PRIMARY KEY -NOT NULL和UNIQUE的结合。确保某列（或两个列多个列的结合）有唯一标识，有助于更容易更快速的找到表中的一个特定的记录
* FOREIGN KEY -保证一个表中的数据匹配另一个表中的值的参照完整性
* CHECK -保证列中的值符合指定的条件
* DEFAULT -规定没有给列赋值时的默认值

SQL NOT NULL约束

NOT NULL约束强制列不接受NULL值

NOT NULL约束强制字段始终包含值。如果不向字段添加值，就无法插入新纪录或者更新记录。

下面的SQL强制“ID”列，“LastName”列以及“FirstName”列不接受NULL值：

实例：

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255) NOT NULL,

Age int

);

添加NOT NULL约束

在一个已创建的表的“Age”字段中添加NOT NULL约束如下所示：

实例：

ALTER TABLE Persons

MODIFY Age int NOT NULL;

删除NOT NULL约束

字啊一个已创建的表的“Age”字段中删除NOT NULL约束如下所示：

实例：

ALTER TABLE Persons

MODIFY Age int NULL;

SQL UNIQUE 约束

UNIQUE约束唯一标识数据库表中的每条记录

UNIQUE和PRIMARY KEY约束均为列或列集合提供了唯一性的保证

PRIMARY KEY约束拥有自动定义的UNIQUE约束

注意：每个表可以有多个UNIQUE约束，但是每个表只能有一个PRIMARY KEY约束

CREATE TABLE时的SQL UNIQUE约束

下面的SQL在“Persons”表创建时在“P\_Id”列上创建UNIQUE约束：

CREATE TABLE Persons  
(  
P\_Id int NOT NULL UNIQUE,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

如需命名UNIQUE约束，并定义多个列的UNIQUE约束，使用下面的SQL语法：

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT uc\_PersonID UNIQUE (P\_Id,LastName)  
)

ALTER TABLE时的SQL UNIQUE约束

当表已被创建时，如需在“P\_Id”列创建UNIQUE约束，使用下面的SQL：

ALTER TABLE Persons  
ADD UNIQUE (P\_Id)

如需命名UNIQUE约束，并定义多个列的UNIQUE约束，使用下面的SQL语法：

ALTER TABLE Persons  
ADD CONSTRAINT uc\_PersonID UNIQUE (P\_Id,LastName)

撤销UNIQUE约束

如需撤销UNIQUE约束，使用下面的SQL：

ALTER TABLE Persons  
DROP CONSTRAINT uc\_PersonID

create table tb2(

tb2\_id int unique,

tb2\_name varchar(20),

tb2\_age int,

unique(tb2\_name)

);

select \* from tb2;

insert into tb2(tb2\_id,tb2\_name,tb2\_age) values (1,'张三',20);

--违反唯一约束

insert into tb2 values(2,'张三',25);

--建表时，创建约束，有约束名

create table tb3(

tb3\_id int ,

tb3\_name varchar(20),

tb3\_age int,

constraint no\_id unique (tb3\_id)

);

insert into tb3 values (1,'张三',20);

insert into tb3(tb3\_id,tb3\_age) values(2,24);

select \* from tb3;

--已经有了tb3\_id为1的行记录，再次插入，违反唯一约束

insert into tb3(tb3\_id,tb3\_name,tb3\_age) values(1,'李四',25);

--给tb2表添加主键约束，主键名为：pk\_id

alter table tb3 add constraint pk\_id primary key (tb3\_id);

--给tb3\_name添加唯一约束

alter table tb3 add constraint un\_name unique (tb3\_name);

--已存在姓名为张三的记录，违反唯一约束

insert into tb3 values (3,'张三',26);

--删除约束后，允许存在多个tb3\_name为张三的记录

insert into tb3 values (3,'张三',26);

SQL PRIMARY KEY约束

PRIMARY KEY约束唯一标识数据库表中的每条记录

主键必须包含唯一的值

主键列不能包含NULL值

每个表都应该有一个主键，并且每个表只能有一个主键

CREATE TABLE时的SQL PRIMARY KEY约束

下面的SQL在“Persons”表创建时在“P\_Id”列上创建PRIMARY KEY约束：

CREATE TABLE Persons  
(  
P\_Id int NOT NULL PRIMARY KEY,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

如需命名PRIMARY KEY约束，并定义多个列的PRIMARY KEY约束，使用下面的SQL语法：

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT pk\_PersonID PRIMARY KEY (P\_Id,LastName)  
)

注释：在上面的实例中，只有一个主键PRIMARY KEY(pk\_PersonID)。然鹅，pk\_PersonID的值是由两个列（P\_Id和LastName）组成的

ALTER TABLE时的SQL PRIMARY KEY约束

当表已被创建时，如需在“P\_Id”列创建PRIMARY KEY约束，使用下面的SQL:

ALTER TABLE Persons  
ADD PRIMARY KEY (P\_Id)

如需命名PRIMARY KEY约束，并定义多个列的PRIMARY KEY约束，使用下面的SQL语法：

ALTER TABLE Persons  
ADD CONSTRAINT pk\_PersonID PRIMARY KEY (P\_Id,LastName)

注释：如果使用ALTER TABLE语句添加主键，必须把主键声明为不包含NULL值（在表首次创建时）

撤销PRIMARY KEY约束

如需撤销PRIMARY KEY约束，使用下面的SQL：  
ALTER TABLE Persons  
DROP CONSTRAINT pk\_PersonID

SQL FOREIGN KEY约束

一个表中的FOREIGN KEY指向另一个表中的UNIQUE KEY（唯一约束的键）

通过一个实例来解释外键，看下面两个表：

“Persons”表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_Id | LastName | FirstName | Address | City |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendon | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

“Orders”表：

|  |  |  |
| --- | --- | --- |
| O\_Id | OrderNo | P\_Id |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

注意：

“Orders”表中的“P\_Id”列指向“Persons”表中的“P\_Id”列。

“Persons”表中的“P\_Id”列是“Persons”表中的PRIMARY KEY

“Orders”表中的“P\_Id”列是“Orders”表中的FOREIGN KEY

FOREIGN KEY约束拥有预防破坏表之间连接的行为

FOREIGN KEY约束也能防止非法数据列插入外键列，因为他必须是他指向的那个表中的值之一

CREATE TABLE时的SQL FOREIGN KEY约束

下面的SQL在“Orders”表创建时在“P\_Id”列上创建FOREIGN KEY约束：

CREATE TABLE Orders  
(  
O\_Id int NOT NULL PRIMARY KEY,  
OrderNo int NOT NULL,  
P\_Id int FOREIGN KEY REFERENCES Persons(P\_Id)  
)

如需命名FOREIGN KEY约束，并定义多个列的FOREIGN KEY约束，使用下面的SQL语法：

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
PRIMARY KEY (O\_Id),  
CONSTRAINT fk\_PerOrders FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)  
)

ALTER TABLE时的SQL FOREIGN KEY约束

当“Orders”表已被创建时，如需咋“P\_Id”列创建FOREIGN KEY约束，使用下面的SQL:

ALTER TABLE Orders  
ADD FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)

如需命名FOREIGN KEY约束，如需在“P\_Id”列创建FOREIGN KEY约束，使用下面的SQL:

ALTER TABLE Orders  
ADD CONSTRAINT fk\_PerOrders  
FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)

撤销FOREIGN KEY约束

如需撤销FOREIGN KEY约束，使用下面的SQL:

ALTER TABLE Orders  
DROP CONSTRAINT fk\_PerOrders

创建外键约束的方式

在创建表的时候指定外键约束

CREATE TABLE 表名

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT 外键约束名 FOREIGN KEY (column1,column2,... column\_n)

REFERENCES 外键依赖的表 (column1,column2,...column\_n)

ON DELETE CASCADE--级联删除

);

在创建表后增加外键约束

ALTER TABLE 表名

ADD CONSTRAINT 外键约束名

FOREIGN KEY (column1, column2,...column\_n)

REFERENCES 外键所依赖的表 (column1,column2,...column\_n)

ON DELETE CASCADE;--级联删除

使用工具plsql来新增外键约束

注意：再创建外键约束时，必须先创建外键约束所依赖的表，并且该列为该表的主键

Oracle数据库中，对指定外键的表进行增删改的情况：

子表：谁创建外键谁就是子表

父表：这个外键所依赖的表

create table father(

f\_no number(2));--插入数据1 2 3 4

create table son(

s\_no number(2),

f\_no number(2))--插入两列1 2 3 4

--删除父表数据

delete from father where f\_no=1 --报错违反约束

--删除父表 drop table father --报错违反约束

--更新父表数据

--禁用约束可以更新ALTER TABLE son disable constraint f\_f;

update father set f\_no =7 where f\_no=1--报错违反约束

--对父表插入数据

insert into father values(6)--可以插入

--删除子表数据

delete from son where s\_on=1--可以删除

--删除子表

drop table son--可以删除

--更新子表数据

update son set s\_no=6 where s\_no=1 --可以更新

update son set f\_no=6 where s\_no=6 --可以更新

--对子表插入数据 insert into son values(7,7)--父表没有7，违反约束

insert into son values(1,1)--父表有1可以插入

insert into son values(8,1)--父表有1可以插入

总结：

一：删除时，未指定cascade时

1. 删除父表/数据

* 因为子表与父表一一对应，删除父表数据时，需要先把子表对应数据删除否则无法删除
* 同理，删除表的时候，也需要先删除子表在删除父表

解决方案：

* 指定cascade，删除父表，数据

CASCADE指当删除主表中被引用列的数据时，级联删除子表中相应的数据行

* 禁用约束（子表的外键约束）

  ALTER TABLE 表名 disable constraint 约束名;

1. 删除子表：可以了删除子表或者数据不报错

二：更新时

1. 更新父表会违反约束
2. 可以更新子表
3. 没有针对约束的级联更新
4. 插入时
5. 父表可以插入
6. 子表插入会违反约束

SQL CHECK约束

CHECK约束用于限制列中的值的范围

如果对单个列定义CHECK约束，那么该列只允许特定的值

如果对一个表定义CHECK约束，那么此约束会基于行中其他列的值在特定的列中对值进行限制

CREATE TABLE时的SQL CHECK约束

下面的SQL在“Persons”表创建时在“P\_Id”列上创建CHECK约束。CHECK约束规定“P\_Id”列必须只包含大于0的整数

CREATE TABLE Persons  
(  
P\_Id int NOT NULL CHECK (P\_Id>0),  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

如需命名CHECK约束，并定义多个列的CHECK约束，使用下面的SQL语法：

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT chk\_Person CHECK (P\_Id>0 AND City='Sandnes')  
)

ALTER TABLE时的SQL CHECK约束

当表已被创建时，如需在“P\_Id”列创建CHECK约束，使用下面的SQL:

ALTER TABLE Persons  
ADD CHECK (P\_Id>0)

如需命名CHECK约束，并定义多个列的CHECK约束，使用下面的SQL约束：

ALTER TABLE Persons  
ADD CONSTRAINT chk\_Person CHECK (P\_Id>0 AND City='Sandnes')

撤销CHECK约束

如需撤销CHECK约束，使用下面的SQL：

ALTER TABLE Persons  
DROP CONSTRAINT chk\_Person

SQL DEFAULT约束

DEFAULT约束用于向列中插入默认值

如果没有规定其他的值，那么会将默认值添加到所有的新纪录

CREATE TABLE时的SQL DEFAULT约束

下面的SQL在“Persons”表创建时在“City”列上创建DEFAULT约束：

CREATE TABLE Persons

(

P\_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255) DEFAULT 'Sandnes'

)

通过使用类似GETDATE()这样的函数，DEFAULT约束也可以用于插入系统值。

CREATE TABLE Orders

(

O\_Id int NOT NULL,

OrderNo int NOT NULL,

P\_Id int,

OrderDate date DEFAULT GETDATE()

)

ALTER TABLE时的SQL DEFAULT约束

当表已被创建时，如需在“City”列创建DEFAULT约束，使用下面的SQL:

ALTER TABLE Persons

ADD CONSTRAINT ab\_c DEFAULT 'SANDNES' for City

撤销DEFAULT约束

如需撤销DEFAULT约束，使用下面的SQL：

ALTER TABLE Persons

ALTER COLUMN City DROP DEFAULT

SQL CREATE INDEX语句

CREATE INDEX语句用于在表中创建索引

在不读取整个表的情况下，索引使数据库应用程序可以更快的查找数据

索引

可以在表中创建索引，以便更加快速高效的查询数据

用户无法看到索引，他们只能被用来加速搜索/查询

注释：更新一个包含索引的表需要比更新一个没有索引的表花费更多的时间，这是由于索引本身也需要更新。因此，理想的做法是仅仅在常常被搜索的列（以及表）上面创建索引。

SQL CREATE INDEX语法

在表上创建一个简单的索引，允许使用重复的值：

CREATE INDEX index\_name  
ON table\_name (column\_name)

SQL CREATE UNIQUE INDEX语法

在表上创建一个唯一的索引，不允许使用重复的值：唯一的索引意味着两个行不能拥有相同的索引值

CREATE UNIQUE INDEX index\_name  
ON table\_name (column\_name)

注释：用于创建索引的语法在不同的数据库中不一样，因此检查数据库中创建索引的语法

CREATE INDEX实例

下面的SQL语句在“Persons”表的“LastName”列上创建一个名为“PINDEX”的索引：

CREATE INDEX PIndex  
ON Persons (LastName)

如果希望索引不止一个列，可以在括号中列出这些列的名称，用逗号隔开：

CREATE INDEX PIndex  
ON Persons (LastName, FirstName)

SQL撤销索引，撤销表以及撤销数据库

通过使用DROP语句，可以轻松的删除索引，表和数据库

DROP INDEX语句

DROP INDEX语句用于删除表中的索引

DROP INDEX table\_name.index\_name

DROP TABLE语句

DROP TABLE语句用于删除表

DROP TABLE table\_name

DROP DATABASE语句

DROP DATABASE语句用于删除数据库

DROP DATABASE database\_name

TRUNCATE TABLE语句

如果仅仅需要删除表内的数据，并不删除表本身，使用TRUNCATE TABLE语句：

TRUNCATE TABLE table\_name

SQL ALTER TABLE语句

ALTER TABLE语句用于在已有的表添加，删除或修改列

SQL ALTER TABLE语法：

如需在表中添加列，使用下面的语法：

ALTER TABLE table\_name  
ADD column\_name datatype

如需删除表中的列，使用下面的语法（注意，某些数据库不允许这种在数据库表中删除列的方式）：

ALTER TABLE table\_name  
DROP COLUMN column\_name

要改变表中列的数据类型，使用下面的语法：

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype

SQL ALTER TABLE实例：

“Persons”表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_Id | LastName | FirstName | Address | City |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

现在，在“Persons”表中添加一个名为“DateOfBirth”的列

使用下面的SQL语句：

ALTER TABLE Persons  
ADD DateOfBirth date

注意：新列“DateOfBirth”的类型是date,可以存放日期，数据类型规定列中可以存放的数据的类型

“Persons”表将如下所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P\_Id | LastName | FirstName | Address | City | DateOfBirth |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |  |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |  |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |  |

改变数据类型的实例：

改变“Persons”表中“DateOfBirth”列的数据类型

使用下面的SQL语句：

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year

注意：现在“DateOfBirth”列的类型是year，可以存放2位或4位格式的年份。

DROP COLUMN实例

删除“Person”表中的“DataOfBirth”列

使用下面的语句：

ALTER TABLE Persons  
DROP COLUMN DateOfBirth

现在，“Persons”表将如下所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_Id | LastName | FirstName | Address | City |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

SQL AUTO INCREMENT字段

AUTO-INCREMENT会在新纪录插入表中时生成一个唯一的数字

AUTO INCREMENT字段

通常希望在每次插入新纪录时，自动的创建主键字段的值

可以在表中创建一个auto-increment字段

下面的SQL语句把“Persons”表中的“ID”列定义为auto-increment主键字段：

CREATE TABLE Persons  
(  
ID int IDENTITY(1,1) PRIMARY KEY,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

使用IDENTITY关键字来执行auto-increment任务

在上面的实例中，IDENTITY的开始值是1，每条新纪录递增1

提示：要规定“ID”列以10起始且递增5，把identity改为IDENTITY(10,5)

要在“Persons”表中插入新纪录，不必为“ID”列规定值（会自动添加一个唯一的值）：

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen')

上面的SQL语句会在“Persons”表中插入一条新纪录，“ID”列会被赋予一个唯一的值。“FirstName”列会被设置为“Lars”,“LastName”列会被设置为“Monsen”

给已经存在的column添加自增语法：

ALTER TABLE table\_name CHANGE column\_name column\_name data\_type(size) constraint\_name AUTO\_INCREMENT;

比如：

ALTER TABLE student CHANGE id id INT( 11 ) NOT NULL AUTO\_INCREMENT;

SQL视图

视图是可视化的表

SQL CREATE VIEW语句

在SQL中，视图是基于SQL语句的结果集的可视化的表

视图包含行和列，就像一个真实的表。视图中的字段就是来自一个或多个数据库的真实的表中的字段

可以向视图添加SQL函数，WHERE以及JOIN语句，也可以呈现数据，就像这些数据来自于某个单一的表一样

SQL CREATE VIEW语法

CREATE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

注释：视图总是显示最新的数据，每当用户查询视图时，数据库引擎通过使用视图的SQL语句重建数据。

SQL CREATE VIEW实例

CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName  
FROM Products  
WHERE Discontinued=No

可以像这样查询上面这个视图：

SELECT \* FROM [Current Product List]

Northwind样本数据库的另一个视图会选取“Products”表中所有单位价格高于平均单位价格的产品：

CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName,UnitPrice  
FROM Products  
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)

可以像这样查询上面这个视图：

SELECT \* FROM [Products Above Average Price]

Northwind样本数据库的另一个视图会计算在1997年每个种类的销售总数，视图会从另一个名为“Products Sales for 1997”的视图那里选取数据：

CREATE VIEW [Category Sales For 1997] AS  
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales  
FROM [Product Sales for 1997]  
GROUP BY CategoryName

可以像这样查询上面这个视图：

SELECT \* FROM [Category Sales For 1997]

也可以像查询添加条件，仅仅查看“Beverages”类的销售总数：

SELECT \* FROM [Category Sales For 1997]  
WHERE CategoryName='Beverages'

SQL更新视图

可以使用下面的语法来更新视图：

SQL CREATE OR REPLACE VIEW语法

CREATE OR REPLACE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

在“Current Product List”视图添加“Category”列，通过下列SQL更新视图：

CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName,Category  
FROM Products  
WHERE Discontinued=No

ALTER VIEW [ schema\_name . ] view\_name [ ( column [ ,...n ] ) ]

[ WITH <view\_attribute> [ ,...n ] ]

AS select\_statement

[ WITH CHECK OPTION ] [ ; ]

<view\_attribute> ::=

{

[ ENCRYPTION ]

[ SCHEMABINDING ]

[ VIEW\_METADATA ]

}

* Schema\_name:视图所属架构的名称
* View\_name:需要更改的视图
* Column：将成为指定视图的一部分的一个或多个列的名称（以逗号分隔）

SQL撤销视图

可以通过DROP VIEW命令来删除视图

DROP VIEW view\_name

视图的作用

1. 视图隐藏了底层的表结构，简化了数据访问操作，客户端不在需要知道底层表的结构及其之间的关系
2. 视图提供了一个统一访问数据的接口（即可以允许用户通过视图访问数据的安全机制，而不授予用户直接访问底层表的权限）
3. 从而加强了安全性，使用户只能看到视图所显示的数据
4. 视图还可以被嵌套，一个视图中可以嵌套另一个视图。

SQL DATE函数

SQL日期

当我们处理日期时，最难的任务就是确保所插入的日期的格式，与数据库中日期列的格式相匹配

只要数据包含的只是日期部分，运行查询就不会出问题。但是，如果涉及时间部分，情况就有点复杂

Date函数

下面表格列出来重要的内建日期函数

|  |  |
| --- | --- |
| 函数 | 描述 |
| GETDATE() | 返回当前的日期和时间 |
| DATEPART() | 返回日期/时间的单独部分 |
| DATEADD() | 在日期中添加或减去指定的事件间隔 |
| DATEDIFF() | 返回两个日期之间的时间 |
| CONVERT() | 用不同的格式显示日期/时间 |

SQL Date数据类型

* DATE -格式：YYYY-MM-DD
* DATETIME -格式：YYYY-MM-DD HH:MM:SS
* SMALLDATETIME -格式：YYYY-MM-DD HH:MM:SS
* TIMESTAMP -格式：唯一的数字

注释：当在数据库创建一个新表时，需要为列选择数据类型

SQL日期处理

如果不涉及时间部分，可以轻松的比较两个日期

假设有如下的“Orders”表：

|  |  |  |
| --- | --- | --- |
| OrderId | ProducrName | OrderDate |
| 1 | Geitost | 2008-11-11 |
| 2 | Camembert Pierrot | 2008-11-09 |
| 3 | Mozzarella Giovanni | 2008-11-11 |
| 4 | Mascarpone Fabioli | 2008-10-29 |

从表中选取OrderDate为“2008-11-11”的记录

使用下面的SELECT语句：

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

结果集如下所示：

|  |  |  |
| --- | --- | --- |
| OrderId | ProducrName | OrderDate |
| 1 | Geitost | 2008-11-11 |
| 3 | Mozzarella Giovanni | 2008-11-11 |

假设“Orders”表如下所示（注意“OrderDate”列中的时间部分）：

|  |  |  |
| --- | --- | --- |
| OrderId | ProducrName | OrderDate |
| 1 | Geitost | 2008-11-11 13:23:44 |
| 2 | Camembert Pierrot | 2008-11-09 15:45:21 |
| 3 | Mozzarella Giovanni | 2008-11-11 11:12:01 |
| 4 | Mascarpone Fabioli | 2008-10-29 14:56:59 |

如果使用和上面一样的SELECT语句：

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

或

SELECT \* FROM Orders WHERE OrderDate='2008-11-11 00：00：00'

那么将得不到结果，因为表中没有“2008-11-11 00:00:00”日期，如果没有时间部分，默认时间为00:00:00

提示：如果希望使查询简单且更易维护，不要再日期使用时间部分

SQL NULL值

NULL值代表遗漏的未知数据

默认的，表的列可以存放NULL值

SQL NULL值

如果表中的某个列是可选的，可以在不向该列添加值的情况下插入新纪录或更新已有的记录，这意味着该字段将以NULL值保存

NULL值的处理方式与其他值不同

NULL用作未知的或不适用的值的占位符

注释：无法比较NULL和0，他们不等价

SQL的NULL值处理

“Persons”表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_Id | LastName | FirstName | Address | City |
| 1 | Hansen | Ola |  | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari |  | Stavanger |

假如“Persons”表中的“Address”列是可选的，意味着如果在“Address”列插入一条不带值的记录，“Address”列会使用NULL值保存

使用IS NULL和IS NOT NULL操作符

SQL IS NULL

必须使用IS NULL操作符：  
SELECT LastName,FirstName,Address FROM Persons  
WHERE Address IS NULL

结果集如下所示：

|  |  |  |
| --- | --- | --- |
| LastName | FirstName | Address |
| Hansen | Ola |  |
| Pettersen | Kari |  |

提示：始终使用IS NULL来查找NULL值

SQL IS NOT NULL

如何选取不带有NULL值的记录

必须使用IS NOT NULL操作符：

SELECT LastName,FirstName,Address FROM Persons  
WHERE Address IS NOT NULL

结果集如下所示：

|  |  |  |
| --- | --- | --- |
| Svendson | Tove | Borgvn 23 |

SQL NULL函数

“Products”表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_Id | ProductName | UnitPrice | UnitsInStock | UnitsOnOrder |
| 1 | Jarlsberg | 10.45 | 16 | 15 |
| 2 | Mascarpone | 32.56 | 23 |  |
| 3 | Gorgonzola | 15.67 | 9 | 20 |

假如“UnitsOnOrder”是可选的，而且可以包含NULL值

使用下面的SELECT语句：

SELECT ProductName,UnitPrice\*(UnitsInStock+UnitsOnOrder)  
FROM Products

在上面的实例中，如果有“UnitsOnOrder”值是NULL,那么结果是NULL

如果“UnitsOnOrder”是NULL,不会以你选哪个计算，如果值是NULL则ISNULL()返回0：

SELECT ProductName,UnitPrice\*(UnitsInStock+ISNULL(UnitsOnOrder,0))  
FROM Products

--如果alexa列为null值，则赋予0，否则，取原值

select id,name,url,ifnull(alexa,0)from websites;

select id,name,url,COALESCE(alexa,0) from websites;

SQL函数

SQL拥有很多可用于计数和计算的内建函数

SQL Aggregate函数

SQL Aggregate函数计算从列中取得的值，返回一个单一的值

有用的Aggregate函数：

* AVG() -返回平均值
* COUNT() -返回行数
* FIRST() -返回第一个记录的值
* LAST() -返回最后一个记录的值
* MAX() -返回最大值
* MIN() -返回最小值
* SUM() -返回总和

SQL Scalar函数

基于输入值，返回一个单一的值

有用的Scalar函数：

* UCASE() -将某个字段转换为大写
* LCASE() -将某个字段转换为小写
* MID() -从某个文本字段提取字符
* SubString(字段，1，end） -从某个文本字段提取字符
* LEN() -返回某个文本字段的长度
* ROUND() -对某个数值字段进行指导小数位数的四舍五入
* NOW() -返回当前的系统日期和时间
* FORMAT() -格式化某个字段的显示方式

SQL AVG()函数

AVG()函数返回数值列的平均值

SQL AVG()语法

SELECT AVG(column\_name) FROM table\_name

SQL AVG()实例

下面的SQL语句从“access\_log”表的“count”列获取平均值：

实例：

SELECT AVG(count) AS CountAverage FROM access\_log;

下面的SQL语句选择访问量高于平均访问量的“site\_id”和“count”：

SELECT site\_id, count FROM access\_log  
WHERE count > (SELECT AVG(count) FROM access\_log);

SQL COUNT()函数

COUNT()函数返回匹配指定条件的行数

SQL COUNT（column\_name）语法

COUNT函数返回指定列的值的数目（NULL不计入）：

SELECT COUNT(column\_name) FROM table\_name;

SQL COUNT(\*)语法

COUNT(\*)函数返回表中的记录数：

SELECT COUNT(\*) FROM table\_name;

SQL COUNT(DISTINCY column\_name)语法

COUNT(DISTINCY column\_name)函数返回指定列的不同值的数目：

SELECT COUNT(DISTINCT column\_name) FROM table\_name;

SQL COUNT(column\_name)实例

计算“access\_log”表中“site\_id”=3的总访问量：

实例：

SELECT COUNT(count) AS nums FROM access\_log  
WHERE site\_id=3;

SQL COUNT(\*)实例

计算“access\_log”表中总记录数：

实例：

SELECT COUNT(\*) AS nums FROM access\_log;

SQL COUNT(DISTICY column\_name)实例

就散“access\_log”表中不同site\_id的记录数：

实例：

SELECT COUNT(DISTINCT site\_id) AS nums FROM access\_log;

-- 查询所有记录的条数

select count(\*) from access\_log;

-- 查询websites 表中 alexa列中不为空的记录的条数

select count(alexa) from websites;

-- 查询websites表中 country列中不重复的记录条数

select count(distinct country) from websites;

count (表达式）--分组里非空记录数

count (表达式）--分组里非空记录数

count(\*)--所有记录

count(1)--所有记录

count(case job = 'CLERK' then 2 end )--CLERK 人数

count(comm)--有奖金的人数

count(distinct job)--distinct(去重），共有多少种工作

实例：以EMP表为例

select deptno,

count(1) 总人数,

count(case when job ='SALESMAN' then '1' end) 销售人数,

count(case when job ='MANAGER' then '1' end) 主管人数

from emp

group by deptno;--如果不group，会认为所有数据是一组，返回一个数据

SQL FIRST()函数

FIRST()函数返回指定的列中第一个记录的值

SQL FIRST()语法

SELECT FIRST(column\_name) FROM table\_name;

注释：只有MS ACCESS支持FIRST()函数

SELECT TOP 1 *column\_name* FROM *table\_name*ORDER BY *column\_name* ASC;

实例：

SELECT TOP 1 name FROM Websites  
ORDER BY id ASC;

SQL FIRST()实例

下面的SQL语句选取“Websites”表的“name”列中第一个记录的值：

实例：

SELECT name AS FirstSite FROM Websites LIMIT 1;

SQL LAST()函数

LAST()返回指定的列中最后一个记录的值

SQL LAST()语法

SELECT LAST(column\_name) FROM table\_name;

SELECT TOP 1 *column\_name* FROM *table\_name*ORDER BY *column\_name* DESC;

实例：

SELECT TOP 1 name FROM Websites   
ORDER BY id DESC;

SQL LAST()实例

下面的SQL语句选取“Websites”表的“name”列中最后一个记录的值：

实例：

SELECT name FROM Websites  
ORDER BY id DESC  
LIMIT 1;

SQL MAX()函数

MAX()函数返回指定列的最大值

SQL MAX()语法

SELECT MAX(column\_name) FROM table\_name;

SQL MAX()实例

下面的SQL语句从“Websites”表的“alexa”列获取最大值：

实例：

SELECT MAX(alexa) AS max\_alexa FROM Websites;

SQL MIN()函数

MIN()函数返回指定列的最小值

SQL MIN()语法

SELECT MIN(column\_name) FROM table\_name;

SQL MIN()实例

下面的SQL语句从“Websites”表的“alexa”列获取最小值：

实例：

SELECT MIN(alexa) AS min\_alexa FROM Websites;

SQL SUM()函数

SUM()函数返回数值列的总数

SQL SUM()语法

SELECT SUM(column\_name) FROM table\_name;

SQL SUM()实例

下面的SQL语句查找“access\_log”表的“count”字段的总数：

实例：

SELECT SUM(count) AS nums FROM access\_log;

SQL GROUP BY语句

GROUP BY语句用于结合聚合函数，根据一个或多个列对结果集进行分组

SQL GROUP BY语法

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name;

GROUP BY简单应用

统计access\_log各个site\_id的访问量：

实例：

SELECT site\_id, SUM(access\_log.count) AS nums  
FROM access\_log GROUP BY site\_id;

SQL GROUP BY多表连接

下面的SQL语句统计有记录的网站的记录数量：

实例：  
SELECT Websites.name,COUNT(access\_log.aid) AS nums FROM access\_log  
LEFT JOIN Websites  
ON access\_log.site\_id=Websites.id  
GROUP BY Websites.name;

SQL HAVING子句

在SQL中增加HAVING子句原因是，where关键字无法与聚合函数一起使用

HAVING子句可以让我们筛选分组后的各组数据

SQL HAVING语法

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name  
HAVING aggregate\_function(column\_name) operator value;

SQL HAVING实例

查找总访问量大于200的网站

使用SQL语句：

实例：

SELECT Websites.name, Websites.url, SUM(access\_log.count) AS nums FROM (access\_log  
INNER JOIN Websites  
ON access\_log.site\_id=Websites.id)  
GROUP BY Websites.name  
HAVING SUM(access\_log.count) > 200;

查找总访问量大于200的网站，并且alexa排名小于200

在SQL语句中增加一个普通的WHERE子句：

实例：

SELECT Websites.name, SUM(access\_log.count) AS nums FROM Websites  
INNER JOIN access\_log  
ON Websites.id=access\_log.site\_id  
WHERE Websites.alexa < 200   
GROUP BY Websites.name  
HAVING SUM(access\_log.count) > 200;

SQL UCASE()函数

UCASE()函数把字段的值转换为大写

SQL UCASE()语法

SELECT UPPER(column\_name) FROM table\_name;

SQL UCASE()实例

下面的语句从“Websites”表中选取“name”和“url”列，并把“name”列的值转换为大写：

实例：

SELECT UCASE(name) AS site\_title, url  
FROM Websites;

SQL SERVER大小写用UPPER()和LOWER()

大写：

SELECT UPPER(name) AS site\_title, url FROM Websites;

小写：

SELECT LOWER (name) AS site\_title, url FROM Websites;

SQL LCASE()函数

LCASE()函数把字段的值转换为小写

SQL LCASE()语法

SELECT LCASE(column\_name) FROM table\_name;

SQL LCASE()实例

下面的SQL语句从“Websites”表中选取“name”和“url”列，并把“name”列的值转换为小写：

实例：

SELECT LCASE(name) AS site\_title, url  
FROM Websites;

SQL MID()函数

MID()函数用于从文本字段中提取字符

SQL MID()语法

SELECT MID(column\_name,start[,length]) FROM table\_name;

|  |  |
| --- | --- |
| 参数 | 描述 |
| Column\_name | 必需。要提取字符的字段 |
| start | 必需。规定开始位置（起始值是1） |
| length | 可选。要返回的字符数。如果省略，则MID()函数返回剩余文本。 |

SQL MID()实例

下面的SQL语句从“Websites”表的“name”列中提取前4个字符：

实例：

SELECT MID(name,1,4) AS ShortTitle  
FROM Websites;

SQL LEN()函数

LEN()函数返回文本字段中值的长度

SQL LEN()语法

SELECT LEN(column\_name) FROM table\_name;

SQL LEN()实例

下面的SQL语句从“Websites”表中选取“name”和“url”列中值的长度：

实例：

SELECT name, LENGTH(url) as LengthOfURL  
FROM Websites;

SQL ROUND()函数

ROUND()函数用于把数值字段舍入为指定的小数位数

SQL ROUND()语法

SELECT ROUND(column\_name,decimals) FROM table\_name;

|  |  |
| --- | --- |
| 参数 | 描述 |
| Column\_name | 必需。要舍入的字段 |
| decimals | 必需。规定要返回的小数位数 |

注意：ROUND返回值被变换为一个BIGINT

SQL NOW()函数

NOW()函数返回当前系统的日期和时间

SQL NOW()语法

SELECT NOW() FROM table\_name;

SQL NOW()实例

下面的SQL语句从“Websites”表中选取name，url，及当前日期：

实例：

SELECT name, url, Now() AS date  
FROM Websites;

SQL FORMAT()函数

FORMAT()函数用于对字段的显示进行格式化

SQL FORMAT()语法

SELECT FORMAT(column\_name,format) FROM table\_name;

|  |  |
| --- | --- |
| 参数 | 描述 |
| Column\_name | 必需。要格式化的字段 |
| format | 必需。规定格式 |

SQL FORMAT()实例

下面的SQL语句从“Websites”表中选取name，url以及格式化为YYYY-MM-DD的日期：

实例：

SELECT name, url, DATE\_FORMAT(Now(),'%Y-%m-%d') AS date  
FROM Websites;

**SQL快速参考**

|  |  |
| --- | --- |
| SQL语句 | 语法 |
| AND/OR | SELECT column\_name(s)  FROM table\_name  WHERE condition  AND/OR condition |
| ALTER TABLE | ALTER TABLE table\_name  ADD column\_name datatype  Or  ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS(alias) | SELECT column\_name AS column\_alias  FROM table\_name  Or  SELECT column\_name  FROM table\_name AS table\_alias |
| BETWEEN | SELECT column\_name(s)  FROM table\_name  WHERE column\_name  BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name  (  Column\_name1 data\_type,  Column\_name2 data\_type,  Column\_name3 data\_type,  ...  ) |
| CREATE INDEX | CREATE INDEX index\_name  ON table\_name(column\_name)  Or  CREATE UNIQUE INDEX index\_name  ON table\_name(column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS  SELECT column\_name(s)  FROM table\_name  WHERE condition |
| DELETE | DELETE FROM table\_name  WHERE some\_column=some\_value  Or  DELETE FROM table\_name  (NOTE:Deletes the entire table!!)  DELETE \* FROM table\_name  (NOTE:Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name,aggregate\_function(column\_name)  FROM table\_name  WHERE column\_name operator value  GROUP BY column\_name |
| HAVING | SELECT column\_name,aggregate\_function(column\_name)  FROM table\_name  WHERE column\_name operator value  GROUP BY column\_name  HAVING aggregate\_function(column\_name) operator value |
| IN | SELECT column\_name(s)  FROM table\_name  WHERE column\_name  IN(value1,value2,...) |
| INSERT INTO | INSERT INTO table\_name  VALUES(value1,value2,value3,...)  Or  INSERT INTO table\_name  (column1,column2,column3,...) |
| INNER JOIN | SELECT column\_name(s)  FROM table\_name1  INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LEFT JOIN | SELECT column\_name(s)  FROM table\_name1  LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN | SELECT column\_name(s)  FROM table\_name1  RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN | SELECT column\_name(s)  FROM table\_name1  FULL JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE | SELECT column\_name(s)  FROM table\_name  WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s)  FROM table\_name  ORDER BY column\_name [ASC/DESC] |
| SELECT | SELECT column\_name(s)  FROM table\_name |
| SELECT \* | SELECT \*  FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s)  FROM table\_name |
| SELECT INTO | SELECT \*  INTO new\_table\_name[IN externaldatabase]  FROM old\_table\_name  Or  SELECT column\_name(s)  INTO new\_table\_name [IN externaldatabase]  FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s)  FROM table\_name |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | SELECT column\_name(s) FROM table\_name1  UNION  SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | SELECT column\_name(s) FROM table\_name1  UNION ALL  SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name  SET column1=value,column2=value,...  WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s)  FROM table\_name  WHERE column\_name operator value |