SQL高级教程

LIMIT子句

用于规定要返回的记录的条数； 大型表格查询值得拥有； 并非每一个数据库都支持TOP子句；

MySQL语法：

```
SELECT 列名 FROM 表名 LIMIT number;
```

Persons表

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing
4	Obama	Barack	Pennsylvania Avenue	Washington

例子：

```
选取前两条数据：
SELECT * FROM Persons LIMIT 2;
```

Like子句

用于在WHERE子句中搜索列中的指定模式。

LIKE操作符LIKE操作符用于WHERE子句中搜索列的指定模式

```
SELECT 列名 FROM 表名 WHERE 列名 LIKE 模式;
```

例子 选取表中居住在以N开始的城市里的人：

```
SELECT * FROM persons WHERE City LIKE 'N%';
```

从 "Persons" 表中选取居住在以 "g" 结尾的城市里的人：

```
SELECT * FROM persons WHERE City LIKE '%g';
```

从 "Persons" 表中选取居住在包含 "lon" 的城市里的人：

```
SELECT * FROM persons WHERE City LIKE '%lon%';
```

从 "Persons" 表中选取居住在不包含 "lon" 的城市里的人：

```
SELECT * FROM persons WHERE City NOT LIKE '%lon%';
```

SQL通配符

搜索数据库时，可以使用SQL通配符；

通配符可以代替一个或多个字符，一般与LIKE连用，类似于正则表达式；

通配符	描述
%	代替一个或多个字符
_	仅代表一个字符
escape	转义字符

例子 "Persons" 表中选取居住在以 "Ne" 开始的城市里的人：

```
SELECT * FROM persons WHERE City LIKE 'Ne%';
```

"Persons" 表中选取名字的第一个字符之后是 "eorge" 的人：

```
SELECT * FROM persons WHERE City LIKE '_eorge';
```

IN操作符

IN允许我们在WHERE子句中规定的多个值；

```
SELECT 列名 FROM 表名 WHERE 列名 IN (value1,value2);
```

表中选取姓氏为 Adams 和 Carter 的人：

```
SELECT * FROM persons WHERE LastName IN ('Adams','Carter');
```

BETWEEN操作符

BETWEEN操作符在WHERE子句中使用，选取介于两个值之间的数据范围；

```
SELECT 列名 FROM 表名 WHERE 列名 BETWEEN value1 AND value2;
```

字母顺序显示介于 "Adams"（包括）和 "Carter"（不包括）之间的人，请使用下面的 SQL：

```
SELECT * FROM persons WHERE LastName BETWEEN 'Adams' AND 'Carter';
```

上面的例子显示范围之外的人;

```
SELECT * FROM persons WHERE LastName NOT BETWEEN 'Adams' AND 'Carter';
```

Alias(别名)

指定列名和表名的指定别名；

表的Alias的语法：

```
SELECT 列名 FROM 表名 AS 别名;
```

列的SQL Alias语法：

```
SELECT 列名 AS 别名 FROM 表名;
```

两个表分别是："Persons" 和 "Product_Orders"。我们分别为它们指定别名 "p" 和 "po"。我们希望列出 "John Adams" 的所有定单。

```
SELECT po.OrderID, p.LastName, p.FirstName FROM persons AS p, Product_Orders As po
WHERE p.LastName = 'Adams' AND p.FirstName = 'John';
```

使用一个列别名：

```
SELECT LastName AS, Family FirstName AS Name FROM persons;
```

JOIN语句

为了获得完整结果，我们需要从两个或者多个表格中获取结构，我们需要执行join；

主键Primary Key是一个列，值唯一，在表中主键值都是唯一的，这样的目的就是不重复每个表中的所有数据的情况下，把表间数据交叉捆绑在一起；

谁订购了产品，并且他们订购了什么产品?

```
select p.LastName,p.FirstName,o.OrderNo from persons As p,orders AS o WHERE p.Id_p
= o.Id_p;
```

使用JOIN语句：

```
SELECT persons.LastName, persons.FirstName,orders.OrderNo FROM persons INNER JOIN
orders ON persons.Id_P = order.Id_p ORDER BY persons.LastName;
```

JOIN： 如果表中至少有一个匹配，则返回行； INNER JOIN 内连接； LEFT JOIN： 即使右表中没有匹配，也从右表返回所有的行； RIGHT JOIN： 即使左表中没有匹配，也从右表中返回所有的行； FULL JOIN： 只要其中一个表中存在匹配，就返回行；

INNER JOIN关键字

表中至少一个匹配时，INNER JOIN关键字返回行；

```
SELECT 列名 FROM 表名1 INNER JOIN 表名2 ON 表名1.列名 = 表名2.列名;
```

例子 希望列出所有人的订购：

```
SELECT persons.LastName, persons.FirstName, orders.OrderNo FROM persons INNER JOIN
orders ON persons.Id_p = orders.Id_p ORDER BY persons.LastName;
```

LEFT JOIN关键字

LEFT JOIN 关键字从左表哪里返回所有的行，即使右表中没有匹配的行。

LEFT JOIN关键字

```
SELECT 列名 FROM 表名1 LEFT JOIN 表名2 ON 表名1.列名 = 表名2.列名;
```

列出所有的人，以及他们的订购，如果有的话；

```
SELECT persons.LastName,persons.FirstName,orders.OrderNo FROM persons LEFT JOIN
orders ON persons.Id_P = orders.Id_P ORDER BY persons.LastName;
```

左联就是前表匹配就输出，后表匹配是次要的；

RIGHT JOIN关键字

只要右表中有匹配的就返回所有的行，即便是左表中没有匹配的行；

```
SELECT 列名 FROM 表名1 RIGHT JOIN 表名2 ON 表名1.列名 = 表名2.列名;
```

希望列出所有的订单，以及订购他们的人，如果有的话；

```
SELECT persons.LastName, persons.FirstName,orders.OrderNo FROM persons RIGHT JOIN
orders ON persons,Id_P = orders.Id_P ORDER BY persons.LastName;
```

FULL JOIN关键字（Orecle支持，MYSQL不支持，可以使用UNION将左联右联链接起来）

只要其中某个表中存在匹配，FULL JOIN关键字就会返回行；

FULL JOIN关键字语法

```
SELECT 列名 FROM 表名1 FULL JOIN 表名2 ON 表名1.列名 = 表名2.列名;
```

希望列出所有的人，以及他们的定单，以及所有的定单，以及定购它们的人。

```
SELECT persons.LastName, persons.FirstName,orders.OrderNo FROM persons FULL JOIN
orders ON persons.Id_p = orders.Id_p;
```

UNION关键字

UNION用于合并两个或者多个SELECT的结果集：

UNION内部的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SELECT语句中的列顺序必须相同；

```
SELECT 列名 FROM 表名1
UNION
SELECT 列名 FROM 表名2
```

UNION操作选取不同的值。如果允许重复的值，请使用UNION ALL；

```
SELECT 列名 FROM 表名1
UNION ALL
SELECT 列名 FROM 表名2
```

Employees_China:

E_ID	E_Name
01	Zhang, Hua
02	Wang, Wei
03	Carter, Thomas
04	Yang, Ming

Employees_USA:

E_ID	E_Name
01	Adams, John
02	Bush, George
03	Carter, Thomas
04	Gates, Bill

列出所有在中国和美国的不同的雇员名：

```
SELECT E_Name FROM employees_china
UNION
SELECT E_Name FROM employees_usa;
```

列出在中国和美国的所有的雇员：

```
SELECT E_Name FROM employees_china
UNION ALL
SELECT E_Name FROM employees_usa;
```

SELECT INTO语句

SELECT INTO 从一个表中选取数据，然后把数据插入到另一个表中； 常用于表的复制和备份或者对记录进行存档；

把查找到的列插入到新表中：

```
SELECT 列名 INTO 新表名 FROM 旧表名;
```

persons表备份：

```
SELECT * INTO persons_backup FROM persons;
```

如果只拷贝某些域，使用：

```
SELECT LastName,FirstName INTO persons_backup FROM persons;
```

"Persons" 表中提取居住在 "Beijing" 的人的信息，创建了一个带有两个列的名为 "Persons_backup" 的表：

```
SELECT LastName,FirstName INTO persons_backup FROM persons WHERE City = 'Beijing';
```

DDL数据定义语言

CREATE DATABASE语法

```
CREATE DATABASE 数据库名;
```

例：

```
CREATE DATABASE my_db;
```

CREATE TABLE

```
CRRATE TABLE 表名(
    列名称1 数据类型,
    列名称2 数据类型,
    列名称3 数据类型,
    ……
)
```

数据类型	描述
integer(size)，int(size)，smallint(size)，tinyint(size)	仅容纳整数。在括号内规定数字的最大位数。
decimal(size,d)，numeric(size,d)	容纳带有小数的数字。"size" 规定数字的最大位数。"d" 规定小数点右侧的最大位数。
char(size)	
varchar(size)	
date(yyyymmdd)	容纳日期。

persons表创建实例：

```
CREATE TABLE persons(
    Id_P int,
    LastName varchar(255),
    FirstName varchar(255),
    City varchar(255)
)
```

SQL约束

约束用于限制加入表的数据的类型 可以创建表时规定约束，通过CREATE TABLE语句，也可以使用ALTER TABLE语句；

几种约束

NOT NULL UNIQUE PRIMARY KEY FORIEIGN KEY CHEACK DEFAULT

NOT NULL

非空，不接受NULL值

```
CREATE TABLE persons(
    Id_P int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
)
```

UNIQUE

值唯一，约束唯一标识数据库表中的每条记录。 UNIQUE和PRIMARY KEY约束为列和列集合提供唯一性的保证； PRIMARY KEY自动定义UNIQUE约束；

```
CREATE TABLE persons(
    Id_P int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    UNIQUE(Id_P)
)
```

为多个列定义UNIQUE约束，使用下面的SQL语法：

```
CREATE TABLE persons(
    Id_P int NOT NULL,
    LastName vachar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CONSTRAINT uc_personID UNIQUE (Id_P,LastName)
)
```

ALTER 创建UNIQUE约束：

```
ALTER TABLE persons ADD UNIQUE(Id_P);
```

多个列联合约束：

```
ALTER TABLE persons ADD CONSTRAINT UNIQUE(Id_P,LastName);
```

撤销UNIQUE约束

```
ALTER TABLE Persons DROP INDEX uc_PersonID;

ALTER TABLE Persons DROP CONSTRAINT uc_PersonId;
```

PRIMARY KEY

PRIMARY KEY约束唯一标识数据库表中的每一条记录。 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。

SQL PRIMARY KEY Constraint ON CREATE TABLE 在表创建时在Id_P列创建PRIMARY KEY约束：

```
CREATE TABLE persons(
    Id_P int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    PRIMARY KEY (Id_P)
)
```

多个列联合做主键：

```
CREATE TABLE persons(
    Id_P int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CONSTRAINT pk_personsID PRIMARY KEY (Id_P,LastName)
)
```

使用ALTER单独设置PRIMARY KEY约束：用法如下

```
ALTER TABLE persons ADD PRIMARY KEY (Id_P);
ALTER TABLE persons ADD CONSTRAINT pk_persons PRIMARY KEY (Id_P,LastName);
```

撤销PRIMARY KEY的约束：

```
ALTER TABLE persons DROP PRIMARY KEY;
ALTER TABLE persons DROP CONSTRAINT pk_PersonID;
```

FOREIGN KEY

一个表中的外键指向另一个表中的PRIMARY KEY；

在persons表和orders表中，orders表中的Id_P字段是外键，指向了persons的主键；

FOREIGN KEY约束用于预防破坏表之间的链接的动作； 也能防止非法数据插入外键列，因为他必须是他指向的那个表中的值之一；

创建Orders表时指定Id_P为外键：

```
CREATE TABLE Orders (
    Id_O int NOT NULL,
```

```
    OrderNO int NOT NULL,
    Id_P int,
    PRIMARY KEY(Id_O),
    FROEIGN KEY (Id_P) REFERENCES persons(Id_P)
)
```

需要命名FOREIGN KEY 约束，以及为多个列定义FOREIGN KEY 约束，使用下面的SQL语法：

```
CREATE TABLE Orders(
    Id_O int NOT NULL,
    OrderNo int NOT NULL,
    Id_P int,
    PRIMARY KEY (Id_O),
    CONSTRAINT fk_PerOrders FOREIGN KEY (Id_P) REFERENCES Persons(Id_P)
)
```

使用ALTER命令：

```
ALTER TABLE Orders ADD FOREIGN KEY (Id_P) REFERENCES (Id_P);
ALTER TABLE Orders ADD CONSTRAINT fk_Perorders FOREIGN KEY (Id_P) REFERENCES
Persons(Id_P);
```

撤销FOREIGN KEY

```
ALTER TABLE Orders DROP FOREIGN fk_Persons;
```

CHECK约束

用于限制列中值的范围 单个列定义CHECK约束，那么该列只允许特定的值； 一个表定义CHECK约束，那么此约束会在特定的列中对值进行限制；

在 "Persons" 表创建时为 "Id_P" 列创建 CHECK 约束。CHECK 约束规定 "Id_P" 列必须只包含大于 0 的整数。

```
CREATE TABLE Persons(
    Id_P int NOT NULL,
    LastName vachar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CHECK (Id_P > 0)
)
```

重新命名CHECK约束，以及多个列定义CHECK约束，使用：

```
CREATE TABLE Persons(
    Id_P int NOT NULL,
    LastName vachar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CONSTRAINT chk_Person CHECK (Id_P > 0 AND City = 'Sandnes')
)
```

ALTER创建：

```
ALTER TABLE persons ADD CHECK (Id_P > 0);
ALTER TABLE persons ADD CONSTRAINT fk_persons CHECK (Id_P > 0 AND City =
'Sandnes');
```

撤销CHECK约束：

```
ALTER TABLE Persons DROP CHECK chk_Person;
```

DEFAULT

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。

Persons的City创建DEFAULT约束：

```
CREATE TABLE Persons(
    Id_P int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255) DEFAULT 'Sandnes'
    或者使用GETDATE()函数将系统值插入默认值;
    OrderDate date DEFAULT GETDATE()
)
```

使用ALTER创建:

```
ALTER TABLER Persons ALTER City SET DEFAULT 'SANDNES'
```

撤销DEFAULT约束：

```
ALTER TABLE Persons ALTER City DROP DEFAULT;
```

CREATE INDEX

表中创建索引，不读取整个表的情况下，索引使整个数据库应用程序可以更快的查找数据；

您可以在表中创建索引，以便于快速的高效查询数据。 用户无法看到索引，他们只能被用来加速搜索查询；

更新一个包含索引的表需要比更新一个没有索引的表更多的时间，由于索引本身也需要被更新。因此理想做法是在差用的列中创建索引；

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);
coluumn_name规定需要索引的列。
```

SQL CREATE UNIQUE INDEX语法：

```
CREATE UNIQUE INDEX index_name ON table_name (column_name);
```

CREATE INDEX实例： 建一个简单的索引，名为 "PersonIndex"，在 Person 表的 LastName 列：

```
CREATE INDEX PersonIndex ON Person (LastName);
```

以降序索引某个列中的值，您可以在列名称之后添加保留字 DESC：

```
CREATE INDEX PersonIndex ON Person (LastName DESC);
```

希望索引不止一个列，您可以在括号中列出这些列的名称，用逗号隔开：

```
CREATE INDEX PersonIndex ON Person (LastName,FirstName);
```

DROP

删除索引、表、以及数据库

```
删除索引： ALTER TABLE table_name DROP INDEX index_name;
删除表： DROP TABLE 表名称;
```

```
删除数据库： DROP DATABASE 数据库名;
清空数据库： TRUNCATE TABLE 表名称;
```

ALTER

ALTER TABLE 语句用于在已有的表中添加、修改和删除列；

```
表中添加列：ALTER TABLE table_name ADD column_nam datatype;
删除表中列：ALTER TABLE table_name DROP COLUMN column_name;
更改表中数据类型：ALTER TABLE table_name ALTER COLUMN column_name datatype;
```

AUTO INCREMENT

自增

CREATE VIEW

视图是基于SQL语句的结果集的可视化的表；视图包括行和列，就像是一个真实的表。视图中的字段就是来自一个或者多个数据库中真实的表的字段。我们可以向视图中添加SQL函数、WHERE以及JOIN语句，我们可以提交数据，就像这些来自某个单一的表；

```
CREATE VIEW view_name AS SELECT 列名 FROM 表名 WHERE condition;
```

可以从某个查询内部、某个存储过程内部，或者另一个视图内部来使用视图。通过向视图添加函数，join等等，我们可以向用户精确的提交我们希望提交的数据；

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

我们可以查询这上面的视图：

```
SELECT * FROM [Current Product List];
```

更新视图：

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name FROM table_name
WHERE condition;

CREATE VIEW [Current Product List] AS
```

```
SELECT ProductID,ProductName,Category
FROM Products WHERE Discontinued = No;
```

Date函数

当我们处理日期时，最难的任务恐怕是确保所插入的日期的格式，与数据库中日期列相匹配；只要数据包含的是日期部分，运行查询就不会出现问题。但是如果涉及时间，情况就有点复杂了。

MySQL Date函数

函数	描述
NOW()	返回当前的日期时间
CURDATE()	返回当前的日期
CURTIME()	返回当前的日期
DATE()	提取日期或日期/时间表达式的日期部分
EXTRACT()	返回日期/时间按的单独部分
DATE_ADD()	给时间添加指定的时间间隔
DATE_SUB()	从日期减去指定的时间间隔
DATEDIFF()	返回两个日期之间的天数
DATE_FORMAT()	用不同的格式显示日期/时间

SQL DATE数据类型

数据类型	格式
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYY-MM-DD HH:MM:SS
YEAR	YYYY

OrderId	ProductName	OrderDate
1	computer	2008-12-26
2	printer	2008-12-26
3	electrograph	2008-11-12
4	telephone	2008-10-19

从上表中选取 OrderDate 为 "2008-12-26" 的记录

```
SELECT * FROM Orders WHERE OrderDate='2018-12-26';
```

OrderId	ProductName	OrderDate
1	computer	2008-12-26 16:23:55
2	printer	2008-12-26 10:45:26
3	electrograph	2008-11-12 14:12:08
4	telephone	2008-10-19 12:56:10

如果继续使用上述的查询语句则不会有结果，由于该查询不含有时间部分的日期 如果希望简单维护，就使用日期中不使用时间部分；