Select

用途：

从指定表中取出指定的列的数据

语法：

```
SELECT column_name(s) FROM table_name
```

解释：

从数据库中选取资料列，并允许从一或多个资料表中，选取一或多个资料列或资料行。SELECT 陈述式的完整语法相当复杂，但主要子句可摘要为：

SELECT select_list
[INTO new_table]
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]]

例：

“Persons” 表中的数据有

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

选出字段名” LastName”、” FirstName” 的数据

```
SELECT LastName,FirstName FROM Persons
```

返回结果:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

选出所有字段的数据

```
SELECT * FROM Persons
```

返回结果:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Where

用途：

被用来规定一种选择查询的标准

语法：

```
SELECT column FROM table WHERE column condition value
```

下面的操作符能被使用在WHERE中：

=,< >,>,<,> =,< =,BETWEEN,LIKE

注意： 在某些SQL的版本中不等号< >能被写作为!=

解释：

SELECT语句返回WHERE子句中条件为true的数据

例：

从” Persons”表中选出生活在” Sandnes” 的人

```
SELECT * FROM Persons WHERE City='Sandnes'
```

"Persons" 表中的数据有:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951

Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

返回结果:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

And & Or

用途：

在WHERE子句中AND和OR被用来连接两个或者更多的条件

解释：

AND在结合两个布尔表达式时，只有在两个表达式都为 TRUE 时才传回 TRUE

OR在结合两个布尔表达式时，只要其中一个条件为 TRUE 时，OR便传回 TRUE

例：

"Persons" 表中的原始数据:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

用AND运算子来查找"Persons" 表中FirstName为"Tove"而且LastName为" Svendson"的数据

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

返回结果:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

用OR运算子来查找"Persons" 表中FirstName为"Tove"或者LastName为" Svendson"的数据

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

返回结果:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

你也能结合AND和OR (使用括号形成复杂的表达式),如:

```
SELECT * FROM Persons WHERE
(FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

返回结果:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Between…And

用途：

指定需返回数据的范围

语法：

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

例：

“Persons”表中的原始数据

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

用BETWEEN…AND返回LastName为从”Hansen”到”Pettersen”的数据：

```
SELECT * FROM Persons WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

返回结果：

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

为了显示指定范围之外的数据，也可以用NOT操作符：

```
SELECT * FROM Persons WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

返回结果：

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Distinct

用途：

DISTINCT关键字被用作返回唯一的值

语法：

```
SELECT DISTINCT column-name(s) FROM table-name
```

解释：

当column-name(s)中存在重复的值时，返回结果仅留下一个

例：

“Orders”表中的原始数据

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

用DISTINCT关键字返回Company字段中唯一的值：

```
SELECT DISTINCT Company FROM Orders
```

返回结果：

Company
Sega
W3Schools
Trio

Order by

用途：

指定结果集的排序

语法：

```
SELECT column-name(s) FROM table-name ORDER BY { order_by_expression [ ASC | DESC ] }
```

解释：

指定结果集的排序，可以按照ASC(递增方式排序，从最低值到最高值)或者DESC(递减方式排序，从最高值到最低值)的方式进行排序，默认的方式是ASC

例：

“Orders”表中的原始数据：

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

按照Company字段的升序方式返回结果集：

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

返回结果：

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

按照Company字段的降序方式返回结果集：

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

返回结果：

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

Group by

用途：

对结果集进行分组，常与汇总函数一起使用。

语法：

```
SELECT column,SUM(column) FROM table GROUP BY column
```

例：

“Sales”表中的原始数据：

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

按照Company字段进行分组，求出每个Company的Amout的合计：

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

返回结果：

Company	SUM(Amount)
W3Schools	12600
IBM	4500

Having

用途：

指定群组或汇总的搜寻条件。

语法：

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

解释：

HAVING 通常与 GROUP BY 子句同时使用。不使用 GROUP BY 时，HAVING 则与 WHERE 子句功能相似。

例：

“Sales”表中的原始数据：

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

按照Company字段进行分组，求出每个Company的Amout的合计在10000以上的数据：

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company HAVING SUM(Amount)>10000
```

返回结果：

Company	SUM(Amount)
W3Schools	12600

Join

用途：

当你要从两个或者以上的表中选取结果集时，你就会用到JOIN。

例：

“Employees”表中的数据如下，（其中ID为主键）：

ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

“Orders”表中的数据如下：

ID	Product
01	Printer
03	Table
03	Chair

用Employees的ID和Orders的ID相关联选取数据：

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.ID = Orders.ID
```

返回结果：

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

或者你也可以用JOIN关键字来完成上面的操作：

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.ID = Orders.ID
```

INNER JOIN的语法：

```
SELECT field1, field2, field3
FROM first_table
INNER JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

解释：

INNER JOIN返回的结果集是两个表中所有相匹配的数据。

LEFT JOIN的语法：

```
SELECT field1, field2, field3
FROM first_table
LEFT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

用”Employees”表去左外联结”Orders”表去找出相关数据：

```
SELECT Employees.Name, Orders.Product
FROM Employees
LEFT JOIN Orders
ON Employees.ID = Orders.ID
```

返回结果：

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

解释：

LEFT JOIN返回”first_table”中所有的行尽管在” second_table”中没有相匹配的数据。

RIGHT JOIN的语法：

```
SELECT field1, field2, field3
FROM first_table
RIGHT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

用”Employees”表去右外联结”Orders”表去找出相关数据：

```
SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.ID = Orders.ID
```

返回结果：

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

解释：

RIGHT JOIN返回” second_table”中所有的行尽管在”first_table”中没有相匹配的数据。

Alias

用途：

可用在表、结果集或者列上，为它们取一个逻辑名称

语法：

给列取别名：

```
SELECT column AS column_alias FROM table
```

给表取别名：

```
SELECT column FROM table AS table_alias
```

例：

“Persons”表中的原始数据：

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes

Pettersen	Kari	Storgt 20	Stavanger

运行下面的SQL：

```
SELECT LastName AS Family, FirstName AS Name
FROM Persons
```

返回结果：

Family	Name
Hansen	Ola
Svendson	Tove
Pettersen	Kari

运行下面的SQL：

```
SELECT LastName, FirstName
FROM Persons AS Employees
```

返回结果：

Employees中的数据有：

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Insert Into

用途：

在表中插入新行

语法：

插入一行数据

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

插入一行数据在指定的字段上

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

例：

“Persons”表中的原始数据：

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

运行下面的SQL插入一行数据：

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

插入后”Persons”表中的数据为：

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

运行下面的SQL插入一行数据在指定的字段上：

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

插入后”Persons”表中的数据为：

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

Update

用途：

更新表中原有数据

语法：

```
UPDATE table_name SET column_name = new_value
```

```
WHERE column_name = some_value
```

例：

“Person”表中的原始数据：

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

运行下面的SQL将Person表中LastName字段为”Rasmussen”的FirstName更新为”Nina”：

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

更新后”Person”表中的数据为：

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

同样的，用UPDATE语句也可以同时更新多个字段：

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

更新后”Person”表中的数据为：

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Delete

用途：

删除表中的数据

语法：

```
DELETE FROM table_name WHERE column_name = some_value
```

例：

“Person”表中的原始数据：

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

删除Person表中LastName为”Rasmussen”的数据：

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

执行删除语句后”Person”表中的数据为：

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Create Table

用途：

建立新的资料表。

语法：

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
.......
)
```

例：

创建一张叫“Person”的表，该表有4个字段"LastName", "FirstName", "Address", "Age"：

```
CREATE TABLE Person
(
LastName varchar,
FirstName varchar,
Address varchar,
```

```
Age int
)
```

如果想指定字段的最大存储长度，你可以这样：

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar(30),
Address varchar(120),
Age int(3)
)
```

下表中列出了在SQL的一些数据类型：

Data Type	Description
integer(size) int(size) smallint(size) tinyint(size)	Hold integers only. The maximum number of digits are specified in parenthesis.
decimal(size,d) numeric(size,d)	Hold numbers with fractions. The maximum number of digits are specified in "size". The maximum number of digits to the right of the decimal is specified in "d".
char(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis.
varchar(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis.
date(yyyymmdd)	Holds a date

Alter Table

用途：

在已经存在的表中增加后者移除字段

语法：

```
ALTER TABLE table_name
ADD column_name datatype
ALTER TABLE table_name
DROP COLUMN column_name
```

注意：某些数据库管理系统不允许移除表中的字段

例：

“Person”表中的原始数据：

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

在Person表中增加一个名为City的字段：

```
ALTER TABLE Person ADD City varchar(30)
```

增加后表中数据如下：

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

移除Person表中原有的Address字段：

```
ALTER TABLE Person DROP COLUMN Address
```

移除后表中数据如下：

LastName	FirstName	City
Pettersen	Kari	

Drop Table

用途：

在数据库中移除一个数据表定义及该数据表中的所有资料、索引、触发程序、条件约束及权限指定。

语法：

```
DROP TABLE table_name
```

Create Database

用途：

建立新的数据库.

语法：

```
CREATE DATABASE database_name
```

Drop Database

用途：

移除原有的数据库

语法：

```
DROP DATABASE database_name
```

聚集函数

count

用途：

传回选取的结果集中行的数目。

语法：

```
SELECT COUNT(column_name) FROM table_name
```

例：

“Persons”表中原始数据如下：

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

选取记录总数：

```
SELECT COUNT(Name) FROM Persons
```

执行结果：

```
3
```

sum

用途：

以表达式传回所有值的总和，或仅 DISTINCT 值。SUM 仅可用于数值资料行。已忽略 Null 值。

语法：

```
SELECT SUM(column_name) FROM table_name
```

例：

“Persons”表中原始数据如下：

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

选取”Persons”表中所有人的年龄总和：

```
SELECT SUM(Age) FROM Persons
```

执行结果：

```
98
```

选取”Persons”表中年龄超过20岁的人的年龄总和：

```
SELECT SUM(Age) FROM Persons WHERE Age>20
```

执行结果：

```
79
```

avg

用途：

传回选取的结果集中值的平均值。已忽略 Null 值。

语法：

```
SELECT AVG(column_name) FROM table_name
```

例：

“Persons”表中原始数据如下：

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

选取”Persons”表中所有人的平均年龄：

```
SELECT AVG(Age) FROM Persons
```

执行结果：

```
32.67
```

选取”Persons”表中年龄超过20岁的人的平均年龄：

```
SELECT AVG(Age) FROM Persons WHERE Age>20
```

执行结果：

```
39.5
```

max

用途：

传回选取的结果集中值的最大值。已忽略 Null 值。

语法：

```
SELECT MAX(column_name) FROM table_name
```

例：

“Persons”表中原始数据如下：

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

选取”Persons”表中的最大年龄：

```
SELECT MAX(Age) FROM Persons
```

执行结果：

```
45
```

min

用途：

传回选取的结果集中值的最小值。已忽略 Null 值。

语法：

```
SELECT MIN(column_name) FROM table_name
```

例：

“Persons”表中原始数据如下：

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

选取”Persons”表中的最小年龄：

```
SELECT MIN(Age) FROM Persons
```

执行结果：

```
19
```

算术函数

abs

用途：

传回指定数值表达式 (Numeric Expression) 的绝对正值。

语法：

```
ABS(numeric_expression)
```

例：

```
ABS(-1.0) ABS(0.0) ABS(1.0)
```

执行结果：

```
1.0         0.0         1.0
```

ceil

用途：

传回大于等于给定数值表达式的最小整数。

语法：

```
CEIL(numeric_expression)
```

例：

```
CEIL(123.45)    CEIL(-123.45)
```

执行结果：

```
124.00              -123.00
```

floor

用途：

传回小于或等于给定数值表达式的最大整数。

语法：

```
FLOOR(numeric_expression)
```

例：

```
FLOOR(123.45)    FLOOR(-123.45)
```

执行结果：

```
123.00              -124.00
```

cos

用途：

在指定表达式中传回指定角度 (以弧度为单位) 的三角余弦值的数学函数。

语法：

```
COS(numeric_expression)
```

例：

```
COS(14.78)
```

执行结果：

```
-0.599465
```

cosh

用途：

传回以弧度为单位的角度值，其余弦为指定的 **float** 表达式，也称为反余弦。

语法：

```
COSH(numeric_expression)
```

例：

```
COSH(-1)
```

执行结果：

```
3.14159
```

sin

用途：

以近似的数值 (float) 表达式传回给定角度 (以弧度) 之三角正弦函数 (Trigonometric Sine)。

语法：

```
SIN(numeric_expression)
```

例：

```
SIN(45.175643)
```

执行结果：

```
0.929607
```

sinh

用途：

传回以弧度为单位的角度，其正弦为指定的 float 表达式 (也称为反正弦)。

语法：

```
SINH(numeric_expression)
```

例：

```
SINH(-1.00)
```

执行结果：

```
-1.5708
```

tan

用途：

传回输入表达式的正切函数。

语法：

```
TAN(numeric_expression)
```

例：

```
TAN(3.14159265358979/2)
```

执行结果：

```
1.6331778728383844E+16
```

tanh

用途：

传回以弧度为单位的角度，其正切为指定的 **float** 表达式 (也称为反正切)。

语法：

```
TANH(numeric_expression)
```

例：

```
TANH(-45.01)
```

执行结果：

```
-1.54858
```

exp

用途：

传回给定的 float 表达式的指数 (Exponential) 值。

语法：

```
EXP(numeric_expression)
```

例：

```
EXP(378.615345498)
```

执行结果：

```
2.69498e+164
```

log

用途：

传回给定的 float 表达式之自然对数。

语法：

```
LOG(numeric_expression)
```

例：

```
LOG(5.175643)
```

执行结果：

```
1.64396
```

power

用途：

传回给定表达式指定乘幂的值。

语法：

```
POWER(numeric_expression,v)
```

例：

```
POWER(2,6)
```

执行结果：

```
64
```

sign

用途：

传回给定的表达式之正 (+1)、零 (0) 或负 (-1) 号。

语法：

```
SIGN(numeric_expression)
```

例：

```
SIGN(123)    SIGN(0)    SIGN(-456)
```

执行结果：

```
1            0          -1
```

sqrt

用途：

传回给定表达式的平方。

语法：

```
SQRT(numeric_expression)
```

例：

```
SQRT(10)
```

执行结果：

```
100
```