1、只检索头5行的数据：

①SQL SERVER&ACCESS

select top 5 \* from product

②DB2:

select \* from product fetch first 5 rows only

③oracle：

select \* from product where rownum <=5

④mysql,mariaDB,postgreSQL,SQLite:

select \* from product LIMIT 5

⑤从第五行起的5行数据

select \* from product limit 5 offset 5

2、按位置多列排序

①SELECT SYSTEMID,SYSTEMNAME FROM REPORTVALUE ORDER BY SYSTEMNAME

②或者是：SYSTEMNAME在第二列：SELECT SYSTEMID,SYSTEMNAME FROM REPORTVALUE ORDER BY 2

3、通配符

①%通配符不用多讲

②下划线（\_）通配符

select \* from products where prod\_name like '\_inch tech bear'

DB2不支持此通配符，只能匹配单个字符，不能多个

③方括号（[]）通配符

select \* from product where name like '[JM]'

仅有ACCESS与SQL Server支持,否定写法为[!JM]

4、拼接字段

使用+或者||

Access和Sql server使用+号，其他的一般使用||

select name + '('+country+')' from product

select name || '(' || country || ')' from product

去右边空格（rtri()））（其他的还有去左边空格:ltrim()，以及左右空格trim()）的话，可以：

select RTRIM(name) || '(' || RTRIM (country) || ')' from product

5、函数

soundex()函数

是一个将任何文本串转换为描述其语言表示的字母数字模式的算法。不支持ACCESS和PostgreSQL

例子：Customers表中有个客户Kids Place，其联系名为Michelle Green，但实际应该是Michael Green：

select name,contact from Customers where soundex(name)=soundex('Michael Green');

这里Michelle Green与Michael Green发音相似，所以soundex值匹配

日期和时间处理函数

查询2012年所有的订单

sql server：select \* from orders where datepart(yy,order\_date)=2012

PostgreSQL: select \* from orders where date\_part('year',order\_date)=2012

oracle:select \* from orders where to\_number(to\_char(order\_date,'YYYY'))=2012

select \* from orders where order\_date between to\_date('01-01-2012') and to\_date('12-31-2012')

MySQL&&MariaDB: select \* from orders where YEAR(order\_date)=2012

数值处理函数

abs() 返回一个数的绝对值

cos() 返回一个角度的余弦

exp() 返回一个数的指数值

pi() 返回圆周率

sin() 返回一个角度的正弦

sqrt() 返回一个数的人平方根

tan() 返回一个角度的正切

6、过滤分组

两个及以上订单：

select id,count(\*) as orders from Order group by id having count(\*)>=2

7、作为计算字段使用子查询

select name,state,(select count(\*) from Orders where order\_id=customer.id) as orders from customers order by name

8、从一个表复制到另一个表

select \* into customercopy from customer

要想只复制部分列，可以明确给出列名，而不是用\*,也可以用:

create table custcopy as select \* from customers;

9、获得系统时间

Access: now() DB2:current\_date mysql:current\_date()

oracle:sysdate postgreSQL:current\_date sqlserver:getdate()

sqlite:date('now')

10、更新表

alter table vendors add vend\_phone char(20) 插入列

alter table vendors drop column vend\_phone 删除咧

11、创建视图

create view productcustomer as select name,contact from customers;

select \* from productcustomer;

12、创建存储过程

CREATE PROCEDURE MailingListCount(

ListCount OUT INTERGER

)

IS

v\_rows INTERGER;

BEGIN

SELECT COUNT(\*) INTO v\_rows

FROM Customers

WHERE NOT cust\_email IS NULL;

ListCount :=v\_rows;

END;

分析：这个存储过程有个ListCount的参数。此参数从存储过程返回一个值而不是传递一个值给存储过程。关键字out用来指示这种行为。oracle支持in（传递值给存储过程）、out（从存储过程返回值）、inout（既传递值给存储过程也从存储过程传回值）类型的参数。存储过程的代码括在Begin和end语句中，这里执行一条简单的select语句，它检索具有邮件地址的顾客。然后用检索出的行数设置ListCount（要传递的输出参数）。

var ReturnValue NUMBER

EXEC MailingListCount(:ReturnValue);

SELECT ReturnValue;

sql server 版本：

create procedure MailingListCount

AS

DECLARE @cnt INTERGER

SELECT @cnt = COUNT(\*)

FROM Customers

WHERE NOT cust\_email IS NULL;

RETURN @cnt;

此存储过程没有参数，调用程序检索SQL Server的返回代码支持的值，其中用DECLARE语句声明了一个名为@cnt的局部变量（sql server中所有局部变量名都以@起头）；然后在select语句中使用这个变量，让它包含count（）函数返回的值；最后，用return @cnt语句将计数返回给调用程序。

DECLARE @ReturnValue INT

EXECUTE @ReturnValue=MailingListCount;

SELECT @ReturnValue;

在orders表中插入一个新订单，使用sql server:

CREATE PROCEDURE NewOrder @cust\_id CHAR(10)

AS

--DECLARE VARIABLE FOR ORDER NUMBER

DECLARE @order\_num INTEGER

--GET CURRENT HIGNTEST ORDER NUMBER

SELECT @order\_num=MAX(order\_num)

FROM Orders

--DETERMINE NEXT ORDER NUMBER

SELECT @order\_num=@order\_num+1

--Insert new order

INSERT INTO Orders(order\_num,order\_date,cust\_id)

VALUES(@order\_num,GETDATE(),@cust\_id)

--Return order number

RETURN @order\_num;

分析：首先声明一个局部变量来存储订单号。接着，检索当前最大订单号（使用MAX（函数））并增加1（使用select语句）。然后用insert语句插入由新生成的订单号、当前系统日期（用getdate()函数检索）和传递的顾客ID组成的订单。最后，用RETURN @order\_num返回订单号（处理订单物品需要它）。

另一个sql server代码的不用版本：

CREATE PROCEDURE NewOrder @cust\_id CHAR(10)

AS

--Insert new order

INSERT INTO Orders(cust\_id)

VALUES(@cust\_id)

--Return order number

SELECT order\_num=@@IDENTITY;

13、控制事务处理

①事务处理块的开始和结束：

sql server:

BEGIN TRANSACTION

...

COMMIT TRANSACTION

MariaDB&&MySQL:

START TRANSACTION

...

Oracle:

SET TRANSACTION

...

PostgreSQL:

BRGIN

...

②使用ROLLBACK命令用来回退（撤销）SQL语句：

DELETE FROM Orders;

ROLLBACK;

③使用COMMIT

SQL server:

BEGIN TRANSACTION

DELETE OrderItems WHERE order\_num=12345

DELETE Oders WHERE order\_num=12345

COMMIT TRANSACTION

分析：从系统中完全删除订单12345.因为涉及更新两个数据库Orders和OrderItems，所以使用事务处理块来保证订单不被部分删除。最后的COMMIT语句尽在不出错时写出更改，如果第一条delete起作用，但第二条失败，则delete不会提交；

oracle：

SET TRANSACTION

DELETE OrderItems WHEREorder\_num=12345;

DELETE Orders WHERE order\_num=12345;

COMMIT;

④创建保留点(占位符)

MariaDB、MySQL和oracle：

SAVEPOINT delete1;

sql server:

SAVE TRANSACTION delete1;

每个保留点都要取能够标识它的唯一名字，一遍回退时，DBMS知道回退到何处。

回退到保留点：

sql server:

ROLLBACK TRANSACTION delete1;

MariaDB/mysql/oracle:

ROLLBACK TO delete1;

一个完整的sql server例子：

BEGIN TRANSACTION

INSERT INTO Customers(cust\_id,cust\_name)

VALUES('1000000010','Toys Emporium');

SAVE TRANSACTION StartOrder;

INSERT INTO Orders(order\_num,order\_date,cust\_id)

VALUES(20100,'2001/12/1','1000000010');

IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;

INSERT INTO OrderItems(order\_num,order\_item,prod\_id,qualitity,item\_price)

VALUES(20100,1,'BR01',100,5.49);

IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;

INSERT INTO OrderItems(order\_num,order\_item,prod\_id,qualitity,item\_price)

VALUES(20100,2,'BR03',100,10.99);

IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;

COMMIT TRANSATION

分析：这里的事务处理块中包含了4条insert语句。在第一条insert语句之后定义了一个保留点，因此，如果后面的任何一个insert操作失败，事务处理最近回退到这里。在sql server中，可检查一个名为@@ERROR的变量，看操作是否成功。（其他DBMS使用不同的函数或变量返回此消息。）如果@@ERROR返回一个非0的值，表示有错误发生，事务处理回退到保留点。如果整个事务处理成功，发布commit以保留数据。

14、游标

游标：是一个存储在DBMS服务器上的数据库查询，它不是一条select语句，而是被该语句检索出来的结果集。

创建游标：

DB2、MariaDB、MySql和sql server：

DECLARE CustCursor CURSOR

FOR

SELECT \* FROM Customers

WHERE cust\_email IS NULL

oracle和postgresql版本：

DECLARE CURSOR CustCursor

IS

SELECT \* FROM Customers

WHERE cust\_email IS NULL

使用游标：

打开游标：

OPEN CURSOR CustCursor

例：

①使用oracle语句从游标中检索第一行：

DECLARE TYPE CustCursor IS REF CURSOR

RETURN Customers%ROWTYPE;

DECLARE CustRecord Customers%ROWTYPE

BEGIN

OPEN CustCursor;

FETCH CustCursor INTO CustRecord;

CLOSE CustCursor;

END;

②使用oracle语法从游标中检索第一行到最后一行：

DECLARE TYPE CustCursor IS REF CURSOR

RETURN Customers%ROWTYPE;

DECLARE CustRecord Customers%ROWTYPE

BEGIN

OPEN CustCursor;

LOOP

FETCH CustCursor INTO CustRecord;

EXIT WHEN CustCursor%NOTFOUND;

...

END LOOP;

CLOSE CustCursor;

END;

③使用sql server语句：

DECLARE @cust\_id CHAR(10),

@cust\_name CHAR(50),

@cust\_address CHAR(50),

@cust\_city CHAR(50),

@cust\_state CHAR(50),

@cust\_zip CHAR(50),

@cust\_country CHAR(50),

@cust\_contact CHAR(50),

@cust\_email CHAR(50)

OPEN CustCorsor

FETCH NEXT FROM CustCursor

INTO @cust\_id,@cust\_name,@cust\_address,

@cust\_city,@cust\_state,@cust\_zip,

@cust\_country,@cust\_contact,@cust\_email

WHILE @@FETCH\_STATUS=0

BEGIN

FETCH NEXT FROM CustCursor

INTO @cust\_id,@cust\_name,@cust\_address,

@cust\_city,@cust\_state,@cust\_zip,

@cust\_country,@cust\_contact,@cust\_email

END

CLOSE CustCursor

关闭游标：

close custcursor

close语句用来关闭游标。一旦游标关闭，如果不再次打开，将不能使用。第二次使用它时不需要再声明，只需要open打开它即可。

15、①主键

create table vendors(

vend\_id char(10) not null primary key

);

或者：alter table vendors add constraint promary key (vend\_id);

②外键

外键是表中的一列，其值必须列在另一表的主键中。

create table orders(

order\_id integer not null primary key,

cust\_id char(10) not null references customers(cust\_id)

);

或者：alter table orders add contraint foreign key (cust\_id) references customers (cust\_id)

create table orderItems(

quatity integer not null check (quatity>0)

);

或者：add constraint check (gender like '[MF]')

16、索引

create index prod\_name\_int on products (prod\_name);

分析：索引必须唯一命名。这里的索引名prod\_name\_ind 在关键字create index 之后定义。on用来指定被索引的表，而索引中包含的列在表名后的圆括号中给出。

检查索引：索引的效率随表数据的增加或改变而变化。许多数据库管理员发现，过去创建的某个理想的索引经过几个月的数据处理后，可能变得不再理想。最好定期检查索引，并根据需要对索引进行调整。

17、触发器

触发器是特殊的存储过程，它在特定的数据库活动发生时自动执行。

例：将Customers表中的cust\_state列转换为大写：

sql server:

CREATE TRIGGER customer\_state

ON Customers

FOR INSERT,UPDATE

AS

UPDATE Customers

SET cust\_state = Upper(cust\_state)

WHERE Customers.cust\_id=inserted.cust\_id;

oracle和postgresql:

CREATE TRIGGER customer\_state

AFTER INSERT OR UPDATE

FOR EACH ROW

BEGIN

UPDATE Customers

SET cust\_state = Upper(cust\_state)

WHERE Customers.cust\_id=:OLD.cust\_id

END;

17、数据库安全

需要保护的操作有：

①对数据管理功能（创建表，更改或删除已存在的表等）的访问

②对特定数据库或表的访问

③访问的类型（只读、对特定列的访问等）

④仅通过视图或者存储过程对表进行访问

⑤创建多层次的安全措施，从而允许多种基于登录的访问和控制

⑥限制管理用户账号的能力

安全性使用SQL的grant和revoke语句管理。

18.相关语句：

更新表结构：

ALTER TABLE tablename(

ADD/DROP column datatype [NULL/NOT NULL] [CONSTRAINTS],

.....

);

DEOP永久地删除数据库对象（表、视图、索引等）。

DROP INDEX/PROCEDURE/TABLE/VIEW

indexname/procedurename/tablename/viewname;