华易数据库管理系统 Huayisoft DB Server

_SQL 语言手册

华易软件

V6.1.12.12.31

目录

1. Huayisoft DB Server 概述

1.1 Huayisoft DB Server 是什么

Huayisoft DB Server是一套由华易软件 (九江华易软件有限公司 2001-2011、北京华易方圆软件有限公司 2011—至今)自主开发的数据库管理系统（或称数据库组件），Huayisoft DB Server的中文名称为华易数据库管理系统。Huayisoft DB Server完全支持 sql92、部分支持 sql99标准。Huayisoft DB Server就是以支持 SQL标准的核心内容为基本要求，以简化应用开发、提高开发效率为宗旨 ,以面向中小型网络应用 (B/S结构)为服务对象的一款新型对象关系数据库管理系统。

1.2 Huayisoft DB Server 的特点

Huayisoft DB Server (华易数据库管理系统)全部采用纯 JAVA语言开发，并有机地与 JAVA结合在一起，充分利用 JAVA先进的面向对象及错误捕捉等技术，并提供了许多有益的功能扩展，为开发安全稳定、高质量高效率的应用提供了坚实的基础：

一、数据类型对象化

1. Huayisoft DB Server的所有数据类型都对应一个 JAVA类

   如： 'abc' 等价于 new String('abc')

2. 任何能调用 Huayisoft DB Server字段数据的地方都可直接调用其对应的方法

   如： 'abc'.toString ()

3. 所有 Huayisoft DB Server数据对象极其方法执行结果又可作为中间参数组成新的数据

   如：'abc'.toString ().toString ().toString ()

   new Integer ('123456'.substring(1).substring(2).length()) (结果 3)

   这种自由的组合及直接调用 java原有丰富的类 (方法)可以极大地提高了数据库的整体功能，比如 :CHAR数据类型对应 java.lang.String类 ,6.0版的 String就 87个方法 (其中 15个同构方法 ,72个一般方法)，与传统数据库依赖大量函数及过程来提高整体功能的做法相比 ,使用面向对象技术不仅简化数据库系统本身，也简化了学习和使用数据库成本，也有利于开发高质量的应用程序。

二、可支持任何复杂的对象数据类型，并极为简单化了用户自定义数据类型的定义过程，不需要象 SQL99那样繁琐的定义过程。

三、可直接调用其它的 java类 (方法)，如可以直接调用 java.lang.Math中的所有方法。

四、采用对象方法来代替非对象化的存储过程

五、采用 Java API(JDBC)应用编程接口方式编程。并提供简化了的编程接口：本数据库封装了装载驱动程序、建立联接、创立语句执行等操作，仅提供一个对象化接口，利用本对象的方法，

直接执行语句并获得结果对象。在客户机的浏览器中也可直接获得此编程接口。其结果是简化了客户端的应用模块。不支持非对象化的既不是主机语言，也不是数据库语言，即所谓嵌入式 SQL编程。

六、数据库与应用高度有机结合 (独特的 EXECUTE CLASS调用)：一般的数据库与应用程序的关系是应用程序调用数据库，而 Huayisoft DB Server的 EXECUTE调用则恰恰相反。这样做的最大好处是可简化服务器端的应用模块，甚至不需要服务器端的应用模块。在这里 ,应用模块 (Java方法) 就好像是数据库功能的扩展。对应于一种高效的应用模式就是 :客户端的应用模块将直接连接到服务器端的数据库管理系统而不是服务器端的应用模块 ,也不是所谓的应用服务器。

七、编程时，可传递的参数或返回的结果不仅仅是 Huayisoft DB Server中定义的数据类型对象，也可以是可系列化或符合 Huayisoft DB Server规范的任何数据对象。

八、当有错误发生时， Huayisoft DB Server将直接返回错误信息，其中可能包括所运行的类信息。这里没有错误代号（代码）的概念。

九、集 B/S、C/S结构之优点：客户端的应用模块既可以在浏览器中运行 (APPLET),也可以通过一个所谓的引子程序调用服务器端的代码及数据 (B/S的优点)而不在浏览器中运行 (APPLICATION);也提供了浏览器版 Huayisoft DB Server 命令解释器 ,可在浏览器中直接执行各 Huayisoft DB Server语句。

总之， Huayisoft DB Server数据库管理系统通过采用面向对象以及可调用其它的 java类 (方法)等技术措施，打造了一款功能灵活强大、使用又非常简单的对象关系数据库。

1.3 Huayisoft DB Server 语法结构中的符号说明

<> 尖括号内的内容是语法定义名称

::= 语法定义符，左边是语法定义名称，右边是其定义

[] 方括号内的内容是可选的(也可不选的),但 ARRAY后面的方括号则是数组单元的分隔符。

{} 大括号内的内容是必须选择其中之一

| 多项选择符号，选择其中之一

... 表示其之前的单元是可以重复多个的

2. Huayisoft DB Server 的数据类型

数据类型作为数据库最基本的性能，是简化应用程序开发、提高开发效率最基本的功能部件！Huayisoft DB Server 不仅提供了丰富的（抽象 结构化）数据类型，同时结合对象技术使数据类型的性能得到极大的扩展！

语法：

```
<数据类型>  ::=  <基本数据类型>
               | <数组数据类型>
               | <记录数据类型>
               | <抽象数据类型>
               | <实用数据类型>
```

2.1 基本数据类型

语法：

```
<基本数据类型>  ::=  <字符数据类型>
                   | <数值数据类型>
                   | <二进制数据类型>
                   | <位数据类型>
                   | <布尔数据类型>
                   | <日期时间数据类型>
                   | <时间间隔数据类型>
```

2.1.1 CHARACTER 字符数据类型

语法：

```
<字符数据类型>  ::=  <定长字符数据类型>
                   | <变长字符数据类型>
                   | <字符大型对象数据类型>
                   | <双字节定长字符数据类型>
                   | <双字节变长字符数据类型>
                   | <双字节字符大型对象数据类型>
```

（1）定长字符数据类型

语法：

```
<定长字符数据类型>  ::=  CHAR[(长度)]
```

```
                    |   CHARACTER[(长度)]
```

功能：定义固定长度的字符串。最大长度为 16K，缺省时为 1。

例如：CHAR(99)。

（2）变长字符数据类型

语法：

```
  <变长字符数据类型>   ::=   VARCHAR[(长度)]
                          |   CHARACTER VARYING [(长度)]
                          |   CHAR VARYING [(长度)]
```

功能：定义变长字符串，最大长度为 16K，缺省时为 1。

例如：VARCHAR(99)。

（3）字符大型对象数据类型

语法：

```
  <字符大型对象数据类型>   ::=   CLOB [(长度)]
                             |   CHAR LARGE OBJECT [(长度)]
                             |   CHARACTER LARGE OBJECT [(长度)]
```

（4）双字节定长字符数据类型

语法：

```
  <双字节定长字符数据类型>   ::=   UNICODE[(长度)]
                               |   NATIONAL CHARACTER [(长度)]
                               |   NATIONAL CHAR[(长度)]
                               |   NCHAR[(长度)]
```

（5）双字节变长字符数据类型

语法：

```
  <双字节变长字符数据类型>   ::=   NATIONAL CHARACTER VARYING[(长度)]
                               |   NATIONAL CHAR VARYING [(长度)]
                               |   NCHAR VARYING[(长度)]
```

（6）双字节字符大型对象数据类型

语法：

```
<双字节字符大型对象数据类型>  ::=  NCLOB [(长度) ]
                                 |  ULOB [(长度) ]
                                 |  NCHAR LARGE OBJECT [(长度) ]
                                 |  NATIONAL CHARACTER LARGE OBJECT [(长度) ]
```

2.1.2 二进制数据类型

语法：

```
<二进制数据类型>  ::=  <二进制定长数据类型>
                     |  <二进制大型对象数据类型>
```

（1）二进制定长数据类型

语法：

```
<二进制定长数据类型>  ::=  BINARY[(长度)]
                         |  BYTE[(长度)]
```

功能：定义二进制数据类型，长度指字节长度，默认长度为 1。

例如：BINARY(3)，定义了 3 个字节的二进制数据类型。

X'09ADFC99'

X'09 AD FC 99' 是合法的。

（2）二进制大型对象数据类型

语法：

```
<二进制大型对象数据类型>  ::=  BLOB [(长度) ]
                             |  BINARY LARGE OBJECT [(长度) ]
```

2.1.3 位数据类型

语法：

```
<位数据类型>  ::=  BIT[(长度)]
                 |  BIT VARYING [(长度)]
```

功能：定义位数据类型，长度指位长度，默认长度为 1。

例如：BIT(3) ，定义了 3 个位长度的位数据类型。

B'101' 是合法的。

2.1.4 数值数据类型

语法：

```
<数值数据类型>  ::=  <精确数值数据类型>
                   |  <近似数值数据类型>
```

一、精确数值数据类型

语法：

```
<精确数值数据类型>  ::=  <整数数据类型>
                        |  <带小数位的精确数值数据类型>

<整数数据类型>  ::=  <微整数数据类型>
                   |  <小整数数据类型>
                   |  <整数数据类型>
                   |  <大整数数据类型>

<带小数位的精确数值数据类型>  ::=  <NUMERIC 数据类>
                                  |  <DECIMAL 数据类型>
```

（1）微整数数据类型

TINYINT 类型

语法：

```
<微整数数据类型>  ::=  TINYINT
```

功能：定义 1 字节整数数据类型，取值范围为：-128 ～ +127。

（2）小整数数据类型

SMALLINT 类型

语法：

```
<小整数数据类型>  ::=  SMALLINT
```

功能：定义 2 字节整数数据类型，取值范围为：-32768 ～ +32767。

（3）整数数据类型

INT 类型

语法：

```
<整数数据类型>  ::=  INT  |  INTEGER
```

功能：定义 4 字节整数数据类型，取值范围为：-2147483648～ +2147483647。

（4）大整数数据类型

BIGINT 数据类型

语法：

```
<大整数数据类型>  ::=  BIGINT  |  LONG
```

功能：定义 8 字节整数数据类型，取值范围为：-9223372036854775808～9223372036854775807。

（5）带小数位的精确数值数据类型

NUMERIC 数据类型

语法：

```
<NUMERIC 数据类型>  ::=  NUMERIC [ (  精度  [ ,标度  ] ) ]
```

功能：定义带小数位的精确数值类型，精度为数值的总数字位数，取值范围是 1 至 19，缺省是 19。标度为小数点右边的数字位数，取值范围是 0 至精度，标度缺省是 0。

当精度为 1 至 2，标度为 0 时，则与 TINYINT 类型相同；

当精度为 3 至 4，标度为 0 时，则与 SMALLINT 类型相同；

当精度为 5 至 9，标度为 0 时，则与 INTEGER 类型相同；

当精度为 10 至 19，标度为 0 时，则与 BIGINT 类型相同；

例如：NUMERIC(9,3)定义了小数点左边 6 位和小数点右边 3 位，共 9 位的数字，范围在 -999999.999 到 999999.999。

DECIMAL 数据类型

语法：

```
<DECIMAL 数据类型>  ::=  DECIMAL[(精度[,标度])]
                       |  DEC[(精度[,标度])]
```

功能：与 NUMERIC 相似。

二、 近似数值数据类型

语法：

```
<近似数值数据类型>  ::=  <浮点数数据类型>
                       |  <单精度浮点数数据类型>
                       |  <双精度浮点数数据类型>
```

（1）单精度浮点数数据类型

REAL 类型

语法：

<浮点数数据类型> ::= REAL

功能：定义 4 字节浮点数数据类型，取值范围-3.4E + 38 ～ 3.4E + 38。

（2）双精度浮点数数据类型

语法：

<双精度浮点数数据类型> ::= DOUBLE PRECISION

功能：定义 8 字节浮点数数据类型，取值范围-1.7E + 308 ～ 1.7E + 308。

（1）单精度浮点数数据类型

语法：

<浮点数数据类型> ::= FLOAT[(精度)]

功能：当精度为 1 至 7，则与 REAL 类型相同；

否则与 DOUBLE PRECISION 类型相同；

2.1.5 布尔数据类型

语法：

<布尔数据类型> ::= { BOOLEAN | BOOL } [<真值> ,<假值> [,<未知值>]]

<真值> ::= 字符串

<假值> ::= 字符串

<未知值> ::= 字符串

功能：定义布尔数据类型，默认值是 TURE 、FALSE 或 UNKNOWN，在数据库中用 1、0 或 2 进行存储，1 表示 TRUE，0 表示 FALSE，2 或其它值为 UNKNOWN。

例如：

```
CREATE TABLE BOOL_TEST (
        姓名  CHAR(8),
        性别  BOOL ('男',  '女',  '不知道'),
        结婚否  BOOL);

INSERT INTO BOOL_TEST VALUES ('李某',  '男', TRUE);
INSERT INTO BOOL_TEST VALUES ('王某',  '不知道', UNKNOWN);

SELECT * FROM BOOL_TEST;
```

姓名	性别	结婚否
李某	男	TRUE
王某	不知道	UNKNOWN

2.1.6 日期时间数据类型

语法：

```
<日期时间数据类型>  ::=  <日期数据类型>
                       |  <时间数据类型>
                       |  <时间戳数据类型>
```

（1）DATE 日期数据类型

DATE 类型

语法：

```
<日期数据类型>  ::=  DATE
```

功能：定义日期数据类型，DATE 类型包括年、月、日信息。

年，取值范围：1~32767

月，取值范围：1~12

日，取值范围：1~31

例如：

'2010 年 02 月 25 日'

'2001.01.08'

'2001/01/08'

'2001-01-08'

（2）TIME 时间数据类型

TIME 类型

语法：

```
<时间数据类型>  ::=  TIME [ ( 小数秒精度 ) ]
```

功能：定义时间数据类型，TIME 类型包括时、分、秒信息，定义了一个在'00:00:00'和'23:59:59'之间的有效时间。例如：

'08 时 08 分 08 秒'

'08:08:08'

（3）TIMESTAMP 时间戳数据类型

TIMESTAMP 类型

语法：

```
<时间戳数据类型> ::=  TIMESTAMP [ ( 小数秒精度 ) ]
                    |  DATETIME
```

功能：定义日期时间数据类型，TIMESTAMP 类型包括年、月、日、时、分、秒信息，定义了一个在'0001-01-01 00:00:00'和'9999-12-31 23:59:59'之间的有效格里高利日期时间。

2.1.7 INTERVAL 时间间隔数据类型

语法：

```
<时间间隔数据类型> ::=  <年间隔数据类型>
                      |  <月间隔数据类型>
                      |  <年月间隔数据类型>
                      |  <日间隔数据类型>
                      |  <日小时间隔数据类型>
                      |  <日分钟间隔数据类型>
                      |  <日秒间隔数据类型>
                      |  <小时间隔数据类型>
                      |  <小时分钟间隔数据类型>
                      |  <小时秒间隔数据类型>
                      |  <分钟间隔数据类型>
                      |  <分钟秒间隔数据类型>
                      |  <秒间隔数据类型>
```

（1）INTERVAL YEAR 年间隔数据类型

语法：

```
<年间隔数据类型>  ::=  INTERVAL YEAR［ ( 引导精度 ) ］
```

功能：定义年间隔数据类型，引导精度规定年的取值范围，其默认值是 2。

例如：INTERVAL YEAR(3)，则取值范围为-999～ 999 年。

INTERVAL '99' YEAR 是合法的。

（2）INTERVAL MONTH 月间隔数据类型

语法：

```
<月间隔数据类型>  ::=  INTERVAL MONTH ［ ( 引导精度 ) ］
```

功能：定义月间隔数据类型，引导精度规定月的取值范围，其默认值是 2。

例如：INTERVAL MONTH(3)，则取值范围为-999～ 999 月。

INTERVAL '99' MONTH 是合法的.

（3）INTERVAL YEAR TO MONTH 年月间隔数据类型

语法：

<年月间隔数据类型> ::= INTERVAL YEAR［ (引导精度) ］TO MONTH

功能：定义年月间隔数据类型，引导精度规定年的取值范围，其默认值是 2，月的取值范围为 0～ 11。

例如：INTERVAL YEAR(3) TO MONTH，则取值范围为-999 年零 12 月到 999 年零 12 月。

INTERVAL '99-09' YEAR TO MONTH 是合法的。

（4）INTERVAL DAY 日间隔数据类型

语法：

<日间隔数据类型> ::= INTERVAL DAY［ (引导精度) ］

功能：定义日间隔数据类型，引导精度规定日的取值范围，其默认值是 2。

例如：INTERVAL DAY(3)，则取值范围为-999～ 999 日。

INTERVAL '99' DAY 是合法的。

（5）INTERVAL DAY TO HOUR 日小时间隔数据类型

语法：

<日小时间隔数据类型> ::= INTERVAL DAY［ (引导精度) ］ TO HOUR

功能：定义日小时时间隔数据类型，引导精度规定日的取值范围，其默认值是 2。小时的取值范围为 0～ 23。

例如：INTERVAL DAY(3) TO HOUR，则取值范围为负 999 日零 23 小时到正 999 日零 23 小时。

INTERVAL '99 09' DAY TO HOUR 是合法的。

（6）INTERVAL DAY TO MINUTE 日分钟间隔数据类型

语法：

<日分钟间隔数据类型> ::= INTERVAL DAY［ (引导精度) ］ TO MINUTE

功能：定义日分间隔数据类型，引导精度规定日的取值范围，其默认值是 2。小时的取值范围为 0～ 23，分钟的取值范围为 0～ 59。

例如：INTERVAL DAY(3) TO MINUTE，则取值范围为负 999 日零 23 小时零 59 分到正 999 日零 23 小时零 59 分。

INTERVAL '099 09:09' DAY TO MINUTE 是合法的。

（7）INTERVAL DAY TO SECOND 日秒间隔数据类型

语法：

<日秒间隔数据类型> ::= INTERVAL DAY［ (引导精度) ］ TO SECOND [(小数秒精度)]

功能：定义日秒间隔数据类型，引导精度规定日的取值范围，其默认值是 2。小时的取值范围为 0～ 23，分钟的取值范围为 0～ 59，秒整数部分的取值范围为 0～ 59。小数秒精度规定秒字段中小数点后面的位数，其取值范围为 0～6，默认值是 6。

例如：INTERVAL DAY(3) TO SECOND(1)，则取值范围为负 999 日零 23 小时零 59 分零 59.9 秒到正 999 日零 23 小时零 59 分零 59.9 秒。

INTERVAL '099 09:09:09.9' DAY TO SECOND 是合法的。

（8）INTERVAL HOUR 小时间隔数据类型

语法：

<小时间隔数据类型> ::= INTERVAL HOUR ［ (引导精度) ］

功能：定义小时间隔数据类型，引导精度规定小时的取值范围，其默认值是 2。

例如：INTERVAL HOUR(3)，则取值范围为负 999 小时到正 999 小时。

INTERVAL '099' HOUR 是合法的。

（9）INTERVAL HOUR TO MINUTE 小时分钟间隔数据类型

语法：

<小时分钟间隔数据类型> ::= INTERVAL HOUR［ (引导精度) ］ TO MINUTE

功能：定义小时分钟间隔数据类型，引导精度规定小时的取值范围，其默认值是 2，分钟的取值范围为 0～ 59。

例如：INTERVAL HOUR(3) TO MINUTE，则取值范围为负 999 小时零 59 分到正 999 小时零 59 分。

INTERVAL '99:09' HOUR TO MINUTE 是合法的。

（10）INTERVAL HOUR TO SECOND 小时秒间隔数据类型

语法：

<小时秒间隔数据类型> ::= INTERVAL HOUR［ (引导精度) ］ TO SECOND [(小数秒精度)]

功能：定义小时秒间隔数据类型，引导精度规定小时的取值范围，其默认值是 2。分钟的取值范围为 0～ 59 之间，秒整数部分的取值范围为 0～ 59。小数秒精度规定秒字段中小数点后面的位数，其取值范围为 0～6，默认值是 6。

例如：INTERVAL HOUR(3) TO SECOND(1)，则取值范围为负 999 小时零 59 分零 59.9 秒到正 999 小时零 59 分零 59.9 秒。

INTERVAL '99:09:09.9' HOUR TO SECOND 是合法的。

（11）INTERVAL MINUTE 分钟间隔数据类型

语法：

<分钟间隔数据类型> ::= INTERVAL MINUTE［ (引导精度) ］

功能：定义分钟间隔数据类型，引导精度规定分钟的取值范围，其默认值是 2。

例如：INTERVAL MINUTE(3)，则取值范围为负 999 分钟到正 999 分钟。

INTERVAL '99' MINUTE 是合法的。

（12）INTERVAL MINUTE TO SECOND 分钟秒间隔数据类型

语法：

<分钟秒间隔数据类型> ::= INTERVAL MINUTE［ (引导精度) ］
TO SECOND [(小数秒精度)]

功能：定义分钟秒间隔数据类型，引导精度规定分钟的取值范围，其默认值是 2，秒整数部分的取值范围为 0～ 59，小数秒精度规定秒字段中小数点后面的位数，其取值范围为 0～6，默认值是 6。

例如：INTERVAL MINUTE(3) TO SECOND(1)，则取值范围为负 999 分零 59.9 秒到正 999 分零 59.9 秒。

INTERVAL '99:09.9' MINUTE TO SECOND 是合法的。

（13）INTERVAL SECOND 秒间隔数据类型

语法：

<秒间隔数据类型> ::= INTERVAL SECOND ［ (引导精度 [, 小数秒精度]) ］

功能：定义秒间隔数据类型，引导精度规定秒的取值范围，其默认值是 2，小数秒精度规定秒字段中小数点后面的位数，其取值范围为 0～6，默认值是 6。

例如：INTERVAL SECOND(3,1)，则取值范围为负 999.9 秒到正 999.9 秒。

INTERVAL '99.9' SECOND 是合法的。

2.2 数组数据类型

ARRAY 数据类型

语法：

<数组数据类型> ::= <数据类型> ARRAY [长度]

注：此处的方括号为语法元素

功能：定义多个同一数据类型的数据。

例如：

INT ARRAY[3] 定义了包含 3 个整型的数组数据类型。ARRAY[123,123,123] 是合法的。

```
CREATE TABLE ARRAY_TEST (
        编号  INT,
        坐标点  INT ARRAY [2]);

INSERT INTO ARRAY_TEST VALUES (1,ARRAY[8,9]);
INSERT INTO ARRAY_TEST VALUES (2,ARRAY[10,11]);

SELECT  编号,  坐标点 FROM ARRAY_TEST;

编号    坐标点
1       [8,9]
2       [10,11]

UPDATE ARRAY_TEST SET  坐标点[1]=118 WHERE  坐标点[1]=10;

SELECT 坐标点[1] FROM ARRAY_TEST;

坐标点[1]
8
118
```

2.3 记录数据类型

ROW 记录（结构化）数据类型

语法：

<记录数据类型> ::= ROW（列名 <数据类型> [{, 列名 <数据类型> } …] ）

功能：定义多个不同数据类型的数据。

例如：ROW (username CHAR(12), password CHAR(8)) 。

ROW('huayisoft', 'www.com') 是合法的。

```
CREATE TABLE ROW_TEST (
        编号  INT,
        坐标点  ROW(x INT, y INT));
```

```
INSERT INTO ROW_TEST VALUES (1, ROW(8,9));
INSERT INTO ROW_TEST VALUES (2, ROW(10,11));

SELECT 编号, 坐标点 FROM ROW_TEST;

编号    坐标点
3       (8,9)
4       (10,11)

UPDATE ROW_TEST SET 坐标点.x=118 WHERE 坐标点.x=10;

SELECT 编号,坐标点.x, 坐标点.y FROM ROW_TEST;
编号        坐标点.x     坐标点.y
1           8            9
2           118          11
```

2.4 抽象数据类型

语法：

```
<抽象数据类型>  ::= 抽象数据类型名称
```

功能：用户可以将以上数据类型扩展为用户自定义的数据类型。通常用户可以使用自己开发的类对象来处理对应的数据。

例如：

```
CREATE TYPE PCODE AS BIGINT CLASS db.jdbc.type.Pcode
```

则：

```
CREATE TABLE TYPE_TEST (身份证号 PCODE, 姓名 CHAR(8)) 是合法的。
```

也可删除用户定义类型

```
DROP TYPE PCODE
```

2.5 实用数据类型

语法：

```
<实用数据类型> ::=  <中华人民共和国身份证数据类型>
                  |  <IP 地址数据类型>
                  |  <版本号数据类型>
```

```
                    |   <货币数据类型>
```

（1）PCODE 中华人民共和国身份证数据类型

PCODE 类型

语法：

```
  <中华人民共和国身份证数据类型>   ::=   PCODE
```

功能：定义有效中华人民共和国身份证号。满足 GB11643-1999<<公民身份号码>>中的规定。

例如：’11010519491231002X’ 是合法的。

’110105194912310021’ 是不合法的。

```
CREATE TABLE PCODE_TEST (身份证号  PCODE, 姓名  CHAR(8));
INSERT INTO    PCODE_TEST VALUES ('11010519491231002X', '张某');
INSERT INTO    PCODE_TEST VALUES ('440524188001010014', '王某');

SELECT 身份证号, 姓名,
        CAST(身份证号.getSex()    AS CHAR(2)) AS  性别,
        CAST(身份证号.getBirthday()    AS DATE) AS  出生日期
        FROM PCODE_TEST;
```

身份证号	姓名	性别	出生日期
11010519491231002X	张某	女	1949 年 12 月 31 日
440524188001010014	王某	男	1880 年 1 月 1 日

(2) IP 地址数据类型

语法：

```
  <IP 地址数据类型>   ::=   IP
```

功能：定义有效的 IP 地址。

例如：'127.0.0.1' 是合法的。

(3) VERSION 版本号数据类型

语法：

```
  <版本号数据类型>   ::=   VERSION
```

功能：定义有效的版本号。

例如：'1.6.0' 是合法的。

（4）货币数据类型

MONEY 数据类型

语法：

<货币数据类型> ::= MONEY [(精度 [, 标度])]

功能：该类型用于存储货币数据，货币数据用正、负货币值表示。货币数据存储的是精度为 19，标度为 4 的精确数值。货币数值用小数点将局部的货币单位（如角分）从总体货币单位中分隔出来。例如 2.15 表示 2 元 15 分。功能与 DEC（19，4）相同。

3. Huayisoft DB Server 的值及其表达式

Huayisoft DB Server 支持的表达式有 数值表达式、字符串表达式、记录构造表达式、数组构造表达式、布尔值表达式等。

数值及其表达式

语法：

```
<值> ::= <常量>
        | <字段值>
        | <数组单元引用>
        | <记录值字段引用>
        | <集函数>
        | <子查询>
        | <静态方法>
        | <new 类对象>
        | <CAST 表达式>
        | <CASE 表达式>
        | <COALESCE表达式>
        | <NULLIF表达式>
        | (<值>)
```

3.1 NULL值

为了更真实地反应现实世界的对象，SQL通过 NULL值来表示数据的不可知或不适用。

例如：一个学生数学考 0分与他没有参加考试可能要加以区分！

NULL值就是用来处理这类情况的，NULL值不是数字数据的 0,也不是字符串的''.

它只是个符号，表示不可知或不适用的数据．

3.2 常量

语法：

```
<常量> ::= <数字常量>
          | <字符串常量>
          | <二进制常量>
          | <位常量>
```

3.2.1 数字常量

语法：

<数字常量> ::= <精确数字常量> | <近似数字常量>

1、精确数字常量（整数和小数常量）：

99

-99.99

所有 HUAYISOFT DB SERVER数据类型都对应一个类对象，常量也不例外，

精确数字常量类对象对应规则是：

没有小数位时：

如果值在-2147483648～ +2147483647 范围内，则对应 java.lang.Integer

否则对应 java.lang.Long

有小数位时对应 db.jdbc.Numeric

因此，SELECT (99).toString().length()

是合法的，且结果是 2

2、近似数字常量（浮点数常量），即带 E符号的浮点数：

9.0E3

-99.99E3

近似数字常量类对象对应规则是：

如果值在-3.4E + 38 ～ 3.4E + 38 范围内，则对应 java.lang.Float

否则对应 java.lang.Double

3.2.2 字符串常量,即包含在一对单引号内的文本:

'Hello World!'

'abc.123'

如果字符串中包含单引号，则要求写成两个连续的单引号：

'It isn''t ...'

结果：It isn't ...

字符串常量的类对象是 java.lang.String

因此，SELECT 'It isn''t ...'.indexOf ('''')

结果：6

3.2.3 二进制常量,即以 X 打头且包含在一对单引号内的文本:

X’44 55 66’
二进制常量的类对象是 db.jdbc.Binary
因此 ,SELECT X'44 55 66’.readByte ()
结果： 68

3.2.4 位常量,即以 B 打头且包含在一对单引号内的文本:

B’1100’
位常量的类对象是 db.jdbc.Bits
因此 ,SELECT B'1100’.get (0)
结果： true

3.3 表达式

3.3.1 CAST 表达式

语法：
<CAST 表达式> :: = CAST ({ <值表达式> | NULL } AS { <数据类型> | 数据域名 })

SELECT CAST (8 AS CHAR(5)) || 'a'
结果： 8a

3.3.2 CASE 表达式

语法：
<CASE 表达式> ::= CASE WHEN <搜索条件> THEN <RESULT 子句>
[<ELSE 子句>] END
| CASE <测试值> <简单 WHEN 子句>... [<ELSE 子句>]

<简单 WHEN 子句> ::= WHEN <值表达式> THEN <RESULT 子句>

<ELSE 子句> ::= ELSE <RESULT 子句>

<RESULT 子句> ::= <值表达式> | NULL

<测试值> ::= <值表达式>

SELECT CASE WHEN 2>1 THEN 8 ELSE 9 END
结果： 8

SELECT CASE WHEN 2>3 THEN 8 ELSE 9 END
结果： 9

SELECT CASE 2 WHEN 1 THEN 8 ELSE 9 END
结果： 9

SELECT CASE 2 WHEN 2 THEN 8 ELSE 9 END
结果： 8

3.3.3 COALESCE 表达式

语法：
<COALESCE表达式> :: = COALESCE (<值表达式> {, <值表达式> }...)

SELECT COALESCE (8,9)
结果： 8

SELECT COALESCE (8,0,0,9)
结果： 8

3.3.4 NULLIF 表达式

语法：
<NULLIF表达式> :: = NULLIF (<值表达式> , <值表达式>)

SELECT NULLIF (8, 9)
结果： 8

3.3.5 静态方法

静态方法是可以直接调用的方法：

SELECT Math::abs (-99.99)
结果： 99.99

SELECT Math::min (9, 10)
结果： 9

SELECT Math::cos (0.1 as double)

结果： 0.9950041651292624

注意：类名与方法名之间是用双冒号（::）分隔的，这与 Java 调用不同，主要是与字段方法调用加以区分。

3.3.6 new 类对象

由于 Huayisoft DB Server 数据类型都对应一个类对象，因此可以使用 new 生成一个 Huayisoft DB Server 数据：

SELECT new db.jdbc.Date (‘2009 年 09 月 09 日’)

结果： 2009 年 09 月 09 日

SELECT new db.jdbc.interval.Year (2009)

结果： 2009

SELECT new db.jdbc.type.PCode (‘11010519491231002X’).getSex ()

结果：女

3.3.7 数值表达式

语法：

<数值表达式> ::= <数值项> [{ { + | - } <数值项> }...]

<数值项> ::= <数值子项> [{ { * | / | % } <数值子项> }...]

<数值子项> ::= [{ + | - }] <值>

3.3.8 字符值表达式

语法：

<字符值表达式> ::= <字符值> [{|| <字符值> }...]

SELECT '1'||'+'||'2'||'='||'3'

结果： 1+2=3

3.3.9 记录构造表达式

语法：

<记录构造表达式> ::= ROW (<值表达式> [{ , <值表达式> }...])

3.3.10 数组构造表达式

语法：

```
<数组构造表达式> ::= <数组构造> [ { || <数组构造> } ... ]

<数组构造> ::= ARRAY [ <值表达式> [ { , <值表达式> } ... ] ]
```

3.3.11 布尔值表达式

语法：

```
<布尔表达式> ::= <布尔项> [ { OR <布尔项> } ... ]

<布尔项> ::= <布尔子项> [ { AND <布尔子项> } ... ]

<布尔子项> ::= [ NOT ] { <谓词> | ( <布尔表达式> ) }
                [ IS [ NOT ] { TRUE | FALSE | UNKNOWN } ]

<谓词> ::= <比较谓词>
         | < BETWEEN 谓词>
         | <LIKE 谓词>
         | <EXISTS 谓词>
         | <IN 谓词>
         | <UNIQUE 谓词>
         | <NULL 谓词>
         | <MATCH 谓词>
         | <OVERLAPS 谓词>
         | <QUANTIFIED 比较谓词>

<比较谓词> ::= <行值构造> <比较符> <行值构造>

<比较符> ::= =
           | >
           | <
           | >=
           | <=
           | <>
```

< BETWEEN 谓词> ::= <行值构造> [NOT] BETWEEN <行值构造> AND <行值构造>

< LIKE 谓词> ::= <字符值表达式> [NOT] LIKE <字符值表达式> [ESCAPE 字符]

<EXISTS 谓词> ::= EXISTS <子查询>

<IN 谓词> ::= <行值构造> [NOT] IN { <子查询> | (<值表达式> [, <值表达式>]…) }

<UNIQUE 谓词> ::= UNIQUE <子查询>

<NULL 谓词> ::= <行值构造> IS [NOT] NULL

<MATCH 谓词> ::= <行值构造> MATCH [UNIQUE] [PARTIAL | FULL] <子查询>

<OVERLAPS 谓词> ::= <行值构造> OVERLAPS <行值构造>

<QUANTIFIED 比较谓词> ::= <行值构造> <比较符> { ALL | { SOME | ANY } <子查询>

4. Huayisoft DB Server 数据类型与类对象

所有的 HUAYISOFT DB SERVER 数据类型都对应一个 Java 类对象：

4.1 Huayisoft DB Server 数据类型与类对象对应表

数据类型	类对象
CHAR	java.lang.String
VARCHAR	java.lang.String
CLOB	db.jdbc.type.Clob
NATIONAL CHARACTER	java.lang.String
NATIONAL CHARACTER VARYING	java.lang.String
NCLOB	db.jdbc.type.NClob
BINARY	db.jdbc.Binary
BLOB	db.jdbc.type.Blob
BIT	db.jdbc.Bits
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INTEGER	java.lang.Integer
BIGINT	java.lang.Long
NUMERIC	db.jdbc.Mumeric
DECIMAL	
REAL	java.lang.Float
DOUBLE PRECISION	java.lang.Double
BOOL	java.lang.Boolean
DATE	db.jdbc.Date
TIME	db.jdbc.Time
TIMESTAMP	db.jdbc.Timestamp
INTERVAL YEAR	db.jdbc.interval.Year
INTERVAL MONTH	db.jdbc.interval.Month
INTERVAL YEAR TO MONTH	db.jdbc.interval.YearMonth
INTERVAL DAY	db.jdbc.interval.Day
INTERVAL DAY TO HOUR	db.jdbc.interval.DayHour
INTERVAL DAY TO MINUTE	db.jdbc.interval.DayMinute
INTERVAL DAY TO SECOND	db.jdbc.interval.DaySecond
INTERVAL HOUR	db.jdbc.interval.Hour

数据类型	类对象
INTERVAL HOUR TO MINUTE	db.jdbc.interval.HourMinute
INTERVAL HOUR TO SECOND	db.jdbc.interval.HourSecond
INTERVAL MINUTE	db.jdbc.interval.Minute
INTERVAL MINUTE TO SECOND	db.jdbc.interval.MinuteSecond
INTERVAL SECOND	db.jdbc.interval.Second
ARRAY	db.jdbc.type.Array
ROW	db.jdbc.Record
PCODE	db.jdbc.type.PCode
IP	db.jdbc.type.IP
VERSION	db.jdbc.type.Version
MONEY	db.jdbc.type.Money

注: 接口 java.io.Serializable 及类 java.lang.Byte、java.lang.Short、java.lang.Integer、java.lang.Long、java.lang.Float、java.lang.Double、java.lang.String、java.lang.Boolean 请参考有关 java 语言书。

4.2 接口 Interface

4.2.1 Interface db.jdbc.Comparable

```
public interface Comparable
    public int compareTo (Object another)
    public boolean equals (Object another)
```

4.2.2 Interface db.jdbc.StringAdapter

```
public interface StringAdapter
    public void set (String value) throws Exception
    public String toString ()
```

4.2.3 Interface db.jdbc.DataAdapter

```
public interface DataAdapter
    public void in (DataInputStream is) throws Exception
    public void out (DataOutputStream os) throws Exception
```

4.2.4 Interface db.jdbc.DataAdapterCA

```
public interface DataAdapterCA
```

```
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
```

4.2.5 Interface db.jdbc.ToJavaObject

```
public interface ToJavaObject
    public Object to ()
```

4.2.6 Interface db.jdbc.interval.Adapter

```
public interface Adapter
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
```

4.3 类对象 Class

4.3.1 Class db.jdbc.Numeric

```
public class Numeric
    extends Number
    implements DataAdapterCA,StringAdapter, ToJavaObject, Comparable, Serializable

    public Numeric ()
    public Numeric (long g)
    public Numeric (double d) throws Exception
    public Numeric (long g, byte scale)
    public Numeric (String s) throws Exception

    public void set (String s) throws Exception
    public void set (long n)
    public Numeric add (Numeric n) throws Exception
    public Numeric subtract (Numeric n) throws Exception
    public Numeric multiply (Numeric n) throws Exception
    public Numeric divide (Numeric n) throws Exception
    public Numeric mod (Numeric n) throws Exception

    public byte  byteValue ()
    public short shortValue ()
```

```
public int    intValue ()
public long   longValue ()
public float  floatValue ()
public double doubleValue ()

public static final Numeric getNumeric (Object obj) throws Exception
public static final Numeric valueOf (String value) throws Exception
public static final String toString (long g,int dec)

public Numeric negate () throws Exception
public void setScale (byte to_scale)
public int getScale ()
public long getLong (int scale)
public Number getNumber (int type)
public void in (DataInputStream is, DataType ca) throws Exception
public void out (DataOutputStream os, DataType ca) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.2 Class db.jdbc.Char

```
public class Char
    implements DataAdapterCA,StringAdapter, ToJavaObject,
Comparable,java.io.Serializable

    public Char ()
    public Char (String value)

    public void set (String value) throws Exception
    public Object to () {return value;}
    public void in (DataInputStream is, DataType ca) throws Exception
    public void out (DataOutputStream os, DataType ca) throws Exception
    public int compareTo (Object y)
    public boolean equals (Object y)
    public String toString ()
```

4.3.3 Class db.jdbc.type.Blob

```
public class Blob
    extends Char

    public Blob ()
    public Blob (String name)

    public String getName ()
    public long length () throws Exception
    public byte[] getBytes (long pos, int length) throws Exception
    public InputStream getBinaryStream () throws Exception
    public InputStream getBinaryStream (long pos, long length) throws Exception
    public long position (byte[] pattern , long start) throws Exception
    public long position (Blob pattern , long start) throws Exception
    public int setBytes (long pos, byte[] bytes) throws Exception
    public int setBytes (long pos, byte[] bytes,int offset, int len)
        throws Exception
    public OutputStream setBinaryStream () throws Exception
    public OutputStream setBinaryStream (long pos) throws Exception
```

4.3.4 Class db.jdbc.type.Clob

```
public class Clob
    extends Char

    public Clob ()
    public Clob (String filename)

    public String getName ()
    public long length () throws Exception
    public String getSubString (long pos, int length) throws Exception
    public Reader getCharacterStream () throws Exception
    public Reader getCharacterStream (long pos, long length) throws Exception
    public InputStream getAsciiStream (long pos, long length) throws Exception
    public long position (String pattern , long start) throws Exception
    public long position (Clob pattern , long start) throws Exception
    public int setString (long pos, String str) throws Exception
    public int setString (long pos, String str,int offset, int len) throws Exception
```

```
    public OutputStream setAsciiStream () throws Exception
    public OutputStream setAsciiStream (long pos) throws Exception
    public Writer setCharacterStream () throws Exception
    public Writer setCharacterStream (long pos) throws Exception
```

4.3.5 Class db.jdbc.type.NClob

```
public class NClob
    extends Char

    public NClob ()
    public NClob (String filename)

    public String getName ()
    public long length () throws Exception
    public String getSubString (long pos, int length) throws Exception
    public Reader getCharacterStream () throws Exception
    public Reader getCharacterStream (long pos, long length) throws Exception
    public InputStream getAsciiStream (long pos, long length) throws Exception
    public long position (String pattern , long start) throws Exception
    public long position (NClob pattern , long start) throws Exception
    public int setString (long pos, String str) throws Exception
    public int setString (long pos, String str,int offset, int len) throws Exception
    public Writer setCharacterStream () throws Exception
    public Writer setCharacterStream (long pos) throws Exception
```

4.3.6 Class db.jdbc.Binary

```
public class Binary
    extends ByteArray
    implements StringAdapter, Comparable, Serializable

    public Binary (int size)
    public Binary (byte[] b)
    public Binary (String s)

    public void set (String s)
    public int compareTo (Object another) ;
    public boolean equals (Object another)
```

```
    public String toString ()
```

4.3.7 Class db.jdbc.ByteArray

```
public class ByteArray
    implements DataAdapter, java.io.Serializable

    public ByteArray ()
    public ByteArray (byte[] b)
    public ByteArray (int size)

    public void insert (byte[] buf)
    public void insert (int p, byte[] buf)
    public void delete (int p,int width)

    public byte[] getBuffer ()
    public void setPosition (int p)
    public int getPosition ()
    public int length ()
    public void move (int from,int to,int size)

    public boolean readBoolean ()
    public byte readByte ()
    public char readBChar ()
    public char readChar ()
    public short readShort ()
    public int readMedium ()
    public int readInt ()
    public long readLong ()
    public int readInt (int width) throws Exception
    public long readLong (int width) throws Exception
    public float readFloat ()
    public double readDouble ()
    public String readString (int byteNum)
    public String readUnicode (int byteNum)
    public byte[] read (int length)

    public void writeBoolean (boolean bool)
    public void writeByte (byte bb)
```

```
public void writeBChar (char ch)
public void writeChar (char ch)
public void writeShort (short n)
public void writeMedium (int n)
public void writeInt (int n)
public void writeLong (long n)
public void writeInt (int value, int width) throws Exception
public void writeLong (long value, int width) throws Exception
public void writeFloat (float f)
public void writeDouble (double d)
public void writeString (String str, int byteNum) throws Exception
public void writeUnicode (String str,int byteNum)
public void write (byte bs[]) throws Exception
public void write (byte bs[],int offset,int length) throws Exception

public void in (DataInputStream is) throws Exception
public void out (DataOutputStream os) throws Exception
```

4.3.8 Class db.jdbc.Bits

```
public class Bits
    extends BitArray
    implements StringAdapter,Comparable,Serializable

    public Bits (int nbits)
    public Bits (byte[] b)
    public Bits (String s)

    public void set (String sbit)
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.9 Class db.jdbc.BitArray

```
public class BitArray
    implements DataAdapter, Serializable

    public BitArray ()
```

```
public BitArray (int nbits)

public int size ()
public byte[] getBytes ()
public int getByteLength ()

public void insert ()
public void insert (int pos)
public void delete (int pos)
// 与
public void and (BitArray bits)
// 或
public void or (BitArray bits)
// 异或
public void xor (BitArray bits)

public boolean get (int pos)
public void set (int pos)
public void clear (int pos)
public void clearAll ()
public void setAll ()

public boolean all ()
public boolean zero ()

public void in (DataInputStream is) throws Exception
public void out (DataOutputStream os) throws Exception
public String toString ()
```

4.3.10 Class db.jdbc.Date

```
public class Date
    implements DataAdapter,StringAdapter, Comparable,java.io.Serializable

    public Date ()
    public Date (int year,int month,int day)
    public Date (String s) throws Exception

    public int getYear ()
```

```
public int getMonth()
public int getDay ()
public boolean check (int year, int month, int day)
public static int getLastDay (int y,int m)
public void addDay (int dayNum)
public void set (String s) throws Exception
public void in (DataInputStream is) throws Exception
public void out (DataOutputStream os) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.11 Class db.jdbc.Time

```
public class Time
    implements DataAdapter,StringAdapter, Comparable,java.io.Serializable

    public Time ()
    public Time (int hour,int minute,int second)
    public Time (String hmsStr) throws Exception

    public int getHours ()
    public int getMinutes ()
    public int getSeconds ()
    public static final Time getTime (Object obj) throws Exception
    public void set (String s) throws Exception
    public void in (DataInputStream is) throws Exception
    public void out (DataOutputStream os) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.12 Class db.jdbc.Timestamp

```
public class Timestamp
    implements DataAdapter, StringAdapter,Comparable, java.io.Serializable
    public Timestamp ()
    public Timestamp (long time)
    public Timestamp (Date date, Time time) throws Exception
```

```
public Timestamp (String str) throws Exception

public long getTime ()

public int getYear ()
public int getMonth()
public int getDay ()
public int getDate ()
public int getHours ()
public int getMinutes()
public int getSeconds()
public int getNanos ()

public Date dateOf ()
public Time timeOf ()
public void set (long time)
public void set (String str) throws Exception
public void setDate (Date date)
public void setTime (Time time)
public void setYMD (int year,int month,int day)
public void setHMS (int hour,int minute,int second)
public static final Timestamp getTimestamp (Object obj) throws Exception
public static String getDate (long time) throws Exception
public static String getHTTP_date (long time) throws Exception
public static long parseHTTP_date (String s) throws Exception
public void in (DataInputStream is) throws Exception
public void out (DataOutputStream os) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.13 Class db.jdbc.interval.Year

```
public class Year
    implements Adapter, DataAdapterCA,StringAdapter, Comparable, Serializable

    public Year ()
    public Year (int year)
    public Year (String s) throws Exception
```

```
public void set (String value) throws Exception;
public int getYear ()
public Object negate ()
public void multiply (int value)
public void divide (int value) throws Exception

public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.14 Class db.jdbc.interval.Month

```
public class Month
    implements  Adapter, DataAdapterCA,StringAdapter,Serializable, Comparable

public Month ()
public Month (int month)
public Month (String s) throws Exception

public int getMonth ()
public Object negate ()
public void multiply (int value)
public void divide (int value) throws Exception
public void set (String value) throws Exception
public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.15 Class db.jdbc.interval.YearMonth

```
public class YearMonth
    implements Adapter,DataAdapterCA,StringAdapter, Serializable, Comparable

public YearMonth ()
```

```
public YearMonth (int year,int month)
public YearMonth (String s) throws Exception

public int getYear ()
public int getMonth ()
public Object negate ()
public void multiply (int value)
public void divide (int value) throws Exception
public void set (String value) throws Exception
public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.16 Class db.jdbc.interval.Day

```
public class Day
    implements Adapter, DataAdapterCA,StringAdapter,Serializable, Comparable

    public Day ()
    public Day (int day)
    public Day (String s) throws Exception

    public int getDay ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.17 Class db.jdbc.interval.DayHour

```
public class DayHour
    implements Adapter, DataAdapterCA,StringAdapter,Serializable, Comparable
```

```
public DayHour ()
public DayHour (int day,int hour)
public DayHour (String s) throws Exception

public int getDay ()
public int getHour ()
public Object negate ()
public void multiply (int value)
public void divide (int value) throws Exception
public void set (String value) throws Exception
public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.18 Class db.jdbc.interval.DayMinute

```
public class DayMinute
    implements Adapter,DataAdapterCA,StringAdapter, Serializable, Comparable

    public DayMinute ()
    public DayMinute (int day,int hour,int minute)
    public DayMinute (String s) throws Exception

    public int getDay ()
    public int getHour ()
    public int getMinute ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.19 Class db.jdbc.interval.DaySecond

```
public class DaySecond
    implements Adapter,DataAdapterCA,StringAdapter, Serializable, Comparable

    public DaySecond ()
    public DaySecond (int precision)
    public DaySecond (int day,int hour,int minute,int second, int precision)
    public DaySecond (int day,int hour,int minute,int second,int fraction,int precision)
    public DaySecond (String s) throws Exception

    public int getDay ()
    public int getHour ()
    public int getMinute ()
    public int getSecond ()
    public int getFraction ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.20 Class db.jdbc.interval.Hour

```
public class Hour
    implements Adapter, DataAdapterCA,StringAdapter,Serializable, Comparable

    public Hour ()
    public Hour (int hour)
    public Hour (String s) throws Exception

    public int getHour ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
```

```
public void set (String value) throws Exception
public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.21 Class db.jdbc.interval.HourMinute

```
public class HourMinute
    implements Adapter,DataAdapterCA,StringAdapter, Serializable, Comparable

    public HourMinute ()
    public HourMinute (int hour,int minute)
    public HourMinute (String s) throws Exception

    public int getHour ()
    public int getMinute ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.22 Class db.jdbc.interval.HourSecond

```
public class HourSecond
    implements Adapter,DataAdapterCA,StringAdapter, Serializable, Comparable

    public HourSecond ()
    public HourSecond (int precision)
    public HourSecond (int hour,int minute,int second, int precision)
    public HourSecond (int hour,int minute,int second,int fraction,int precision)
    public HourSecond (String s) throws Exception
```

```
public int getHour ()
public int getMinute ()
public int getSecond ()
public int getFraction ()
public Object negate ()
public void multiply (int value)
public void divide (int value) throws Exception
public void set (String value) throws Exception
public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.23 Class db.jdbc.interval.Minute

```
public class Minute
    implements Adapter, DataAdapterCA,StringAdapter,Serializable, Comparable

    public Minute ()
    public Minute (int minute)
    public Minute (String s) throws Exception

    public int getMinute ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.24 Class db.jdbc.interval.MinuteSecond

```
public class MinuteSecond
    implements Adapter, DataAdapterCA,StringAdapter,Serializable, Comparable
```

```
public MinuteSecond ()
public MinuteSecond (int precision)
public MinuteSecond (int minute,int second,int precision)
public MinuteSecond (int minute,int second,int fraction, int precision)
public MinuteSecond (String s) throws Exception

public int getMinute ()
public int getSecond ()
public int getFraction ()
public Object negate ()
public void multiply (int value)
public void divide (int value) throws Exception
public void set (String value) throws Exception
public void in (DataInputStream is, DataType dt) throws Exception
public void out (DataOutputStream os, DataType dt) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.25 Class db.jdbc.interval.Second

```
public class Second
    implements Adapter,DataAdapterCA,StringAdapter,  Serializable, Comparable

    public Second ()
    public Second (int precision)
    public Second (int second,int precision)
    public Second (int second,int fraction,int precision)
    public Second (String s) throws Exception

    public int getSecond ()
    public int getFraction ()
    public Object negate ()
    public void multiply (int value)
    public void divide (int value) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is, DataType dt) throws Exception
    public void out (DataOutputStream os, DataType dt) throws Exception
    public int compareTo (Object another)
```

```
    public boolean equals (Object another)
    public String toString ()
```

4.3.26 Class db.jdbc.type.Money

```
public class Money
    extends Numeric

    public Money ()
    public Money (long money)
    public Money (long g, byte scale)
    public Money (double money) throws Exception
    public Money (String s) throws Exception

    public String toString ()
```

4.3.27 Class db.jdbc.type.PCode

```
public class PCode
    implements DataAdapter,StringAdapter, Comparable, Serializable

    public PCode ()
    public PCode (long code) throws Exception
    public PCode (String s) throws Exception

    public long getCode ()
    public boolean isWoman ()
    public String getSex ()
        //"女" 或 "男"
    public Date getBirthday ()
    public static void check (String s) throws Exception
    public void set (String value) throws Exception
    public void in (DataInputStream is) throws Exception
    public void out (DataOutputStream os) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.28 Class db.jdbc.type.IP

```
public class IP
    implements DataAdapter, StringAdapter, Comparable,java.io.Serializable

    public IP ()
    public IP (byte[] address)
    public IP (int n)
    public IP (String host) throws Exception

    public byte[] getBytes () {
    public int getInt () {
    public String get (int n)
    public byte getByte (int n)
    public void set (String value) throws Exception
    public void in (DataInputStream is) throws Exception
    public void out (DataOutputStream os) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.29 Class db.jdbc.type.Version

```
public class Version
    implements DataAdapter,StringAdapter, Comparable,java.io.Serializable

    public Version ()
    public Version (byte[] VERSION)
    public Version (String version) throws Exception

    public byte[] getBytes ()
    public void set (String value) throws Exception
    public void in (DataInputStream is) throws Exception
    public void out (DataOutputStream os) throws Exception
    public int compareTo (Object another)
    public boolean equals (Object another)
    public String toString ()
```

4.3.30 Class db.jdbc.type.Array

```
public class Array
    implements DataAdapter,StringAdapter,Comparable, Serializable
    public Array (ArrayType atype) throws Exception
    public Array (ArrayType atype, Array source) throws Exception
    public Array (ArrayType atype, ObjectArray values) throws Exception

    public ArrayType getArrayType ()
    public void updateArray (Array source) throws Exception
    public void updateValues (ObjectArray values) throws Exception
    public void insertColumn (Object obj) throws Exception
    public void insertColumn (int col, Object obj) throws Exception
    public void deleteColumn (int col) throws Exception
    public String getString (int col) throws Exception
    public char getChar (int col) throws Exception
    public boolean getBoolean (int col) throws Exception
    public byte getByte (int col) throws Exception
    public short getShort (int col) throws Exception
    public int getInt (int col) throws Exception
    public long getLong (int col) throws Exception
    public float getFloat (int col) throws Exception
    public double getDouble (int col) throws Exception
    public Date getDate (int col) throws Exception
    public Time getTime (int col) throws Exception
    public Timestamp getTimestamp (int col) throws Exception
    public InputStream getBinaryStream (int col) throws Exception
    public byte[] getBytes (int col) throws Exception
    public Object getObject (int col) throws Exception
    public void updateBoolean (int col,boolean t) throws Exception
    public void updateByte (int col,byte value) throws Exception
    public void updateShort (int col,short value) throws Exception
    public void updateInt (int col,int value) throws Exception
    public void updateLong (int col,long value) throws Exception
    public void updateFloat (int col,float value) throws Exception
    public void updateDouble (int col,double value) throws Exception
    public void updateDate (int col,Date value) throws Exception
    public void updateTime (int col,Time value) throws Exception
```

```
public void updateTimestamp (int col,Timestamp value) throws Exception
public void updateString (int col,String value) throws Exception
public void updateChar (int col,char value) throws Exception
public void updateBytes (int col, byte[] value) throws Exception
public void updateObject (int col, Object obj) throws Exception
public byte[] read () throws Exception
public void write (byte[] buf) throws Exception
public void in (DataInputStream is) throws Exception
public void out (DataOutputStream os) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

4.3.31 Class db.jdbc.Record

```
public class Record
    implements Result, DataAdapter

    public Record (ResultSetMetaData hd) throws Exception
    public Record (ResultSetMetaData hd, Object[] objs) throws Exception
    public Record (ResultSetMetaData hd, Record source) throws Exception

    public ResultSetMetaData getMetaData ()
    public void insertColumn (Object obj) throws Exception
    public void insertColumn (int col, Object obj) throws Exception
    public void deleteColumn (int col) throws Exception
    public boolean wasNull ()
    public String getString (int col) throws Exception
    public char getChar (int col) throws Exception
    public boolean getBoolean (int col) throws Exception
    public byte getByte (int col) throws Exception
    public short getShort (int col) throws Exception
    public int getInt (int col) throws Exception
    public long getLong (int col) throws Exception
    public float getFloat (int col) throws Exception
    public double getDouble (int col) throws Exception
    public Date getDate (int col) throws Exception
    public Time getTime (int col) throws Exception
    public Timestamp getTimestamp (int col) throws Exception
```

```
public byte[] getBytes (int col) throws Exception
public InputStream getBinaryStream (int col) throws Exception
public Object getObject (int col) throws Exception

public void updateNull (int col) throws Exception
public void updateBoolean (int col, boolean t) throws Exception
public void updateBoolean (int col) throws Exception
public void updateChar (int col, char value) throws Exception
public void updateByte (int col, byte value) throws Exception
public void updateShort (int col, short value) throws Exception
public void updateInt (int col, int value) throws Exception
public void updateLong (int col, long value) throws Exception
public void updateFloat (int col, float value) throws Exception
public void updateDouble (int col, double value) throws Exception
public void updateDate (int col, Date value) throws Exception
public void updateTime (int col,Time value) throws Exception
public void updateTimestamp (int col,Timestamp value) throws Exception
public void updateString (int col, String value) throws Exception
public void updateBytes (int col, byte[] buf) throws Exception {
public void updateObject (int col, Object obj) throws Exception
public byte[] read () throws Exception
public void write (byte[] buf) throws Exception
public void in (DataInputStream is) throws Exception
public void out (DataOutputStream os) throws Exception
public int compareTo (Object another)
public boolean equals (Object another)
public String toString ()
```

5. Huayisoft DB Server 数据库的查询概述

5.1 Huayisoft DB Server 查询的总体说明

查询表达式：从一个或多个数据库表中提取数据。

语法：

```
<查询> :: = <查询表达式> [ <ORDER BY子句> ] [ <INTO子句> ]

<子查询> ::= ( <查询> )

<查询表达式> ::= { <无连接查询表达式> | <连接表> }

<无连接查询表达式> ::= <无连接查询>
                        |<查询表达式> UNION [ ALL ] [<列对应说明>] <查询项>
                        | <查询表达式> EXCEPT [ ALL ] [<列对应说明> <查询项>

<列对应说明> ::= CORRESPONDING [ BY ( <列名列表>) ]

<无连接查询> ::= <无连接主查询>
                 | <主查询> INTERSECT [ ALL ] [<列对应说明> <查询项>

<无连接主查询> ::= <简单表>
                   | ( <无连接查询表达式> )

<主查询> ::= <无连接主查询> | <连接表>

<查询项> ::= <无连接查询> | <连接表>

<简单表> ::= <查询说明>
             | <表值构造>
             | <直接表>
             | <文本记录>

<查询说明> ::= SELECT [DISTINCT | ALL ] <选择项列表> [<表表达式>] [<ORDER BY 子句>]

<选择项列表> ::= * | <选择项> [ ,<选择项>]...
```

<选择项> ::= 表名 .* | <值表达式> [AS] 列别名

<表表达式> ::= <FROM子句>
| <WHERE子句>
| <GROUP BY子句>
| <HAVING子句>
| <LIMIT子句>

<FROM子句> ::= FROM <表参考> [,表参考]...

<表参考> ::= <主表> | <连接表>

<主表> ::= 表名 [[AS] 表别名 [(<列名列表>)]]

<WHERE 子句> ::= WHERE <搜索条件>

<GROUP BY 子句> ::= GROUP BY 列名 | <值表达式> [,列名 | <值表达式>]...

<HAVING 子句> ::= HAVING <搜索条件>

<LIMIT子句> ::= LIMIT [起始行 ,] 行数

<行值构造单元> ::= NULL | DEFAULT | <值表达式>

<直接表> ::= TABLE 表名

<连接表> ::= <交叉连接表>
| <连接表>
| <自然连接表>
| <合并连接表>

<交叉连接表> ::= <表参考> CROSS JOIN <主表>

<连接表> ::= <表参考> [<连接类型>] JOIN <表参考> <连接说明>

<自然连接表> ::= <表参考> NATURAL [<连接类型>] JOIN <主表>

<合并连接表> ::= <表参考> UNION JOIN <主表>

```
<连接类型> ::= INNER | <外连接类型> [ OUTER]

<外连接类型> ::= LEFT | RIGHT | FULL

<连接说明> ::= ON <搜索条件> | USING ( <列名列表> )

<ORDER BY子句> ::= ORDER BY
                    { 列名 | 列号 } [ ASC | DESC ]
                   [,{ 列名 | 列号 } [ ASC | DESC ] ]...
                   [ BY DICTIONARY ]

<INTO子句> ::= INTO { TABLE临时表名 | TEXT 文本文件名 }]

<搜索条件> ::= <布尔表达式>
```

注意：

<表值构造>及<文本记录>请参考<插入记录语句>

5.2 单表查询

5.2.1 SELECT 子句

选择项可以是下列之一：

(a) FROM子句所指定的表中的列名。

(b) 常量。

(c) 表达式。

1, 查看指定字段的数据

列出每个销售点的编号、城市名称、所属区域、经理、销售定额、销售额

```
SELECT 编号,城市名称,所属区域,经理,销售定额,销售额
    FROM 销售点
```

编号	城市名称	所属区域	经理	销售定额	销售额
1	北京	华北	108	300000	186042
2	南昌	华中	104	575000	692637
3	合肥	华中	105	800000	735042

4	武汉	华中	101	350000	367911
5	天津	华北	108	725000	835915

2，选择所有列 （SELECT *） *：表示 FROM 表中的全部字段

列出每个销售点所有信息

```
SELECT  *
    FROM  销售点
```

编号	城市名称	所属区域	经理	销售定额	销售额
1	北京	华北	108	300000	186042
2	南昌	华中	104	575000	692637
3	合肥	华中	105	800000	735042
4	武汉	华中	101	350000	367911
5	天津	华北	108	725000	835915

3，常量可以直接成为选择项

列出每个销售点的城市名称、销售定额

```
SELECT  城市名称, '的销售定额是',销售定额
    FROM  销售点
```

城市名称	的销售定额是	销售定额
北京	的销售定额是	300000
南昌	的销售定额是	575000
合肥	的销售定额是	800000
武汉	的销售定额是	350000
天津	的销售定额是	725000

4，去重 (DISTINCT)，即查询结果中没有相同的列。

列出每个销售点经理的编号

```
SELECT DISTINCT 经理
    FROM  销售点
```

经理

```
-----
108
104
105
101
```

5, 字段计算

列出每个销售点的城市名称及销售额与销售定额之差

```
SELECT 城市名称, 销售额 - 销售定额
    FROM 销售点

城市名称  销售额-销售定额
----------  --------------------
北京        -113958
南昌         117637
合肥         -64958
武汉         17911
天津         110915
```

6, 给选择项另取一个更富于意义的名称

列出每个销售点经理的编号

```
SELECT DISTINCT 经理 AS 经理编号
    FROM 销售点

经理编号
-----
108
104
105
101
```

5.2.2 记录选择（WHERE 子句）

如上述所示，缺省 WHERE 子句，表示永真条件，即选择 FROM 表中所有的记录。

1, 比较测试

操作符说明如下：

操作符	说明
=	等于
<>	不等于
>	大于
>=	大于等于
<	小于
<=	小于等于

列出北京的经理编号

```
SELECT 经理
    FROM  销售点
    WHERE  城市名称='北京'
```

```
经理
-----
108
```

2, 范围测试（BETWEEN）

列出金额在 2 万至 3 万之内的订单

```
SELECT 编号, 金额
    FROM  订单
    WHERE 金额  BETWEEN 20000 AND 29999.99
```

```
编号    金额
-----   -------
8236    22500
8187    27500
8242    22500
```

以上 BETWEEN 完全等价于：

```
    金额  >= 20000 AND 金额<=29999.99
```

3，组成员测试([NOT] IN)

列出在北京、天津工作的销售员

```
SELECT 编号,姓名
    FROM 销售员
    WHERE 销售点 IN (1,5)
```

```
编号    姓名
-----    --------
103    杨刚
108 李立发
110    蔡晓
```

以上 IN 完全等价于：

```
销售点=1 OR 销售点=5
```

4，模式匹配词（[NOT] LIKE）

查询所有姓’柯’ 销售员

```
SELECT 编号,姓名
    FROM 销售员
    WHERE 姓名 LIKE  '柯%'
```

```
编号    姓名
-----    --------
101    柯长富
```

这里百分符（%）表示任何 0 个或多个字符的字符串，下划线（_）表示任何单个字符。

5，空值测试（IS [NOT] NULL）

查询还没有分配到销售点的销售员

```
SELECT 编号,姓名
    FROM 销售员
    WHERE 销售点 IS NULL
```

```
编号    姓名
-----    --------
107    夏大宝
```

注：“ 销售点 = NULL ” 是非法的。

6，复合搜索条件（AND，OR，NOT）

WHERE 中可以包含多个搜索条件。搜索条件之间用 AND 或者 OR 连接。

列出在北京或天津工作的销售员

```
SELECT 编号,姓名
    FROM 销售员
    WHERE 销售点=1 OR 销售点=5
```

```
编号    姓名
-----    --------
103    杨刚
109    李立发
110    蔡晓
```

5.2.3 对查询结果显示顺序的控制（ORDER BY 子句）

对查询结果按一个或多个列进行排序.排序列必需在查询结果中

列出每个销售点的城市名称、销售额,并按销售额从大到小排序

```
SELECT 城市名称, 销售额
    FROM 销售点
    ORDER BY 销售额 DESC
```

```
城市名称    销售额
----------    ------------
天津        835915
合肥        735042
南昌        692637
武汉        367911
北京        186042
```

其中关键字 DESC 表明按销售额降序排列，如果是升序，关键字 ASC 可以省略。排序列可以

不止一个，按其排列的先后顺序决定其主序列。

5.3 多表查询（连接查询）

两个或两个以上表的查询称为多表查询。

5.3.1 等连接查询

列出销售点及其经理的姓名

```
SELECT 城市名称, 姓名
    FROM 销售点, 销售员
    WHERE 经理=销售员.编号
```

```
城市名称    姓名
----------    ----------
北京        李立发
南昌        张民山
合肥        罗明贵
武汉        柯长富
天津        李立发
```

这里要查询销售点经理的姓名，必须涉及到两个表：销售点表及销售员表，连接条件是销售点表中经理代号必须与销售员表的编号相同。

注意：其中 FROM 子句（列出查询数据所在的表）中的表与表之间要用逗号’,’ 隔开，连接条件需要用表名（销售员）作为前缀以便区分同名字段（编号）。

在 FROM子句中可以给表另取一个更便于使用的名称，上述语句可以写成：

```
SELECT 城市名称, 姓名
    FROM 销售点, 销售员 AS B
    WHERE 经理=B.编号
```

5.3.2 多个匹配字段

列出销售点及其经理的姓名

```
SELECT 编号, 产品名称, 金额
    FROM 订单 A，产品 B
```

```
WHERE    A.类别=B.类别
    AND A.号=B.代号
```

```
编号    产品名称       金额
------  ------------    ------
8161    笔记本电脑    31500
8212    冲电器 B       3745
8189    化妆品 A       1458
.
.
.
```

注：必须使用 AND操作符连接多个连接条件。

5.3.3 内连接

INNER JOIN 表明查询结果仅包含两个表中相匹配的行。

列出销售点及其经理的姓名

```
SELECT 城市名称, 姓名
    FROM 销售点 INNER JOIN 销售员
        ON 经理=销售员.编号
```

```
城市名称    姓名
-----------    ------------
北京          李立发
南昌          张民山
合肥          罗明贵
武汉          柯长富
天津          李立发
```

从这个查询结果中可以看出与上述‘5.3.1 等连接查询’中的结果是一样的；但是，这里使用 JOIN 操作明确地指明要连接的二个表，并把它们的连接条件放在 ON子句中，

连接条件 JOIN,查询条件 WHERE 是分开的。

5.3.4 外连接

LEFT JOIN 表明查询结果中包含左表中的所有行以及与右表相匹配的行。如果与右表中不匹配时，将用空值来代替。

RIGHT JOIN 表明查询结果中包含右表中的所有行以及与左表相匹配的行。如果与左表中不匹配时，将用空值来代替。建议把 RIGHT JICN 语句转化为 LEFT JOIN 语句来使用。

FULL JOIN 表明查询结果中包含两个表中相匹配的和不匹配的所有行。

列出销售员及其他们的城市名称

```
SELECT 编号,姓名, 年龄,城市名称
    FROM 销售员 LEFT JOIN 销售点
        ON 销售员.销售点 =销售点.编号
```

```
编号   姓名      城市名称
-----   --------   ----------
101    柯长富    武汉
102    王美丽    南昌
103    杨刚      天津
104    张民山    南昌
105    罗明贵    合肥
106    刘应东    合肥
107    夏大宝    NULL
108    李立发    天津
109    吴小虎    合肥
110    蔡晓      北京
```

列出销售员及其他们的城市名称

```
SELECT 编号,姓名, 年龄,城市名称
    FROM 销售点 RIGHT JOIN 销售员
        ON 销售点.编号= 销售员.销售点
```

```
编号   姓名      城市名称
-----   --------   ----------
101    柯长富    武汉
102    王美丽    南昌
103    杨刚      天津
104    张民山    南昌
```

105　罗明贵　合肥
106　刘应东　合肥
107　夏大宝　NULL
108　李立发　天津
109　吴小虎　合肥
110　蔡晓　　北京

以上左连接与右连接其结果是一样的，也就是说，只要主表次表关系不变，书写成左连接还是右连接是没有关系的。

5.4 统计查询

统计查询是通过集函数以及 GROUP BY 分组子句来完成的。

5.4.1 集函数

HUAYISOFT DB SERVER 支持标准集函数：即求和 (SUM)、最大值 (MAX)、最小值(MIN)、平均值 (AVG)、记录数(COUNT)。语法如下：

```
语法：
  <集函数 > ::= COUNT (*)
              | AVG  ( <选择项> )
              | COUNT  ( <选择项> )
              | MIN  ( <选择项> )
              | MAX  ( <选择项> )
              | SUM  ( <选择项> )
```

1，求数值型字段的平均值(AVG)

求茶叶的平均价格

```
SELECT  AVG (价格)
  FROM 产品
      WHERE 类别= '茶叶'

AVG (价格)
---------------
352.33
```

2，求选择项中的最小值和最大值（MIN 和 MAX）。

求茶叶的最小价格和最大价格

```
SELECT    MIN (价格),MAX(价格)
  FROM  产品
        WHERE  类别= '茶叶'

MIN (价格)    MAX(价格)
--------------    --------------
180             652
```

3，求一列数据的和(SUM)

求所有销售员的销售总额

```
SELECT    SUM (销售额)
  FROM  销售员
        WHERE  类别= '茶叶'

SUM (销售额)
---------------
2893532
```

4，求一列数据的和(COUNT)

COUNT (*) 统计查询结果中的记录数。

COUNT (<选择项>),统计一列中选择项的个数。

求销售员的人数

```
SELECT    COUNT (*)
  FROM  销售员

COUNT (*)
---------------
10
```

求已编号的销售员人数

```
SELECT    COUNT (编号)
  FROM  销售员
```

```
COUNT (编号)
---------------
10
```

求已分配销售点的销售员人数

```
SELECT    COUNT (销售点)
   FROM  销售员

COUNT (销售点)
---------------
9
```

5.4.2 分组查询（GROUP BY 子句）

可以对一个或多个列进行分组。

通常与集函数配合使用

由于分组最终是要为每组计算出一个结果元组（而且每组也只能有一个元组），而聚集函数正是实现这一目的的，因此集函数大量地被使用在分组场合。

计算销售员订单的平均金额

```
SELECT    销售员,AVG (金额)
   FROM  销售员
         GROUP BY  销售员

销售员      AVG (金额)
--------       -----------------
101          7865.40
102          3552.50
103          5694
104          16479
106          8876
107          11566
108          8376
109          1350
110          11477.33
```

注意 SELECT 目标列之间的一致性

能够从每组元组中计算出结果元组的，除了集函数外，还可以是分组列，其它的任何内容都不能成为结果元组的组成部分。也就是说分组的计算结果必须是描述组的属性。

5.4.3 分组搜索条件（HAVING 子句）

HAVING 条件必须是刻划分组的属性或其表达式，或者是集函数。

计算销售员订单的平均金额大于 10000 的平均金额

```
SELECT   销售员,AVG (金额)
  FROM  销售员
      GROUP BY  销售员
          HAVING    AVG (金额) > 10000
```

```
销售员     AVG (金额)
--------    -----------------
104         16479
107         11566
110         11477.33
```

分清 WHERE 与 HAVING 的作用

WHERE 条件是对原始表的元组的限制条件，而 HAVING 条件是对分组后的中间表的元组的限制，所以一般情况下二者是不可混淆也不可交换的。但有一个特例，既 HAVING 条件是单纯的作用于分组属性上的条件，这时可以将其转化为 WHERE 条件以提高执行效率。

计算销售员 101 订单的平均金额

```
SELECT   销售员,AVG (金额)
  FROM  销售员
      GROUP BY  销售员
  HAVING  销售员=101
```

```
销售员     AVG (金额)
--------    -----------------
101         7865.40
```

该查询等价于:

```
SELECT   销售员,AVG (金额)
    FROM  销售员
```

```
        WHERE  销售员=101
    GROUP BY    销售员
```

```
销售员      AVG (金额)
--------    -----------------
101         7865.40
```

5.5 合并查询结果（UNION 子查询）

查询结果包含主查询结果以及 UNION 子查询的查询结果. 可以包含多个 UNION 子句。

列出单价超出 2000 或在一个订单中订购了超过 30000 元所有产品

```
SELECT 类别, 代号
    FROM  订单
        WHERE 金额>30000
UNION
SELECT DISTINCT 类别, 代号
    FROM  产品
        WHERE 价格>2000
```

```
类别     代号
------   ------
电脑      A02
电脑      A03
玉器      D02
手机      B01
手机      B02
```

UNION 子句遵循以下规则：

主查询以及各 UNION 子查询必须包含相同的结果列。

各结果列的类型及宽度必须相同。

5.6 显示指定范围的记录（LIMIT 子句）

列出头 3 个销售员的姓名

```
    SELECT  姓名
        FROM  销售员
         LIMIT   1,3
```

```
姓名
--------
柯长富
王美丽
杨刚
```

6. Huayisoft DB Server 的数据定义

数据定义语言 (Data Definition Language,DDL)用以完成数据库中各种对象的定义。包括数据库、用户、模式、基本表、索引、视图等。HUAYISOFT DB SERVER使用 CREATE、ALTER、DROP语句来对数据库结构进行动态地定义、修改与删除。

语法：

```
<数据定义> ::= <数据库定义>
             | <用户定义>
             | <模式定义>
             | <表定义>
             | <抽象数据类型定义>
             | <数据域定义>
             | <断言定义>
             | <索引定义>
             | <视图定义>
             | <记录文本定义>
```

6.1 数据库定义

Huayisoft DB Server 数据库 (catalog)是数据库模式（schema）的集合。

语法：

```
<数据库定义> ::= <创建数据库语句>
               | <删除数据库语句>
```

6.1.1 创建数据库

语法：

```
<创建数据库语句> ::= CREATE { CATALOG | DATABASE } [IF NOT EXISTS] 数据库名 路径名
```

HUAYISOFT DB SERVER中的一个数据库对应一个路径。如果要创建的数据库已经存在，将产生一个警告，使用“ IF NOT EXISTS”将不会产生一个警告。

6.1.2 删除数据库

语法：

```
<删除数据库语句> ::= DROP { CATALOG | DATABASE } [IF EXISTS] 数据库名
```

如果删除的数据库名不存在，将出现一个警告，使用“ IF EXISTS ”将不会出现警告。

6.1.3 设置当前数据库

语法：

```
<设置当前数据库语句> ::= SET DATABASE | CATALOG 数据库名
```

6.2 用户定义

语法：

```
<用户定义> ::= <创建用户语句>
             | <修改用户语句>
             | <删除用户语句>
```

6.2.1 创建用户

语法：

```
<创建用户语句> ::= CREATE USER [IF NOT EXISTS] 用户名 密码
                   [ <修改用户子句> ]
```

如果要创建的用户已经存在，将产生一个警告，使用“ IF NOT EXISTS”将不会产生一个警告。

Huayisoft DB Server 华易数据库管理系统安全策略：

6.2.2 修改用户

语法：

```
<修改用户语句> ::= ALTER USER 用户名 <修改用户子句>

<修改用户子句>  ::=  <修改用户值>  [,<修改用户值>] …

<修改用户值>  ::=  <用户列名> [ 新值 ]

<用户列名>  ::=  NAME
               | USERNAME
               | PASSWROD
               | SCHEMA
```

```
        | BOOT
        | MENU
        | STARTDATE
        | ENDDATE
        | SERVICE
```

其中： NAME 为用户的真实姓名；
USERNAME 为用户名称；
PASSWROD 为用户密码；
SCHEMA 为缺省的模式名称；
BOOT 为用户登录类名称；
MENU 为用户菜单名称；
STARTDATE 为用户账号使用期的起始日期；
ENDDATE 为用户账号使用期的结束日期；
SERVICE 是否为用户提供某服务（如：FTP ）标记；

6.2.3 删除用户

```
语法：
  <删除用户语句> ::= DROP USER [IF EXISTS] 用户名
```

如果删除的用户名不存在，将出现一个警告，使用“ IF EXISTS”将不会出现警告。

6.3 模式定义

Huayisoft DB Server 模式 (schema)是表 (table)、视图 (view)、权限 (privilege)等数据库对象的集合。

```
语法：
  <模式定义> ::=  <创建模式语句>
                | <删除模式语句>
```

6.3.1 创建模式

```
语法：
  <创建模式语句> ::= CREATE SCHEMA [IF NOT EXISTS]
                { 模式名 [AUTHORIZATION 用户名 ] | [模式名 ] AUTHORIZATION用户名 }
                [ <模式单元> ]...
```

```
<模式单元> ::= <表定义>
             | <视图定义>
             | <数据域定义>
             | <断言定义>
             | <触发器定义>
             | <抽象数据类型定义>
             | <授权语句>
```

如果要创建的已经模式存在，将产生一个警告，使用“ IF NOT EXISTS”将不会产生一个警告。

使用 CREATE SCHEMA语句可以方便地创建有引用循环 (利用主键字 /外键字对 ,两个或多个表可以互相引用)的表

如：

```
CREATE SCHEMA REF_CYCLE
CREATE TABLE PRIMARY_TABLE
  (CODE INTEGER NOT NULL PRIMARY KEY,
   ID INTEGER REFERENCES FOREIGN)

CREATE TABLE FOREIGN_TABLE
   (ID INTEGER NOT NULL PRIMARY KEY,
   CODE INTEGER REFERENCES PRIMARY_TABLE)
```

6.3.2 删除模式

语法：

```
<删除模式语句> ::= DROP SCHEMA [IF EXISTS] 模式名
```

如果删除的模式名不存在，将出现一个警告，使用“ IF EXISTS”将不会出现警告。

6.3.3 设置当前模式

语法：

```
<设置当前数据库语句> ::= SET SCHEMA模式名
```

6.4 表定义

语法：

```
<表定义> ::= <创建表语句>
           | <修改表语句>
           | <删除表语句>
```

注意：Huayisoft DB Server 中的 .dbf文件与其它数据库的 .dbf文件除了在文件后缀名相同之外，并不存在任何关联，它们之间是不能互用的。

6.4.1 创建表

语法：

```
<创建表语句> ::= CREATE TABLE [IF NOT EXISTS] 表名
                 { AS <查询> | ( <表单元> [,<表单元>]... ) }

<表单元> ::= <列定义> | <表约束定义>

<列定义> ::= 列名 { 数据类型 | 域名 } [ DEFUALT 值 ] [<列约束定义>
                                                    [,<列约束定义>]... ]

<列约束定义> ::= [ CONSTRAINT 约束名称 ] <列约束> [ <约束属性> ]

<列约束> ::= NOT NULL
           | PRIMARY KEY
           | <外键字约束>
           | UNIQUE
           | <检查约束定义>

<外键字约束> ::= REFERENCES 表名 [ <列列表> ]
                 [ MATCH { FULL | PARTIAL | SIMPLE } ]
                 [ CASCADE | SET NULL | SET DEFAULT | NO ACTION ]

<表约束定义> ::= [ CONSTRAINT 约束名 ] <表约束> [ <约束属性> ]

<表约束> ::= <主键字约束定义>
           | <外键字约束定义>
           | <唯一性约束定义>
           | <检查约束定义>

<主键字约束定义> ::= PRIMARY KEY ( <列名列表> )
```

<外键字约束定义> ::= FOREIGN KEY (<列名列表>) <外键字约束>

<唯一性约束定义> ::= UNIQUE (<列名列表>)

<检查约束定义> ::= CHECK (<搜索条件>)

<列名列表> ::= 列名 [, 列名]...

<约束属性> ::= [NOT] DEFERRABLE { INITIALLY DEFERRED | INITIALLY IMMEDIATE }

如果要创建的基本表已经存在，将产生一个警告，使用“ IF NOT EXISTS”将不会产生一个警告。

6.4.2 修改表

语法：

<修改表语句> ::= ALTER TABLE 表名 <修改表操作>

<修改表操作> ::= ADD [COLUMN] <列定义>
| INSERT 列插入位置 <列定义>
| ALTER [COLUMN] 列名 SET DEFAULT 缺省值
| ALTER [COLUMN] 列名 DROP DEFAULT
| DROP [COLUMN] 列名
| ADD <表约束定义>
| DROP CONSTRAINT 约束名称

以上分别为：

加入一列
在指定列位置插入一列
设置列缺省值
删除列缺省值
删除指定的列
加入表约束
删除表约束

6.4.3 删除表

语法：

<删除表语句> ::= DROP TABLE [IF EXISTS] 表名 [,表名]…

如果删除的基本表不存在，将出现一个警告，使用“ IF EXISTS”将不会出现警告。

6.5 抽象数据类型定义

语法：

<抽象数据类型定义> ::= <创建抽象数据类型语句>
| <删除抽象数据类型语句>

6.5.1 创建抽象数据类型

语法：

<创建抽象数据类型语句> ::= CREATE TYPE 类型名 AS <数据类型> CLASS 类对象名

6.5.2 删除指定的自定义抽象数据类型

语法：

<删除抽象数据类型语句> ::= DROP TYPE [IF EXISTS] 类型名

6.6 数据域定义

语法：

<数据域定义> ::= <创建数据域语句>
| <删除数据域语句>

6.6.1 创建数据域

语法：

<创建数据域语句> ::= CREATE DOMAIN 域名
AS] <数据类型> [DEFAULT 值] <域约束定义>

<域约束定义> ::= [CONSTRAINT 约束名] <检查约束定义> [<约束属性>]

6.6.2 删除数据域

语法：

<删除数据域语句> ::= DROP DOMAIN [IF EXISTS] 域名

6.7 断言定义

语法：

<断言定义> ::= <创建断言语句>
| <删除断言语句>

6.7.1 创建断言

语法： <创建断言语句> ::= CREATE ASSERTION 断言名 <检查约束定义> [<约束属性>]

6.7.2 删除断言

语法：

<删除断言语句> ::= DROP ASSERTION [IF EXISTS] 断言名

6.8 索引定义

语法：

<索引定义> ::= <创建索引语句>
| <删除索引语句>

6.8.1 创建索引

语法：

<创建索引语句> ::= CREATE [UNIQUE] INDEX [IF NOT EXISTS] 索引名
ON 表名 (列名 [DESC | ASC] [,列名 [DESC | ASC]] ...)

其中关键字 DESC表明按降序排列，如果是升序，关键字 ASC可以省略。

(1) 索引可以建立在若干个字段上，称为组合索引。对于组合索引必须把最重要的字段加在最前面。

(2) UNIQUE选项，指出索引是唯一索引，即在索引字段上的值不能重复，唯一索引用来维护字段的唯一性。

(3) 一个表上可以建多个索引。

6.8.2 删除索引

语法：

<删除索引语句> ::= DROP INDEX [IF EXISTS] 索引名 ON 表名

6.9 视图定义

语法：

<视图定义> ::= <创建视图语句>
| <删除视图语句>

6.9.1 创建视图

语法：

<创建视图语句> ::= CREATE VIEW [IF NOT EXISTS] 视图名 [(<列名列表>)] AS 查询语句

6.9.2 删除指定的视图

语法：

<删除视图语句> ::= DROP VIEW [IF EXISTS] 视图名

6.10 记录文本定义

语法：

<记录文本定义> ::= <创建记录文本语句>

6.10.1 创建记录文本(数据库的导出)

语法：

<创建记录文本语句> ::= CREATE <文本记录> AS <查询>

<文本记录> ::= TEXT 文本文件名 [<字段分隔符>] [<记录分隔符>]

<字段分隔符> ::= FIELDS TERMINATED BY ‘字符串’

<记录分隔符> ::= LINES TERMINATED BY ‘字符串’

Huayisoft DB Server 数据库的导出是导入的逆过程，是指把 HUAYISOFT DB SERVER数据库表中的记录转换成一定格式化的 .txt 文本文件。即把查询结果的每一条记录转换成对应文本存入文本文件中 ,同时 ,在记录之间存入指定的记录分隔符 ,字段之间存入指定的字段分隔符。

缺省时 (CREATE TEXT 文本文件名 AS <查询>) ,记录分隔符是回车换行' \r\n，字段分隔符是。

最常见的转换是一条记录转换成文本文件中的一行 (DOS,Macintosh,UNIX 分别以回车换行' \r\n ,回车' \r ,换行' \n为行结束符) ,字段之间用英文逗号' ,'为隔开。

7. Huayisoft DB Server 数据库记录的插入、修改与删除

数据库记录的操作主要包括插入、修改和删除。

7.1 插入数据到表中（INSERT)

语法：

```
<插入记录语句>  ::=  INSERT INTO 表名
                     { [(<列名列表>)] <源数据子句> | DEFAULT VALUES }
                     [ AT 行位置 ]

<源数据子句> ::= <表值构造>
                 | <查询>
                 | EXECUTE语句
                 | URL_ENCODE子句
                 |<文本记录>

<表值构造> ::=  VALUES <行值构造> [,<行值构造>]...

<行值构造> ::= <行值构造单元>
               | ( <行值构造单元> [,<行值构造单元>]... )
               | <子查询>

<文本记录>  ::=  TEXT 文本文件名 [ <字段分隔符> ] [ <记录分隔符> ]

<字段分隔符>  ::=  FIELDS TERMINATED BY ‘字符串’

<记录分隔符>  ::=  LINES TERMINATED BY ‘字符串’
```

7.1.1 插入单条记录

例：新聘用了一个销售员（欧阳娜）：

```
INSERT INTO 销售员 (编号,姓名,年龄,销售点,聘用日期,经理,销售定额,销售额)
VALUES (128,'欧阳娜', 26,1, '2008.08.09',108 ,200000,0)
```

当 VALUES 子句中的常量个数与表的字段数相等，而且常量的先后次序与表中各字段的先后次序一致， 则表的字段名列表可以省略：

```
INSERT INTO 销售员
    VALUES (128,'欧阳娜',26, 1, '2008.08.09',108  ,200000,0 )
```

7.1.2 插入一个查询的结果

例：把销售员表中的编号、姓名、销售点、聘用日期等信息复制到员工信息表中：

```
INSERT INTO 员工信息 (编号 ,姓名 ,销售点 ,聘用日期 )
    SELECT编号 ,姓名 ,销售点 ,聘用日期
        FROM 销售员
```

7.1.3 在指定的行位置插入一记录

例：把新设立一个销售点（深圳）放在表的第 3行中：

```
INSERT INTO 销售点
    VALUES (88,'深圳', '华南',108  ,200000,0 )
    AT 3
```

7.1.4 数据库的导入

Huayisoft DB Server 数据库的导入是把一定格式化的 .txt 文本文件转换成一条条记录插入到指定的 Huayisoft DB Server 数据库表中。即 把文本文件中的文本内容用记录分隔符分解成一条条记录 ,且每条记录用字段分隔符分解成一个个字段 ,无条件地插入到表中 ;分解成的字段应与字段名列表一一对 .

例：开始时，把含有产品信息的文本文件中的内容装载到产品表中：

```
INSERT INTO 产品 TEXT产品 .txt
    LINES TERMINATED BY '\r\n'
    FIELDS TERMINATED BY ','
```

注意 :当上述语句应缺省时 (INSERT INTO 表名 FROM 文本文件名) ,即说明：

字段名列表为表中的所有字段的以次排列 ,文本文件中的每一行作为一条记录 (以 "\r"或 "\n" 或 "\r\n"为行结束标志) ,且字段之间应以英文逗号' ;为分隔符 ,文本文件中的字段顺序应与表名中的字段顺序相一致。

7.2 修改表中的数据(UPDATE)

语法：

```
<删除记录语句> ::= UPDATE 表名
                    SET 字段=表达式 [,字段=表达式]...
                     [ WHERE 条件 | AT行位置 ]
```

7.2.1 修改满足条件的记录

即将指定表中满足条件的记录按要求进行修改。所以 WHERE条件注意不能省略，否则将对全表作设定的修改。

例：将销售员王美丽的销售定额提高 10%:

```
UPDATE 销售员
    SET 销售定额=销售定额*1.1
      WHERE 姓名='王美丽'
```

7.2.2 修改表中的所有记录

例：将销售点的销售定额提高 10%:

```
UPDATE 销售点
  SET 销售定额=销售定额*1.1
```

7.2.3 对指定的某一记录进行修改

例：将销售员王美丽的经理改为李立发（编号=108 ）

```
UPDATE销售员
    SET经理 = 108
    AT 2
```

7.3 从表中删除数据（DELETE）

语法：

```
<删除记录语句> ::= DELETE FROM表名 [ <删除条件> ]
```

```
<删除条件> ::= WHERE <搜索条件>
             | AT 行位置 [ BY ORDER ]
```

7.3.1 删除满足条件的所有记录。

例：新聘用的销售员（欧阳娜）辞职了：

```
DELETE FROM 销售员
    WHERE 编号 =128
```

7.3.2 删除表中的所有记录。

例：删除所有订单：

```
DELETE FROM 订单
```

7.3.3 删除指定的某一记录

例：撤销临时成立的销售点（深圳）：

```
DELETE FROM 销售点
    AT 3
```

8. Huayisoft DB Server 的安全策略

8.1 数据控制

控制用户对数据的存取权限。

8.1.1 授权语句

```
语法：
  <授权语句>  ::=  GRANT 特权 ON 表名 TO 用户 [,用户 ]...
                   [ WITH GRANT OPTION]
```

其中：

‘特权’可为 SELECT、INSERT、UPDATE、DELETE、ALL PRIVILEGES。

‘用户’可以是 PUBLIC,表示为所有用户；

(a) 当特权也为 GRANT时说明授权者把 GRANT也作为一种权限授予别的用户 ,被授权的用户此时可以将获得的权限再授予别的用户；WITH GRANT OPTION子句不推荐使用；

(b) 授权者在此表名上必需拥有 GRANT特权；

(c) 授权者在此表名上必需拥有将要授出的’特权’；

8.1.2 收回权限语句

```
语法：
  <收回权限语句> ::=  REVOKE [GRANT OPTION FOR] 特权 ON 表名 FROM 用户
```

(a) 收回权限者在此表名上必需拥有 REVOKE特权；

(b) 收回权限者在此表名上必需拥有将要收回的’特权’；

9. Huayisoft DB Server 中的 GET 语句

Huayisoft DB Server 中的 GET 语句是为了简化所对应的应用而特设的。

语法：

```
<GET 语句> ::=   GET   MENU
               | GET   FTP   HOME
               | GET   FTP   ACCOUNT
               | GET   FTP   WRITE   ACCOUNT
               | GET   HTTP   HOME
               | GET   HTTP   ACCOUNT
               | GET   FILE   PATH
               | GET   FILE   NAME
```

9.1 获取用户的缺省菜单

GET MENU

9.2 获取用户的 FTP 信息

检测用户是否有 FTP 账号

GET FTP ACCOUNT

获取用户的 FTP 的路径

GET FTP HOME

检测用户是否有 FTP 写（修改）账号

GET FTP WRITE ACCOUNT

9.3 获取用户的 HTTP 信息

检测用户是否有 HTTP 账号

GET HTTP ACCOUNT

获取用户的 HTTP 的路径

GET HTTP HOME

9.4 获取用户当前目录下的文件信息

获取用户当前目录下的所有目录名称

GET FILE PATH

获取用户当前目录下的所有文件名称

GET FILE NAME

10. Huayisoft DB Server 的 LOCK、UNLOCK 语句说明

10.1 人工加锁(直到人工减锁或用户退出)

语法：

<人工加锁语句> ::= LOCK TABLES 表名 <锁> [,表名 <锁>] ...

<锁 > ::= READ [READ_LOCAL] | [LOW_PRIORITY] | WRITE

对表的读写操作进行加锁；当写操作锁加载成功时 ,任何其他用户都不能对该文件进行存取；

而当读操作锁加载成功时 ,其他用户也可以对该表进行读操作；即：多个读取操作可以共享资源 ,而写操作将独占资源。

返回对象

结果说明 :失败时抛出一错误信息

10.2 人工减锁

语法：

<人工减锁语句> ::= UNLOCK TABLES

释放所有的读写锁。

返回对象 :空

结果说明：

11. Huayisoft DB Server 中的 EXECUTE 语句

语法：

```
<EXECUTE语句> ::= EXECUTE LOGIN
                | EXECUTE BATCH SQL FROM 文本文件名
                | EXECUTE BATCH SQL STATEMENT <语句> [; <语句> ]...
                | EXECUTE CLASS [SHARE] 类名 [AS 别名 ] [ 方法名 [字符串参数 ] ]
```

11.1 执行登录获取用户的应用程序

EXECUTE LOGIN

11.2 执行批语句

执行文本文件中的批语句

EXECUTE BATCH SQL FROM SQL.txt

执行批语句

```
EXECUTE BATCH SQL STATEMENT
    SET CATALOG AAA;
    SET SCHEMA SAMPLE;
```

11.3 执行服务器端的应用程序的方法

EXECUTE CLASS hy.app.pay.server.Server getBank

注： 1,类名为不带 .class后缀的全称类名 ,如 :hy.app.pay.server.Server;
2,带 SHARE时将指示服务器把执行的应用程序放到共享集中 ,以便提高速度 .

12. Huayisoft DB Server 的编程

Huayisoft DB Server 是用纯 Java开发的，提供了 JDBC编程接口。

12.1 JDBC 应用编程接口

示例 1（下列内容可使用任何文本编辑器进行编辑，编辑好后保存到 DB1.java文件中）：

```
/**
 @(#)DB1.java 1.2 2010.10.18

 假设 db.jdbc.jar及 DB1.java文件都在 c:\sample路径中，则可使用下列命令进行编译：

    C:\sample>javac -cp db.jdbc.jar DB1.java

 编译好后，则可使用下列命令执行：

  C:\sample>java -cp db.jdbc.jar;c:\sample DB1

注： ;后面的 c:\sample不能省去，即：
        C:\sample>java -cp db.jdbc.jar DB1
        则会出现错误：提示没有发现类 DB1
 **/

import java.sql.*;
import db.jdbc.*;
import db.jdbc.x.*;

public class DB1
{

   public static void main (String args [])
   {
       String driverName = "db.jdbc.x.jDriver";
       String url = Def.DB_PROTOCOL + "//localhost:2000/AAA";
       String username = "test";
       String password = "test";
```

```
        String sql = "SELECT * FROM SAMPLE.OFFICES";

        try {
            Driver driver = (Driver)Class.forName (driverName).newInstance();
            DriverManager.registerDriver (driver);
            Connection con = DriverManager.getConnection (url, username, password);
            Statement statement = con.createStatement();
            java.sql.ResultSet set = statement.executeQuery (sql);
            statement.close();
            con.close ();
        } catch(Exception e) {
            e.printStackTrace ();
            System.out.println (e.getMessage());
            System.exit(-1);
        }
    }
}
```

另一种简单的编程接口示例 2（下列内容可使用任何文本编辑器进行编辑，编辑好后保存到DB2.java文件中）：

```
/**
 @(#)DB2.java 1.2 2010.10.18

 假设 db.jdbc.jar及 DB2.java文件都在 c:\sample路径中 ,则可使用下列命令进行编译：

    C:\sample>javac -cp db.jdbc.jar DB2.java

 编译好后，则可使用下列命令执行：

  C:\sample>java -cp db.jdbc.jar;c:\sample DB2

注： ;后面的 c:\sample不能省去 ,即：
        C:\sample>java -cp db.jdbc.jar DB2
        则会出现错误：提示没有发现类 DB2
 **/
```

```
import db.jdbc.*;

public class DB2
{

   public static void main (String args [])
   {
       String url = Def.DB_PROTOCOL + "//localhost:2000/AAA";
       String username = "test";
       String password = "test";

       String sql = "SELECT * FROM SAMPLE.销售点 ";

       try {
           User user = new User (url,username,password.toCharArray ());
           DBAdapter adapter = DBAdapter.getInstance (user);
           ResultSet offices = adapter.executeQuery (sql);
       } catch(Exception e) {
           e.printStackTrace ();
           System.out.println (e.getMessage ());
           System.exit(-1);
       }
   }
}
```

12.2 类 db.jdbc.DBAdapter

在上述示例 2中，在获得了 Huayisoft DB Server 应用编程接口 adapter后，保持 adapter这个应用编程接口 ,就可随时执行 Huayisoft DB Server 语句 ,并返回需要的数据对象。

DBAdapter类提供的方法：

```
public class DBAdapter

    public static DBAdapter getInstance (User user) throws Exception
    public DBAdapter (User user, Adapter adapter) throws Exception

    public User getUser ()
```

```
public boolean isBusy ()

public ResultSet executeQuery (String query) throws Exception
public int executeUpdate (String sql) throws Exception
public Object execute (String cmd) throws Exception
public Object execute (String cmd, Object args) throws Exception

public void clearBatch () throws Exception
public void addBatch (String sql) throws Exception
public Object executeBatch () throws Exception
public boolean hasMoreResults () throws Exception
public Object getResult () throws Exception

public void close () throws Exception
protected void finalize () throws Throwable
```

12.3 Huayisoft DB Server 的运行环境

软件环境：Huayisoft DB Server 采用纯 Java 开发，跨平台，可运行于 Windows、Linux、Unix 等等操作系统。

硬件环境：Huayisoft DB Server 可运行于 CPU 为 486、内存为 16MB 以上微机。

附录 A. 样本数据库

本样本数据库共有 5 张表：
客户 表中每个客户对应一条记录。
定单 表中每个客户订单对应一条记录。
产品 表中每种产品对应一条记录。
销售点 表中每个销售地点对应一条记录。
销售员 表中每名销售员对应一条记录。

这 5 张表可以使用下列语句创建：

```
SET CATALOG AAA;

CREATE SCHEMA IF NOT EXISTS SAMPLE

CREATE TABLE IF NOT EXISTS 产品
(
    类别 CHAR (6) NOT NULL,
    代号 CHAR (5) NOT NULL,
    产品名称 VARCHAR (20) NOT NULL,
    数量 INTEGER NOT NULL,
    价格 MONEY NOT NULL,
    PRIMARY KEY ( 类别,代号)
)

CREATE TABLE IF NOT EXISTS 销售点
(
    编号 INTEGER NOT NULL PRIMARY KEY,
    城市名称 VARCHAR(32) NOT NULL,
    所属区域 VARCHAR(16) NOT NULL,
    经理 INTEGER REFERENCES 销售员(编号) ON DELETE SET NULL,
    销售定额 MONEY,
    销售额 MONEY NOT NULL
)

CREATE TABLE IF NOT EXISTS 销售员
(
```

```
    编号  INTEGER NOT NULL PRIMARY KEY,
    姓名  VARCHAR (8) NOT NULL,
     年龄  INTEGER,
     销售点  INTEGER REFERENCES  销售点(编号) ON DELETE SET NULL,
    聘用日期  DATE NOT NULL,
    经理  INTEGER    REFERENCES  销售员(编号) ON DELETE SET NULL,
    销售定额  MONEY,
    销售额  MONEY NOT NULL
)

CREATE TABLE IF NOT EXISTS  客户
(
    编号  INTEGER NOT NULL PRIMARY KEY,
    单位名称  VARCHAR(32) NOT NULL,
    销售员  INTEGER REFERENCES  销售员(编号) ON DELETE SET NULL,
    信用额度  MONEY
)

CREATE TABLE IF NOT EXISTS  定单
(
    编号  INTEGER NOT NULL PRIMARY KEY,
    日期  DATE NOT NULL,
    客户  INTEGER NOT NULL    REFERENCES  客户(编号) ON DELETE CASCADE,
    销售员  INTEGER NOT NULL    REFERENCES  销售员(编号) ON DELETE SET NULL,
    类别  CHAR (6) NOT NULL,
    代号  CHAR (5) NOT NULL,
    数量  INTEGER NOT NULL,
    金额  MONEY NOT NULL,
    CONSTRAINT  产品  FOREIGN KEY (类别,代号) REFERENCES  产品(类别,代号) ON DELETE
RESTRICT
);
```

这 5 张表的初始内容如下：

客户表：

编号	单位名称	销售员	信用额度
201	九江华易软件有限公司	109	50000
202	北京首创科技有限公司	106	65000

203	南昌安达日化有限公司	101	50000
204	合肥运捷电子有限公司	103	40000
205	北京紫华国际集团	107	35000
206	天津易大软件有限公司	106	20000
207	武汉光达科技有限公司	104	65000
208	武汉硅科电子厂	108	50000
209	合肥南方进出口有限公司	109	45000
210	北京北方进出口有限公司	103	20000
211	南昌兄弟纺织厂	110	40000
212	武汉兄弟纺织厂	102	55000
213	九江科辉五金厂	104	35000
214	南昌三路硅电子厂	101	30000
215	南昌三好商场	103	50000
216	天津长安古玩大市场	103	65000
217	天津六都文化有限公司	102	25000
218	深圳华南电子集团	108	60000
219	成都民生系统工程集团	105	20000
220	南昌昌盛商务有限公司	109	25000
221	新疆华西葡萄加工集团	106	45000

订单表：

编号	日期	客户	销售员	类别	代号	数量	金额
8161	2009.12.17	213	104	电脑	A02	7	31500
8212	2010.01.11	201	101	手机	B03	35	3745
8189	2010.01.03	207	104	化妆品	F02	6	1458
8251	2010.02.10	218	108	衣服	E01	4	1420
8168	2009.10.12	202	106	手机	B04	34	3978
8236	2010.01.30	205	107	手机	B06	9	22500
8245	2010.02.02	208	106	电脑	A03	10	45000
8163	2009.12.17	203	107	手机	B04	28	3276
8213	2010.01.14	218	108	茶叶	C02	1	652
8258	2010.02.23	212	102	化妆品	F01	10	1480
8197	2010.01.08	211	110	茶叶	C02	1	652
8183	2009.12.27	203	101	手机	B04	6	702
8224	2010.01.20	210	108	衣服	E01	20	7100
8262	2010.02.24	211	110	化妆品	F02	10	2430
8179	2009.10.12	210	103	手机	B01	6	15000
8227	2010.01.22	203	101	手机	B06	54	4104

8207	2010.01.08	208	108	玉器	D04	3	2925
8269	2010.03.02	220	110	玉器	D02	22	31350
8234	2010.01.29	205	107	电脑	A01	8	632
8192	2009.11.04	218	108	手机	B06	10	760
8175	2009.10.12	201	109	电脑	A04	6	2100
8255	2010.02.15	212	106	手机	B07	6	150
8248	2010.02.10	215	103	玉器	D01	2	3750
8193	2009.01.04	216	103	电脑	A01	24	1896
8265	2010.02.27	216	103	衣服	E01	6	2130
8203	2010.01.25	212	102	玉器	D01	3	5625
8249	2010.02.10	218	108	衣服	E01	2	776
8187	2009.12.31	203	101	手机	B02	11	27500
8257	2010.02.18	201	109	手机	B07	24	600
8242	2010.02.02	219	106	电脑	A03	5	22500

销售员表：

编号	姓名	年龄	销售点	聘用日期	经理	销售定额	销售额
101	柯长富	37	4	2008.02.12	105	350000	367911
102	王美丽	31	2	2009.10.12	104	300000	392725
103	杨刚	48	5	2006.12.10	108	350000	474050
104	张民山	52	2	2008.07.14	NULL	275000	299912
105	罗明贵	33	3	2007.05.19	104	200000	142594
106	刘应东	45	3	2006.10.20	105	300000	305673
107	夏大宝	41	NULL	2010.01.13	106	NULL	75985
108	李立发	62	5	2009.10.12	104	350000	361865
109	吴小虎	29	3	2007.03.01	105	275000	286775
110	蔡晓	49	1	2008.11.14	108	300000	186042

销售点表：

编号	城市名称	所属区域	经理	销售定额	销售额
1	北京	华北	108	300000	186042
2	南昌	华中	104	575000	692637
3	合肥	华中	105	800000	735042
4	武汉	华中	101	350000	367911
5	天津	华北	108	725000	835915

产品表：

类别	代号	产品名称	数量	价格

电脑	A01	键盘	210	79
电脑	A02	笔记本电脑	12	4500
电脑	A03	笔记本电脑	12	4500
电脑	A04	硬盘	14	350
手机	B01	手机 B	28	2500
手机	B02	手机 A	25	2750
手机	B03	冲电器 B	207	107
手机	B04	冲电器 A	139	117
手机	B05	冲电器 B	277	55
手机	B06	冲电器 A	167	76
手机	B07	手机套	37	25
茶叶	C01	绿茶	0	180
茶叶	C02	龙井	3	652
茶叶	C03	红茶	78	225
玉器	D01	白玉龙摆件	9	1875
玉器	D02	白玉小马	5	1425
玉器	D03	白玉小环	24	250
玉器	D04	白玉小猴	28	975
玉器	D05	白玉小扣	223	54
玉器	D06	白玉小猪	32	475
衣服	E01	男上衣	38	355
衣服	E02	裙子	203	134
衣服	E03	男裤	37	177
化妆品	F01	化妆品 B	115	148
化妆品	F02	化妆品 A	15	243

附录 E. Huayisoft DB Server 信息模式

Huayisoft DB Server 通过下列视图来达到自我描述的：

E.1 CATALOGS 视图

每个目录在 CATALOG视图中有一条记录：

字段名	数据类型	说明
CATALOG_NAME	VARCHAR (*32*)	目录名
CATALOG_PATH	VARCHAR *(64*)	目录名对应的路径名
CLASS_ID	INTEGER	目录的默认执行类
ATTRIBUT	BINARY(*4*)	目录的属性字
USER_COUNT	INTEGER	目录的最大用户数
STARTDATE	DATE	目录的起始使用日期
ENDDATE	DATE	目录的结束使用日期
OWNER	INTEGER	目录的拥有者
DEFINER	INTEGER	目录的定义者

E.1 CLASSES 视图

每个 CALSSES类定义在视图中有一条记录：

字段名	数据类型	说明
CLASS_ID	INTEGER	类编号
CLASS_NAME	VARCHAR (32)	类名称
DEFINER	INTEGER	类的定义者

E.1 USERS 视图

每个用户在 USERS视图中有一条记录：

字段名	数据类型	说明
USER_ID	INTEGER	用户编号
NAME	VARCHAR *(32*)	用户的真实名
USERNAME	VARCHAR (12)	用户名
PASSWORD	BINARY(20)	用户密码
SCHEMA_ID	INTEGER	用户的默认模式
CLASS_ID	INTEGER	用户的默认执行类

SERVER	BINARY(4)	用户的相关服务设置
MENUS	VARCHAR(12)	用户的菜单名
STARTDATE	DATE	用户的起始使用日期
ENDDATE	DATE	用户的结束使用日期
DEFINER	INTEGER	用户的定义者

E.1 SCHEMATA 视图

每个模式在 SCHEMATA视图中有一条记录：

字段名	数据类型	说明
CATALOG_NAME	VARCHAR (*32*)	模式的目录名
SCHEMA_NAME	VARCHAR (*32*)	模式名
SCHEMA_OWNER	VARCHAR (*18*)	模式的拥有者
SCHAME_DEFINER	VARCHAR (*18*)	模式的定义者

E.2 TABLES 视图

每张表在 TABLES视图中有一条记录：

字段名	数据类型	说明
TABLE_CATALOG	VARCHAR(*32*)	表的目录名
TABLE_SCHEMA	VARCHAR(*32*)	表的模式名
TABLE_NAME	VARCHAR(*32*)	表名
TABLE_TYPE	VARCHAR(*1*)	表类型

E.4 VIEWS视图

每张视图在 VIEWS视图中有一条记录：

字段名	数据类型	说明
TABLE_CATALOG	VARCHAR(*32*)	视图的目录名
TABLE_SCHEMA	VARCHAR(*32*)	视图的模式名
TABLE_NAME	VARCHAR(*32*)	视图名
VIEW_DEFINITION	VARCHAR(*1024*)	视图的 SQL SELECT语句文本
CHECK_OPTION	VARCHAR(*8*)	视图的检验选项 (CASCADED/LOCAL/NONE)
IS_UPDATABLE	BOOLEAN	视图是否可以更新 (YES/NO)

E.5 VIEW_TABLE_USAGE视图

视图上的每张表在 VIEW_TABLE_USAGE视图中有一条记录：

字段名	数据类型	说明
VIEW_CATALOG	VARCHAR(*32*)	视图的目录名
VIEW_SCHEMA	VARCHAR(*32*)	视图的模式名
VIEW_NAME	VARCHAR(*32*)	视图名
TABLE_CATALOG	VARCHAR(*32*)	视图依赖的表的目录名
TABLE_SCHEMA	VARCHAR(*32*)	视图依赖的表的模式名
TABLE_NAME	VARCHAR(*32*)	视图依赖的表名

E.7 TABLE_CONSTRAINTS视图

每条表约束在 TABLE_CONSTRAINTS视图中有一条记录：

字段名	数据类型	说明
CONSTRAINT_CATALOG	VARCHAR(*32*)	约束的目录名
CONSTRAINT_SCHEMA	VARCHAR(*32*)	约束的模式名
CONSTRAINT_NAME	VARCHAR(*32*)	约束名
TABLE_CATALOG	VARCHAR(*32*)	表的目录名
TABEL_SCHEMA	VARCHAR(*32*)	表的模式名
TABEL_NAME	VARCHAR(*32*)	该约束的表名
CONSTRAINT_TYPE	VARCHAR(*12*)	约束类型 (UNIQUE/FRIMARY KEY/FOREIGN KEY/CHECK)
IS_DEFERRABLE	BOOLEAN	约束是否是可延迟的 (YES/NO)
INITIALLY_DEFERRED	BOOLEAN	约束是否延迟 (YES/NO)

E.8 REFEENTIAL_CONSTRAINTS视图

每条引用约束在 REFERENTIAL_CONSTRAINSTS视图中有一条记录：

字段名	数据类型	说明
CONSTRAINT_CATALOG	VARCHAR(*32*)	约束的目录名
CONSTRAINT_SCHEMA	VARCHAR(*32*)	约束的模式名
CONSTRAINT_NAME	VARCHAR(*32*)	约束名
UNIQUE_CONSTRAINT_CATALOG	VARCHAR(*32*)	被引用约束的目录名
UNIQUE_CONSTRAINT_SCHEMA	VARCHAR(*32*)	被引用约束的模式名
UINQUE_CONSTRAINT_NAME	VARCHAR(*32*)	被引用约束名
MATCH_OPTION	VARCHAR(*8*)	部分外键字匹配的类型 (NONE/PARTIAL/FULL)

UNDATE_RULE	VARCHAR(*12*)	更新引用约束的规则 (CASCADE/SET NULL/SET DEFAULT/NO ACTION)
DELETE_RULE	VARCHAR(*12*)	删除引用约束的规则 (CASCADE/SET NULL/SET DEFAULT/NO ACTION)

E.9 CHECK CONSTRAINTS视图

每条检验约束 (检验约束、域检验约束或声明定义)在 CHECK_CONSTRAINTS视图中有一条记录：

字段名	数据类型	说明
CONSTRAINT_CATALOG	VARCHAR(*32*)	约束的目录名
CONSTRAINT_SCHEMA	VARCHAR(*32*)	约束的模式名
CONSTRAINT_NAME	VARCHAR(*32*)	约束名
CHECK_CLAUSE	VARCHAR(*256*)	约束的 SQL搜索条件的文本定义

E.10 KEY_COLUMN_USAGE视图

关键字的各个字段在 KEY_COLUMN_USAGE视图中有一条记录：

字段名	数据类型	说明
CONSTRAINT_CATALOG	VARCHAR(*32*)	关键字约束的目录名
CONSTRAINT_SCHEMA	VARCHAR(*32*)	关键字约束的模式名
CONSTRAINT_NAME	VARCHAR(*32*)	关键字约束名
TABLE_CATALOG	VARCHAR(*32*)	关键字所在表的目录名
TABLE_SCHEMA	VARCHAR(*32*)	关键字所在表的方案名
TABLE_NAME	VARCHAR(*32*)	关键字字段的表名
COLUMN_NAME	VARCHAR(*128*)	字段名
ORDINAL_POSTION	INTEGER > 0	关键字中的字段位置

E.11 ASSERTIONS视图

各条声明在 ASSERTIONS视图中有一条记录：

字段名	数据类型	说明
CONSTRAINT_CATALOG	VARCHAR(*32*)	声明的目录名
CONSTRAINT_SCHEMA	VARCHAR(*32*)	声明的模式名
CONSTRAINT_NAME	VARCHAR(*32*)	声明名
IS_DEFERRABLE	BOOLEAN	声明是否是可延迟的 (YES/NO)

INITIALLY_DEFERRED	BOOLEAN	声明是否延迟 (YES/NO)

E.14 TABLE_PRIVILEGES视图

授予用户的表上的各种权限在 TABLE_PRIVILEGES视图中有一条记录：

字段名	数据类型	说明
GRANTOR	VARCHAR(*len*)	授权的用户
GRANTEE	VARCHAR(*len*)	被授权的用户
TABLE_CATALOG	VARCHAR(*32*)	表的目录名
TABLE_SCHEMA	VARCHAR(*32*)	表的模式名
TABLE_NAME	VARCHAR(*32*)	表名
PRIVILEGE_TYPE	VARCHAR(*12*)	权限的类型 (SELECT/INSERT/DELETE/UPDATE/REFERENCES)
IS_GRANTABLE	BOOLEAN	是否用 GRANT选项授予权限 (YES/NO)

E.17 DOMAINS视图

各个域在 DIMAINS视图中有一条记录：

字段名	数据类型	说明
DOMAINS_CATALOG	VARCHAR(*32*)	域的目录名
DOMAINS_SCHEMA	VARCHAR(*32*)	域的模式名
DOMAINS_NAME	VARCHAR(*18*)	域名
DATA_TYPE	VARCHAR(*32*)	域的基本的 SQL2数据类型 (用文本表示)
CHARACTER_MAXIMUM_LENGTH	INTERGER > 0	可变长度字符类型按字符计算的最大长度
CHARACTER_OCTET_LENGTH	INTERGER > 0	可变长度字符类型按字节计算的最大长度
COLLATION_CATALOG	VARCHAR(*32*)	域的校正的目录名
COLLATION_SCHEMA	VARCHAR(*32*)	域的校正的模式名
COLLATION_NAME	VARCHAR(*32*)	域的校正名
CHARACTER_SET_CATALOG	VARCHAR(*32*)	域的字符集的目录名
CHARACTER_SET_SCHEMA	VARCHAR(*32*)	域的字符集的模式名
CHARACTER_SET_NAME	VARCHAR(*32*)	域的字符集名
NUMERIC_PRECISION	INTERGER > 0	基于数学数据类型的域的精度
NUMERIC_PRECISION_RADIX	INTERGER > 0	精度的基数
NUMERIC_SCALE	INTERGER > 0	基于数字数据类型的域的大小
DATATIME_PRECISION	INTERGER > 0	基于 DateTime类型的域的精度
DOMAIN_DEFAULT	VARCHAR(*32*)	该域默认值的文本表示

E.18 DOMAINS_CONSTRAINTS视图

域上的各个域约束在 DOMAINS_CONSTRAINTS视图中有一条记录：

字段名	数据类型	说明
CONSTRAINT_CATALOG	VARCHAR(*32*)	约束的目录名
CONSTRAINT_SCHEMA	VARCHAR(*32*)	约束的模式名
CONSTRAINT_NAME	VARCHAR(*32*)	约束名
DOMAIN_CATALOG	VARCHAR(*32*)	域的目录名
DOMAIN_SCHEMA	VARCHAR(*32*)	域的模式名
DOMAIN_NAME	VARCHAR(*18*)	域名
IS_DEFERRABLE	BOOLEAN	
INITIALLY_DEFERRED	BOOLEAN	

E.19 DOMAINS_COLUMN_USAGE视图

使用域的各个字段在 DOMAINS_COLUMN_USAGE视图中有一条记录：

字段名	数据类型	说明
DOMAIN_CATALOG	VARCHAR(*32*)	域的目录名
DOMAIN_SCHEMA	VARCHAR(*32*)	域的模式名
DOMAIN_NAME	VARCHAR(*18*)	域名
TABLE_CATALOG	VARCHAR(*32*)	字段定义的目录名
TABLE_SCHEMA	VARCHAR(*32*)	字段定义的模式名
TABLE_NAME	VARCHAR(*32*)	字段的表名
COLUMN_NAME	VARCHAR(*128*)	字段名