HXTT's HXTT Access Packages

Welcome to the HXTT Access pages

You should read carefully License, Introduction, and Components first. If you have JDBC programming experience and SQL92 knowledge, you can start easily your project after you know com.hxtt.sql.access.AccessDriver (the suitable JDBC driver class name) and jdbc:access:///[DatabasePath] (the correct embedded JDBC url) from here. You will get up to date information relating to the HXTT Access, and look at current documentation from here. For questions and general support, you should submit your request at HXTT's technical support site.

Chapter 1. Quick Start

Index:

What Is the HXTT Access?

HXTT Access provides a type 4 JDBC driver for Microsoft Access (MS Access) version from 95, 97, 2000, XP, 2002, to 2003. It supports JDBC1.2, JDBC2.0, and JDBC3.0. It supports Personal Java, JDK1.0.X, JDK1.1.X, JDK1.2.X, JDK1.3.X, JDK1.4.X and JDK1.5.X. It supports all transactions level of READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. It supports JBuilder's Database Pilot, Oracle's JVM, JDeveloper 10G, Dreamweaver UltraDev, Dreamweaver ColdFusion, ObJectRelationalBridge, DBVisualizer, iSQL-Viewer, AquaDataStudio, Sunopsis, MySQL Migration Toolkit, Tomcat, vqServer, Hibernate, SQuirreL SQL Client, and DbEdit Database Utilites for Eclipse Platform. It supports XOPEN SQLState, RMI, Jini, JNDI, and serialization. It supports { UNION | INTERSECT | EXCEPT | MINUS } [ALL] query , INNER JOIN, FULL JOIN, LEFT JOIN, RIGHT JOIN, NATURAL JOIN, CROSS JOIN, self join, multiple-row VALUES table, PIVOT table, UNPIVOT table, and subquery which includes single-row subquery, multirow subquery, multiple-column subquery, inline views, and correlated subquery. The current version of the HXTT Access packages are available here:

Follow Me

First, you need to download JDK 1.3.X, 1.4.X, or 1.5.X from www.javasoft.com if you use Access JDBC 3.0 package(Access_JDBC30.jar). You can download JDK1.2.X too if you use Access JDBC 2.0 package(Access_JDBC20.jar). You can download JDK1.1.X too if you use Access JDBC 1.2 package(Access_JDBC12.jar).

Secondly, please add Access_JDBC30.jar, Access_JDBC20.jar or Access_JDBC12.jar to your Java class path, for instance, "SET CLASSPATH=c:\javalib\Access_JDBC20.jar;%classpath%". You can also use "java -classpath c:\javalib\Access_JDBC20.jar yourAccessclass" to run your class. More information about classpath, please read the "Setting the Classpath" topic in file:///yourdriver|/jdk1.2/docs/tooldocs/tools.html . You can use "java -classpath c:\javalib\Access_JDBC20.jar yourAccessclass" too.

Thirdly, you can use 'Class.forName("com.hxtt.sql.access.AccessDriver").newInstance();' or Class.forName("com.hxtt.sql.access.AccessDriver");' to load this driver.

Fourth, if you have used other JDBC driver, you only need to know the correct URL format for DriverManager.getConnection(url,"",""); You can find the Access URL format below. If you were a Java novice, please read also other Java examples code in Access_JDBC30demo.zip, Access_JDBC20demo.zip or Access_JDBC12demo.zip.
Access URL format:

```
        Embedded:
                jdbc:access:[//]/[DatabasePath][?prop1=value1[;prop2=value2]] (You
can omit that "//" characters sometimes)
                        For example:
                                "jdbc:access:/."
                                "jdbc:access:/c:/data" for Windows driver
                                "jdbc:access:///c:/data" for Windows driver
                                "jdbc:access:////usr/data" for unix or linux
                                "jdbc:access://///192.168.10.2/sharedir" for UNC path
                                "jdbc:access:/./data"
                                "jdbc:access:/./data/mydata.mdb"
        Remote Access (client/server mode):
                jdbc:access://host:port/[DatabasePath]
                        For example: "jdbc:access://domain.com:3099/c:/data" if one
AccessServer is run on the 3099 port of domain.com

"jdbc:access://domain.com:3099/c:/data/mydata.mdb"
        Compressed Database:(.ZIP, .JAR, .GZ, .TAR, .BZ2, .TGZ, .TAR.GZ, .TAR.BZ2)
                jdbc url format is the same as embedded url and remote url.
                        For example:
                                "jdbc:access:/c:/test/testaccess.zip
        Memory-only Database:
```

```
                jdbc:access:/_memory_/
        URL Database:(http protocol, https protocol, ftp protocol)
                jdbc:access:http://httpURL
                jdbc:access:https://httpsURL
                jdbc:access:ftp://ftpURL
                        For example:
                                "jdbc:access:http://www.hxtt.com/test"
        SAMBA Database:(smb protocol)

jdbc:access:smb://[[[domain;]username[:password]@]server[:port]/[[share/[dir/]file]]][?[param=value]]
                        For example:

"jdbc:access:smb://test1:123@100.100.13.94/accessfiles".zone"
        Free JDBC url:(Warning: only use it for special project)
                jdbc:access:/" or "jdbc:access:///". Then you can use some full UNC
path names in SQL to visit anywhere where your Java VM has right to access.
                        For instance:
                                select * from \\amd2500\e$\accessfiles\test;
                                elect * from "\\amd2500\d$\accessiles".test;
                                select * from ".".test;

         HXTT Access supports seamlessly data mining on memory-only table, physical
table, url table, compressed table, SAMBA table in a sql. More details
         is in Advanced Programming chapter.
```

Last, Access driver is a standard JDBC driver so that you will find most of valuable information at file:///yourdrive|/jdk1.2/docs/api/java/sql/package-frame.html .

Access supports SQL-92. It supports { UNION | INTERSECT | EXCEPT | MINUS } [ALL] query , INNER JOIN, FULL JOIN, LEFT JOIN, RIGHT JOIN, NATURAL JOIN, CROSS JOIN, self join, and subquery which includes single-row subquery, multirow subquery, multiple-column subquery, inline views, and correlated subquery. The major syntax is listed at here.

Access driver will use index to speed up the query which contains some indexed expressions. Access supports utilizing index file for LIKE, BETWEEN, IN, DISTINCT, ORDER, and some OR operations.

Chapter 6. SQL Syntax

Index:

Use ";" to separate multi sql statements. For instance, "insert into test (int1) values(1);insert into test (int1) values(2);". "reserved word", [reserved word] or {v 'reserved word'} is used to quote a column with reserved word name in SQL statement, for instance, 'select {v 'RIGHT'},'other' from states where {v

'RIGHT'}=32. The HXTT Access supports using DATE, TIME, TIMESTAMP, GROUP, ORDER, KEY, DESC, SEQUENCE, INCREMENT, MINVALUE, MAXVALUE, CACHE, CHECK, CYCLE, OTHER, SET, INT, UNIQUE, LEVEL, RELEASE, INDEX, TOP, PACK, CALL, CONNECT, and UPDATE directly in SQL, although they're reserved words too.

SELECT [ALL | DISTINCT [ON (expression [, ...])]] | DISTINCTROW [TOP n [PERCENT]] select_list [INTO variable [, ...]] FROM table_reference_list [WHERE condition_expression] [group_by_clause] [HAVING condition_expression] [union_clause] [order_by_clause] [FOR UPDATE]

select_list: { expression [[AS] columnAlias] | table.* | * } [,...]

table_reference_list: {table_reference | table_join} [,...]

table_reference: { { table_name | subquery | (table_join) | (VALUES expression[, ...]) AS tableName(columnName[,...])} [[AS] tableAlias] } [pivot_clause] [unpivot_clause]

table_name: { [catalog.]tableName} | {UNC path}

table_join: table_reference join_clause [join_clause,...]

join_clause: [NATURAL] { INNER | { [LEFT | RIGHT | FULL] [OUTER] } } JOIN table_reference [ON condition_expression | USING(column1,column2,...)]

condition_expression: an expression which should return a boolean value.

pivot_clause: PIVOT (aggregate_function(value_column) FOR pivot_column IN (column_list)) [AS] tableAlias

unpivot_clause: UNPIVOT (value_column FOR pivot_column IN (column_list)) [AS] tableAlias

group_by_clause: GROUP BY expression [,...]

union_clause: { UNION | INTERSECT | EXCEPT | MINUS } [ALL] select_statement [union_clause ...]

order_by_clause: ORDER BY expression [ASC|DESC] [,...]

rowid, is a virtual column as primary key.

DISTINCT specifies that duplicate rows are discarded. A duplicate row is when each corresponding select_list column has the same value. DISTINCT has no effect on constant, and _rowid_. For instance,

"select distinct 'First Name',name,age from users". 'First Name' will be ignored since it's a constant.

expression: a complicated expression which can include parentheses, logical operator(NOT, AND, OR), positives/minus sign(+, -), arithmetical operator(+,-,*,/,%), string operator(|| (left string concat right string), +(left string concat right string), -(trim left string then concat rightstring), $(check whether left string is contained in right string), condition operator(>,>=,=,==,<=,<,!=,<>), [NOT] LIKE value {escape 'escape_character'},[NOT] ILIKE value {escape 'escape_character'}, IS [NOT] NULL, BETWEEN ... AND ..., [NOT] IN, [NOT] EXISTS, [ALL|ANY|SOME] (subquery), [NOT] CASE WHEN expr THEN result [WHEN expr THEN result ...] [ELSE expr] END, CASE expr WHEN compare_expr THEN result [WHEN compare_expr THEN result ...] [ELSE result] END, SQL Escape Syntax({d 'yyyy-mm-dd'}, {t 'hh:mm:ss'}, {ts 'yyyy-mm-dd hh:mm:ss.f...'},{v 'reserved_word'}, {fn functionExpression}, {escape 'oneEcapeCharacter'}, {"varbinary" 'string'}), function(more than 200), aggregate function(MAX, MIN, AVG, COUNT, SUM, STD, STDDEV), constant(null, true, false, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, date, time, timestamp, number, string), column, parameter(?), subquery(single-row subquery, multirow subquery, multiple-column subquery, inline views, correlated subquery) and so on.

"SELECT select_list" can be used to get some calculated values through an one-row ResultSet. Column can be used in all sql except for "SELECT select_list". Parameter(?) can only be used in PreparedStatement.

For instance:

```
select val('123.222')
select CONVERT('123',SQL_INTEGER) as a,TTOC({d '1999-10-10'},1) as b,
IFNULL(1,33) as c, 123 in(456,123,789,'abc') as d, EXTRACT(DECADE FROM '2001-
02-16 20:38:40'), '88'+IIF(3<6,'1','0')
select encode('adsdfsdf');
select decode(encode('adsdfsdf'))+";
SELECT top 8 percent * FROM data.sz9010;
SELECT * FROM "data"."testtable"; SELECT * FROM "datadir\data"."testtable"; select
distinct top 10 * from test where not deleted() order by int1,char1 desc;
select int1,float1 from test where int1>0 group by int1,double1;
select distinct on (int1) int1,double1 from test;
select sum(int1),max(dec1),min(double1) from test;
SELECT SUM(apmast.fnamount),
SUM(glcshi.fnadjamt),SUM(glcshi.fncashamt),SUM(glcshi.fndiscount) FROM apmast,
glcshi WHERE apmast.fcinvoice +apmast.fvendno = glcshi.fcinvoice + glcshi.fcnameid
AND apmast.fduedate between {d '1999-01-01'} AND {d '1999-11-30'} AND
apmast.finvdate <= {d '1999-11-30'};
SELECT cellID, columnID, reference, function, parameter FROM repLayout WHERE
reportID = '1' AND rowID = 0 ORDER BY columnID;
select distinct int1,double1 from test group by int1,double1,float1;
```

```
select distinct * from test where int1>0
select distinct int1,count(*),sum(int1) from brain.user group by int1
SELECT date1,time1,int1 FROM test where
TIMESTAMPdIFF(SQL_TSI_YEAR,time1,{ts '3999-03-24 00:59:23.22222'})<-2000 and
date1>{d '1900-01-01'} and date1>{d '1960-01-01'} and date1<{d '2000-01-02'}+20;
select int1 as a,c+23 as b,a+b as c from test where a=1;
SELECT INT1,FLOAT1,A.* FROM TEST A WHERE {fn abs(-TEST.INT1)}>0 or
a.float1<0 order by int1 asc,currency1,double1*5+int1 desc;
select int1,count(*),sum(int1+count(*)),sum(int1)+int1 from test group by int1 having
int1>10;
SELECT SCHOOLNUM, STULINK,CHGNUMBER, {v 'ABSEN$0101'}, {v
'ABSEN$0102'}, USERSTAMP, DATESTAMP, {v 'TIMESTAMP'},SEQUENCE FROM
AATD2019 where {v 'ABSEN$0101'}='1234' ORDER BY SCHOOLNUM, STULINK,
SEQUENCE;
select char1,char1 like 'Z%',char1 in('ZZAA','Z'),char1 between 'A' and 'ZZZ',char1
in('ZZAA','Z') or char1 between 'A' and 'Z',* from test where char1='Z';
select int1 from test where int1=(select distinct top 1 int1 from test where int1>0);
select int1 from test where int1 in(select int1 from test where not deleted());
select recno(),int1 from test where (recno(),int1) in(select top 2 recno(),int1 from test
where int1>0);
select subquery.int1,recno('subquery') from (select top 2 recno(),int1 from test where
int1>0) as subquery;
select subquery.int1,recno('test'),test.int1,recno('subquery') from (select top 2 recno(),int1
from test where int1>0) as subquery, test where test.int1=subquery.int1;
ELECT INT1 FROM test as a WHERE EXISTS(SELECT 1 FROM test WHERE int1 >0);
SELECT INT1 FROM test as a WHERE int1>=all(SELECT int1 FROM test);
SELECT INT1 FROM test as a WHERE int1>=any(SELECT int1 FROM test);
SELECT INT1 FROM test as a WHERE int1>=some(SELECT int1 FROM test);
select int1,recno() from test where (int1,recno())>(3,5);
select int1,recno() from test where (recno(),int1)=(6,222);
SELECT * FROM (SELECT * FROM test WHERE int1 = 222 ) as a WHERE
EXISTS(SELECT 1 FROM test WHERE int1 >0);
select recno('a'),recno('b'),a.int1,a.char1,b.int1,b.char1 from test a, test as b where
recno('a')=recno('b');
select a.int1,a.char1,b.int1,b.char1 from test a inner join test as b on a.int1=b.int1;
select a.int1,a.char1,b.int1,b.char1 from test a NATURAL inner join test as b on
a.int1=b.int1;
select recno('a'),recno('b'),a.int1,a.char1,b.int1,b.char1 from test a left join test as b on
a.int1=b.int1;
select a.int1,a.char1,b.int1,b.char1 from test a right join test as b on a.int1=b.int1;
select a.int1,a.char1,b.int1,b.char1 from test a full join test as b on a.int1=b.int1;
select recno('a'),recno('b'),a.int1,a.char1,b.int1,b.char1 from test a full join test as b on
a.int1==b.int1 and recno('a')!=recno('b');
```

```
SELECT * FROM test a LEFT JOIN (test b JOIN test c ON (b.int1 = c.int1)) as d ON
(a.int1 = d.int1);
SELECT * FROM test a,test b,test c WHERE a.int1 = b.int1 AND b.int1 = c.int1;
SELECT * FROM test a NATURAL CROSS JOIN test b CROSS JOIN test c WHERE
a.int1 = b.int1 AND b.int1 = c.int1;
SELECT * FROM test a LEFT JOIN (test b JOIN test c ON (b.int1 = c.int1)) on
recno('a')=recno(2);
SELECT int1 FROM test where int1>0 UNION ALL select int1 from test where
int1>3000 order by int1 desc
SELECT int1,* FROM test where int1>0 UNION select int1,* from test where int1>3000
order by int1
ELECT int1,* FROM test where int1>0 INTERSECT all select int1,* from test where
int1>3000 order by int1;
SELECT int1,* FROM test where int1>0 EXCEPT select int1,* from test where int1>3000
order by int1 descl
SELECT int1,* FROM test where int1>0 MINUS select int1,* from test where int1>3000
order by int1,double1 desc;
select double1,sum(double1),int1 from test where int1>0 group by int1 having
sum(double1)>0 and double1>0;
select distinct 1,a.int1,sum(a.int1) from test as a,test as b group by a.int1,B.int1
select a.int1,a.char1,b.int1,b.char1 from test a NATURAL inner join test as b
```

INSERT INTO table_name [(column_identifier [,...])] { VALUES (expression [, ...]) | VALUES expression [, ...] | VALUES (expression [, ...]),... | SELECT query | ? }

column_identifier = columnName | "reserved_word" | {v 'reserved_word'}

Adds one or more new rows of data into a table. SQL does't permit that table1 is the same table as table2 when INSERT INTO table1 select * from table2, but the HXTT Access support such an unadvisable operation, for example, INSERT INTO test (INT1,DATE1) select distinct int1,date1 from test.

For instance:

```
INSERT INTO test (INT1,dec1,time1) VALUES(-1999,-222.33333,{ts '1333-11-30
22:22:22.999999999'});
INSERT INTO test ("INT1","DATE1") VALUES(1999.0111,{d '1996-10-21'});
INSERT INTO test ("INT1","DATE1") VALUES(1999.0111,{d '1996-10-21'}),(333,{d
'2006-10-21'});
INSERT INTO test ("INT1") VALUES 1999.0111,333;
insert into ecode values('Maciej', 'Kowalski');
insert into test values (reccount()+1,'abc',date(),{ts '2003-12-18 19:42:17.88'});
INSERT INTO AATD2019 ({v 'ABSEN$0101'}) values('1234');
```

```
insert into test select * from test order by int1 asc;
insert into test select * from test order by int1 asc;
```

UPDATE table_name SET column_identifier = expression [,...] [WHERE condition_expression]

For instance:

```
update test set int1=null where SequenceID=26;
update test set INT1=323232,DEC1=-DEC1 where FLOAT1=3.00 and INT1=222 and
DEC1=3.00 and DOUBLE1=34.0 and TIME1 is NULL and CHAR1='ZZAA' and
CURRENCY1=0 and BOOLEAN1 is NULL
update AATD2019 set {v 'ABSEN$0101'}='1234' where SequenceID=1;
update test set int1=3333555 where exists(SELECT 1 FROM test WHERE int1 = 222 )
and SequenceID=3;
```

DELETE FROM table_name [WHERE condition_expression]

Removes rows in a table according to condition_expression.

For instance:

```
delete from test where SequenceID=4;
```

CREATE CATALOG [IF NOT EXISTS] catalogName

Create a subdirectory to contain database files.

For instance:

```
create catalog if not exists data222;
```

CREATE DATABASE [IF NOT EXISTS] databaseName

Create a new MS ACCESS MDB database, you can assign which JET engine(JET3 or JET4) database would be created by assign the versionNumber property when build connection, it wil create JET4 engine database by default; the URL format must be assigned as a directory, for example, "jdbc:access:////usr/data"

```
create database if not exists testmdb;
```

CREATE TABLE [IF NOT EXISTS] table_name [(column_identifier data_type [constraint] [,...])

[,AUTO_INCREMENT] [, PRIMARY KEY (column [,...])]] [[AS] SELECT query | ?]

data_type: CHAR(n) | CHARACTER(n) | VARCHAR(n) | BINARY (n) | VARBINARY (n) | NUMERIC(n1[,n2]) | DEC[IMAL](n1[,n2]) | INT[EGER] | SMALLINT | FLOAT [(n)] | REAL | DOUBLE | BIT | BOOLEAN | DATE [(dateFormat)] | TIME [(dateFormat)] | TIMESTAMP [(dateFormat)] | LONGVARCHAR [(n)] | LONGVARBINARY [(n)] | JAVA_OBJECT [(n)] | CLOB | BLOB| OTHER(type_name [,n])

n, n1,n2: positive integer, n2 can be 0

constraint: [NULL| NOT NULL] [UNIQUE] [DEFAULT expression] [PRIMARY KEY]

The HXTT Access will ignore (dateFormat) for Access's compatibility. Column can store null values, so that constraint NOT NULL is ignored. Although CREATE INDEX can create UNIQUE and PRIMARY KEY for existent table, the preferable way is using UNIQUE and PRIMARY KEY in CREATE TABLE.

SQL Data Types for Create Table

SQL Type	Access Data Type	Size in table	SQL Syntax	Java Type
CHARACTER	Unavailing	1~2	CHAR[ACTER]	char
CHAR	Text field of max width n	1~255	CHAR[ACTER](n)	String
VARCHAR	Text field of max width n	1~255	VARCHAR (n)	String
LONGVARCHAR	Memo BLOB		LONGVARCHAR	String, char[], java.sql.CLOB
NUMERIC	NUMERIC	1~255	NUMERIC [(n[,d])]	java.math.BigDecimal
DECIMAL	DECIMAL	1~255	DEC[IMAL] [(n[,d])]	java.math.BigDecimal

BIT	Logical	1	BIT	boolean
TINYINT	Unavailing, map TINYINT into SMALLINT		TINYINT	byte
SMALLINT	Short integer	2	SMALLINT	short
INTEGER	Long integer	4	INT[EGER]	int
BIGINT	Unavailing, map BIGINT into NUMERIC(20,0)		BIGINT	long
REAL	float	4	REAL	float
FLOAT	Unavailing, map FLOAT into DOUBLE		FLOAT	double
DOUBLE	double	8	DOUBLE	double
BINARY	Unavailing, map BINARY into VARBINARY		BINARY (n)	byte[]
VARBINARY	VARBINARY		VARBINARY (n)	byte[]
LONGVARBINARY	Unavailing, map LONGVARBINARY into OLE		LONGVARBINARY (n)	byte[], java.sql.BLOB
DATE	Unavailing, map DATE into TIMESTAMP		DATE	java.sql.Date

TIME	Unavailing, map TIME into TIMESTAMP		TIME	java.sql.Time
TIMESTAMP	Timestamp	8	TIMESTAMP	java.sql.Timestamp
BOOLEAN	Logical	1	BOOLEAN	boolean
BLOB	Unavailing, map BLOB into OLE		BLOB	byte[], java.sql.BLOB, Object
CLOB	Unavailing, map CLOB into LONGVARCHAR		CLOB	String, char[], java.sql.CLOB
OTHER	Currency OLE	8	OTHER (Currency) OTHER (OLE)	java.math.BigDecimal byte[]
JAVA_OBJECT	Unavailing, map JAVA_OBJECT into OLE		JAVA_OBJECT	byte[] or Object

For instance:

```
create table if not exists test(TAlpha varchar(25),TMoney currency,TShort
smallint,TLongInt int,TBCD float(20,4),TCalc double,TDate date,TTime time, TMemo
longvarchar, TOle OTHER (OLE), TLogical boolean );
```

DROP TABLE [IF EXISTS] table_name

Removes a table, and its indexes from the database. IF that table doesn't exist without using IF EXIST, an SQLException will be thrown.

For instance:

```
drop table if exists states;
```

ALTER TABLE table_name alter_specification [,...]

alter_specification: {{ADD|MODIFY} column_identifier data_type [constraint]}| DROP column_identifier | RENAME column_identifier 1 TO column_identifier 2 | RENAME TO table_name2

When some alter operations are in one ALTER sql, the HXTT Access will complete all RENAME column operations first, then do all ADD, MODIFY, AND DROP column operations at one time, and RENAME table is the last operation.

For instance:

```
alter table test rename int11 to int1;
alter table test rename int1 to int2, rename to test22;
alter table test22 rename to test;
alter table test add column1 int DEFAULT 3 NULL, drop clob1, modify double1 int;
```

TRUNCATE TABLE [IF EXISTS] table_name

Remove all table rows.

For instance:

```
truncate table test;
```

PACK TABLE [IF EXISTS] table_name

pack the table.

For instance:

```
pack table test;
```

RENAME TABLE table_name TO table_name2

Rename the table.

For instance:

```
RENAME table test to test1;
```

LOCK TABLE table_name

lock the table. Returns 1 if sucess, 0 if failed to lock a table.

For instance:

```
lock table test;
```

UNLOCK TABLE table_name

unlock the table. Returns 1 if sucess, 0 if failed to unlock a table.

For instance:

```
unlock table test;
```

CREATE [UNIQUE] INDEX [IF NOT EXISTS] indexName[(keylength)][,...] [OF indexFileName] ON tableName (expression [UNIQUE] [PRIMARY KEY] [ASC|DESC] [FOR expression][,...])

Create an index file which can contains one or more index expressions for a table. The HXTT Access will utilize index when condition_expression contains indexed expression.

For instance:

DROP INDEX [IF EXISTS] {ALL | indexName[,...]} [of indexFileName] ON table_name

Removes the specified index from the database.

For instance:

```
drop index all on test;
```

REINDEX {ALL | indexName[,...]} ON table_name

Rebuild the specified index.

For instance:

```
reindex all on test;
```

CREATE SEQUENCE [IF NOT EXISTS] sequence_name [AS {INT|SMALLINT|TINYINT|BIGINT}] [START [WITH] n] [INCREMENT [BY] n] [MINVALUE n | NO MINVALUE] [MAXVALUE n | NO

MAXVALUE] [CACHE n | NO CACHE] [[NO] CYCLE]

sequence_name: [catalog.]sequenceName

The optional clause START WITH n allows the sequence to begin anywhere. The default starting value is minvalue for ascending sequences and maxvalue for descending ones. The optional clause INCREMENT BY n specifies which value is added to the current sequence value to create a new value. A positive value will make an ascending sequence, a negative one a descending sequence. The default value is 1. The optional clause MINVALUE n determines the minimum value a sequence can generate. If this clause is not supplied or NO MINVALUE is specified, then defaults will be used. The defaults are 1 and -128(-32768,0x80000000,0x8000000000000000L) for ascending and descending sequences, respectively. The optional clause MAXVALUE n determines the maximum value for the sequence. If this clause is not supplied or NO MAXVALUE is specified, then default values will be used. The defaults are 127(32767,0x7fffffff,0x7fffffffffffffffL)and -1 for ascending and descending sequences, respectively. The optional clause CACHE cache specifies how many sequence numbers are to be preallocated and stored in memory for faster access. The minimum value is 1 (only one value can be generated at a time, i.e., no cache), and this is also the default. The maximum value for cache is 65535. The CYCLE option allows the sequence to wrap around when the maxvalue or minvalue has been reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated will be the minvalue or maxvalue, respectively. If NO CYCLE is specified, any calls to nextval after the sequence has reached its maximum value will throw an exception. If neither CYCLE or NO CYCLE are specified, NO CYCLE is the default.

For instance:

```
create sequence if not exists userID start WITH 100 increment by 2 maxvalue 2000 cache
5 cycle;
```

DROP SEQUENCE [IF EXISTS] sequence_name

Removes a sequence from the database. IF that sequence doesn't exist without using IF EXIST, an SQLException will be thrown.

For instance:

```
drop sequence if exists userID;
```

ALTER SEQUENCE sequence_name [AS {INT|SMALLINT|TINYINT|BIGINT}] [RESTART [WITH] n] [INCREMENT [BY] n] [MINVALUE n | NO MINVALUE] [MAXVALUE n | NO MAXVALUE] [CACHE n | NO CACHE] [[NO] CYCLE]

ALTER SEQUENCE changes the parameters of an existing sequence generator. Any parameter not

specifically set in the ALTER SEQUENCE command retains its prior setting.

For instance:

```
alter sequence userID restart WITH 100 increment by 1 maxvalue 5000;
```

SET TRANSACTION transaction_mode [, ...]

transaction_mode: { ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE } | { READ WRITE | READ ONLY }

Sets the transaction characteristics of the current transaction. It effects any subsequent transactions in the same connection. java.sql.Connection.setTransactionIsolation(int level) and java.sql.Connection.setReadOnly(boolean readOnly) can do the same task.

For instance:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

START TRANSACTION [transaction_mode [, ...]]

Begins a new transaction block. java.sql.Connection.setAutoCommit(false), java.sql.Connection.setTransactionIsolation(int level), and java.sql.Connection.setReadOnly(boolean readOnly) can do the same task.

For instance:

```
START TRANSACTION;
```

COMMIT [WORK]

Terminates the current transaction and makes all changes under the transaction persistent. It commits the changes to the database. java.sql.Connection.commit() can do the same task.

For instance:

```
commit;
```

ROLLBACK [WORK] [TO [SAVEPOINT] savepoint_name]

Without savepoint_name, terminates the current transaction and rescinds all changes made under the

transaction. It rolls back the changes to the database. With savepoint_name, rolls back all commands that were executed after the savepoint was established. java.sql.Connection.rollback() can do the same task of ROLLBACK [WORK] sql.

For instance:

```
rollback;
```

SAVEPOINT savepoint_name

SAVEPOINT establishes a new savepoint within the current transaction. java.sql.Connection.setSavepoint(String name) and java.sql.Connection.setSavepoint() can do the same task.

For instance:

```
savepoint t1;
```

RELEASE SAVEPOINT savepoint_name

Destroys a savepoint previously defined in the current transaction. java.sql.Connection.releaseSavepoint(Savepoint savepoint) can do the same task.

{ [? =] call procedure_name [(? [, ? [, ...]])] }

Now only HXTT Access supports stored procedure.

For instance:

```
release savepoint t1;
```

DECLARE variable_name[,...] type [DEFAULT expression]

Variable is visiable only in the same connection.

For instance:

```
DECLARE abc CHAR(20) DEFAULT 'Hello';
DECLARE x, y INT;
```

SET variable_name = expression [,...]

expression can be a complicated expresion. BTW, INTO variable[,...] clause of SELECT syntax can set selected columns directly into variables. SELECT id,data INTO x,y FROM test.t1 LIMIT 1;

For instance:

```
SET x = 1+int(55.5),y=2;
SELECT name,id INTO x,y FROM table1 WHERE id=33;
SELECT date(),pi() INTO x,y;
```

Comment Syntax

```
#one-line comment
--one-line comment
/*multiline comment*/
```

For instance:

```
select * /* column list */ from test;#This is a select sql.
```

SQL States

SQL State	Description
01001	Cursor operation conflict
01427	single-row subquery returns more than one row
01428	single-column subquery returns more than one column
01429	subquery returns mismatch column number
01430	single-row subquery returns none row
07006	Restricted data type attribute violation
08000	Connection exception
08003	Connection not open
08007	Connection failure during transaction
08S01	Remote database access failure
0A000	Feature not supported
0A001	Multiple server transactions
21S01	Insert value list does not match column list
22000	Data exception

22019	Invalid escape character
22023	Invalid parameter value
23000	Integrity constraint violation
24000	Invalid cursor state
25000	Invalid transaction state
26000	Invalid SQL statement name
2A000	Direct SQL syntax error or access rule violation
2D000	Invalid transaction termination
2E000	Invalid connection name
34000	Invalid cursor name
34102	Invalid variable name
34103	Invalid funciton name
34104	Invalid index file name
3C000	Duplicate cursor name
3D000	Invalid catalog name
3F000	Invalid schema name
40000	Transaction rollback
42000	Syntax error or access violation
42001	Syntax error
42002	Access violation
42003	Statement has been closed
60000	System errors
99999	Catch all others
C0100	Unknown CodePageID
C0101	Unknown File Format
C0102	Unknown Table Version
C0103	Unknown Index Version
C0104	Corrupt Index File
C0105	Invalid Record Number
C0106	Convert dirty data into null value
S0001	Base table or view already exists
S0021	Index already exists

S0022	Column not found
S1002	Invalid column number
S1009	Invalid Argument value
S1T00	Timeout expired

Introduction of HXTT Access Packages

HXTT Access contains the only type 4 JDBC(1.2, 2.0, 3.0) driver packages for Microsoft Access version from 95, 97, 2000, XP, 2002, to 2003, which supports transaction, embedded access, client/server mode, and remote access(map network drive, SAMBA protocol, HTTP protocol, HTTPS protocol, FTP protocol, and UNC path).

Registration benefits:

- Full version of HXTT Access without limitations
- Free technical support by forum and email
- Free online major and minor updates in the guarantee period

Trial version is for your evaluation only. If you want to use HXTT Access after a trial period, you have to purchase a licensed copy from HXTT.

NOTE:
Differences between the trial version and the licensed version:

- The trial version of the driver is available to use free for a **30-day** trial period.
- The trial version of the driver allows executing not more than **50** queries once.
- SELECT queries return the first **1000** rows in the result set.

Our Other JDBC Products

HXTT DBF - JDBC(1.2, 2.0, 3.0) driver packages for Xbase database (dbase, Visual DBASE, SIx Driver, SoftC, Codebase, Clipper, Foxbase, Foxpro, VFP, xHarbour, Halcyon, Apollo, Goldmine, and BDE)

HXTT Paradox - JDBC(1.2, 2.0, 3.0) driver packages for Corel Paradox version from 3.0, 3.5, 4.x, 5.x, 7.x to 11.x

HXTT Text (CSV) - JDBC(1.2, 2.0, 3.0) driver packages for raw data, flat text , CSV file, TSV file, PSV file, fixed-length, and variable-length binary file

HXTT Excel - JDBC(1.2, 2.0, 3.0) driver packages for Microsoft Excel version from 95, 97, 98, 2000, 2001, 2002, 2003, to 2004.

Components of HXTT Access Packages

These components are included in the HXTT Access license descriptions :

Component	Description
Access Documentation	Development Documentation
Access JDBC 1.2 Package	JDBC 1.2 and Personal Java compliant driver, and a Database GUI manager for swing1.0.3.
Access JDBC 1.2 Embedded Package	JDBC 1.2 and Personal Java compliant driver without client/server mode support.
Access JDBC 1.2 Remote Access Package	The client side JDBC 1.2 and Personal Java compliant ultra light applet driver
Access JDBC 1.2 Driver's Demo	Demo Code for JDBC 1.2
Access JDBC 2.0 Package	JDBC 2.0 compliant driver, and a Database GUI manager.
Access JDBC 2.0 Embedded Package	JDBC 2.0 compliant driver without client/server mode support.
Access JDBC 2.0 Remote Access Package	The client side JDBC 2.0 compliant ultra light applet driver
Access JDBC 2.0 Driver's Demo	Demo Code for JDBC 2.0
Access JDBC 3.0 Package	JDBC 3.0 compliant driver, and a Database GUI manager.
Access JDBC 3.0 Embedded Package	JDBC 3.0 compliant driver without client/server mode support.
Access JDBC 3.0 Remote Access Package	The client side JDBC 3.0 compliant ultra light applet driver
Access JDBC 3.0 Driver's Demo	Demo Code for JDBC 3.0
HXTT JDBC 3.0 Common Package and Access JDBC 3.0 Core Package	You can use common package if you employ more than one of HXTT JDBC products.

HXTT's JDBC Packages Documentation

Welcome to the HXTT Access v2.1 Documentation

Current documentation can be found here. This documentation is not intended as a complete guide to JDBC programming, but should help to get you started. For more information, refer to the standard JDBC API documentation(supplied with Sun's JDK). Also, take a look at the examples included with the HXTT Access packages. The basic example is used here.

Index:

Chapter 5. Advanced Programming

Chapter 6. SQL Syntax

Chapter 7. Scalar Functions and Aggregate Functions

Chapter 8. Answers to Frequently Asked Questions (FAQ) for the HXTT Access packge

Chapter 9. OpenAPI Programming

Chapter 2. Installation

Index:

System Requirements

HXTT Access packages include a Type 4 JDBC driver. Type 4 indicates that the driver is written in Pure Java, and communicates in the database system's own network protocol. Because of this, the driver is platform independent; once compiled, the driver can be used on any system. HXTT Access can run on any platforms with Java VM, which includes Microsoft Windows, Novell Netware, OS2, UNIX, and LINUX. HXTT Access supports Personal Java, JDK1.0.X, JDK1.1.X, JDK1.2.X, JDK1.3.X, JDK1.4.X and JDK1.5.X. HXTT Access includes a database engine which can support multi-user access. It supports { UNION | INTERSECT | EXCEPT | MINUS } [ALL] query , INNER JOIN, FULL JOIN, LEFT JOIN, RIGHT JOIN, NATURAL JOIN, CROSS JOIN, and subquery which includes single-row subquery, multirow subquery, multiple-column subquery, inline views, and correlated subquery.

Setting the CLASSPATH

When java loads any class, it searches a list known as the classpath. This is a list of directories where classes are placed, or a list of jar files (archives containing classes and other resources) or both. HXTT Access driver is a Type 4 driver. You can do this in many different methods, but the most command are:

1. Setting the CLASSPATH environment variable.
2. putting it on the command line using the -cp parameter.
3. placing it in the JVM/s lib/ext directory.
4. extract all files in jar file into the directory of your application.

You can know detailed information about "Setting the Classpath" from your JDK Tools and Utilities. Let's use JDBC3.0 package as a simple sample. To put Access_JDBC30.jar into your class path, you should use "export CLASSPATH=/usr/share/lib/Access_JDBC30.jar:$CLASSPATH" on Solaris and Linux, and "SET CLASSPATH=\javalib\Access_JDBC30.jar;%classpath%" on Windows.

Loading the Driver

Any source that uses JDBC needs to import the java.sql package by using " import java.sql.*;".

HXTT Access driver' name is **com.hxtt.sql.access.AccessDriver**, and you can uses it without involving hard coding the driver into your code. You do this by setting the jdbc.drivers system property. For example, for command line apps you can use:
java -Djdbc.drivers=com.hxtt.sql.access.AccessDriver yourApp
Then, the JVM upon startup will load the drivers automatically. Some applications (JBoss, Tomcat etc) support a .properties file which they use to save putting this on the command line.

The second method is the most common and involves you loading the driver yourself. It's simple:
Class.forName("com.hxtt.sql.access.AccessDriver");
From then on you can get connections from DriverManager.
Note: If Class.forName() throws ClassNotFoundException, you should check your classpath.

Connecting to the Database

After the driver has been registered with the DriverManager, you can obtain a Connection instance that is connected to a particular database by calling DriverManager.getConnection(). With JDBC, a database is represented by a URL (Uniform Resource Locator).

```
        Embedded:
                jdbc:access:[//]/[DatabasePath][?prop1=value1[;prop2=value2]] (You
can omit that "//" characters sometimes)
                        For example:
                                "jdbc:access:/."
```

```
                                "jdbc:access:/c:/data" for Windows driver
                                "jdbc:access:///c:/data" for Windows driver
                                "jdbc:access:////usr/data" for unix or linux
                                "jdbc:access://///192.168.10.2/sharedir" for UNC path
                                "jdbc:access:/./data"
                                "jdbc:access:/./data/mydata.mdb"
        Remote Access (client/server mode):
                jdbc:access://host:port/[DatabasePath]
                        For example: "jdbc:access://domain.com:3099/c:/data" if one
AccessServer is run on the 3099 port of domain.com

"jdbc:access://domain.com:3099/c:/data/mydata.mdb"
        Compressed Database:(.ZIP, .JAR, .GZ, .TAR, .BZ2, .TGZ, .TAR.GZ, .TAR.BZ2)
                jdbc url format is the same as embedded url and remote url.
                        For example:
                                "jdbc:access:/c:/test/testaccess.zip
        Memory-only Database:
                jdbc:access:/_memory_/
        URL Database:(http protocol, https protocol, ftp protocol)
                jdbc:access:http://httpURL
                jdbc:access:https://httpsURL
                jdbc:access:ftp://ftpURL
                        For example:
                                "jdbc:access:http://www.hxtt.com/test"
        SAMBA Database:(smb protocol)

jdbc:access:smb://[[[domain;]username[:password]@]server[:port]/[[share/[dir/]file]]][?[param=value]]
                        For example:

"jdbc:access:smb://test1:123@100.100.13.94/accessfiles".zone"
        UNC path JDBC url:
                jdbc:access:/uncpath
                jdbc:access:///uncpath
                        For example:
                                "jdbc:access:/\\PC17\c$\values"
                                "jdbc:access:/\\PC17\val"
        Free JDBC url:(Warning: only use it for special project)
                jdbc:access:/" or "jdbc:access:///". Then you can use some full UNC
path names in SQL to visit anywhere where your Java VM has right to access.
                        For instance:
                                select * from \\amd2500\e$\accessfiles\test;
                                elect * from "\\amd2500\d$\accessiles".test;
                                select * from ".".test;

         HXTT Access supports seamlessly data mining on memory-only table, physical
table, url table, compressed table, SAMBA table in a sql. More details
         is in Advanced Programming chapter.
```

To connect, you need to get a Connection instance from JDBC. To do this, you use the DriverManager.getConnection() method:

```
Connection con = DriverManager.getConnection(url, properties);
```

There are a few different signatures for the getConnection() method. You should see the API documentation that comes with your JDK for more specific information on how to use them. You can specify additional properties to the JDBC driver by placing them in a java.util.Properties instance and passing that instance to the DriverManager when you connect.

Property Name	Definition	Default Value
host	The remote host on which one AccessServer is running	null
port	The port on which one AccessServer is listening	null

serverType	The type of AccessServer on the remote host	null
user	The user to connect as	null
password	The password to use when connecting	null
charSet	To specify a Character Encoding Scheme other than the client default. You can find a Supported Encodings list of file:///c|/jdk1.2/docs/guide/internat/encoding.doc.html. Cp895(Czech MS - DOS 895), Cp620(Polish MS - DOS 620) and Mazovia are extra supported although JVM doesn't support those.	null
lockType	To specify a compatible lock for other applications of Access. This function hasn't been complemented.	null
lockTimeout	To specify Access driver's timeout in milliseconds to wait until other processes or Access applications released record lock or table lock. 0 means a default value, and <0 means no wait.	1000
cryptType	To specify a crypt type for Table Encryption and Column Level Encryption. All new created table in this connection will become crypted table. You can use DES, TRIDES, and BLOWFISH now. It is not compatiable with MS Access.	null
cryptKey	To specify a crypt key. Without encrypt key, CREATE TABLE won't create crypted table.	null
storeCryptKey	Indicates whether crypt key is stored in crypted table. If stored, crypted table can be opened automatically in any connection without predefined crypt properites. If not stored, cryptd table can only be opened with correct key, and none include us can help you in cracking your data without the correct key.	false
tmpdir	Indicates whether set a temp directory, Default: the value of JVM's "java.io.tmpdir" property. If that value is incorrect, uing the directory of JDBC url.	null
delayedClose	Indicates the delayed seconds for close transaction. That option is used to avoid frequent close/open table operations for following sqls. You can use 0~120 seconds. Default: 3.	null
maxIdleTime	Indicates the max idle time in minute for remote connection. That option is mainly used to avoid closing automatically idle remote connection for connection pool. Embedded idle connectoin won't be closed automatically except for garbage collection. You can use 1~1440 minutes. Default: 30.	null
maxCacheSize	Indicates the max memory utilization for per table on automatic temporary index or matched result cache. You can use 16~65536 kilo bytes. Default: 1024.	null
versionNumber	Access JET Version Number. You can use null, "JET3"(JET version 3.0), "JET4" (JET version 4.0). This parameter is only used for CREATE DATABASE .	JET4
ODBCTrimBehavior	Indicates whether works like MS Access ODBC driver to ignore tail space characters in condition expression. You can use null, true, false	true
otherExtension	Indicates whether Access driver supports other extension beside 'MDB' and 'MDE'. You can use comma to assign more than one extension, for instance, otherExtension=DB,ACR .	null

When your code then tries to open a Connection, and you get a No driver available SQLException being thrown, this is probably caused by the driver not being in the class path, or the JDBC url not being correct.

To close the database connection, simply call the close() method to the Connection:

```
con.close();
```

Chapter 3. Statement

Index:

Creating a Statement Instance

Once a Connection is established, it can be used to create Statements and PreparedStatements. Any time you want to issue SQL statements to the database, you require a Statement or PreparedStatement instance. To get a Statement object, you call the createStatement() method on the Connection object you have retrieved via the DriverManager.getConnection() method. Once you have a Statement object, you can execute a SELECT query by calling the executeQuery(String SQL) method with the SQL you want to use. To update data in the database use the executeUpdate(String SQL) method. This method returns the number of rows affected by the update statement. If you don't know ahead of time whether the SQL statement will be a SELECT or an UPDATE/INSERT, then you can use the execute(String SQL) method. This method will return true if the SQL query was a SELECT, or false if an UPDATE/INSERT/DELETE query. If the query was a SELECT query, you can retrieve the results by calling the getResultSet() method. If the query was an UPDATE/INSERT/DELETE query, you can retrieve the affected rows count by calling getUpdateCount() on the Statement instance. This is explained in the following sections.

Issuing a Query

A simple sample can illustrates more than some words:

```
String sql = "select * from test where int1>0";

Connection con = DriverManager.getConnection(url, "", "");

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(sql);

ResultSetMetaData resultSetMetaData = rs.getMetaData();
int iNumCols = resultSetMetaData.getColumnCount();
for (int i = 1; i <= iNumCols; i++) {
    System.out.println(resultSetMetaData.getColumnLabel(i)
                       + "  " +
                       resultSetMetaData.getColumnTypeName(i));
}

Object colval;
while (rs.next()) {
    for (int i = 1; i <= iNumCols; i++) {
        colval = rs.getObject(i);
        System.out.print(colval + "  ");
    }
```

```
        System.out.println();
    }

    rs.close();
    stmt.close();
    con.close();
```

This example issues the same query as before but uses a PreparedStatement and a bind value in the query.

```
    String sql = "select * from test where int1>?";

    Connection con = DriverManager.getConnection(url, "", "");

    PreparedStatement stmt = con.prepareStatement(sql);

    stmt.setInt(1, 0);

    ResultSet rs = stmt.executeQuery();

    ResultSetMetaData resultSetMetaData = rs.getMetaData();
    int iNumCols = resultSetMetaData.getColumnCount();
    for (int i = 1; i <= iNumCols; i++) {
        System.out.println(resultSetMetaData.getColumnLabel(i)
                           + "  " +
                           resultSetMetaData.getColumnTypeName(i));
    }

    Object colval;
    while (rs.next()) {
        for (int i = 1; i <= iNumCols; i++) {
            colval = rs.getObject(i);
            System.out.print(colval + "  ");
        }
        System.out.println();
    }

    rs.close();
    stmt.close();
    con.close();
```

You can use a single Statement instance as many times as you want. You could create one as soon as you open the connection and use it for the connection's lifetime. But you have to remember that only one ResultSet can exist per Statement or PreparedStatement at a given time. When you are done using the Statement or PreparedStatement, you should close it.

Before reading any values from ResultSet, you have to call next(). This returns true if there is a result, but more importantly, it prepares the row for processing. Under the JDBC specification, you should access a column only once. It is safest to stick to this rule, although the HXTT Access driver will allow you to access a column as many times as you want. You should close a ResultSet by calling close() once you have finished using it too. Once you make another query with the Statement used to create a ResultSet, the currently open ResultSet instance is closed automatically. The HXTT

Access driver supports updatable result sets, but an updatable query can only span one table (i.e. no joins).

Performing Updates

To change data (perform an INSERT, UPDATE, or DELETE), you should use the executeUpdate() method. This method is similar to the method executeQuery() used to issue a SELECT statement, but it doesn't return a ResultSet; instead it returns the number of rows affected by the INSERT, UPDATE, or DELETE statement. This example will issue a simple UPDATE statement and print out the number of rows updated.

```
String sql="update test set boolean1=not boolean1 where recno()%5=0";

Connection con = DriverManager.getConnection(url, "", "");

Statement stmt=con.createStatement();

int updateCount=stmt.executeUpdate(sql);
System.out.println(sql+" : "+updateCount);

stmt.close();
con.close();
```

Creating and Modifying Database Objects

To create, modify or drop a database object like a table, index, or view, you should use the execute() method. This method is similar to the method executeUpdate(), but it doesn't return a result. This example will drop a table.

```
String sql="drop table test";

Connection con = DriverManager.getConnection(url, "", "");

Statement stmt=con.createStatement();

stmt.execute(sql);

stmt.close();
con.close();
```

The HXTT Access driver can create, modify or drop a database object like a table, index, or view through executeUpdate(), but the returned result is unexpectable. For instance, dropping a table can return 1(only one table file), 2(two table files, or one table files and one index file), 3, 4, and so on. The returned result of executeUpdate() is valuable when it creates something with IF NOT EXISTS clause, or drops something with IF EXISTS clause. This example will drop a test table if that table exists.

```
String sql="drop table if exists test";

Connection con = DriverManager.getConnection(url, "", "");
```

```
Statement stmt=con.createStatement();

boolean droppedFlag=stmt.executeUpdate(sql)!=0;

stmt.close();
con.close();
```

Chapter 4. ResultSet

Index:

ResultSet Overview

A ResultSet is a Java object that contains the results of executing an SQL query. In other words, it contains the rows that satisfy the conditions of the query. The data stored in a ResultSet object is retrieved through a set of get methods that allows access to the various columns of the current row. The ResultSet.next method is used to move to the next row of the ResultSet, making it the current row.

A ResultSet object maintains a cursor, which points to its current row of data. The cursor moves down one row each time the method next is called. When a ResultSet object is first created, the cursor is positioned before the first row, so the first call to the next method puts the cursor on the first row, making it the current row. ResultSet rows can be retrieved in sequence from top to bottom as the cursor moves down one row with each successive call to the method next. A scrollable result set's cursor can move both forward and backward as well as to a particular row. The following methods move the cursor backward, to the first row, to the last row, to a particular row number, to a specified number of rows from the current row, and so on: previous, first, last, absolute, relative, afterLast, and beforeFirst. As with scrollability, making a ResultSet object updatable increases overhead and should be done only when necessary. That said, it is often more convenient to make updates programmatically, and that can only be done if a result set is made updatable.

The HXTT Access driver supports scrollable updatable result set.

Providing Performance Hints

The number of rows that should be fetched from the database each time new rows are needed. The number of rows to be fetched is called the fetch size, and it can be set by two different methods: Statement.setFetchSize and ResultSet.setFetchSize. The statement that creates a ResultSet object sets the default fetch size for that ResultSet object, using the Statement method setFetchSize. The following code fragment sets the fetch size for the ResultSet object rs to 10. Until the fetch size is changed, any result set created by the Statement object stmt will automatically have a fetch size of 10.

```
Statement stmt = con.createStatement();
stmt.setFetchSize(10);
ResultSet rs = stmt.executeQuery("SELECT * FROM test");
```

A result set can, at any time, change its default fetch size by setting a new fetch size with the ResultSet version of the method setFetchSize. Continuing from the previous code fragment, the following line of code changes the fetch size of rs to 50:

```
stmt.setFetchSize(50);
```

Normally the most efficient fetch size is already the default for the HXTT Access driver. The method setFetchSize simply

allows a programmer to experiment to see if a certain fetch size is more efficient than the default for a particular application.

Performing Updates

A ResultSet object may be updated (have its rows modified, inserted, or deleted) programmatically if its concurrency type is CONCUR_UPDATABLE. The following example demonstrates show how to update, delete, and insert data.

```
PreparedStatement stmt = con.prepareStatement(
    "select int1,float1,clob1 from test where double1<=?",
    ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

stmt.setFetchSize(12);

stmt.setDouble(1, 0);
ResultSet rs = stmt.executeQuery();

ResultSetMetaData resultSetMetaData = rs.getMetaData();
int iNumCols = resultSetMetaData.getColumnCount();
for (int i = 1; i <= iNumCols; i++) {
    System.out.println(resultSetMetaData.getColumnLabel(i));
}

Object colval;
while (rs.next()) {
    for (int i = 1; i <= iNumCols; i++) {
        colval = rs.getObject(i);
        System.out.print(colval + "  ");
    }
    System.out.println();
}

rs.first();
rs.relative(5);
rs.updateString(3, "eeees333ee3");
rs.updateFloat("float1", 11111.2111f);
rs.updateRow();

rs.absolute(6);
rs.deleteRow();

rs.relative( -2);
rs.refreshRow();

rs.moveToInsertRow();
rs.updateInt(1, 10000);
rs.updateFloat(2, 1000000.0f);
rs.updateObject(3,
                "abc" + (new java.sql.Time(System.currentTimeMillis())));
rs.insertRow();
rs.moveToCurrentRow();

System.out.println("After be updated:");
```

```
            rs.beforeFirst();
            while (rs.next()) {
                for (int i = 1; i <= iNumCols; i++) {
                    colval = rs.getObject(i);
                    System.out.print(colval + "  ");
                }
                System.out.println();
            }

            rs.close();
            stmt.close();
            con.close();
```

Serializing ResultSet

The HXTT Access driver's result set is Serializable.

```
        // serialize the resultSet
        java.io.FileOutputStream fileOutputStream = new
java.io.FileOutputStream("testrs.tmp");
        java.io.ObjectOutputStream objectOutputStream = new
java.io.ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(rs);
        objectOutputStream.flush();

        rs.close();
        rs = null;

        // deserialize the resultSet
        java.io.FileInputStream fileInputStream = new
java.io.FileInputStream("testrs.tmp");
        java.io.ObjectInputStream objectInputStream = new
java.io.ObjectInputStream(fileInputStream);
        rs = (ResultSet) objectInputStream.readObject();
```

Chapter 5. Advanced Programming

Index:

Sending Very Large IN Parameters

The methods setBytes, setString, setBinaryStream, setAsciiStream, setCharacterStream, setBlob, and setClob are capable of sending unlimited amounts of data. The following code illustrates using a stream to send the contents of a file as an IN parameter.

```
            String sql="update test SET clob1 = ?, blob1=? WHERE float1>=?*PI()%5 or
float1=0";
            java.sql.PreparedStatement pstmt = con.prepareStatement(sql);

            java.io.File file = new java.io.File(dir+"/somechar.txt");
            int fileLength =(int) file.length();
            java.io.InputStream fin = new java.io.FileInputStream(file);
            pstmt.setCharacterStream(1,new java.io.InputStreamReader(fin),
fileLength);
            pstmt.setObject(2, "A serialized class");
            pstmt.setFloat(3,0);
            pstmt.executeUpdate();
            pstmt.close();
```

Set Record Lock Manually

LockFlag is a virtual column for row lock flag. You can use "select _LockFlag_,* from yourTable" to get an Updatable ResultSet, then use three functions below:
boolean ResultSet.setBoolean("_LockFlag_",true)//Lock the current row.
boolean ResultSet.setBoolean("_LockFlag_",false);//Unlock the current row.
boolean ResultSet.getBoolean("_LockFlag_")//indicates whether the current row has been locked by other process or application.
If ResultSet.close() is called, all pending record locks will be released automatically. "update yourTable set _LockFlag_=true where condition", and "update yourTable set _LockFlag_=false where condition" can lock/unlock records too, but you have to take care of every record lock.

```
            Connection connection1=
DriverManager.getConnection("jdbc:DBF:/.",properties);

            Statement stmt1 =
connection1.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
            stmt1.executeUpdate("CREATE TABLE IF NOT EXISTS testlock(int1 int,char1
varchar(100));"
                                +"INSERT INTO testlock VALUES(1,'DFFDFSDF');"
                                +"INSERT INTO testlock VALUES(2,'aaaa');"
                                );

            ResultSet rs=stmt1.executeQuery("select _lockFlag_,* from testlock where
int1=1");
//            ResultSet rs=stmt1.executeQuery("select recno(),_lockFlag_,* from
testlock where int1=1");

            rs.next();

            boolean lockResult=rs.getBoolean("_LockFlag_");//indicates whether the
current row has been locked by other process or application
            if(lockResult){
                System.out.println("Maybe other application has locked it!");
            }

            //Through moving the cursor of ResultSet, many rows can be locked  at the
same time.
            rs.updateBoolean("_LockFlag_",true);//Lock Row
            rs.updateRow();

            boolean isLockedResult=rs.getBoolean("_lockFlag_");//indicates whether
the current row has been locked by other process or application
            if(!isLockedResult){
                System.out.println("It's impossible since the current row is just
locked!");
            }

            Connection connection2=
DriverManager.getConnection("jdbc:DBF:/.",properties);
            Statement stmt2 =
connection2.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
            if(true){//Whether show a wrong code block.
                try{
                    int result = stmt2.executeUpdate(
                        "UPDATE testlock set int1=1 where int1=1");
                    System.out.println("update count:" + result);
                }catch(SQLException e){
                    System.out.println("update error:"+e);//lock error
                }

                rs.updateInt("int1",1);
```

```
                rs.updateRow();//Pass since it's locked by rs.
            }else{
                int result=stmt2.executeUpdate("UPDATE testlock set int1=1 where
int1=1 and not rowlocked()");
                System.out.println("update count:"+result);

                rs.updateInt("int1",1);
                rs.updateRow();//Pass since it's locked by rs.

                result= connection1.createStatement().executeUpdate("UPDATE testlock
set int1=1 where int1=1");//Pass since it's a statement of the same connection.
                System.out.println("update count:"+result);


                rs.updateBoolean("_LockFlag_",false);////Unlock Row
                rs.updateRow();
                isLockedResult=rs.getBoolean("_lockFlag_");//indicates whether the
current row has been locked by other process or application
                if(isLockedResult){
                    System.out.println("Falied to unlock the current row!");
                }

                result=stmt2.executeUpdate("UPDATE testlock set int1=1 where
int1=1");
                //BTW, you can use "UPDATE testlock set int1=int1+1 where ..." in a
multi-user. DBF will fetch the latest int1 value for calculation.
                System.out.println("update count:"+result);
            }

            rs.close();


            stmt2.close();
            connection2.close();

            stmt1.close();
            connection1.close();
```

Table Level Encryption

If you create table in a connection with crypt properites, those table will become encrypted tables. You needn't take care too much about encrypt/decrypt since it's a Table LEVEL Encryption.

```
            properties.setProperty("cryptType", "des");//To specify an crypt type for
Table Encryption and Column Level Encryption. All new created table in this
connection will become crypted table. You can use DES, TRIDES, and BLOWFISH now.
Deafult:null
            properties.setProperty("cryptKey", "123 myKey 456");//To specify an
encrypt key. Without encrypt key, CREATE TABLE won't create crypted table.
            properties.setProperty("storeCryptKey", "true");//Indicates whether crypt
key is stored in crypted table. If stored, crypted table can be opened automatically
in any connection without predefined crypt properites. If not stored, cryptd table
can only be opened with correct key. Default:false
```

```
Connection con = DriverManager.getConnection(url,properties);
```

You needn't encrypt/decrypt a total table sometimes, then you can used some crypt functions to protect your sensitive data:
ENCRYPT(content,cKey,cCryptMethod): Returns a crypted byte[]. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH' now. ENCRYPT function is used for VARBINARY column.
DECRYPT(content,cKey,cCryptMethod): Returns a decrypted byte[]. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH' now.
ENCODE(content): Encodes a BASE64 encoding string.
DECODE(content): Returns a byte[] from a BASE64 string.
ENCODE(content,cKey,cCryptMethod): Crypts and encodes content. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH'. ENCRYPT function is used for VARCHAR column.
DECODE(content,cKey,cCryptMethod): Decodes and decrypts content. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH' now.
For instance:

```
select encode('adsdfsdf');
select decode(encode('adsdfsdf'))+";
select decode(encode('dfdffd233','12345','trides'),'12345','trides')+':('
select decrypt(encrypt('25355','12345','trides'),'12345','trides')+':('
select decrypt(encrypt('25355','12345','des'),'12345','des')+':('
select decrypt(passwd,'12345','des') from test;
insert into users (user,passwd) values('abc',encode('abcpasswd','a key','trides');
select count(*) from user where users=? and passwd=encode(?,'a key','trides');
select count(*) from user where users=? and decode(passwd,'a key','trides')=?;
```

VARBINARY's Encrypted Data Column Length=Maximum length of the non-encrypted data + 1 byte + The number of bytes to the next 8-byte boundary. For instance, your data is 8 byte, you can use varbinary of 9 byte length (or binary of 8 byte) to stored the encrypted data. Your data is 12 byte, you can use varbinary of 17 byte length to stored the encrypted data.
VARCHAR's Encrypted Data Column Length= (VARBINARY's Encrypted Data Column Length)*4/3. For instance, your data is 8 byte, you need 12 byte to stored the BASE64 encoding encrypted data.

Bulk Insert

"CREATE TABLE [IF NOT EXISTS] table-name [(column-identifier data-type [constraint] [,...])] [AS] [SELECT query]", and "INSERT INTO table-name [(column-identifier [,...])] SELECT query" can copy a table to another table or allow insert of multiple rows in one statement. For instance, "CREATE TABLE newtable select * from table1 where column1!=null order by column2;", and "insert into test (int1,char1) select id1,name1 from abc where id1>50 and value1>300". SQL does't permit that table1 is the same table as table2 when INSERT INTO table1 select * from table2, but the HXTT Access supports such an unadvisable operation, for instance,"insert into table1 select * from table1;".

Bulk Insert A ResultSet from any JDBC driver

The HXTT Access supports to insert data from other JDBC drivers. "CREATE TABLE [IF NOT EXISTS] table-name [(column-identifier data-type [constraint] [,...])] ?", and "INSERT INTO table-name [(column-identifier [,...])] ?" is for that purpose.

```
//rs is an open ResultSet from any JDBC driver.
String sql="insert into test ?;";

PreparedStatement pstmt = con.prepareStatement(sql);
```

```
            pstmt.setObject(1,rs);//insert a resultSet into table test.
            pstmt.executeUpdate();

            pstmt.close();


            sql="create table if not exists abcd ?;";
            pstmt = con.prepareStatement(sql);

            pstmt.setObject(1,rs);//insert a resultSet into a new table abcd
            pstmt.executeUpdate();

            pstmt.close();
```

Notes: If your ResultSet.getType()==ResultSet.TYPE_FORWARD_ONLY, and you have used ResultSet.next() to browsed some rows, you won't insert those browsed rows. Other conditions, all rows will be inserted.

BTW, the HXTT Access driver's result set is Serializable.

```
    // serialize the resultSet
    try {
        java.io.FileOutputStream fileOutputStream = new
java.io.FileOutputStream("yourfile.tmp");
        java.io.ObjectOutputStream objectOutputStream = new
java.io.ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(rs);
        objectOutputStream.flush();
        objectOutputStream.close();
        fileOutputStream.close();
    }
    catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
        System.exit(1);
    }


    // deserialize the resultSet
    try {
        java.io.FileInputStream fileInputStream = new
java.io.FileInputStream("yourfile.tmp");
        java.io.ObjectInputStream objectInputStream = new
java.io.ObjectInputStream(fileInputStream);
        rs = (ResultSet) objectInputStream.readObject();
        objectInputStream.close();
        fileInputStream.close();
    }
    catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
        System.exit(1);
    }
```

RowSet

com.hxtt.sql.HxttRowSet can work with any descendent class of java.sql.DataSource. For instance:

```
import java.sql.*;
import java.util.Properties;


import com.hxtt.sql.HxttDataSource;
import com.hxtt.sql.HxttRowSet;

public class testRowSet{
    public static void main(String argv[]){
        try{
            Class.forName("com.hxtt.sql.access.AccessDriver").newInstance();

            HxttDataSource ds=new HxttDataSource();
            ds.setUrl("jdbc:dbf:/f:/dbfiles");

            HxttRowSet rowSet=new HxttRowSet(ds);
            /*
             Another way:
              HxttRowSet rowSet=new HxttRowSet();
              rowSet.setDataSourceName(dsName);
              will use
                         Context ctx = new InitialContext();
                         return (DataSource) ctx.lookup(dataSourceName);
               to load the ds.
             */

            rowSet.setCommand("select * from test");

            rowSet.execute();

            ResultSetMetaData resultSetMetaData = rowSet.getMetaData();
            int iNumCols = resultSetMetaData.getColumnCount();
            for (int i = 1; i <= iNumCols; i++) {
                System.out.println(resultSetMetaData.
                    getColumnLabel(i)
                    + "  " +
                    resultSetMetaData.getColumnTypeName(i));
            }

            rowSet.beforeFirst();
            while (rowSet.next()) {
                for (int i = 1; i <= iNumCols; i++) {
                    System.out.print(rowSet.getObject(i) + "  ");
                }
```

```
                System.out.println();
            }

            rowSet.close();

        }
        catch( SQLException sqle )
        {
            do
            {
                System.out.println(sqle.getMessage());
                System.out.println("Error Code:"+sqle.getErrorCode());
                System.out.println("SQL State:"+sqle.getSQLState());
                sqle.printStackTrace();
            }while((sqle=sqle.getNextException())!=null);
        }
        catch( Exception e )
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

PooledConnection

For instance:

```
            com.hxtt.sql.HxttConnectionPoolDataSource pds=new
com.hxtt.sql.HxttConnectionPoolDataSource();
            pds.setUrl("jdbc:dbf:/f:/dbffiles");
            javax.sql.PooledConnection pc=pds.getPooledConnection();
```

SSL Connection

SSL Connection has been provided since JDK1.4.X. To use SSL Connection, you should know how to use javax.net.ssl package first. With hxtt.socketclass=SSL **system** property, all of HXTT AccessServer's receiving connections in one JVM will become SSL connection. For client side, using hxtt.socketclass=SSL or hxtt.socketclass=null as **connection** property will overlay hxtt.socketclass **system** property so that it's possible that some connections are SSL connection, but other connections are common connections or customer connections.
For instnace, you can use java -Djavax.net.ssl.keyStore=yourKeyStore -Djavax.net.ssl.keyStorePassword=yourKeyStorePassword -Djavax.net.ssl.trustStore=yourTruststore -Djavax.net.ssl.trustStorePassword=yourTrustStorePassword -Dhxtt.socketclass=ssl -cp yourClassPath com.hxtt.sql.admin.Admin to start a HXTT AccessServer with SSL Connection capability. If you wish to use HXTT AccessServer as Linux(Solaris) dameon or Windows Service without GUI, you should read Run HXTT AccessServer as Windows Service or Linux(Solaris) Daemon too.
java -Djavax.net.ssl.trustStore=yourTruststore -Djavax.net.ssl.trustStorePassword=yourTrustStorePassword -

Dhxtt.socketclass=ssl -cp yourClassPath yourApplication will let your application to use SSL for remote connection.
If you wish to write customer connection, please click Customer Connection.

Run HXTT AccessServer as Windows Service or Linux(Solaris) Daemon

In Linux(Solaris),we assume that you save it to /jdbclib directory.
In Windows,we assume it is c:/ . You should have built the database server configuration by com.hxtt.sql.admin.Admin program. It will create a file named
urlconfig.properties which locate on the user home directory.
For example,in Linux(Solaris),you build the database server configuration in root user,the urlconfig.properties will located at /root directory if the root's home directory is /root;in windows,it will be the C:\Documents and Settings\Administrator.
You should copy the file to other directory for the service program maybe not access the file.In Linux(Solaris),we assume you copy it
to /jdbclib;in windows,we assume it is c:/.

In windows,you can use JavaService.exe(Here to download) to register a window service to start the servers.

Here is a simple bat file to tell you how to register a service,you should change some options accord your enviromnent.After you download these two files ,you can run the bat file to register and start the service at the Control Panel.

In Linux(Solaris),you can use jsvc(Here to download) as a daemon to start the servers for remote connection.
1.You should download the Apache common daemons package(Here to download).
We assume that you save this two files to /commondaemon directory.
2.please run the follows command to enable the exec file property.
chmod +x /commondaemon/jsvc
Attention,the jsvc program has tested at RedHat 9.0 and Sun Open Desktop System 1.0.If it don't work at your enviroment,please download the jsvc source and make a binary program or tell us your environment.
3.run the follows command to know the default run level of your machine.
cat /etc/inittab | grep :initdefault
it's result will be as follows: id:3:initdefault
or
runlevel
it's result will be as follows:N 3
In common,the default run level should be 3 or 5.
4.Please download the hxttjsvcserv script to save it to /etc/init.d directory and run the follows command to enable the file executable bit mask .
chmod +x /etc/init.d/hxttjsvcserv
Attension ,if you don't put HXTT Access Package to /jdbclib directory or jsvc and commons-daemon.jar to /commondaemon directory,you should modify the
hxttjsvcserv file to fit your configuration.
BTW,the default user run this service is root,maybe you should changed it to another low right user.Please see the dbfjsvcserv for more detail
information.
5.cd /etc/rcx.d (x is the run level,in some os,the rcx.d is not directly located in /etc directory,you can use find . -name rcx.d to find where is it)
At first you should list all the file for find the new service's running sequence number;
run the command
ls
You will see some files which starts with K or S,for example,S99local and K99local.

S99local is the run script file when start this machine.
K99local is the stop script file when shut down this machine.
local is the service name.K represent kill and S represent the start.
This two files all are a file linked to /etc/init.d/local.This is,when starting machine,OS will run local script with start parameter and when stopping with stop parameter.
99 is the run sequence number when start this machine.
For example,httpd service will start before this local service and stop after the local service for its start script file name is S15httpd and end script file name is K15httpd.
Find the max running sequence number,in my machine,it is 99,so the new service's running sequence number will be 100.
run the command to build this two file.
ln -s /etc/init.d/hxttjsvcserv S100hxttjsvcserv
ln -s /etc/init.d/hxttjsvcserv K100hxttjsvcserv
now you can run /etc/init.d/hxttjsvcserv start to start the service or reboot your machine to test if this service can auto start.

For Novell Netware OS console without GUI, you can also run directly com.hxtt.sql.admin.HxttService with above same parameters.
On LINUX and UNIX, if you got "Cannot connect to X11 window server. The environment variable DISPLAY is not set.", you should use -Djava.awt.headless=true to run Java in headless mode.

How to Use Memory-only Table, Physical Table, Url table, Compressed table, SAMBA table in a SQL.

1. Compressed Database:(.ZIP, .JAR, .GZ, .TAR, .BZ2, .TGZ, .TAR.GZ, .TAR.BZ2)
   jdbc url format is the same as embedded url and remote url.For example, "jdbc:access:/c:/test/testaccess.zip ,then you can use slect * from aTable to visit aTable table in testaccess.zip file.
   No special requirement for sql. Both of the compressed file name and directory name in compressed file are also used as catalog name. For instance, "jdbc:access:/c:/test", select * from "testaccess.zip".a; select * from "testaccess.zip/files/a.csv"; select * from "b.tar.bz2/java"."history.txt";
   For TAR and BZ2 support, you should download Apache's tarbz2.jar package.
   You can use compressed table in sql with the common table. For instance, select * from "testaccess.zip/files/a.csv",test;
   For case-insensitive sql compatibility, all name of directory and file in compressed file are case-insensitive too.
   Compressed database is reaonly, and all data modification won't be flushed into compressed file.

2. Memory-only Database:
   jdbc url: jdbc:access:/_memory_/
   No special requirement for sql. For instance, create table abc (a char(10));insert into abc values(333);select * from abc;drop table abc;
   Memory-only database is hold commonly in memory, but it will be stored into temporary directory if its length exceed 8MB limitation to avoid memory overburden.
   memory is a speical catalog name for memory-only database. Through _memory_ catalog, memory-only database is visible for all applications in the same JVM. For instance, in an embedded connection, you can use create table _memory_.abc (a char(10));insert into _memory_.abc values(333);select * from _memory_.abc;drop table _memory_.abc; to do the same things.
   You can use memory-only table in sql with the common table. For instance, select * from _memory_.abc,test;
   Memory-only database is volatile, and you can't see your old data any more after restart a JVM.

3. URL Database:(http protocol, https protocol, ftp protocol)
   jdbc:access:http://httpURL
   jdbc:access:https://httpsURL
   jdbc:access:ftp://ftpURL
   For example, "jdbc:access:http://www.hxtt.com/test", then you can use "slect * from aTable to visit aTable table. Because All of http, https, and ftp protocol are case-sensitive, you have to take care of your sql, and use proper table file suffix to avoid FileNotFound exception.

Without URL database url, you can access url database in an embedded connection too. For instance, select * from "http://www.hxtt.com/test/a.tar".a; select * from "http://www.hxtt.com/test/a.jsp?aaa=33"

You can use url table in sql with the common table. For instance, select * from "http://www.hxtt.com/test/a.tar".a,abc;

URL database is reaonly, and all data modification won't be flushed into URL content. If you're using a dial-up network, don't waste time to access too big URL database.

For https support in JDK 1.2.x and 1.3.x, you should download JSSE 1.0.3 package.

4. SAMBA Database:(smb protocol)

jdbc:access:smb://[[[domain;]username[:password]@]server[:port]/[[share/[dir/]file]]][?[param=value]]

For example, "jdbc:access:smb://test1:123@100.100.13.94/accessfiles", then you can use "slect * from aTable to visit aTable table.

Without SAMBA database url, you can access SAMBA database in an embedded connection too. For instance, select * from "smb://test1:123@100.100.13.94/accessfiles/zone"

You can use SAMBA table in sql with the common table. For instance, select * from "smb://test1:123@100.100.13.94/accessfiles".zone,abc;

For SAMBA support, you should download Java CIFS Client Library, which is developed by Michael B. Allen.

HXTT Access supports seamlessly data mining on memory-only table, physical table, url table, compressed table, SAMBA table in a sql. A compressed database can be a URL database or SAMBA database at the same time. It's powerful, and you should ask for HXTT's support if you don't know how to use it for special purpose.

Create Table from any java.io.InputStream object

At Bulk Insert A ResultSet from any JDBC driver, we discuss how to use "CREATE TABLE [IF NOT EXISTS] table-name [(column-identifier data-type [constraint] [,...])] ?" to create a table from any JDBC ResultSet. In fact, that sql syntax can be used to copy and create a table from any java.io.InputStream object. Let's see a sample, which creates a physical table and a memory-only table from a HTTP stream.

```
import java.net.URL;
import java.net.URLConnection;

import java.io.IOException;
import java.io.InputStream;
import java.sql.*;
import java.util.Properties;

public class testInputCreate{
    private static void test(String url){
        System.out.println(url);
        try {
            Class.forName("com.hxtt.sql.text.TextDriver").newInstance();

            Properties properties=new Properties();
            Connection con = DriverManager.getConnection(url,properties);

            String sql;
            PreparedStatement pstmt;

            sql="create table testaaa ?";
            pstmt = con.prepareStatement(sql);

            URL httpurl=new URL("http://www.hxtt.com/test/a.csv");
            URLConnection urlConnection=httpurl.openConnection();
            InputStream is=urlConnection.getInputStream();
```

```
            pstmt.setObject(1,is);//create a table from a HTTP stream
            pstmt.executeUpdate();

            pstmt.close();
            in.close();

            sql = "select * from testaaa";
            Statement stmt=con.createStatement();
            ResultSet rs = stmt.executeQuery(sql);

            ResultSetMetaData resultSetMetaData = rs.getMetaData();
            int iNumCols = resultSetMetaData.getColumnCount();
            for (int j = 1; j <= iNumCols; j++) {
                System.out.println(resultSetMetaData.getColumnLabel(j)
                    + "  " + resultSetMetaData.getColumnTypeName(j)
                    + "  " + resultSetMetaData.getColumnDisplaySize(j)
                    );
            }
            Object colval;

            rs.beforeFirst();
            long ncount = 0;
            while (rs.next()) {
                ncount++;
                for (int j = 1; j <= iNumCols; j++) {
                    colval = rs.getObject(j);
                    System.out.print(colval + "  ");
                }
                System.out.println();
            }
            System.out.println("row count:"+ncount);

            rs.close();

            stmt.execute("drop table testaaa");//remove that testaaa table.

            stmt.close();

            con.close();
        }
        catch( SQLException sqle )
        {
            do
            {
                System.out.println(sqle.getMessage());
                System.out.println("Error Code:"+sqle.getErrorCode());
                System.out.println("SQL State:"+sqle.getSQLState());
                sqle.printStackTrace();
            }while((sqle=sqle.getNextException())!=null);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
```

```
        }
    }

    public static void main(String argv[]) {
        test("jdbc:csv:/f:/textfiles/");
        test("jdbc:csv:/_memory_/");
    }
}
```

Transaction Processing

Index:

Commit Mode

There are two modes for managing transactions within JDBC:

- auto-commit
- manual-commit

java.sql.Connection.setAutoCommit(boolean autoCommit) is used to switch between the two modes. If a connection is in auto-commit mode, then all its SQL statements will be executed and committed as individual transactions. Otherwise, its SQL statements are grouped into transactions that are terminated by a call to either the method java.sql.Connection.commit or the method java.sql.Connection.rollback. By default, new connections are in auto-commit mode. After an application turns auto-commit off, a transaction is started. The transaction continues until either the java.sql.Connection.commit meothod, COMMIT [WORK] sql, the java.sql.Connection.rollback method, or ROLLBACK [WORK] sql is called; after that a new transaction is automatically started.

Calling the commit method ends the transaction. At that stage, HXTT Access checks whether the transaction is valid and raises an exception if a conflict is identified. If a conflict is encountered, your application should determine how to continue, for example whether to automatically retry the transaction or inform the user of the failure. A request to rollback a transaction causes HXTT Access to discard any changes made since the start of the transaction and to end the transaction.

```
connection.setAutoCommit(false); // Explicit transaction handling

Statement stmt = connection.createStatement();

// Loop until transaction successful (or max retry exceeded)
for(int count=0;; count++) {
    stmt.executeUpdate(yourSQL);
    try{
        connection.commit(); // Commit transaction
        break;
    }catch(SQLException sqe) {
        // Check commit error
        if(sqe.getSQLState().equals("40000")) {
            //You can use sqle.getNextException() to know more information

            // Check number of times the transaction has been attempted
            if (count<3) {
                continue;
```

```
                }
            }
            throw sqle;
        }
    }
```

Isolation Levels

An isolation level represents a particular locking strategy employed in the HXTT Access to improve data consistency. The higher the isolation level, the more locking or snapshot involved, and the more time users may spend waiting for data to be freed by another user. The isolation level provided by the HXTT Access determines whether a transaction will encounter the following behaviors in data consistency:

- dirty read: A row changed by one transaction can be read by another transaction before any changes in that row have been committed. For instance, User 1 modifies a row. User 2 reads the same row before User 1 commits. User 1 performs a rollback. User 2 has read a row that has never really existed in the database. User 2 may base decisions on false data.
- non-repeatable read: Where one transaction reads a row, a second transaction alters the row, and the first transaction rereads the row, getting different values the second time (a "non-repeatable read"). For instance, User 1 reads a row but does not commit. User 2 modifies or deletes the same row and then commits. User 1 rereads the row and finds it has changed (or has been deleted).
- phantom read: When one transaction reads all rows that satisfy a WHERE condition, a second transaction inserts a row that satisfies that WHERE condition, and the first transaction rereads for the same condition, retrieving the additional "phantom" row in the second read. For instance, User 1 uses a search condition to read a set of rows but does not commit. User 2 inserts one or more rows that satisfy this search condition, then commits. User 1 rereads the rows using the search condition and discovers rows that were not present before.

Isolation Levels and Data Consistency Definition

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
None	Yes	Yes	Yes
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable	No	No	No

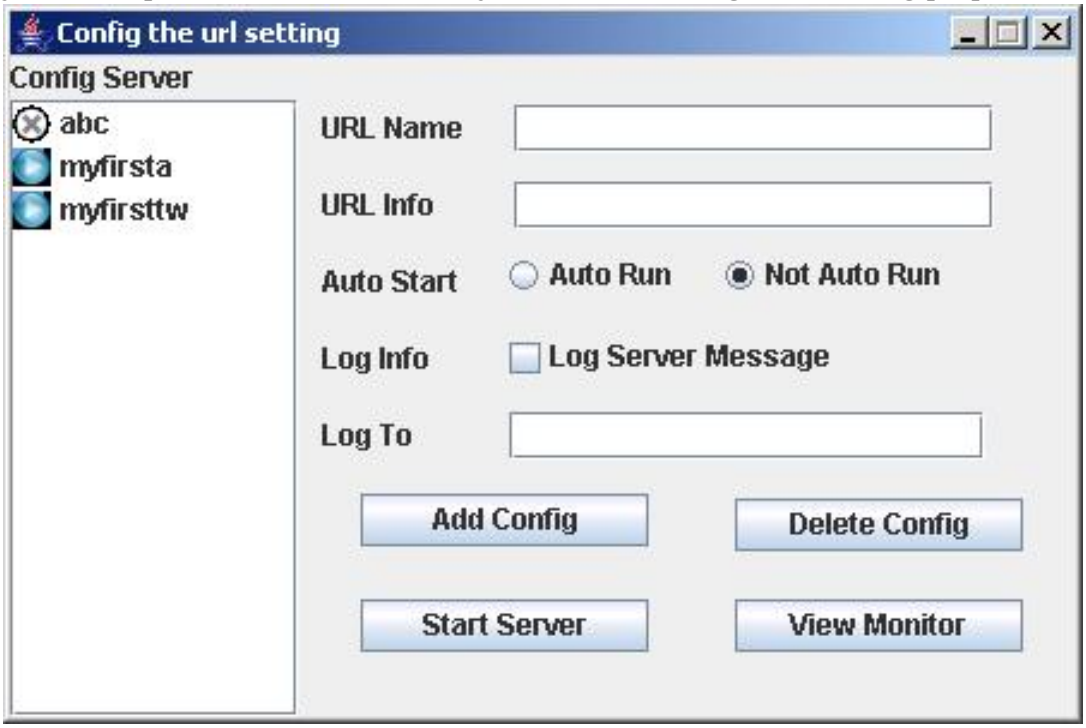
HXTT Access supports all isolation levels.

Performance Hints

- With auto-commit mode, all operations will be done in TRANSACTION NONE level with concurrent support.
- READ UNCOMMITTED level is always faster than three other transaction levels if you don't do many roolback operations.

- Under REPEATABLE READ or SERIALIZABLE mode, the default CLOSE_CURSORS_AT_COMMIT for ResultSet holdability is faster than HOLD_CURSORS_OVER_COMMIT.

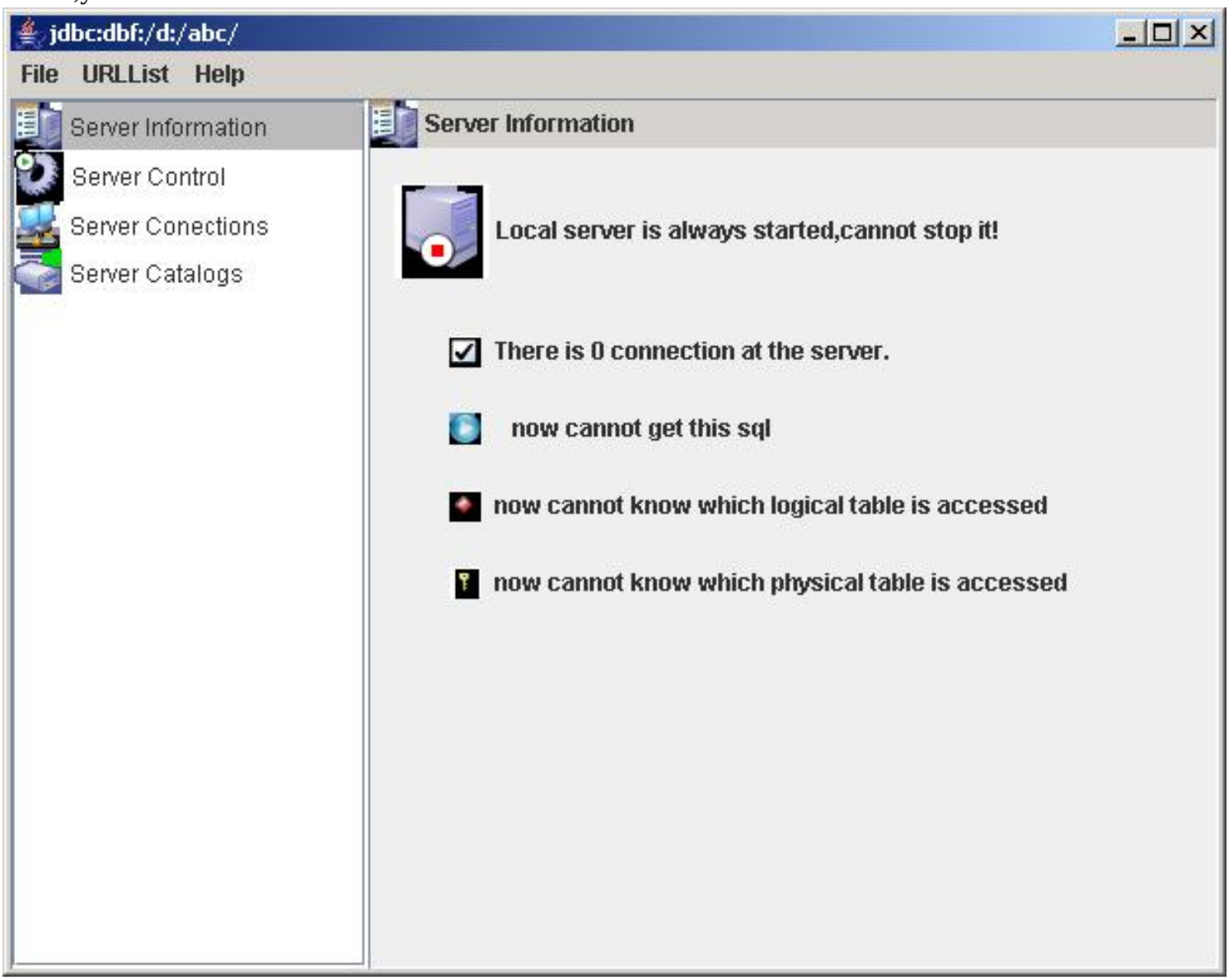
DBAdmin is intergrated enviroment for start,stop,test,manage and monitor the HXTT database software! DBAdmin is contained in the all hxtt java database software package ,you can download the package from here for test use! How to start this DBAdmin program? For example,if you have download the HXTT Access 3.0 package whose filename is Access_JDBC30.jar and save it to c disk,you can run it by java -classpath c:/Access_JDBC30.jar com.hxtt.sql.admin.Admin you will see this window!
In default ,this program will product a file named urlconfig.properties locate in user.home enviroment variable. You can assign the hxtt.urlconfig environment variable to assign the urlconfig.properties path.For example, java -classpath c:/Access_JDBC30.jar -Dhxtt.urlconfig=c:/urlconfig.properties com.hxtt.sql.admin.Admin



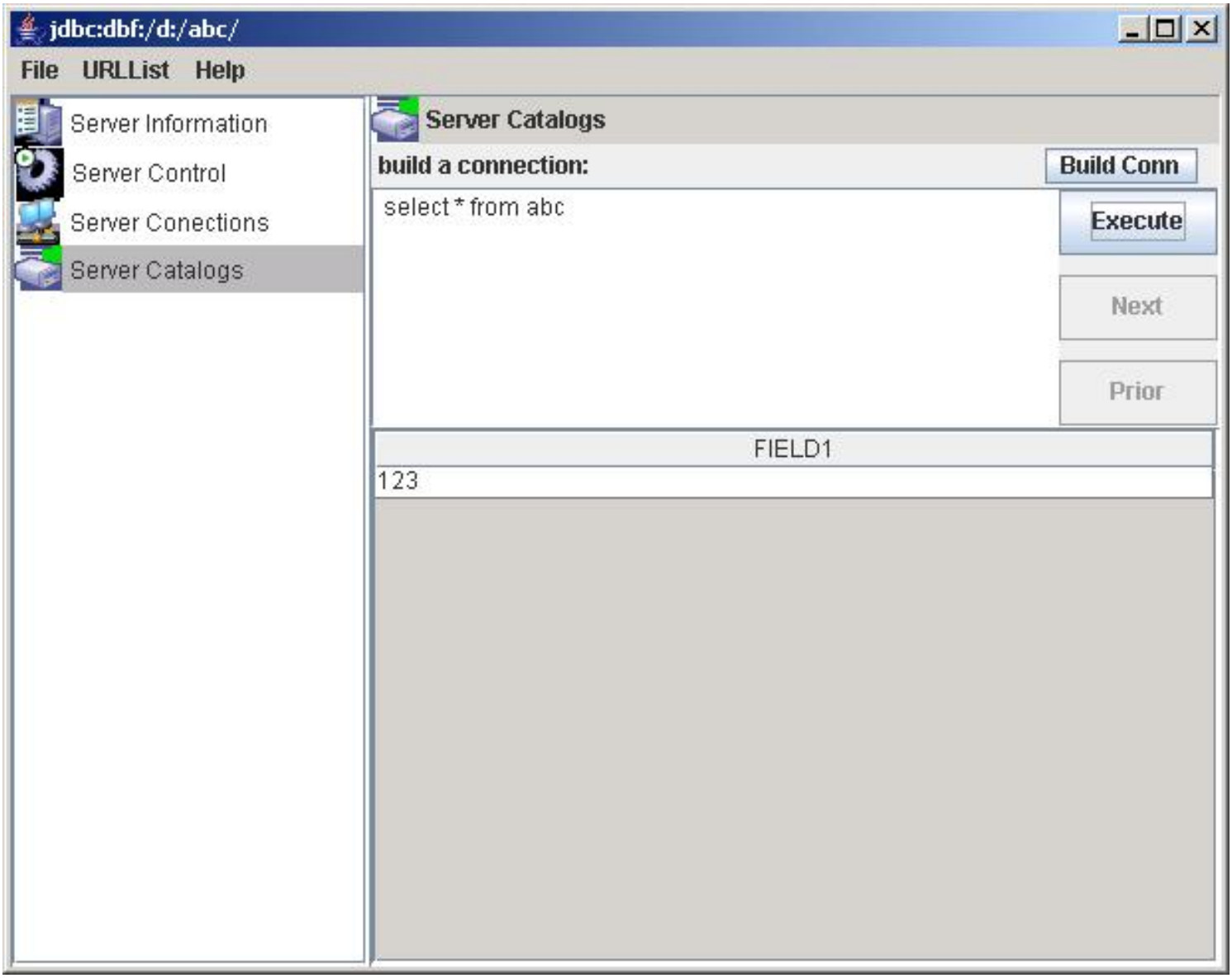
This left list is the configed url list,click a url in this list ,you can see this url config information at the right window!

This url name is used to represented this url config information; this url information text is this url information for start,stop,manage and monotor url information, this url information must be a correct embedded jdbc url or remote jdbc url(this sample describes only an embedded url service, if you wish to access remote, you should use remote url service, for

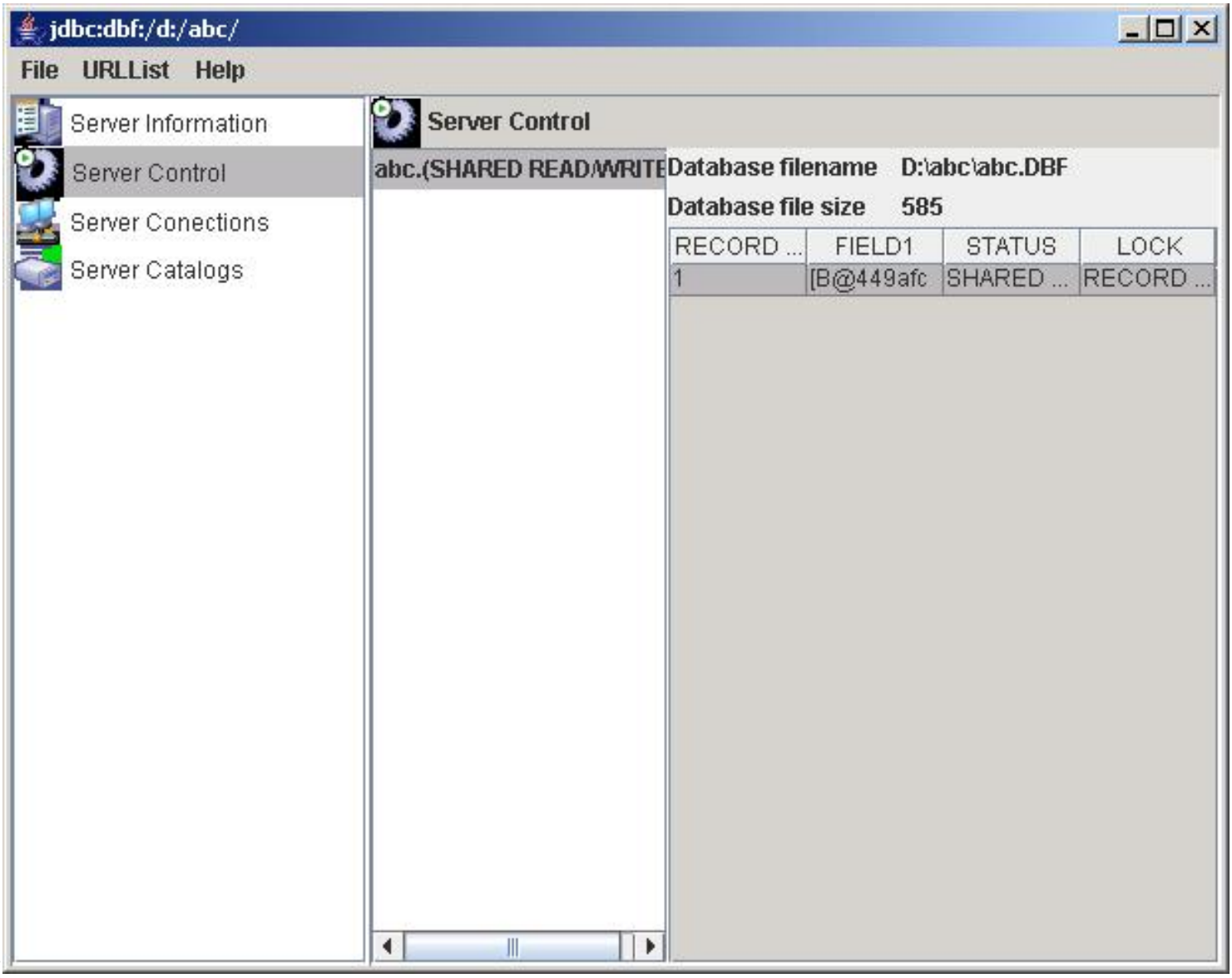
instance, jdbc:access://localhost:3099//usr/datadir); this auto start is used to assign if this remote url start when this dbadmin program start,it is general used to start the hxtt java database server after start the rmi service ! this log information is used to assign if log this server access information to a disk file ; Click this View Monitor button,you can view the select url monitor window!



This Server Information item in the left list is used to show the general information about this selected url! If the server have not started,you can click the right image to start this server, after start this server,you can see four item at this left list! This Server Catalogs is used to build a connection and execute some sql at the server,you can see the result at this right-bottom window!

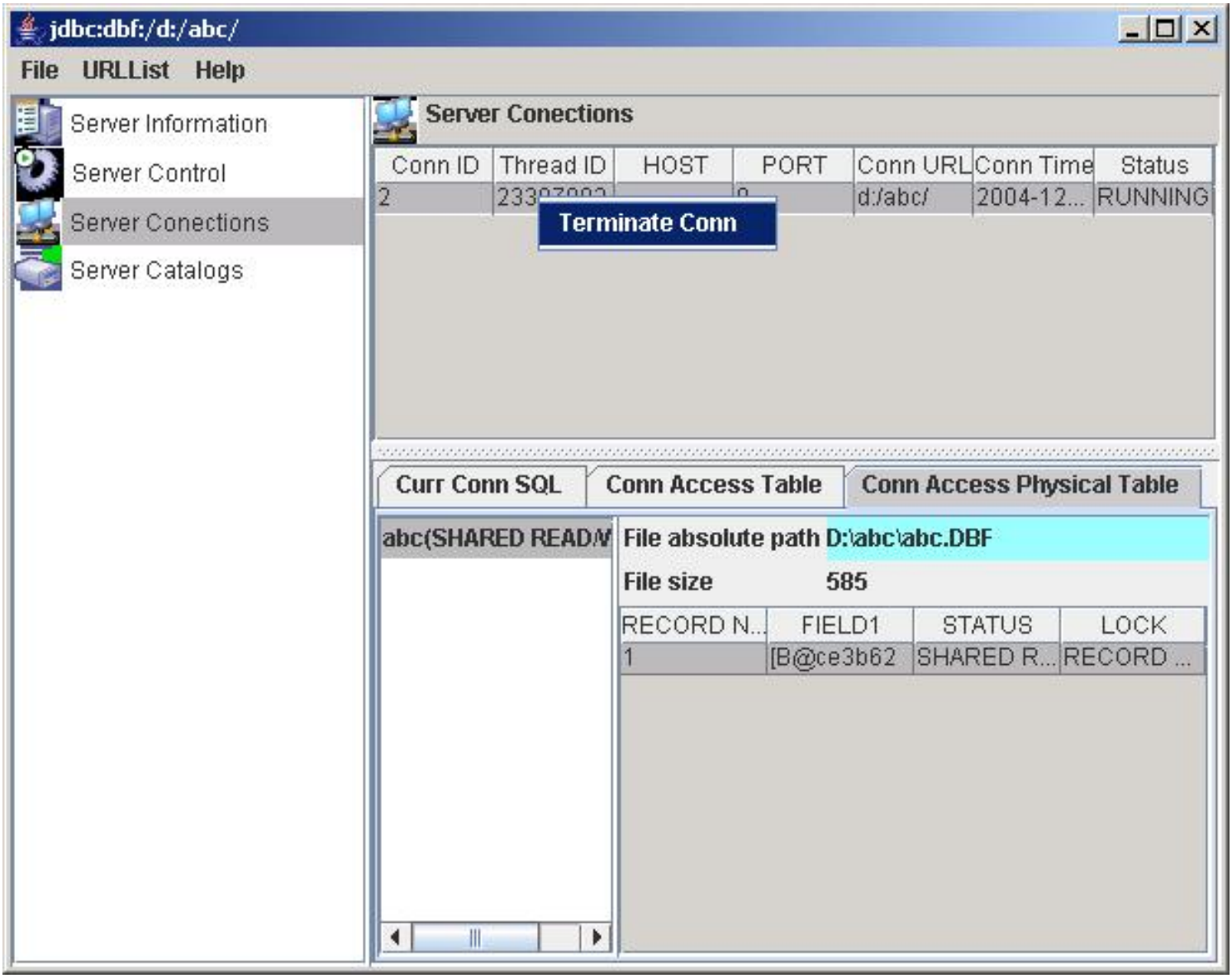


This Server Controls is used to monitor the physical file which is now accessed by the server!

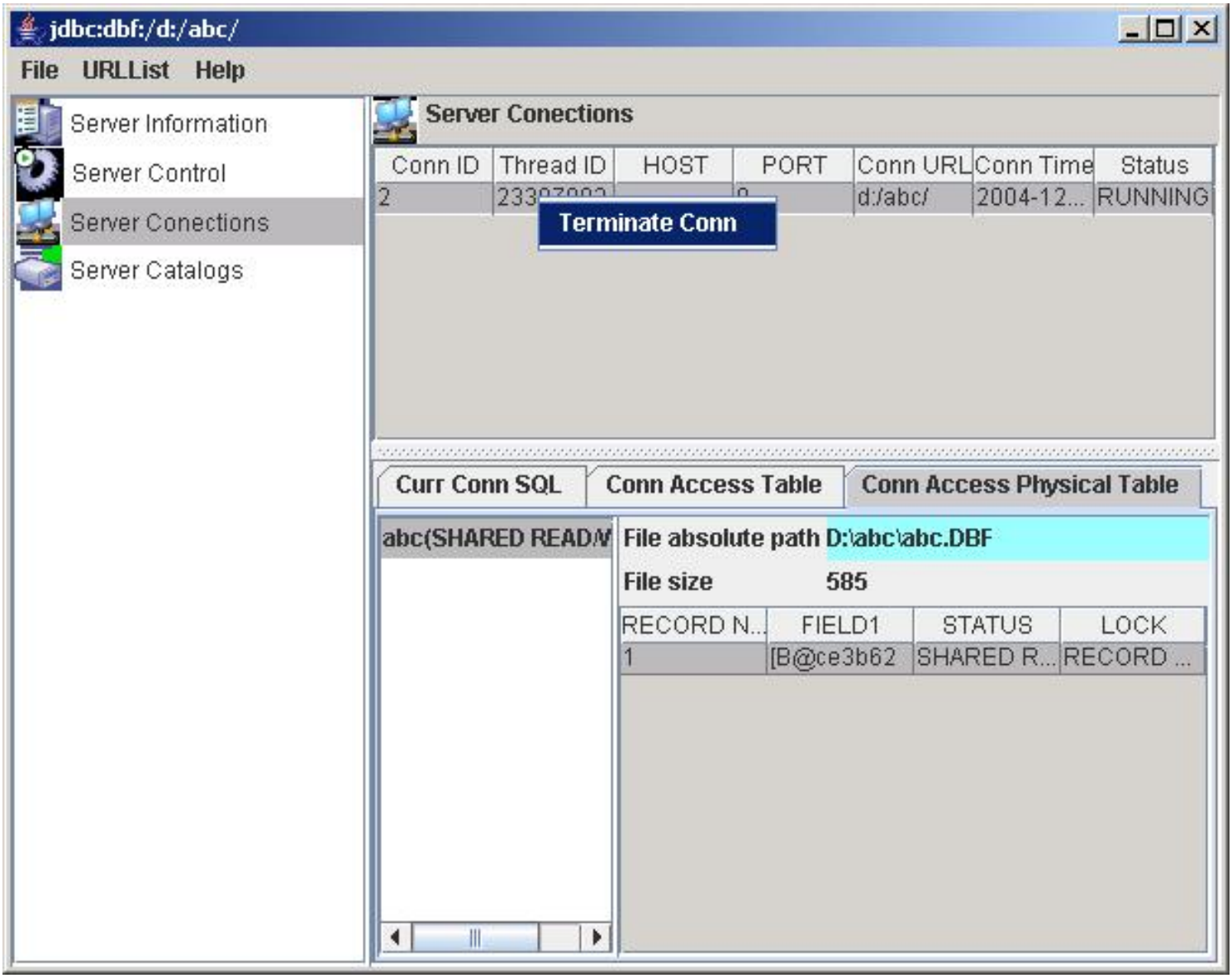


¡¡

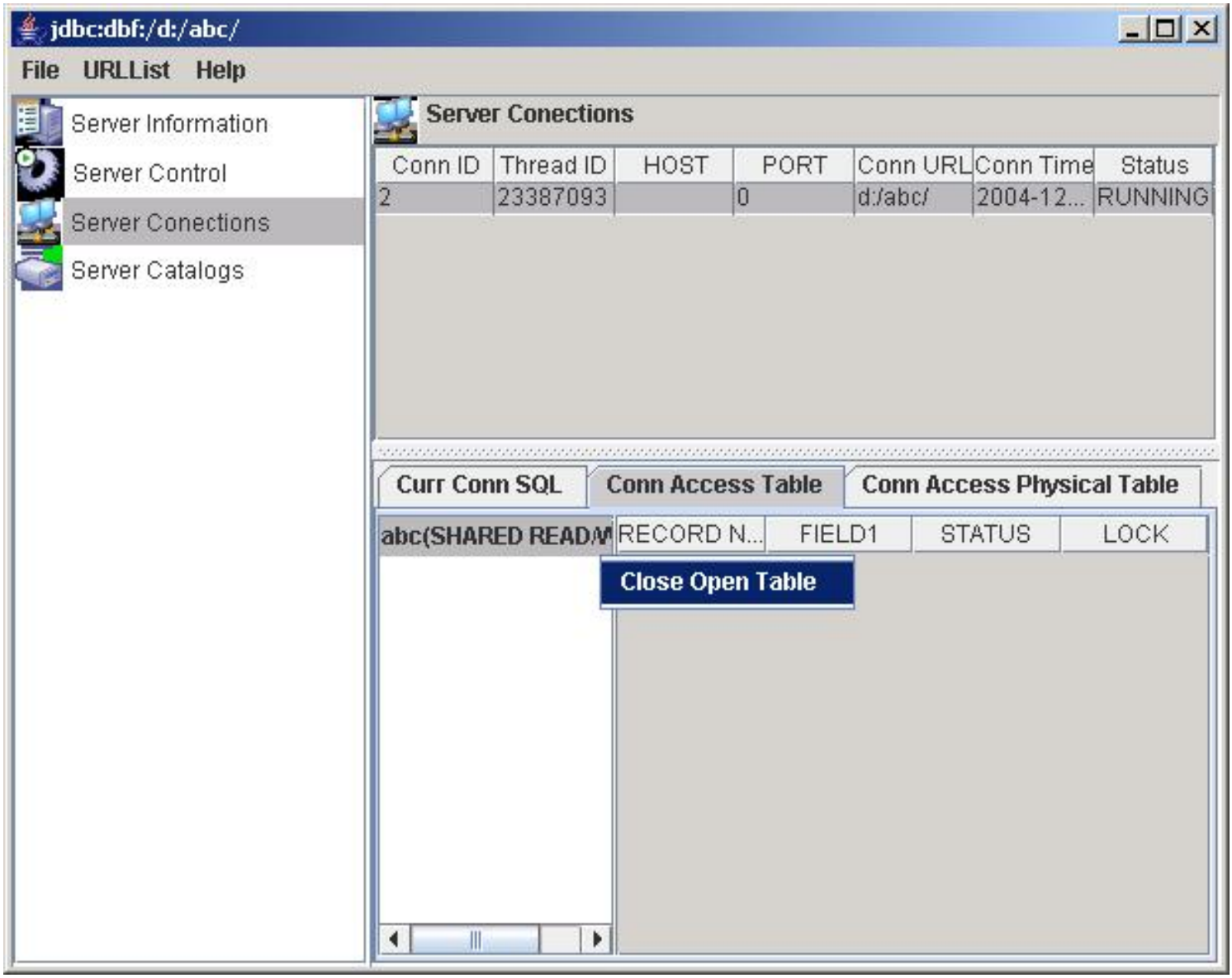
This Server Connections is used to monitor the current connections connected this server and their activity. There are a connection list at the right top window to show the current connections! At the right bottom window,there are three tabpanes! The first is used to show the current sql sentence executed by the selected connection ; this second is used to show the logical table accessed by the selected connection; this third is used to show the physical table accessed by the selected connection;



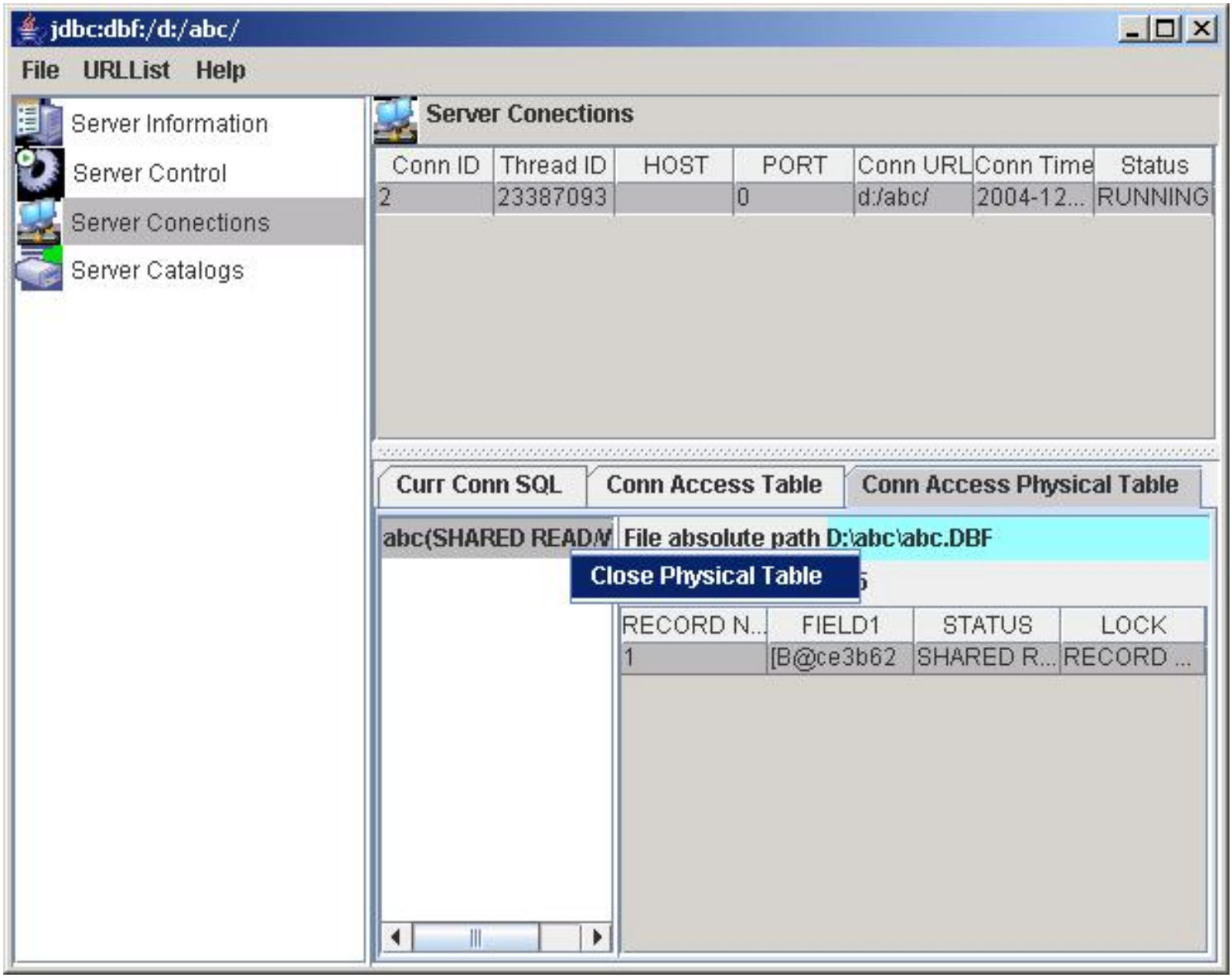
At the connections list,select a connection and right-click mouse ,you can close a connection at the popup menu!



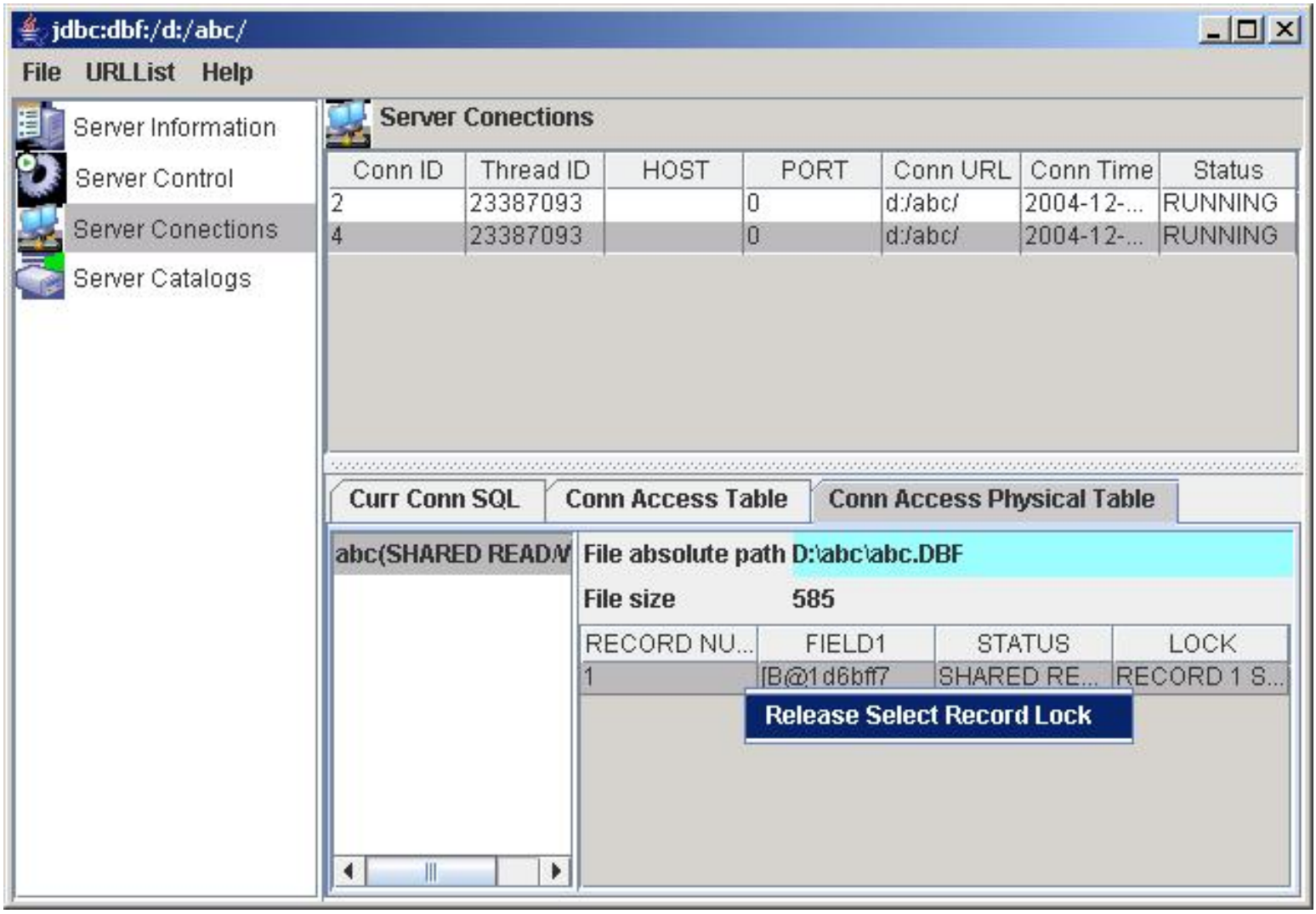
At the accessed logical table list ,select a item and right click mouse,you can close the this logical table opened by this selected connection!



At the accessed physical table list ,select a item and right click mouse,you can close the physical table ,but you should be careful,because this operation will close the physical table no matter this table is accessed by other connection!



It a physical table is locked a lot of records and you can only release some records locks,you should select a item at the physical table record locks list and right click mouse,click the close menu to release the selected record lock,and redo this step to release other record lock until you don't want to do so.



Chapter 9. OpenAPI Programming

Index:

Extend SQL functions

HXTT Access supports more than 210 SQL functions. Please email us if you wish to complement some new SQL functions. HXTT Access supports also user-defined SQL functions, and you should use only this feature to provide special SQL functions in your project.
First, you need to implement com.hxtt.sql.ExtendedFunctionInterface.

```
public interface ExtendedFunctionInterface {
    /**
     * Used to verify whether functionName is supported, and has a correct prarameter
count.
     * @param functionName the name of function
     * @param parameters the parameter list of function,  which can be null
     * @return value
     * @throws SQLException if has an incorrect parameter number
     */
     public boolean isExtendedFunction(String functionName,Object[] parameters)throws
SQLException;

    /**
     * Used to evaluate function value.
     * @param functionName the name of function
     * @param values the value list of function, which can be null
     * @return value
     * @throws SQLException if failed to calculate the function
     */
    public Object evaluate(String functionName,Object[] values)throws SQLException;

    /**
     * Used to get the SQL type of the value that is expected to be returned when
evaluate() is called.
     * @param functionName
```

```
     * @return the SQL type or Types.NULL if functionName is supported
     */
    public int getType(String functionName);

    /**
     * Used to get the SQL types of the parameter values that are expected to be
returned when evaluate() is called.
     * return null if function hasn't any parameter, or you wish to use the default
SQL types.
     * use Types.NULL for that specific parameter if you wish to get the default SQL
type.
     * @param functionName
     * @return the SQL type list or null if functionName is supported
     */
    public int[] getParameterTypes(String functionName);

    /**
     * Used to estimate the maximum number of characters that should be contained in
a String returned by evaluate(String functionName,Object[] values).
     * Zero is returned if this value object does not represent Types.VARCHAR,
Types.BINARY, Types.LONGVARCHAR, or Types.LONGBINARY.
     * @param functionName
     * @return maximum size
     * @throws SQLException  if functionName is supported
     */
    public int estimateValueSize(String functionName) throws SQLException;
}
```

Let us see a sample:

```
import com.hxtt.sql.ExtendedFunctionInterface;
import java.sql.SQLException;
import java.sql.Types;

/**
 * Show how to complement some sql functions.
 * This sample complements tostring(value) and random() for demo purpose
 */
public class Functions implements ExtendedFunctionInterface {
    public Functions() {
```

```
    }

    /**
     * Used to verify whether functionName is supported, and has a correct prarameter
 count.
     * @param functionName the name of function
     * @param parameters the parameter list of function,  which can be null
     * @return value
     * @throws SQLException if has an incorrect parameter number
     */
    public boolean isExtendedFunction(String functionName, Object[] parameters)
 throws SQLException {
        if (functionName.equalsIgnoreCase("tostring")) {
            if (parameters != null && parameters.length == 1) {
                return true;
            }
            else {
                throw new SQLException("Invalid parameter value in tostring
 function");
            }
        }
        else if (functionName.equalsIgnoreCase("random")) {
            if (parameters == null) {
                return true;
            }
            else {
                throw new SQLException("Invalid parameter value in random function");
            }
        }
        return false;
    }

    /**
     * Used to evaluate function value.
     * @param functionName the name of function
     * @param values the value list of function, which can be null
     * @return value
     * @throws SQLException if failed to calculate the function
     */
    public Object evaluate(String functionName, Object[] values) throws SQLException
 {
```

```
        if (functionName.equalsIgnoreCase("tostring")) {
            return values[0] + "";
        }
        else if (functionName.equalsIgnoreCase("random")) {
            return new Double(Math.random());
        }
        throw new SQLException("Inner Error:(");
    }

    /**
     * Used to get the SQL type of the value that is expected to be returned when
evaluate() is called.
     * @param functionName
     * @return the SQL type or Types.NULL if functionName is supported
     */
    public int getType(String functionName) {
        if (functionName.equalsIgnoreCase("tostring")) {
            return Types.VARCHAR;
        }
        else if (functionName.equalsIgnoreCase("random")) {
            return Types.DOUBLE;
        }
        return Types.NULL;

    }

    /**
     * Used to get the SQL types of the parameter values that are expected to be
returned when evaluate() is called.
     * return null if function hasn't any parameter, or you wish to use the default
SQL types.
     * use Types.NULL for that specific parameter if you wish to get the default SQL
type.
     * @param functionName
     * @return the SQL type list or null if functionName is supported
     */
    public int[] getParameterTypes(String functionName) {
        if (functionName.equalsIgnoreCase("tostring")) {
            return new int[] {
                Types.VARCHAR};
        }
```

```
        return null;
    }

    /**
     * Used to estimate the maximum number of characters that should be contained in
a String returned by evaluate(String functionName,Object[] values).
     * Zero is returned if this value object does not represent Types.VARCHAR,
Types.BINARY, Types.LONGVARCHAR, or Types.LONGBINARY.
     * @param functionName
     * @return maximum size
     * @throws SQLException  if functionName is supported
     */
    public int estimateValueSize(String functionName) throws SQLException {
        if (functionName.equalsIgnoreCase("tostring")) {
            return 20;
        }
        else if (functionName.equalsIgnoreCase("random")) {
            return 8;
        }
        return 10;

    }
}
```

Then you can use *com.hxtt.sql.OpenAPI.registerExtendedFunction("Functions");* to regiester Functions class. Then you can use those user-defined functions in SQL. For instance, "select abs(random()),tostring(date) from test;".

Create/Remove/Start/Stop Server Programmatically

If you wish to create,remove,start a server for remote connections from your application, you can call four functions of com.hxtt.sql.admin.Admin class:
public String createServer(String serverConfigName,String serverConfigURL,boolean serverAutoStart,boolean isServerLog,String serverLogFilePath) throws Exception
public void removeServer(String serverName)
public void startServer(String serverName)throws SQLException
public void stopServer(String serverName)throws SQLException

```
For instance:
        try {
```

```
            com.hxtt.sql.admin.Admin admin = new com.hxtt.sql.admin.Admin();

            admin.show();//It can be invisible too.

              String createResult =
admin.createServer("test1","jdbc:access://192.168.1.1:1027/mnt/accessfiles",true,true,"/tmp/test1.log");

            if (createResult!=null)
               System.out.println("Failure to create this server for " +
createResult);

            admin.startServer("test1");

            admin.stopServer("test1");

            admin.stopServer("test4");

            admin.removeServer("test1");

        }
        catch (SQLException e) {
            System.out.println(e.getMessage());
        }
```

Customer Connection

First, let us know the relation of TCP/IP connection and java.sql.Connection. java.sql.Connection objects can share TCP/IP connection. The max number of alive TCP/IP connections between one client and one server is 20, but maybe more than 1000 alive java.sql.Connection objects are using those 20 TCP/IP connections. One java.sql.Connection object maybe build 0, 1, or more than one TCP/IP connections too.
If you haven't read SSL Connection, please read.

To construct your customer connection, you need to implement two interface(com.hxtt.sql.common.SocketLayer and com.hxtt.sql.common.ServerSocketLayer). For SocketLayer, you should have one construction method(public YourSocketLayer(String host, int port)throws IOException). For ServerSocketLayer, you should have one construction method(public YourServerSocketLayer(int port, int backlog, InetAddress bindAddr) throws IOException). Then you can use:
java -Dhxtt.socketclass=yourPackage.YourServerSocketLayer -cp yourClassPath com.hxtt.sql.admin.Admin
Or
java -Dhxtt.socketclass=yourPackage.YourSocketLayer -cp yourClassPath com.hxtt.sql.admin.Admin
Or
java -Dhxtt.socketclass=yourPackage.YourServerSocketLayer -cp yourClassPath yourApplication

Or

java -Dhxtt.socketclass=yourPackage.YourSocketLayer -cp yourClassPath yourApplication

hxtt.socketclass can be used for client connection property too. The class name should be yourPackage.*Socket* and yourPackage.*ServerSocket* so that HXTT AccessServer can guess the other class name according to one class name.

com.hxtt.sql.common.SocketLayer and com.hxtt.sql.common.ServerSocketLayer are pasted below. A simple sample for ip filter, id verfication, and XOR encrypt/decrypt, is showed below too. To keep code neat, there's no remark since you can find all functions in java.net.Socket or java.net.ServerSocket. If you need help, please email us.

```
/******* SocketLayer.java *********/
package com.hxtt.sql.common;

import java.io.IOException;
import java.net.SocketException;
import java.net.InetAddress;

public interface SocketLayer {
    public boolean isClosed();
    public void close() throws IOException;

    public void write(byte b[], int off, int len) throws IOException;
    public void flush() throws IOException;

    public int read(byte b[], int off, int len) throws IOException;

    public int getSoTimeout() throws SocketException;
    public void setSoTimeout(int timeout) throws SocketException;

    public InetAddress getLocalAddress();
    public int getLocalPort();

    public InetAddress getInetAddress();
    public int getPort();
}

/******* ServerSocketLayer.java *********/
package com.hxtt.sql.common;

import java.net.Socket;
import java.io.IOException;
import java.net.SocketException;

public interface ServerSocketLayer{
```

```
    public boolean isClosed();
    public void close() throws IOException;

    public SocketLayer accept() throws IOException;

    public void setSoTimeout(int timeout) throws SocketException;

}

/******* XorSocketLayer.java *********/
package demo;

import java.net.Socket;
import java.io.OutputStream;
import java.io.InputStream;
import java.io.IOException;

import java.net.SocketException;
import java.net.InetAddress;

import com.hxtt.sql.common.SocketLayer;

public class XorSocketLayer implements SocketLayer{
    private Socket socket;
    private InputStream in;
    private OutputStream out;

    public XorSocketLayer(String host, int port)throws IOException {
        Socket socket=new java.net.Socket(host, port);

        //just a check demo
        try{
            check(socket);
        }catch(IOException ioe){
            socket.close();
            throw ioe;
        }

        init(socket);
    }

    private void check(Socket socket)throws IOException{
```

```
            if(socket.getInetAddress().getHostAddress().startsWith("192.168.10")
               || socket.getInetAddress().getHostAddress().startsWith("127.0.0.1")
                ){
                socket.getOutputStream().write("1234".getBytes("ISO8859_1"));
            }else{
                throw new IOException("Prevent logon based upon IP address");
            }
        }

        protected XorSocketLayer(Socket socket)throws IOException {
            init(socket);
        }

        private void init(Socket socket)throws IOException{
            this.socket = socket;
            try{
                in = socket.getInputStream();
                out = socket.getOutputStream();
            }catch(IOException ioe){
                socket.close();
                throw ioe;
             }
        }

        public boolean isClosed() {
            //Valid for JDK1.4.X
            return socket.isClosed();
    //        return false;//For older JDK1.3.X, JDK1.2.X,...
        }

        public void close() throws IOException{
            out = null;
            in = null;
            socket.close();
        }

        public void write(byte b[], int off, int len) throws IOException{
            for(int i=0;i< len;i++){
                out.write( (b[off+i] ^ pattern) & 0xFF);
            }
        }
```

```
    public void flush() throws IOException {
        out.flush();
    }

    private static final byte pattern=(byte)0x21;

    public int read(byte b[], int off, int len) throws IOException {
        int numBytes = in.read(b, off, len);

        if (numBytes <= 0)
            return numBytes;

        for (int i = 0; i < numBytes; i++) {
            b[off + i] = (byte) ( (b[off + i] ^ pattern) & 0xFF);
        }

        return numBytes;
    }

    public int getSoTimeout() throws SocketException{
        return socket.getSoTimeout();
    }

    public void setSoTimeout(int timeout) throws SocketException{
        socket.setSoTimeout(timeout);
    }

    public InetAddress getLocalAddress(){
        return socket.getLocalAddress();
    }

    public int getLocalPort(){
        return socket.getLocalPort();
    }

    public InetAddress getInetAddress(){
        return socket.getInetAddress();
    }

    public int getPort(){
        return socket.getPort();
```

```
    }

}

/******* XorServerSocketLayer.java *********/
package demo;

import java.io.*;
import java.net.*;

import com.hxtt.sql.common.SocketLayer;
import com.hxtt.sql.common.ServerSocketLayer;

public class XorServerSocketLayer implements ServerSocketLayer {
    private ServerSocket serverSocket;

    public XorServerSocketLayer(int port, int backlog, InetAddress bindAddr) throws
IOException {
        this.serverSocket=new ServerSocket(port,  backlog, bindAddr);
    }

    public boolean isClosed(){
        return serverSocket.getLocalPort()<=0;
    }

    public void close() throws IOException{
        serverSocket.close();
    }

    public SocketLayer accept() throws IOException {
        Socket socket=serverSocket.accept();

         //just a check demo
        try{
            check(socket);
        }catch(IOException ioe){
            socket.close();
            throw ioe;
        }

        return new XorSocketLayer(socket);
    }
```

```
    private void check(Socket socket)throws IOException{
        if(socket.getInetAddress().getHostAddress().startsWith("192.168.10")
           || socket.getInetAddress().getHostAddress().startsWith("127.0.0.1")
            ){
            byte[] id=new byte[4];
            int count=socket.getInputStream().read(id);
            if(count!=id.length || !"1234".equals(new String(id))){
                throw new IOException("Prevent logon based upon id");
            }
        }else{
            throw new IOException("Prevent logon based upon IP address");
        }
    }

    public void setSoTimeout(int timeout) throws SocketException {
        serverSocket.setSoTimeout(timeout);
    }
}
```

Chapter 7. Scalar Functions and Aggregate Functions

Index:

Mathematical Functions

1. ABS(x): the absolute value
2. CEIL(x), CEILING(x): the smallest integer that is not less than x
3. DEGREES(x): converts radians to degrees
4. EXP(x): exponential, e(2.718...) raised to the power of x
5. FLOOR(x): the largest integer not greater than argument x
6. INT(x) : truncates x to nearest integer
7. LOG(x), LN(x): the natural logarithm
8. LOG(b,x): returns the logarithm of X for an arbitary base B
9. LOG10(x): the base 10 logarithm
10. LOG2(X): the base 2 logarithm
11. LN(x): the natural logarithm
12. MOD(y, x): the remainder of y/x, you can use y%x too.
13. PI(): pi constant, 3.14159265358979323846.
14. POW(x, y), POWER(x, y): x raised to the power of y
15. RADIANS(x): converts degrees to radians
16. RAND([seed]): a random value between 0.0 and 1.0
17. ROUND(x [,y]): rounds x to nearest integer without y, or round x to y digits after the decimal point.
18. SIGN(x): returns -1 if x is smaller than 0, 0 if x==0 and 1 if x is bigger than 0.
19. SQRT(x): the square root
20. TRUNC(x[,y]), TRUNCATE(x[,y]): truncates x to nearest integer without y, truncates x to y digits after the decimal point

Trigonometric Functions

1. ACOS(x): the inverse cosine of an angle
2. ASIN(x): the inverse sine of an angle

3. ATAN(x),ATN(x): the inverse tangent of an angle
4. ATAN2(x, y): the inverse tangent of x/y
5. COS(x): the cosine of an angle
6. COT(x): the cotangent of an angle
7. SIN(x): the sine of an angle
8. TAN(x): the tangent of an angle

String Functions

1. ALLTRIM(string1): removes all leading and trailing blanks in string1
2. ASC(string1), ASCII(string1): the ASCII code of the leftmost character of the argument
3. AT(cSearchExpression, cExpressionSearched [, nOccurrence]): returns the beginning numeric position of the first occurrence of a character expression or memo field within another character expression or memo field, counting from the leftmost character. If the character expression isn't found, AT( ) returns 0.
4. BIN(number1): returns a string representation of the binary value of number1, where number1 is a integer(TINYINT, SMALLINT, INT, or BIGINT) number. Returns NULL if N is NULL.
5. BIT_LENGTH(string1): the length of the string str in bits
6. CHAR_LENGTH(string1), CHARACTER_LENGTH(string1): the number of characters in string1
7. CHAR(integer), CHR(integer): a character with the given ASCII code
8. CHAR(integer1,...): interprets the arguments as integers and returns a string consisting of the characters given by the unicode values of those integers. NULL values are skipped.
9. CHRTRAN(cSearchedExpression, cSearchExpression, cReplacementExpression): Replaces each character in a character expression that matches a character in a second character expression with the corresponding character in a third character expression. CHRTRAN( ) translates the character expression cSearchedExpression using the translation expressions cSearchExpression and cReplacementExpression and returns the resulting character string. If a character in cSearchExpression is found in cSearchedExpression, the character in cSearchedExpression is replaced by a character from cReplacementExpression that's in the same position in cReplacementExpression as the respective character in cSearchExpression. If cReplacementExpression has fewer characters than cSearchExpression, the additional characters in cSearchExpression are deleted from cSearchedExpression. If cReplacementExpression has more characters than cSearchExpression, the additional characters in cReplacementExpression are ignored.
10. CONCAT(string1, string2): string concatenation, you can use string1+string2 too.
11. CONCAT(string1, string2,...): returns the string that results from concatenating the arguments. NULL values are skipped.
12. CONCAT_WS(separator,string1, string2,...): returns the string that results from concatenating the arguments. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. If the separator is NULL, the result is NULL. The function skips any NULL values after the separator argument.
13. CONV(number1,base): returns a string representation of the first argument in the radix specified by the second argument. The minimum base is 2 and the maximum base is 36.
14. DIFFERENCE(string1, string2): the difference between the sound of string1 and string2
15. HEX(number1): returns a string representation of the hexadecimal value of number1, where number1 is a integer(TINYINT, SMALLINT, INT, or BIGINT) number. Returns NULL if N is NULL.
16. INITCAP(string1): converts first letter of each word (whitespace-separated) to upper case
17. INSERT(string1, start1, length1, string2): a string where length1 number of characters beginning at start1 has been replaced by string2
18. INSTR(string1, string2 [,start1]): the first index (>0:left location, 0:not found) where string2 is found in string1, starting at start1
19. LCASE(string1): converts string1 to lower case
20. LEFT(string1, count1): the leftmost count1 of characters of string1
21. LENGTH(string1), LEN(string1): the number of characters in string1
22. LOCATE(string1, string2 [,start1]): the first index (>0:left location, 0:not found) where string1 is found in string2, starting at start1
23. LOWER(string1): converts string1 to lower case
24. LPAD(string1, length1 [, cPadCharacter]): returns a string from an expression, padded with character(a space by default) to a specified length on the left. If the

string is already longer than length then it is truncated (on the right).

25. LTRIM(string1): removes all leading blanks in string1
26. MID(string1 FROM start1 [FOR length1]), MID(string1, start1 [,length1]): extracts the substring starting at start1 with length length1. MID is a synonym for SUBSTRING.
27. OCT(number1): returns a string representation of the octal value of number1, where number1 is a integer(TINYINT, SMALLINT, INT, or BIGINT) number. Returns NULL if N is NULL.
28. OCTET_LENGTH(string1): the number of octets (8-bit bytes) needed to represent the string1.
29. PADC(string1, length1 [, cPadCharacter]): returns a string from an expression, padded with character(a space by default) to a specified length on both sides. If the string is already longer than length then it is truncated (on the right).
30. PADL(string1, length1 [, cPadCharacter]): returns a string from an expression, padded with character(a space by default) to a specified length on the left. If the string is already longer than length then it is truncated (on the right).
31. PADR(string1, length1 [, cPadCharacter]): returns a string from an expression, padded with character(a space by default) to a specified length on the right. If the string is already longer than length then it is truncated (on the right).
32. POSITION(s1 IN s2), POSITION(substr,str): location of specified substring
33. PROPER(STRING1) : returns from a character expression a string capitalized as appropriate for proper names.
34. REPEAT(string1, count1): repeats string1 count1 times
35. REPLICATE(string1, count1): same as REPEAT(string1,count1)
36. REPLACE(string1, string2, string3): replaces all occurrences in string1 of substring string2 with substring string3.
37. RIGHT(string1, count1): the rightmost count1 of characters of string1
38. RPAD(string1, length1 [, cPadCharacter]): returns a string from an expression, padded with character(a space by default) to a specified length on the right. If the string is already longer than length then it is truncated (on the right).
39. RTRIM(string1): removes all trailing blanks in string1
40. SOUNDEX(string1): a four character code representing the sound of string1
41. SPACE(nSpaces): returns a character string composed of a specified number of spaces.
42. STRCAT(string1, string2): string concatenation, you can use string1+string2 too,same as CONCAT.
43. STRCAT(string1, string2,...): returns the string that results from concatenating the arguments, NULL values are skipped,same as CONCAT.
44. STRCMP(expr1,expr2): returns 0 if the strings are the same, -1 if the first argument is smaller than the second, and 1 otherwise.
45. STRCONV(expr1 [, charsetName]): returns a string by decoding the specified array of bytes using the specified charset. Cp895(Czech MS - DOS 895), Cp620(Polish MS - DOS 620) and Mazovia are extra supported although JVM doesn't support those. The omitted charsetName is 'ISO8859_1'.
46. STRTRAN(cSearched, cSearchFor [, cReplacement][, nStartOccurrence] [, nNumberOfOccurrences]): searches a character expression or memo field for occurrences of a second character expression or memo field, and then replaces each occurrence with a third character expression or memo field.
47. STUFF(cExpression, nStartReplacement, nCharactersReplaced, cReplacement): returns a string created by replacing a specified number of characters in a character expression with another character expression. cExpression specifies the string expression in which the replacement occurs. nStartReplacement specifies the position in cExpression where the replacement begins. nCharactersReplaced specifies the number of characters to be replaced. If nCharactersReplaced is 0, the replacement string cReplacement is inserted into cExpression. cReplacement specifies the replacement string expression. If cReplacement is the empty string, the number of characters specified by nCharactersReplaced are removed from cExpression.
48. SUBSTR(string1, start1 [,length1]): extracts the substring starting at start1 with length length1
49. SUBSTRING(string1 FROM start1 [FOR length1]), SUBSTRING(string1, start1 [,length1]): extracts the substring starting at start1 with length length1
50. TRANSLATE(string1, string2, string3): any character in string1 that matches a character in the string2 is replaced by the corresponding character in the string3.
51. TRIM([[BOTH | LEADING | TRAILING] [removedstring1] FROM] string1): remove the removedstring1 (a space by default) from the start/end/both ends of the string1.
52. UCASE(string1): converts string1 to upper case
53. UPPER(string1): converts string1 to upper case
54. CHARMIRR(string1 [,lDontMirrorSpaces]): mirrors string1 at character level. string1 is the string that should be mirrored. If lDontMirrorSpaces equal to true, spaces at the end of string1 will not be mirrored but kept at the end. lDontMirrorSpaces's default value is false, which means to mirror the whole string.
55. REVERSE(string1[,lDontMirrorSpaces]): mirrors string1 at byte level.

Date/Time Functions

1. ADDTIME(expr,expr2): adds expr2 to expr and returns the result. expr is a date or timestamp expression, and expr2 is a time expression.
2. CDOW(date) Returns the day-of-the-week(Sunday,Monday, Tuesday, Wednesday, Thursday, Friday,Saturday) from a given date,
3. CMONTH(date) the name of the month
4. CURDATE(): the current date
5. CURTIME(): the current time
6. DATE(): the current date
7. DATE(expr): extracts the date part of the date or timestamp expression expr.
8. DATEDIFF(expr,expr2): returns the number of days between the start date expr and the end date expr2. expr and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.
9. DATETIME(): the current timestamp
10. DATESERIAL(year,month,day): returns a date value representing a specified year, month, and day.
11. DATE_ADD(date,INTERVAL expr type), DATE_SUB(date,INTERVAL expr type), ADDDATE(date,INTERVAL expr type), SUBDATE(date,INTERVAL expr type). For instance, SELECT DATE_ADD(date1,INTERVAL hour(now())+1 HOUR), adddate(date1,interval 3 hour) FROM test;

type Value	Expected expr Format
MICROSECOND[S]	MICROSECONDS
MILLISECOND[S]	MILLISECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
DECADE	DECADES
CENTURY	CENTURYS
MILLENNIUM	MILLENNIUMS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'

DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

12. DAY(date1), DAYOFMONTH(date1): the day of the month (1-31)
13. DAYNAME(date1): the name of the day
14. DAYOFWEEK(date1): the day of the week (1 means Sunday)
15. DAYOFYEAR(date1): the day of the year (1-366)
16. EXTRACT(type FROM expr): extracts parts from the date.

type Value	Expected Result
MICROSECOND[S]	MILLISECOND*1000
MILLISECOND[S]	indicats the millisecond within the second.
SECOND	indicats the second within the minute
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
DECADE	DECADES
CENTURY	CENTURYS
MILLENNIUM	MILLENNIUMS
DOW	indicates the day of the week, SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY(1~7).
DOY	indicates the day number within the year. The first day of the year has value 1.
WEEK,WOM	indicats the ordinal number of the day of the week within the current month.
WOY	indicats the ordinal number of the day of the week within the current year.
EPOCH	the current time as UTC milliseconds from the epoch(1970-01-01 00:00:00).

17. DOW(date1) get the day of the week, SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY(1~7)
18. FROM_DAYS(expr1): given a day number expr1, returns a DATE value.
19. GOMONTH(expr1,numMonths) : give a date,return the date before or after a number months
20. HOUR(time1): the hour (0-23)

21. LAST_DAY(date1): takes a date or timestamp value and returns the corresponding date for the last day of the month.
22. MINUTE(time1): the minute (0-59)
23. MILLISECOND(time1): the milliseconds from the time or timestamp expression time1.
24. MICROSECOND(time1): the microseconds from the time or timestamp expression time1.
25. MONTH(time1): the month (1-12)
26. MONTHNAME(date1): the name of the month
27. NOW(): the current date and time as a timestamp
28. QUARTER(date1): the quarter (1-4)
29. SECOND(time1): the second (0-59)
30. SUBTIME(expr,expr2): subtracts expr2 from expr and returns the result. expr is a date or timestamp expression, and expr2 is a time expression.
31. SYSDATE(): the current date and time as a timestamp. Asynonym for NOW().
32. TIME(): returns the current system time in 24-hour, eight-character string (hh:mm:ss) format.
33. TIME(expr): extracts the time part of the time or timestamp expression expr.
34. TIMEDIFF(expr,expr2) returns the time between the start time expr and the end time expr2. Only the time parts of the values are used in the calculation.
35. TIMESERIAL(hour,minute,second): returns a Time value representing a specified hour, minute, and second.
36. TIMESTAMP(expr): returns the date or timestamp expression expr as a timestamp value.
37. TIMESTAMPADD(interval, count, timestamp1): adds the integer expression count to the date or timestamp expression timestamp1. interval can be SQL_TSI_FRAC_SECOND, SQL_TSI_SECOND, SQL_TSI_MINUTE, SQL_TSI_HOUR, SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, SQL_TSI_YEAR, FRAC_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.
38. TIMESTAMPDIFF(interval, timestamp1, timpestamp2): returns the integer difference between the date or timestamp expressions timestamp1 and timpestamp2 (timpestamp2-timestamp1). interval can be SQL_TSI_FRAC_SECOND, SQL_TSI_SECOND, SQL_TSI_MINUTE, SQL_TSI_HOUR, SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, SQL_TSI_YEAR, FRAC_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.
39. TO_DAYS(date1): given a date date1, returns a day number.
40. WEEK(date1), WEEKOFYEAR(date1): the week of this year (1-53)
41. YEAR(date1): the year

Boolean Functions

1. BETWEEN(expression1,expression2,expression3) : determines whether the value of an expression1 lies between the expression2 and expression3, return true or false.
2. EMPTY(expression): determines whether an expression evaluates to empty or null. The expression you include can be a string, numeric, date, or logical expression. EMPTY() returns true, when a string is empty string, spaces, tabs, carriage returns, linefeeds, or any combination of these, numeric value equals to 0, and logical expression is false.
3. ISBLANK(expression): determines whether an expression evaluates to empty or null. The expression you include can be a string, numeric, date, or logical expression. ISBLANK() returns true, when a string is empty string or spaces, numeric value equals to null, and logical expression is null.
4. ISALPHA(expression): determines whether the leftmost character in a character expression is alphabetic.
5. ISDIGIT(expression): determines whether the leftmost character of the specified character expression is a digit (0 through 9).
6. ISDIGITS(expression): determines whether a string contains only digits(0 through 9).
7. ISNULL(expression): determines whether an expression evaluates to null. The expression you include can be a string, numeric, date, or logical expression. If expression is NULL, ISNULL() returns true, otherwise it returns false.

System Functions

1. DATABASE(): the name of the database of this connection

2. USER(): the user name of this connection
3. DELETED([cTableAlias | nWorkArea]): returns a logical value that indicates whether the current record is marked for deletion.
4. RECCOUNT([nWorkArea | cTableAlias]): returns the number of records, which includes all deleted records.
5. RECNO([nWorkArea | cTableAlias]): returns the current record number in the current or specified table. nWorkArea specifies the work area number for a table open in another work area. cTableAlias specifies the table alias for a table open in another work area.
6. ROWLOCKED([nWorkArea | cTableAlias]): indicates whether the current row has been locked by process or application.
7. TABLELOCKED(cTableName): indicates whether a table has been locked by process or application. For instance, select tablelocked('test').
8. CURRVAL(cTableName, cColumnName): returns the last generated IDENTITY(auto_increment) value for a particuar table. If that table hasn't IDENTITY(auto_increment) column, it will return silently null value. Only when there's more than one auto_increment column in a table, it will check cColumnName's validity.

Conversion Functions

1. CAST(expression AS SQLtype1): converts value1 to another data type SQLtype1. SQLtype1 may be SQL_BIGINT(Types.BIGINT), SQL_BINARY(Types.BINARY), SQL_BIT(Types.BIT), SQL_CHAR(Types.CHAR), SQL_DATE(Types.DATE), SQL_DECIMAL(Types.DECIMAL), SQL_DOUBLE(Types.DOUBLE), SQL_FLOAT(Types.FLOAT), SQL_INTEGER(Types.INTEGER), SQL_LONGVARBINARY(Types.LONGVARBINARY), SQL_LONGVARCHAR(Types.LONGVARCHAR), SQL_REAL(Types.REAL), SQL_SMALLINT(Types.SMALLINT), SQL_TIME(Types.TIME), SQL_TIMESTAMP(Types.TIMESTAMP), SQL_TINYINT(Types.TINYINT), SQL_VARBINARY(Types.VARBINARY), SQL_VARCHAR(Types.VARCHAR), BIGINT(Types.BIGINT), BINARY(Types.BINARY), BIT(Types.BIT), CHAR(Types.CHAR), DATE(Types.DATE), DECIMAL(Types.DECIMAL), DOUBLE(Types.DOUBLE), FLOAT(Types.FLOAT), INTEGER(Types.INTEGER), LONGVARBINARY(Types.LONGVARBINARY), LONGVARCHAR(Types.LONGVARCHAR), REAL(Types.REAL), SMALLINT(Types.SMALLINT), TIME(Types.TIME), TIMESTAMP(Types.TIMESTAMP), TINYINT(Types.TINYINT), VARBINARY(Types.VARBINARY), and VARCHAR(Types.VARCHAR). For instance, cast('456' AS SQL_INTEGER),cast('123.456' AS SQL_DECIMAL), and cast('2004-12-23' as sql_date).
2. CONVERT(value1, SQLtype1): converts value1 to another data type SQLtype1. SQLtype1 may be SQL_BIGINT(Types.BIGINT), SQL_BINARY(Types.BINARY), SQL_BIT(Types.BIT), SQL_CHAR(Types.CHAR), SQL_DATE(Types.DATE), SQL_DECIMAL(Types.DECIMAL), SQL_DOUBLE(Types.DOUBLE), SQL_FLOAT(Types.FLOAT), SQL_INTEGER(Types.INTEGER), SQL_LONGVARBINARY(Types.LONGVARBINARY), SQL_LONGVARCHAR(Types.LONGVARCHAR), SQL_REAL(Types.REAL), SQL_SMALLINT(Types.SMALLINT), SQL_TIME(Types.TIME), SQL_TIMESTAMP(Types.TIMESTAMP), SQL_TINYINT(Types.TINYINT), SQL_VARBINARY(Types.VARBINARY), and SQL_VARCHAR(Types.VARCHAR). value1 may be any complicated expression. For instance, CONVERT("123",SQL_INTEGER).
3. CBOOL(expression): returns a Boolean value from an expression.
4. CBYTE(expression): returns a Byte value from an expression.
5. CCUR(expression): returns a Currency value with four decimal digits of precision to the right of the decimal from an expression.
6. CDATE(expression,pattern): returns a Date value according a pattern from an expression. For instance, CDATE('21111947','ddMMyyyy').
7. CDBL(expression): returns a Double value from an expression.
8. CINT(expression): returns an Integer value from an expression.
9. CLNG(expression): returns a Long value from an expression.
10. CSNG(expression): returns a Float value from an expression.
11. CSTR(expression): returns a String value from an expression.
12. CTOD(cExpression): converts a string expression to a date expression.
13. CTOT(cExpression): returns a timestamp value from a string expression.
14. DTOC(date1 | timestamp1): returns a string from a date or timestamp expression.
15. DTOT(dDateExpression): returns a timestamp value from a date expression.
16. DTOS(date1 | timestamp1): returns a string in a yyyymmdd format from a specified date or timestamp expression.
17. DTOS(date1 | timestamp1,pattern): returns a string according to a pattern format from a specified date or timestamp expression.
18. TTOC(tExpression [, 1 | 2]): converts a timestamp expression to a string value of a specified format.
19. TTOD(tExpression): returns a date value from a timestamp expression.

20. POSIXTOT(expression): returns a timestamp value from a POSIX timestamp value.
21. TTOPOSIX(tExpression): converts a timestamp expression to a POSIX timestamp value.
22. STR(nExpression [, nLength [, nDecimalPlaces]]): Returns the character equivalent of a specified numeric expression. nExpression specifies the numeric expression STR( ) evaluates. nLength specifies the length of the character string STR( ) returns. The length includes one character for the decimal point and one character for each digit to the right of the decimal point. nDecimalPlaces specifies the number of decimal places in the character string STR( ) returns. If you specify fewer decimal places than are in nExpression, the extra digits are truncated. STR( ) pads the character string it returns with leading spaces if you specify a length larger than the number of digits to the left of the decimal point. STR( ) returns a string of asterisks, indicating numeric overflow, if you specify a length less than the number of digits to the left of the decimal point.
23. STRZERO(nExpression, nLength[, nDecimals]): convert a numeric expression to a string padded with leading zeros.
24. VAL(string1): returns a numeric value from a string1 composed of numbers.
25. COLLATE(string1[,collation]): For multilingual sort in ORDER BY clause. Now collation can be 'DUTCH', 'GERMAN', 'ICELAND', 'SPANISH', 'RUSSIAN', 'CZECH', 'GREEK', 'SLOVAK', 'POLISH', 'TURKISH', 'HUNGARY', CP850, CP852, CP866, CROATIAN, HEBREW, SWEDISH, and 'MAZOVIA'. Without collation parameter, COLLATE function will try to utilize charSet property in Connection properties.
26. PasToJava(str): get a Java string from a Pascal-style string
27. JavaToPas(str): get a Pascal-style string from a Java string
28. PasToJava(str): get a null-terminated string from a Pascal-style string
29. CToPas(str): get a Pascal-style string from a null-terminated string
30. CToJava(str): get a Java string from a null-terminated string
31. JavaToC(str): get a null-terminated from a Java string
32. BToInt_LE(binary): get int value from bytes with little-endian.
33. BToInt_BE(binary): get int value from bytes with big-endian.
34. IntToB_LE(binary): get bytes with little-endian from int value.
35. IntToB_BE(binary): get bytes with big-endian from int value.
36. BToShort_LE(binary): get short value from bytes with little-endian.
37. BToShort_BE(binary): get short value from bytes with big-endian.
38. ShortToB_LE(binary): get bytes with little-endian from short value.
39. ShortToB_BE(binary): get bytes with big-endian from short value.
40. BToLong_LE(binary): get long value from bytes with little-endian.
41. BToLong_BE(binary): get long value from bytes with big-endian.
42. LongToB_LE(binary): get bytes with little-endian from long value.
43. LongToB_BE(binary): get bytes with big-endian from long value.
44. GetNumber(str[, defaultValue]): return a number value(int, long, double) according to str. If failed to parse, return defaultValue(null is omitted value).
45. GetInt(str[, defaultValue]): return an int value according to str. If failed to parse, return defaultValue(null is omitted value).
46. GetLong(str[, defaultValue]): return a long value according to str. If failed to parse, return defaultValue(null is omitted value).
47. GetDouble(str[, defaultValue]): return a double value according to str. If failed to parse, return defaultValue(null is omitted value).

Security Functions

1. COMPRESS(content) : Return a compressed byte[]
2. UNCOMPRESS(compressedBytes) : Return an uncompressed byte[],please don't use it for non-compressed data
3. ENCRYPT(content,cKey,cCryptMethod): Returns a crypted byte[]. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH' now. ENCRYPT function is used for VARBINARY column. Data Encryption Standard (DES) algorithm, adopted by the U.S. government in 1977, is a block cipher that transforms 64-bit data blocks under a 56-bit secret key, by means of permutation and substitution. It is officially described in FIPS PUB 46. The DES algorithm is used for many applications within the government and in the private sector. Triple-DES is an improvement over DES. It uses three DES keys k1, k2 and k3. A message is encrypted with k1 first, then decrypted with k2 and encrypted again with k3 (DESencryptiondecryptionencryption). This increases security as the key length effectively increases from 56 to 112 or 168 (two or three keys may be used in TriDES). The DES key size is 128 or 192 bit and block size 64 bit.

4. DECRYPT(content,cKey,cCryptMethod): Returns a decrypted byte[]. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH' now.
5. ENCODE(content): Encodes a BASE64 encoding string.
6. DECODE(content): Returns a byte[] from a BASE64 string.
7. ENCODE(content,cKey,cCryptMethod): Crypts and encodes content. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH'. ENCRYPT function is used for VARCHAR column.
8. DECODE(content,cKey,cCryptMethod): Decodes and decrypts content. cCryptMethod should be 'DES', 'TRIDES', or 'BLOWFISH' now.
9. MD5(string1): Calculates a MD5(Message-Digest Algorithm 5) checksum for the string1.
10. SHA1(string1): Calculates a SHA-1(Secure Hash Algorithm 1) hash for the string1.
11. Crypt3(word[, salt]): Returns a hashed string of 13 printable ASCII characters, with the first two characters represent the salt. It can be used to accept typed passwords from the user, or attempting to crack Unix passwords with a dictionary.

Sequence Functions

1. NEXTVAL(cSequenceName): advances sequence and returns new value.
2. CURRVAL(cSequenceName): returns value most recently obtained with nextval.

Miscellaneous Functions

Function	Argument Type	Return Type	Description
GREATEST(expression1,expression2[,...]) MAX(expression1,expression2[,...])	any numeric, string, date/time, or boolean type	same as argument type	maximum value of all expressions
LEAST(expression1,expression2[,...]) MIN(expression1,expression2[,...])	any numeric, string, date/time, or boolean type	same as argument type	minimum value of all expressions
IF(lExpression, eExpression1, eExpression2) IIF(lExpression, eExpression1, eExpression2)	lExpression specifies the logical expression that IF()/IIF( ) evaluates.	Returns one of two values depending on the value of a logical expression.	If lExpression evaluates to true , eExpression1 is returned. If lExpression evaluates to false, eExpression2 is returned.
NVL(expression, value) IFNULL(expression, value)	any numeric, string, date/time, or boolean type	Returns one of two values depending on whether expression is null.	If expression evaluates to null , value is returned. Otherwise, expression is returned.

INLIST(eExpression1, eExpression2 [, eExpression3 ...])	eExpression1 specifies the expression INLIST( ) searches for in the set of expressions. eExpression2 [, eExpression3 ...] specifies the set of expressions to search. You must include at least one expression (eExpression2), and can include up to 24 expressions (eExpression2, eExpression3, and so on).	Determines whether an expression matches another expression in a set of expressions.	All the expressions in the set of expressions must be of the same data type.
COALESCE(value [, ...])	any numeric, string, date/time, or boolean type	the type of the first of its arguments that is not null	returns the first of its arguments that is not null
ELT(numberExpression,value1Expression,[value2Expression,...])	numberExpression must be a integer type,value expression can be any type	Returns value depending on the numberExpression,value1Expression,...valuexExpression	Returns value1Expression if numberExpression = 1, value2Expression if numberExpression = 2, and so on. Returns NULL if N is less than 1 or greater than the number of arguments.
INTERVAL(expression,expr1,expr2,...,exprn)	any numeric, string, date/time, or boolean type	integer value	returns 0 if expression< expr1, 1 if expression< expr2 and so on or -1 if expressionN is NULL. If expression>exprn, returns n.

<table>
<tr>
<td>TRANSFER (expression, search_1, result_1) TRANSFER (expression, search_1, result_1, search_2, result_2) TRANSFER (expression, search_1, result_1, search_2, result_2, ..., search_n, result_n) TRANSFER (expression, search_1, result_1, default) TRANSFER (expression, search_1, result_1, search_2, result_2, default) TRANSFER (expression, search_1, result_1, search_2, result_2, ..., search_n, result_n, default)</td>
<td>any numeric, string, date/time, or boolean type,or null</td>
<td>Returns value depending on the expression,search_x,result_x and default</td>
<td>TRANSFER compares expression to the search_x expressions and, if matches, returns result_x. If not, returns default, or, if default is left out, return null .</td>
</tr>
<tr>
<td></td>
<td>expression specifies the character, currency, date, or numeric expression to format.
formatcode specifies one format code that determine how the expression is formatted. The following table lists the available format codes
<table>
<tr><th>Format Code</th><th>Description</th></tr>
<tr><td>@C</td><td>CR is appended to positive currency or numeric values to indicate a credit.</td></tr>
<tr><td>@D</td><td>act as DTOS function.</td></tr>
<tr><td>@E</td><td>act as DTOS function.</td></tr>
<tr><td>@T</td><td>leading and trailing spaces are trimmed from character values.</td></tr>
</table>
</td>
<td></td>
<td></td>
</tr>
</table>

TRANSFORM(expression, formatcodes)	@X	db is appended to negative currency or numeric values to indicate a debit.	return the formatted string	returns a character string from an expression in a format determined by a format code	
	@Z	if 0, currency or numeric values are converted to spaces.			
	@(	encloses negative currency or numeric values in parentheses.			
	@^	converts currency or numeric values to scientific notation.			
	@0	converts numeric or currency values to their hexadecimal equivalents. The numeric or currency value must be positive and less than 4,294,967,296.			
	!	converts a character to uppercase.			

$	adds the current currency symbol specified by SET CURRENCY to currency and numeric values. By default, the symbol is placed immediately before or after the value. However, the currency symbol and its placement (specified with SET CURRENCY), the separator character (specified with SET SEPARATOR) and the decimal character (specified with SET POINT) can all be changed.
X	specifies the width of character values. For example, if cFormatCodes is XX? 2 characters are returned.

		Y	converts logical true (.T.) and false (.F.) values to Y and N, respectively.	
		@!	converts a string to uppercase.	

Aggregate Functions

1. FIRST(expression): the value of a specified field in the first record, respectively, of the result set returned by a query. Because records are usually returned in no particular order (unless the query includes an ORDER BY clause), the records returned by this functions will be arbitrary.
2. LAST(expression): the value of a specified field in the last record, respectively, of the result set returned by a query. Because records are usually returned in no particular order (unless the query includes an ORDER BY clause), the records returned by this functions will be arbitrary.
3. AVG(expression): the average (arithmetic mean) of all input values.
4. COUNT(*): the number of input values.
5. COUNT(expression): the number of input values for which the value of expression is not null.
6. MAX(expression): the maximum value of expression across all input values.
7. MIN(expression): the minimum value of expression across all input values.
8. STD(expression): the sample standard deviation of the input values.
9. STDDEV(expression): the sample standard deviation of the input values.
10. SUM(expression): the sum of expression across all input values.
11. GROUP_CONCAT([DISTINCT] expr_list [order_by_clause] [SEPARATOR str_val]): returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values. SEPARATOR is followed by the string value that should be inserted between values of result. The default is a comma (','). You can eliminate the separator altogether by specifying SEPARATOR ''. The result will be truncated to the maximum length of 2048 sometimes.

FAQ for HXTT Access Packages of type 4 JDBC Driver for Microsoft Access (MS Access) 95/97/2000/XP/2002/2003

The most recent version of this document can be viewed at here.

Table of Contents

General Questions

1. How to know the detailed version information of HXTT Access package?
1st way: "java com.hxtt.sql.access.AccessDriver" will print that information.
2nd way: check that MANIFEST.MF file in jar file.

2. Can I use it in an iSeries OS/400 IBM machine that has Java 1.4 running in it?
The HXTT Access packages can run on any platform with Java VM, which includes Microsoft Windows, Novell Netware, OS2, UNIX, and LINUX. It supports Personal Java, JDK1.0.X, JDK1.1.X, JDK1.2.X, JDK1.3.X, JDK1.4.X and JDK1.5.X. It supports JDBC1.2, JDBC2.0, and JDBC3.0 now.

3. What is difference between the HXTT Access Package, Embedded Package, and Remote Access Package? Can I get some sample code to use the HXTT Access?
The HXTT Access supports Embedded and Remote Access. HXTT Access Package includes a Database GUI manager. If you're accessing the local data, you can use the HXTT Access Package or Embedded Package. If you're accessing the remote data, you can use the HXTT Access Package or Remote Access Package. There is no any difference for your code to use anyone of three packages. Please download the demo package from here.

4. What causes the 'No suitable driver' SQLException?
This error usually occurs during a call to DriverManager.getConnection(). The cause can be failing to load the appropriate JDBC driver before calling getConnection(), or specifying an invalid JDBC URL that isn't recognized by your JDBC driver. If you're using a trial version, you will get "No suitable driver" SQLException, and "Evaluation period over" after using about 30 days. The HXTT Access driver's name is com.hxtt.sql.access.AccessDriver, and its JDBC URL:

```
        Embedded:
                jdbc:access:[//]/[DatabasePath][?prop1=value1[;prop2=value2]] (You
can omit that "//" characters sometimes)
                        For example:
                                "jdbc:access:/."
                                "jdbc:access:/c:/data"
                                "jdbc:access:////usr/data" for unix or linux:
                                "jdbc:access:/./data"
```

```
        Access by Access Server: Skip it if you don't use TCP, RMI or JINI.
                jdbc:access://host:port/[DatabasePath]
                        For example: "jdbc:access://domain.com:3099/c:/data" if one
AccessServer is run on the 3099 port of domain.com
```

5. How to setup Access url on the Novell Server?
Access driver can run on Novell server. You can use directly access or AccessServer to visit your data on Novell server. If your Access files is at sys:/java/yourdata, the direct URL should be:
jdbc:access:///sys:/java/yourdata
or
jdbc:access:////java/yourdata

6. I got "java.io.IOException: Permission denied" sometimes for my SELECT query.
Please figure out what directory Java's java.io.tmpdir system property points to, and make sure that directory is writable by the user that runs your Java applications, otherwise you should set tmpdir properity in Connection properity to a writable directory. tmpdir properity indicates whether set a temp directory, Default: the value of JVM's "java.io.tmpdir" property. If that value is incorrect, uing the directory of JDBC url.

7. When I used jdbc:access:/<DatabasePath>, the connection's schema was empty. "create catalog if not exists accessfiles". What is Catalog?
Access's schema is always empty. You can use catalog to query subdirectory. Catalog means a directory, which contains some Access files.

8. Can HXTT Access support JDK 1.0.2?
Yeah. You need to download JDBC 1.22 from the Sun's JDBC download page and add JDBC1.22 into JDK 1.0.2. HXTT Access hasn't be tested on JDK1.0.X since we have not received such a complement request from our users. If you meet any problem, please let us know.

Applet Questions

1. I already configured the .java.policy for my applet, but I continue with problems of "acess denied".
For instance, you're using "jdbc:access:/C:/test", and grant codeBase "file:/C:/test" in your policy file, but your applet is running from "D:\sample\CargaStatApplet.html". You should grant codeBase "file:/D:/sample", not "file:/c:/test".

2. http://localhost:8080/jdbcapplet.html, the applet started but returns a Classnotfound com.hxtt.sql.access.AccessDriver error in the gui list.
Please add a codebase tag. For instance, "<applet code="jdbcapplet.class" codebase="Access_Remote_Access_JDBC20.jar"></applet>". The Access_Remote_Access_JDBC20.jar should be at the same directory of jdbcapplet.html.

Remote Access Questions and Client/Server Mode Questions

1. Client/Server mode question: The data directory is not in the IBM machine where the Java program should run, but instead those Access files are in another machine with Windows operating system.
com.hxtt.sql.admin.Admin provides a GUI manger for Access server. For instance, you wish to provide JDBC3.0 remote data access. Please use "java -cp yourdirectrory/Access_JDBC30.jar com.hxtt.sql.admin.Admin" to start GUI manager, and add a url setting of "'jdbc:access://10.32.90.48:" + 8029 +"/"+databaseDirectory' on your host of 10.32.90.48(just an IP sample), then click Start button. Third, you can use 'String url =
"jdbc:access://10.32.90.48:" + 8029 +"/"+databaseDirectory;' to visit your Access database from your IBM machine. If you're running that GUI manager on "yourNT.com" host to visit "c:/database" directory, you can use "jdbc:access://yourNT.com:8029/c:/database" on your web application.

jdbc:access://yourNT.com:8029/c:/database?user=oneuser&password=onepassword can provide a simply user/password verification for client/server mode. If you wish to write a secure Access server for some sensitive information, embedded encrypt/decrypt functions can help you.

2. Remote access through map network drive question: How to remote access Access data without AccessServer?
You can share your remote directory which contains your data files, then map it to a local driver.
For Windows: You can connect remote Access database by sharing the directory and map it to local drive. You should disable the OPLOCKS of your Samba/NT/2000 server. This is done by manipulating the following registry key:
\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters
EnableOplocks REG_DWORD 0 or 1
Default: 1 (true)
For Linux: You can use mounting. One user uses Samba to maped NTFS partitions in Linux servers, and Access driver works normally like mapping any mount point in Linux.
For Novell: You can map NCP directory as driver or mount NCP directory.

3. Remote access through SAMBA protocol question: How to let my servlet on Linux to access over 300 hundred shared folders that all are on Windows boxes

You need to use SAMBA table, which needn't to map or mount driver.

4. Remote access through http/https/ftp protocol question: How to let my program to fetch data daily from our web host?

You need to use url database, which supports http protocol, https protocol, and ftp protocol.

5. Remote access through UNC path question: Can I setup only one datasource to access four servers for my Cold Fusion?
To access one unc path, you can use jdbc:access:/\\PC17\c$\values or jdbc:access:/\\PC17\val.
To access four unc pathes in the same connection, you need to use a free JDBC url, "jdbc:access:/" or "jdbc:access:///". Then you can use some full UNC path names in SQL to visit your four servers where your Java VM has right to access.. For instance:

```
select * from \\amd2500\e$\accessfiles\test;
select * from "\\amd2500\d$\accessiles".test;
select * from ".".test;
```

6. I can't get the com.hxtt.sql.admin.Admin runnig for internet --> intranet
HXTT Access supports port mapping and NAT route. Let HXTT Access listening a port on the database server, and modify your route table or NAT table to map an external port to that internal port. You can use "start java -Djava.security.policy=policy com.hxtt.sql.admin.Admin" to start GUI manager. You should add a remote url, for instance, jdbc:access://localhost:8029/d:/dbffiles, and click Start button to start that server. Then on your internet client side, you can use jdbc:access://externalIP:8029/d:/dbffiles to access your intranet host. externalIP means an external IP or domain name address of your gateway or database server.
BTW, except for TCPServer protocol, HXTT Access can use also RMIServer protocol. For instance, you have used "start rmiregistry 1099 -J-Djava.security.policy=yourPolicyFile" to startup your rmi service. Then you can use jdbc:access://localhost:1099/d:/dbffiles?serverType=RMIServer to let HXTT Access bind remote service in registry. The key is use "java -Djava.security.policy=policy -Djava.rmi.server.hostname=externalIP com.hxtt.sql.admin.Admin RMISERVER 8029" to start your server. RMIServer protocol is slower much than the default TCPServer protcol.

7. I would like to start a server (TCP) from our application, instead of DBAdmin. I need to be able to programmatically

tell the application which profile to start.
Please read Start/Stop Server Programmatically.

8. Is there a way to specify a file path in the url that will connect to a mapped drive in Windows 2000. ie drive \\gomer\pyle\db which is mapped to f drive on the server.
Access driver can work with mapped driver, and you should use "jdbc:access:/f:" to access your data.

Note: If you're using a database file through a UNC path or a mapped drive of Windows, there is a Windows Security restriction. If you run ColdFusion (Tomcat, or tanuki sw wrapper) as a service on Windows, it operates by default as System, and cannot access directories on a remote system or mapped drive; to resolve this issue, do not run ColdFusion (Tomcat, or tanuki sw wrapper) using the local system account.

9. When I click Start button to start a remote service, I get a security excaption: access denied (java.net.SocketPermission 127.0.0.1:8029 connect,resolve)
You have to enable java.net.SocketPermission right in your policy file if you run a Access server. Please read file:///yourdriver|/jdk1.2/docs/guide/security/PolicyFiles.html for more information about policy file. It is unnecessary to know the specific content of a policy file, since you can use policy tool to create and maintain your policy files. Please read file:///yourdriver|/jdk1.2/docs/tooldocs/win32/policytool.html for policy tool.

10. How to start remote service as MS Windows service and Linux(Solaris) Daemon?
Please read Run HXTT AccessServer as Windows Service or Linux(Solaris) Daemon.

11. How to start remote control when AccessServer is running as Windows service or Linux(Solaris) Daemon?
You can use "java com.hxtt.sql.admin.Admin TCPCLIENT [host:]port [remoteControlPassword]" to start your remote control.

SQL Questions

1. I need to use tables stored in a subdirectory.
table-name: [catalog.]tableName
For instance, you have many Access files on c:\data. You can use "jdbc:access:/c:/data" as JDBC url. Then you can use "select * from subdirectory1.table1" to visit table1 file at subdirectory1. For instance, "select tableAlias.* from "sales/2004/04".sale as tableAlias" can access sale table at "c:\data\sales\2004\04".

2. I can't use "select RIGHT from deldob"
RIGHT is a reserved SQL keyword. "variableName", [variableName] or {v 'variableName'} is used to quote those columns which use reserved keyword, so that you should use "RIGHT" or {v 'RIGHT'} to quote the RIGHT field, for instance, *select {v 'RIGHT'},'other' from states where "RIGHT"=32*. HXTT Access supports using DATE, TIME, TIMESTAMP, GROUP, ORDER, KEY, DESC, UPDATE directly in SQL, although they're reserved words too.

3. Can I get an example on how to do a query involving a boolean value. eg. " Select * from tableName where exported = true", where exported is a boolean column in a Access file.
Supports. You can use "select * from tableName where exported" too. All of NOT, AND, and OR operation are supported.

4. How to specify dates?
Please use SQL Escape Syntax, a date is specified in a JDBC SQL statement with the syntax {d `yyyy-mm-dd'} where yyyy-mm-dd provides the year, month, and date, e.g. 1996-02-28. There are analogous escape clauses for TIME and TIMESTAMP type: {t `hh:mm:ss'} and {ts `yyyy-mm-dd hh:mm:ss.f...'}. The fractional seconds (.f...) portion of the TIMESTAMP can be

omitted. For instance,{d '1999-11-01'} and {ts '3999-03-24 00:59:23.22222'}. You can use PreparedStatement.setDate to set date columns too.

5. How to handle date range selection, e.g. SELECT * FROM CALLS WHERE START >= '2001-01-01' AND END <= '2002-01-01'
Although the HXTT Access supports "SELECT * FROM CALLS WHERE START >= '2001-01-01' AND END <= '2002-01-01'", but that sql syntax is unadvisable. Please use SQL Escape Syntax, {d `yyyy-mm-dd'} and {ts `yyyy-mm-dd hh:mm:ss.f...'}, for Date and timestamp type according to JDBC standard. You can learn more about Escape Syntax at file:///yourdriver|/jdk1.2/docs/guide/jdbc/spec/jdbc-spec.frame11.html . You should use "select * from calls where start>={d '2001-01-01'} and end <={d '2002-01-01'}".

6. Can {d '2999-11-21'}={ts '2999-11-21 23:22:20.3335'} and {t '23:22:20'}={ts '1999-01-01 23:22:20.333'}?
Supports.

7. I think this one is for use functions {fn abs(TEST.int1)}
You can use abs(TEST.int1) too. HXTT Access supports more than 210 functions.

8. Update table_name set (fieldname1=X, fieldname2=X2, ...) where primary_index='blah' throws a parse exception.
You should use "update table_name set fieldname1=X, fieldname2=X2, ... where primary_index='blah'".

9. How to delete all deleted records permanently?
"PACK TABLE [IF EXISTS] table_name" will pack database.
"TRUNCATE TABLE [IF EXISTS] table-name" will zap database.

Index Questions

1. How to rebuilding index in case of corrupted index?
REINDEX {ALL | indexFileName[,indexfileName2,...]} ON table-name

2. I receive 1 record back, however there should be 8 records returned.My SQL is "SELECT * FROM Schshift@brian WHERE PSCHED='0001092478'"
You should have a UNIQUE index restriction on your PSCHED column in your index file. You should use "CREATE INDEX PSCHED on Schshift (PSCHED)", not "CREATE INDEX PSCHED on Schshift (PSCHED UNIQUE)". Then you can get all ten records. Access driver will use index to speed up the query which contains some index expressions.

3. We tried to set a PRIMARY KEY constraint with: create unique index PROVA on PROVA (COD)
You should try "CREATE INDEX prova ON prova (cod PRIMARY KEY).

4. I have a table that lists an index using: STR(ClassLink,4,0)+STR(StuLink,5,0) as the column_name. I want to join it to another table that has an index that uses the same columns... What should the join statement look like in order to take advantage of the indexes?
For instance, you can use "select * from ACLS3295,AGRD3295 where STR(ACLS3295.ClassLink,4,0)+STR(ACLS3295.StuLink,5,0)='1234abcde' and STR(AGRD3295.ClassLink,4,0)+STR(AGRD3295.StuLink,5,0)='5678abcde'", or "select * from ACLS3295 as a,AGRD3295 as b where STR(a.ClassLink,4,0)+STR(a.StuLink,5,0)='1234abcde' and STR(b.ClassLink,4,0)+STR(b.StuLink,5,0)='5678abcde'".

Performance Questions

1. What is the most efficient methodto insert records in a table, to use an updatable RecordSet or to use a PreparedStatement?

PreparedStatement is smally quicker than updatable RecordSet. An updatable RecordSet is quicker than PreparedStatement if you insert into more than 200 columns with constant values. It can only cope with constant values. PreparedStatement can cope with complicated expressions so that you can insert timestamp, function, ResultSet, and so on.

2. "select count(*) from table" are worked a long time for large tables.
You should use "select reccount() from table" to get the number of records. Count(*) sums always up all records except deleted row.

3. Are there any data row count, data volume, memory minimums, maximums imposed when using the HXTT Access?
No limitation. The HXTT Access supports to join query big databases with DISTINCT, GROUP BY, and ORDER BY.

Concurrence Questions

1. Does HXTT Access support multi-user access?
The HXTT Access supports multi-user access, record lock, and table lock.

2. Is there any way to lock/unlock record programatically.
We have provided a _LockFlag_ virtual column as row lock flag. You can know it from Set Record Lock Manually.

Internationalization Questions

1. Can the HXTT Access support Czech MS - DOS 895?

The HXTT Access supports all codepage, multilingual collation sequence, and unicode character set. Cp895(Czech MS - DOS 895), Cp620(Polish MS - DOS 620) and Mazovia are extra supported although JVM doesn't support those.

2. Do you have a solution for character translation to the right encoding?
The HXTT Access supports CharacterEncoding. Please use charSet property.

```
//Default: null
//You can find a Supported Encodings list of
files:///yourdriver/jdk1.2/docs/guide/internat/encoding.doc.html
//Extra supports:
// Cp895 is supported by HXTT Access driver. //Czech MS - DOS 895
// Cp620 is supported by HXTT Access driver. //Polish MS - DOS 620
// Mazovia is supported by HXTT Access driver. //Polish
Properties properties=new Properties();
properties.setProperty("charSet","sv_SE");
Connection con = DriverManager.getConnection(url,properties);
```

3. While reading encrypted data in a Access file using u'r parser in java. The data retreived is different from the data in the Access file, certain characters are read as ? marks.(the encryption is done using ASCII values).
You can use ResultSet.getBytes(int columnIndex), not ResultSet.getString(int columnIndex) and ResultSet.getObject(int columnIndex), to get your encrypted data, since your encrypted data is binary stream.

4. When they insert accented characters, it comes out different at the Java end. There seem to be some character set conversion problems. Is there a way to solve that?
You can use ResultSet.getBytes() and ResultSet.setBytes() to avoid CharacterEncoding.

Interoperability Questions

1. How to set up HXTT Access with Tomcat4.1 as PoolableConnection?

This sample shows three PoolableConnections ways through Database Connection Pool (DBCP) Configurations and JNDI Resources(You should read JNDI Datasource HOW-TO and JNDI Resources HOW-TO also.):

In server.xml:

```
<Context path="" docBase="ROOT" debug="5" reloadable="true" crossContext="true">

    <Resource name="jdbc/testaccessPool1" auth="Container"
type="javax.sql.DataSource"/>
        <ResourceParams name="jdbc/testAccessPool1">
                <parameter>
                        <name>factory</name>
                        <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
                </parameter>

                <parameter>
                        <name>maxActive</name>
                        <value>50</value>
                </parameter>

                <parameter>
                        <name>maxIdle</name>
                        <value>10</value>
                </parameter>

                <parameter>
                        <name>maxWait</name>
                        <value>10000</value>
                </parameter>

                <parameter>
                        <name>username</name>
                        <value></value>
                </parameter>

                <parameter>
                        <name>password</name>
                        <value></value>
                </parameter>

                <parameter>
                        <name>driverClassName</name>
                        <value>com.hxtt.sql.access.AccessDriver</value>
                </parameter>

                <parameter>
                        <name>url</name>
                        <value>jdbc:access:///d:/accessfiles</value>
                </parameter>
    </ResourceParams>

    <Resource name="jdbc/testAccessPool2" auth="Container"
type="com.hxtt.sql.HxttConnectionPoolDataSource"/>
```

```
        <ResourceParams name="jdbc/testAccessPool2">
                <parameter>
                        <name>factory</name>
                        <value>org.apache.naming.factory.BeanFactory</value>
                </parameter>

                <parameter>
                        <name>url</name>
                        <value>jdbc:access:///d:/accessfiles</value>
                </parameter>

                <parameter><name>username</name><value></value></parameter>
                <parameter><name>password</name><value></value></parameter>
                <parameter><name>host</name><value></value></parameter>
                <parameter><name>port</name><value>8029</value></parameter>

    </ResourceParams>


    <Resource name="jdbc/testAccessPool3" auth="Container"
type="com.hxtt.sql.HxttConnectionPoolDataSource"/>
        <ResourceParams name="jdbc/testAccessPool3">
                <parameter>
                        <name>factory</name>
                        <value>com.hxtt.sql.HxttObjectFactory</value>
                </parameter>
                <parameter>
                        <name>url</name>
                        <value>jdbc:access:///d:/accessfiles</value>
                </parameter>

                <parameter><name>username</name><value></value></parameter>
                <parameter><name>password</name><value></value></parameter>
                <parameter><name>host</name><value></value></parameter>
                <parameter><name>port</name><value>8029</value></parameter>
    </ResourceParams>

</Context>
```

Then you can use the below code to test those PoolableConnections:

```
        Context initContext = new InitialContext();
        Context envContext = (Context)initContext.lookup("java:/comp/env");

        DataSource ds1 = (DataSource)envContext.lookup("jdbc/testAccessPool1");
        Connection conn1 = ds1.getConnection();
        out.println("testAccessPool1 OK:)<br/>");
        Statement stmt1 = conn1.createStatement();
        ResultSet rs1 = stmt1.executeQuery("select * from test");
        if(rs1.next())
           out.println(rs1.getString(1)+":)<br/>");
        rs1.close();
        stmt1.close();
```

```
conn1.close();


DataSource ds2 = (DataSource)envContext.lookup("jdbc/testAccessPool2");
Connection conn2 = ds2.getConnection();
out.println("testAccessPool2 OK:)<br/>");
Statement stmt2 = conn2.createStatement();
ResultSet rs2 = stmt2.executeQuery("select * from test");
if(rs2.next())
   out.println(rs2.getString(1)+":)<br/>");
rs2.close();
stmt2.close();
conn2.close();

DataSource ds3 = (DataSource)envContext.lookup("jdbc/testAccessPool3");
Connection conn3 = ds3.getConnection();
out.println("testAccessPool3 OK:)<br/>");
Statement stmt3 = conn3.createStatement();
ResultSet rs3 = stmt3.executeQuery("select * from test");
if(rs3.next())
   out.println(rs3.getString(1)+":)<br/>");
rs3.close();
stmt3.close();
conn3.close();
```

If you use org.apache.commons.dbcp.BasicDataSource, but get "Cannot create PoolableConnectionFactory" Error, you should check your commons-pool-1.x.jar and commons-dbcp-1.*.jar file in $TOMCAT/common/lib directory to see whether two files have the same version. DBCP v1.2 requires Pool v1.2 so that you should update Pool v1.1 from the tomcat website.

If you wish to add more Connection properity, you should use connectionProperties, for instance:

<parameter>
<name>connectionProperties</name>
<value>charSet=Cp737</value>
</parameter>

2. How to set up HXTT Access with vqServer 1.9.55 as web server?
The key is to use an absolute path as Java libraries' location, and restart vqServer after modified Java libraries.
For instance, your vqServer is installed at C:\vqServer\.
1. Please use http://yourhost:9090/ to visit your administration server.
2. Click on Java libraries in the vqServer control centre menu (http://yourhost:9090/admin?action=libraries&serial=14)
3 Click New library (http://yourhost:9090/admin?lib=New_library&action=edit)
4. Enter C:\vqServer\classes\Access_JDBC20.jar as location value, Access Driver as Description value, then click OK button.
5. Please copy Access_JDBC20.jar into C:\vqServer\classes directory.
6. Please copy ex01.class into C:\vqServer\servlets\servlets
7. Stop and restart vqServer
8. Please use http://yourhost/servlet/yourServlets to get your result.

3. How to set up HXTT Access with Coldfusion MX 6.1 Application Server?
For instance,your Coldfusion MX is installed at C:\CFusionMX\, and wish to use Access_JDBC30.jar.
1. Please copy Access_JDBC30.jar into C:/CFusionMX/wwwroot/WEB-INF/classes/.
2. Use http://yourhost:8500/CFIDE/administrator/index.cfm to enter the CFMX Administrator.

3. Go to the "Java and JVM" of Server Settings, http://yourhost:8500/CFIDE/administrator/settings/jvm.cfm page, and enter the full path, C:/CFusionMX/wwwroot/WEB-INF/classes/Access_JDBC30.jar, in the Class Path. Then, click "Submit Changes".
4. Restart the CFMX Service.
5. Please go back to the administrator page, and go to the "Data Sources" of Data & Services, http://yourhost:8500/CFIDE/administrator/datasources/index.cfm page, and enter the name for the new datasource, for instance "AccessTest", and select "Other" for the driver. Then Click "Add".
6. Enter the datasource information. JDBC URL is always in the format jdbc:access:[//[host:port]]/[DatabasePath], for instance jdbc:access:/c:/data. Driver class is always com.hxtt.sql.access.AccessDriver. Driver name is used to identify the driver in the datasources view, and you can use Access. Username and password are not required. They can also be specified in the cfquery tag (but datasource verification will fail if you don't enter them). Description is not required.
7. If you wish to set more connection properties, please click "Show Advanced Setting" button, then in the textbox for "Connection String", you can input "delayedClose=15;maxCacheSize=6144;lockTimeout=2000;" (three properites are just a demo, not necessary). Note: Connection String seems abnormal now. You should have to put Connection String into JDBC URL, for instance: jdbc:access:/c:data?delayedClose=15;maxCacheSize=6144;lockTimeout=2000;
8. Lastly, please press "Submit" to finalize the entered data.
9. You can find edit.cfm and edit_action.cfm sample in demo pacakge.

4. HXTT Access with If you run ColdFusion (Tomcat, or alexandria sw and tanuki sw wrapper) on Windows 2000 and Windows XP Pro does not work on mapped drives.

Note: If you're using a database file through a UNC path or a mapped drive of Windows, there is a Windows Security restriction. If you run ColdFusion (Tomcat, or tanuki sw wrapper) as a service on Windows, it operates by default as System, and cannot access directories on a remote system or mapped drive; to resolve this issue, do not run ColdFusion (Tomcat, or tanuki sw wrapper) using the local system account.

The service(For instance, ColdFusion MX Application Server, ColdFusion MX 7 Application Server, or Apache Tomcat) built by ColdFusion (Tomcat, or tanuki sw wrapper) can not access the share directory at other machine by default. But you can do as follows to solve this problem:
1. Right click the service built by ColdFusion (Tomcat, or tanuki sw wrapper) in service manager, and click the property menu.
2. On the open window,select the login tab, click this account radio box, and click the browse button.
3. Select the administrator account(it seems that you should select the administrator account), input the correct password in the password textbox and confirm password textbox.
4. Restart this service, you can find this service can access the share directory at other machine.

5. How to resolve 'DataSet has no unique row identifiers.' issue in JBuilder's QueryDataSet?
You can use _rowid_, a virtual column to avoid that issue, For instance:

```
        //...
        queryDataSet = new QueryDataSet();
                //...
                queryDataSet.setMetaDataUpdate(MetaDataUpdate.ALL-
MetaDataUpdate.ROWID-MetaDataUpdate.TABLENAME);

        queryDataSet.setQuery(new QueryDescriptor(database, "select _rowid_,* from
test", null, true,
                Load.ALL));
        queryDataSet.open();

        queryDataSet.setTableName("test");
        queryDataSet.setRowId("_rowid_", true);
                //...
```

6. How to set HXTT Access with WebSphere Application Server?
You can download a pdf guide from here.

7. How to set HXTT Access with Hibernate?
You should download support package and sample from here.

8. How to set HXTT Access Data Source with Oracle Application Server 10G?
You should read guide at Oracle Application Server 10G(v10.1.3) and Oracle Application Server 10G(v10.1.2.02).

9. How to set HXTT Access Data Source with JBoss Application Server 4.0.1?
For instance,

```
<datasources>
  <local-tx-datasource>
    <jndi-name>TestData</jndi-name>
    <connection-url>jdbc:access:////data</connection-url>
    <driver-class>com.hxtt.sql.access.AccessDriver</driver-class>
    <connection-property name="delayedClose">-1</connection-property>
    <user-name/>
    <password/>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <idle-timeout-minutes>5</idle-timeout-minutes>
  </local-tx-datasource>
</datasources>
```

Released Version Log

Latest feature:

- v3.0.036 supports GROUP_CONCAT function.
- v3.0.035 supports to assign more than one extension for otherExtension connection property.
- v3.0.034 supports lower(upper) function in LIKE expression.
- v3.0.031 fixed a bug on utilizing part expression of component index to quicken IN query.
- v3.0.026 quickened self insert.
- v3.0.025 fixed a bug on utilizing part expression of component index to quicken query.
- v3.0.017 supports invisible VIEW type in MS Access.
- v3.0.001 supports INTO variable[,...] for SELECT sql.
- v3.0.001 supports SET variable = expression [,...].
- v3.0.001 supports DECLARE Local Variables.

2007-10-29 Access v3.0 JDBC1.2 Package(2236KB) JDBC2.0 Package(2225KB) JDBC3.0 Package(2270KB)

- v3.0.001 supports INTO variable[,...] for SELECT sql.
- v3.0.001 supports SET variable = expression [,...].
- v3.0.001 supports DECLARE Local Variables.
- v2.1.084 replaced "? Expression" with "SELECT select_list".
- v2.1.083 provides CURRVAL(cTableName, cColumnName) to fetch the last generated IDENTITY(auto_increment) value for a particuar table.
- v2.1.080 provides SHA1 function besides MD5 and Crypt3.
- v2.1.076 will recycle space for REINDEX operation.
- v2.1.076 fixed a string index bug in JET3 database engine for a few sort orders since v2.1.006.
- v2.1.073 solved a strange issue which some memo messages show abnormally with MS Access 2003 and 2007 on MS Windows 2003 and Vista, but normally with MS Access 200 and 2003 on MS Windows 2000 and XP.
- v2.1.070 can understand very complicated VIEW(QUERY of MS ACCESS) with more than 15 JOIN table relations.
- v2.1.068 changed the visibility of columns in JOIN table with parentheses from invisible to visible.
- v2.1.068 supports JOIN and subquery in PIVOT and UNPIVOT.
- v2.1.068 supports a variety of sort orders for JET4 database(General, German Phone Book, France, Icelandic, Dutch, Traditional Spanish, Spanish, Swedish/ Finnish, Croatian, Czech,

Hungarian, Polish, Romanian, Slovak, Slovenian, Estonian, Latvian, Lithuanian, Macedonian, Ukrainian, Chinese Stroke Count (Taiwan), Georgian Modern, Hungarian Technical, Japanese Unicode, Korean Unicode, Chinese Pronunciation, Chinese Stroke Count, Turkish, Vietnamese, Chinese Bopomofo (Taiwan), Japanese, Korean, and Thai).

- v2.1.057 supports seamlessly https url database in jdbc url and sql.
- v2.1.051 fixed a bug for utilizing index on LIKE '%'.
- v2.1.048 fixed a seldom bug for CREATE TABLE into JET4 version database.
- v2.1.037 fixed a bug for insert very long data into JET3 version database.
- v2.1.010 supports AUTO_INCREMENT in CREATE TABLE sql
- v2.1.001 supports SAMBA table, which needn't to map or mount driver.

2007-01-11 Access v2.1 JDBC1.2 Package(905KB) JDBC2.0 Package(895KB) JDBC3.0 Package(935KB)

- v2.0.047 optimizes memory occupation for UNION ALL.
- v2.0.044 added CP850, CP852, CP866, CROATIAN, HEBREW, and SWEDISH sort for COLLATE function.
- v2.0.043 fixed a slow speed issue since v2.0.035 which was resulted by cascading check.
- v2.0.043 utilizes constant DEFAULT value for create table sql and insert sql.
- v2.0.041 fixed a bug for insert very long data into JET3 version database.
- v2.0.041 supports fully ALTER TABLE sql.
- v2.0.041 supports DROP INDEX sql.
- v2.0.036 fixed a bug on DELETE operation for a special seldom occasion.
- v2.0.035 supports cascading updates and cascading deletes of MS Access.
- v2.0.023 supports to use more than one column as primary key in create table sql.
- v2.0.022 optimizes speed of ResultSet.absolute function.
- v2.0.021 fixed a bug in using ResultSet.absolute function.
- v2.0.018 fixed an incompatible issue for MS Access 97's CLOB value with more than 2048 length. HXTT Access can read those older inserted CLOB values, but MS Access will show an error data prompt, then can show correctly those "error" CLOB data.
- v2.0.014 optimizes speed for UNION.
- v2.0.012 provides maxIdleTime connection property. maxIdleTime indicates the max idle time in minute for remote connection. That option is mainly used to avoid closing automatically idle remote connection for connection pool. Embedded idle connectoin won't be closed automatically except for garbage collection. You can use 1~1440 minutes. Default: 15.
- v2.0.011 supports REINDEX sql.
- v2.0.004 can detect the latest inserted rows by other MS Access applications with delayedClose=0.

2006-08-07 Access v2.0 JDBC1.2 Package(980KB) JDBC2.0 Package(966KB) JDBC3.0 Package(1005KB)

- v1.2.025 fixed a bug in string index optimization since v1.2.021 released on June 21, 2006.
- v1.2.022 optimizes VIEW query.

- v1.2.021 supports TRANSFORM VIEW.
- v1.2.020 supports PIVOT and UNPIVOT.
- v1.2.015 provides CDATE(expression,pattern) for date conversion.
- v1.2.008 fixed a bug in index result cache.
- v1.2.004 supports index on special (national) characters, like ä.
- v1.2.003 supports RENAME TABLE sql.

2006-05-17 Access v1.2 JDBC1.2 Package(852KB) JDBC2.0 Package(843KB) JDBC3.0 Package(883KB)

- v1.1.036 fixed a bug in inserting row into mdb file with 127MB size.
- v1.1.035 fixed a bug in parsing value for numeric column with scale>10.
- v1.1.032 fixed a bug in DatabaseMetaData.getTables() since v1.1.028.
- v1.1.028 supports most simple stored procedures (select, insert, update, and delete) for JDBC3.0 package. You should test your stored procedure first to see whether it's supported by HXTT Access
- v1.1.024 supports seamlessly url(http, ftp) database in jdbc url and sql.
- v1.1.024 supports seamlessly memory-only database in jdbc url and sql for internal data processing, applets, or certain special applications.
- v1.1.024 supports seamlessly files and directories in TAR and BZ2 file formats(.TAR, .BZ2, .TGZ, .TAR.GZ, .TAR.BZ2) in jdbc url and sql.
- v1.1.017 provides otherExtension connection property to support other extension beside 'MDB' and 'MDE'.
- v1.1.015 provides TABLELOCKED(cTableName) function.
- v1.1.001 provides lock table and unlock table sql.

2006-03-07 Access v1.1 JDBC1.2 Package(807KB) JDBC2.0 Package(822KB) JDBC3.0 Package(853KB)

- v1.0.142 optimizes IN, NOT IN, ALL, and ANY on subquery.
- v1.0.133 provides SSL connection and customer connection for client/server mode.
- v1.0.130 supports seamlessly files and directories in ZIP and GZIP file formats(.ZIP, .JAR, .GZ) in jdbc url and sql.
- v1.0.129 changed CREATE SCHEMA sql to CREATE CATALOG
- v1.0.129 changed [schemas.]tableName@[catalog] format to [catalog.]tableName
- v1.0.128 supports MDE suffix
- v1.0.115 supports column numbers in ORDER BY clause
- v1.0.114 supports Multiple-row VALUES tables.
- v1.0.104 optimizes speed.
- v1.0.100 supports MySQL Migration Toolkit v1.0.21
- v1.0.98 supports aggregate function first(x), last(expression).
- v1.0.96 supports function ATN(x), CBOOL(expression), CBYTE(expression), CDBL(expression), CINT(expression), CLNG(expression), CSNG(expression),

CSTR(expression), and CDATE(expression).
- v1.0.62 supports DROP TABLE
- v1.0.28 supports all of four transaction levels.
- supports CREATE DATABASE

2005-09-12 Access v1.0 JDBC1.2 Package(731KB) JDBC2.0 Package(752KB) JDBC3.0 Package(777KB)

- supports MySQL Migration Toolkit v1.0.20
- supports CREATE TABLE and CREATE INDEX
- supports MS Windows service and Linux Daemon for remote connection and remote control
- supports VIEW(QUERY of MS ACCESS) with/without parameter.
- supports DISTINCTROW.
- supports Linked Table to other Access databases
- provides ILIKE syntax support,ignore upper and case like

2005-06-01 Access Beta1.0 JDBC1.2 Package(698KB) JDBC2.0 Package(712KB) JDBC3.0 Package(732KB)

- Development Documentation is available.
- provides the quicker TCPServer to replace the slow RMIServer.
- provides Database GUI Manager.
- supports JDBC3.0, JDBC2.0 and JDBC1.2.
- provides Table Encryption and ColumnLevel Encryption.
- provides _LockFlag_ virtual column as row lock flag for Borland's dataset.
- provides encrypt/decrypt function for Row-Column (Cell) Level Encryption.
- provides transaction sql.