Ashpool - XML Database Command Reference

Rob Rohan

Revision History
Revision 3rc3 2003-04-20
Draft

Table of Contents

Select Statement

Select statments are run against XML documents in the datastore (the specifed directory). XML documents can be added using a **create table** command, or by simply moving an XML document into the datastore.

Note tables must end in ".xml". The ".xml", however, is not added when writing a query. For example, **select * from test_tbl** is correct, but **select * from test_tbl.xml** is incorrect when querying an XML document named

test_tbl.xml in the datastore.

When selecting from an XML document, the column names are case-sensitive. Whatever case the XML document's elements use - that is the case the columns need to be. For example, a column defined as <FirstName>Bob </FirstName> would need a query like **select FirstName from ...** SQL keywords are not case-sensitive; however Ashpool is a case sensitive database so **where a = 'this'** is different from **where a = 'This'**

Supported SQL Syntax

`SELECT` [* | column1 [AS string], column2, ...columnN] FROM [table]
[WHERE column [= | != | < | <= | > | >= | like | not like] [number | 'string' | 'date'] [AND | OR] ...]
[ORDER BY column [ASC | DESC], ...]

Ashpool Specific Syntax

`SELECT TABLES` - gets a list of all the tables in the datastore
`SELECT COLUMNS` [table] - gets a list of column names (and types, nilibilty, etc if an xsd is defined)
`SELECT TEST` - to test connections
`SELECT TYPES` - gets the supported datatypes

Known Oddities

- Function usage in the where clase is limited.
- datetimes should be handled the same as strings, and datetimes should be in iso-8601 format (yyyy-mm-ddThh:mm:ss or yyyy-mm-dd)

Select Example

Example 1.

```
select
    fristname,
    lastname,
    clientid,
    date
from test_tbl
where clientid > 3 and (firstname like 'oh' or lastname = 'fresh')
order by clientid asc;
```

Drop Statement

Droping a table deletes the XML file from the datastore. If there is a schema and a sequence file with the same name as the XML file (table) they are deleted as well (schema XML files end with an “.xsd” and sequence files end with “.icf”).

Supported SQL Syntax

`DROP TABLE` [table]

Drop Example

Example 2.

drop table test_tbl;

Create Statement

The create statment will create two or three files. It will create an empty XML document (the table), it will create a schema file with the constraint criteria provided (table schema), and if the *serial* datatype is used it will create a sequence files to track the autonumber.

Supported SQL Syntax

`CREATE TABLE` [table] ([columnname] [columntype [(size)]] [,...])

Data Types

In an effort to be a bit more portable, Ashpool trys to accept datatypes specified in standard terms. For instance, **create table booga(name varchar(30))** will internally be a string. The following is a list of what supported foregin datatypes map to.

Table 1. Ashpool Data Types

Ashpool Datatype	Synonyms
string	char varchar
integer	int int4 int8
float	float4 float8 money currency
boolean	bool flag yes/no
dateTime	timestamp

Create Example

Example 3.

```
create table fresh_tbl (
    id serial,
    birthday date,
    firsname string(30),
    lastname string(50),
    good boolean,
    data base64Binary
);
```

Example 4.

```
create table fresh_tbl (
    id serial,
    birthday date,
    firsname varchar(30),
    lastname char(50),
    good bool,
);
```

Insert Statement

There are two versions of the INSERT statement. The first is the “Supported SQL Syntax” the second is the Ash-“pool Specific” syntax. The supported syntax acts like a simple normal insert statement, and the Ashpool specific syntax is mostly used to create temp tables (sort of like an array).

> **Note**
> Sub-selects are not totally supported yet

If there is a schema available for the table being inserted into, the constraints defined by they schema will be in play when trying to do an insert. For example, if a column is defined as **string(2)** and an insert trys to add *'hello out there'*, an error will occur. On the other hand if the table has no schema, all columns are treated as unlimited string fields and must be inserted as strings (i.e. enclosed in ').

When inserting dates or timestamps, format the date in iso-8601 format. iso-8601 defines datetimes as yyyy-“mm-ddThh:mm:ss” and dates as “yyyy-mm-dd” - quotes are not needed.
I am working on better date support.

Supported SQL Syntax

`INSERT INTO` [table] (
[column name1], [column name2], ... [column nameN],) `VALUES` ([column value1 | (select ...)], [column value2], ... [column valueN],);

Ashpool Specific Syntax

`INSERT INTO` [~ [table]]
[SELECT ...]

Known Oddities

- Non-flexable date handling
- Sub-selects are not fully supported
- INSERT INTO table SELECT *... not supported

Insert Into Example

Example 5.

```
insert into orders_tbl (
    firstname,
    lastname,
    date,
    sent
) values (
    'Rob',
    'Rohan',
    2003-03-15,
    true
);
```

Example 6.

```
insert into ~mytemp
select
    firstname,
    date
from test_tbl
where firstname like 'O'
order by date desc;
```

Delete Statement

The delete statement is used to delete rows from a table.

Supported SQL Syntax

`DELETE` [* FROM] [table] [WHERE ...]

Delete Example

Example 7.

```
delete * from Customers
where cust_id >= 3 or cust_name like 'B%';
```

Example 8.

```
delete from Customers
where cust_id = 12;
```

Example 9.

```
delete Customers
where cust_name = 'Ok I get it';
```

Update Statment

Update statements are used to change data already in a table. Ashpool has basic update support.

Supported SQL Syntax

```
UPDATE [table]
[SET column1 = value] [,]
[[columnN = valueN]...]
[WHERE critera]
```

Update Example

Example 10.

```
UPDATE foobar
SET fname = 'Hi there',
comments = 'oh thats nice',
price = 12.22
WHERE foobar_id = 123;
```

Alter Statement

The alter statement can only adjust sequences at present. It can not alter tables. The integer portion of the alter sequence statement makes the sequence start at the next number from the one specified. In other words **ALTER sequence foobar 100** will cause the next insert statment to use 101 as the autonumber value.

Ashpool Specific Syntax

```
ALTER [sequence] [table] [integer]
```

Alter Sequence Example

Example 11.

ALTER sequence foobar 100

Functions

The following is a quick reference of Ashpools built in functions. It is important to note that all of these functions can be applied to the column section of a query only. They can not be used in search criteria (the *WHERE* clause). In addition, when a function is used on a column the *AS* keyword will almost always need to be used.

String Functions

Table 2. String Functions

String **concat**([Field0 | 'String0'], [Field1 | 'String1'],... [FieldN | 'StringN'])	Creates a single string out of the supplied parameters.
boolean **contains**([Field0 | 'String0'], [Field1 | 'String1'])	Returns true if string1 or field1 is in string0 or field0
boolean **starts-with**([Field0 | 'String0'], [Field1 | 'String1'])	Returns true if string0 or field0 starts with string1 or field1
integer **string-length**([Field0 | 'String0']) integer **length**([Field0 | 'String0'])	Returns the length of the given field or string
String **substring**([Field0 | 'String0'], [Start Index] [,End Index])	Returns the portion of Field0 or String0 begining from start index. An end index can also be specified.
String **substring-after**([Field0 | 'String0'], [Field1 | 'String1'])	Returns the portion of the first parameter that comes after the second parameter
String **substring-before**([Field0 | 'String0'], [Field1 | 'String1'])	Returns the portion of the first parameter that comes before the second parameter
String **upper**([Field0 | 'String0'])	Converts the passed parameter to upper case (English only).
String **lower**([Field0 | 'String0'])	Converts the passed parameter to lower case (English only).

Date Time Functions

Table 3. Date Time Functions

dateTime or date **now**()	Used in insert into statements, creates a dateTime or a date string for the current time.
dateTime **date-time**()	Returns the current date and time in iso-8601 format
date **date**()	Returns the current date in iso-8601 format
time **time**()	Returns the current time in iso-8601 format
integer **year**(['date' | 'dateTime'])	Returns the year contained in the passed date or dateTime string. If no argument is passed, returns the current year.
boolean **leap-year**(['date' | 'dateTime'])	Returns true if the passed date or dateTime is a leap year
integer **month-in-year**(['date' | 'dateTime'])	Returns the month part of a date or dateTime string (i.e. 3 for '2003-03-07')
String **month-name**(['date' | 'dateTime'])	Returns a long string version of a months name. I.e. 'March' for '2003-03-07' (English only)
String **month-abbreviation**(['date' | 'dateTime'])	Returns a short string version of a months name. I.e. 'Mar' for '2003-03-07' (English only).
integer **week-in-year**(['date' | 'dateTime'])	Returns the week the passed date falls in given 52 weeks in a year.
integer **week-in-month**(['date' | 'dateTime'])	Returns the week of the month the passed date falls on.
integer **day-in-year**(['date' | 'dateTime'])	Returns the day of the year the passed day is on. For example, 2003-12-31 would be 365.
integer **day-in-month**(['date' | 'dateTime'])	Returns the day of the month the passed day is on.
integer **day-of-week-in-month**(['date' | 'dateTime'])	Returns the day of the week in the month the passed day is on. For example, 2003-03-07 returns 1.
integer **day-in-week**(['date' | 'dateTime'])	Returns the day in the week the passed day is on. For example, 2003-03-07 returns 6.
String **day-abbreviation**(['date' | 'dateTime'])	Returns a short string version of the day in the week the passed day is on. For example, 2003-03-07 returns 'Fri'.
integer **hour-in-day**(['date' | 'dateTime'])	Returns the hour in the passed dateTime.
integer **minute-in-hour**(['date' | 'dateTime'])	Returns the minutes in the passed dateTime.
integer **second-in-minute**(['date' | 'dateTime'])	Returns the seconds in the passed dateTime.

Math Functions

In the following functions the word *number* is used to mean whatever data type is passed in to a function is the data type that is returned. For example, **abs(-1.232323)** will yield 1.232323 and **abs(-2)** will yield 2.
Most of these function were created by www.exslt.org, and the descriptions are taken from that site as well.

Table 4. Math Functions

number **abs**([number])	Returns the absolute value of a number.
number **acos**([number])	Returns the arccosine value of a number in radians.
number **asin**([number])	Returns the arcsine value of a number in radians.
number **atan**([number])	Returns the arctangent value of a number in radians.
number **atan2**([number], [number])	Returns the angle (in radians) from the X axis to a point (y,x). Value1 is a number argument cooresponding to y of point (y,x). Value2 is a number argument cooresponding to x of point (y,x).
number **constant**([string], [number])	Returns a constant to a specified precision. The possible constants are • PI • E • SQRRT2 • LN2 • LN10 • LOG2E • SQRT1_2
number **cos**([number])	Returns cosine of the passed argument in radians.
number **exp**([number])	Returns e (the base of natural logarithms) raised to a power. The return value is the power of the constant e. The constant e is Euler's constant, approximately equal to 2.178.
number **highest**([node-set])	...TBD...
number **log**([number])	Returns the natural logarithm of a number. The return value is the natural logarithm of number. The base is e.
number **lowest**([node-set])	...TBD...

number **power**([number base], [number power])	Returns the value of a base expression taken to a specified power.
number **random**()	Returns a random number from 0 to 1.
number **round**([number])	Rounds a number
number **sin**([number])	Returns the sine of the number in radians.
number **sqrt**([number])	Returns the square root of a number. If the argument is a negative number, the return value is zero.
number **tan**([number])	returns the tangent of the number passed as an argument in radians.

Aggregate Functions

Table 5. Aggregate Functions

number **sum**([Field])	Returns the sum of the specified field
integer **count**([Field])	Returns a count of the specified field. If the passed field is *, the function returns a count of all rows and columns.
number **max**([Field])	Returns the maximum of the specified field
number **min**([Field])	Returns the minimum of the specified field
number **avg**([Field])	Returns the average of the specified field