

SQLite Functions, Aggregators and Collations

Walt Mankowski (walt@cs.drexel.edu)
Drexel University

PLUG North
13 July 2009

SQLite's different...

SQLite's different...

- embedded, in-process library

SQLite's different...

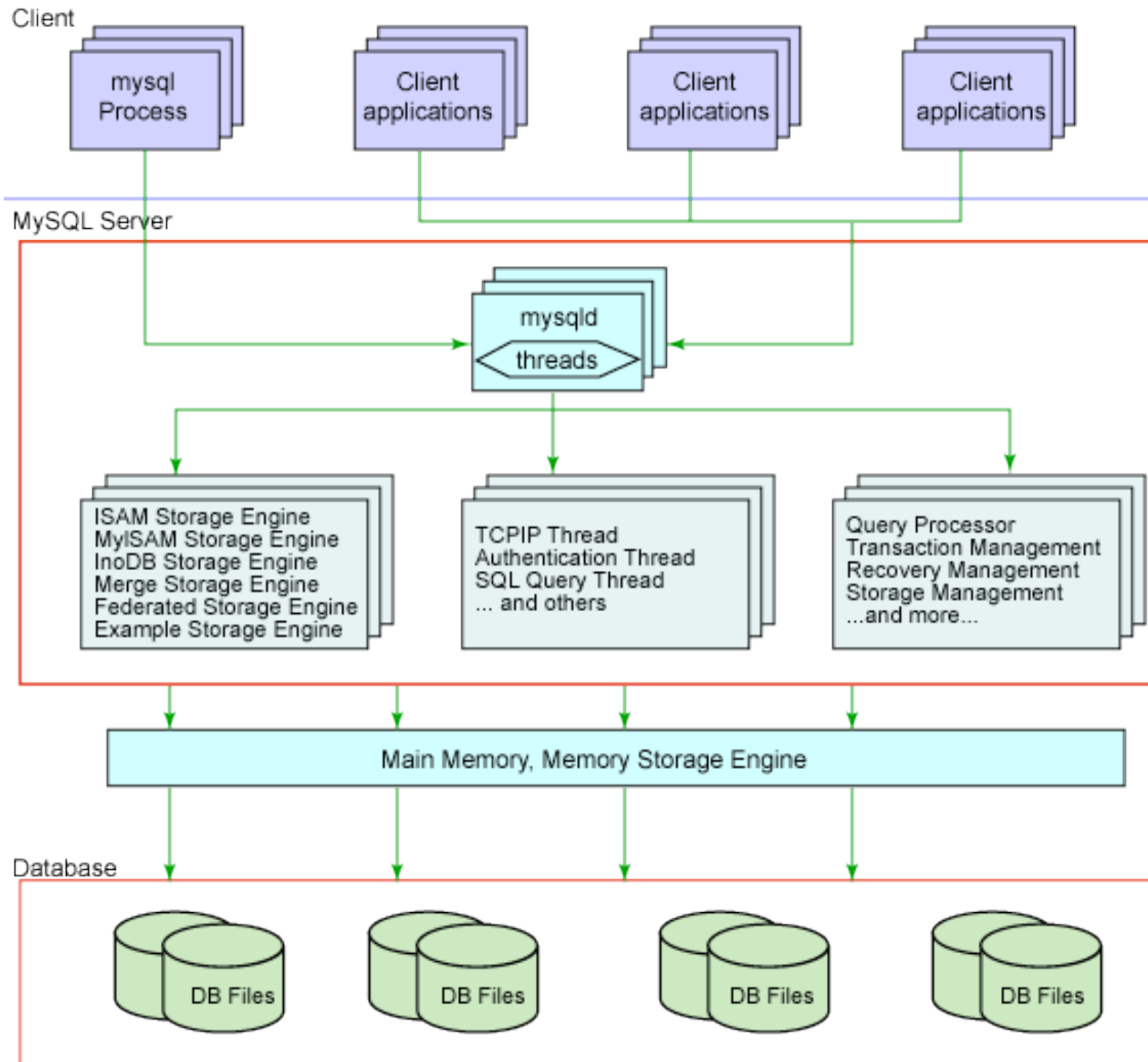
- embedded, in-process library
- very small

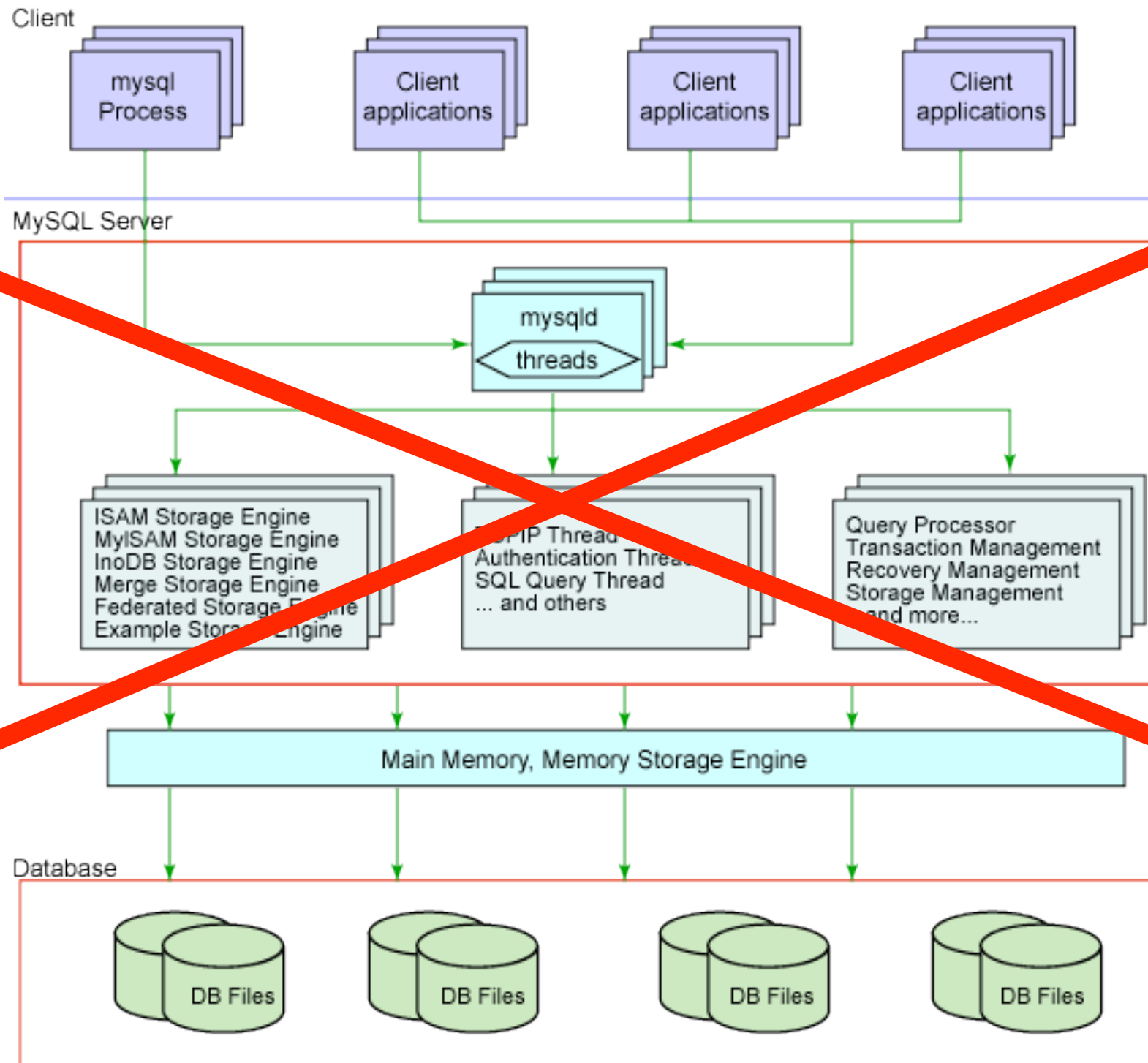
SQLite's different...

- embedded, in-process library
- very small
- very fast

SQLite's different...

- embedded, in-process library
- very small
- very fast
- fewer features than larger RDMBSs





SQLite's Popular...

SQLite's Popular...



SQLite's Popular...



SQLite's Popular...



Google™

SQLite's Popular...



McAfee®



Google™

SQLite's Popular...



McAfee®



Microsoft®

Google™

SQLite's Popular...



McAfee®



Microsoft®

Google™



SQLite's Popular...



McAfee®



Microsoft®

Google™

skype™

SQLite's Popular...



McAfee®



Microsoft®



Google™



SQLite's Popular...



McAfee®



Microsoft®



Google™



Extending SQLite

Extending SQLite

Aggregators

Extending SQLite

Aggregators

- `count()`, `avg()`

Extending SQLite

Aggregators

- `count()`, `avg()`

Functions

Extending SQLite

Aggregators

- `count()`, `avg()`

Functions

- `length()`, `substr()`

Extending SQLite

Aggregators

- `count()`, `avg()`

Functions

- `length()`, `substr()`

Collations

Extending SQLite

Aggregators

- `count()`, `avg()`

Functions

- `length()`, `substr()`

Collations

- `ORDER BY`

How to extend SQLite?

How to extend SQLite?

Traditional RDBMSs

- stored procedures

How to extend SQLite?

Traditional RDBMSs

- stored procedures

SQLite

- callbacks



SQLite Aggregate Functions

SQLite Aggregate Functions

`avg()`

SQLite Aggregate Functions

`avg()`

`count()`

SQLite Aggregate Functions

avg()

count()

max()

SQLite Aggregate Functions

avg()

count()

max()

min()

SQLite Aggregate Functions

avg()

count()

max()

min()

sum()

SQLite Aggregate Functions

avg()

count()

max()

stddev()?

min()

sum()

Add `stddev()` to
SQLite

Add stddev() to SQLite

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

```
# returns the standard deviation of a list
sub stddev {
    my $ar = shift;
    return 0 unless defined $ar && @$ar > 1;
    my ($mean, $sum_sqr) = (mean($ar), 0);
    for (@$ar) {
        $sum_sqr += ($mean - $_) * ($mean - $_)
    }
    return sqrt($sum_sqr / (@$ar - 1));
}
```

```
# returns the mean of a list
sub mean {
    my $ar = shift;
    my $sum = 0;
    $sum += $_ for @$ar;
    return @$ar > 0 ? $sum / @$ar : 0;
}
```

Create Aggregator Object

Create Aggregator Object

Need an object with 3 methods:

Create Aggregator Object

Need an object with 3 methods:

1. `new()`
initialization

Create Aggregator Object

Need an object with 3 methods:

1. `new()`
initialization
2. `step()`
called once for each row

Create Aggregator Object

Need an object with 3 methods:

1. `new()`
initialization
2. `step()`
called once for each row
3. `finalize()`
called after final row

```
package stddev_agg;
```

```
sub new { bless [], shift; }
```

```
sub step {  
    my ( $self, $value ) = @_;  
  
    push @$self, $value;  
}
```

```
sub finalize {  
    my $self = $_[0];  
    return stddev($self);  
}
```

```
package stddev_agg;
```

```
sub new { bless [], shift; }
```

```
sub step {  
    my ( $self, $value ) = @_;  
  
    push @$self, $value;  
}
```

```
sub finalize {  
    my $self = $_[0];  
    return stddev($self);  
}
```

```
package stddev_agg;

sub new { bless [], shift; }

sub step {
  my ( $self, $value ) = @_;

  push @$self, $value;
}

sub finalize {
  my $self = $_[0];
  return stddev($self);
}
```

```
package stddev_agg;

sub new { bless [], shift; }

sub step {
    my ( $self, $value ) = @_;

    push @$self, $value;
}

sub finalize {
    my $self = $_[0];
    return stddev($self);
}
```

Using the aggregator

```
$dbh->func ($name,  
           $argc,  
           $pkg,  
           "create_aggregate");
```

Using the aggregator

```
$dbh->func ($name,  
           $argc,  
           $pkg,  
           "create_aggregate");
```

Using the aggregator

```
$dbh->func ($name,  
           $argc,  
           $pkg,  
           "create_aggregate");
```

Using the aggregator

```
$dbh->func ($name,  
           $argc,  
           $pkg,  
           "create_aggregate") ;
```

Example

```
$dbh->func ("stddev",  
           1,  
           stddev_agg,  
           "create_aggregate");
```

```
my $sql = "  
  SELECT foo, avg(bar), stddev(bar)  
  FROM baz  
  GROUP BY foo";
```

```
my @r = $dbh->selectrow_array($sql);
```

Example

```
$dbh->func ("stddev",  
          1,  
          stddev_agg,  
          "create_aggregate");
```

```
my $sql = "  
  SELECT foo, avg(bar), stddev(bar)  
  FROM baz  
  GROUP BY foo";
```

```
my @r = $dbh->selectrow_array($sql);
```

That's cool, but...

That's cool, but...

Not so bad for one aggregator.

That's cool, but...

Not so bad for one aggregator.

What if I want to make a lot of them?

That's cool, but...

Not so bad for one aggregator.

What if I want to make a lot of them?

Do I have to do all that work for each one?

No.

Use inheritance

```
package DBD::SQLite::Aggregate::base;

sub new { bless [], shift; }

sub step {
    my ( $self, $value ) = @_;

    push @$self, $value;
}
}
```

```
package DBD::SQLite::Aggregate::stddev;  
use base 'DBD::SQLite::Aggregate::base';  
  
use Stats::Walt;  
  
sub finalize {  
    my $self = $_[0];  
    return stddev($self);  
}
```

```
package DBD::SQLite::Aggregate::median;  
use base 'DBD::SQLite::Aggregate::base';  
  
use Stats::Walt;  
  
sub finalize {  
    my $self = $_[0];  
    return median($self);  
}
```

```
$dbh->func (  
    "stddev",  
    1,  
    "DBD::SQLite::Aggregate::stddev",  
    "create_aggregate"  
);
```

```
$dbh->func (  
    "median",  
    1,  
    "DBD::SQLite::Aggregate::median",  
    "create_aggregate"  
);
```

New SQLite Functions

New SQLite Functions

- Same idea as aggregators

New SQLite Functions

- Same idea as aggregators
- Single elements instead of columns

New SQLite Functions

- Same idea as aggregators
- Single elements instead of columns
- Function refs instead of objects

```
$dbh->func('dnsname', 1, \&dnsname, 'create_function');  
find_ip($dbh);
```

```
sub find_ip {  
    my $dbh = shift;  
  
    my $sth = $dbh->prepare('SELECT addr, dnsname(addr)  
                             FROM ip_addresses');  
  
    $sth->execute;  
  
    while (my ($ip, $name) = $sth->fetchrow_array) {  
        print "$ip\t$name\n";  
    }  
}
```

```
sub dnsname {  
    my $ip = shift;  
    my $name = `dnsname $ip`; # from djbdns  
    chomp $name;  
    return $name;  
}
```

```
$dbh->func('dnsname', 1, \&dnsname, 'create_function');  
find_ip($dbh);
```

```
sub find_ip {  
    my $dbh = shift;  
  
    my $sth = $dbh->prepare('SELECT addr, dnsname(addr)  
                             FROM ip_addresses');  
  
    $sth->execute;  
  
    while (my ($ip, $name) = $sth->fetchrow_array) {  
        print "$ip\t$name\n";  
    }  
}
```

```
sub dnsname {  
    my $ip = shift;  
    my $name = `dnsname $ip`; # from djbdns  
    chomp $name;  
    return $name;  
}
```

```
$dbh->func('dnsname', 1, \&dnsname, 'create_function');  
find_ip($dbh);
```

```
sub find_ip {  
    my $dbh = shift;  
  
    my $sth = $dbh->prepare('SELECT addr, dnsname(addr)  
                             FROM ip_addresses');  
  
    $sth->execute;  
  
    while (my ($ip, $name) = $sth->fetchrow_array) {  
        print "$ip\t$name\n";  
    }  
}
```

```
sub dnsname {  
    my $ip = shift;  
    my $name = `dnsname $ip`; # from djbdns  
    chomp $name;  
    return $name;  
}
```

SQLite Collations

SQLite Collations

- Write your own ORDER BY functions

SQLite Collations

- Write your own ORDER BY functions
- Builtins:

SQLite Collations

- Write your own ORDER BY functions
- Builtins:
 - nocase

SQLite Collations

- Write your own ORDER BY functions
- Builtins:
 - nocase
 - binary

SQLite Collations

- Write your own ORDER BY functions
- Builtins:
 - nocase
 - binary
 - perlocale

SQLite Collations

- Write your own ORDER BY functions
- Builtins:
 - nocase
 - binary
 - perlocale
- It's just a sort comparison function

SQLite Collations

- Write your own ORDER BY functions
- Builtins:
 - nocase
 - binary
 - perlocale
- It's just a sort comparison function

```
$dbh->func ('rev',  
           \&rev,  
           "create_collation");
```

```
$dbh->func ('rev',  
           \&rev,  
           "create_collation");
```

```
sub rev {  
    return  
        reverse($_[0]) cmp  
        reverse($_[1]);  
}
```

```
$dbh->func ('rev',  
           \&rev,  
           "create_collation");
```

```
sub rev {  
    return  
        reverse($_[0]) cmp  
        reverse($_[1]);  
}
```

```
SELECT * FROM foo  
ORDER BY bar  
COLLATE rev
```

Caveats

Caveats

- New functions, aggregators and collations don't live in the database

Caveats

- New functions, aggregators and collations don't live in the database
- They only work within DBI

Caveats

- New functions, aggregators and collations don't live in the database
- They only work within DBI
 - (or C, Tcl, etc.)

Caveats

- New functions, aggregators and collations don't live in the database
- They only work within DBI
 - (or C, Tcl, etc.)
- Don't work in sqlite3

Caveats

- New functions, aggregators and collations don't live in the database
- They only work within DBI
 - (or C, Tcl, etc.)
- Don't work in sqlite3
- Could write your own sqlite3 using DBI...


```
$ ./sql_sel demo.sqlite 'select \  
avg(legs) from animals'
```

```
$ ./sql_sel demo.sqlite 'select \  
avg(legs) from animals'
```

```
4.5555555555555556
```

```
$ ./sql_sel demo.sqlite 'select \  
avg(legs) from animals'
```

```
4.5555555555555556
```

```
$ ./sql_sel demo.sqlite 'select \  
stddev(legs) from animals'
```

```
$ ./sql_sel demo.sqlite 'select \  
avg(legs) from animals'
```

```
4.5555555555555556
```

```
$ ./sql_sel demo.sqlite 'select \  
stddev(legs) from animals'
```

```
1.9436506316151
```

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query) ;
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

use Aggregate;

```
my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";
```

```
my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

```
use Aggregate;

my $db = shift @ARGV;
my $query = join " ", @ARGV;
my $dbh = DBI->connect("DBI:SQLite:$db", '', '',
                      {PrintError=>1, RaiseError=>1 } )
    or die "connecting: $DBI::errstr";

$dbh->func("stddev", 1,
  'DBD::SQLite::Aggregate::stddev',
  "create_aggregate" );

my $sth = $dbh->prepare($query);
$sth->execute;
while (my @f = $sth->fetchrow_array) {
    print join "\t", @f, "\n";
}
```

THANKS!