sqlite_table.pql

by *Pequel*

sample@youraddress.com

Sqlite Tables Example Script

2.2

Table of Contents
Sqlite Tables Example Script

SCRIPT NAME

sqlite_table.pql

DESCRIPTION

Demonstrates the use of external Sqlite tables.

1. PROCESS DETAILS

Input records are read from standard input. The input record contains ***8*** fields. Fields are delimited by the '|' character.

Output records are written to standard output. The output record contains ***8*** fields. Fields are delimited by the '|' character.

Input stream is **sorted** by the input field ***PRODUCT_CODE*** (*string*).

Input records are **grouped** by the input field ***PRODUCT_CODE*** (*string*).

1.1 *PRODUCT_CODE*

Output Field

Description
Set to input field ***PRODUCT_CODE***

1.2 *RECORD_COUNT*

Output Field

Description
Count aggregation.

1.3 *SALES_QTY_SAMPLE1*

Output Field

Description
Sum aggregation on input field ***SALES_QTY***.

Aggregation condition
exists %SAMPLE1(PRODUCT_CODE);

1.4 *S1_DESCRIPTION*

Output Field

Description
Set to input field ***S1_DESCRIPTION***

Derived Input Field Evaluation

```
=> %SAMPLE1(PRODUCT_CODE)->DESCRIPTION
```

1.5 *S1_LOCATION*

Output Field

Description

Set to input field ***S1_LOCATION***

Derived Input Field Evaluation

```
=> %SAMPLE1(PRODUCT_CODE)->LOCATION
```

1.6 *SALES_QTY_SAMPLE2*

Output Field

Description
Sum aggregation on input field ***SALES_QTY***.

Aggregation condition
exists %SAMPLE2(PRODUCT_CODE);

1.7 *S2_DESCRIPTION*

Output Field

Description
Set to input field ***S2_DESCRIPTION***

Derived Input Field Evaluation

```
=> %SAMPLE2(PRODUCT_CODE)->DESCRIPTION
```

1.8 *S2_LOCATION*

Output Field

Description
Set to input field ***S2_LOCATION***

Derived Input Field Evaluation

```
=> %SAMPLE2(PRODUCT_CODE)->LOCATION
```

2. CONFIGURATION SETTINGS

2.1 *pequeldoc*
generate pod / pdf pequel script Reference Guide.: pdf

2.2 *detail*
Include Pequel Generated Program chapter in Pequeldoc: 1

2.3 *script_name*
script filename: sqlite_table.pql

2.4 *header*
write header record to output.: 1

2.5 *optimize*
optimize generated code.: 1

2.6 *doc_title*
document title.: Sqlite Tables Example Script

2.7 *doc_email*
document email entry.: sample@youraddress.com

2.8 *doc_version*
document version for pequel script.: 2.2

2.9 *inline_cc*
Inline: CC: CC

2.10 *inline_force_build*
Inline: force_build: 1

2.11 *inline_optimize*
Inline: OPTIMIZE: -xO5 -xinline=%auto

2.12 *inline_ccflags*
Inline: CCFLAGS: -xchip=ultra3 -DSS_64BIT_SERVER -DBIT64 -DMACHINE64

3. TABLES

3.1 *SAMPLE1*

Table Type: ***sqlite***
Data Source Filename: ***sample.data***
Key Field Number: ***1***
Key Field Type: ***VARCHAR***
Database Filename: ***/_TABLE_SAMPLE1.sqlt***

3.1.1 *DESCRIPTION = 3*
3.1.2 *LOCATION = 8*

3.2 *SAMPLE2*

Table Type: ***sqlite merge***
Data Source Filename: ***sample.data***
Key Field Number: ***1***
Key Field Type: ***VARCHAR***
Database Filename: ***/_TABLE_SAMPLE2.sqlt***

3.2.1 *DESCRIPTION = 3*
3.2.2 *LOCATION = 8*

4. TABLE INFORMATION SUMMARY

4.1 Table List Sorted By Table Name

SAMPLE1 — ***1*** (*sqlite*)
SAMPLE2 — ***2*** (*sqlite merge*)

5. SQLITE_TABLE.PQL

options

```
pequeldoc(pdf)
detail(1)
script_name(sqlite_table.pql)
header(1)
optimize(1)
doc_title(Sqlite Tables Example Script)
doc_email(sample@youraddress.com)
doc_version(2.2)
inline_cc(CC)
inline_force_build(1)
inline_optimize(-xO5 -xinline=%auto)
inline_ccflags(-xchip=ultra3 -DSS_64BIT_SERVER -DBIT64 -DMACHINE64)
```

description

```
Demonstrates the use of external Sqlite tables.
```

load table sqlite

```
SAMPLE1 /* Table Name */ \
    sample.data /* Data Source Filename */ \
    1 /* Key Column Number */ \
    VARCHAR /* Key Type */ \
    DESCRIPTION = 3 \
    LOCATION = 8
```

load table sqlite merge

```
SAMPLE2 /* Table Name */ \
    sample.data /* Data Source Filename */ \
    1 /* Key Column Number */ \
    VARCHAR /* Key Type */ \
    DESCRIPTION = 3 \
    LOCATION = 8
```

sort by

```
PRODUCT_CODE string
```

group by

```
PRODUCT_CODE string
```

input section

```
PRODUCT_CODE
COST_PRICE
DESCRIPTION
SALES_CODE
SALES_PRICE
SALES_QTY
SALES_DATE
LOCATION
S1_DESCRIPTION => %SAMPLE1(PRODUCT_CODE)->DESCRIPTION

S1_LOCATION => %SAMPLE1(PRODUCT_CODE)->LOCATION

S2_DESCRIPTION => %SAMPLE2(PRODUCT_CODE)->DESCRIPTION

S2_LOCATION => %SAMPLE2(PRODUCT_CODE)->LOCATION
```

output section

```
string    PRODUCT_CODE      PRODUCT_CODE
numeric   RECORD_COUNT      count *
numeric   SALES_QTY_SAMPLE1 sum SALES_QTY where exists %SAMPLE1(PRODUCT_CODE)
string    S1_DESCRIPTION    S1_DESCRIPTION
string    S1_LOCATION       S1_LOCATION
numeric   SALES_QTY_SAMPLE2 sum SALES_QTY where exists %SAMPLE2(PRODUCT_CODE)
string    S2_DESCRIPTION    S2_DESCRIPTION
string    S2_LOCATION       S2_LOCATION
```

6. PEQUEL GENERATED PROGRAM

```
# vim: syntax=perl ts=4 sw=4
#-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
#Generated By: pequel Version 2.3-2, Build: Thursday September 29 19:56:03 BST 2005
#            : https://sourceforge.net/projects/pequel/
#Script Name : sqlite_table.pql
#Created On  : Thu Sep 29 14:05:33 2005
#For         :
#-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
#Options:
#pequeldoc(pdf) generate pod / pdf pequel script Reference Guide.
#detail(1) Include Pequel Generated Program chapter in Pequeldoc
#script_name(sqlite_table.pql) script filename
#header(1) write header record to output.
#optimize(1) optimize generated code.
#doc_title(Sqlite Tables Example Script) document title.
#doc_email(sample@youraddress.com) document email entry.
#doc_version(2.2) document version for pequel script.
#inline_cc(CC) Inline: CC
#inline_force_build(1) Inline: force_build
#inline_optimize(-xO5 -xinline=%auto) Inline: OPTIMIZE
#inline_ccflags(-xchip=ultra3 -DSS_64BIT_SERVER -DBIT64 -DMACHINE64) Inline: CCFLAGS
#-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
use strict;
use Fcntl;
local $\="\n"; local $,="|";
print STDERR '[sqlite_table.pql ' . localtime() . "] Init";
use constant VERBOSE => int 10000;
use constant LAST_ICELL => int 11;
my @I_VAL;
my @O_VAL;
my $key__I_PRODUCT_CODE;
my $previous_key__I_PRODUCT_CODE = undef;
foreach my $f (1..8) { $O_VAL[$f] = undef; }
&LoadTableSAMPLE2; # Create database for SAMPLE2
&LoadTableSAMPLE1; # Create database for SAMPLE1
use constant _I_PRODUCT_CODE         => int    0;
use constant _I_COST_PRICE           => int    1;
use constant _I_DESCRIPTION          => int    2;
use constant _I_SALES_CODE           => int    3;
use constant _I_SALES_PRICE          => int    4;
use constant _I_SALES_QTY            => int    5;
use constant _I_SALES_DATE           => int    6;
use constant _I_LOCATION             => int    7;
use constant _I_S1_DESCRIPTION       => int    8;
use constant _I_S1_LOCATION          => int    9;
use constant _I_S2_DESCRIPTION       => int   10;
use constant _I_S2_LOCATION          => int   11;
use constant _O_PRODUCT_CODE         => int    1;
use constant _O_RECORD_COUNT         => int    2;
use constant _O_SALES_QTY_SAMPLE1    => int    3;
use constant _O_S1_DESCRIPTION       => int    4;
use constant _O_S1_LOCATION          => int    5;
use constant _O_SALES_QTY_SAMPLE2    => int    6;
use constant _O_S2_DESCRIPTION       => int    7;
use constant _O_S2_LOCATION          => int    8;
use constant _T_SAMPLE2_FLD_DESCRIPTION   => int    0;
use constant _T_SAMPLE2_FLD_LOCATION      => int    1;
use constant _T_SAMPLE1_FLD_DESCRIPTION   => int    0;
use constant _T_SAMPLE1_FLD_LOCATION      => int    1;
use constant _I_SAMPLE1_PRODUCT_CODE_FLD_KEY          => int   12;
use constant _I_SAMPLE1_PRODUCT_CODE_FLD_DESCRIPTION  => int   13;
use constant _I_SAMPLE1_PRODUCT_CODE_FLD_LOCATION     => int   14;
use constant _I_SAMPLE2_PRODUCT_CODE_FLD_KEY          => int   15;
use constant _I_SAMPLE2_PRODUCT_CODE_FLD_DESCRIPTION  => int   16;
use constant _I_SAMPLE2_PRODUCT_CODE_FLD_LOCATION     => int   17;
open(DATA, q{cat - | sort -t'|' -y -k 1,1 |}) || die "Cannot open input: $!";
my $fd = fileno(DATA);
OpenSortStream($fd);
&PrintHeader();
print STDERR '[sqlite_table.pql ' . localtime() . "] Start";
use Benchmark;
my $benchmark_start = new Benchmark;
SqliteConnect($fd);
print STDERR '[sqlite_table.pql ' . localtime() . "] Tables opened.";
my $i;
while (readsplit(\@I_VAL))
{
    ++$i;
    print STDERR '[sqlite_table.pql ' . localtime() . "] $i records." if ($i % VERBOSE == 0);
    $key__I_PRODUCT_CODE = $I_VAL[_I_PRODUCT_CODE];
```

```
        if (!defined($previous_key__I_PRODUCT_CODE))
        {
            $previous_key__I_PRODUCT_CODE = $key__I_PRODUCT_CODE;
        }

        elsif ($previous_key__I_PRODUCT_CODE ne $key__I_PRODUCT_CODE)
        {
            print
                $O_VAL[_O_PRODUCT_CODE],
                $O_VAL[_O_RECORD_COUNT],
                $O_VAL[_O_SALES_QTY_SAMPLE1],
                $O_VAL[_O_S1_DESCRIPTION],
                $O_VAL[_O_S1_LOCATION],
                $O_VAL[_O_SALES_QTY_SAMPLE2],
                $O_VAL[_O_S2_DESCRIPTION],
                $O_VAL[_O_S2_LOCATION]
            ;
            $previous_key__I_PRODUCT_CODE = $key__I_PRODUCT_CODE;
            @O_VAL = undef;
        }

        $O_VAL[_O_PRODUCT_CODE] = $I_VAL[_I_PRODUCT_CODE];
        $O_VAL[_O_RECORD_COUNT]++;
        $I_VAL[_I_S1_DESCRIPTION] = $I_VAL[_I_SAMPLE1_PRODUCT_CODE_FLD_DESCRIPTION];
        $O_VAL[_O_S1_DESCRIPTION] = $I_VAL[_I_S1_DESCRIPTION];
        $I_VAL[_I_S1_LOCATION] = $I_VAL[_I_SAMPLE1_PRODUCT_CODE_FLD_LOCATION];
        $O_VAL[_O_S1_LOCATION] = $I_VAL[_I_S1_LOCATION];
        $I_VAL[_I_S2_DESCRIPTION] = $I_VAL[_I_SAMPLE2_PRODUCT_CODE_FLD_DESCRIPTION];
        $O_VAL[_O_S2_DESCRIPTION] = $I_VAL[_I_S2_DESCRIPTION];
        $I_VAL[_I_S2_LOCATION] = $I_VAL[_I_SAMPLE2_PRODUCT_CODE_FLD_LOCATION];
        $O_VAL[_O_S2_LOCATION] = $I_VAL[_I_S2_LOCATION];

        if (exists $I_VAL[_I_SAMPLE1_PRODUCT_CODE_FLD_KEY] ne '') {
            $O_VAL[_O_SALES_QTY_SAMPLE1] += $I_VAL[_I_SALES_QTY] unless ($I_VAL[_I_SALES_QTY] eq '');
        }

        if (exists $I_VAL[_I_SAMPLE2_PRODUCT_CODE_FLD_KEY] ne '') {
            $O_VAL[_O_SALES_QTY_SAMPLE2] += $I_VAL[_I_SALES_QTY] unless ($I_VAL[_I_SALES_QTY] eq '');
        }
    }

    print
        $O_VAL[_O_PRODUCT_CODE],
        $O_VAL[_O_RECORD_COUNT],
        $O_VAL[_O_SALES_QTY_SAMPLE1],
        $O_VAL[_O_S1_DESCRIPTION],
        $O_VAL[_O_S1_LOCATION],
        $O_VAL[_O_SALES_QTY_SAMPLE2],
        $O_VAL[_O_S2_DESCRIPTION],
        $O_VAL[_O_S2_LOCATION]
    ;
    SqliteDisconnect();
    print STDERR '[sqlite_table.pql ' . localtime() . "] $i records.";
    my $benchmark_end = new Benchmark;
    my $benchmark_timediff = timediff($benchmark_start, $benchmark_end);
    print STDERR '[sqlite_table.pql ' . localtime() . "] Code statistics: @{[timestr($benchmark_timediff)]}";
    #-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    #++++++ Table SAMPLE1 --> Type :Pequel::Type::Table::Sqlite ++++++
    sub LoadTableSAMPLE1
    {
        my %_TABLE_SAMPLE1;
        print STDERR '[sqlite_table.pql ' . localtime() . "] Loading (lookup) table SAMPLE1 from sample.data...";
        my $exists = (-e "/_TABLE_SAMPLE1.sqlt");
        use DBI;
        my $dbh = DBI->connect
        (
            "dbi:SQLite:dbname=/_TABLE_SAMPLE1.sqlt", '', '',
            { RowCacheSize => 5000, RaiseError => 0, AutoCommit => 0 }
        )

        or die "Cannot open /_TABLE_SAMPLE1.sqlt:$!";
        $dbh->{PrintError} = 0;
        $dbh->do("PRAGMA synchronous=OFF");
        $dbh->do("PRAGMA count_changes=OFF");
        $dbh->do("PRAGMA full_column_names=OFF");
        my $sqlite = $dbh->{sqlite_version};
        if (!$exists)
        {
            print STDERR '[sqlite_table.pql ' . localtime() . "] Creating table SAMPLE1 from sample.data...";
            my @flist =
            (
                'key',
                'description',
                'location'
            );
```

```
        my %flist =
        (
            key => 'VARCHAR NOT NULL PRIMARY KEY',
            description => 'varchar',
            location => 'varchar'
        );

        $dbh->do("CREATE TABLE SAMPLE1 ( @{[ join(',', map(qq{$_ $flist{$_}}, @flist)) ]} )");
        $dbh->commit;
        print STDERR '[sqlite_table.pql ' . localtime() . "] Loading sqlite (v$sqlite) table SAMPLE1 from samp
le.data...";
        open(SAMPLE1, "sort -u -t'|' -k 1 sample.data |");
        $dbh->do("BEGIN;");
        my $sth = $dbh->prepare("INSERT INTO SAMPLE1 ( @{[ join(', ', @flist) ]} ) VALUES( @{[ join(', ', map(
'?', @flist)) ]} )");
        while (<SAMPLE1>)
        {
            chomp;
            my (@flds) = split("[|]", $_, -1);
            $sth->execute($flds[0], @flds[ 2,7 ]);
            print STDERR '[sqlite_table.pql ' . localtime() . "] Table SAMPLE1 loaded $. records..." if ($. %
100000 == 0);
        }

        close(SAMPLE1);
        $sth->finish;
        $dbh->commit;
    }

    my $sth = $dbh->prepare("SELECT count(1) FROM SAMPLE1");
    $sth->execute;
    my $loaded = $sth->fetchrow_array;
    $sth->finish;
    $dbh->disconnect;
    print STDERR '[sqlite_table.pql ' . localtime() . "] $loaded records for table SAMPLE1.";
}

#++++++ Table SAMPLE2 --> Type :Pequel::Type::Table::Sqlite::Merge ++++++
sub LoadTableSAMPLE2
{
    my %_TABLE_SAMPLE2;
    print STDERR '[sqlite_table.pql ' . localtime() . "] Loading (merge) table SAMPLE2 from sample.data...";
    my $exists = (-e "/_TABLE_SAMPLE2.sqlt");
    use DBI;
    my $dbh = DBI->connect
    (
        "dbi:SQLite:dbname=/_TABLE_SAMPLE2.sqlt", '', '',
        { RowCacheSize => 5000, RaiseError => 0, AutoCommit => 0 }
    )

    or die "Cannot open /_TABLE_SAMPLE2.sqlt:$!";
    $dbh->{PrintError} = 0;
    $dbh->do("PRAGMA synchronous=OFF");
    $dbh->do("PRAGMA count_changes=OFF");
    $dbh->do("PRAGMA full_column_names=OFF");
    my $sqlite = $dbh->{sqlite_version};
    if (!$exists)
    {
        print STDERR '[sqlite_table.pql ' . localtime() . "] Creating table SAMPLE2 from sample.data...";
        my @flist =
        (
            'key',
            'description',
            'location'
        );

        my %flist =
        (
            key => 'VARCHAR NOT NULL PRIMARY KEY',
            description => 'varchar',
            location => 'varchar'
        );

        $dbh->do("CREATE TABLE SAMPLE2 ( @{[ join(',', map(qq{$_ $flist{$_}}, @flist)) ]} )");
        $dbh->commit;
        print STDERR '[sqlite_table.pql ' . localtime() . "] Loading sqlite (v$sqlite) table SAMPLE2 from samp
le.data...";
        open(SAMPLE2, "sort -u -t'|' -k 1 sample.data |");
        $dbh->do("BEGIN;");
        my $sth = $dbh->prepare("INSERT INTO SAMPLE2 ( @{[ join(', ', @flist) ]} ) VALUES( @{[ join(', ', map(
'?', @flist)) ]} )");
        while (<SAMPLE2>)
        {
            chomp;
            my (@flds) = split("[|]", $_, -1);
```

```
            $sth->execute($flds[0], @flds[ 2,7 ]);
            print STDERR '[sqlite_table.pql ' . localtime() . "] Table SAMPLE2 loaded $. records..." if ($. %
100000 == 0);
        }

        close(SAMPLE2);
        $sth->finish;
        $dbh->commit;
    }

    my $sth = $dbh->prepare("SELECT count(1) FROM SAMPLE2");
    $sth->execute;
    my $loaded = $sth->fetchrow_array;
    $sth->finish;
    $dbh->disconnect;
    print STDERR '[sqlite_table.pql ' . localtime() . "] $loaded records for table SAMPLE2.";
}

sub PrintHeader
{
    local $\="\n";
    local $,="|";
    print
        'PRODUCT_CODE',
        'RECORD_COUNT',
        'SALES_QTY_SAMPLE1',
        'S1_DESCRIPTION',
        'S1_LOCATION',
        'SALES_QTY_SAMPLE2',
        'S2_DESCRIPTION',
        'S2_LOCATION'
        ;
}

#-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
#***** I N L I N E *****
use Inline
    C => Config =>
    NAME => 'sqlite_table',
    CC => 'CC',
    CLEAN_AFTER_BUILD => '1',
    CLEAN_BUILD_AREA => '1',
    PRINT_INFO => '0',
    BUILD_NOISY => '0',
    BUILD_TIMERS => '0',
    FORCE_BUILD => '1',
    LIBS => '  -L/bld/.libs -lpthread -lexc -lsqlite',
    INC => '  -I/bld -I/src',
    CCFLAGS => '-xchip=ultra3 -DSS_64BIT_SERVER -DBIT64 -DMACHINE64  -DNDEBUG=1',
    OPTIMIZE => '-xO5 -xinline=%auto  '
    ;

use Inline C => q~

#include "sqlite.h"
#include "sqliteInt.h"

#define GFMAXPIPBUFFER      3072
#define GFMAXPIPFLDS        13
#define GFCACHERECS         100

#define _I_PRODUCT_CODE            0
#define _I_COST_PRICE              1
#define _I_DESCRIPTION             2
#define _I_SALES_CODE              3
#define _I_SALES_PRICE             4
#define _I_SALES_QTY               5
#define _I_SALES_DATE              6
#define _I_LOCATION                7
#define _I_S1_DESCRIPTION          8
#define _I_S1_LOCATION             9
#define _I_S2_DESCRIPTION         10
#define _I_S2_LOCATION            11

#define _I_SAMPLE1_PRODUCT_CODE_FLD_KEY            12
#define _I_SAMPLE1_PRODUCT_CODE_FLD_DESCRIPTION    13
#define _I_SAMPLE1_PRODUCT_CODE_FLD_LOCATION       14
#define _I_SAMPLE2_PRODUCT_CODE_FLD_KEY            15
#define _I_SAMPLE2_PRODUCT_CODE_FLD_DESCRIPTION    16
#define _I_SAMPLE2_PRODUCT_CODE_FLD_LOCATION       17

static sqlite *db_SAMPLE2 = 0;
sqlite_vm *ppVm_SAMPLE2_PRODUCT_CODE;

static sqlite *db_SAMPLE1 = 0;
```

```
sqlite_vm *ppVm_SAMPLE1_PRODUCT_CODE;

int sqlite_open_all();
int sqlite_pragma_all();
int sqlite_prep_all();

static const char *fields[GFCACHERECS][GFMAXPIPFLDS];
static FILE *fstream = (FILE*)0;

sqlite *sqlite_open_db (char *DbFilename)
{

    sqlite *db = 0;

    if ( !sqliteOsFileExists(DbFilename) )
    {
        fprintf(stderr,"Database %s does not exist\n", DbFilename);
        return 0;
    }
    char *zErrMsg = 0;
    if ((db = sqlite_open(DbFilename, 0666, &zErrMsg)) == 0)
    {
        if ((db = sqlite_open(DbFilename, 0444, &zErrMsg)) == 0)
        {
            if ( zErrMsg )
            {
                fprintf(stderr,"Unable to open database %s: %s\n", DbFilename, zErrMsg);
                freemem(zErrMsg);
            }
            else
            {
                fprintf(stderr,"Unable to open database %s\n", DbFilename);
            }
            return 0;
        }
        else
        {
            fprintf(stderr,"Database %s opened READ ONLY!\n", DbFilename);
        }
    }
    return db;

}
int SqliteConnect (int fd)
{
    sqlite_open_all();
    sqlite_pragma_all();
    sqlite_prep_all();
    fstream = fdopen(fd, "r");
    return 1;
}

int sqlite_open_all ()
{
    char *pzErrMsg = 0;
//-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//++++++ Table SAMPLE2 --> Type :Pequel::Type::Table::Sqlite::Merge ++++++
    if ((db_SAMPLE2 = sqlite_open_db("/_TABLE_SAMPLE2.sqlt")) == 0)
    {
        return 0;
    }

//-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//++++++ Table SAMPLE1 --> Type :Pequel::Type::Table::Sqlite ++++++
    if ((db_SAMPLE1 = sqlite_open_db("/_TABLE_SAMPLE1.sqlt")) == 0)
    {
        return 0;
    }

    return 1;
}

int sqlite_pragma_all ()
{
    char *pzErrMsg = 0;
    register int ret;
//++++++ Table SAMPLE2 --> Type :Pequel::Type::Table::Sqlite::Merge ++++++
    if ((ret = sqlite_exec(db_SAMPLE2, "PRAGMA count_changes = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE2: Cannot execute PRAGMA count_changes=OFF (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE2, "PRAGMA empty_result_callbacks = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
```

```
    {
        fprintf(stderr, "** db_SAMPLE2: Cannot execute PRAGMA empty_result_callbacks=OFF (%d-%s)\n", ret, pzEr
rMsg);
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE2, "PRAGMA full_column_names = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE2: Cannot execute PRAGMA full_column_names=OFF (%d-%s)\n", ret, pzErrMsg)
;
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE2, "PRAGMA show_datatypes = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE2: Cannot execute PRAGMA show_datatypes=OFF (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE2, "PRAGMA synchronous = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE2: Cannot execute PRAGMA synchronous=OFF (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

//++++++ Table SAMPLE1 --> Type :Pequel::Type::Table::Sqlite ++++++
    if ((ret = sqlite_exec(db_SAMPLE1, "PRAGMA count_changes = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE1: Cannot execute PRAGMA count_changes=OFF (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE1, "PRAGMA empty_result_callbacks = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE1: Cannot execute PRAGMA empty_result_callbacks=OFF (%d-%s)\n", ret, pzEr
rMsg);
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE1, "PRAGMA full_column_names = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE1: Cannot execute PRAGMA full_column_names=OFF (%d-%s)\n", ret, pzErrMsg)
;
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE1, "PRAGMA show_datatypes = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE1: Cannot execute PRAGMA show_datatypes=OFF (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    if ((ret = sqlite_exec(db_SAMPLE1, "PRAGMA synchronous = OFF;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE1: Cannot execute PRAGMA synchronous=OFF (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    return 1;
}

int sqlite_prep_all ()
{
    char *pzErrMsg = 0;
    char sql[4096];
    register int ret;
//++++++ Table SAMPLE2 --> Type :Pequel::Type::Table::Sqlite::Merge ++++++
    if ((ret = sqlite_exec(db_SAMPLE2, "BEGIN TRANSACTION ON CONFLICT ABORT;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE2: Cannot execute BEGIN statement (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    sprintf(sql, "select key, description, location from SAMPLE2 order by key");
    if (sqlite_compile(db_SAMPLE2, sql, 0, &ppVm_SAMPLE2_PRODUCT_CODE, &pzErrMsg) != SQLITE_OK)
```

```
    {
        fprintf(stderr, "** Error compiling sql for db_SAMPLE2->SAMPLE2_PRODUCT_CODE (%s)\n", pzErrMsg);
        return 0;
    }

//++++++ Table SAMPLE1 --> Type :Pequel::Type::Table::Sqlite ++++++
    if ((ret = sqlite_exec(db_SAMPLE1, "BEGIN TRANSACTION ON CONFLICT ABORT;", 0, 0, &pzErrMsg)) != SQLITE_OK)
    {
        fprintf(stderr, "** db_SAMPLE1: Cannot execute BEGIN statement (%d-%s)\n", ret, pzErrMsg);
        freemem(pzErrMsg);
        return 0;
    }

    sprintf(sql, "select key, description, location from SAMPLE1 where key = ?");
    if (sqlite_compile(db_SAMPLE1, sql, 0, &ppVm_SAMPLE1_PRODUCT_CODE, &pzErrMsg) != SQLITE_OK)
    {
        fprintf(stderr, "** Error compiling sql for db_SAMPLE1->SAMPLE1_PRODUCT_CODE (%s)\n", pzErrMsg);
        return 0;
    }

    return 1;
}

void SqliteDisconnect ()
{
    sqlite_exec(db_SAMPLE2, "END;", 0, 0, 0);sqlite_close(db_SAMPLE2);
    sqlite_exec(db_SAMPLE1, "END;", 0, 0, 0);sqlite_close(db_SAMPLE1);
}

int readcache ()
{

    register char *p;
    register int recs;
    register int f;
    static char inp[GFCACHERECS][GFMAXPIPBUFFER];
    static eof=0;

    if (eof) return 0;
    recs = 0;
    while (recs < GFCACHERECS)
    {
        f=0;
        if (!fgets(inp[recs], GFMAXPIPBUFFER, fstream) ) { eof=1; return recs; }
        inp[recs][strlen(inp[recs])-1] = '\0';
        memset(fields[recs], 0, sizeof(fields[recs]));
        p = inp[recs];
        fields[recs][0] = p;
        while (*p)
        {
            if (*p == '|')
            {
                fields[recs][++f] = p + 1;
                *p = '\0';
            }
            p++;
        }
        recs++;
    }
    return recs;

}
int readsplit (SV* I_VAL_ref)
{
    char sql[4096];
    int ret;
    char *pzErrMsg = 0;
    static int current_cache_maxrecs=GFCACHERECS;
    static int current_cache_rec=GFCACHERECS;
    if (++current_cache_rec >= current_cache_maxrecs)
    {
        if (current_cache_maxrecs < GFCACHERECS) return 0;
        if ((current_cache_maxrecs = readcache()) == 0) return 0;
        current_cache_rec = 0;
    }

    register AV* I_VAL = (AV*)SvRV(I_VAL_ref);
    if (!SvROK(I_VAL_ref)) croak("I_VAL_ref is not a reference");
    av_clear(I_VAL);
    register int f=0;
    for (f=0; f < GFMAXPIPFLDS; f++)
    {
        if (fields[current_cache_rec][f] == 0) av_store(I_VAL, f, newSVpvn("", 0));
        else av_store(I_VAL, f, newSVpvn(fields[current_cache_rec][f], strlen(fields[current_cache_rec][f])));
    }
```

```
    int pN;
    static const char **pazValue_SAMPLE2_PRODUCT_CODE;
    static int last_step_SAMPLE2_PRODUCT_CODE = SQLITE_ROW;
    static const char **pazValue_SAMPLE1_PRODUCT_CODE;
//++++++ Table SAMPLE1 (PRODUCT_CODE) --> Type :Pequel::Type::Table::Sqlite ++++++
    sqlite_reset(ppVm_SAMPLE1_PRODUCT_CODE, 0);
    if ((ret = sqlite_bind(ppVm_SAMPLE1_PRODUCT_CODE, 1, fields[current_cache_rec][_I_PRODUCT_CODE], -1, 0)) !
= SQLITE_OK)
    {
        fprintf(stderr, "** Error binding to db_SAMPLE1->SAMPLE1_PRODUCT_CODE (%s)\n", sqlite_error_string(ret
));
        croak("exiting");
    }

    if (sqlite_step(ppVm_SAMPLE1_PRODUCT_CODE, &pN, &pazValue_SAMPLE1_PRODUCT_CODE, 0) == SQLITE_ROW)
    {
        av_store(I_VAL, _I_SAMPLE1_PRODUCT_CODE_FLD_KEY, newSVpvn(pazValue_SAMPLE1_PRODUCT_CODE[0], strlen(paz
Value_SAMPLE1_PRODUCT_CODE[0])));
        av_store(I_VAL, _I_SAMPLE1_PRODUCT_CODE_FLD_DESCRIPTION, newSVpvn(pazValue_SAMPLE1_PRODUCT_CODE[1], st
rlen(pazValue_SAMPLE1_PRODUCT_CODE[1])));
        av_store(I_VAL, _I_SAMPLE1_PRODUCT_CODE_FLD_LOCATION, newSVpvn(pazValue_SAMPLE1_PRODUCT_CODE[2], strle
n(pazValue_SAMPLE1_PRODUCT_CODE[2])));
    }

//++++++ Table SAMPLE2 (PRODUCT_CODE) --> Type :Pequel::Type::Table::Sqlite::Merge ++++++
    if (pazValue_SAMPLE2_PRODUCT_CODE == 0 && last_step_SAMPLE2_PRODUCT_CODE == SQLITE_ROW)
    {
        last_step_SAMPLE2_PRODUCT_CODE = sqlite_step(ppVm_SAMPLE2_PRODUCT_CODE, &pN, &pazValue_SAMPLE2_PRODUCT
_CODE, 0);
    }

    while
    (
        last_step_SAMPLE2_PRODUCT_CODE == SQLITE_ROW
        && pazValue_SAMPLE2_PRODUCT_CODE != 0
        && strcmp(fields[current_cache_rec][_I_PRODUCT_CODE], pazValue_SAMPLE2_PRODUCT_CODE[0]) > 0
        && (last_step_SAMPLE2_PRODUCT_CODE = sqlite_step(ppVm_SAMPLE2_PRODUCT_CODE, &pN, &pazValue_SAMPLE2_PRO
DUCT_CODE, 0)) == SQLITE_ROW
    )

    {
        if (pazValue_SAMPLE2_PRODUCT_CODE == 0) break;
        if (strcmp(fields[current_cache_rec][_I_PRODUCT_CODE], pazValue_SAMPLE2_PRODUCT_CODE[0]) <= 0) break;
    }

    if (pazValue_SAMPLE2_PRODUCT_CODE != 0 && strcmp(fields[current_cache_rec][_I_PRODUCT_CODE], pazValue_SAMP
LE2_PRODUCT_CODE[0]) == 0)
    {
        av_store(I_VAL, _I_SAMPLE2_PRODUCT_CODE_FLD_KEY, newSVpvn(pazValue_SAMPLE2_PRODUCT_CODE[0], strlen(paz
Value_SAMPLE2_PRODUCT_CODE[0])));
        av_store(I_VAL, _I_SAMPLE2_PRODUCT_CODE_FLD_DESCRIPTION, newSVpvn(pazValue_SAMPLE2_PRODUCT_CODE[1], st
rlen(pazValue_SAMPLE2_PRODUCT_CODE[1])));
        av_store(I_VAL, _I_SAMPLE2_PRODUCT_CODE_FLD_LOCATION, newSVpvn(pazValue_SAMPLE2_PRODUCT_CODE[2], strle
n(pazValue_SAMPLE2_PRODUCT_CODE[2])));
    }

    return 1;
}

int OpenSortStream (int fd)
{
    if ((fstream = fdopen(fd, "r")) == (FILE*)0)
        croak("sqlite_table.pql:Unable to open input file stream.");
    return 1;
}

~; #End of Inline-C Code
```

7. ABOUT PEQUEL

This document was generated by Pequel.

https://sourceforge.net/projects/pequel/