Build Your Own SQLite3

hasumikin

RubyKaigi 2023

Mar. 13, 2023

How do you pronounce "SQLite"?

- /eskjuːel-aɪt/ エスキューエルアイト
- /eskjuːelaɪt/ エスキューエライト
- /eskjuːlaɪt/ エスキューライト
- /siːkuəl-aɪt/ スィークエルアイト

Just curious. Answer on
https://twitter.com/hasumikin

SQLite3 - Features

- Public domain
  - No limitation for commercial use
- Single-file database. No client-server style
  - Easy to backup
  - High performance while low resource usage

Good for embedded system

Embedded SQLite3 in

- Car navigation system
- iOS | Android applications
- Web applications such as Rails
  - Master data such as postal code
- Browsers like Google Chrome
  - "Local" DB for browser apps
- **One-chip microcontroller**

Embedded SQLite3 in

- Car navigation system
- iOS | Android applications
- Web applications such as Rails
  - Master data such as postal code

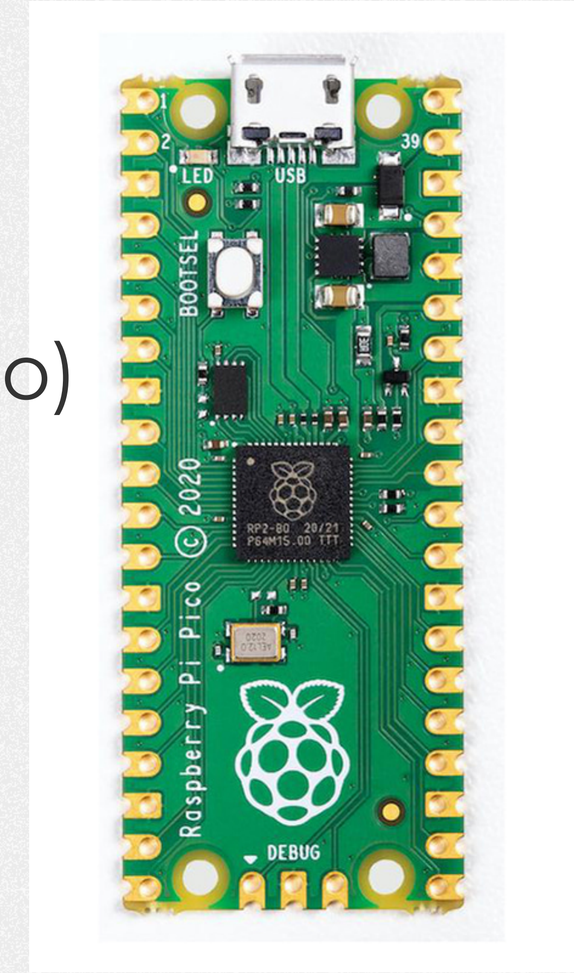
with OS

- Browsers like Google Chrome
  - "Local" DB for browser apps
- **One-chip microcontroller**

without OS

Target device and OS

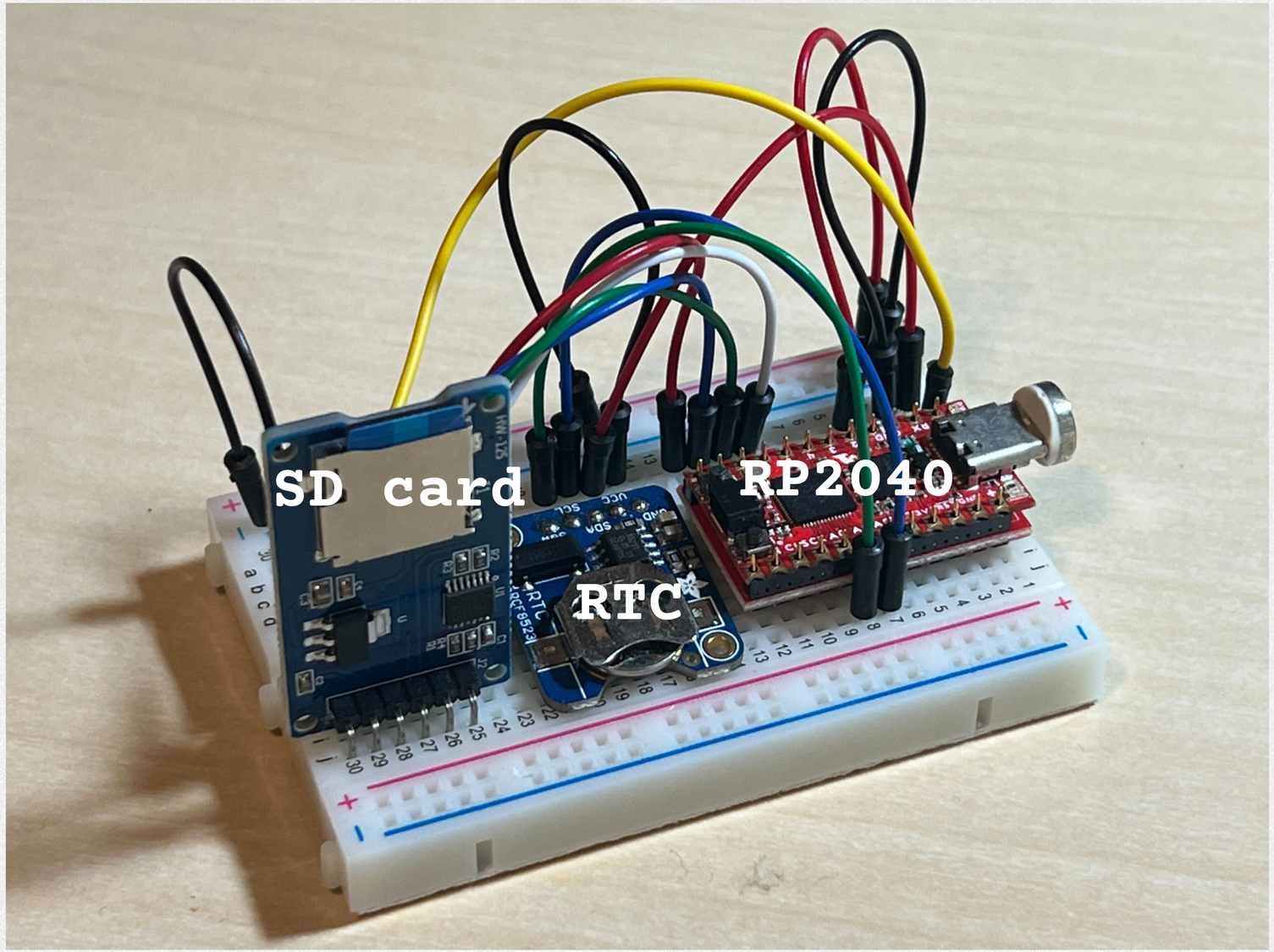
- RP2040
  - RAM: 264 KB
  - CPU: 32 bit Cortex-M0+ (dual)
  - Flash ROM: 2 MB (Raspberry Pi Pico)
- Bare metal (No OS)
- Instead, **PicoRuby** takes care of everything



self.inspect

- hasumikin | ハスミキン(Twitter, GitHub)
  - Creator of PicoRuby and PRK Firmware
  - Committer of mruby/c
  - Maintainer of IRB and Reline
- First prize of Fukuoka Ruby Award (2020 and 2022✌)
- A final nominee of Ruby Prize 2021
- Monstarlab Shimane Dev. branch

Demo



Demo (review or fallback)

- R2P2 (Unix-like shell of PicoRuby)
- PicoIRB
- require 'sqilte3'
- db = SQLite3::database.new 'test.db'
- Interoperable database file between the laptop and the microcontroller

Building an SQLite3 shared library

```
# Download and unzip sqlite3.c and sqlite3.h
# Then,
# Typical build on Unix systems
gcc -shared -fPIC sqlite3.c -o libsqlite3.so

# On Windows (Microsoft Visual C++)
cl sqlite3.c -link -dll -out:sqlite3.dll
```

Easy-peasy 🍋

Compile-time options

```
# Platform Configuration
_HAVE_SQLITE_CONFIG_H
HAVE_FDATASYNC
HAVE_GMTIME_R
HAVE_ISNAN
HAVE_LOCALTIME_R
HAVE_LOCALTIME_S
HAVE_MALLOC_USABLE_SIZE
HAVE_STRCHRNUL
HAVE_UTIME
SQLITE_BYTEORDER=(0|1234|4321)

# Options To Set Default Parameter Values
SQLITE_DEFAULT_AUTOMATIC_INDEX=<0 or 1>
See also: SQLITE_OMIT_AUTOMATIC_INDEX
SQLITE_DEFAULT_AUTOVACUUM=<0 or 1 or 2>
SQLITE_DEFAULT_CACHE_SIZE=<N>
SQLITE_DEFAULT_FILE_FORMAT=<1 or 4>
SQLITE_DEFAULT_FILE_PERMISSIONS=N
SQLITE_DEFAULT_FOREIGN_KEYS=<0 or 1>
SQLITE_DEFAULT_MMAP_SIZE=N
SQLITE_DEFAULT_JOURNAL_SIZE_LIMIT=<bytes>
SQLITE_DEFAULT_LOCKING_MODE=<1 or 0>
SQLITE_DEFAULT_LOOKASIDE=SZ,N
SQLITE_DEFAULT_MEMSTATUS=<1 or 0>
SQLITE_DEFAULT_PCACHE_INITSZ=N
SQLITE_DEFAULT_PAGE_SIZE=<bytes>
SQLITE_DEFAULT_SYNCHRONOUS=<0-3>
SQLITE_DEFAULT_WAL_SYNCHRONOUS=<0-3>
SQLITE_DEFAULT_WAL_AUTOCHECKPOINT=<pages>
SQLITE_DEFAULT_WORKER_THREADS=N
SQLITE_DQS=N
SQLITE_EXTRA_DURABLE
SQLITE_FTS3_MAX_EXPR_DEPTH=N
SQLITE_LIKE_DOESNT_MATCH_BLOBS
SQLITE_MAX_MEMORY=N
SQLITE_MAX_MMAP_SIZE=N
SQLITE_MAX_SCHEMA_RETRY=N
SQLITE_MAX_WORKER_THREADS=N
SQLITE_MEMDB_DEFAULT_MAXSIZE=N
SQLITE_MINIMUM_FILE_DESCRIPTOR=N
SQLITE_POWERSAFE_OVERWRITE=<0 or 1>
SQLITE_PRINTF_PRECISION_LIMIT=N
SQLITE_QUERY_PLANNER_LIMIT=N
SQLITE_QUERY_PLANNER_LIMIT_INCR=N
SQLITE_REVERSE_UNORDERED_SELECTS
SQLITE_SORTER_PMASZ=N
SQLITE_STMTJRNL_SPILL=N
SQLITE_WIN32_MALLOC
YYSTACKDEPTH=<max_depth>

# Options To Set Size Limits
SQLITE_MAX_ATTACHED
SQLITE_MAX_COLUMN
SQLITE_MAX_COMPOUND_SELECT
SQLITE_MAX_EXPR_DEPTH
SQLITE_MAX_FUNCTION_ARG
SQLITE_MAX_LENGTH
SQLITE_MAX_LIKE_PATTERN_LENGTH
SQLITE_MAX_PAGE_COUNT
SQLITE_MAX_SQL_LENGTH
SQLITE_MAX_VARIABLE_NUMBER

# Options To Control Operating Characteristics
SQLITE_4_BYTE_ALIGNED_MALLOC
SQLITE_CASE_SENSITIVE_LIKE
SQLITE_DIRECT_OVERFLOW_READ
SQLITE_HAVE_ISNAN
SQLITE_MAX_ALLOCATION_SIZE=N
SQLITE_OS_OTHER=<0 or 1>
SQLITE_SECURE_DELETE
SQLITE_THREADSAFE=<0 or 1 or 2>
SQLITE_CONFIG_SINGLETHREAD
SQLITE_CONFIG_MULTITHREAD
SQLITE_CONFIG_SERIALIZED
SQLITE_TEMP_STORE=<0 through 3>
SQLITE_TRACE_SIZE_LIMIT=N
SQLITE_TRUSTED_SCHEMA=<0 or 1>
SQLITE_USE_URI

# Options To Enable Features Normally Turned Off
SQLITE_ALLOW_URI_AUTHORITY
SQLITE_ALLOW_COVERING_INDEX_SCAN=<0 or 1>
SQLITE_ENABLE_8_3_NAMES=<1 or 2>
SQLITE_ENABLE_API_ARMOR
SQLITE_ENABLE_ATOMIC_WRITE
SQLITE_ENABLE_BATCH_ATOMIC_WRITE
SQLITE_ENABLE_BYTECODE_VTAB
SQLITE_ENABLE_COLUMN_METADATA
SQLITE_ENABLE_DBPAGE_VTAB
SQLITE_ENABLE_DBSTAT_VTAB
SQLITE_ENABLE_DESERIALIZE
SQLITE_ENABLE_EXPLAIN_COMMENTS
SQLITE_ENABLE_FTS3
SQLITE_ENABLE_FTS3_PARENTHESIS
SQLITE_ENABLE_FTS3_TOKENIZER
SQLITE_ENABLE_FTS4
SQLITE_ENABLE_FTS5
SQLITE_ENABLE_GEOPOLY
SQLITE_ENABLE_ICU
SQLITE_ENABLE_IOTRACE
SQLITE_ENABLE_MATH_FUNCTIONS
SQLITE_ENABLE_JSON1
SQLITE_ENABLE_LOCKING_STYLE
SQLITE_ENABLE_MEMORY_MANAGEMENT
SQLITE_ENABLE_MEMSYS3
SQLITE_ENABLE_MEMSYS5
SQLITE_ENABLE_NORMALIZE
SQLITE_ENABLE_NULL_TRIM
SQLITE_ENABLE_OFFSET_SQL_FUNC
SQLITE_ENABLE_PREUPDATE_HOOK
SQLITE_ENABLE_QPSG
SQLITE_ENABLE_RBU
SQLITE_ENABLE_RTREE
SQLITE_ENABLE_SESSION
SQLITE_ENABLE_SNAPSHOT
SQLITE_ENABLE_SORTER_REFERENCES
SQLITE_ENABLE_STMT_SCANSTATUS
SQLITE_ENABLE_STMTVTAB
SQLITE_RTREE_INT_ONLY
SQLITE_ENABLE_SQLLOG
SQLITE_ENABLE_STAT2
SQLITE_ENABLE_STAT3
SQLITE_ENABLE_STAT4
SQLITE_ENABLE_TREE_EXPLAIN
SQLITE_ENABLE_UPDATE_DELETE_LIMIT
SQLITE_ENABLE_UNKNOWN_SQL_FUNCTION
SQLITE_ENABLE_UNLOCK_NOTIFY
SQLITE_INTROSPECTION_PRAGMAS
SQLITE_SOUNDEX
SQLITE_USE_ALLOCA
SQLITE_USE_FCNTL_TRACE
SQLITE_HAVE_ZLIB
YYTRACKMAXSTACKDEPTH

# Options To Disable Features Normally Turned On
SQLITE_DISABLE_LFS
SQLITE_DISABLE_DIRSYNC
SQLITE_DISABLE_FTS3_UNICODE
SQLITE_DISABLE_FTS4_DEFERRED
SQLITE_DISABLE_INTRINSIC

# Options To Omit Features
SQLITE_OMIT_ALTERTABLE
SQLITE_OMIT_ANALYZE
SQLITE_OMIT_ATTACH
SQLITE_OMIT_AUTHORIZATION
SQLITE_OMIT_AUTOINCREMENT
SQLITE_OMIT_AUTOINIT
SQLITE_OMIT_AUTOMATIC_INDEX
SQLITE_OMIT_AUTORESET
SQLITE_OMIT_AUTOVACUUM
SQLITE_OMIT_BETWEEN_OPTIMIZATION
SQLITE_OMIT_BLOB_LITERAL
SQLITE_OMIT_BTREECOUNT
SQLITE_OMIT_BUILTIN_TEST
SQLITE_OMIT_CASE_SENSITIVE_LIKE_PRAGMA
SQLITE_OMIT_CAST
SQLITE_OMIT_CHECK
SQLITE_OMIT_COMPILEOPTION_DIAGS
SQLITE_OMIT_COMPLETE
SQLITE_OMIT_COMPOUND_SELECT
SQLITE_OMIT_CTE
SQLITE_OMIT_DATETIME_FUNCS
SQLITE_OMIT_DECLTYPE
SQLITE_OMIT_DEPRECATED
SQLITE_OMIT_DESERIALIZE
SQLITE_OMIT_DISKIO
SQLITE_OMIT_EXPLAIN
SQLITE_OMIT_FLAG_PRAGMAS
SQLITE_OMIT_FLOATING_POINT
SQLITE_OMIT_FOREIGN_KEY
SQLITE_OMIT_GENERATED_COLUMNS
SQLITE_OMIT_GET_TABLE
SQLITE_OMIT_HEX_INTEGER
SQLITE_OMIT_INCRBLOB
SQLITE_OMIT_INTEGRITY_CHECK
SQLITE_OMIT_INTROSPECTION_PRAGMAS
SQLITE_OMIT_JSON
SQLITE_OMIT_LIKE_OPTIMIZATION
SQLITE_OMIT_LOAD_EXTENSION
SQLITE_OMIT_LOCALTIME
SQLITE_OMIT_LOOKASIDE
SQLITE_OMIT_MEMORYDB
SQLITE_OMIT_OR_OPTIMIZATION
SQLITE_OMIT_PAGER_PRAGMAS
SQLITE_OMIT_PRAGMA
SQLITE_OMIT_PROGRESS_CALLBACK
SQLITE_OMIT_QUICKBALANCE
SQLITE_OMIT_REINDEX
SQLITE_OMIT_SCHEMA_PRAGMAS
SQLITE_OMIT_SCHEMA_VERSION_PRAGMAS
SQLITE_OMIT_SHARED_CACHE
SQLITE_OMIT_SUBQUERY
SQLITE_OMIT_TCL_VARIABLE
SQLITE_OMIT_TEMPDB
SQLITE_OMIT_TRACE
SQLITE_OMIT_TRIGGER
SQLITE_OMIT_TRUNCATE_OPTIMIZATION
SQLITE_OMIT_UTF16
SQLITE_OMIT_VACUUM
SQLITE_OMIT_VIEW
SQLITE_OMIT_VIRTUALTABLE
SQLITE_OMIT_WAL
SQLITE_OMIT_WINDOWFUNC
SQLITE_OMIT_WSD
SQLITE_OMIT_XFER_OPT
SQLITE_UNTESTABLE
SQLITE_ZERO_MALLOC

# Analysis and Debugging Options
SQLITE_DEBUG
SQLITE_MEMDEBUG

# Windows-Specific Options
SQLITE_WIN32_HEAP_CREATE
SQLITE_WIN32_MALLOC_VALIDATE

# Compiler Linkage and Calling Convention Control
SQLITE_API
SQLITE_APICALL
SQLITE_CALLBACK
SQLITE_CDECL
SQLITE_EXTERN
SQLITE_STDCALL
SQLITE_SYSAPI
SQLITE_TCLAPI
```

https://sqlite.org/compile.html

Compile-time options

```
SQLITE_OS_OTHER=<0 or 1> (default: 0)
```

- If the OS is other than Unix, Windows or OS/2,
- You'll specify `SQLITE_OS_OTHER=1`,
- Then you must provide:
  - int sqlite3_os_init(void);
  - int sqlite3_os_end(void);

int sqlite3_os_init(void);

```
sqlite3_mem_methods my_mem_methods = {...}
sqlite3_vfs my_vfs = {...}
/*
 * Typical implementation
 */
int
sqlite3_os_init(void)
{
  sqlite3_config(SQLITE_CONFIG_MALLOC, &my_mem_methods);
  sqlite3_initialize();
  return sqlite3_vfs_register(&my_vfs, 1);
}
```

When a Ruby app opens a DB

SQLite3::Database.open

CRuby SQLite3 module

sqlite3_open_v2(3)

libsqlite3

open(2)

System call interface

Device Driver

Hardware(HDD/SSD)

SQLite3::Database.open

PicoRuby SQLite3 class

sqlite3_open_v2(3)

(Ruby)File.open

PicoRuby File class

Hardware(SD card)

When a Ruby app opens a DB

SQLite3::Database.open
CRuby SQLite3 module
sqlite3_open_v2(3)
libsqlite3
open(2)
System call interface
Device Driver
Hardware(HDD/SSD)

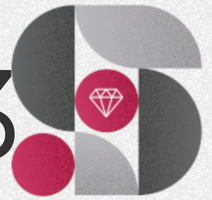
SQLite3::Database.open
PicoRuby SQLite3 class
sqlite3_open_v2(3)
WHAT HAPPENS HERE?
(Ruby)File.open
PicoRuby File class
Hardware(SD card)

SQLite3 needs VFS

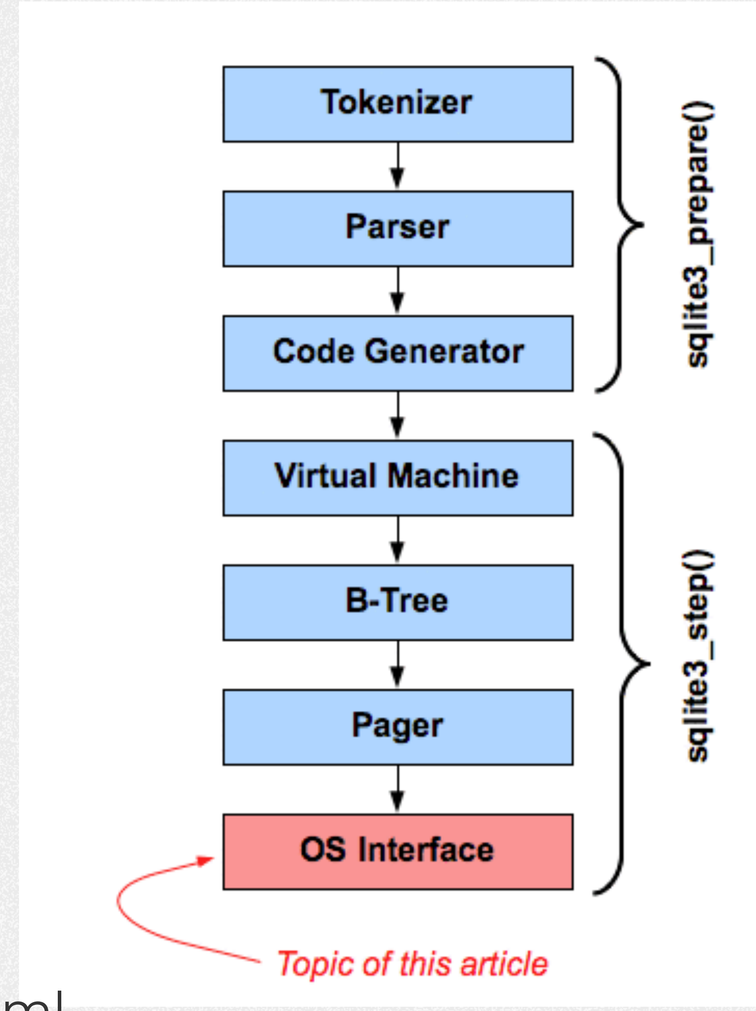
SQLite3::Database.open	SQLite3::Database.open
CRuby SQLite3 module	PicoRuby SQLite3 class
sqlite3_open_v2(3)	**sqlite3_open_v2(3)**
libsqlite3	
open(2)	**(Ruby)File.open**
System call interface	PicoRuby File class
Device Driver	
Hardware(HDD/SSD)	Hardware(SD card)

VFS

VFS (Virtual FileSystem) in SQLite3



- “The module at the bottom of the SQLite implementation stack that provides portability across operating systems.”
- “ (...) Hence, porting SQLite to a new operating system is simply a matter of writing a new OS interface layer or "VFS".”



https://www.sqlite.org/vfs.html

Structs to be prepared

- sqlite3_mem_methods
  - Pointers to functions to manage **memory**
- sqlite3_vfs
  - Pointers to functions of **filesystem** management, **utility** functions and **global app data**
- sqlite3_io_methods
  - Pointers to functions to handle **an open file**
- sqlite3_file (mentioned later)

struct sqlite3_mem_methods

```
struct sqlite3_mem_methods {
  void *(*xMalloc)(int);         /* Memory allocation function */
  void (*xFree)(void*);          /* Free a prior allocation */
  void *(*xRealloc)(void*,int); /* Resize an allocation */
  int (*xSize)(void*);           /* Return the size of an allocation */
  int (*xRoundup)(int);          /* Round up request size to allocation size */
  int (*xInit)(void*);           /* Initialize the memory allocator */
  void (*xShutdown)(void*);      /* Deinitialize the memory allocator */
  void *pAppData;                /* Argument to xInit() and xShutdown() */
};
```

struct sqlite3_vfs

```
struct sqlite3_vfs {
  int iVersion;            /* Structure version number (currently 3) */
  int szOsFile;            /* Size of subclassed sqlite3_file */
  int mxPathname;          /* Maximum file pathname length */
  sqlite3_vfs *pNext;      /* Next registered VFS */
  const char *zName;       /* Name of this virtual file system */
  void *pAppData;          /* Pointer to application-specific data */
  int (*xOpen)(sqlite3_vfs*, sqlite3_filename zName, sqlite3_file*,
               int flags, int *pOutFlags);
  int (*xDelete)(sqlite3_vfs*, const char *zName, int syncDir);
  int (*xAccess)(sqlite3_vfs*, const char *zName, int flags, int *pResOut);
  int (*xFullPathname)(sqlite3_vfs*, const char *zName, int nOut, char *zOut);
  void *(*xDlOpen)(sqlite3_vfs*, const char *zFilename);
  void (*xDlError)(sqlite3_vfs*, int nByte, char *zErrMsg);
  void (*(*xDlSym)(sqlite3_vfs*,void*, const char *zSymbol))(void);
  void (*xDlClose)(sqlite3_vfs*, void*);
  int (*xRandomness)(sqlite3_vfs*, int nByte, char *zOut);
  int (*xSleep)(sqlite3_vfs*, int microseconds);
  int (*xCurrentTime)(sqlite3_vfs*, double*);
  int (*xGetLastError)(sqlite3_vfs*, int, char *);
  int (*xCurrentTimeInt64)(sqlite3_vfs*, sqlite3_int64*);
  int (*xSetSystemCall)(sqlite3_vfs*, const char *zName, sqlite3_syscall_ptr);
  sqlite3_syscall_ptr (*xGetSystemCall)(sqlite3_vfs*, const char *zName);
  const char *(*xNextSystemCall)(sqlite3_vfs*, const char *zName);
};
```

struct sqlite3_io_methods

```
struct sqlite3_io_methods {
  int iVersion;
  int (*xClose)(sqlite3_file*);
  int (*xRead)(sqlite3_file*, void*, int iAmt, sqlite3_int64 iOfst);
  int (*xWrite)(sqlite3_file*, const void*, int iAmt, sqlite3_int64 iOfst);
  int (*xTruncate)(sqlite3_file*, sqlite3_int64 size);
  int (*xSync)(sqlite3_file*, int flags);
  int (*xFileSize)(sqlite3_file*, sqlite3_int64 *pSize);
  int (*xLock)(sqlite3_file*, int);
  int (*xUnlock)(sqlite3_file*, int);
  int (*xCheckReservedLock)(sqlite3_file*, int *pResOut);
  int (*xFileControl)(sqlite3_file*, int op, void *pArg);
  int (*xSectorSize)(sqlite3_file*);
  int (*xDeviceCharacteristics)(sqlite3_file*);
  int (*xShmMap)(sqlite3_file*, int iPg, int pgsz, int, void volatile**);
  int (*xShmLock)(sqlite3_file*, int offset, int n, int flags);
  void (*xShmBarrier)(sqlite3_file*);
  int (*xShmUnmap)(sqlite3_file*, int deleteFlag);
  int (*xFetch)(sqlite3_file*, sqlite3_int64 iOfst, int iAmt, void **pp);
  int (*xUnfetch)(sqlite3_file*, sqlite3_int64 iOfst, void *p);
};
```

struct sqlite3_file

```c
struct sqlite3_file { /* Defined in sqlite3.h */
  const struct sqlite3_io_methods *pMethods;  /* Methods for an open file */
};

struct PRBFile { /* Defined in PicoRuby */
  sqlite3_file base;
  mrbc_vm *vm;
  mrbc_value *file;
  char pathname[PATHNAME_MAX_LEN];
  int sector_size;
};

PRBFile *prbfile = (PRBFile *)pFile;
```

A trick to let SQLite3 carry Ruby

prbVFSOpen is called to open DB

```c
sqlite3_vfs prb_vfs = {
  ...
  sizeof(PRBFile), /* szOsFile. SQLite3 knows the size of PRBFile👀 */
  ...
  vm,              /* pAppData -> PicoRuby's VM */
  prbVFSOpen,      /* xOpen */
  ...
}

int prbVFSOpen(sqlite3_vfs *pVfs, const char *zName,
               sqlite3_file *pFile, int flags, int *pOutFlags) {
  PRBFile *prbfile = (PRBFile *)pFile; /* Cast sqlite3_file to PRBFile */
  prbfile->vm = prb_vfs.pAppData;      /* prb_vfs.pAppData points to VM */
  pFile->pMethods = &prb_io_methods;   /* Attach IO mehtods to the file */
  prb_file_new(prbfile, zName, flags); /* Wrapper of Ruby's File.new */
  return SQLITE_OK;
}
```

prb_file_new == File.new

```c
int prb_file_new(PRBFile *prbfile, const char *zName, int flags) {
  mrbc_vm *vm = (mrbc_vm *)prbfile->vm;
  mrbc_value v[3];
  v[0] = mrbc_nil_value();
  v[1] = mrbc_string_new_cstr(vm, zName);
  char *mode = ... /* "r", "w", "w+", etc. from flags */
  v[2] = mrbc_string_new_cstr(vm, mode);
  prb_funcall(vfs_methods.file_new, &v[0], 2);
  mrbc_decref(&v[1]);
  mrbc_decref(&v[2]);
  prbfile->file = mrbc_alloc(vm, sizeof(mrbc_value));
  memcpy(prbfile->file, &v[0], sizeof(mrbc_value)); /* File Object */
  return OK;
}
```

prb_funcall(vfs_methods.file_new, &v[0], 2)🤔

prbIORead is called to read DB

```
sqlite3_io_methods prb_io_methods = {
  ...
  prbIORead, /* xRead */
  ...
}

int prbIORead(sqlite3_file *pFile, void *zBuf, int iAmt, sqlite3_int64 iOfst) {
  PRBFile *prbfile = (PRBFile *)pFile;
  prb_file_read(prbfile, zBuf, iAmt);  /* Wrapper of Ruby's File#read */
  return SQLITE_OK;
}
```

prb_file_read == File#read

```c
int prb_file_read(PRBFile *prbfile, void *zBuf, size_t nBuf) {
  mrbc_value v[2];
  v[0] = *prbfile->file;
  v[1] = mrbc_integer_value(nBuf);
  prb_funcall(vfs_methods.file_read, &v[0], 1);
  size_t retSize = v[0].string->size;
  memcpy(zBuf, v[0].string->data, retSize);
  mrbc_decref(&v[0]);
  return retSize;
}
```

prb_funcall(vfs_methods.file_read, &v[0], 1)🤔

prb_funcall

```
void prb_funcall(
  void (*func)(mrbc_vm *, mrbc_value *, int),
  mrbc_value *v, int argc
)
{
  mrbc_incref(&v[0]);
  func(prbvfs.pAppData, &v[0], argc);
}
```

Takes a pointer to a C function representing a Ruby method.

But how does SQLite3 class know File methods?

File::VFS class exposes methods 😈

```c
void FileVFS_vfs_methods(mrbc_vm *vm, mrbc_value v[], int argc)
{
  prb_vfs_methods vfs_methods = {
    File_new,
    File_close,
    File_read,
    ...
  };
  mrbc_value methods = mrbc_instance_new(
                         vm, class_FAT_VFSMethods, sizeof(prb_vfs_methods)
                       );
  memcpy(methods.instance->data, &vfs_methods, sizeof(prb_vfs_methods));
  SET_RETURN(methods);
}
mrbc_define_method(0, class_FAT, "vfs_methods", FileVFS_vfs_methods);
```

SQLite3.vfs_methods = File::VFS.vfs_methods

From the point of Ruby's view

```ruby
SQLite3.vfs_methods = File::VFS.vfs_methods
# Pointers to File methods are told to SQLite3

db = SQLite3::Database.new("database.db")
# sqlite3_os_init() -> sqlite3_vfs_register()
# sqlite3_open_v2() -> prbVFSOpen() -> prb_file_new() -> File.new

db.prepare "SELECT * FROM table;"
# sqlite3_prepare_v2() -> prbIORead() -> prb_file_read() -> File#read
```

SQLite3 (Ruby) -> SQLite3 (C) -> File (Ruby)
In bare-metal (no-OS) PicoRuby,
methods of File class work as system call

And so forth

File.new, File#read, File#write,
File#fsync, File#seek, File#tell,
File#close, File.unlink, File.exist?,
File.stat, and Time.now

Map these methods as SQLite3 VFS functions

We made it 🎉

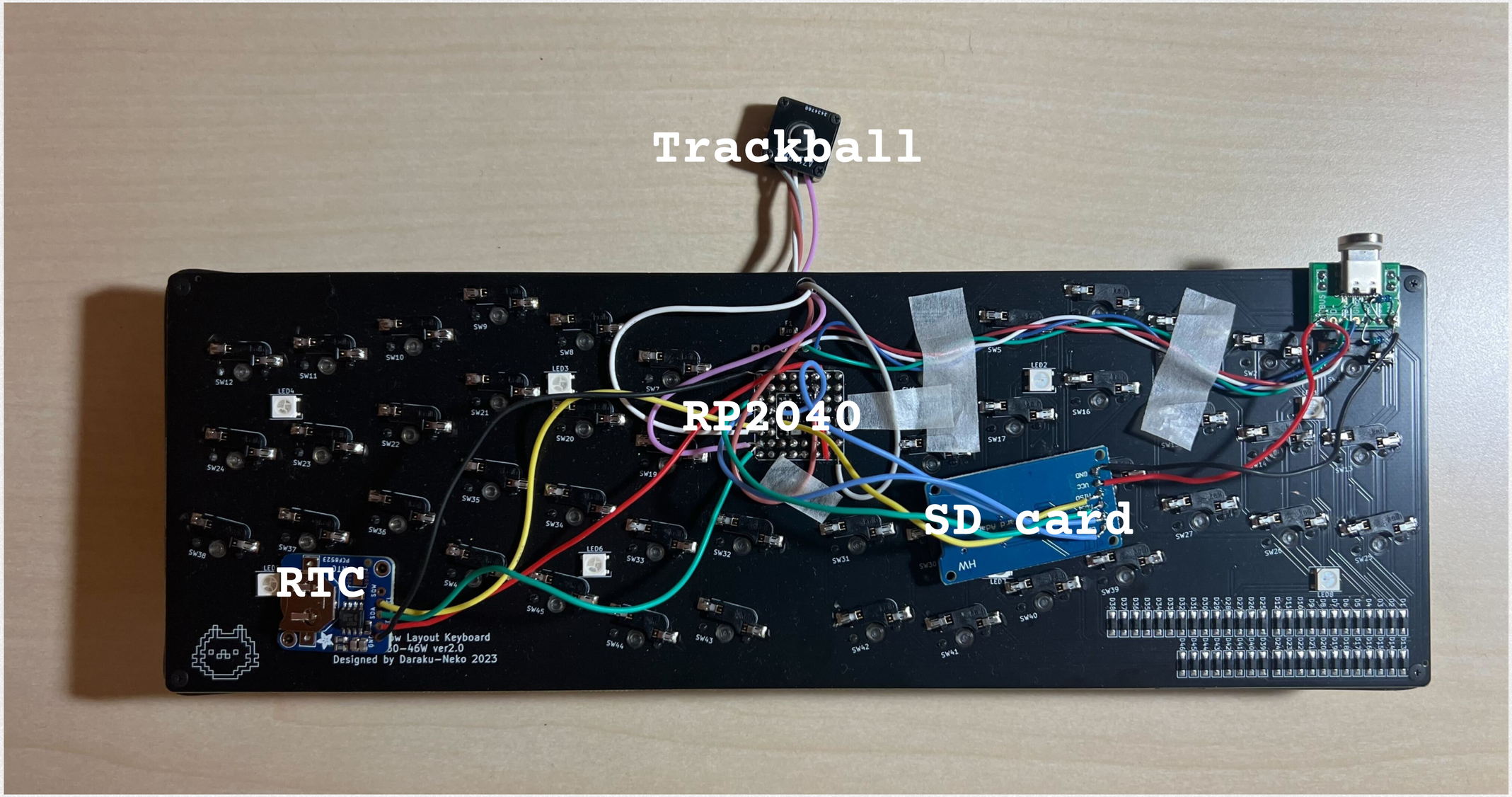
(As you saw in the demo)

SQLite3 in Micon for what? Guess...

- IoT, General electrical appliance
  - Store structured sensor data (in case no network)
  - Log firmware update history
  - Store, backup and share configuration
- Embedded in-memory-database
  - (CRuby(Ractor[PicoRuby(in-memory-SQLite3) * n]))

App for PRK Firmware

App for PRK Firmware



Make a better keymap

Make a better keymap

- On the keyboard,
  - Log every press of the alphabet key with a timestamp
  - Exclude consecutive same key
- On the laptop/desktop,
  - Analyze the data, then make a better keymap

keymap.rb (1/2)

```
begin
  dbfile = "/keylog.db"
  jounalfile = "/keylog.db-journal"
  if File.exist?(dbfile)
    File.rename(dbfile, "/keylog-#{Time.now.to_i}.db")
    File.unlink(jounalfile) if File.exist?(jounalfile)
  end
  sqlite3 = SQLite3::Database.new(dbfile)
  sql = "CREATE TABLE IF NOT EXISTS keylog (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          key TEXT, unixtime REAL);"
  sqlite3.execute(sql)
  sqlite3_stmt = sqlite3.prepare(
                   "INSERT INTO keylog (key, unixtime) VALUES (?, ?);")
rescue => e
  puts "Not available"
  puts "#{e.message} (#{e.class})"
end
```

keymap.rb (2/2)

```ruby
kbd = Keyboard.new
last_keycode = nil
kbd.before_report do |keyboard|
  # KC_A..KC_Z == 4..29
  if (keycode = keyboard.keycodes[0]) && (keycode < 30)
    next if last_keycode == keycode
    last_keycode = keycode
    key = (keycode + 61).chr # (4 + 61).chr => "A"
    puts "key: #{key}"
    begin
      sqlite3_stmt&.execute(key, Time.now.to_f)
    rescue => e
      puts "SQLite3 error: #{e.message} (#{e.class})"
    end
  end
end
```

keylog.db looks like

```
[[ 1, "H", 1683536977.633],
 [ 2, "E", 1683536977.8],
 [ 3, "L", 1683536977.968],
 [ 4, "O", 1683536978.696],
 [ 5, "W", 1683536979.16],
 [ 6, "O", 1683536979.24],
 [ 7, "R", 1683536979.456,
 [ 8, "L", 1683536980.008,
 [ 9, "D", 1683536980.984],
 [10, "R", 1683536981.616],
 [11, "U", 1683536983.480],
 [12, "B", 1683536983.607],
 [13, "Y", 1683536984.18],
 (...array goes...)
```

Analysis strategy

- Order by "most hit key"
- Order by "key combination most frequently hit in succession"
  - Place each character of the combination on opposite sides of the keyboard
    Eg: If "RUBY" is the most hit word,
    "RU" -> "R" for right, "U" for left
    "BY" -> "B" for right, "Y" for left

Copy keylog.db to laptop, then

```
db = SQLite3::Database.new('keylog.db')
db.results_as_hash = true
db.execute "CREATE TABLE IF NOT EXISTS key_interval
            (id INTEGER PRIMARY KEY, prev_key TEXT, next_key TEXT, interval REAL);"
db.execute "DELETE FROM key_interval;"

stmt = db.prepare("INSERT INTO key_interval
                  (prev_key, next_key, interval) VALUES (?, ?, ?)")
prev_row = nil
db.execute("SELECT id, key, unixtime FROM keylog order by id") do |next_row|
  if prev_row
    interval = next_row['unixtime'] - prev_row['unixtime']
    stmt.execute(prev_row['key'], next_row['key'], interval)
  end
  prev_row = next_row
end
```

Make another table that consists of the time difference between adjoining keystrokes

Prepare objects for result

```ruby
ALPHABET = Hash.new.tap do |h|
  ('A'..'Z').each { |c| h[c] = false }
end

COLUMN_SIZE = 5
ROW_SIZE = 3
KEYMAP_RIGHT = Array.new(COLUMN_SIZE).map do |colomn|
  colum = Array.new(ROW_SIZE)
end
KEYMAP_LEFT = Array.new(COLUMN_SIZE).map do |colomn|
  colum = Array.new(ROW_SIZE)
end

COLUMN_PRIORITY = [1, 2, 0, 3, 4]
ROW_PRIORITY = [1, 2, 0]

ASSIGN_ORDER = Array.new.tap do |a|
  COLUMN_PRIORITY.each do |column|
    ROW_PRIORITY.each do |row|
      a << [column, row]
    end
  end
end
```

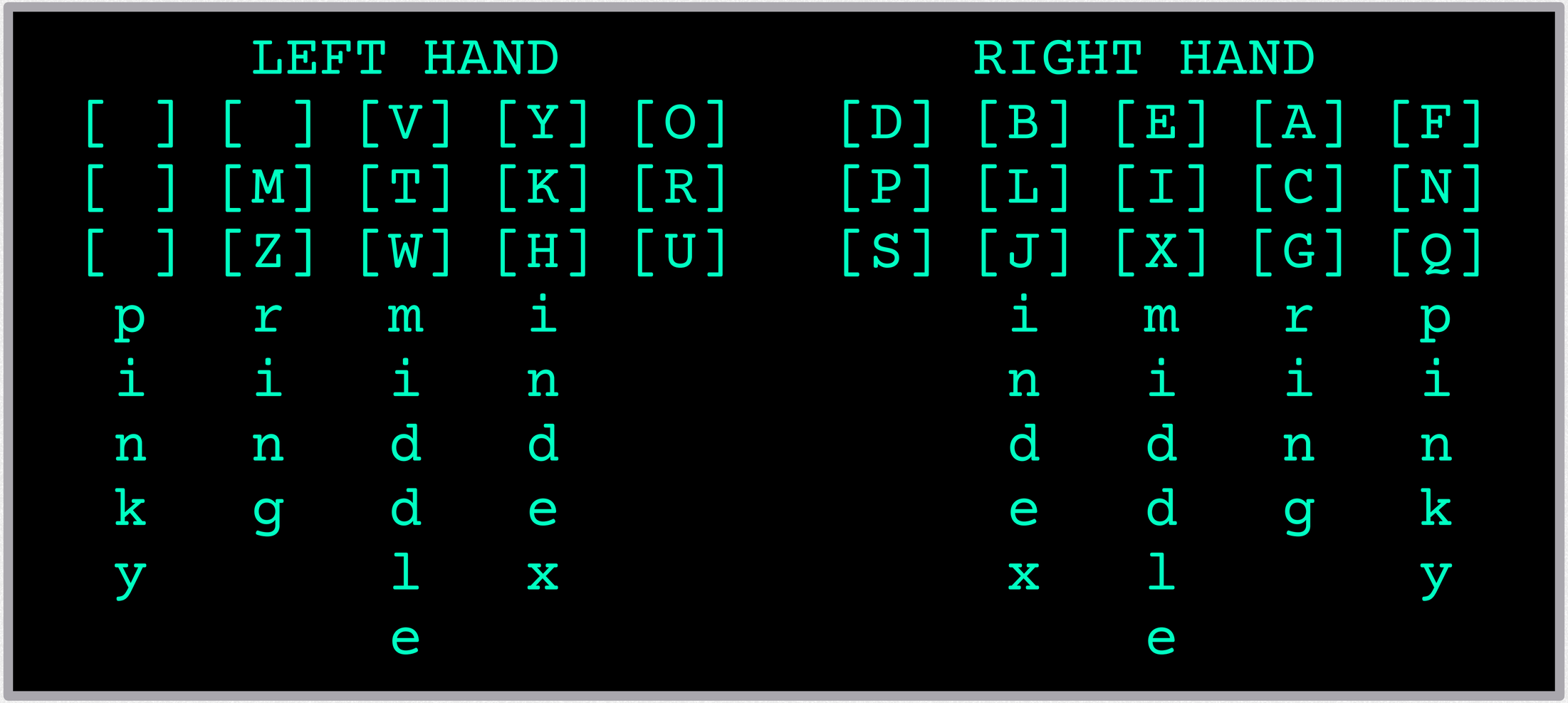
Calculate a better keymap

```
right_num, left_num = 0, 0
db.execute("SELECT key, COUNT(*) AS count
            FROM keylog GROUP BY key ORDER BY count DESC;") do |row|
  key = row["key"]
  next if ALPHABET[key]
  db.execute("SELECT next_key, COUNT(*) AS count FROM key_interval
              WHERE prev_key = '#{key}' AND interval < 1
              GROUP BY next_key ORDER BY count DESC;") do |row|
    right_x, right_y = ASSIGN_ORDER[right_num]
    break if right_x.nil? || right_y.nil?
    KEYMAP_RIGHT[right_x][right_y] = key
    ALPHABET[key] = true
    next_key = row["next_key"]
    next if ALPHABET[next_key]
    left_x, left_y = ASSIGN_ORDER[left_num]
    left_num += 1
    KEYMAP_LEFT[left_x][left_y] = next_key
    ALPHABET[next_key] = true
    break
  end
  right_num += 1
end
db.close
```

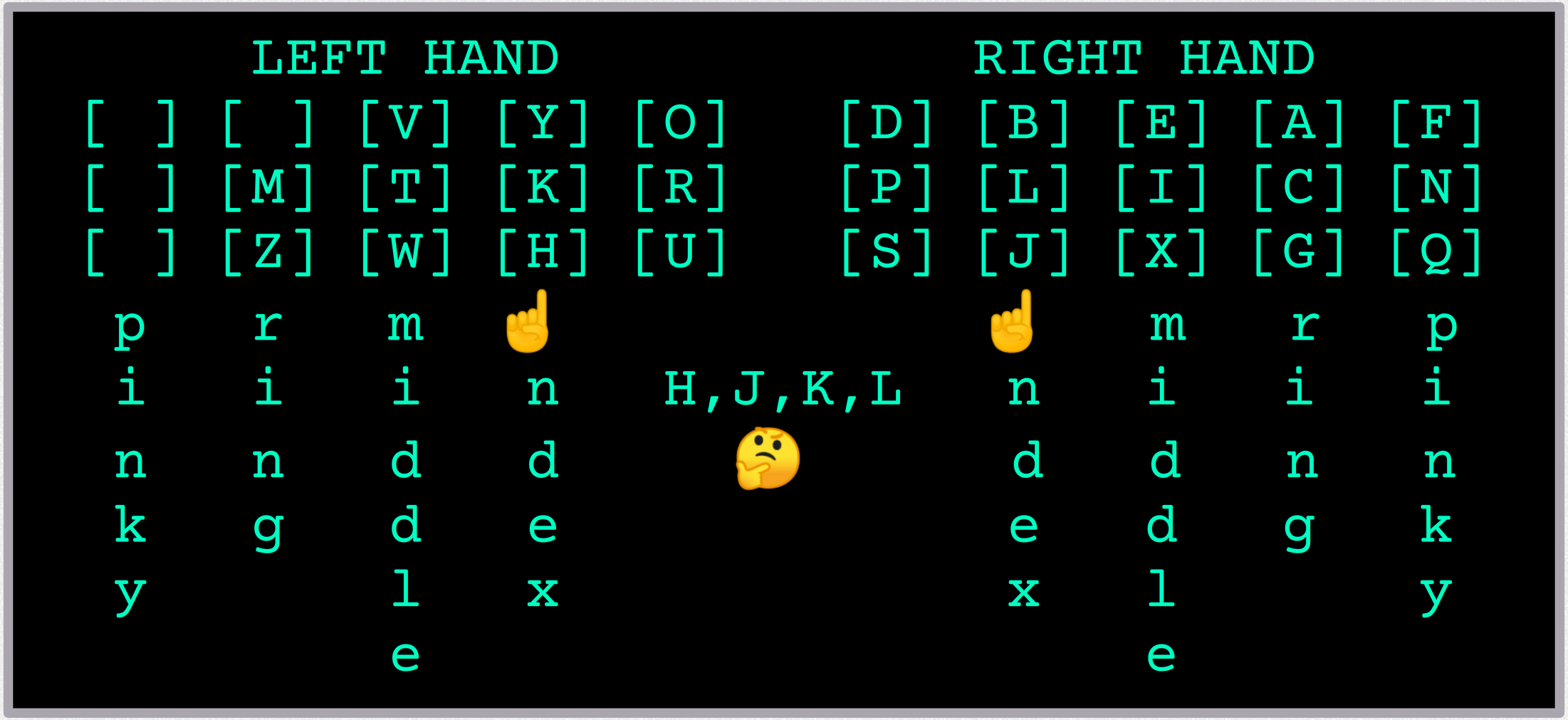
Show the result on terminal

```
puts "     LEFT HAND                RIGHT HAND"
0.upto(ROW_SIZE - 1) do |y|
  (COLUMN_SIZE - 1).downto(0) do |x|
    print "[#{KEYMAP_LEFT[x][y] || ' '}] "
  end
  print "  "
  0.upto(COLUMN_SIZE - 1) do |x|
    print "[#{KEYMAP_RIGHT[x][y] || ' '}] "
  end
  puts
end
FINGERS = %w(index middle ring pinky)
0.upto(4) do |c|
  FINGERS.reverse.each do |finger|
    print " #{finger[c] || ' '}  "
  end
  print "          "
  FINGERS.each do |finger|
    print " #{finger[c] || ' '}  "
  end
  puts
end
```

The result 🎉


LEFT HAND
[ ] [ ] [V] [Y] [O]
[ ] [M] [T] [K] [R]
[ ] [Z] [W] [H] [U]
p r m i
i i i n
n n d d
k g d e
y l x
e
RIGHT HAND
[D] [B] [E] [A] [F]
[P] [L] [I] [C] [N]
[S] [J] [X] [G] [Q]
i m r p
n i i i
d d n n
e d g k
x l y
e

The result 🥳


LEFT HAND
RIGHT HAND
[ ] [ ] [V] [Y] [O] [D] [B] [E] [A] [F]
[ ] [M] [T] [K] [R] [P] [L] [I] [C] [N]
[ ] [Z] [W] [H] [U] [S] [J] [X] [G] [Q]
p r m ☝ ☝ m r p
i i i n H,J,K,L n i i i
n n d d 🤔 d d n n
k g d e e d g k
y l x x l y
e e

https://github.com/tenderlove/initial-v

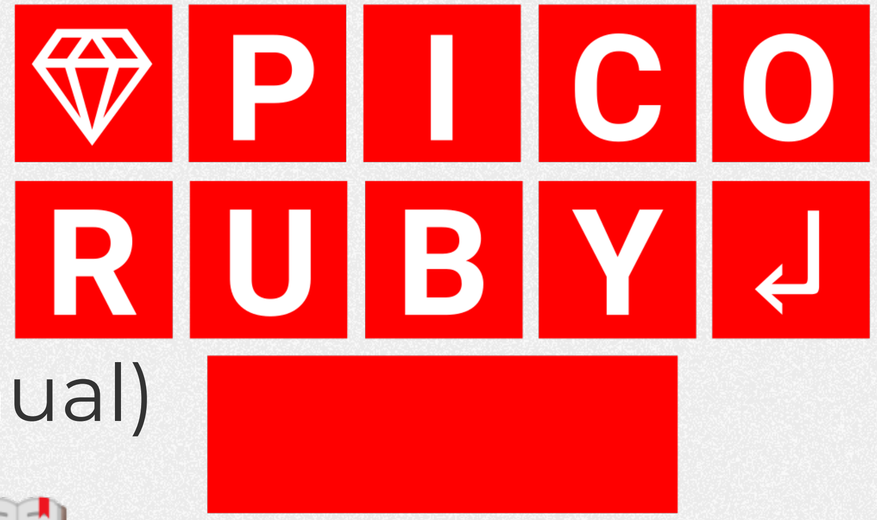
The cursor keys of Vim

H, **J**, **K**, **L**

occupy a prime location😂

Database's ready, what's next?

- Improve shell🐚
- Network, Socket, TCP
- Bluetooth Low Energy🦷
- API documentation🔍
  - Picoるりま(Ruby reference manual)
- Write a doujinshi (thin book)📖
  - In English🇬🇧 🤔



Visit repos and stargaze🌟

github.com/picoruby/picoruby

github.com/picoruby/prk_firmware

Thank you!!!!q