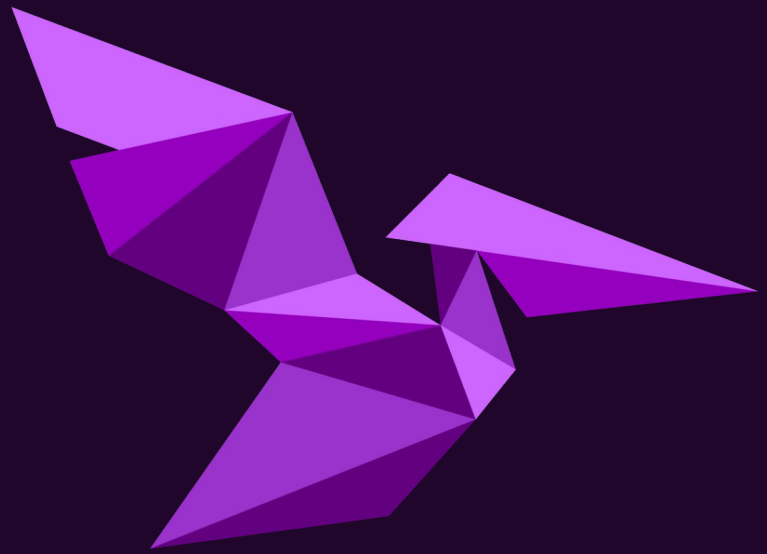

VillageSQL

“VillageSQL’s mission is to drive innovation in the MySQL community”

Who is VillageSQL?

40+ Years of Database experience at Google, MySQL, AWS, Oracle

Tomas Ulin

Village Historian

MySQL, Sun Microsystems, Oracle

Issue: **Toward a Stable, Versioned Extension API and ABI for MySQL**

Problem:

- Challenges with extending MySQL using plugins and components
- API and ABI compatibility across versions and distributions are not guaranteed
- Plugins are hard to use. Components are incomplete.

Proposed Solution:

- **Common/compatible extension API/ABI aimed at improving portability, stability, and long-term extensibility for the MySQL ecosystem**

https://bugs.mysql.com/bug.php?id=120427

C villagesql/abi/types.h

```c
// The only symbol the server looks up
vef_registration_t *vef_register(vef_protocol_t server_protocol);

// Stable input struct — no internal headers required
typedef struct {
  vef_type_id  type;        // STRING, INT, REAL, CUSTOM
  bool         is_null;
  const char  *str_value;
  size_t       str_len;
} vef_invalue_t;

// Stable output struct — server allocates the buffer
typedef struct {
  vef_return_value_type_t  type;        // VALUE, NULL, WARNING, ERROR
  char                    *str_buf;
  size_t                   max_str_len;
  size_t                   actual_len;
} vef_vdf_result_t;
```

The server negotiates protocol version at load time. Extensions declare what they need; the server decides whether to load. **This is the entire surface area an extension needs to target.**

C++ vsql-ai/src/ai_functions.cc

```cpp
void prompt_impl(vef_context_t* ctx,
                 vef_invalue_t* provider, vef_invalue_t* model,
                 vef_invalue_t* api_key,  vef_invalue_t* prompt,
                 vef_vdf_result_t* result) {
  if (provider->is_null || prompt->is_null) {
    result->type = VEF_RESULT_NULL; return;
  }
  auto response = call_provider(provider, model, api_key, prompt);
  result->type       = VEF_RESULT_VALUE;
  result->actual_len = response.length();
  memcpy(result->str_buf, response.c_str(),
         std::min(response.length(), result->max_str_len - 1));
}

VEF_GENERATE_ENTRY_POINTS(
  make_extension("vsql_ai", "0.0.3")
    .func(make_func<&prompt_impl>("ai_prompt")
      .returns(STRING)
      .param(STRING)   // provider
      .param(STRING)   // model
      .param(STRING)   // api_key
      .param(STRING)   // prompt
      .buffer_size(65535).build())
)
```

RUST vsql-rust-sdk · ai_prompt example

```rust
fn ai_prompt_impl(args: &[InValue]) -> VdfReturn {
    match args {
        [InValue::String(provider), InValue::String(model),
         InValue::String(api_key),  InValue::String(prompt)] => {
            VdfReturn::string(call_provider(provider, model, api_key, prompt))
        }
        _ => VdfReturn::null(),
    }
}

vsql::extension! {
    funcs: [
        vsql::func!(ai_prompt_impl, "ai_prompt",
            [Type::String, Type::String, Type::String, Type::String]
            -> Type::String),
    ]
}
```

All FFI marshaling is hidden. NULL handling is explicit. No C boilerplate. **The ABI is unchanged — the Rust SDK is tooling on top of the same stable contract.**

Issue: **Toward a Stable, Versioned Extension API and ABI for MySQL**

Problem:

- Challenges with extending MySQL using plugins and components
- API and ABI compatibility across versions and distributions are not guaranteed
- Plugins are hard to use. Components are incomplete.

Proposed Solution:

- **Common/compatible extension API/ABI aimed at improving portability, stability, and long-term extensibility for the MySQL ecosystem**

Questions & Discussion

www.villagesql.com