C++20 ❤️ SQL

John R Bandela, MD

Overview

- SQL
- C++ Example
- Implementation techniques
  - fixed_string
  - meta_struct
  - Parsing compile time strings into meta_structs

Overview

- SQL
- C++ Example
- Implementation Techniques
- https://github.com/google/cpp-from-the-sky-down/tree/master/meta_struct_20/cppcon_version

SQL

- Probably the highest level mainstream programming language
- It matches the capabilities of a relational database
- Very widely used. You will be able to find all sorts of information about it

Example Database

Customers	
id	INTEGER PRIMARY KEY
name	TEXT

Orders	
id	INTEGER PRIMARY KEY
item	TEXT
customerid	INTEGER
price	REAL

Create the customers table

```
CREATE TABLE customers(
    id INTEGER NOT NULL PRIMARY KEY,
    name TEXT NOT NULL
    );
```

Customers	
id	INTEGER PRIMARY KEY
name	TEXT

Create the orders table

```
CREATE TABLE orders(
    id INTEGER NOT NULL PRIMARY KEY,
    item TEXT NOT NULL,
    customerid INTEGER NOT NULL,
    price REAL NOT NULL,
    discount_code TEXT
    );
```

Orders	
id	INTEGER PRIMARY KEY
item	TEXT
customerid	INTEGER
price	REAL

Insert Customer

```
INSERT INTO customers(name)
    VALUES("John")
```

id	name
1	John

Insert Order

```
INSERT INTO orders(item , customerid , price, discount_code )
    VALUES ("Hoodie",1,10.0,"CppCon");
```

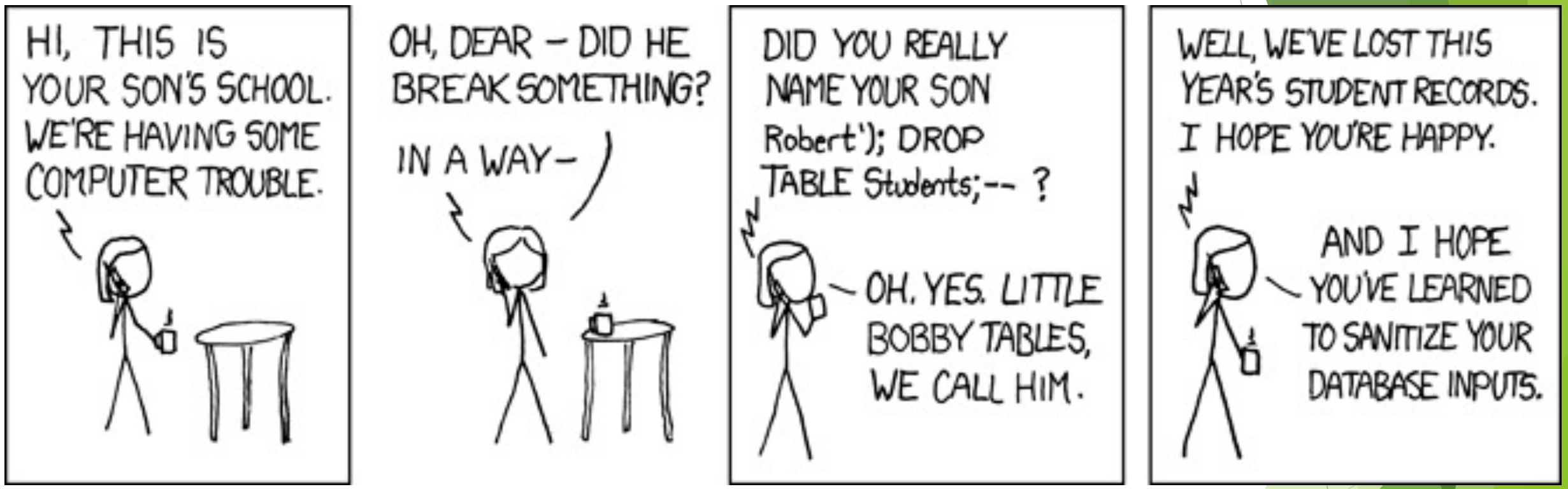
Id	item	customerid	price	discount_code
1	“Hoodie”	1	10.0	CppCon

Query for orders

```
SELECT orders.id, name, item, price, discount_code
    FROM orders JOIN customers ON customers.id = customerid
    WHERE price > 5;
```

id	name	item	price	discount_code
1	“John”	“Hoodie”	10.0	“CppCon”

Query for orders with input from user



Avoiding SQL Injection Attacks

```
SELECT orders.id, name, item, price, discount_code
    FROM orders JOIN customers ON customers.id = customerid
    WHERE price > ?;
```

SQL Library Options

- Traditional database library with strings
- Domain specific language
- Object Relational Mapping

Traditional Library with Strings

- What people think about first
- Many of these are written by the database team itself
- Allows full power of the database, may have extensions
- Lots of information available
- Vulnerable to sql injection of developer not careful
- Use dynamic typing

Domain Specific Language

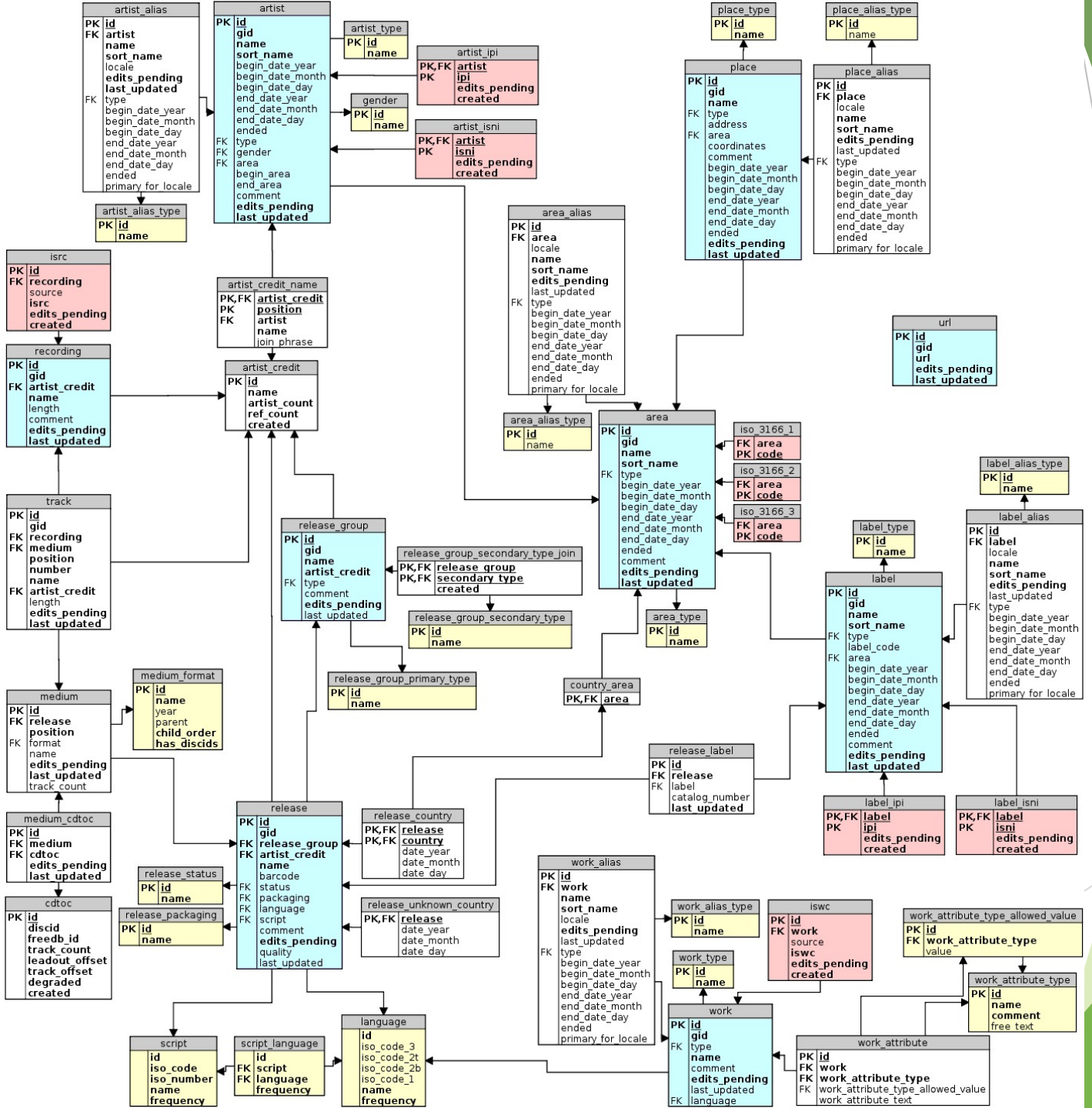
- Use types to encode SQL
- Looks more like regular C++
- Typically requires the database schema
- Example sqlpp11

Object Relational Model

- Focuses on objects and how to store them in a database
- Typically generates the SQL itself
- Very convenient if you are on the happy path
- Can be difficult if you need to do something that is not explicitly supported

SQL Library with compiled strings

- Advantage of using (almost) raw SQL
- Still have typed rows and parameters
- No sql injection
- Don’t have to try to support full power of SQL ourselves
- Supports more interactive development
- Don’t have to provide schema


artist_alias
PK id
FK artist
name
sort_name
locale
edits_pending
last_updated
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
primary_for_locale
artist_alias_type
PK id
name
artist
PK id
gid
name
sort_name
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
FK type
FK gender
FK area
begin_area
end_area
comment
edits_pending
last_updated
artist_type
PK id
name
gender
PK id
name
artist_ipi
PK,FK artist
PK ipi
edits_pending
created
artist_isni
PK,FK artist
PK isni
edits_pending
created
place_type
PK id
name
place_alias_type
PK id
name
place
PK id
gid
name
FK type
address
FK area
coordinates
comment
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
edits_pending
last_updated
place_alias
PK id
FK place
locale
name
sort_name
edits_pending
last_updated
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
primary_for_locale
area_alias
PK id
FK area
locale
name
sort_name
edits_pending
last_updated
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
primary_for_locale
area_alias_type
PK id
name
area
PK id
gid
name
sort_name
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
comment
edits_pending
last_updated
iso_3166_1
FK area
PK code
iso_3166_2
FK area
PK code
iso_3166_3
FK area
PK code
area_type
PK id
name
url
PK id
gid
url
edits_pending
last_updated
isrc
PK id
FK recording
source
isrc
edits_pending
created
recording
PK id
gid
FK artist_credit
name
length
comment
edits_pending
last_updated
artist_credit_name
PK,FK artist_credit
PK position
FK artist
name
join_phrase
artist_credit
PK id
name
artist_count
ref_count
created
track
PK id
gid
FK recording
FK medium
position
number
name
FK artist_credit
length
edits_pending
last_updated
release_group
PK id
gid
name
artist_credit
FK type
comment
edits_pending
last_updated
release_group_secondary_type_join
PK,FK release_group
PK,FK secondary_type
created
release_group_secondary_type
PK id
name
release_group_primary_type
PK id
name
label_alias_type
PK id
name
label_alias
PK id
FK label
locale
name
sort_name
edits_pending
last_updated
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
primary_for_locale
label_type
PK id
name
label
PK id
gid
name
sort_name
FK type
label_code
FK area
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
comment
edits_pending
last_updated
country_area
PK,FK area
medium_format
PK id
name
year
parent
child_order
has_discids
medium
PK id
FK release
position
FK format
name
edits_pending
last_updated
track_count
release_label
PK id
FK release
FK label
catalog_number
last_updated
label_ipi
PK,FK label
PK ipi
edits_pending
created
label_isni
PK,FK label
PK isni
edits_pending
created
medium_cdtoc
PK id
FK medium
FK cdtoc
edits_pending
last_updated
cdtoc
PK id
discid
freedb_id
track_count
leadout_offset
track_offset
degraded
created
release
PK id
gid
FK release_group
FK artist_credit
name
barcode
FK status
FK packaging
FK language
FK script
comment
edits_pending
quality
last_updated
release_country
PK,FK release
PK,FK country
date_year
date_month
date_day
release_unknown_country
PK,FK release
date_year
date_month
date_day
release_status
PK id
name
release_packaging
PK id
name
work_alias
PK id
FK work
name
sort_name
locale
edits_pending
last_updated
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
primary_for_locale
work_alias_type
PK id
name
work_type
PK id
name
work
PK id
gid
FK type
name
comment
edits_pending
last_updated
FK language
iswc
PK id
FK work
source
iswc
edits_pending
created
work_attribute_type_allowed_value
PK id
FK work_attribute_type
value
work_attribute_type
PK id
name
comment
free text
work_attribute
PK id
FK work
FK work_attribute_type
FK work_attribute_type_allowed_value
work_attribute_text
script
id
iso_code
iso_number
name
frequency
script_language
id
FK script
FK language
frequency
language
id
iso_code_3
iso_code_2t
iso_code_2b
iso_code_1
name
frequency

MetaStructSQLite

- Written for SQLite, but could support others
- No macros
- No code generators

Create Customers

```
sqlite3 *sqldb;
 sqlite3_open(":memory:", &sqldb);

 ftsd::prepared_statement<  //
    R"( CREATE TABLE customers(
    id INTEGER NOT NULL PRIMARY KEY,
    name TEXT NOT NULL
    );)"                  //
    >{sqldb}
    .execute();
```

Create Orders

```
ftsd::prepared_statement<  //
    R"( CREATE TABLE orders(
    id INTEGER NOT NULL PRIMARY KEY,
    item TEXT NOT NULL,
    customerid INTEGER NOT NULL,
    price REAL NOT NULL,
    discount_code TEXT
    );)"                //
    >{sqldb}
    .execute();
```

Insert Customer

```
ftsd::prepared_statement<
    R"(INSERT INTO customers(name)
    VALUES(? /*:name:text*/);)"  //
    >{sqldb}
    .execute({arg<"name"> = "John"});
```

Lookup a customer by name

```
auto customer_id_or =
    ftsd::prepared_statement<
                R"(SELECT id /*:integer*/ from customers
                WHERE name = ? /*:name:text*/;)"  //
    >{sqldb}
    .execute_single_row({arg<"name"> = "John"});
```

Insert orders

```
auto customer_id = get<"id">(customer_id_or.value());


ftsd::prepared_statement<
 R"(INSERT INTO orders(item , customerid , price, discount_code )
 VALUES (?/*:item:text*/, ?/*:customerid:integer*/, ?/*:price:real*/,
  ?/*:discount_code:text?*/ );)"> insert_order{sqldb};
 insert_order.execute({arg<"item"> = "Phone", arg<"price"> = 1444.4,
                       arg<"customerid"> = customer_id});
 insert_order.execute({arg<"item"> = "Laptop", arg<"price"> = 1300.4,
                       arg<"customerid"> = customer_id});
 insert_order.execute({arg<"customerid"> = customer_id, arg<"price"> = 2000,
                       arg<"item"> = "MacBook",
                       ftsd::arg<"discount_code"> = "BIGSALE"});
```

Query orders

```
ftsd::prepared_statement<
    R"(SELECT orders.id /*:integer*/, name/*:text*/, item/*:text*/,
    price/*:real*/,
    discount_code/*:text?*/
    FROM orders JOIN customers ON customers.id = customerid
    WHERE price > ?/*:min_price:real*/;)"  //
    >
    select_orders{sqldb};
```

Query orders

```
for (auto &row :
  select_orders.execute_rows({arg<"min_price"> = min_price})) {
   std::cout << get<"orders.id">(row) << " ";
   std::cout << get<"price">(row) << " ";
   std::cout << get<"name">(row) << " ";
   std::cout << get<"item">(row) << " ";
   std::cout << ftsd::get<"item">(row) << " ";
   std::cout << get<"discount_code">(row).value_or("<NO CODE>") << "\n";
}
```

Key Design Insight

- For types with SQL, you only need to worry about input and out
- If you can just annotate the select columns and the parameters, you are good
- SQL can have c-style /* */ comments
- Thus all the previous statements were full, valid SQL
- We can easily copy and paste both ways from and interactive SQL terminal

Annotations

- Columns
  - /*:type*/
  - The column name is immediately to the left of the comment
- Parameters
  - /*:name:type*/

Type annotation	C++ type
text	std::string_view
integer	int64_t
real	double
? at end	Turns the type into std::optional

Implementation

- fixed_string
- meta_struct
- required
- Turning the query string into type specifications
- Binding parameters
- Reading rows

Fixed string

```
template <std::size_t N>
struct fixed_string {
  constexpr fixed_string(const char (&foo)[N + 1]) {
    std::copy_n(foo, N + 1, data);
  }
  auto operator<=>(const fixed_string&) const = default;
  char data[N + 1] = {};
};
template <std::size_t N>
fixed_string(const char (&str)[N]) -> fixed_string<N - 1>;
```

meta_struct

```
using Person = meta_struct<     //
   member<"id", int>,           //
   member<"name", std::string>  //
   >;

Person p;
get<"id">(p) = 1;
get<"name">(p) = "John";

std::cout << get<"id">(p) << " " << get<"name">(p) << "\n";
```

Construction

```
using Person = meta_struct<     //
    member<"id", int>,          //
    member<"name", std::string>  //
    >;

Person p{arg<"id"> = 1, arg<"name"> = "John"};

std::cout << get<"id">(p) << " " << get<"name">(p) << "\n";
p = Person{arg<"name"> = "John", arg<"id"> = 1};
std::cout << get<"id">(p) << " " << get<"name">(p) << "\n";
```

Required members

```
using Person = meta_struct<                                    //
    member<"id", int, required>,
    member<"score">,  //
    member<"name", std::string, required>                      //
    >;

Person p{arg<"id"> = 2, arg<"name"> = “John"};
```

Turning the query string into meta_structs for fields and parameters

```
template <fixed_string query_string>
struct meta_structs_from_query {
  static constexpr auto ts = parse_type_specs<query_string>();
  using fields_type = typename meta_struct_from_type_specs<
      query_string, ts.fields, false,
      std::make_index_sequence<ts.fields.size()>>::type;
  using parameters_type = typename meta_struct_from_type_specs<
      query_string, ts.params, true,
      std::make_index_sequence<ts.params.size()>>::type;
};
```

Parsing the type specs

```
template <fixed_string query_string>
 constexpr auto parse_type_specs() {
  constexpr auto sv = query_string.sv();
  constexpr auto ret_counts = get_type_spec_count<query_string>();
  combined_type_specs<ret_counts.fields, ret_counts.params> ret = {};
  …
  return ret;
}
```

combined_type_specs and type_specs

```
template <std::size_t N>
struct type_specs {
  auto operator<=>(const type_specs &) const = default;
  type_spec data[N];
  static constexpr std::size_t size() { return N; }
  constexpr auto &operator[](std::size_t i) const { return data[i]; }
  constexpr auto &operator[](std::size_t i) { return data[i]; }
};
template <std::size_t Fields, std::size_t Params>
struct combined_type_specs {
  type_specs<Fields> fields;
  type_specs<Params> params;
  auto operator<=>(const combined_type_specs &) const = default;
};
```

type_spec and query_substring

```
struct query_substring {
  std::size_t offset;
  std::size_t count;
  constexpr auto operator<=>(const query_substring &other) const = default;
};
struct type_spec {
  query_substring name;
  query_substring type;
  bool optional;
  constexpr auto operator<=>(const type_spec &other) const = default;
};
```

Why not string_view?

Turning the query string into meta_structs for fields and parameters

```
template <fixed_string query_string>
struct meta_structs_from_query {
  static constexpr auto ts = parse_type_specs<query_string>();
  using fields_type = typename meta_struct_from_type_specs<
      query_string, ts.fields, false,
      std::make_index_sequence<ts.fields.size()>>::type;
  using parameters_type = typename meta_struct_from_type_specs<
      query_string, ts.params, true,
      std::make_index_sequence<ts.params.size()>>::type;
};
```

Turn type_specs into meta_struct

```
template <fixed_string query_string, type_specs ts, bool required,
          typename Sequence>
struct meta_struct_from_type_specs;


template <fixed_string query_string, type_specs ts, bool required,
          std::size_t... I>
struct meta_struct_from_type_specs<query_string, ts, required,
                                   std::index_sequence<I...>> {
  using type = typename meta_struct<
      typename member_from_type_spec<query_string, ts[I], required>::type...>;
};
```

member_from_type_spec

```
template <fixed_string query_string, type_spec ts, bool required>
struct member_from_type_spec {
  static constexpr auto sv = query_string.sv();
  static constexpr auto name_str =
      fixed_string<ts.name.count>::from_string_view(
          sv.substr(ts.name.offset, ts.name.count));
  static constexpr auto type_str =
      fixed_string<ts.type.count>::from_string_view(
          sv.substr(ts.type.offset, ts.type.count));

  using type_from_string = string_to_type_t<type_str>;
```

string_to_type

```
template <fixed_string Tag>
struct string_to_type;
template <>
struct string_to_type<"integer"> {using type = std::int64_t;};
template <>
struct string_to_type<"text"> {using type = std::string_view;};
template <>
struct string_to_type<"real"> { using type = double;};
template <fixed_string Tag>
using string_to_type_t = typename string_to_type<Tag>::type;
```

member_from_type_spec

```
using value_type =
    std::conditional_t<ts.optional, std::optional<type_from_string>,
                       type_from_string>;

 using type = ftsd::member<name_str, value_type, []() {
  if constexpr (required && !ts.optional)
    return ftsd::required;
  else
    return ftsd::default_init<value_type>();
 }()>;
};
```

Binding SQL Parameters

```
template <typename ParametersMetaStruct>
void bind_parameters(sqlite3_stmt *stmt, ParametersMetaStruct parameters) {
  int index = 1;
  meta_struct_for_each(
      [&](auto &m) mutable {
        auto r = bind_impl(stmt, index, m.value);
        check_sqlite_return<bool>(r, true);
        ++index;
      },
      parameters);
}
```

Reading SQL rows

```
template <typename RowType>
auto read_row(sqlite3_stmt *stmt) {
  RowType row = {};
  std::size_t count = sqlite3_column_count(stmt);
  auto size = meta_struct_size(row);  assert(size == count);
  int index = 0;
  meta_struct_for_each(
      [&](auto &m) mutable {
        read_row_into(stmt, index, m.value);
        ++index;
      },
      row);
  return row;
}
```

C++20 ❤️ SQL

https://github.com/google/cpp-from-the-sky-down/tree/master/meta_struct_20/cppcon_version