Handling persistenter Daten mit C++ leicht gemacht

Markus Klemm
C++ usergroup Dresden

Motivation

2 Perspektiven

- „Ich muss irgendwo ‚hinspeichern'“ == Daten werden in Anwendung erzeugt und sollen dauerhaft bzw. persistent gespeichert werden

- „Ich hab’ da mal ‘ne Datenbank“ == Daten kommen von extern; Datenbank-Design war zuerst; data-driven-x

Daten „wegschaufeln“

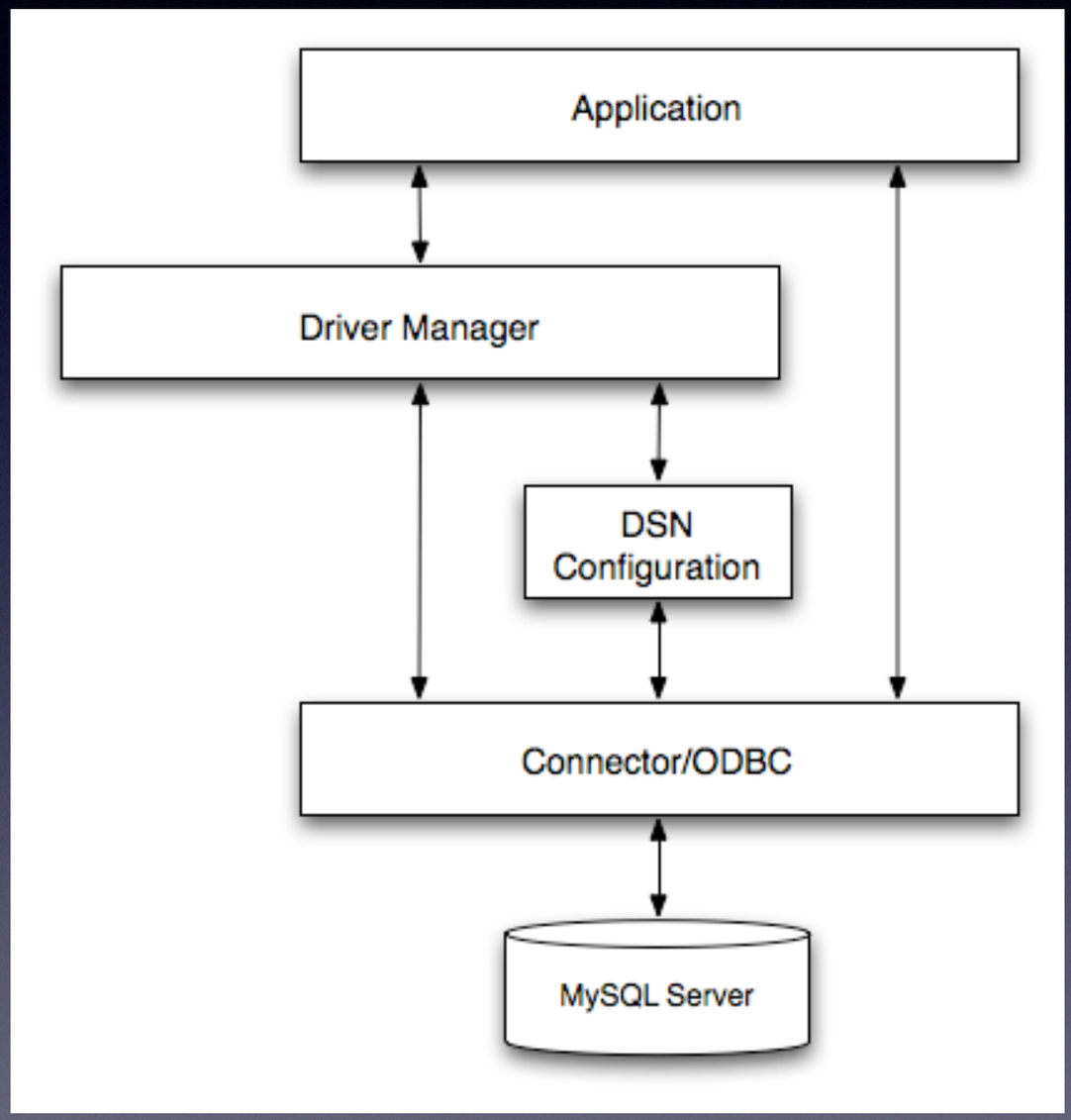
- Verschiedene Ansätze
  - Datei basierend
    - Binärdateien
    - Textdateien
  - Datenbank basierend
    - Nicht relational/NoSQL
    - Relational

„NoSQL? Alles Mode.“

–Beliebiger konservativer Datenbank-Profi/Prof

Erster Ansatz: ODBC

- „Industrie-Standard“
- Thread-Safe & Async schon seit langer Zeit
- Datenbank-Hersteller unabhängig
- Tatsächlich etwas Gutes von MS



```
dbc_stmt mddb_connect (void)
{
    dbc_stmt ret;

    if (SQLAllocHandle(SQL_HANDLE_DBC, global_env_handle, & ret.dbc_handle) != SQL_SUCCESS)
    {
        fprintf(stderr,"Allocating ODBC connection handler failed\n");
        exit(EXIT_FAILURE);
    }

    //**********************
    //Connect to the database
    //**********************
    //Edit here for other connection/database selection mechanism



    SQLSMALLINT completed_connection_string_count;
    SQLRETURN connect_return = SQLDriverConnect (ret.dbc_handle,
                            NULL, //We don't want any dialog boxes
                            connection_string,
                            strlen(connection_string)+1,
                            NULL,
                            0,
                            &completed_connection_string_count,
                            SQL_DRIVER_NOPROMPT);

    if  (connect_return == SQL_SUCCESS) |
    {
        fprintf(stdout,"Connecting to MDDB database succeded \n");

    } else if (connect_return == SQL_SUCCESS_WITH_INFO)
    {
        SQLSMALLINT rec = 1; SQLCHAR  my_sql_state[SQL_SQLSTATE_SIZE+1];  SQLINTEGER my_err_code; SQLSMALLINT
            truncated;
        SQLCHAR  err_msg[1024];
        SQLGetDiagRec(SQL_HANDLE_DBC,ret.dbc_handle,rec,my_sql_state,&my_err_code,err_msg,1024-1,&truncated);

        fprintf(stdout,"Connecting to MDDB database succeded with INFO, SQLSTATE: %s, Message: %s\n",my_sql_s
            err_msg);
    } else {

        SQLSMALLINT rec = 1; SQLCHAR  my_sql_state[SQL_SQLSTATE_SIZE+1];  SQLINTEGER my_err_code; SQLSMALLINT
            truncated;
        SQLCHAR  err_msg[1024];
        SQLGetDiagRec(SQL_HANDLE_DBC,ret.dbc_handle,rec,my_sql_state,&my_err_code,err_msg,1024-1,&truncated);

        fprintf(stderr,"Connecting to database failed with SQLSTATE %s\n",my_sql_state);
        exit(EXIT_FAILURE);
```

Doch ach

- Win32-API Kollegen haben leider wohl auch mitgemischt

- C-API (for the better or the worse)

```
_Bool is_already_uploaded(measurement_set *restrict mes_set,char *file_id)
{
    dbc_stmt mddb = mddb_connect();
    SQLLEN MY_SQL_NULL_DATA = SQL_NULL_DATA;
    SQLLEN MY_SQL_NTS = SQL_NTS;
    SQLCHAR mes_set_query[512] = "SELECT measurements.measurement_id FROM measureme
       WHERE file_id = ? AND product_id = (SELECT product_id from products where pr

    if(mes_set->channel_dbc == NULL)
    {
        strcat(mes_set_query,"is null");
    }else
    {
        strcat(mes_set_query,"= ?");
    }
    SQLDEBUG(SQLPrepare(mddb.stmt_handle,mes_set_query,SQL_NTS));

    SQLDEBUG(SQLBindParameter(mddb.stmt_handle,1,SQL_PARAM_INPUT,
                SQL_C_CHAR,SQL_VARCHAR,64,0,
                file_id,strlen(file_id)+1,
                &MY_SQL_NTS));

    SQLDEBUG(SQLBindParameter(mddb.stmt_handle,2,SQL_PARAM_INPUT, ...
    SQLDEBUG(SQLBindParameter(mddb.stmt_handle,3,SQL_PARAM_INPUT,
                SQL_C_SLONG,SQL_INTEGER,0,0,
                mes_set->order_number,0,
                NULL
                ));
    SQL_TIMESTAMP_STRUCT tmp_measurement_time;
tmp_measurement_time.year = 1900 + mes_set->measurement_time->tm_year; ...
SQLDEBUG(SQLBindParameter(mddb.stmt_handle,4,SQL_PARAM_INPUT, ...
    SQLDEBUG(SQLBindParameter(mddb.stmt_handle,5,SQL_PARAM_INPUT, ...
    if(mes_set->channel_dbc != NULL)
    {
   SQLDEBUG(SQLBindParameter(mddb.stmt_handle,6,SQL_PARAM_INPUT, ...
    }


    SQLDEBUG(SQLExecute(mddb.stmt_handle));

    SQLULEN row_count;
```

Sprung zu C++

- Return-Werte und Error-Präprozessor-Makros?!
- Ressourcen-Management?!
- Hidden State?

```
y {
  //  if (!LoadLibrary("libsoci_core_3_2.dll") || !LoadLibrary("libsoci_odbc_3_2.dll")) //Dynamic linking of odb
  soci::register_factory_odbc(); //Hopefully static linking will work
  soci::connection_parameters main_session_parameters(soci::odbc, MDDB_Service::mddb_con_string);
  main_session_parameters.set_option("odbc_option_driver_complete", "0");

  /*? connection pool for the directory objects*/
  auto conn_pool =
          std::make_shared<MDDB_Service::util::RAII_connection_pool>(std::max(std::thread::hardware_concurrency(
  /*? Vector including all momentarily active directories */
  std::vector<std::unique_ptr < MDDB_Service::Directory>> directories;
```

```
inline void Directory::register_file(File &file_to_register) {
    std::unique_ptr<soci::session> mddb(new soci::session(pool->pool));
    try {
        std::string file_id; soci::indicator file_id_ind;
        *mddb << "SELECT file_id FROM file_view WHERE file_name = :name AND directory = :path",
                soci::use(file_to_register.name, "name"), soci::use(path, "path"), soci::into(file_id,file_id_ind);
        if (!mddb->got_data()) {
            *mddb << "INSERT INTO files(file_name,directory_id) VALUES (:name,(select directory_id from directories wher
                    soci::use(file_to_register.name, "name"), soci::use(path, "path");
            assert(file_to_register.name.length() > 0 && path.length() > 0);
            *mddb << "SELECT file_id FROM file_view WHERE file_name = :name AND directory = :path",
                soci::use(file_to_register.name, "name"), soci::use(path, "path"), soci::into(file_id,file_id_ind);
        }
        assert(file_id_ind == soci::i_ok);
        file_to_register.uid = std::move(file_id);
    } catch (soci::odbc_soci_error const &e) {
        std::cerr << "Cought execption when tried to register file: " << e.what() << util::get_soci_odbc_exp_text(e);
        std::rethrow_exception(std::current_exception());
    } catch (std::exception const &e) {
        std::cerr << "Cought execption when tried to register file: " << e.what() << std::endl;
        std::rethrow_exception(std::current_exception());
    }
}
```

End of line?

- Typ-Sicherheit
- Immer noch SQL im Code
- Bind dies, bind das, bind jenes
- Oft Laufzeit DB-Fehlermeldungen

ORM FTW

- Object Relational Mapping
- Typ-Sicherheit
- Kein Query-Rumdoktern
- Gleich die „Fremdschlüssel“ laden

```
odb::sqlite::database db ("people.db");

person john ("john@doe.org", "John Doe", 31);
person jane ("jane@doe.org", "Jane Doe", 29);

odb::transaction t (db.begin ());

db.persist (john);
db.persist (jane);

typedef odb::query<person> person_query;

for (person& p: db.query<person> (person_query::age < 30));
  cerr << p << endl;

jane.age (jane.age () + 1);
db.update (jane);

t.commit ();
```

```
private:
    std::shared_ptr<odb::database> mddb;
    mutable std::vector<mddb_type> cache;
    mutable std::future<decltype(cache)> cache_future;

    static std::vector<mddb_type>
        load_cache(
        std::shared_ptr<odb::database> mddb,
        const odb::query<mddb_type> &query = odb::query<mddb_type>{},
        std::function<void(void)> callback = std::function<void(void)>{})
    {
        std::vector<mddb_type> container;
        odb::session s;
        odb::transaction t(mddb->begin());
        //t.tracer(odb::stderr_tracer);
        odb::result<mddb_type> results(mddb->query<mddb_type>(query));
        std::move(results.begin(), results.end(), std::inserter(container,

        if (callback){ callback(); }
        return container;
    }
};
template <>
struct mddb_type_trait<views::order_product>
{
```

```
#pragma db object table("measurement_data") bulk(500) session pointer(std::shared_ptr<meas
        class measurement_data
        {
        public:
#pragma db column("measurement_data_id") type("uniqueidentifier") not_null id
            boost::uuids::uuid measurement_data_id;

#pragma db column("measurement_id") not_null
            std::weak_ptr<measurement> measurement;

#pragma db column("measurement_name") type("nvarchar(128)") not_null
            std::wstring measurement_name;

#pragma db column("measurement_value") type("float(53)") null
            boost::optional<double> measurement_value;

#pragma db column("measurement_unit") type("nvarchar(32)") null
            boost::optional<std::wstring> measurement_unit;

#pragma db column("measurement_flag") not_null
            std::shared_ptr<measurement_flag> measurement_flag;

#pragma db column("measurement_lower_spec") type("float") null
            boost::optional<double> measurement_lower_spec;

#pragma db column("measurement_upper_spec") type("float") null
            boost::optional<double> measurement_upper_spec;

            measurement_data() {
                boost::uuids::nil_generator uuid_gen;
                this->measurement_data_id = uuid_gen();
            }

            bool operator==(const measurement_data& r) const {
                return this->measurement_data_id == r.measurement_data_id;
            }
            bool operator<(const measurement_data& r) const {
                return this->measurement_data_id < r.measurement_data_id;
            }
```

```
odb::query<MDDB_Service::MDDB_Web::measurement> get_query(const std::wstring &measurement_parameter_name)
    odb::query<MDDB_Service::MDDB_Web::measurement> query;
    odb::query<mp_grouped_by_order_t> product_query =
        odb::query<MDDB_Service::MDDB_Web::product>::product_name.equal(this->product_name);

    odb::query<mp_grouped_by_order_t> measurement_name_query =
        odb::query<MDDB_Service::MDDB_Web::measurement_data>::measurement_name.equal(measurement_paramete

    odb::query<mp_grouped_by_order_t> test_type_query;
    if (!this->test_type_names.empty()){//ODB in_range has a confirmed bug when the range is empty
        test_type_query = odb::query<MDDB_Service::MDDB_Web::test_type>::
            test_type_name.in_range(this->test_type_names.cbegin(), this->test_type_names.cend());
        query +=
            (odb::query<mp_grouped_by_order_t>(product_query) && measurement_name_query && odb::query<mp_
    }
    else{
        query += (odb::query<mp_grouped_by_order_t>(product_query && measurement_name_query));
    }
    return query;
}
odb::query<MDDB_Service::MDDB_Web::measurement> get_query(
    const std::wstring &measurement_parameter_name,
    int32_t order_number)
    const{
    auto base_query =
        get_query(measurement_parameter_name);
    auto extended_part = odb::query<order>::order_number.equal(order_number);

    return (base_query && extended_part);
}
```

```
    }
    //Get new ones
    std::map<std::wstring, std::future<std::vector<mp_grouped_by_order_t>>> new_ones;
    for (const auto &mp : measurement_names){
        new_ones.emplace(mp,
            std::async(std::launch::async, load_mp_grouped_by_order,
            mddb, get_query(mp)));
    }
    for (auto &new_item : new_ones){
        mp_grouped_by_order_cache.emplace(new_item.first, new_item.second.get());
    }
}
```