数据库应用系统

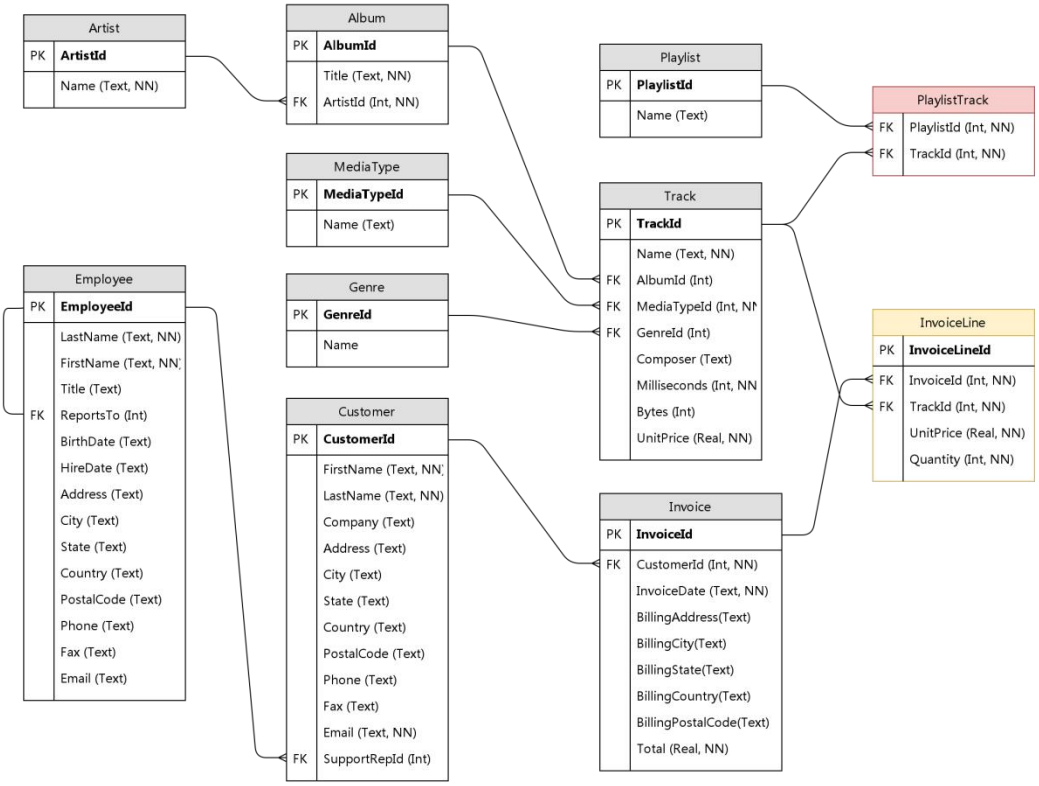
一．系统背景意义

在数据库教学中，常常会涉及到讲解数据表结构和数据表之间关系。普通的教学只能通过 ERD 图来讲解数据库的表结构和数据表之间的关系，无法直观的让学生理解。并且 SQL 语句常常非常的抽象，写起来也非常容易出错，学生在练习是总是一头雾水，很难快速的去理解数据库的运作方式。基于以上问题，我们开发了一种数据库可视化系统。该系统能够直观的反映各个表之间的关系，并且能够直接看到每一个对象的查询语句和方便的得到 SQL 的查询的结果。

该系统基于 web 开发，底层的查询数据库连接查询功能使用 C++开发，能够做到跨平台使用。

本系统采用开源的 Chinook 数据库作为数据源，该数据库符合第一范式、第二范式、第三范式和 BC 范式的要求。我们在该数据库上创建了两个 VIEW，方便进行查询操作。

下图为 Chinook 数据库的 ERD。



二．需求分析

2.1 语言选择

由于 C/C++主要用来做底层开发，无法找到一套非常好的图形库能来适用该项目，我们选择了 web 作为我们的展示平台，但是底层的数据库连接和查询操作还是由 C++来完成。使用 C++来开发该项目的后端，CMAKE 进行编译，方便跨平台使用。

2.2 后端

后端使用 C++开发，主要负责数据库连接、查询和处理来自前端的查询请求。同时也要负责返回前端的页面请求。在运行时需要输出查询的日志记录，方便进行开发调试和后续维护。

2.3 前端

2.3.1 顶部导航栏

顶部导航栏应该包括一些基本的操作按钮，包括系统使用说明按钮，结果表打开按钮，当前 SQL 语句的复制按钮。和当前 SQL 语句的显示区域。

2.3.2 左侧导航栏

左侧导航栏应该包括该系统的展示的节点的导航条目，包括了艺术家、专辑、艺术体裁、播放列表、雇员、客户、发票的导航条目。基本设置按钮，用来设置最大节点数和最大表格记录数。

2.3.3 SQL 结果窗口

该窗口应该包括两个部分，第一个部分为 SQL 语句查询区域，第二个部分为 SQL 语句查询结果表。

SQL 语句查询区域应该由查询语句输入框和查询按钮组成。

查询结果表应该能够展示所有种类的 SQL 语句查询结果。包括了错误结果，单记录结果，多记录结果和空结果。

2.3.4 系统提示

为了给用户带来更好的使用体验，我们在用户执行每一个关键操作时候都有系统提示，让用户对于该系统有一个非常清晰的掌控。

2.3.5 chinook 问题内置

将 chinook 中的问题直接内置在该系统，方便用户直接学习该问题的解法。

2.3.6 节点展示区域

该区域使用 d3.js 库驱动，能够对用户的操作作出流畅的相应，该部分包括：

节点单击拖拽事件，能够对结点进行拖动，并对节点位置进行重新调整。

节点双击展开/折叠事件。当用户双击时能对未展开的节点进行展开，对已展开的节点进行折叠。

鼠标悬浮事件，能够对鼠标的悬浮事件进行相应，将当前节点的 SQL 查询语句暂时在顶部导航栏。

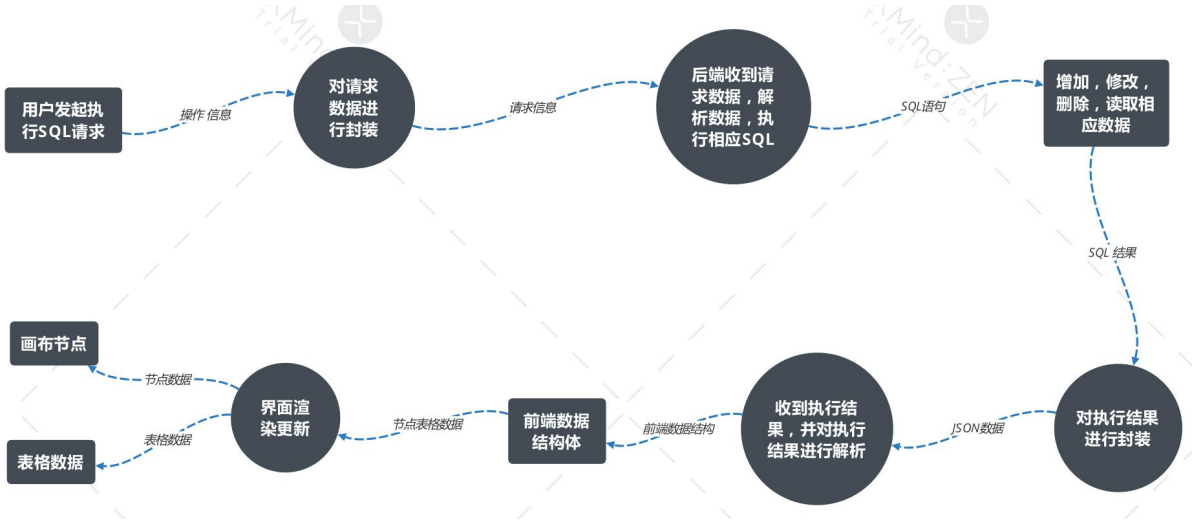
画布缩放事件，当用户对画布进行拖拽缩放时能够对画布进行相应的调整，以适应用户的需求。

节点初始化，当用户切换到其他节点时，能够对画布状态进行重新初始化。

三．系统结构及功能

3.1 系统结构

3.2 数据流图



3.3 关键代码

3.3.1 数据库连接

```
sqlite3 *db;if (SQLITE_OK != sqlite3_open_v2(DATABASE_FILE, &db,
SQLITE_OPEN_READONLY, NULL)){
    std::cerr << "Can't open database: " << sqlite3_errmsg(db) << std::endl;
    return 1;}
log_info("Open database successfully!");
```

3.3.2 查询和回调代码

```
int query_callback(void *data, int argc, char **argv, char **azColName){
    QueryResult *query_result = (QueryResult *)data;
    if (!query_result->has_set_header)
        query_result->set_header(argc, azColName);

    query_result->set_record(argv);
    std::string json;
    return 0;}
int query(sqlite3 *db, QueryResult *query_result, char const *sql){
    char *zErrMsg = NULL;
    if (SQLITE_OK != sqlite3_exec(db, sql, query_callback, query_result,
&zErrMsg))
    {
        query_result->has_error = 1;
        query_result->error_msg = zErrMsg;
        return 0;
    }
    return 1;}
```

3.3.3 接收和执行前端 SQL 请求代码

```
svr.Post("/chinook/query", [db](const Request &req, Response &res) {
    if (!req.has_param(SQL))
    {
        log_err("Request /query must have sql parameter!");
        res.set_content("Request /query must have sql parameter!\n",
"text/plain");
        return;
    }
    std::string sql = req.get_param_value(SQL);

    log_info("执行 SQL: ", sql);
    QueryResult query_result;
    if (!query(db, &query_result, sql.c_str()))
    {
        log_err(query_result.error_msg);
        res.set_content(query_result.error_msg.c_str(), "text/plain");
        return;
    }
    log_info("执行成功!");
    std::string json;
    query_result.dump_json(json);
    res.set_content(json.c_str(), "application/json");});
```

3.3.4 SQL 结果解析封装代码

```
class QueryResult{public:
    int has_error;
    int has_set_header;
    int col_count;
    int record_count;
    std::string error_msg;
    std::vector<std::string> header;
    std::vector<std::vector<std::string>> records;
    QueryResult() : has_error(0), has_set_header(0)
    {
    }
    ~QueryResult()
    {
    }
    void set_header(int argc, char **azColName)
    {
        col_count = argc;
        for (int i = 0; i < col_count; ++i)
            header.push_back(azColName[i]);
        has_set_header = 1;
    }
    void set_record(char **argv)
    {
        std::vector<std::string> _record;
        records.push_back(_record);
        std::vector<std::string> &record = records.back();
        for (int i = 0; i < col_count; ++i){
            record.push_back(argv[i] ? argv[i] : "");
        }
        ++record_count;
    }
    std::string &dump_json(std::string &json)
    {
        json += "[";
        dump_string_vector(json, header);

        for (std::vector<std::vector<std::string>>::iterator it =
records.begin(), end = records.end(); it != end; ++it)
        {
            json += ",";
            dump_string_vector(json, *it);
        }
        json += "]";
        return json;
    }};
```

3.3.5 前端数据库模型代码

```
const T = {
    PK: 1 << 0, //主键
    KEY: 1 << 1, //普通字段
    S_FK: 1 << 2, //一对一外键
    M_FK: 1 << 3, //一对多外键
    D_KEY: 1 << 4, //简略展示字段 ,定义会在节点上展示的字段
}

const database = { //数据库信息用来创建 SQL 查询语句。
    Artist: {
        column: ['ArtistId', 'Name'],
        column_name: ['ID', '名称', '专辑'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.M_FK],
        FK_table: [null, null, 'Album'],
    },
    Album: {
        column: ['AlbumId', 'Title', 'ArtistId'],
        column_name: ['ID', '标题', '艺术家', '音乐'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.S_FK, T.M_FK],
        FK_table: [null, null, 'Artist', 'Track'],
    },
    MediaType: {
        column: ['MediaTypeId', 'Name'],
        column_name: ['ID', '文件编码', '音乐'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.M_FK],
        FK_table: [null, null, 'Track'],
    },
    Genre: {
        column: ['GenreId', 'Name'],
        column_name: ['ID', '艺术体裁', '音乐'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.M_FK],
        FK_table: [null, null, 'Track'],
    },
    Track: {
        column: ['TrackId', 'Name', 'AlbumId', 'MediaTypeId', 'GenreId',
    'Composer', 'Milliseconds', 'Bytes', 'UnitPrice'],
```

```
        sql_fun: [, , , , , , d => `strftime('%M:%S',${d}/1000,'unixepoch')`, d => `printf('%,d kB',${d}/1024)`, d => `'$'||${d}`],
        column_name: ['ID', '歌曲名称', '专辑', '文件编码', '艺术体裁', '作曲家', '时长', '大小', '价格', '播放列表', '发票记录'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.S_FK, T.S_FK, T.S_FK, T.KEY, T.KEY, T.KEY, T.KEY, T.M_FK, T.M_FK],
        FK_table: [null, null, 'Album', 'MediaType', 'Genre', null, null, null, null, 'TrackList', 'InvoiceLine'],
    },
    Playlist: {
        column: ['PlaylistId', 'Name'],
        column_name: ['ID', '歌单', '音乐列表'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.M_FK],
        FK_table: [null, null, 'ListTrack'],
    },
    TrackList: {
        column: ['TrackId', 'PlaylistId', 'Name'],
        column_name: ['音乐 ID', '歌单 ID', '歌单', '音乐列表'],
        column_type: [T.KEY, T.PK, T.KEY | T.D_KEY, T.M_FK],
        FK_table: [null, null, null, 'ListTrack'],
    },
    ListTrack: {
        column: ['PlaylistId', 'TrackId', 'Name', 'AlbumId', 'MediaTypeId', 'GenreId', 'Composer', 'Milliseconds', 'Bytes', 'UnitPrice'],
        sql_fun: [, , , , , , , d => `strftime('%M:%S',${d}/1000,'unixepoch')`, d => `printf('%,d kB',${d}/1024)`, d => `'$'||${d}`],
        column_name: ['歌单 ID', '音乐 ID', '歌曲名称', '专辑', '文件编码', '艺术体裁', '作曲家', '时长', '大小', '价格', '播放列表', '发票记录'],
        column_type: [T.KEY, T.PK, T.KEY | T.D_KEY, T.S_FK, T.S_FK, T.S_FK, T.KEY, T.KEY, T.KEY, T.KEY, T.M_FK, T.M_FK],
        FK_table: [null, null, null, 'Album', 'MediaType', 'Genre', null, null, null, null, 'TrackList', 'InvoiceLine'],
    },
    Employee: {
        column: ['EmployeeId', 'LastName', 'FirstName', 'Title', 'ReportsTo', 'BirthDate', 'HireDate', 'Address', 'City', 'State', 'Country', 'PostalCode', 'Phone', 'Fax', 'Email'],
        column_name: ['ID', '姓', '名', '职称', '直属上级', '生日', '入职时间', '详细地址', '城市', '州', '国家', '邮政编码', '电话', '传真', '邮箱', '直属下级', '客户'],
```

```
        column_type: [T.PK, T.KEY | T.D_KEY, T.KEY | T.D_KEY, T.KEY, T.S_FK, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.M_FK, T.M_FK],
        FK_table: [null, null, null, null, 'Employee', null, null, null, null, null, null, null, null, null, null, 'Employee', 'Customer'],
    },
    Customer: {
        column: ['CustomerId', 'FirstName', 'LastName', 'Company', 'Address', 'City', 'State', 'Country', 'PostalCode', 'Phone', 'Fax', 'Email', 'SupportRepId'],
        column_name: ['ID', '姓', '名', '公司', '详细地址', '城市', '州', '国家', '邮政编码', '电话', '传真', '邮箱', '客户经理', '发票'],
        column_type: [T.PK, T.KEY | T.D_KEY, T.KEY | T.D_KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.S_FK, T.M_FK],
        FK_table: [null, null, null, null, null, null, null, null, null, null, null, null, 'Employee', 'Invoice'],
    },
    Invoice: {
        column: ['InvoiceId', 'CustomerId', 'InvoiceDate', 'BillingAddress', 'BillingCity', 'BillingState', 'BillingCountry', 'BillingPostalCode', 'Total'],
        sql_fun: [, , , , , , , , d => `'$'||${d}`],
        column_name: ['ID', '客户', '发票日期', '发票地址', '城市', '州', '国家', '邮政编码', '总金额', '发票记录'],
        column_type: [T.PK, T.S_FK, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY, T.KEY | T.D_KEY, T.M_FK],
        FK_table: [null, 'Customer', null, null, null, null, null, null, null, 'InvoiceLine'],
    },
    InvoiceLine: {
        column: ['InvoiceLineId', 'InvoiceId', 'TrackId', 'UnitPrice', 'Quantity'],
        sql_fun: [, , , d => `'$'||${d}`, ],
        column_name: ['ID', '发票', '音乐', '单价', '数量'],
        column_type: [T.PK, T.S_FK, T.S_FK, T.KEY | T.D_KEY, T.KEY | T.D_KEY],
        FK_table: [null, 'Invoice', 'Track', null, null],
    },
};
```

3.3.6 前端创建 SQL 语句

```
function build_SQL(TABLE, WHERE) {
    let db = database[TABLE];
    let COL = db.column.map(function (ele, i) {
        return `${db.sql_fun&&db.sql_fun[i]?db.sql_fun[i](db.column[i]):db.column[i]} AS ${db.column_name[i]}`;
    }).join(',');
    let sql = `SELECT ${COL} FROM ${TABLE}${WHERE?' WHERE ':''}${WHERE?WHERE:''};`;
    return sql;
}
```

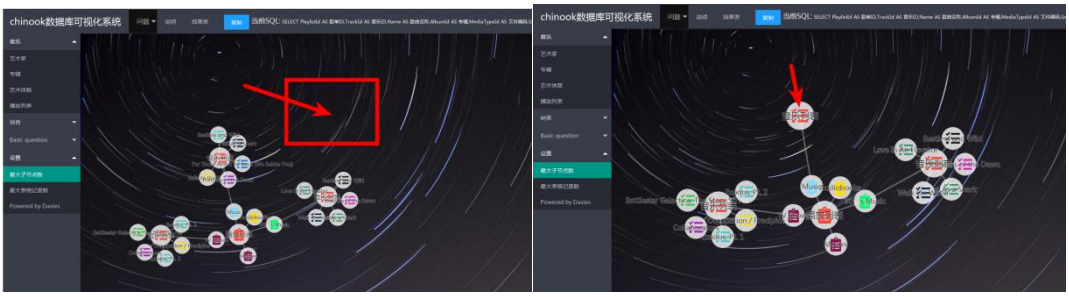
3.3.7 其他部分代码

由于其他部分代码过多，并且与数据库不相关或者相关性很小。这里不作展示，如需要请查看项目源代码。

3.4 系统功能

3.4.1 画布导航

- 拖动空白区域进行画布平移，任意唯一使用鼠标滚轮进行缩放。
- 双击节点进行节点的展开或者折叠。
- 鼠标悬停于节点进行 SQL 语句查看，拖拽节点进行位置调整。



3.4.2 布局说明

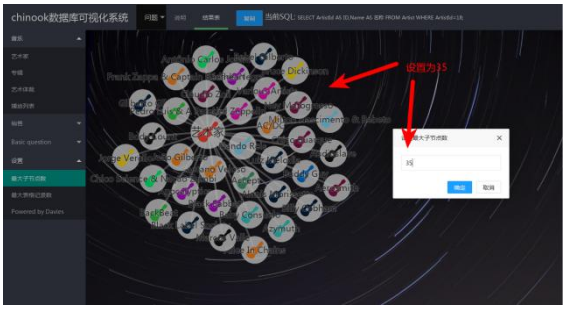
见以下图示说明：

3.4.3 查询结果数据表

见以下图示说明：

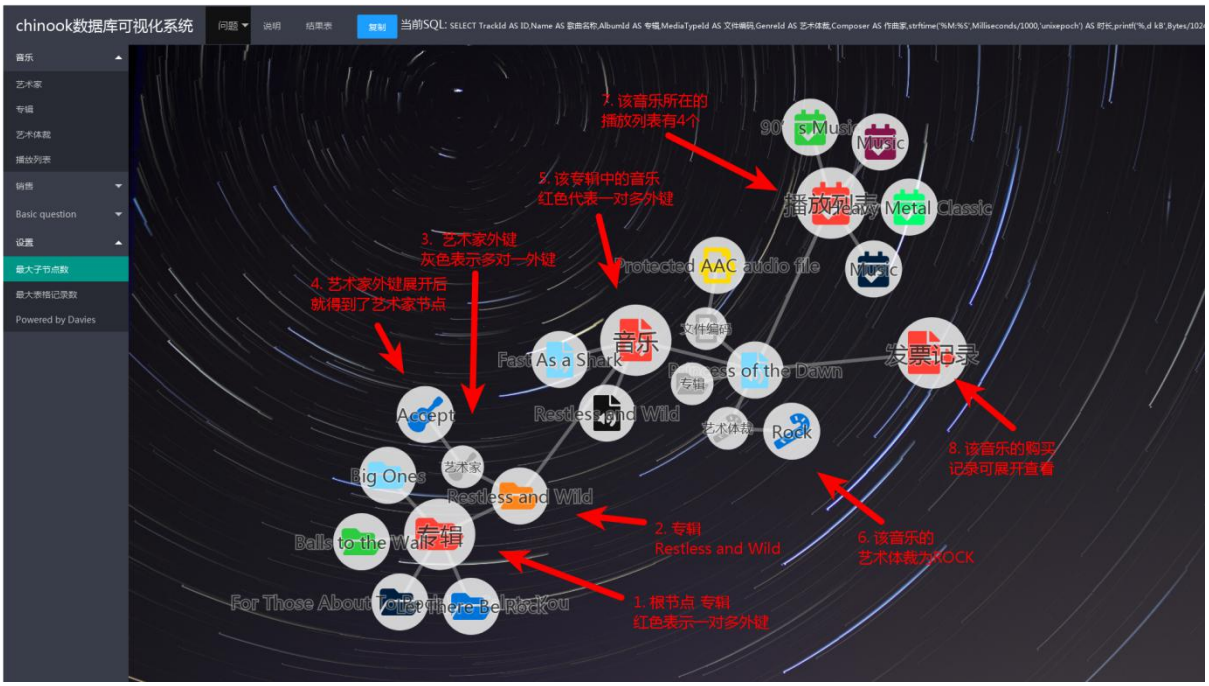
3.4.4 设置

下图为最大子节点数设置的图示说明，最大表格记录数同理。最大子节点数默认为 8，最大表格记录数默认为 200，默认设置你可以在 config.js 中修改。



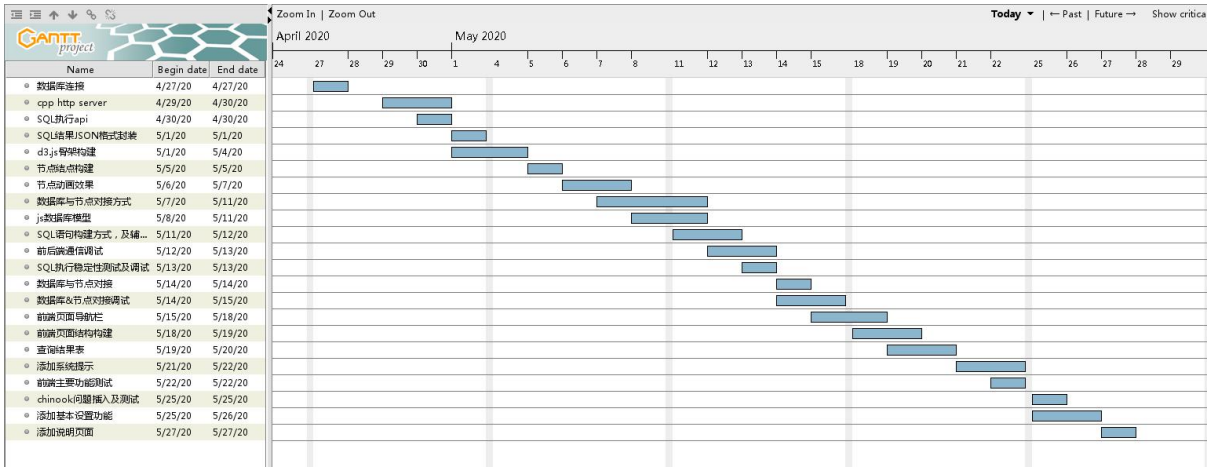
3.4.5 节点意义解释

见以下图示说明（请按顺序阅读）：



四．分工及进度

4.1 项目进度



4.2 项目分工

任务名称	负责人	开始时间	用时
数据库连接	Davies	2020-04-27	1 天
cpp http server	Davies	2020-04-29	2 天
SQL 执行 api	Davies	2020-04-30	1 天
SQL 结果 JSON 格式封装	Davies	2020-05-01	1 天
d3.js 骨架构建	Davies	2020-05-01	2 天
节点结点构建	Davies	2020-05-05	1 天
节点动画效果	Davies	2020-05-06	2 天
数据库与节点对接方式	Davies	2020-05-07	3 天
js 数据库模型	Davies	2020-05-08	2 天
SQL 语句构建方式，及辅助函数	Davies	2020-05-11	2 天
前后端通信调试	Davies	2020-05-12	2 天
SQL 执行稳定性测试及调试	Davies	2020-05-13	1 天
数据库与节点对接	Davies	2020-05-14	1 天
数据库&节点对接调试	Davies	2020-05-14	2 天
前端页面导航栏	Davies	2020-05-15	2 天
前端页面结构构建	Davies	2020-05-18	2 天
查询结果表	Davies	2020-05-19	2 天
添加系统提示	Davies	2020-05-21	2 天
前端主要功能测试	Davies	2020-05-22	1 天
chinook 问题插入及测试	Davies	2020-05-25	1 天
添加基本设置功能	Davies	2020-05-25	2 天
添加说明页面	Davies	2020-05-27	1 天

五．软件运行效果

主要功能模块核心代码见 ***3.2 关键代码***部分。
运行图示见 ***3.3 系统功能***部分。

六．总结

整个项目结构还是非常复杂的，项目开始时每个人都需要对该项目的结构有一个清楚的认识，这样每个人才知道什么时间做哪一部分内容，有利于提高团队效率。项目开发的时候应该边写接口边测试，这样才能及时发现错误及时解决，以免到最后全是 bug，导致项目无法继续进行。

七．参考文献

[1]杨晶晶.网站管理系统中数据库设计的应用[J].福建茶叶,2020,42(04):39.
[2]史雪辉,穆加艳,魏兵.一种基于轻量级数据库的任务管理系统[J].雷达与对抗,2020,40(01):64-68.
[3]冯小洁.以体系结构为中心的数据库设计方法及应用[J].中国教育信息化,2020(05):89-93.
[4]丁红艳.基于计算机软件工程的数据库编程技术[J].科学技术创新,2020(06):90-91.
[5]王竞阳.信息管理中的计算机数据库技术应用[J].无线互联科技,2020,17(04):160-161.
[6]梁利亭.计算机软件数据库设计原则探讨[J].信息与电脑(理论版),2020,32(02):116-118.
[7]何彬,冯巍,蒋帅.高铁故障数据库的建立与应用[J].电子技术与软件工程,2020(01):113-114.
[8]游思奇.计算机软件工程的数据库编程技术[J].电子技术与软件工程,2020(01):135-136.
[9]吴小欣.突出实时测控软件数据库系统设计与实现[J].电子设计工程,2020,28(01):23-26+31.
[10]房婷玲，曹菡，王长缨。基于微信公众平台的智慧课堂教学模式初探--以“数据库原理与应用”课程为例[J].工业和信息化教育，2017,（03）：77-83.
[11]基于关系数据库的关键词查询[J]. 林子雨,杨冬青,王腾蛟,张东站. 软件学报. 2010(10)
[12]S-CBR:基于数据库模式展现数据库关键词检索结果[J]. 彭朝晖,张俊,王珊. 软件学报. 2008(02)