# 概述

# 分类

## MySQL HeatWave

## MySQL Enterprise

## MySQL Standard Edition

## MySQL Classic Edition

## MySQL Cluster CGE

## MySQL Embedded（OEM/ISV）

# 主要模块

源码结构（MySQL-5.5.0-m2）

BUILD: 内含在各个平台、各种编译器下进行编译的脚本。如compile-pentium-debug表示在pentium架构上进行编译的脚本。

client: 客户端工具，如mysql, mysqladmin之类。

cmd-line-utils: readline, libedit工具。

config: 给aclocal使用的配置文件。

dbug: 提供一些调试用的宏定义。

extra: 提供innochecksum，resolveip等额外的小工具。

include: 包含的头文件。

libmysql: 库文件，生产libmysqlclient.so。

libmysql\_r: 线程安全的库文件，生成libmysqlclient\_r.so。

libservices: 5.5.0中新加的目录，实现了打印功能。

man: 手册页。

mysql-test: mysqld的测试工具一套。

mysys: 为跨平台计，MySQL自己实现了一套常用的数据结构和算法，如string, hash等。

netware: 在netware平台上进行编译时需要的工具和库。

plugin: mysql以插件形式实现的部分功能。

pstack: 异步栈追踪工具。

regex: 正则表达式工具。

Scripts: 提供脚本工具，如mysql\_install\_db等

sql: mysql主要代码，将会生成mysqld文件。

sql-bench: 一些评测代码。

sql-common: 存放部分服务器端和客户端都会用到的代码。

storage: 存储引擎所在目录，如myisam, innodb, ndb等。

这个目录包含了所谓的MySQL存储引擎（storage engine）。存储引擎是数据库系统的核心，封装了数据库文件的操作，是数据库系统是否强大最重要的因素。MySQL实现了一个抽象接口层，叫做handler（sql/handler.h），其中定义了接口函数，比如：ha\_open，ha\_index\_end，ha\_create等等，存储引擎需要实现这些接口才能被系统使用。对于具体每种引擎的特点，推荐大家去看mysql的在线文档: <http://dev.mysql.com/doc/refman/5.1/en/storage-engines.html>。

Strings: string库。

support-files: my.cnf示例配置文件。

tests: 测试文件所在目录。

unittest: 单元测试。

vio: virtual io系统，是对network io的封装。

win: 给windows平台提供的编译环境。

zip: zip库工具

## client

## mysys

## plugin

参考：

<https://blog.csdn.net/fpcc/article/details/120596120>

<https://www.cnblogs.com/zhedan/p/12518091.html>

## sql

## sql-common

## storage

## Strings

## vio

# 架构

参考：<https://cloud.tencent.com/developer/article/2119477>

# 存储引擎

参考：

<https://blog.csdn.net/lzb348110175/article/details/106555504>

Secondary Engine：<http://mysql.taobao.org/monthly/2020/11/04/>

一些流行引擎存储格式简介：<http://mysql.taobao.org/monthly/2017/10/04/>

## InnoDB

## MyISAM

## Archive

## Blackhole

## CSV

参考：

CSV 引擎详解：<http://mysql.taobao.org/monthly/2021/10/05/>

## temptable

## TokuDB

参考：<https://article.itxueyuan.com/0e3Xkp>

## Federated

## heap

## rocksdb

## Spider

参考：<https://cloud.tencent.com/developer/article/1992931>

# 主要数据结构

参考：

语法解析调用栈：<https://www.cnblogs.com/jkin/p/16848540.html>

## THD

代码位置：sql/sql\_class.h

包含处理用户请求时需要的相关数据，每个连接会有一个线程来处理，在一些高层函数中，此数据结构常被当作第一个参数传递。

THD类继承了三个父类：

MDL\_context\_owner：作为Metadata Lock 拥有者相关的接口，其中元数据边界控制（进/退），元数据信息通知等，它是一个虚类。

Query\_arena：管理语句元素的类，主要针对指语句（包括store procedure）解释成抽象语法树后的节点

Open\_tables\_state：管理线程打开和加锁的表的状态，保持了一个临时表信息和锁信息，子类有Open\_tables\_backup

除这些外，还包括与锁相关的数据结构如MDL\_context，接口语法语义相关如LEX，日志和事务以及执行数据和优化参数等。

NET net; // 客户连接描述符

Protocol \*protocol; // 当前的协议

Protocol\_text protocol\_text; // 普通协议

Protocol\_binary protocol\_binary; // 二进制协议

HASH user\_vars; //用户变量的hash值

String packet; // 网络IO时所用的缓存

String convert\_buffer; // 字符集转换所用的缓存

struct sockaddr\_in remote; //客户端socket地址

THR\_LOCK\_INFO lock\_info; // 当前线程的锁信息

THR\_LOCK\_OWNER main\_lock\_id; // 在旧版的查询中使用

THR\_LOCK\_OWNER \*lock\_id; //若非main\_lock\_id, 指向游标的lock\_id

pthread\_mutex\_t LOCK\_thd\_data;

//thd的mutex锁，保护THD数据（thd->query, thd->query\_length）不会被其余线程访问到

Statement\_map stmt\_map;

//prepared statements和stored routines 会被重复利用

int insert(THD \*thd, Statement \*statement); // statement的hash容器

class Statement::

LEX\_STRING name;

LEX \*lex; //语法树描述符

bool set\_db(const char \*new\_db, size\_t new\_db\_len)

void set\_query(char \*query\_arg, uint32 query\_length\_arg);

{

pthread\_mutex\_lock(&LOCK\_thd\_data);

set\_query\_inner(query\_arg, query\_length\_arg);

pthread\_mutex\_unlock(&LOCK\_thd\_data);

}

## Lex

MySQL通过THD的父类字段Statement::lex保存解析出来的语法树信息。struct LEX继承于Query\_tables\_list。LEX中涉及的查询语法树的核心成员包括：

unit：语法树顶层节点；

select\_lex：顶层第一个select语句节点；

all\_selects\_list：通过link\_next和link\_prev构成所有select子句的双向链表；

query\_tables：通过next\_global和prev\_global构成所有talbe的双向链表；

query\_tables\_last：指向query\_tables中最后一个元素；

注：其他成员不作描述，在sql/sql\_lex.h源码文件中。

### select语法树

#### 核心数据结构

select语法树由SELECT\_LEX\_UNIT（即st\_select\_lex\_unit）节点和SELECT\_LEX（即st\_select\_lex）节点组成。两个class都继承自st\_select\_lex\_node。SELECT\_LEX的核心成员如下：

table\_list：通过next\_local串联起来的表结构，也就是from后面的表；

top\_join\_list：将from后面的表根据jion语法规则生成的jion树；

where：where语句后面的条件表示树；

group\_list：group分组条件；

having：having条件；

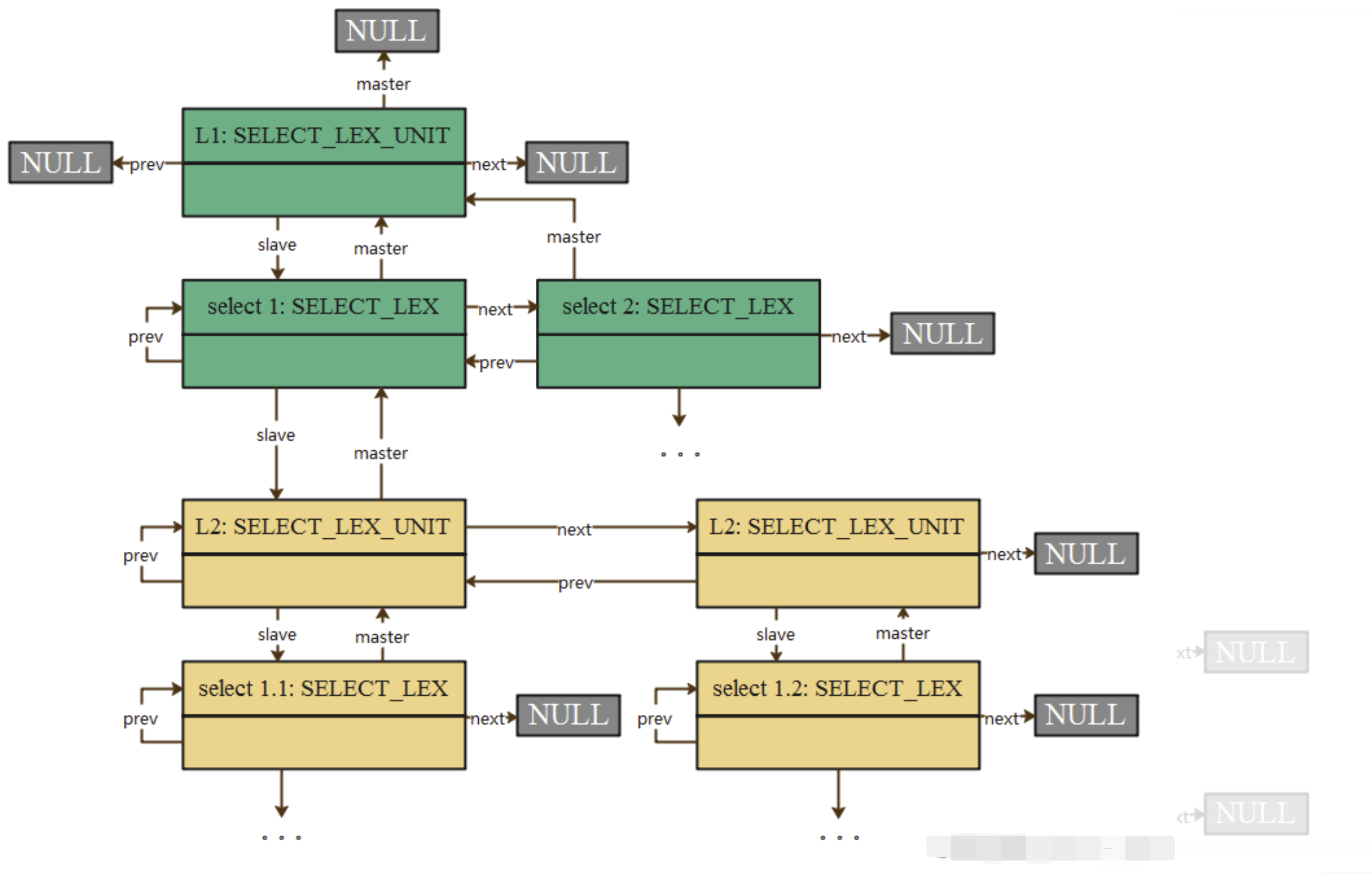
item\_list：select语句后面的字段列表；

order\_list：order排序方式；

select\_limit，offset\_limit：行限制条件；

#### 语法树结构

查询语句的语法树结构如下：



整个语法树由SELECT\_LEX\_UNIT作为顶层，下面挂unit的多个select语句；每个select子句的顶层SELECT\_LEX\_UNIT挂到所属的select主句上面。THD::lex->unit指向顶层L1；THD::lex->select\_lex指向顶层select 1。

### from子句

#### 解析原理

整个from表格解析规则如下：

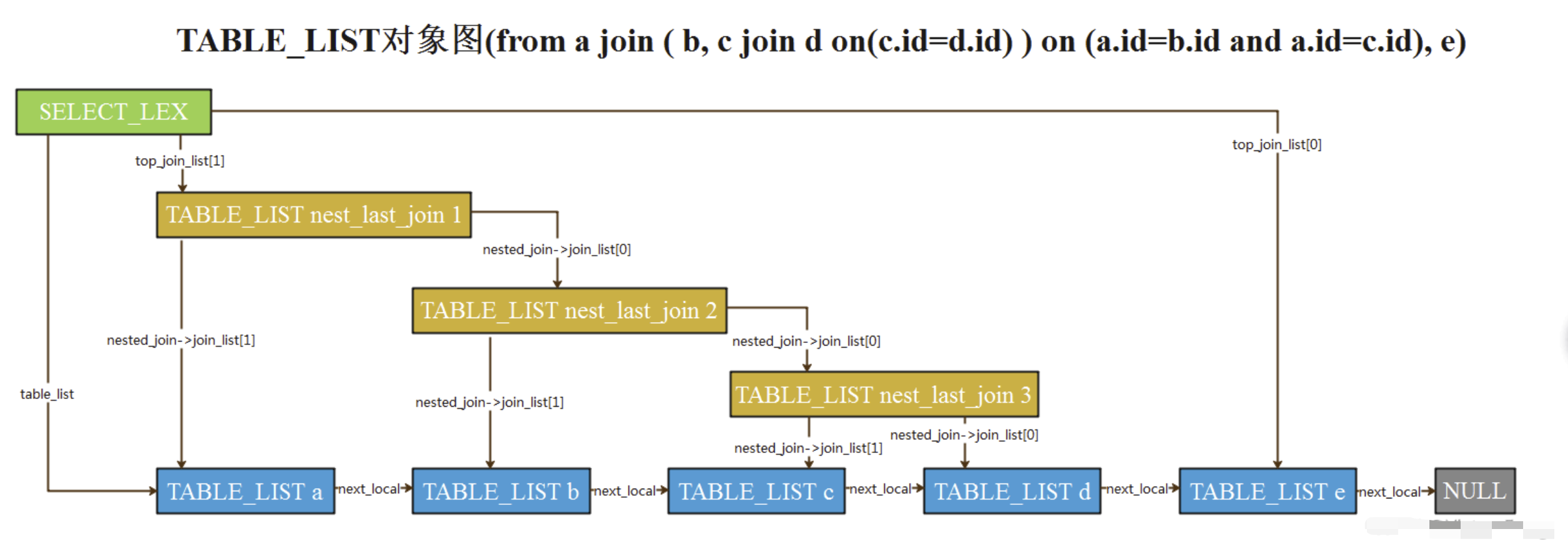
规则1：JOIN表达式：=（JOIN表达式 | JOIN嵌套）JOIN（JOIN表达式 | JOIN嵌套）

规则2：JOIN嵌套：=’(’ （表引用 | JOIN表达式）（，表引用 | ，JOIN表达式）\* ‘)’ | 表引用

也就是说一个jion表达式是由join表达式或join嵌套组成；一个join嵌套可以是一个表引用，或者多个表引用或join表达式通过“,”相隔、且首尾括号包围。

#### 语法树结构

from后面的表解析成TABLE\_LIST结构，所有的TABLE\_LIST按如下方式串联：



### where语句

where语句都是抽象类Item的子类一层一层组成语法树结构。

## NET

**代码位置：**sql/mysql\_com.h

网络连接描述符，对内部数据包进行了封装，是client和server之间的通信协议。

Vio \*vio; //底层的网络I/O socket描述符

unsigned char \*buff,\*buff\_end,\*write\_pos,\*read\_pos; //缓存相关

unsigned long remain\_in\_buf,length, buf\_length, where\_b;

unsigned long max\_packet,max\_packet\_size; //当前值;最大值

unsigned int pkt\_nr,compress\_pkt\_nr; //当前（未）压缩包的顺序值

my\_bool compress; //是否压缩

unsigned int write\_timeout, read\_timeout, retry\_count; //最大等待时间

unsigned int \*return\_status; //thd中的服务器状态

unsigned char reading\_or\_writing;

unsigned int last\_errno; //返回给客户端的错误号

unsigned char error;

## TABLE

参考：

TABLE信息的生命周期：<http://mysql.taobao.org/monthly/2022/01/04/>

**代码位置：**sql/table.h

数据库表描述符，分成TABLE和TABLE\_SHARE两部分。

handler \*file; //指向这张表在storage engine中的handler的指针

THD \*in\_use;

Field \*\*field;

uchar \*record[2];

uchar \*write\_row\_record;

uchar \*insert\_values;

key\_map covering\_keys;

key\_map quick\_keys, merge\_keys;

key\_map keys\_in\_use\_for\_query;

key\_map keys\_in\_use\_for\_group\_by;

key\_map keys\_in\_use\_for\_order\_by;

KEY \*key\_info;

HASH name\_hash; //数据域名字的hash值

MEM\_ROOT mem\_root; //内存块

LEX\_STRING db;

LEX\_STRING table\_name;

LEX\_STRING table\_cache\_key;

enum db\_type db\_type //当前表的storage engine类型

enum row\_type row\_type //当前记录是定长还是变长

uint primary\_key;

uint next\_number\_index; //自动增长key的值

bool is\_view ;

bool crashed;

## FIELD

**代码位置：**sql/field.h

域描述符，是各种字段的抽象基类。

uchar \*ptr; // 记录中数据域的位置

uchar \*null\_ptr; // 记录 null\_bit 位置的byte

TABLE \*table; // 指向表的指针

TABLE \*orig\_table; // 指向原表的指针

const char \*\*table\_name, \*field\_name;

LEX\_STRING comment;

key\_map key\_start, part\_of\_key, part\_of\_key\_not\_clustered;

key\_map part\_of\_sortkey;

enum utype { NONE,DATE,SHIELD,NOEMPTY,CASEUP,PNR,BGNR,PGNR,YES,NO,REL,

CHECK,EMPTY,UNKNOWN\_FIELD,CASEDN,NEXT\_NUMBER,

INTERVAL\_FIELD,BIT\_FIELD, TIMESTAMP\_OLD\_FIELD, CAPITALIZE, BLOB\_FIELD,

TIMESTAMP\_DN\_FIELD, TIMESTAMP\_UN\_FIELD, TIMESTAMP\_DNUN\_FIELD};

…..

virtual int store(const char \*to, uint length,CHARSET\_INFO \*cs)=0;

inline String \*val\_str(String \*str) { return val\_str(str, str); }

## Utility API Calls API

各种核心的工具，例如内存分配，字符串操作或文件管理。标准C库中的函数只使用了很少一部分，C++中的函数基本没用。

void \*my\_malloc(size\_t size, myf my\_flags)

//对malloc的封装

size\_t my\_write(File Filedes, const uchar \*Buffer, size\_t Count, myf MyFlags)

//对write的封装

## Preprocessor Macros处理器宏

Mysql中使用了大量的C预编译，随编译参数的不同最终代码也不同。

#define max(a, b) ((a) > (b) ? (a) : (b)) //得出两数中的大者

do \

{ \

char compile\_time\_assert[(X) ? 1 : -1] \

\_\_attribute\_\_ ((unused)); \

} while(0)

使用gcc的attribute属性指导编译器行为

• Global variables 全局变量

• configuration settings

• server status information

• various data structures shared among threads

主要包括一些全局的设置，服务器信息和部分线程间共享的数据结构。

struct system\_status\_var global\_status\_var; //全局的状态信息

struct system\_variables global\_system\_variables; //全局系统变量