GSP触发器和sql触发器的区别

- 意义一致， 都是用于在执行操作前或者操作后触发一个过程。 广联达触发器指定过程， 并且过程内内不能对数据库操作； sql数据库创建触发器并且绑定到某个操作上（update insert）， 然后指定数据库操作过程。
- sqlite 的设计目的之一就是稳定性， 并且每个版本都会经过大量的测试， 因此sqlite相当的reliable 。如果工程对于bug容忍度比较低， 可以经常查看sqlite 的 bug list， sqlite 会在第一时间把bug发布出来。
- 和C/S 数据库比较， C/S 数据库用于实现企业大量数据的共享。 它们强调扩展性、并发性、集中化管理以及相关操作。SQLite 向设备和应用提供数据的本地存储方案。sqlite强调的是简单、经济、效率、可靠度、独立性

Sqlite 特点

- 无认证
- 内存数据库： ":memory:"做文件名， 或者空文件名。
- 如果试图在一个事务中访问被另一个事务占用的资源（存在冲突的情况）， 该事务就会被阻塞，并且一直尝试读或者写，直到另一个事务结束，或者自己的Timeout。
- Sqlite（QtSql） 没有的特点
  - 自动保存（应该可以设置）
  - 传入回调函数：
    - 数据库级别： 打开关闭、增加表、删除表
    - 表级别： 增删查改记录、 增删查改字段
  - 嵌套表
  - 权限设置

QtSql

- QSqlIndex 表示索引， 可指向表和视图
- QSqlError 可以指定不同数据库、 不同数据库文件相关的信息
- QSqlField 字段
- QSqlRecord 记录
- QSqlQuery sql 语句
- QSqlQueryModel 只读查询结果
- QSqlRelation 外键
- QSqlResult 封装数据库操作
- QSqlTableModel 可读写的表模型
- QDataWidgetMapper 用于把一个model的选区传给某个widget

sqlite 使用过程

```
sqlite 的使用分为4步
```

- 使用 sqlite3_prepare() 创建一个 sqlite3_stmt 类
  - sqlite3_stmt 可以通过执行 sqlite3_reset() 返回执行前状态， 然后再次执行， 这样可以避免多次构造 sqlite3_stmt
  - sqlite3_stmt 生命周期
    - Create the prepared statement object using sqlite3_prepare_v2().
    - Bind values to parameters using the sqlite3_bind_*() interfaces.
    - Run the SQL by calling sqlite3_step() one or more times.
    - Reset the prepared statement using sqlite3_reset() then go back to step 2. Do this zero or more times.
    - Destroy the object using sqlite3_finalize().
- 使用 sqlite3_step() 执行 sqlite3_stmt
- 对每一个 sqlite3_step 执行 sqlite3_column() , 获取查询结果。
- 使用 sqlite3_finalize() 销毁 sqlite3_stmt
- **sqlite3_exec() 可以使用一个函数调用完成以上4步， 它使用回调函数接收返回结果**
- **sqlite3_get_table() 可以使用一个函数调用完成以上4步， 它把返回结果放在堆内存中**

Configuring SQLite

- sqlite3_config() 用于定义全局的、进程级别的 Sqlite 配置。
- sqlite3_config() 必须在打开数据库之前调用
- sqlite3_config() 可用于
  - 定义内存分配器（allocator）
  - 定义进程 error log
  - 定义进程页大小
  - 为线程提供同步机制（锁）
- sqlite3_db_config() 和 sqlite3_config 效果一样， 但它在建立数据库连接以后使用， 是单个连接级别的。

扩展 **Sqlite**

- sqlite3_create_collation()
  - 定义字符串排序函数
- **sqlite3_create_function()**

```
int sqlite3_create_function_v2(
sqlite3 *db,
const char *zFunctionName,
int nArg,
int eTextRep,
void *pApp,
void (*xFunc)(sqlite3_context*,int,sqlite3_value**),
```

```
    void (*xStep)(sqlite3_context*,int,sqlite3_value**),
    void (*xFinal)(sqlite3_context*),
    void(*xDestroy)(void*)
    );
```

- 定义或者重定义 Sql 语句可调用的函数scalar or aggregate
- Aggregate是针对一系列值的操作，返回一个单一的值， 它需要实现 xFinal 和 xStep， xFunc 必须设置为 null
- Scalar 函数是针对一个单一的值的操作，返回基于输入值的一个单一值， 它只需要实现 xFunc
- sqlite3_aggregate_context 第一次调用， 返回一块内存， 之后的调用都会返回这块内存。aggregate 如果有x步，每一步都会调用它获取该内存的地址。结束时， 最好以 sqlite3_aggregate_context(context,0)的方式调用， 以防止内存泄漏；Sqlite 会在回调函数完成后自动释放该内存。
- sqlite3_create_function 通过调用 sqlite3_user_data 获取用户传入的任意数据； 如果要删除已知函数， 把相应的 xFunc 或者 xStep、xFinal设置为null即可。

- sqlite3_create_module()
- sqlite3_vfs_register()

sqlite 虚表机制（virtual Table）

虚表是一种自定义的扩展，允许用户通过代码定制表的数据结构和数据内容；对于数据库引擎，它和普通表一样，允许进行大多数的sql操作。

虚表和普通表的主要不同在于，其表中的数据的来源；对于普通表，来源于数据库的行列值；而对于虚表，来源于用户自定义的函数，可以是数据库中的数据，也可以使其他的外部数据，如：磁盘文件(csv, excel)等；

虚表是sqlite的一种高级特性，它的实现基于sqlite module；
虚表被用于连接数据库引擎和可变的数据源，分为两种：internals and externals；
internal modules的数据来自于数据库文件本身，它的主要目的并不是做普通表不能做的，而是作为智能视图，更具扩展性的、更方便的、更快速的处理一些特定格式的数据；sqlite本身带有两个modules：FTS3和R*Tree，用于全文检索；
external modules的数据来自于数据库文件外部，如cvs、excel文件等；这样，在不导入外部数据到数据库的情况下，用户能够以sql的方式访问和处理外部数据源 （对于10万条记录，速度可以是秒级的）；

Module APIs：

```
    int sqlite3_create_module(sqlite3 *db, const char *name, const sqlite3_module *module, void*udp)
    CREATE VIRTUAL TABLE table_name USING module_name(arg)
```

表操作：

```
    xCreate, xConnect, xDisConnect, xDestroy
    xBestIndex, xFindFunction, xUpdate, xRename
```

表扫描：

```
    xOpen, xClose, xFilter, xNext, xEof
    xRowid, xColumn
```

事务处理：

```
    xBegin, xSync, xCommit, xRollback
```

全文检索

- 全文检索是internals module的一种应用，使用了fts3或fts4 module。

- 该模块优化了对全文索引的插入和查询，减少了用户的工作量；

- 该虚表对应了3~6个影子表，其存储实际的数据；当向虚表中插入文本内容时，相应的数据和索引会被插入到各影子表中

- 对应若干种关键字查询方式；

- 涉及多种toknizer，用户也可以定制词法分析器；

- 索引表采取B-tree的存储方式，每插入一行，都会生成一个新的索引表，这可以降低索引表更新的效率，但是占用了大量的存储空间，因此，需要定期的merge索引表。