Библиотека работы с базой данных системы активного аудита

Для создания базы данных использовался MySQL версии 5.0.45.
В базе данных хранится:

- Вся информация о событиях
- Конфигурация компонент
- Вывод анализаторов

Созданы следующие таблицы:

- Таблица с информацией по сенсорам.
- Таблица событий
- Таблица с параметрами событий
- Таблица команд
- Таблица с параметрами команд

Перейдем к описанию таблиц

Таблица с информацией по сенсорам **Sensor**

Sensor	
SensorType	int
SensorID	char 255
Description	char 255
MessageType	int
EventTime	datetime
Status	int
SocketNameForCommands	char 255

Таблица с информацией по сенсорам

SensorType — тип сенсора.
SensorID — уникальный идентификатор сенсора, по которому можно однозначно восстановить путь к сенсору.
Description — описание сенсора
MessageType — тип сообщения
EventTime — время, когда был отправлен последний пакет от сенсора

Status — текущее состояние сенсора, говорящее о его работоспособности

SocketNameForCommands — имя Unix-domain сокета, через который данный сенсор получает команды.

Таблица событий **MessageEvent**

MessageEvent	
MessageType	Int
SensorID	char 255
MessageID	Smallint
SensorType	Int
EventName	char 255
EventTime	datetime

Таблица событий

Для каждого события возможно наличие параметров, которые хранятся в *MessageEventParam* и соединяются при помощи *MessageID* , где *MessageID* является первичным ключом в таблице *MessageEvent* и вторичным — в*MessageEventParam*. При запросе к базе данных эти таблицы соединяются друг с другом по*MessageID* .

MessageType — тип сообщения

SensorID — идентификатор сенсора, от которого пришло сообщение

MessageID — уникальный идентификатор данного события. Автоматически присваивается базой данных (первичный ключ)

SensorType — тип сенсора

EventName — имя события

EventTime — время, когда произошло данное событие

Таблица с параметрами событий **MessageEventParam**

MessageEventParam	
MyInd	smallint
MessageID	smallint
Type	Int
Name	char 255
Val	char 255

Таблица с параметрами событий

MyInd — уникальный идентификатор параметров, который автоматически присваивается базой данных (первичный ключ)

MessageID — Идентификатор данного события (вторичный ключ)
Type — тип параметра
Name — имя параметра
Val — значение параметра

Таблица команд**MessageCommand**

MessageCommand	
MessageType	int
SensorID	char 255
CommandID	smallint
CommandType	int
EventTime	datetime
CommandName	char 255
Result	int
ResultInfo	char 255

Таблица команд

Для каждой команды возможно наличие параметров, которые хранятся в *MessageCommandParam* и соединяются при помощи*CommandID* , где *CommandID* является первичным ключом в таблице *MessageCommand* и вторичным — в*MessageCommandParam* . При запросе к базе данных эти таблицы соединяются друг с другом по*CommandID* .

MessageType — тип сообщения
SensorID - идентификатор сенсора
CommandID — уникальный идентификатор команды, присваивающийся автоматически базой данных (первичный ключ)
CommandType — тип команды
EventTime — время команды
CommandName — имя команды
Result — результат выполнения команды
ResultInfo — информация о результате выполнения команды

Таблица с параметрами команд **MessageCommandParam**

MessageCommandParam	
MyInd	smallint
CommandID	smallint
Type	int
Name	char 255
Val	char 255

Таблица с параметрами команд

MyInd — уникальный идентификатор параметров, который автоматически присваивается базой данных (первичный ключ)

CommandID — Идентификатор данной команды (вторичный ключ)

Type — тип параметра

Name — имя параметра

Val — значение параметра

Структура БД

База данных имеет структуру таблиц, представленную на следующем рисунке.

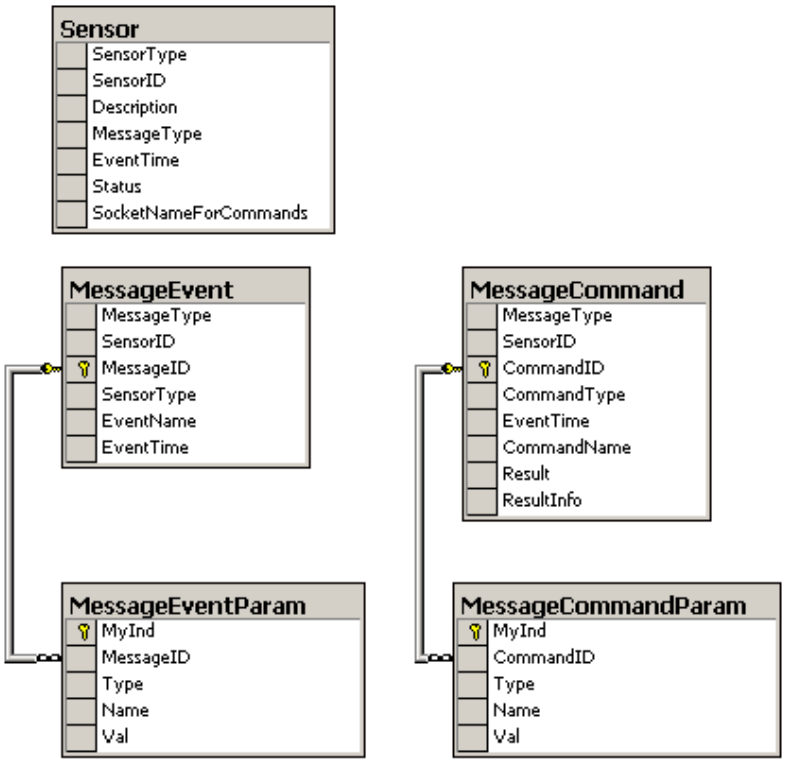

Схема зависимостей таблиц в базе данных

Функции, используемые для обращения к БД

Для удобной и быстрой работы с данными используются различные фильтры, содержащие в себе все возможные ограничения на каждые поля.

Обращение к БД возможно через следующие функции:

*int FInitDBSensors(const char *ConfigWD);*

Инициализирует БД. Получает на входе папку*ConfigWD*, где лежит конфигурационный файл dbconfig с именем БД, пользователем и паролем.

Возвращает 0 в случае удачи.

*int FDBGetSensors(int *N, CSensor **s);*

При N==NULL и s==NULL возвращает размер структуры*CSensor*.

При N!=NULL и *s==NULL выделяет память и заполняет сенсор.

При N==NULL и *s!=NULL освобождает память, отведенную ранее.

Возвращает 0 в случае удачного завершения операции.

*int FDBPutSensor(CSensor *s);*

Вставляет новый сенсор s в таблицу Sensor.

*int FDBDeleteSensor(const char *SensorID);*

Удаляет сенсор из таблицы Sensor с данным*SensorID*.

*int FDBSensorSetStatusMask(const char *SensorID, int Mask, int Operation);*

Устанавливает для сенсора *SensorID* поле *Status* путем логического И (Operation=1)или ИЛИ (Operation=0) с полем*Status* .

*int FDBSensorSetSocketNameForCommands(const char *SensorID, char *SocketNameForCommands);*

Устанавливает соответствующее имя сокета для сенсора с*SensorID*.

*int FDBGetSensor(const char *SensorID, CSensor *s);*

Возвращает сенсор с заданным *SensorID*

*int KeepAlive(int N, CMessageKeepAlive *e);*

Заменяет *EventTime* для заданного сенсора. В соответствии со значением *EventTime* модернизируется значение *Status* в таблице*Sensor*.

*int FInitDBEvents(const char *ConfigWD);*

Инициализирует БД. Получает на входе папку*ConfigWD*, где лежит конфигурационный файл dbconfig с именем БД, пользователем и паролем.

Возвращает 0 в случае удачи.

*int FDBGetEvents(int *N, CMessageEvent **event, CMessageEventRequest *request);*

Если N==NULL и e=NULL и r==NULL, то возвращает информацию о размере структур*sizeof(CMessageEvent)|(sizeof(CMessageEventRequest)«16).*

Если N!=NULL и *e=NULL, то выделяет память, заполняет все структуры. Если request=NULL, то фильтр не используется. Иначе возвращаются только те строки, которые удовлетворяют заданным условиям.

Если N==NULL и *e!=NULL, то освобождает память, отведенную под структуры.

int FDBDeleteEvent(int ID);

Удаляет событие с заданным ID.

*int FDBPutEvent(CMessageEvent *c);*

Размещает событие с в БД в таблице MessageEvent.

*int FInitDBCommands(const char *ConfigWD);*

Инициализирует БД. Получает на входе папку*ConfigWD*, где лежит конфигурационный файл *dbconfig*с именем БД, пользователем и паролем.

Возвращает 0 в случае удачи.

*int FDBGetCommands(int *N, CMessageCommand **Command, CMessageCommandRequest *request);*

Если N==NULL и Command=NULL и request==NULL, то возвращает информацию о размере структур *sizeof(CMessageCommand)|(sizeof(CMessageEventRequest)«16)*

Если N!=NULL и *Command=NULL, то выделяет память, заполняет все структуры. Если request=NULL, то фильтр не используется. Иначе возвращаются только те строки, которые удовлетворяют заданным условиям.

Если N==NULL и *Command!=NULL, освобождает память, отведенную под структуры.

*int FDBPutCommand(CMessageCommand *c);*

Если*CommandID==0*, вставляет с в таблицу *MessageCommand* и *MessageCommandParam* (автоматически назначает*CommandID*).

Иначе устанавливает параметры для команды с CommandID.

int FDBDeleteCommand(int ID);

Удаляет команду с заданным ID.