VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Dávid Dolobáč

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

AUTOMATIZACE PŘÍSTUPU K TESTOVACÍM SOUBORŮM PRO VIBRAČNÍ SYSTÉM
AUTOMATION OF ACCESS TO TEST FILES FOR THE VIBRATION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Dávid Dolobáč
AUTHOR

VEDOUCÍ PRÁCE — Ing. Miroslav Uher
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Dávid Dolobáč ***ID:*** 211141
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Automatizace přístupu k testovacím souborům pro vibrační systém

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je realizace automatizace obousměrného přístupu k testovacím souborům pro vibrační systém v ZL CVVOZE. Stávající ruční zadávání a získávání dat o zkoušce má být nahrazeno softwarovým řešením. Součástí této práce naopak nemá být řešení grafického výstupu. Předpokládá se přístup do SQlite databází profilů poptávek, snímačů a zkoušek prostřednictvím C++/C#.

1. Nastudujte a dokumentujte strukturu testovacích souborů pro řídící systém SWR1200 vibračního systému RMS SW8142 - SW H600APP.
2. S využitím již existující knihovny navrhněte softwarové řešení, které zajistí přenos potřebných dat o zkoušce z databázového systému zkušebny do testovacích souborů a naopak z testovacích souborů zpět do databázového systému.
3. Proveďte realizaci navrženého softwarového řešení pro samotný automatizovaný obousměrný interface mezi testovacími soubory a databázovým systémem v podobě softwarové knihovny a tuto následně vhodně dokumentujte.
4. Sestavte aplikaci, která umožní operátorovi zkušebny kontrolovanou manipulaci při vytváření testovacích souborů (se samozřejmým využitím metod sestavené knihovny). Zajistěte také ukládání skutečných parametrů testů zpět do databázového systému.
5. Sestavenou aplikaci taktéž vhodně dokumentujte, otestujte, proveďte analýzu potenciálních chybových stavů včetně způsobů jejich ošetření a sumarizujte limity aplikace.

DOPORUČENÁ LITERATURA:

1. Allen, G., Owens, M.: The Definitive Guide to SQLite. Apress, 2010. 348 s. ISBN-13: 978-1-4302-3225-4.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Miroslav Uher

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

Abstrakt

Cieľom tejto práce je popísať štruktúru testovacích súborov používaných pri vibračných skúškach v skúšobnom laboratóriu CVVOZE a na základe nej navrhnúť a realizovať softvérové riešenie, ktoré automatizuje činnosť ich vytvárania. Navrhované softvérové riešenie v podobe desktopovej aplikácie kompletne automatizuje pôvodné ručné zadávanie a získavanie dát o skúške s využitím vytvorených SQLite databáz a poskytne skúšobnému technikovi alternatívu k stávajúcemu softvéru firmy RMS.

Kľúčové slová

Databáza, Profily, Testovací súbor, Automatizácia prístupu, Vibračná skúška, SQLite, Knižnica, Aplikácia

Abstract

The aim of this thesis is to analyse structure of test files used in vibration tests in the CVVOZE testing laboratory and to design and implement a software solution that ensures automation of access to these files. The proposed software solution in the form of a desktop application completely automates the original manual entry and acquisition of test data using the created SQLite databases and provides the test technician with an alternative to the existing RMS software.

Keywords

Database, Profiles, Test file, Automation of access, Vibration test, SQLite, Library, Application

Bibliografická citace:

DOLOBÁČ, Dávid. *Automatizace přístupu k testovacím souborům pro vibrační systém*. Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/134861. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Miroslav Uher.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Dávid Dolobáč
VUT ID studenta:	211141
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Automatizace přístupu k testovacím souborům pro vibrační systém

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.
V Brně dne: 24. května 2021 ----------

podpis autora

Poďakovanie

Ďakujem vedúcemu bakalárskej práce Ing. Miroslavovi Uhrovi za účinnú metodickú, pedagogickú a odbornú pomoc a ďalšie cenné rady pri spracovaní mojej bakalárskej práce.

V Brne dňa: **24. mája 2021** ……

podpis autora

Obsah

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Bakalárska práca sa zaoberá návrhom a realizáciou softvérového rozhrania, ktoré automatizuje prácu s testovacími súbormi pri vibračných skúškach v laboratóriu CVVOZE. V práci sú zadokumentované 3 typy testovacích súborov, ktoré tvoria drvivú väčšinu vykonávaných vibračných skúšok. Potreba pre toto softvérové rozhranie je z toho dôvodu, že ručné zadávanie a získavanie dát môže viesť k chybám pri prepise hodnôt a zaberá čas.

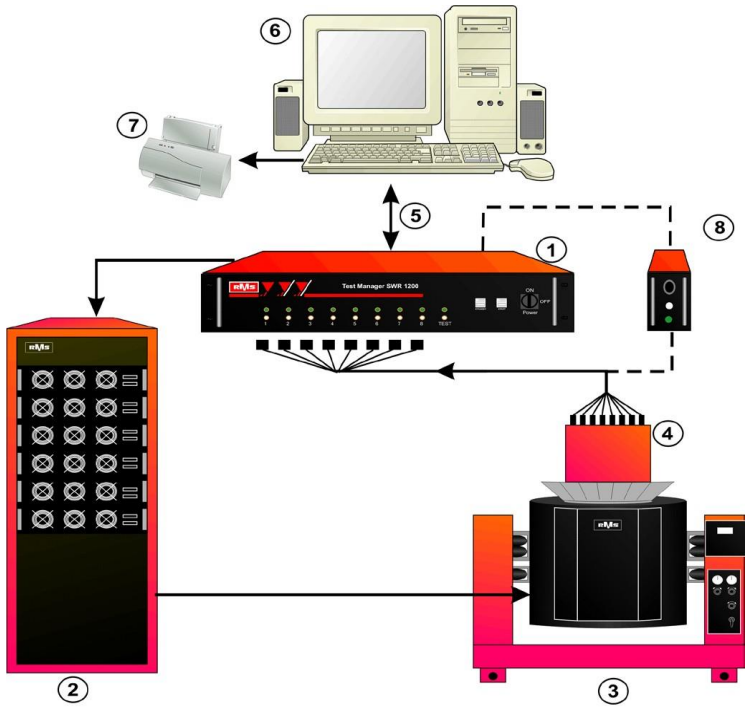
Navrhnuté riešenie, realizované v podobe desktopovej aplikácie, tieto činnosti dokáže kompletne automatizovať, pričom je stále zachovaná možnosť ručného zadávania dát. Aplikácia je vytvorená v jazyku C++ pomocou nástroja, resp. knižnice pre tvorbu užívateľských rozhraní Qt a pracuje s SQLite databázami, v ktorých sú uložené profily skúšok, nastavenia vykonaných testov a objednávková a zákazková evidencia laboratória CVVOZE. Ďalej aplikácia pracuje s testovacími súbormi, ktoré majú databázovú štruktúru a existujú vo formáte Microsoft Access 2.0 (*.mdb). Pre prácu s SQLite databázami, databázami testovacích súborov a pre obojsmerný prenos dát medzi nimi bola vytvorená knižnica, ktorá zabezpečuje zapisovanie, získavanie a aktualizáciu dát v oboch databázových systémoch a slúži ako programové rozhranie medzi týmito dvoma druhmi databáz.

Bakalárska práca sa skladá z teoretického rozboru, ktorý obsahuje dokumentovanú štruktúru testovacích súborov a praktickej časti, ktorá obsahuje popis vytvoreného SQLite databázového systému, návrh a dokumentáciu vytvorenej softvérovej knižnice a dokumentáciu vytvorenej aplikácie.

1. DOKUMENTÁCIA TESTOVACÍCH SÚBOROV

1.1 Skúšobné laboratórium CVVOZE

Skúšobné laboratórium CVVOZE (Centrum výzkumu a využití obnovitelných zdrojů energie) na Ústavu automatizace a měřicí techniky slúži ako pracovisko pre skúšky klimatickej odolnosti a vibrácií a je vybavené klimatickou komorou a dvojosím vibračným budičom chladeným vodou so zosilňovačom a riadiacim systémom firmy RMS. Skúšobný systém umožňuje vykonávať kombinované skúšky vplyvu teploty, vlhkosti a vibrácií na elektronické a mechanické komponenty. Skúšky vibračnej odolnosti sa realizujú vibráciami sínusovými, náhodnými, podľa časového priebehu a vibráciami zmiešanými. Dvojosí vibračný budič so zosilňovačom pre generovanie definovaných mechanických vibrácií pozostáva z elektrodynamického vibračného budiča s vodným chladením a klzného stolu na hydrostatických ložiskách a výkonového zosilňovača. Súčasťou systému je aj programovateľná riadiaca jednotka a riadiaci softvér SWR 1200 s preddefinovanými módmi a testovacími priebehmi mechanických kmitov. [1]



Obrázok 1.1 Bloková schéma meracieho pracoviska [2]

Softvér SWR 1200 dokáže so všetkými dostupnými programovými modulmi produkovať rôzne mechanické vibrácie na akomkoľvek type vibračného zariadenia. Pracuje v grafických operačných systémov Windows 95 / 98 / NT 4.0 / 2000 a XP a spolu s hardvérovou riadiacou jednotkou, ktorá sa nazýva Test Manager, tvoria riadiace centrum, odkiaľ sú monitorované všetky prebiehajúce testy. [2]

Riadiaca jednotka obsahuje štyri vstupné kanály (s možnosťou rozšírenia až na osem), ktoré môžu byť použité ako meracie alebo riadiace vstupy. Súčasne môže prebiehať testovanie až na ôsmich rôznych systémoch, ale k tomu potrebujeme 8 rôznych riadiacich jednotiek. [2]

Konfigurácia pracovného režimu vstupných kanálov prebieha v užívateľskom prostredí softvéru. Prebiehajúci test môže byť kedykoľvek pozastavený (ak chce užívateľ napríklad zmeniť parametre snímačov) a znovu spustený s novými nastaveniami, pričom dáta namerané pri predošlých nastaveniach neostanú uložené, ale prepíšu sa novými.[2]

Pred začatím vibračného testu musí softvér vytvoriť testovací súbor, ktorý bude obsahovať nastavenia vstupných kanálov, nastavenia testovacieho signálu a ďalšie parametre merania, ktoré vyplnil užívateľ. Po vytvorení testovacieho súboru, ktorý má podobu databázy, sa súbor nahrá do ovládača (Test Manager) a odštartuje sa meranie. Samotný ovládač nemusí byť prítomný ako hardvérová jednotka – je možné použiť aj virtuálny ovládač, no pri tejto možnosti sa meranie neuskutoční. Virtuálne ovládače sa preto môžu použiť napríklad na overenie správnosti vytvoreného databázového (testovacieho) súboru pri vzdialenom prístupe. [2]

Parametre testovacieho signálu, namerané dáta a ďalšie nastavenia vibračnej skúšky sú zaznamenávané a ukladané do rovnakého testovacieho súboru, odkiaľ môžu byť tieto dáta spätne spracované. [2]

1.2 Popis a štruktúra testovacích súborov

Softvér RWS 1200 pracuje s dvomi typmi súborov, a to sú testovacie súbory a testovacie šablóny. Tieto súbory majú databázovú štruktúru, ktorá slúži na ukladanie nastavení skúšok a nameraných dát. Súbory sú vytvorené na základe Microsoft Data Base format (*.mdb) a je možné ich otvárať, upravovať alebo konvertovať do iného formátu napríklad v prostredí Microsoft Access. [2]

Testovacie šablóny na rozdiel od testovacích súborov neobsahujú namerané dáta, obsahujú iba parametre testovacieho signálu ako frekvencia, hodnoty zrýchlenia atď. Účel testovacích šablón spočíva v tom, že si z nich môžeme relatívne rýchlo vytvoriť sadu podobných testovacích súborov, ktoré sa líšia iba v malých detailoch, napríklad v hodnotách frekvencie, zrýchlenia alebo amplitúdy testovacej krivky. [2] Samotný softvér SWR 1200 teda ponúka určitú úroveň automatizácie pri práci s testovacími súbormi, ale testovacie šablóny sú obmedzené iba na nastavenia parametrov testovacieho signálu, ďalšie nastavenia musí užívateľ vypĺňať ručne, čo nie je dostačujúce.

Navrhované softvérové riešenie by sa malo vzťahovať na všetky nastavenia merania a malo by byť oveľa komplexnejšie.

Testovacie súbory obsahujú všetky potrebné parametre merania, ktoré nastavuje užívateľ v príslušných oknách v prostredí softvéru a taktiež namerané dáta z posledného vykonaného testu. Tieto súbory majú možnosť cyklicky ukladať namerané dáta v časovom intervale, ktorý zvolí užívateľ pred testom. Vďaka týmto dátam je možné spätne rekonštruovať priebeh dlhej vibračnej skúšky. Po reštartovaní testu sú predošlé namerané dáta prepísané novými. [2]

Testovacie súbory sa delia podľa jednotlivých módov vibrácií nasledovne [2]:

- **Random** – náhodné vibrácie (*.rau)
- **Sine** – sínusové vibrácie (*.sin)
- **Shock –** pulzy (*.shk)
- **Resonancy –** vyhľadávanie rezonancií (*.rso)
- **Sine on Random** – (*.sar) [2]

1.2.1 Random

Databázový súbor pre náhodné vibrácie má príponu *.rau a tvoria ho tabuľky:

Data_1 až Data_n [2]
– tieto tabuľky obsahujú namerané dáta
– n je číslo tabuľky, maximum tabuliek je 99
– prvé 3 riadky obsahujú ďalšie informácie o jednotlivých kanáloch
– od 4. riadku začínajú namerané dáta, ich maximálny počet je 1024 riadkov na jednu tabuľku

Tabuľka 1.1 Ukážková tabuľka *Data_n* súboru *.sin [2]

Drive	**Control**	**X_Data**	**Channel_1**	**...**	**Channel_8**
26.19963	3.35	202	28	...	-20
		57	736839.9	...	0
			0	...	0
4.7E-13	4.7E-15	0	4.72E-15	...	0
4.76E-13	6.5E-05	1.953	0.000118	...	0

Drive – hodnota Drive Spectrum $[(m/s^2)^2/Hz]$
Control – hodnota Control Spectrum $[mV^2/Hz]$
X_Data – frekvencia [Hz]
Channel_1 až 8 – hodnoty amplitúdy signálu $[(m/s^2)^2/Hz]$

Logbook [2]

– obsahuje všetky udalosti, ktoré sa počas testu udiali a udáva ich časové značky (dôležité budú časy začiatku a konca skúšky)

Tabuľka 1.2 Ukážková tabuľka *Logbook* súboru *.sin [2]

Date	**Info**	**Level**	**Index**
--	Start 28.08.20 09:53:00	0	1
00:00:00:00	System Test	0	2
00:00:00:03	Test active	-20	3
00:00:00:11	Test active Warn Limit exceeded	-20	4
00:00:00:20	Test active	-20	5
00:00:01:33	Test active	-9	6
00:00:01:43	Test active	-6	7
00:00:01:54	Test active	-3	8
00:00:02:04	Test active	0	9
00:00:07:04	Ramp Down	0	10
00:00:07:04	Stop Test time elapsed	0	11

Date – uplynulý čas od začiatku testu
Info – popis udalosti

Target Curve [2]

– v tejto tabuľke sa nachádzajú parametre testovacej krivky a nastavenia rozsahu merania, ktoré nastavuje užívateľ pred testom na príslušnej karte
– táto tabuľka bude dôležitá pri softwarovom riešení profilov meraní

Tabuľka 1.3 Ukážková tabuľka *Target Curve* súboru *.rau [2]

WarnLower_Y	**WarnUpper_Y**	**AbortLower_Y**	**AbortUpper_Y**
1.013	4.034	0.5079	8.050
25.059	99.763	12.559	199.05
25.059	99.763	12.559	199.053
10.5	40.03	5.039	79.871

Y_Data	**X_Data**	**Index**
2.022	20	1
50	100	2
50	800	3
20.6	2000	4

Y_Data – hodnota zrýchlenia [m/s^2]
WarnLower_Y, WarnUpper_Y, AbortLower_Y, AbortUpper_Y

– varovné limity zrýchlenia (absolútne hodnoty) [m/s^2]

X_Data – hodnota frekvencie [Hz]

Drive [3]

– táto tabuľka slúži pre optimalizované ukladanie počiatočných údajov a neobsahuje žiadne nastavenia ani namerané dáta

System Data Text [3]

– táto tabuľka obsahuje popisné údaje daného testu a popis k meracím kanálom

Tabuľka 1.4 Ukážková tabuľka *System Data Text* súboru *.rau [2]

Index	**SystemDataText**
1	1.9.2020 10:45
2	random test file
3	Acceleration Spectral Density; View 1
4	Acceleration Spectral Density; View 2
5	Acceleration Spectral Density; View 3
6	Drive Spectral Density; View 1
7	Drive Spectral Density; View 2
8	Drive Spectral Density; View 3
9	
10	Channel 7 - 356A03/LW252876 Z axis -
11	Channel 6 - 356A03/LW252876 Y axis -
12	Channel 5 - 356A03/LW195146 Y axis -
13	Channel 4 - 356A03/LW195146 Z axis -
14	Channel 3 - J353C04/200968 - slip-table
15	Channel 2 - 356A03/LW252876 X axis -
16	Channel 1 - 356A03/LW195146 X axis -

– na prvom riadku je dátum a čas vytvorenia testovacieho súboru
– druhý riadok obsahuje typ testovacieho súboru
– tretí až ôsmy riadok zahŕňa názvy pohľadov jednotlivých grafov
– deviaty riadok je prázdny
– na desiatom až šesťnástom riadku sú názvy jednotlivých meracích kanálov

System Data Float [3]

– v tejto tabuľke sa nachádzajú všetky nastavenia snímačov a skúšok, ktoré majú vplyv na priebeh merania [3]

– tieto údaje vypĺňa užívateľ na jednotlivých kartách sprievodcu softwaru RMS pri vytváraní testovacieho súboru, tým pádom bude dôležitá pri exporte dát do testovacích súborov

Tabuľka 1.5 Ukážková tabuľka *System Data Float* súboru *.rau [2]

SystemDataFloat	**Index**
209	1
15	2
0	3
4	4
0	5
...	...
1000	149

Graph Style [3]

– táto tabuľka nesie informácie o zobrazení kriviek jednotlivých grafov

Tabuľka 1.6 Ukážková tabuľka *Graph Style* súboru *.rau [2]

Farbe	**Dicke**	**D1**	**D2**	**D3**	**LegendeText**	**LegendeFarbe**
98	1	FALSE	FALSE	FALSE		1
38	1	TRUE	TRUE	TRUE	Lower Warn Limit	1
38	1	TRUE	TRUE	TRUE	Upper Warn Limit	1
...	...	...	...	...	...	...
72	1	FALSE	TRUE	FALSE	Control Output / Control Drive	0

LegendeDicke	**Index**
0	1
0	2
0	3
0	...
0	69

Farbe – farba krivky

Dicke – hrúbka krivky

D1, D2, D3 – tieto stĺpce určujú, v ktorom pohľade grafu bude krivka viditeľná (TRUE) alebo neviditeľná (FALSE)

LegendeText – obsahuje názov krivky

LegendeFarbe – obsahuje farbu názvu krivky

LegendeDicke – obsahuje hrúbku písma pri názve krivky

Diagramm Style Text [3]

– obsahuje názvy grafov a osí jednotlivých grafov, každému stĺpcu odpovedá 1 graf

Tabuľka 1.7 Ukážková tabuľka *Diagramm Style Text* súboru *.rau [2]

DiagrammStyleText1	**DiagrammStyleText2**	**Index**
Random Acceleration	Random User Defined	1
Frequency [Hz]	Frequency [Hz]	2

Diagramm Style Float [3]

– obsahuje nastavenia jednotlivých grafov, čísla sú uložené ako dátový typ float

– každému stĺpcu odpovedá jeden graf

Tabuľka 1.8 Ukážková tabuľka *Diagramm Style Float* súboru *.rau [2]

DiagrammStyleFloat2	**DiagrammStyleFloat1**	**Index**
10000	3000	1
10	10	2
...	...	...
-1.289382	-1.318886	178

1.2.2 Sine

Databázový súbor pre sínusové vibrácie má príponu *.sin a tvoria ho tabuľky:

Data_1 až Data_n [2]

– tieto tabuľky obsahujú namerané dáta

– n je číslo tabuľky, maximum tabuliek je 99

– prvé 3 riadky obsahujú informácie o jednotlivých kanáloch

– od indexu číslo 4 začínajú namerané dáta, ich maximálny počet je 2000 hodnôt na jednu tabuľku, pri prekročení sa dáta zapisujú do novej tabuľky

Tabuľka 1.9 Ukážková tabuľka *Data_n* súboru *.sin [2]

Drive	**Control**	**X_Data**	**Channel_1**	**...**	**Channel_8**	**Index**
1931.589	2.32E-08	5.311016	1	...	3	1
0		263	5.311017	...	0	2
			0	...	0	3
166.1321	5.022052	5	5.022052	...	0	4
...	...	...	...	...	...	...
9.9E-39	9.9.1939	7.380813	9.9E-39	...	0	2003

Drive – hodnota výstupu z kontroléru [mV]
Control – hodnota riadiaceho signálu [m/s^2]
X_Data – hodnota frekvencie [Hz]
Channel_1 až 8 – výstupné hodnoty kanálov, ktoré merajú zrýchlenie [m/s^2]

Target Curve [2]

– v tejto tabuľke sa nachádzajú parametre testovacej krivky a nastavenia rozsahu merania, ktoré nastavuje užívateľ pred testom
– táto tabuľka bude dôležitá pri softwarovom riešení profilov meraní

Tabuľka 1.10 Ukážková tabuľka *Target Curve* súboru *.sin [2]

WarnLower_Y	**WarnUpper_Y**	**AbortLower_Y**	**AbortUpper_Y**	**Y_Data**
3.539729	7.062688	2.505936	9.976312	5
3.539729	7.062688	2.505936	9.976312	5
0	0	0	0	0
...	...	...	...	...
0	0	0	0	0

X_Data	**Index**
5	1
2000	2
0	3
...	...
0	100

Y_Data – hodnota zrýchlenia [m/s^2]
WarnLower_Y, WarnUpper_Y, AbortLower_Y, AbortUpper_Y
– varovné limity zrýchlenia (absolútne hodnoty) [m/s^2]
X_Data – hodnota frekvencie [Hz]

System Data Float

– v tejto tabuľke sa nachádzajú všetky nastavenia snímačov a skúšok, ktoré majú vplyv na priebeh merania, dáta sú uložené ako čísla typu float [2]
– tieto údaje vypĺňa užívateľ na jednotlivých kartách sprievodcu softvéru RMS pri vytváraní testovacieho súboru, tým pádom bude táto tabuľka dôležitá pri exporte dát z databáz do testovacích súborov

Tabuľka 1.11 Ukážková tabuľka *System Data Float* súboru *.sin [2]

SystemDataFloat	**Index**
110	1
1	2
0	3
...	...
0	253

System Data Text [2]

– táto tabuľka obsahuje popisné údaje daného testu a popis meracích kanálov

Tabuľka 1.12 Ukážková tabuľka *System Data Text* súboru *.sin [2]

Index	**SystemDataText**
1	1.9.2020 10:05
2	sine test file
3	sine displacement; view 1
4	sine displacement; view 2
...	...
15	Channel 8 - 356A03/LW195146 Z axis -
16	Channel 7 - 356A03/LW252876 Z axis -
17	Channel 6 - 356A03/LW252876 Y axis -
18	Channel 5 - 356A03/LW252876 X axis -
19	Channel 4 - 356A03/LW195146 Z axis -
20	Channel 3 - 356A03/LW195146 Y axis -
21	Channel 2 - 356A03/LW195146 X axis -
22	Channel 1 - J353C04/200968 -

– na prvom riadku je dátum a čas vytvorenia testovacieho súboru
– druhý riadok obsahuje typ testovacieho súboru
– 3. a 4. riadok zahŕňa názvy pohľadov jednotlivých grafov
– na 15. až 16. riadku sa nachádzajú názvy a informácie jednotlivých meracích kanálov

Logbook [2]

– táto tabuľka zaznamenáva všetky udalosti, ktoré sa počas testu udiali, a udáva ich čas v závislosti od začiatku testu a je rovnaká ako Tabuľka 1.2 súboru Random

Drive [3]

– do tejto tabuľky sa neukladajú žiadne nastavenia ani namerané hodnoty a je rovnaká ako v súbore Random

Graph Style [3]

– táto tabuľka nesie informácie o zobrazení kriviek jednotlivých grafov, je rovnaká ako Tabuľka 1.6 súboru Random

Diagramm Style Float [3]

– obsahuje nastavenia jednotlivých grafov, hodnoty sú uložené ako dátový typ float
– každému stĺpcu odpovedá jeden graf
– je rovnaká ako Tabuľka 1.8 súboru Random, len tu sú 4 stĺpce namiesto dvoch

Diagramm Style Text [3]

– obsahuje názvy grafov a osí jednotlivých grafov, každému stĺpcu odpovedá 1 graf
– je rovnaká ako Tabuľka 1.7 súboru Random, len tu sú 4 stĺpce namiesto dvoch

1.2.3 Shock

Databázový súbor s príponou *.shock slúžia na generovanie pulzov a tvoria ich tabuľky:

Data_1 až Data_n [2]

– tieto tabuľky obsahujú namerané dáta
– n je číslo tabuľky, maximum tabuliek je 99
– prvé 3 riadky obsahujú informácie o jednotlivých kanáloch
– od indexu číslo 4 začínajú namerané dáta, ich maximálny počet je 4096 hodnôt (riadkov) na jednu tabuľku, pri prekročení sa dáta zapisujú do novej tabuľky

Tabuľka 1.13 Ukážková tabuľka *Data_n* súboru *.shk [2]

Control	**X_Data**	**Channel_1**	**...**	**Channel_8**	**Index**
6.404839	151.8332	10	...	1	1
0.52514	3	151.1237	...	0	2
6.404839	0.52574	0	...	0	3
...	...	...	...	...	...
8.02831	159.705	0.4890346	...	0	4099

Control – hodnota riadiaceho signálu [mV]
X_Data – čas [ms]
Channel_1 až 8 – výstupné hodnoty kanálov, ktoré merajú zrýchlenie [m/s^2]

Target Curve [3]

– v súboroch *.shk sa do tejto tabuľky nezapisujú žiadne dáta, ktoré nastavuje užívateľ na jednotlivých kartách sprievodcu, je to prázdna tabuľka s rovnakou štruktúrou ako v predošlých súboroch

Tabuľka 1.14 Ukážková tabuľka *Target Curve* súboru *.shk [2]

WarnLower_Y	**WarnUpper_Y**	**AbortLower_Y**	**AbortUpper_Y**
-4.316021E+08	-4.316021E+08	-4.316021E+08	-4.316021E+08

Y_Data	**X_Data**	**Index**
		57
		...
		100
-4.316E+8	-4.316E+8	1
		...
		56

User Curve Type [3]

– v tejto tabuľke sa nachádzajú parametre krivky definované užívateľom

Tabuľka 1.15 Ukážková tabuľka *User Curve Type* súboru *.shk [2]

UserTime	**UserAmplitude**	**Index**
0	0	529
1	10	530
...	...	...
25	0	540

UserTime – čas [ms]

UserAmplitude – amplitúda signálu [m/s^2]

User Tolerance Band [2]

– táto tabuľka obsahuje hodnoty tolerančného pásma, ktoré zadáva užívateľ

Tabuľka 1.16 Ukážková tabuľka *User Tolerance Band* súboru *.shk [2]

CurveNo	**CurveType**	**StandardNo**	**Standard**	**Time**	**AbortPos**	**AbortNeg**	**Index**
5	frei definiert	3	free	-1000	15	-15	265
5	frei definiert	3	free	-10	115	-15	266
5	…	…	…	…	…	…	…
5	frei definiert	3	free	1000	15	-15	270

User Pre/Post Pulses [2]

– v tejto tabuľke sa nachádzajú ďalšie parametre testovacieho signálu, ktoré vypĺňa užívateľ

Tabuľka 1.17 Ukážková tabuľka *User Pre/Post Pulses* súboru *.shk [2]

CurveNo	**CurveType**	**StandardNo**	**Standard**	**PrePositive**	**PreNegative**	**PostPositive**
0	Halbsinus	0	MIL-STD	2	2	15
0	Halbsinus	1	DIN-IEC	15	15	15
...	...	...	...	...	...	...
5	frei definiert	3	frei	23.80	21	22

CurveNo – číslo krivky
CurveType – typ signálu (píla, lichobežník, trojuholník…)
StandardNo – číslo štandardu
Standard – slovné označenie štandardu

System Data Float

– v tejto tabuľke sa nachádzajú všetky nastavenia snímačov a skúšok, ktoré majú vplyv na priebeh merania, dáta sú uložené ako čísla typu float [3]
– tieto údaje vypĺňa užívateľ na jednotlivých kartách sprievodcu softwaru RMS pri vytváraní testovacieho súboru, tým pádom bude časť tejto tabuľky dôležitá v softvérovom riešení

Tabuľka 1.18 Ukážková tabuľka *System Data Float* súboru *.shk [2]

SystemDataFloat	**Index**
310	1
1	2
…	…
0	1140

System Data Text [3]

– táto tabuľka obsahuje popisné údaje daného testu a názvy pohľadov grafov

– na prvom riadku je dátum a čas vytvorenia testovacieho súboru
– druhý riadok obsahuje typ testovacieho súboru
– 3. až 14. riadok zahŕňajú názvy pohľadov jednotlivých grafov

Tabuľka 1.19 Ukážková tabuľka *System Data Text* súboru *.shk [2]

Index	**SystemDataText**
1	10.6.2020 0:47
2	shock test file
3	shock acceleration; view 1
...	...
14	shock displacement; view 3

Logbook [3]
– táto tabuľka zaznamenáva všetky udalosti, ktoré sa počas testu udiali, udáva ich čas v závislosti od začiatku testu a je rovnaká ako Tabuľka 1.2 súboru Random, s tým rozdielom, že pribudol stĺpec ShockNo, ktorý obsahuje počet pulzov

Drive [3]
– do tejto tabuľky sa neukladajú žiadne nastavenia ani namerané hodnoty a je rovnaká ako v súbore Random

Graph Style [3]
– táto tabuľka nesie informácie o zobrazení kriviek jednotlivých grafov a je rovnaká ako Tabuľka 1.6 súboru Random

Diagramm Style Float [3]
– obsahuje nastavenia jednotlivých grafov, hodnoty sú uložené ako dátový typ float
– každému stĺpcu odpovedá jeden graf a je rovnaká ako Tabuľka 1.8 súboru Random, len tu sú 4 stĺpce namiesto dvoch

Diagramm Style Text [3]
– obsahuje názvy grafov a osí jednotlivých grafov, každému stĺpcu odpovedá 1 graf
– je rovnaká ako Tabuľka 1.7 súboru Random, len tu sú 4 stĺpce namiesto dvoch

2. NÁVRH SW RIEŠENIA

2.1 Popis problému a jeho riešenie

Každá vibračná skúška sa začína tým, že obsluhujúci pracovník musí v programe SWR 1200 zadávať ručne všetky nastavenia. Hlavným problémom je, že tu môžu vzniknúť chyby pri prepise hodnôt a potom je takto vykonané meranie neplatné a musí sa zopakovať znovu. Ďalší problém je ten, že ručné zadávanie zaberá veľa času a pri opakujúcich sa podobných testoch je to navyše repetitívna mechanická práca, ktorá sa dá automatizovať. Získanie dát o vykonanej skúške prebieha tiež manuálne, navrhované softvérové riešenie automatizuje aj túto činnosť.

Riešenie spočíva vo vytvorení desktopovej aplikácie vo forme sprievodcu s užívateľským rozhraním, kde užívateľ prechádza voľne jednotlivými krokmi konfigurácie vibračného testu. Na rozdiel od programu SWR 1200 nebude musieť zadávať údaje zakaždým ručne, k tomu budú slúžiť profily a nastavenia skúšok uložené v SQLite databázach. Užívateľ má možnosť si z databázy vybrať konkrétny profil merania, po výbere sa tieto hodnoty zapíšu do tabuľky v užívateľskom rozhraní. Rovnako ďalšie nastavenia testu, ktoré nie sú súčasťou profilov, je možné načítať z už existujúceho súboru (jeho nastavenia sú zaznamenané v SQLite databázach) a tieto dáta sa automaticky zapíšu do užívateľského rozhrania. Aplikácia je navrhnutá tak, že v každom kroku konfigurácie je možnosť ručného prepísania alebo zadania nových nastavení. Po vyplnení všetkých potrebných kariet sa dáta nastavené v aplikácii exportujú do novo vytvoreného testovacieho súboru a do SQLite databázy, čím sa budú všetky nastavenia predošlých testov zaznamenávať pre ďalšie použitie. Užívateľ má taktiež možnosť spätne načítať existujúci testovací súbor, získať z neho časové značky spustenia a ukončenia testu a v prípade zmeny nastavení sa získať zo súboru aktuálne dáta. Pôvodný záznam v SQLite databáze sa následne prepíše. Všetky potrebné presuny dát medzi SQLite databázami a testovacími súbormi má na starosti vytvorená C++ knižnica, ktorá je staticky pripojená k aplikácii.

Podrobnejšie fungovanie použitých databáz, samotnej aplikácie a jej knižnice bude ozrejmené v ďalších kapitolách.

2.2 SQLite

SQLite je vstavaná relačná databáza, ktorá sa radí do kategórie softvéru s otvoreným zdrojovým kódom. Bola navrhnutá tak, aby poskytla aplikáciám pohodlný spôsob správy dát bez veľkých réžií, ktoré sú časté pri iných relačných databázových systémoch ako napríklad Oracle, MySQL alebo Microsoft SQL Server. SQLite je vstavaná databáza, čo znamená, že nebeží nezávisle ako samostatný proces, ale koexistuje vo vnútri aplikácie – v rámci jej procesového priestoru. Jej kód je vložený ako súčasť programu, ktorý SQLite

databázu používa. Pre vonkajšieho pozorovateľa nie je na pohľad zrejmé, že takýto program vôbec využíva nejaký relačný databázový systém, avšak vo vnútri programu pracuje kompletný, sebestačný databázový engine. Jednou z výhod databázového serveru umiestneného v rámci programu je, že neexistuje žiadna požiadavka na sieťovú konfiguráciu alebo administráciu. Klient i server bežia spoločne v rovnakom procese, a to znižuje réžie súvisiace s posielaním a prijímaním požiadaviek po sieti, zjednodušuje správu databázy a uľahčuje nasadzovanie aplikácií. SQLite databázový engine je implementovaný ako knižnica jazyku C a všetky programy písané v programovacích jazykoch ako napríklad C/C++, C#, LabVIEW, MATLAB, Python a ďalšie, ju môžu využívať pomocou jej SQLite C API rozhrania. [4][7]

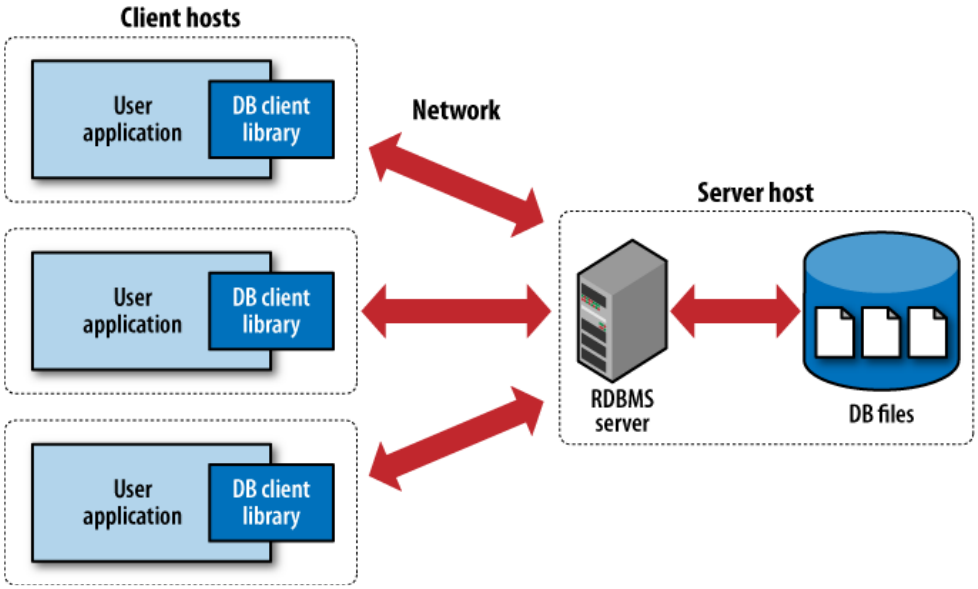
2.3 Prednosti SQLite

2.3.1 Samostatnosť

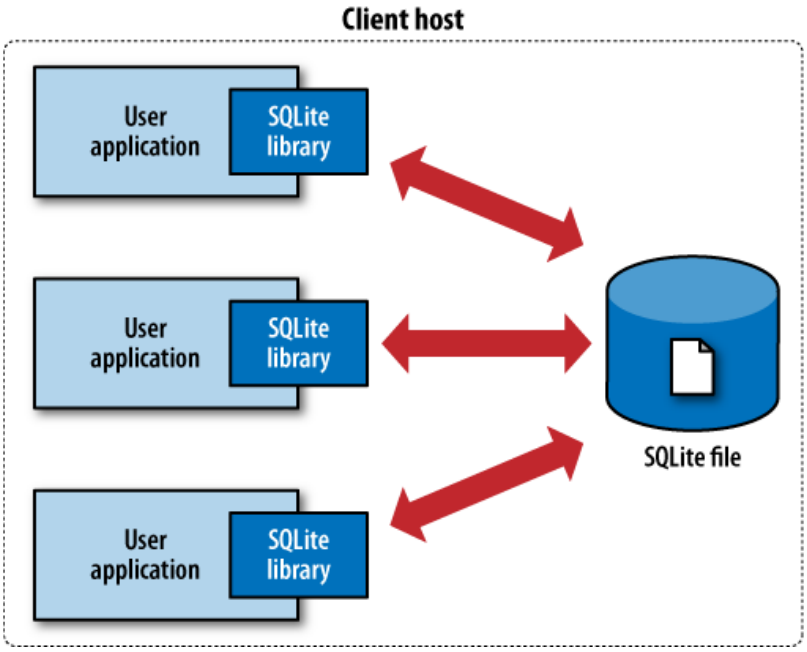
Na rozdiel od väčšiny relačných databázových systémov, SQLite nemá klient/server architektúru. Väčšina rozsiahlych databázových systémov potrebuje k svojej činnosti veľký serverový balík, ktorý tvorí databázový engine. Databázový server často pozostáva z viacerých procesov, ktoré pracujú spoločne, aby zaistili pripojenie klienta k databáze, I/O súborov, optimalizáciu dotazov a ich spracovanie. Jedna inštancia databázy zvyčajne pozostáva z veľkého počtu súborov usporiadaných do jedného alebo viacerých adresárových stromov v súborovom systéme servera a za účelom prístupu do databázy musia byť všetky súbory prítomné a korektné. To môže trochu sťažiť presúvanie alebo zálohovanie danej databázy. [5]

Všetky tieto komponenty vyžadujú zdroje a podporu hostiteľského počítača. Najlepším postupom v tomto prípade je, aby bol počítač nakonfigurovaný s vyhradeným používateľským účtom, spúšťacími skriptami a dedikovaným úložiskom, čo robí z databázového serveru veľmi užívateľsky nepríjemný softvér. Z tohto dôvodu a z hľadiska výkonu je obvykle vyhradený celý jeden počítač výlučne pre potreby databázového servera. Pre prístup k databázam sú potrebné klientske softvérové knižnice, ktoré sú typicky poskytnuté predajcom a musia byť integrované v každej klientskej aplikácii, ktorá sa pokúša k databázovému serveru pripojiť. Obrázok 2.1 ukazuje ako tieto knižnice fungujú v prípade relačného databázového systému typu klient/server. [5]

Oproti tomu SQLite nemá žiaden separátny server, celý databázový engine je integrovaný priamo v kóde aplikácie. Jediným zdieľaným zdrojom medzi aplikáciami je jeden databázový súbor uložený na disku, ktorý predstavuje celú databázovú inštanciu, čím sa presun alebo záloha databázy značne zjednodušuje. Absenciou servera sa odstránia ďalšie zložitosti, nie je potreba pokročilého multitaskingu a nie je požiadavka na medziprocesovú komunikáciu. SQLite vyžaduje iba schopnosť čítať a zapisovať na nejaký typ úložiska. Obrázok 2.2 zobrazuje SQLite infraštruktúru. [5]



Obrázok 2.1: Tradičná RDBMS klient/server architektúra [5]



Obrázok 2.2: SQLite architektúra [5]

2.3.2 Kompaktnosť

SQLite dokáže obsiahnuť celú databázu v jedinom relatívne malom súbore na disku, čo je jej najväčšia výhoda. Tento súbor obsahuje rozloženie databázy, aktuálne dáta a všetky objekty – tabuľky, spúšťače, indexy a náhľady. Databázové súbory nezaberajú veľa miesta, pretože sa využíva záznamov s premennou dĺžkou. Alokuje sa iba také množstvo dát, aké je potrebné na uloženie každého poľa. Súborový formát je multiplatformový, prístupný na akomkoľvek počítači bez ohľadu na endianitu alebo dĺžku slova. Práca s takýmito databázami je oveľa prívetivejšia a nemôže nastať stav, aby bola databáza

poškodená alebo nedostupná, pretože jeden z tucta súborov bol presunutý alebo nechcene premenovaný. [4][5]

2.3.3 Nulová konfigurácia

Z pohľadu koncového užívateľa, SQLite nevyžaduje žiadnu inštaláciu alebo konfiguráciu počítača, čím je nasadzovanie jednoduchšie. SQLite databázy sú dnes používané vo veľkom počte aplikácií a užívateľ často ani nevie, že jeho aplikácia SQLite využíva. Väčšinou sú aplikácie navrhnuté tak, že celá užívateľova interakcia s databázou je zredukovaná iba na výber súboru. [5]

2.3.4 Licencia

Všetok SQLite kód je voľným dielom, neexistuje naň žiadna licencia. Nie je krytý *GNU General Public License* (GPL) alebo ktoroukoľvek podobnou licenciou softvéru s otvoreným zdrojovým kódom. Každý si môže so zdrojovým kódom robiť čo chce, bez nároku na jeho vlastníctvo. Kód a skompilované knižnice môžu byť akýmkoľvek spôsobom použité, upravené, redistribuované alebo predávané. Vďaka tomu je implementácia SQLite knižníc do aplikácie odbremenená od licenčných poplatkov a obmedzení a vývoj je preto celkovo jednoduchší. [5]

2.3.5 Unikátne vlastnosti

SQLite ponúka viaceré funkcie, ktoré poskytujú veľkú mieru „pohodlia“ a u väčšiny databázových systémov ich nenájdeme. [5]

Najvýznačnejším rozdielom je, že SQLite využíva dynamické typovanie. To znamená, že je dovolené vložiť akúkoľvek hodnotu do ľubovoľného stĺpca nezávisle na dátovom type, na rozdiel od tradičných databázových systémov, ktoré používajú statické typovanie. Nemusí sa preto meniť schéma tabuľky alebo dátový typ stĺpca, zmení sa iba spôsob ukladania dát. Vo veľa ohľadoch sa dynamický typový systém SQLite podobá tomu, ktorý by sme našli v populárnych skriptovacích jazykoch. Po čase sa ukázalo, že dynamické typovanie vyriešilo viac problémov ako spôsobilo a je pre SQLite prínosom.[4][5]

Ďalšou užitočnou vlastnosťou je schopnosť manipulovať s viacerými databázami zároveň. SQLite umožňuje k jednému databázovému pripojeniu pridružiť viaceré databázové súbory súčasne pomocou príkazu `ATTACH`. To umožňuje spracovávať dotazy, ktoré navzájom premosťujú viaceré databázy a vďaka tomu je jednoduchšie spájať tabuľky pomocou príkazu `JOIN` z rôznych databáz v jednom dotaze alebo hromadne kopírovať údaje taktiež pomocou jedného príkazu.[5]

Riešenie konfliktov je ďalšou funkciou SQLite, ktorá je zabudovaná do mnohých SQL operácií. Povedzme, že máme záznam, ktorý chceme vložiť príkazom `INSERT` a nevieme, či sa už v databáze nenachádza rovnaký záznam. Namiesto písania `SELECT` príkazu, ktorý bude hľadať zhodu a v prípade ak ju nájde pretvorí príkaz `INSERT` na

UPDATE, funkcia riešenia konfliktov dokáže nájsť sama zhodný záznam a aktualizovať ho namiesto pridania nového. Namiesto troch príkazov (INSERT, SELECT a prípadne UPDATE) nám stačí jeden: INSERT ON CONFLICT REPLACE (...). Ďalej je možnosť túto funkciu zabudovať priamo do definície tabuľky a tu už stačí len volať príkaz INSERT bez ďalšej špecifikácie. [4]

2.4 Limity SQLite

SQLite sa vyznačuje tým, že dokáže pracovať rýchlejšie s jednoduchými príkazmi ako INSERT, SELECT alebo UPDATE v porovnaní s ostatnými databázovými systémami, práve kvôli absencii servera. Rýchlosť je v jej jednoduchosti. No pri zložitejších a dlhších dotazoch už SQLite rýchlostne zaostáva a vtedy je vhodné siahnuť po inej databáze. Navyše ak pracujeme s veľkým množstvom dát, SQLite nie je dobrá voľba, keďže bola navrhnutá na aplikácie menších až stredných rozmerov. [4]

SQLite má hrubozrnné uzamykanie, ktoré umožňuje viacnásobných čitateľov, ale iba jedného zapisovateľa súčasne. Počas doby zápisu je databáza uzamknutá a nikto iný do nej počas toho nemôže zapisovať. SQLite sa síce snaží minimalizovať čas, v ktorom je databáza exkluzívne uzamknutá, ale keď má daná aplikácia spracovávať veľký počet transakcií za krátky čas, tak sa požiadavky na zápis do rovnakej databázy serializujú a to limituje celkovú transakčnú rýchlosť, čo sa môže prejaviť na výkone. Vtedy je efektívnejšie siahnuť po tradičnej klient/server databáze, ktorá má súčasný prístup viacerých entít k jednej databáze vyriešený pokročilejšie a dokáže zvládať vyššiu transakčnú prevádzku. [4]

SQLite nie je navrhnuté na prácu s veľkými databázami, praktický limit je stanovený v desiatkach gigabajtov. Keďže je všetko obsiahnuté v jednom súbore, kladie to vyššie nároky na súborový prehliadač a operačný systém a pri súboroch, ktoré sú v rádoch gigabajtov, sú zaznamenané výrazné poklesy výkonu. Ak je požiadavka na ukladanie a spracovanie niekoľko gigabajtov dát a čas hrá rolu, je lepšie zvážiť produkt viac zameraný na výkon.[4][5]

SQLite napríklad neimplementuje príkazy RIGHT OUTER JOIN a FULL OUTER JOIN. RIGHT OUTER JOIN sa ale dá napísať ako LEFT OUTER JOIN s prehodeným poradím tabuliek a FULL OUTER JOIN sa dá implementovať kombináciou iných relačných operácií podporovaných SQLite. Takýchto obmedzení je viac, napríklad nie je implementovaná plná podpora spúšťačov, príkazu ALTER TABLE, vnorených transakcií atď. Vo väčšine prípadov sa ale vždy nájde cesta ako tieto obmedzenia obísť a dostať sa k cieľu. [4]

2.5 Vhodné použitia SQLite

SQLite nie je priamo porovnateľný s klient/server SQL databázovými systémami ako MySQL, Oracle, PostgreSQL alebo SQL Server, pretože sa snaží vyriešiť iný problém. Hlavné priority SQLite sú efektívnosť, úspora zdrojov, spoľahlivosť, nezávislosť a jednoduchosť. SQLite sa nesnaží konkurovať vyššie uvedeným databázam, konkuruje `fopen()`.[6]

Pretože SQLite databáza nevyžaduje žiadnu administratívu a nezaberá veľa miesta, hodí sa dobre do zariadení, ktoré musia fungovať bez odborného zásahu človeka, napríklad embedded systémy, roboty, mikroprocesory alebo termostaty. V týchto zariadeniach je limitom výkon a práve SQLite databáza je navrhnutá tak, aby spotrebovala čo najmenej zdrojov.[6]

SQLite pracuje výborne pre webové stránky s nízkou až strednou prevádzkou (väčšina webov). Množstvo webovej prevádzky, ktoré môže SQLite zvládnuť závisí na tom, ako veľmi web využíva jej databázu. Všeobecne akýkoľvek web s menej než 100 000 návštevami denne by mal s SQLite databázou pracovať bez problémov. Toto číslo je len hrubý odhad, nie presná horná hranica. Ukázalo sa, že SQLite dokáže pracovať v desaťkrát väčšej prevádzke.[6]

Ďalšími využitiami SQLite sú analýza dát, lokálne úložiská, back-end časť menších aplikácií atď.[6]

2.6 Voľba a popis databázového systému

Voľba typu databázy pre túto aplikáciu, návrh a realizácia databázového systému sú dielom vedúceho záverečnej práce. Tieto prípravné práce boli prerekvizitou k zadaniu a boli hotové ešte pred jeho pridelením. Mojou úlohou bolo vytvorené databázy naštudovať a na základe nich vytvoriť knižnicu, ktorá bude plniť hlavné funkcie aplikácie. Ešte predtým je vhodné zmieniť prečo bolo vybrané práve SQLite.

Po SQLite sa siahlo hlavne preto, lebo aplikácia nepracuje s veľkým množstvom dát a výsledné databázové súbory nemajú veľké rozmery. Ďalším dôvodom je, že softvér je zadarmo, nemusia sa teda platiť žiadne licenčné poplatky za využívanie klientskych knižníc a databázového servera v aplikáciách. Na počítači, kde bude aplikácia pracovať sa nemusí nič konfigurovať, inštalovať ani vytvárať klientsky účet. Pre pohodlnejšie zálohovanie, presun alebo modifikáciu databázy je rozhodne SQLite lepšou voľbou. Databáza nebude ani pod veľkou záťažou (jedná sa o pár počítačov), tým pádom sa súčasné viacnásobné zapisovanie do databázy nijak neprejaví na výkone. Každopádne SQLite si s týmto dokáže viac-menej poradiť, ako je uvedené v kapitole 2.4.

Výhod, ktoré ponúkajú SQLite databázy je viac ako nevýhod a preto sa rozhodlo takto.

2.6.1 Databáza profilov

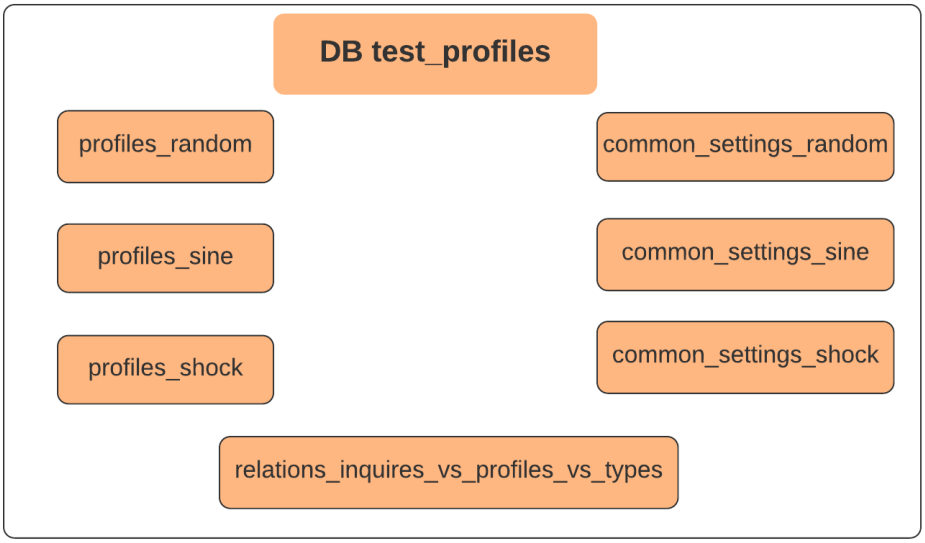
V databáze `test_profiles` je uložená časť nastavení testu, ktorá tvorí práve profily vibrácií a slúži k automatizácii ich zadávania pri tvorbe testovacieho súboru.

Táto časť nastavení bola oddelená od zvyšných nastavení (uložených v databáze testovacích parametrov) v samostatnej databáze, pretože v momente prijatia ponuky na vibračnú skúšku sa nezadávajú hneď všetky nastavenia, teda okrem týchto. Ak by boli profily vibrácií v databáze testovacích parametrov, tak by pri prijatí ponuky ostávalo veľa nevyplnených údajov a manipulácia pre vedúceho/zástupcu vedúceho laboratória by bola nepríjemná, pretože nemusia tušiť, čo všetko sa má zadať na rozdiel od skúšobného technika. Naviac v dobe prijatia ponuky a zadávaní profilov ešte nie je isté, či sa profil zahrnie aj do testovacích parametrov. Zadaný profil môže byť spoločný pre väčšie množstvo testovacích súborov a oddelením profilov sa zmenší veľkosť databázy testovacích parametrov. Profily vibrácií teda sú súčasťou databázy testovacích parametrov, pretože v nej pre každý testovací súbor existuje odkazom profil vibrácií v databáze profilov.

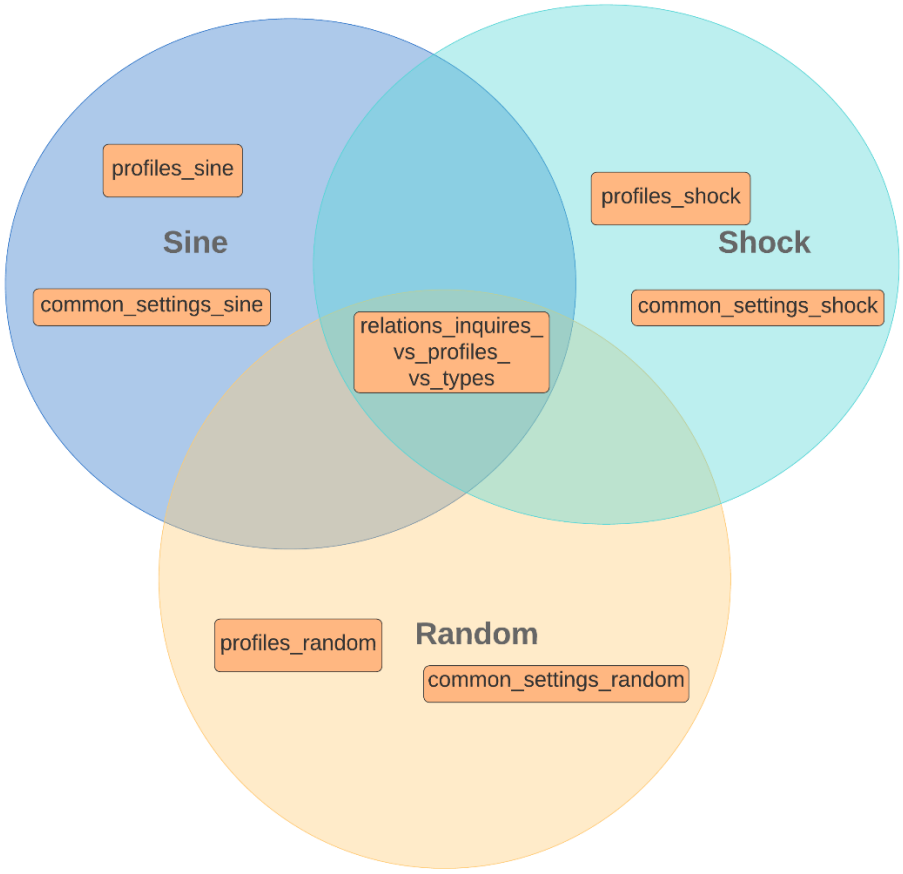
Databáza profilov obsahuje tabuľky, ktoré zobrazuje Obrázok 2.3. Tabuľky sa delia podľa typu testu na profily náhodných a sínusových vibrácií a profily rázov, ako ukazuje Obrázok 2.4. Platí, že jeden profil môže v tabuľke reprezentovať skupinu viacerých záznamov. Každý profil má jednoznačne pridelené ID – odpovedá tomu stĺpec `profile_Id` alebo `adProfileId`. Prípona „ad“ znamená, že sa hodnota nejakého identifikátoru vyskytuje duplicitne na viacerých riadkoch. Napríklad ID profilu, ak jeden profil tvorí viac záznamov. Táto prípona sa v databázach používa často a jej význam je stále rovnaký.

Profil sa zadáva a vzniká spolu s novou ponukou a je na ňu naviazaný pomocou nového záznamu v tabuľke relácií, kde jednej ponuke (určenej jednoznačne jej rokom, číslom a indexom) je priradený buď jeden alebo viac profilov. Zároveň platí, že jeden profil môže byť priradený viacerým ponukám. Identifikácia profilu je v tejto tabuľke doplnená o typ skúšky `adProfileType` (jedná sa iba o vibračné skúšky – hodnota 0) a `adProfileSubtype` (podtyp vibračnej skúšky, kde 0 je sínus, 1 náhodné vibrácie a 2 sú rázy).

Pri vytváraní profilu sa môžu okrem hlavných profilov (tabuľky `profiles_xxx`) zadať aj ďalšie voliteľné nastavenia, ktoré sa ukladajú do tabuliek `common_settings_xxx`. Pri sínusových a náhodných vibráciách sú nastavenia limitov uložené v jednotkách dB.



Obrázok 2.3: Vizualizácia databázy *test_profiles*



Obrázok 2.4 Príslušnosť tabuliek ku typu testu v databáze *test_profiles*

2.6.2 Databáza testovacích parametrov

V databáze `tests_parameters` sa nachádzajú ďalšie nastavenia, ktoré netvoria profily skúšok. Sú tu uložené všetky nastavenia z predošlých vykonaných testov a každý

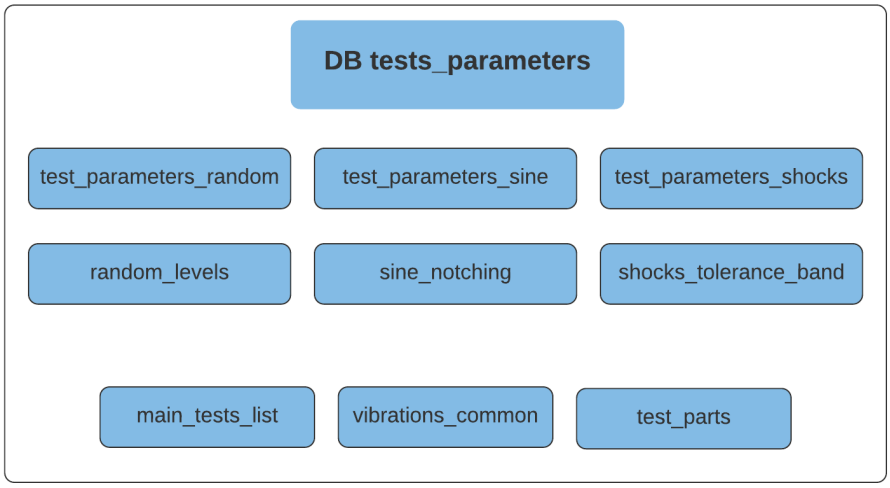
test je jednoznačne identifikovaný rokom a číslom testu. Pomocou tejto databázy je automatizované ručné vypĺňanie dát, kde si užívateľ vyberie konkrétny test a jeho nastavenia sa automaticky prenesú/zapíšu do užívateľského rozhrania aplikácie. Databáza sa skladá z tabuliek, ktoré zobrazuje Obrázok 2.5 a ich príslušnosť k jednotlivým typom vibračných skúšok ukazuje Obrázok 2.6.

Tabuľky `test_parameters_xxx` obsahujú nastavenia z karty „Control Parameters“. Každý riadok je jednoznačne definovaný číslom testu a rokom, v ktorom sa skúška vykonala. Význam a hodnoty nečíselných nastavení sú dokumentované v práci pána Nováka [3] (na základe tejto práce bola vytvorená softvérová knižnica) a v SQLite databázach sú uložené v rovnakej podobe ako v testovacích súboroch.

Tabuľka `vibrations_common` obsahuje nastavenia snímačov z karty „Input Channels“ a tieto nastavenia sú spoločné pre všetky typy vibračných skúšok. Každý riadok odpovedá jednému unikátnemu testu s definovaného rokom a číslom. Z tejto tabuľky sa rovnakým spôsobom ako v `test_parameters_xxx` automaticky vyplňujú nastavenia v aplikácii z užívateľom vybraného testovacieho súboru. Rovnako aj táto tabuľka slúži ako back-end časť a nie je nijak prezentovaná užívateľovi.

V tabuľkách `random_levels` a `shocks_tolerance_band` sú uložené ďalšie nastavenia náhodných vibračných skúšok a rázov z kariet „Level“ a „Tolerance Band“. Ich konfigurácia sa nachádza na samostatných kartách (rovnako aj vo vytvorenej aplikácii) a od predošlých nastavení sa líšia významovo aj štruktúrou, preto sú v samostatných tabuľkách. Platí, že nastavenia jedného testovacieho súboru sa nachádzajú na viacerých riadkoch, takže sa v názve stĺpca pre rok a číslo testu vyskytuje predpona „ad“. Vypĺňanie týchto nastavení je automatizované rovnako ako u predošlých s rozdielom, že prvok užívateľského rozhrania je jedna tabuľka. Tabuľka `sine_notching` v tomto texte popísaná nie je, pretože sa jej nastavenia a funkcionalitu nepodarilo implementovať do výslednej aplikácie. Dôvod bude uvedený v dokumentácii SW knižnice.

Tabuľky `test_parts` a `main_tests_table` obsahujú zoznam všetkých vykonaných testov a zaznamenávajú sa tu informácie o každom testovacom súbore. Každý test je v `test_parts` naviazaný na konkrétny profil, ktorý sa v ňom použil. V tabuľke `main_tests_table` je tiež zoznam všetkých testovacích súborov a každý test je tu zviazaný s konkrétnou ponukou. V týchto tabuľkách sa nachádzajú atribúty ako meno súboru, časová značka začiatku a konca testu, testovacia os, pracovná inštrukcia, názov firmy alebo číslo skúšobnej položky. Tieto parametre poskytujú dostatočné informácie o danom teste tak, aby ho skúšobný technik pri výbere dokázal rozpoznať. Dáta z týchto tabuliek sú spojené prienikom pomocou SQL príkazu `JOIN` a v aplikácii prezentované užívateľovi v tabuľkovom náhľade. Na základe nich si vyberá konkrétny testovací súbor, ktorého nastavenia sa automaticky v aplikácii vyplnia.



Obrázok 2.5 Vizualizácia databázy *test_parameters*

Obrázok 2.6 Príslušnosť tabuliek ku typu testu v databáze *test_parameters*

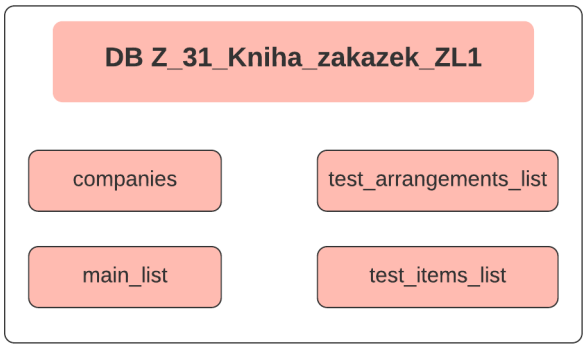
2.6.3 Databáza zákaziek

Databáza zákaziek obsahuje kompletnú evidenciu ponúk, objednávok, zákaziek, firiem a ďalších informácií potrebných pre laboratórium CVVOZE. Je primárne využívaná aplikáciou, ktorá spracováva objednávky, zákazky a ďalšiu administratívu a táto aplikácia nie je súčasťou tejto práce. Vytvorená aplikácia v tejto práci pracuje aj s touto databázou, no nie v celom jej rozsahu, ale iba s niektorými tabuľkami. Pracuje sa s ňou hlavne pri zadávaní, zobrazovaní a priradzovaní profilov k ponuke, kde sa z tejto

databázy získavajú údaje ako meno firmy, popis ponuky atď. Obrázok 2.7 zobrazuje využívané tabuľky databázy zákaziek.

V `main_list` tabuľke sú uložené všetky zaevidované ponuky a k ponukám pridelené zákazky, ak existujú. V kóde aplikácie sa pri práci s touto databázou vždy najprv vchádza do tejto tabuľky, obvykle s už známym číslom ponuky alebo zákazky a podľa týchto parametrov sa dohľadávajú ďalšie informácie ako ID firmy, existencia zákazky, popis ponuky apod. Potom sa podľa ID firmy v tabuľke `companies` nájde jej názov, ktorý je zas potrebný pre tvorbu priečinku, v ktorom bude testovací súbor uložený.

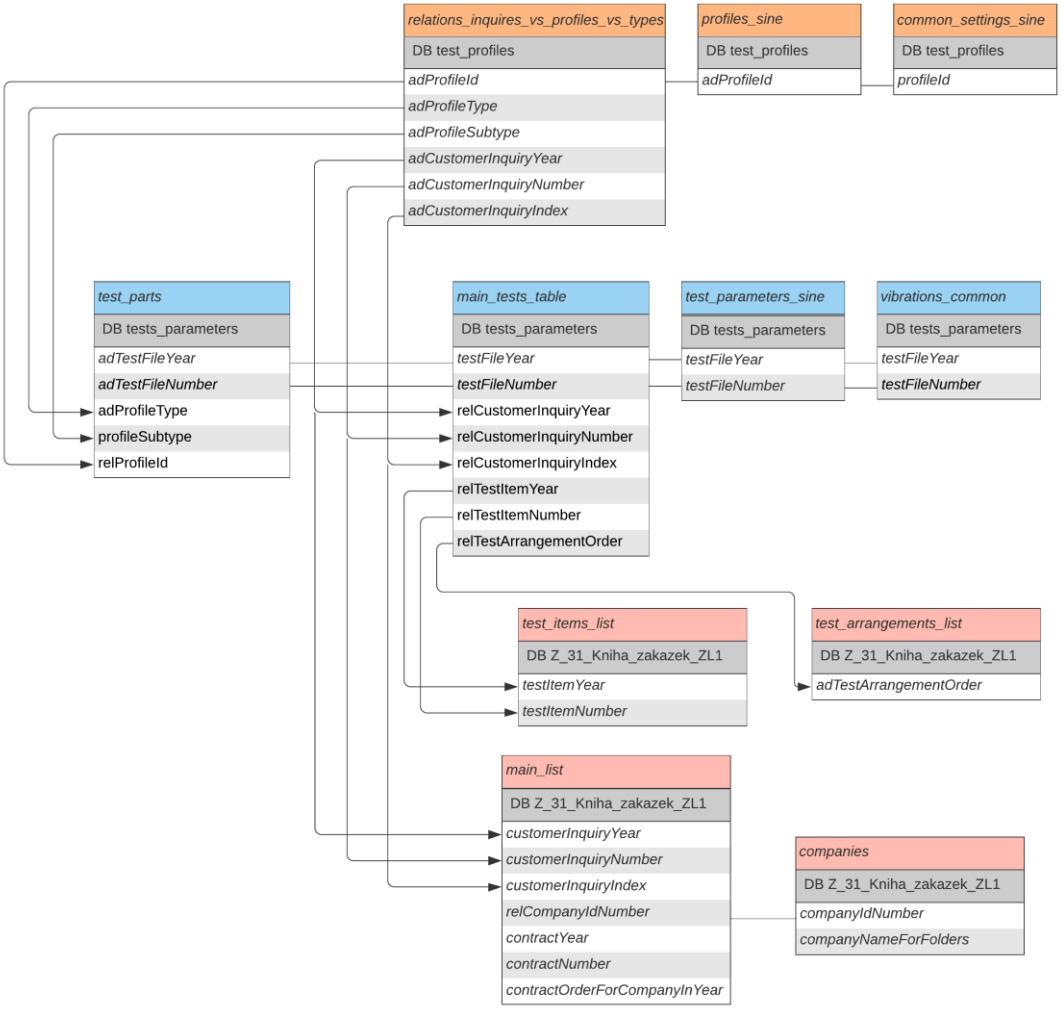
V tabuľkách `test_items_list` je uložený zoznam testovacích položiek a v `test_arrangements_list` zoznam testovacích zostáv. Obe tabuľky sa využívajú v aplikácii na konfiguráciu testu, kde sa z nich vyberie konkrétna testovacia položka alebo poradie zostavy na testovanie a priradí sa novo vytvorenému testovaciemu súboru. Operácie s touto databázou sú nezávislé od typu testu, preto nie je uvedený Vennov diagram.



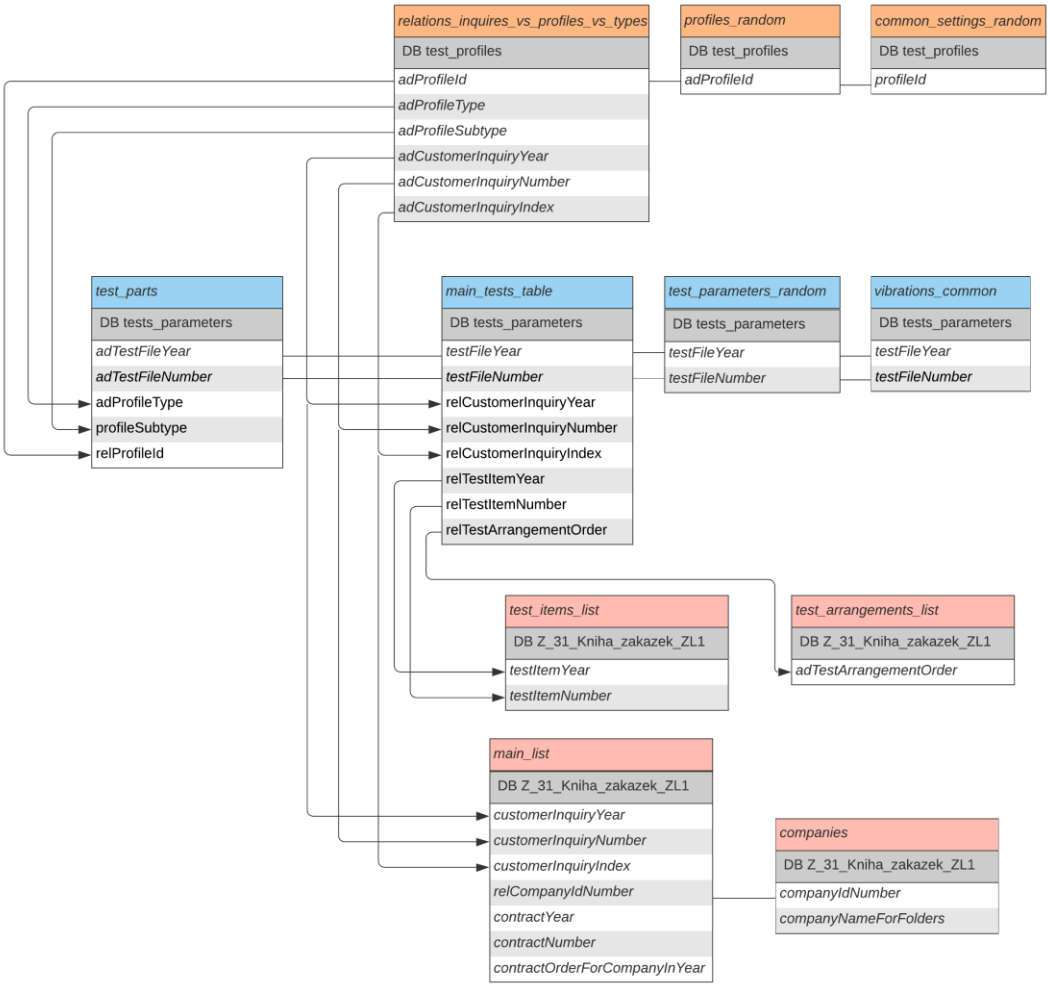
Obrázok 2.7: Vizualizácia databázy *kniha_zakazek*

2.6.4 Relácie medzi databázami

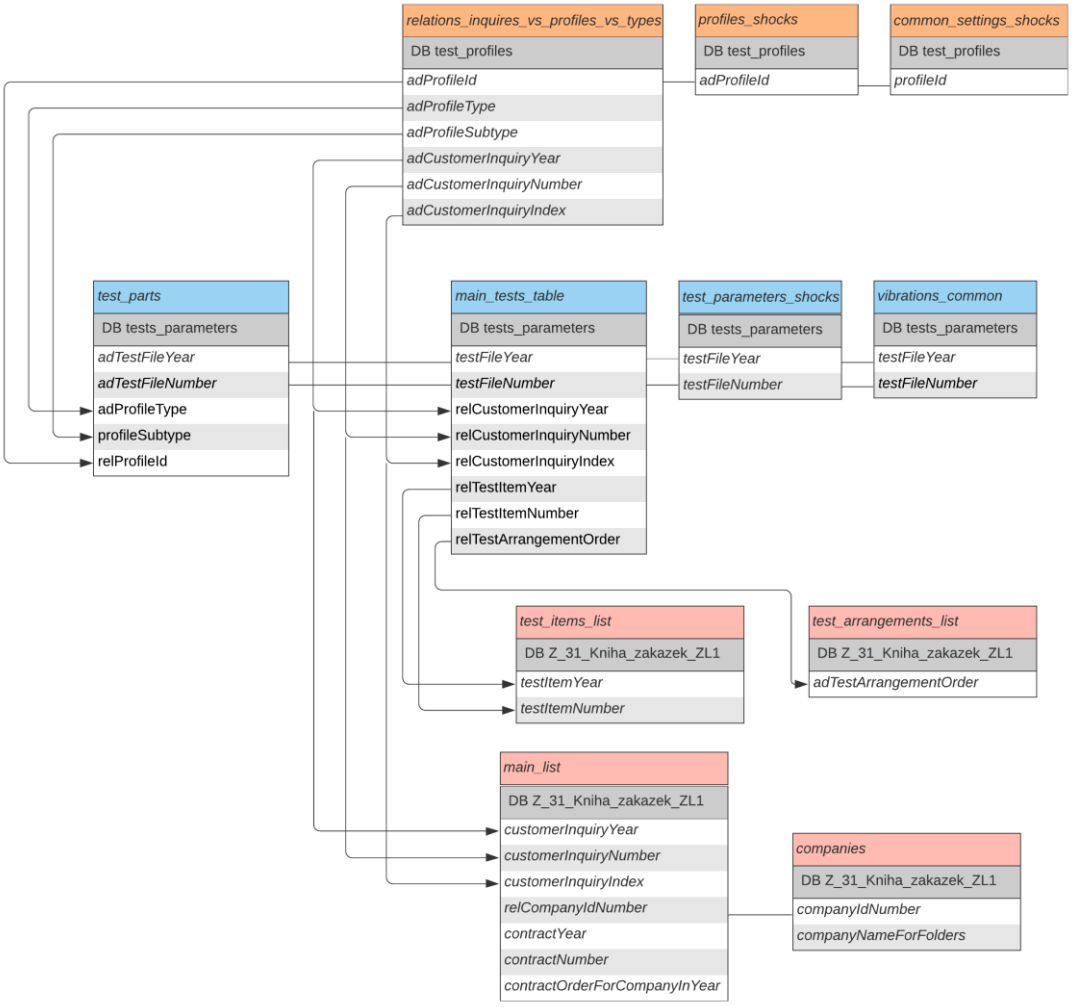
Pomocou prepojenia rôznych identifikátorov sa dajú odkazovať dáta medzi jednotlivými databázami. Databázy sú navrhnuté tak, že je previazané takmer všetko so všetkým a od jedného kľúča sa dá cez viacero odkazov dostať k ľubovoľným dátam v niektorej z databáz. Prepojenia medzi databázami zobrazujú Obrázok 2.8, Obrázok 2.9 a Obrázok 2.10 pre jednotlivé typy vibrácií. Tým, že môže byť jeden profil (určený ID, typom, podtypom) priradený viacerým ponukám a viac profilov môže byť priradených jednej ponuke, nie je správne len pomocou profilu alebo len podľa ponuky vyhľadávať v tabuľke relácií jednu konkrétnu položku, lebo dostaneme viacero výsledkov. Tento spôsob sa skôr hodí pri filtrácii profilov podľa konkrétnej ponuky, čo je v aplikácii implementované. Ďalej platí, že do databázy zákaziek by sa malo vstupovať s už konkrétnou ponukou a podľa nej získavať dáta o zákazke, názov firmy apod. Pri načítaní dát z existujúceho testu sa dá z tabuliek `main_tests_table` a `test_parts` jednoznačne určiť konkrétny profil, ktorý sa v teste použil a konkrétna ponuka, ktorá je s daným profilom a testom zviazaná.



Obrázok 2.8: Relácie medzi databázami - sínusové vibrácie



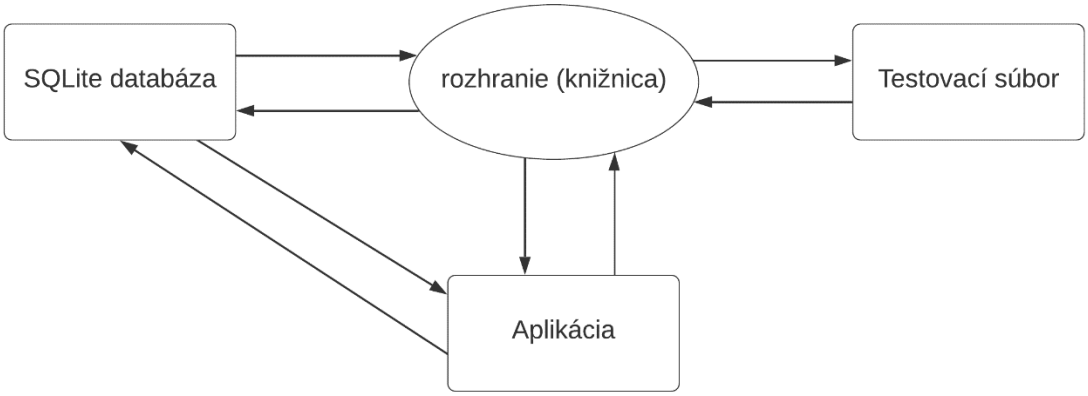
Obrázok 2.9: Relácie medzi databázami - náhodné vibrácie



Obrázok 2.10: Relácie medzi databázami - rázy

3. DOKUMENTÁCIA SW KNIŽNICE

Najprv bolo potrebné vytvoriť knižnicu, ktorej úlohou bude zabezpečovať získavanie dát z databáz a realizovať ich presuny medzi jednotlivými databázovými systémami. Metódy tejto knižnice využívajú hlavne SQL príkazy `INSERT`, `SELECT` a `UPDATE`. Knižnica taktiež slúži ako programové rozhranie medzi SQLite databázovým formátom a MDB databázami testovacích súborov a aplikáciou. Funkcionalitu a jej význam v spolupráci s aplikáciou zachytáva Obrázok 3.1.



Obrázok 3.1: Význam knižnice v aplikácii

Knižnica je písaná v jazyku C++ v prostredí Visual Studio 2019. Jedná sa o statickú knižnicu, čo má výhodu v tom, že kód tejto knižnice sa po skompilovaní kódu aplikácie nachádza v jej .exe súbore a nie je nutné pri nasadzovaní ručne pridávať dynamické knižnice do zložky s aplikáciou. Taktiež je výsledná zložka o niečo menšia.

Pre prácu s SQLite databázami je potrebný ich databázový engine, ktorý je obsiahnutý v zdrojovom súbore `sqlite3.c`, ktorý je voľne stiahnuteľný z oficiálnych stránok SQLite a následne pridaný k projektu. Do každého hlavičkového súboru sa ďalej musí zahrnúť súbor `sqlite3.h`.V prípade C++ a Visual Studia 2019 je potrebné stiahnuť balík („NuGet Package“) „`sqlite3_c_plus_plus`“, ktorý pridá k projektu predkompilované `sqlite3` binárne súbory a všetky potrebné dynamické knižnice. Potom sa už môže v kóde naplno využívať SQLite funkcionality pomocou jej C rozhrania. Pri používaní funkcií SQLite C knižnice je čerpané z ich oficiálnej dokumentácie [8].

Pre prácu s testovacími súbormi, ktoré sú v MDB formáte, sa používa ODBC aplikačné rozhranie. Keďže ide o starší typ súborov, musí sa použiť 32 bitový ODBC ovládač a tým pádom sa tomuto musí celá knižnica prispôsobiť a tiež pracovať v 32 bitovej verzii. Pre správne fungovanie ODBC rozhrania sa musia do hlavičkových súborov zahrnúť knižnice `windows.h`, `sql.h`, `sqltypes.h`, `sqlext.h` a `odbcinst.h`.

V tejto práci sa pri tvorbe niektorých funkcií, ktoré pracujú s testovacími súbormi, vychádzalo z knižnice, ktorá je vytvorená v práci pána Nováka [3]. Pri tvorbe týchto

funkcií bol kód p. Nováka spätne spracovaný a implementovaný tak, aby sedel do tohto riešenia. V zdrojových hlavičkových súboroch sú komentáre pri funkciách, ktoré vychádzali z knižnice p. Nováka. V tomto dokumente to bude uvedené tiež. V jeho bakalárskej práci bolo hlavne dokumentované to, akým spôsobom a kam sa ukladajú nastavenia v testovacom súbore a to bolo najpodstatnejšie. Všetky potrebné nastavenia skúšky sú totiž zahrnuté v jednej tabuľke System Data Float, napr. Tabuľka 1.5, ktorá má stovky riadkov a problém je v tom, že nikde nie je popísané na akých riadkoch sú potrebné nastavenia uložené. V jeho bakalárskej práci je toto prehľadne dokumentované pre všetky nastavenia a bolo to pre následné programovanie a prácu prínosom.

Vytvorená knižnica sa zameriava hlavne na presuny nastavení potrebných k vykonaniu skúšky. To sa týka databázy profilov, testovacích parametrov a testovacích súborov.

3.1 Používané SQLite C funkcie

V tejto kapitole sú popísané základné funkcie z SQLite C rozhrania, ktoré sa v metódach knižnice využívajú:

`sqlite3` [9]
- objekt databázového spojenia alebo ukazovateľ na otvorenú databázovú inštanciu
- vzniká pomocou `sqlite3_open()` a zaniká s `sqlite3_close()`

`sqlite3_open(const char *filename, sqlite3 **ppDb)` [10]
- pripája sa k novej alebo existujúcej databáze
- `filename` je názov databázového súboru a `ppDb` je ukazovateľ na ukazovateľ na databázovú inštanciu

`sqlite3_exec(sqlite3*, const char *sql, int (*callback), void *, char **errmsg)` [11]
- vykonáva `sql` príkaz a je jej predávané databázové spojenie ako prvý parameter
- ako tretí parameter sa predáva `callback` funkcia, čo sa využíva pri `SELECT` príkazoch
- namiesto parametru `void *` sa predáva premenná, ktorú chceme naplniť dátami, ktoré obdržíme `SELECT` príkazom
- `errmsg` je na zachytávanie chybových stavov, môže sa nahradiť `NULL`

`sqlite3_close(sqlite3*)`
- ničí databázové spojenie a dealokuje pamäť
- je deštruktorom pre objekt `sqlite3`

3.2 Popis tried

V tejto kapitole sú popísané triedy pre presun dát medzi databázami. Každej tabuľke v SQLite databázach odpovedá trieda, ktorá má v dátovej časti premenné, ktoré odpovedajú stĺpcom tabuľky. Jeden objekt predstavuje jeden riadok tabuľky a tento spôsob platí pre všetky triedy. Objekt sa naplní dátami z tabuľky a môže sa s ním v programe pracovať ďalej. Ak je potrebné získať a presúvať z tabuľky viac riadkov (jedná sa o niektoré tabuľky databázy profilov a testovacích parametrov), vytvorí sa `vector`, ktorý sa naplní objektami (riadkami) a môže sa s ním rovnako pracovať ďalej. Dátová časť triedy je `public`, čo ide trochu proti pravidlu, že dáta v triede majú byť vždy `private`. Tu je ale trieda využívaná iba ako dátový kontajner a ide o to, aby sa k jej dátam mohlo jednoducho pristupovať. To isté by mohla zabezpečovať aj C štruktúra, ale nemohlo by sa využiť výhod, ktoré zas ponúkajú triedy (takisto sú modernejšie). Každá trieda má svoje metódy, ktoré sa starajú o obojsmerné presuny dát medzi SQLite databázami a testovacími súbormi. Získavanie nastavení z testovacieho súboru je implementované pre triedy testovacích parametrov a snímačov. Získavanie nastavení profilov a nastavení z Level, Tolerance-Band a Notching kariet nie je až tak využívané a malo to v rámci aplikácie najmenšiu prioritu.

3.2.1 Trieda profilov

Trieda má názov `CRandomProfiles` a slúži na prenos dát tabuľky `profiles_random` z databázy profilov a zápis týchto dát do testovacieho súboru pre náhodné vibrácie. Je uvedená ako vzorový príklad, pre sínusové profily a profily rázov (u rázov sa nevyužíva `vector`, jeden profil odpovedá jednému riadku tabuľky) je trieda vytvorená rovnako, len s inými názvami premenných a metód (`CSineProfiles` a `CShockProfiles`).

Dátová časť (`public`)

```
int ProfileID
double Frequency, Amplitude, WarnLimitPositive, WarnLimitNegative,
AbortLimitPositive, AbortLimitNegative
```

Metódy (`public`)

`CRandomProfiles()`

- implicitný konštruktor, incializuje premenné na predvolené hodnoty

`~CRandomProfiles()`

- deštruktor objektu `CRandomProfiles`

```
void PrintVectorRandomProfiles(vector <CRandomProfiles>& aVec)
```

- vypíše na konzolu pod sebou obsah triedy (využívalo sa to pri testovaní funkčnosti knižnice)

- vstupným parametrom je `vector` naplnený objektmi `CRandomProfiles`

`void InsertToRandomProfiles(sqlite3* aDB, vector <CRandomProfiles>& aVec)`

- vloží do SQLite tabuľky nový profil pomocou SQL príkazu `INSERT`
- parameter `aDB` predstavuje aktuálne otvorenú databázová inštanciu a `aVec` je kontajner naplnený novými nastaveniami profilu

`vector <CRandomProfiles> FetchFromRandomProfiles(sqlite3* aDB, int aProfileID)`

- pomocou nej sa získavajú záznamy z SQLite tabuľky na základe ID profilu pomocou SQL príkazu `SELECT`
- parameter `aDB` predstavuje aktuálne otvorenú databázu a `aProfileID` číslo profilu
- vo volanej `sqlite3_exec()` funkcii sa ako tretí parameter predá `callback` funkcia `callbackRandomProfiles` a spolu s ňou ako štvrtý parameter lokálna premenná typu `vector <CRandomProfiles>`, ktorá sa v rámci `callbackRandomProfiles` naplní dátami a vráti sa späť do `FetchFromRandomProfiles`
- vracia `vector` naplnený objektmi `CRandomProfiles`

`friend int callbackRandomProfiles(void* aVector, int argc, char** argv, char** azColName)`

- `callback` funkcia, ktorá je volaná vždy, ak chceme z SQLite databázy získať nejaké dáta
- `aVector` môže byť ľubovoľná premenná, ktorá sa naplní výsledkom zo `SELECT` príkazu
- `argv` je pole, ktoré obsahuje výsledok zo `SELECT` príkazu
- parameter `argc` nie je využitý, rovnako ani `azColName`, čo je pole s názvami stĺpcov
- návratová hodnota `int` nie je využitá

`void InsertFrom_RandomProfiles_ToTestFileRandom(string aFile, vector <CRandomProfiles>& aVec)`

- presunie dáta uložené v `aVec` do MDB testovacieho súboru `aFile`
- limity sú prepočítané podľa vzorca v [3]
- vychádza z knižnice p. Nováka

3.2.2 Trieda testovacích parametrov

Táto trieda má názov podľa typu testovacieho súboru (`CRandomVibrations`, `CSineVibrations` a `CShocks`) a slúži na prenos dát z tabuliek `test_parameters_xxx`,

kde `xxx` je `random`, `sine`, a `shocks`, v databáze `tests_parameters`. Podoba týchto tried sa medzi typmi testovacích súborov okrem názvov premenných a metód nelíši a preto je popísaná jedna vzorová `CRandomVibrations`. Tieto nastavenia sú pre jeden testovací súbor uložené vždy na jednom riadku, takže nie je potreba využívať `vector`.

Dátová časť (`public`)

```
int TestFileYear, TestFileNumber
double Clipping
int MultiChannelControl
double MassTestItems, MassTestFixture
int ConnectionWithSlipTable, WarnLimitWithLines,
AbortLimitWithLines, DegreesOfFreedom, PreTestLevel, LevelShift
double CurveSave
int CurveSaveTime, EnablePosValues, SelectionOfBandwidth
```

Metódy (`public`)

`CRandomVibrations()`

- implicitný konštruktor, inicializuje premenné na predvolené hodnoty

`~CRandomVibrations()`

- deštruktor objektu `CRandomVibrations`

`void PrintVectorRandomVibrations()`

- vypíše na konzolu pod sebou obsah triedy

`void InsertToRandomVibrations(sqlite3* aDB)`

- vloží do SQLite tabuľky nový záznam SQL príkazom `INSERT`
- parameter `aDB` predstavuje aktuálne otvorenú databázová inštanciu

`CRandomVibrations& FetchFromRandomVibrations(sqlite3* aDB, int TestYear, int TestNumber)`

- pomocou nej sa získava záznam z SQLite tabuľky podľa čísla a roku testu pomocou SQL príkazu `SELECT`
- parameter `aDB` predstavuje aktuálne otvorenú databázu a `TestYear` a `TestNumber` rok testu a jeho číslo
- vo volanej `sqlite3_exec()` funkcii sa ako tretí parameter predá `callback` funkcia `callbackRandomVibrations` a spolu s ňou ako štvrtý parameter `this`, ktorý sa v rámci `callbackRandomVibrations` naplní dátami a vráti sa späť do `FetchFromRandomVibrations`
- vracia `CRandomVibrations&`

`friend int callbackRandomVibrations(void* aClass, int argc, char** argv, char** azColName)`

- `callback` funkcia, ktorá je volaná vždy, ak chceme z SQLite databázy získať nejaké dáta
- `aClass` môže byť ľubovoľný objekt, ktorý sa naplní výsledkom zo `SELECT` príkazu
- `argv` je pole, ktoré obsahuje výsledok zo `SELECT` príkazu
- parameter `argc` nie je využitý, rovnako ani `azColName`, čo je pole s názvami stĺpcov
- návratová hodnota `int` tiež nie je využitá

`void InsertFrom_RandomVibrations_ToTestFileRandom(string aFile)`

- presunie dáta uložené v `this` do MDB testovacieho súboru `aFile`
- vychádza z knižnice p. Nováka

`CRandomVibrations&`
`FetchFrom_TestFileRandom_ToRandomVibrations(string aFile)`

- získa dáta z testovacieho súboru `aFile` a naplní nimi `this`
- vracia `CRandomVibrations&`
- vychádza z knižnice p. Nováka

`string UpdateRandomVibrations(sqlite3* aDB, CRandomVibrations old)`

- prepisuje starý záznam v SQLite databáze novým, ak sú záznamy rozdielne
- `aDB` predstavuje aktuálne otvorenú SQLite databázu, `old` je objekt naplnený starými hodnotami z databázy a `this` obsahuje nové nastavenia získané z testovacieho súboru
- dátové časti `this` a `old` sa porovnajú a ak sú rozdielne, prepíše sa záznam (určený číslom a rokom testu v objekte `old`) v SQLite databáze
- ak dôjde k zmene parametrov vráti `string` so správou, že sa dáta v danej tabuľke zmenili, v opačnom prípade vráti prázdny `string`

3.2.3 Trieda nastavení snímačov

Trieda `CVibrationsCommon` sa používa na presun nastavení všetkých ôsmich snímačov z tabuľky `vibrations_common` v databáze `tests_parameters`. V SQLite databáze tieto nastavenia nezávisia na type testu, takže `SELECT`, `INSERT` a `UPDATE` operácie sú rovnaké. Pri presunoch do a z testovacích súborov to už neplatí, nastavenia snímačov sú v `System Data Float` tabuľke uložené na iných pozíciách. Nastavenia pre jeden testovací súbor sú uložené na jednom riadku, takže nie je potreba využívať `vector`.

Dátová časť (`public`)

```
int TestFileYear, TestFileNumber
int TypeOfInput[8]
int RelSensorID[8]
double Sensitivity[8]
int Unit[8]
double LowerLimit[8]
double UpperLimit[8]
vector <string> InfoText
int Notching[8]
```

Metódy (`public`)

`CVibrationsCommon()`

- implicitný konštruktor, inicializuje premenné na predvolené hodnoty

`~CVibrationsCommon()`

- deštruktor objektu `CVibrationsCommon`

`void PrintVibrationsCommon()`

- vypíše na konzolu pod sebou obsah triedy

`void InsertToVibrationsCommon(sqlite3* aDB)`

- vloží do SQLite tabuľky nový záznam príkazom `INSERT`
- parameter `aDB` predstavuje aktuálne otvorenú databázová inštanciu

`CVibrationsCommon& FetchFromVibrationsCommon(sqlite3* aDB, int aTestNumber, int aTestYear)`

- pomocou nej sa získava záznam z SQLite tabuľky podľa čísla a roku testu pomocou SQL príkazu `SELECT`
- parameter `aDB` predstavuje aktuálne otvorenú databázu a `aTestYear` a `aTestNumber` rok testu a jeho číslo
- vo volanej `sqlite3_exec()` funkcii sa ako tretí parameter predá `callback` funkcia `callbackVibrationsCommon` a spolu s ňou ako štvrtý parameter `this`, ktorý sa v rámci `callbackVibrationsCommon` naplní dátami a vráti sa späť do `FetchFromVibrationsCommon`
- vracia `CVibrationsCommon&`

`friend int callbackVibrationsCommon(void* aClass, int argc, char** argv, char** azColName)`

- `callback` funkcia, ktorá je volaná vždy, ak chceme z SQLite databázy získať nejaké dáta
- `aClass` môže byť ľubovoľná premenná, ktorá sa naplní výsledkom zo `SELECT` príkazu
- `argv` je pole, ktoré obsahuje výsledok zo `SELECT` príkazu

- parameter `argc` nie je využitý, rovnako ani `azColName`, čo je pole s názvami stĺpcov
- návratová hodnota `int` tiež nie je využitá

```
void InsertFrom_VibrationsCommon_ToTestFileRandom(string aFile)
```

- presunie nastavenia snímačov uložené v `this` do MDB testovacieho súboru `aFile`

```
CVibrationsCommon&
FetchFrom_TestFileRandom_VibrationsCommon(string aFile)
```

- získa dáta z testovacieho súboru `aFile` typu náhodné vibrácie a naplní nimi `this`
- vracia `CVibrationsCommon&`

Ďalšie `Insert` a `Fetch` funkcie sú pre zvyšné dva typy testovacích súborov rovnaké, líšia sa len pozíciou nastavení v `System Data Float` tabuľke.

```
string UpdateVibrationsCommon(sqlite3* aDB, CVibrationsCommon
oldRecord, bool sinus)
```

- prepisuje starý záznam v SQLite databáze novým, ak sú záznamy rozdielne
- `aDB` predstavuje aktuálne otvorenú SQLite databázu, `oldRecord` je objekt naplnený starými hodnotami z databázy, `this` obsahuje nové nastavenia získané z testovacieho súboru a `sinus` je `true`, ak sa jedná o typ súboru pre sínusové vibrácie
- dátové časti `this` a `oldRecord` sa porovnajú a ak sú rozdielne, prepíše sa záznam (určený číslom a rokom testu v objekte `oldRecord`) v SQLite databáze
- ak dôjde k zmene parametrov vráti `string` so správou, že sa dáta v danej tabuľke zmenili, v opačnom prípade prázdny `string`

3.2.4 Triedy pre nastavenia Levels, Tolerance-Band a Notching

Dátové časti tried `CLevels`, `CToleranceBand`, a `CNotching` odpovedajú tabuľkám `random_levels`, `shocks_tolerance_band` a `sine_notching` v databáze `tests_parameters`. Ich metódy sa nijak nelíšia od predošlých, akurát tu sú nastavenia pre jeden testovací súbor uložené na viacerých riadkoch, takže sa na presun dát používa `vector` naplnený príslušnými objektmi. Trieda `CNotching` je naprogramovaná, ale nebola vo výslednej aplikácii implementovaná, čo bude viac rozvedené v kapitole 4.4.

3.3 Funkcie mimo tried

V tejto kapitole sú popísané funkcie knižnice, ktoré nie sú súčasťou tried.

`void CreateTestFile(string aDefaultFile, string aNewFile)`

- vytvorí nový testovací súbor, do ktorého sa zapíšu nastavenia zadané v aplikácii
- v zložke, v ktorej sa spúšťa aplikácia sú 3 predvolené testovacie súbory `Default.*`, z ktorých sa vytvára kópiou nový testovací súbor a do tohto súboru sú predošlými metódami zapísané nové nastavenia
- `aDefaultFile` je cesta k základnému súboru a `aNewFile` je miesto kam sa má vytvoriť nový testovací súbor

`string RenameToMDB(string aFile)`

- zmení koncovku testovacieho súboru na .mdb
- testovacie súbory sú síce typu MDB, ale majú svoje vlastné koncovky ako .sin, .rau alebo .shk, takže ešte predtým, než sa začne so súborom pracovať, musí sa zmeniť jeho prípona na .mdb
- `aFile` je cesta k danému súboru

```
string RenameToRau(string aMDBFile)
string RenameToSin(string aMDBFile)
string RenameToShk(string aMDBFile)
```

- zmenia koncovku testovacieho súboru na .rau, .sin alebo .shk
- po skončení práce s testovacím súborom sa musí zmeniť jeho koncovka z .mdb na vyššie uvedené, aby sa mohol testovací súbor nahrať do SWR1200 a mohlo sa spustiť meranie
- `aMDBFile` je cesta k .mdb súboru

`vector<string> FetchLogbookTimes(string aFile)`

- získa z testovacieho súboru, z tabuľky `Logbook`, napr. Tabuľka 1.2 čas spustenia a ukončenia testu
- `aFile` je cesta k testovaciemu súboru
- vráti `vector` dvoch `string`-ov, prvý je začiatočný čas a druhý je koncový čas

`vector<string> ReadConfigFile(string location)`

- načíta dáta z konfiguračného súboru
- v zložke, z ktorej sa spúšťa aplikácia je konfiguračný súbor `config.txt`, ktorý obsahuje cesty k databázam pre jednotlivých užívateľov
- v `config.txt` je na prvom riadku cesta k databáze profilov, druhý riadok odpovedá databáze testovacích parametrov, tretí riadok databáze zákaziek, na štvrtom riadku je cesta, do ktorej sa uloží testovací súbor, ak k nemu existuje číslo zákazky a na piatom riadku je cesta, do ktorej sa uloží testovací súbor, ak mu je priradená iba ponuka
- `location` je cesta ku `config.txt` súboru

– vráti `vector` piatich `string`-ov, ktorých poradie odpovedá poradiu v `config.txt`

4. DOKUMENTÁCIA APLIKÁCIE

V tejto práci sú vytvorené dve aplikácie, jedna slúži na správu profilov a druhá na samotnú konfiguráciu testu. Tieto aplikácie boli vytvorené pomocou knižnice Qt v jazyku C++. Budú dokumentované iným štýlom ako knižnica, pretože vytvorených funkcií je veľa a počas písania dokumentácie sa kód často menil, takže popisovať ich všetky nebolo udržateľné. Dokumentácia sa bude hlavne zameriavať na predstavenie funkcionality a na správne používanie aplikácií. Každá funkcia je samozrejme v kóde okomentovaná a popísaná.

4.1 Qt

Qt je multiplatformová knižnica na tvorbu grafických užívateľských rozhraní, ktorá bola vytvorená v jazyku C++.[12] Voľba nástroja padla práve na Qt, pretože sa za jeho používanie nič neplatí, programuje sa v ňom objektovo (lepšia čitateľnosť kódu) a má veľkú komunitu aktívnych používateľov. Ako vývojové prostredie bolo použité Qt Creator, ktoré obsahuje modul Qt Designer na tvorbu formulárov. Kód je kompilovaný pomocou kompilátoru MSVC 2017, ktorý je kompatibilný aj s najnovšou verziou MSVC 2019.

4.2 Správa profilov

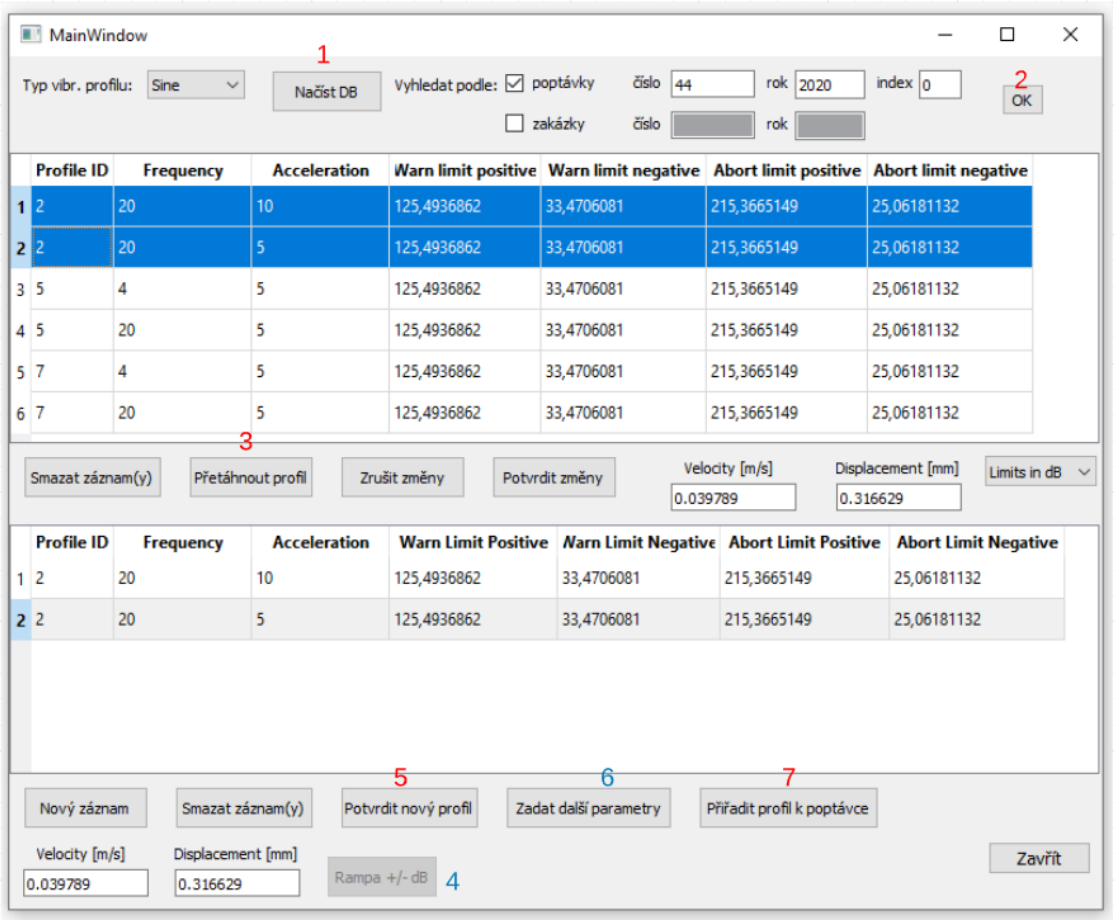
Prvá aplikácia sa zameriava na správu profilov vibrácií, ktoré tvoria časť nastavení z testovacieho súboru. Aplikácia pracuje primárne s databázou profilov a zameriava sa na ich správu. Obvykle sa profily vibrácií zadávajú pri prijatí ponuky a vzápätí sú k nej priradené (veci, ktoré táto aplikácia zabezpečuje), takže pre väčšiu prehľadnosť a menšiu zložitosť bola pre profily vytvorená samostatná aplikácia. Slúži na zadávanie a úpravu profilov, ich priradenie k ponuke alebo filtráciu existujúcich profilov podľa ponuky alebo zákazky a pracuje s databázou profilov a zákaziek. Aplikácia je v 64 bitovej verzii a nevyužíva sa tu vytvorená knižnica, pretože sa tu nepracuje s testovacími súbormi a metódy na presun nastavení v triede profilov tu nenašli využitie (Qt podporuje SQL jazyk a prácu s SQLite databázami). Pred spustením aplikácie je potrebné nainštalovať Visual C++ Redistributable (x64), ktorý sa nachádza v zložke s aplikáciou. Pri nasadzovaní boli všetky potrebné Qt závislosti pridané pomocou nástroja `windeployqt.exe`.

V nasledujúcej časti bude popísaná funkcionalita a odporúčaný prechod aplikáciou. Kód je pravidelne komentovaný a táto kapitola by mala slúžiť hlavne užívateľovi ako príručka k používaniu s popisom, čo sa približne deje v back-end časti.

Postup pri vytváraní nového profilu je nasledovný:

1) Načíta sa databáza profilov podľa typu testu. Profily sú usporiadané vzostupne podľa ID.

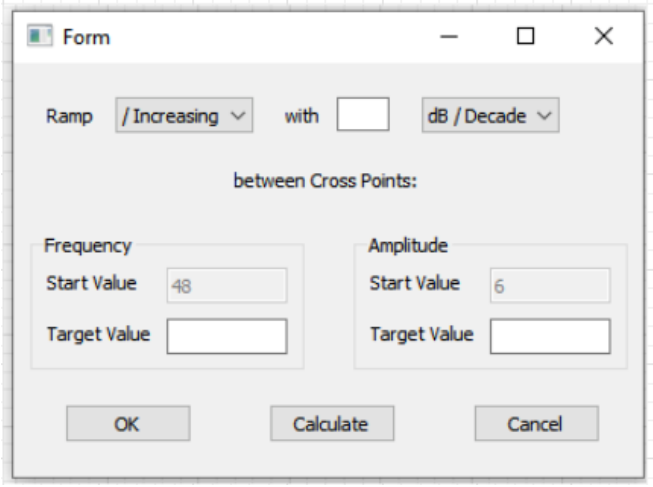
2) Môže sa použiť filter buď podľa parametrov ponuky alebo zákazky, filtre nie sú prepojené, vždy má byť aktívny iba jeden.



Obrázok 4.1: Hlavné okno

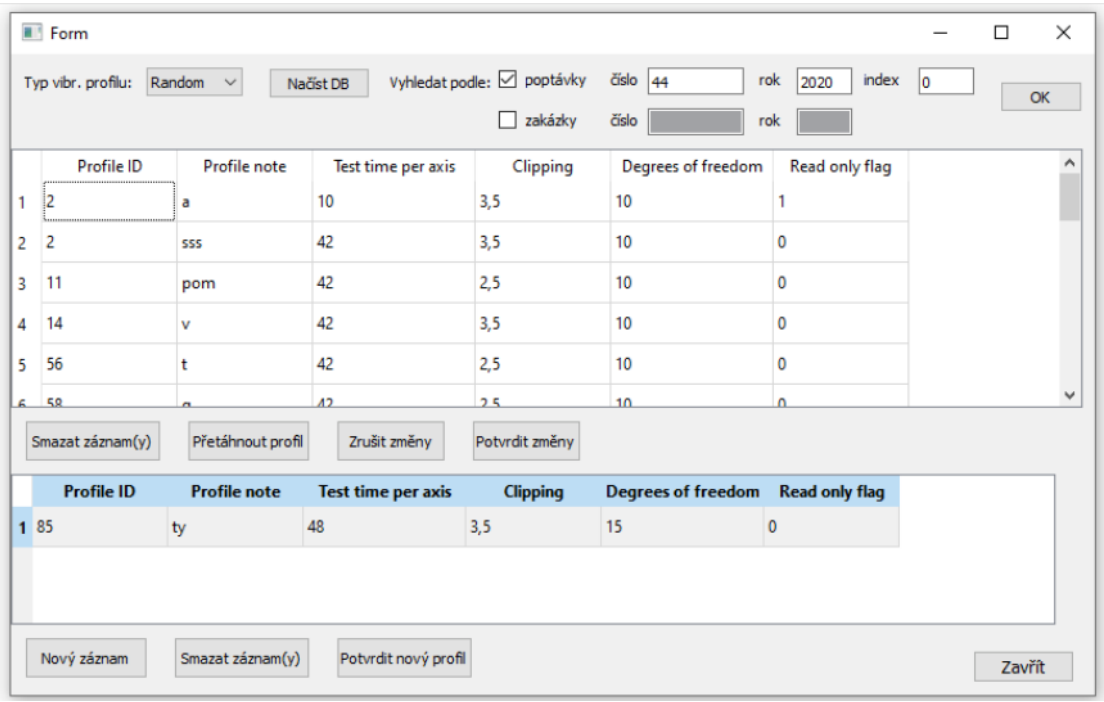
Načítané dáta sa v hornej tabuľke môžu upravovať, mazať a potvrdením zmien sa uložia do databázy. Zmeny v tabuľke môžu byť vrátené späť, no po uložení do databázy už nie. Pre označený riadok sa prepočítajú a zobrazia hodnoty rýchlosti a posunutia. Pri sínusových a náhodných vibráciách sú limity zobrazované v decibeloch alebo percentách, no v databáze sú vždy uložené v decibeloch.

3) Označené riadky sa z hornej tabuľky presunú do spodnej, ktorá slúži na vytváranie nového profilu, keďže väčšina profilov vzniká z už existujúcich. V nej sa môžu dáta upravovať, mazať alebo pridávať nové záznamy.
4) Karta Rampa je voliteľná a je prístupná iba u náhodných vibrácií pri tvorbe nového profilu. Jej funkcia je rovnaká ako v programe SWR 1200. Najprv sa musí označiť jeden záznam zo spodnej tabuľky, na jej karte sa zadajú parametre rampy a koncová frekvencia, následne sa dopočíta amplitúda a do spodnej tabuľky sa vloží nový záznam.



Obrázok 4.2: Karta nastavenia rampy

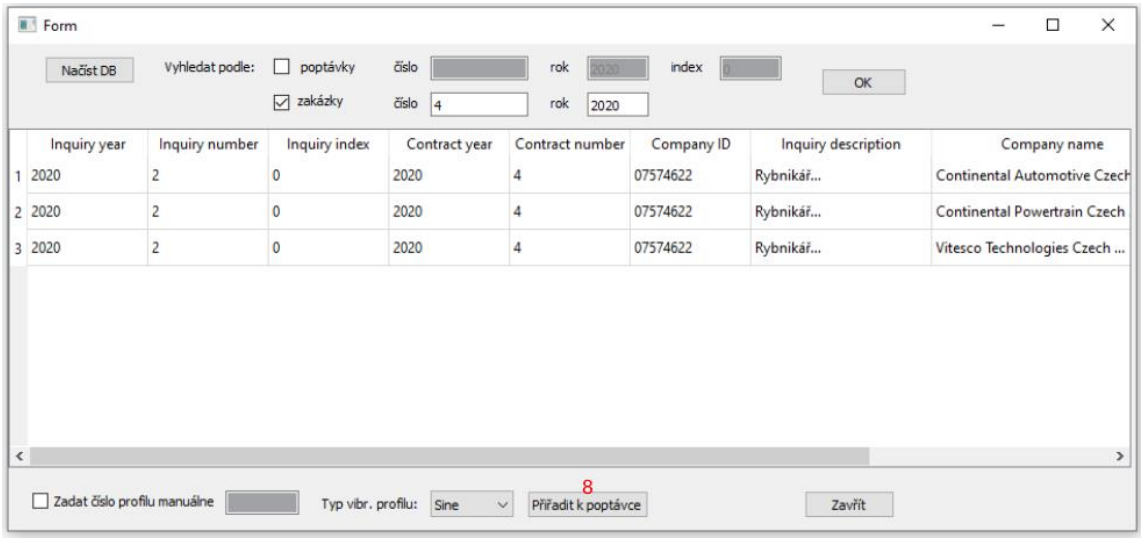
5) Po zadaní nastavení nového profilu sa mu priradí nové ID inkrementáciou posledného a dané záznamy sa presunú do hornej tabuľky a do databázy.
6) Prejde sa na kartu voliteľných nastavení k profilom, ktoré sa týkajú tabuliek `common_settings_xxx` (kde `xxx` je `random`, `sine`, `shock`). Užívateľské rozhranie je tu podobné ako v hlavnom okne. Po potvrdení týchto nastavení sa priradí danému záznamu ID z novo vytváraného profilu v hlavnom okne zo spodnej tabuľky, takže táto operácia musí nasledovať hneď po zadaní hlavného profilu.



Obrázok 4.3: Karta voliteľných nastavení k profilom

7) Prejde sa na kartu odkiaľ sa priradzuje nový profil ponuke. Na nej sa príslušným tlačidlom načítajú všetky ponuky ako prienik viacerých tabuliek z knihy zákaziek a môžu sa dofiltrovať rovnakým spôsobom ako v predošlých kartách.
8) Po označení riadku a stlačení tlačidla sa priradí novo vytváraný profil danej ponuke zápisom nového záznamu do tabuľky relácií v databáze profilov. ID nového profilu sa berie zo spodnej tabuľky v hlavnom okne, takže sa tam ten profil musí nachádzať a táto operácia má nasledovať až po vytvorení profilu v hlavnom okne, resp. po zadaní vedľajších parametrov profilu.

Je tu možnosť priradiť číslo profilu aj manuálne mimo daný sled operácií, vtedy stačí ak profil existuje v databáze.



Obrázok 4.4: Karta na priradenie profilu k ponuke

4.3 Konfigurácia testu

Druhá aplikácia slúži na konfiguráciu a vytváranie testovacích súborov, ktorých nastavenia sa nachádzajú v databáze testovacích parametrov a táto aplikácia spolupracuje primárne s touto databázou. Na jednotlivých kartách sa nastavujú parametre testu (užívateľské rozhranie má podobný vzhľad ako v oficiálnom softvéri RMS) s možnosťou importovania nastavení z už existujúcich súborov, takže odpadá celé ručné zadávanie, no je ponechaná aj táto možnosť. Taktiež je implementovaná väčšina funkcií a rôznych prepočtov tak, aby sa aplikácia v tomto smere podobala čo najviac Test Manager-u od firmy RMS. Manipulácii s testovacím súborom je venovaná karta `Export`, v ktorej sa zadávajú údaje o testovacom súbore (už mimo samotných nastavení), tak aby bol súbor jasne rozpoznateľný a korektný v rámci evidencie. Jeho názov a miesto uloženia závisí na týchto údajoch, ktoré sú zadávané skúšobným technikom alebo sú do určitej miery automatizované. V aplikácii je možné načítať celý testovací súbor a poprípade z neho vytvoriť kópiu nového súboru „1:1“. Ďalej je možné načítaného súboru získať časové

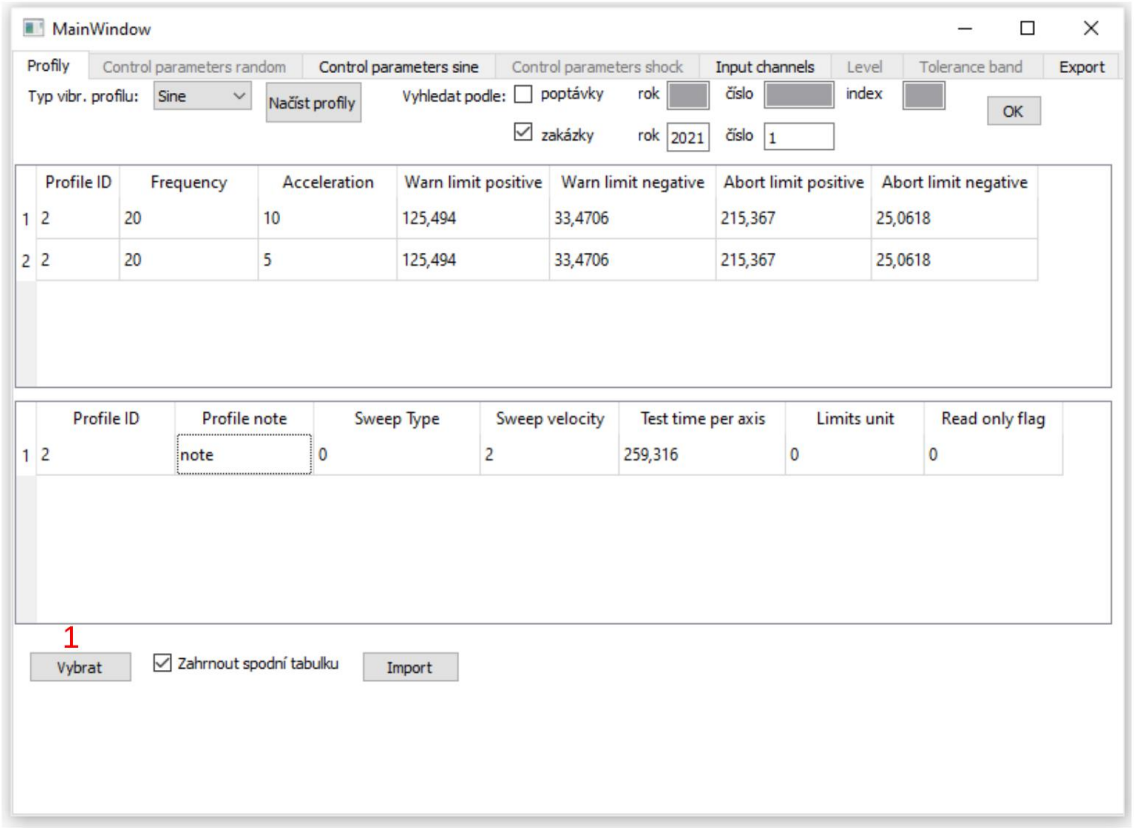
značky začiatku a konca skúšky a uložiť ich do SQLite databázy. V prípade zmeny testovacích parametrov sa dajú zmenené údaje z testovacieho súboru získať a prepísať neaktuálne v SQLite databáze.

V aplikácii je naplno využitá naprogramovaná knižnica, ktorá je k nej pripojená staticky. Keďže knižnica musela byť kvôli testovacím súborom v 32 bitovej verzii, musí sa tomuto prispôsobiť aj aplikácia. Pred jej spustením je potrebné nainštalovať Visual C++ Redistributable (x86), ktorý sa nachádza spolu s ňou v zložke. Pri nasadzovaní boli všetky potrebné Qt závislosti k aplikácii pridané pomocou nástroja `windeployqt.exe`.

V nasledujúcej časti bude popísaný prechod aplikáciou v dvoch režimoch – vytvorenie nového testovacieho súboru (režim 1) a práca s existujúcim súborom (režim 2). Kód je pravidelne komentovaný a táto kapitola by mala slúžiť hlavne užívateľovi ako návod k efektívnemu používaniu programu s jemným popisom back-end časti.

Postup pri vytváraní nového testovacieho súboru je nasledovný (režim 1, červená farba):

Na prvej karte (Obrázok 4.5) sa najprv načítajú a vyfiltrujú profily z databázy, v tomto je postup rovnaký ako v aplikácii profilov. V hornej tabuľke sú hlavné nastavenia profilov a v spodnej voliteľné nastavenia.

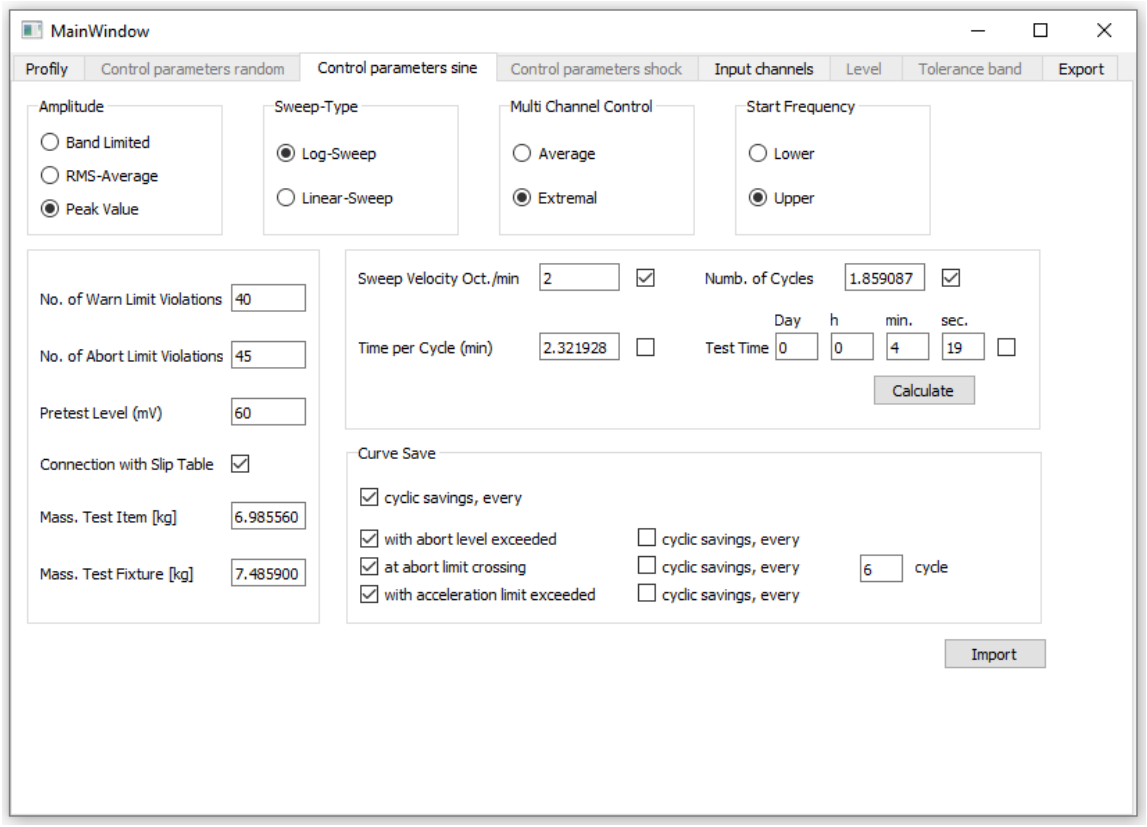


Obrázok 4.5: Výber profilu

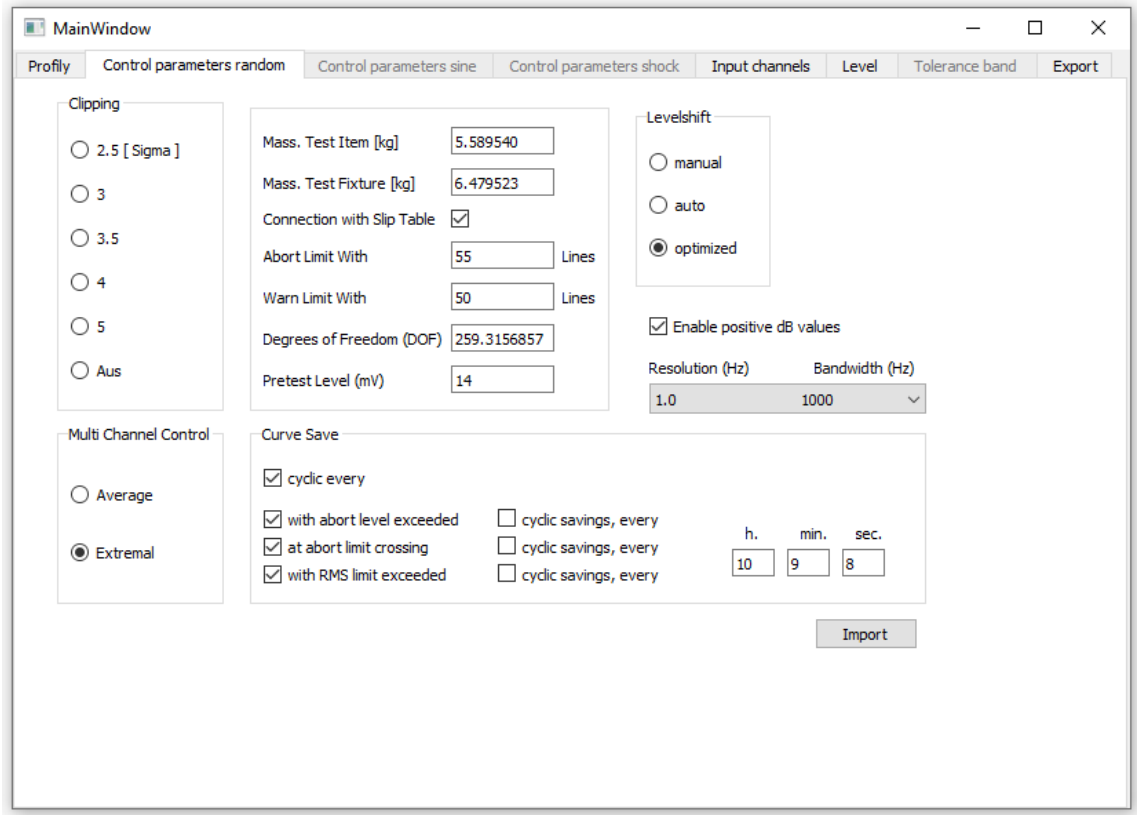
1) Po označení bunky/riadku v hornej tabuľke a kliknutí na „Vybrat“ sa označený profil osamostatní v tabuľke a tým sa pripraví na export do testovacieho súboru.

Ak chceme zahrnúť aj vedľajšie parametre v spodnej tabuľke, musí sa zatrhnúť príslušné políčko ešte pred kliknutím na „Vybrat“. Automaticky je vždy zatrhnuté. Vedľajšie nastavenia (ak boli vybraté) sa následne zapíšu do karty `Control Parameters`. Po výbere profilu užívateľ pokračuje v zadávaní nastavení testu na ďalších kartách. Funkcia „Import“ bude popísaná neskôr.

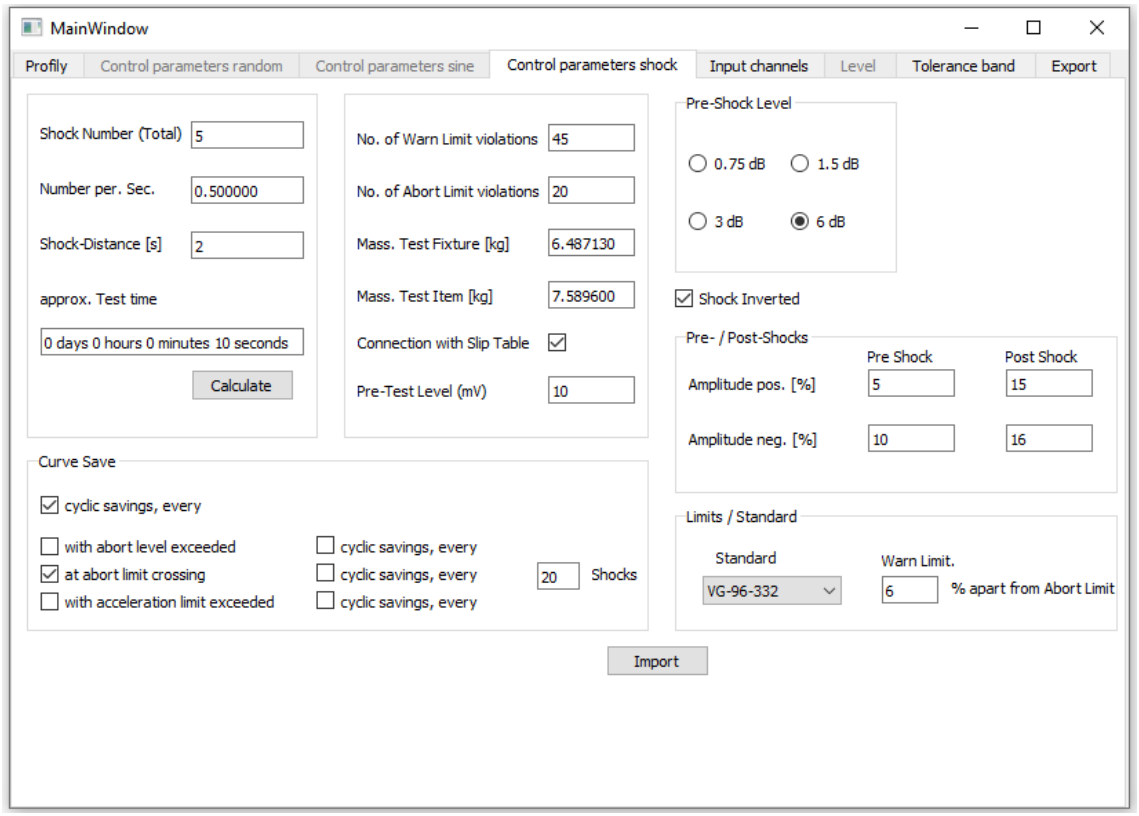
Obrázok 4.6, Obrázok 4.7 a Obrázok 4.8 zachytáva karty, na ktorých sa zadávajú nastavenia `Control parameters`. Užívateľské rozhranie vyzerá rovnako ako v SWR 1200 a sú tu zachované všetky pôvodné prepočty a funkcionalita. Importovanie nastavení prebieha v samostatnom okne po kliknutí na „Import“.



Obrázok 4.6: `Control parameters` - sínus

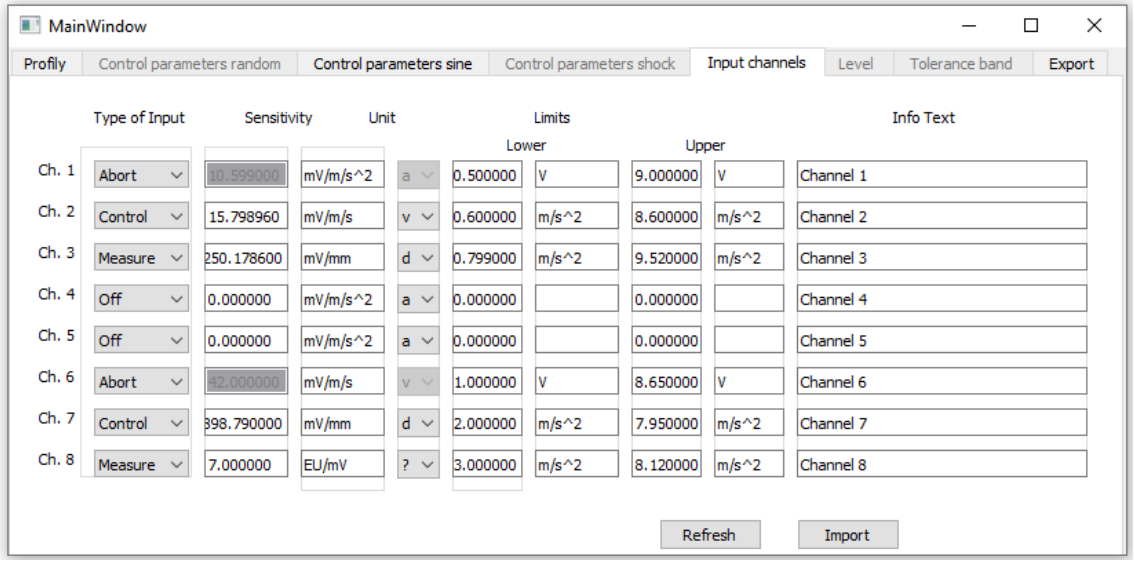


Obrázok 4.7: `Control parameters` - náhodné vibrácie



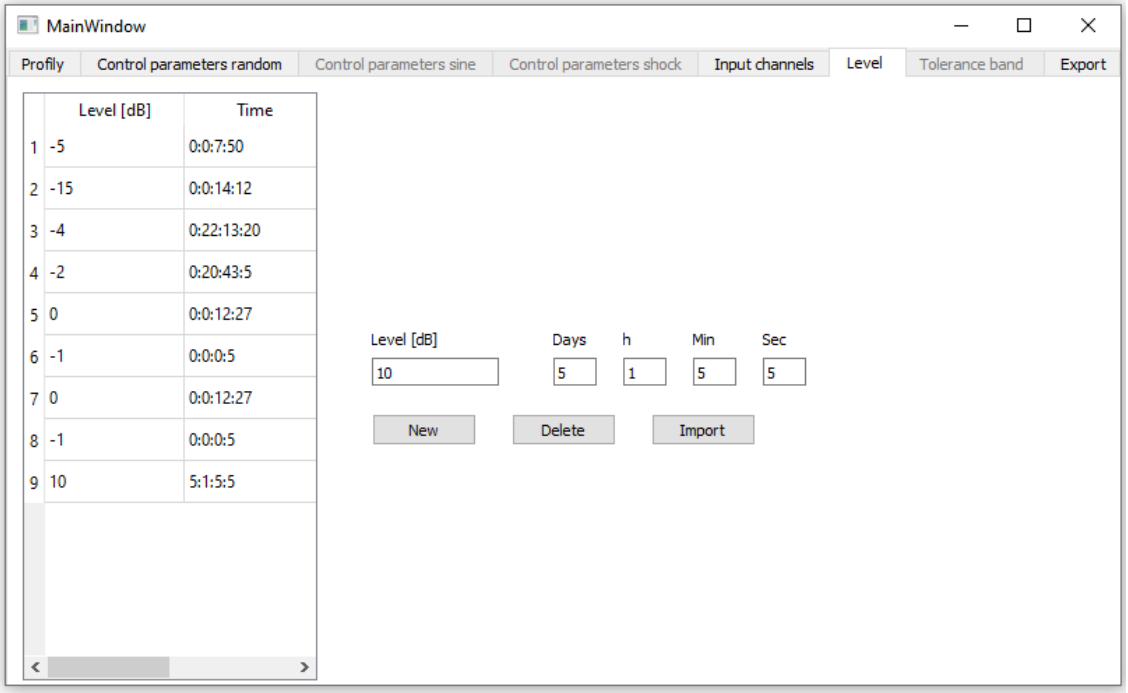
Obrázok 4.8: `Control parameters` - rázy

Obrázok 4.9 zobrazuje kartu nastavení snímačov a vyzerá podobne ako v originálnom programe SWR 1200. „Refresh“ slúži na obnovenie správnych jednotiek v „Unit“ a „Limits“ stĺpcoch podľa typu vstupného kanálu a zvolenej jednotky. Vypĺňanie nastavení je automatizované pomocou „Import“.

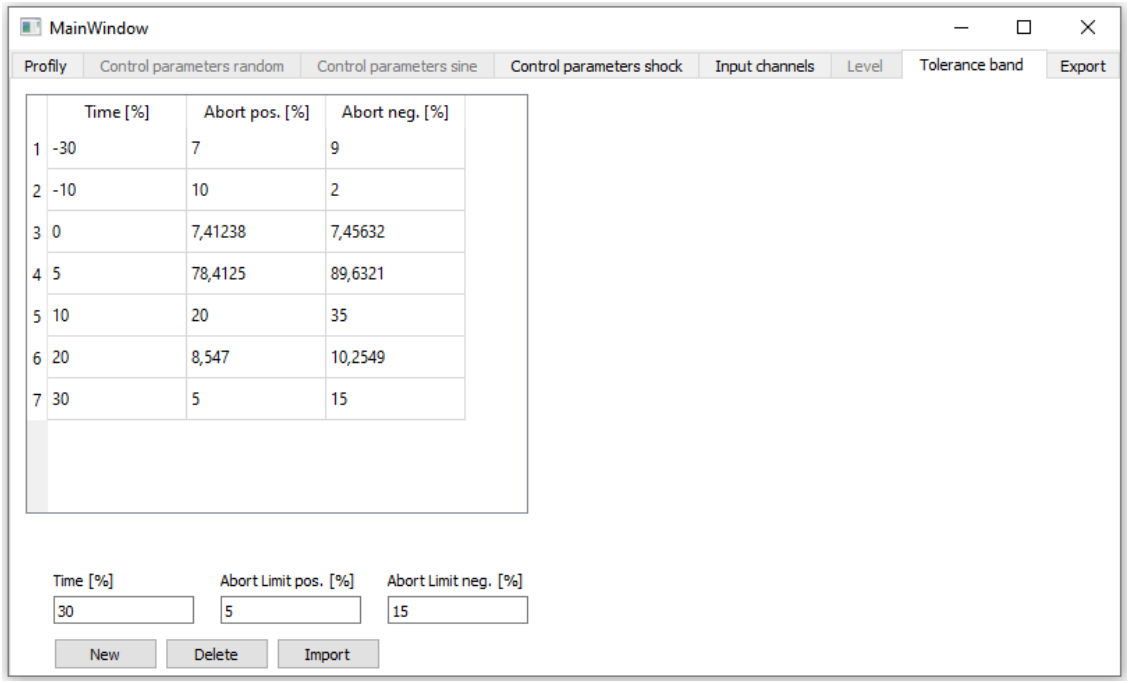


Obrázok 4.9: Karta nastavení snímačov

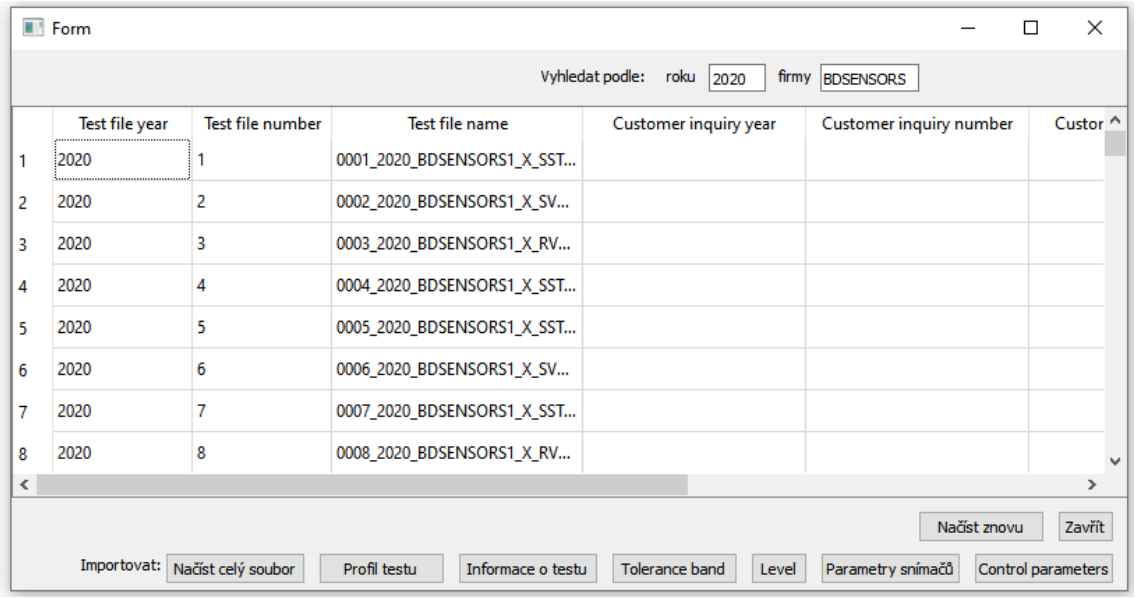
Obrázok 4.10 a Obrázok 4.11 zobrazuje karty nastavení `Tolerance Band` (rázy) a `Levels` (náhodné vibrácie). Implementácia je rovnaká ako v SWR1200, znova tu je možnosť automatického vyplnenia pomocou „Import“.



Obrázok 4.10: Karta `Levels`



Obrázok 4.11: Karta `Tolerance Band`



Obrázok 4.12: Import nastavení

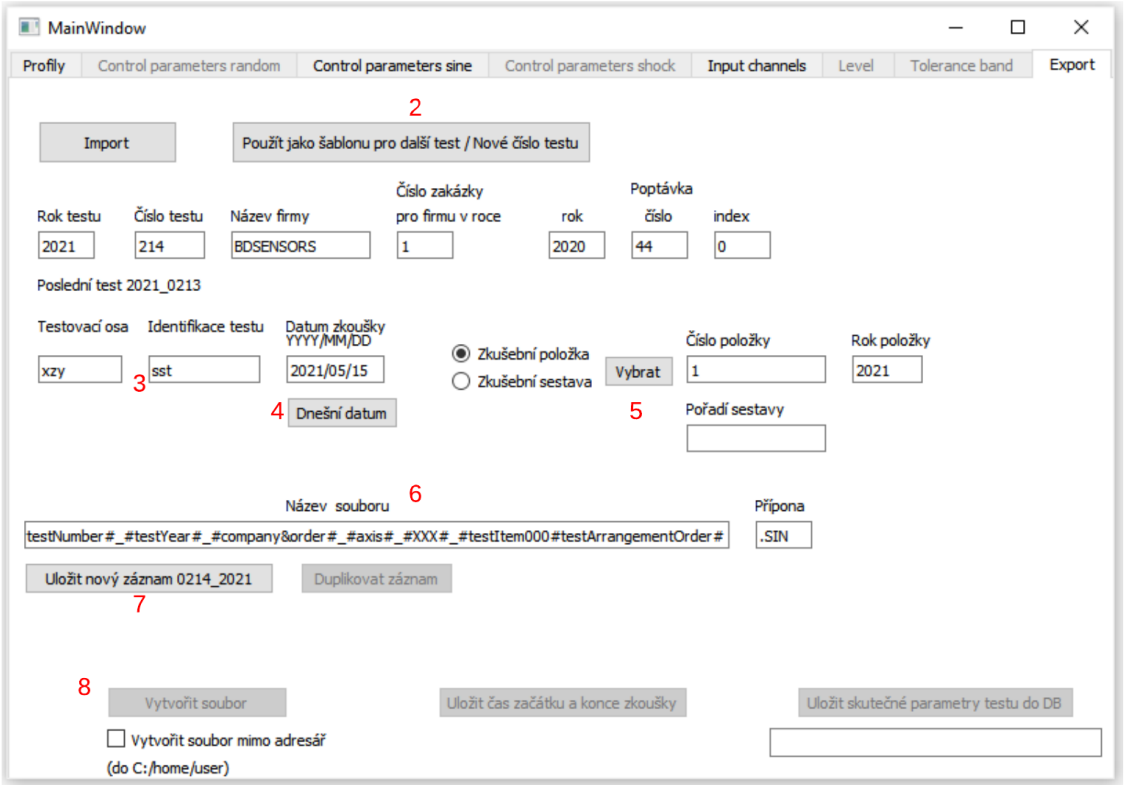
Obrázok 4.12 zobrazuje spomínané okno importu nastavení. Môže sa naň prejsť z každej karty v aplikácii. Po otvorení okna sa automaticky načíta zoznam všetkých testovacích súborov ako prienik tabuliek z databázy testovacích parametrov. Keďže testovacích súborov je veľký počet, sú k dispozícii filtre podľa roku, v ktorom test prebehol a názvu firmy (nájdu sa všetky záznamy, v ktorých meno súboru obsahuje zadaný reťazec). Ak je načítaný existujúci súbor (režim 2), tak sa filtre automaticky vyplnia údajmi (získajú sa z databázy) o danom súbore a aplikujú sa. Ak sa vytvára úplne

nový súbor (režim 1), ktorý ešte nie je v databáze, tak sa filtre doplnia podľa vybraného profilu (karta `Profily`), resp. vezmú sa z tejto karty parametre ponuky alebo zákazky z filtrov a podľa nich sa získa meno firmy (ráta sa s tým, že profily sa pred výberom budú filtrovať). Potom ako sa na Import karte dofiltrovalo, označí sa riadok/bunka a po kliknutí na nejaké zo spodných tlačidiel sa užívateľské rozhranie vyplní nastaveniami z vybraného súboru. V režime 2 sa preferuje načítanie celého súboru.

Po vybratí profilu a zadaní ďalších nastavení (stále sa jedná o popis režimu 1 – tvorba nového súboru) sa prejde na kartu `Export` (Obrázok 4.13), ktorá sa týka informácií o testovacom súbore, ktoré sú mimo samotných nastavení vibračnej skúšky a rieši sa tu jeho vytváranie, uloženie, voľba názvu atď.

Pokračuje sa červeným sledom operácií:

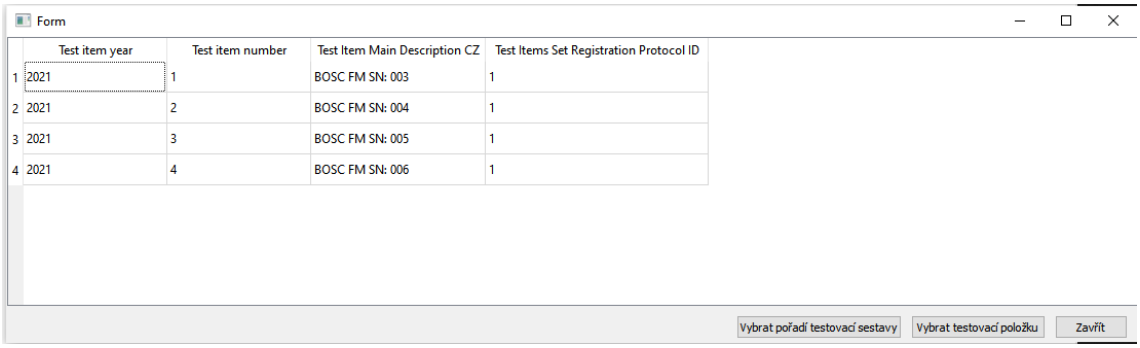
2) Vygeneruje sa nové číslo a rok testu a zapíšu sa do príslušných polí. Číslo vznikne inkrementáciou posledného čísla testu z databázy o 1. Pri roku sa ďalej kontroluje, či posledný rok testu v databáze odpovedá aktuálnemu kalendárnemu, ak nie zapíše sa do poľa aktuálny a číslo testu sa začne počítať od 1. Pod textovými poliami je dynamický text, ktorý zobrazuje vždy posledný uložený záznam v databáze. Po spustení aplikácie je už automaticky predpísané nové číslo a rok testu do príslušných polí.



Obrázok 4.13: Karta `Export`, režim 1

Názov firmy, číslo zákazky a ponuka sú predvyplnené na základe vybraného profilu.

3) Testovacia os a identifikácia testu sa vyplnia užívateľom.
4) Vygeneruje sa aktuálny dátum v danom tvare.
5) Zaškrtne sa buď skúšobná položka alebo zostava a prejde sa na okno ich výberu. V tomto okne (Obrázok 4.14) sa po označení riadku a kliknutí na jedno z tlačidiel zapíšu údaje o testovacej položke/zostave do príslušných polí v `Export` karte.



Obrázok 4.14: Karta testovacej položky/zostavy

6) Užívateľ zadá v akom tvare má byť názov súboru. Tento tvar je vo forme predpísaného makra, ktorého tvar môže užívateľ podľa potreby upraviť. Výhodou použitia makra je jednoduchšia práca s názvom súboru – ak by sa zmenili niektoré časti názvu, napr. testovacia os, tak sa skúšobný technik nemusí zaoberať zmenou názvu súboru, tieto zmeny sa do názvu prenesú automaticky. Tento formát makier sa využíva aj mimo tejto aplikácie napr. pri klimatickej skúške. Podľa tvaru makra sa vytvorí meno súboru, ktoré je už v konkrétnom tvare.
7) Do databázy sa uložia všetky vyplnené nastavenia tohto nového súboru zo všetkých kariet. Týmto je súbor zaevidovaný v databázovom systéme a sprístupní sa možnosť jeho vytvorenia, do tej doby je táto možnosť zablokovaná.
8) Vytvorí sa testovací súbor. Jeho cesta a konkrétny názov sa zložia z informácií na `Export` karte a ich tvar závisí na tom, či k danej ponuke existuje v databáze zákazka alebo nie. Ak sa pred kliknutím zatrhne možnosť dole, vytvorí sa súbor mimo obvyklú adresárovú štruktúru.

Postup pre režim 2 – načítanie existujúceho testu, je nasledovný:

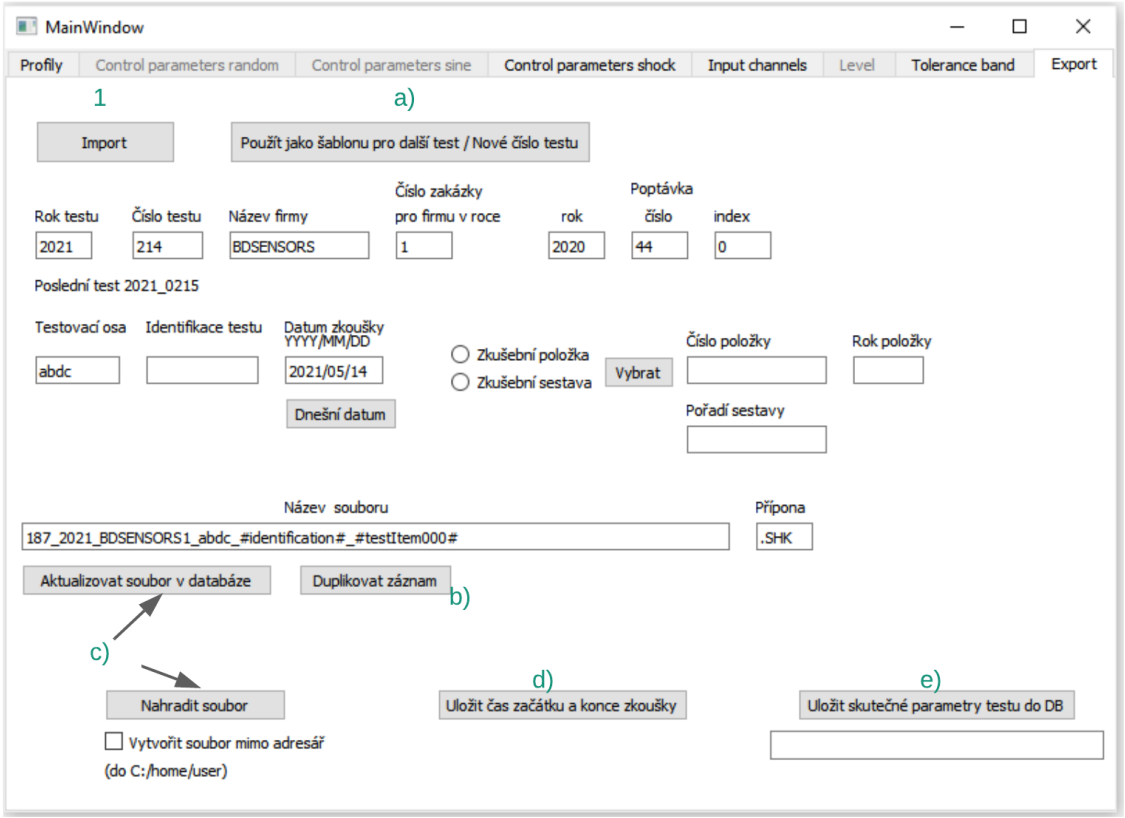
1) Najprv sa súbor načíta z karty Import a na jednotlivých kartách sa môžu ľubovoľne upravovať jeho nastavenia.
2) Po prípadných úpravách je na karte `Export` viac nezávislých možností manipulácie s načítaným súborom:
    a) Môže sa použiť ako šablóna pre nový test (vygeneruje sa mu nové číslo a rok). Ďalej sa postupuje ako v prípade nového súboru, t. j. od 7).

b) Môže sa duplikovať, čo znamená, že sa celý načítaný súbor skopíruje „1:1“ a s novým číslom testu sa zapíše do databázy. Následne sa môže vytvoriť pomocou 8).

c) Upravený načítaný súbor sa môže aktualizovať v databáze (názov tlačidla 7) a jeho funkcia sa zmení na „Aktualizovat...“). Následne sa môže nahradiť starý súbor novým (zmení sa názov tlačidla 8) „Vytvořit…“ a jeho funkcia na „Nahradit...“).

d) Môžu sa získať časové značky začiatku a konca skúšky a zapíšu sa do databázy.

e) Môžu sa získať nastavenia skúšky z testovacieho súboru a ak sa nezhodujú s tým čo je v databáze, prepíšu sa. Textové pole zahlási v akej tabuľke nastala zmena.



Obrázok 4.15: Karta `Export`, režim 2

4.4 Limity a chybové stavy

Limity aplikácie sú napríklad v tom, že nestráži rozsahy pri zadávaní nastavení ako sa to deje v SWR1200. Pre jej hlavný účel to ale nie je vyžadujúce, každopádne sa každý vytvorený súbor nahráva do SWR1200, odkiaľ sa spúšťa meranie a tam sa môžu rozsahy spoľahlivo skontrolovať. Presuny nastavení `Notching` pri sínusových vibráciách sú v knižnici naprogramované, ale v aplikácii to implementované nebolo, pretože sa ich aj

napriek viacerým pokusom nepodarilo programovo meniť. Síce sa tieto nastavenia zapisovali na správne miesta v testovacom súbore, nijak sa to ale vo vytvorenom súbore neprejavilo. Pravdepodobne tam je ďalší skrytý mechanizmus v kóde SWR1200, ktorý nebolo možné iba zo sledovania vstupu-výstupu odhaliť. Na druhú stranu je táto funkcionalita takmer nepoužívaná a po dohode s vedúcim práce sa rozhodlo, že nebude súčasťou výslednej implementácie. Ďalej sa nedá aktualizovať samotné číslo a rok súboru uloženého v databáze. Ak by v tomto nastal problém, musí technik zaviesť zmenu v databáze manuálne.

Časté chybové stavy, ktoré nastávali pri testovaní bolo ukončenie aplikácie pri neoznačení žiadnej bunky/riadku z tabuľky, keď následne užívateľ klikol na nesprávne tlačidlo. Vtedy ostal kontajner s vybranými dátami prázdny a ak sa malo pracovať s jeho obsahom, aplikácia sa kvôli problému s pamäťou ukončila. To je ošetrené podmienkou, v ktorej sa kontroluje, či je kontajner prázdny, a ak áno, nevykoná sa nič.

ZÁVER

V teoretickej časti práce bolo najprv popísané skúšobné laboratórium CVVOZE a používaný softvér firmy RMS, ktorý riadi chod vibračných skúšok. V ďalších kapitolách bola spracovaná dokumentácia testovacích súborov, kde boli popísané 3 typy vibračných skúšok (sínusové, náhodné a pulzy) a ich štruktúra s dôrazom na tabuľky, ktoré sú dôležité pri exporte dát do testovacích súborov. Tieto 3 typy vibrácií tvoria takmer všetky merania vykonávané v laboratóriu CVVOZE, preto bolo výsledné riešenie zamerané hlavne na ne.

V praktickej časti bol najprv definovaný problém pri práci s testovacími súbormi a navrhnuté jeho možné výsledné riešenie. Ďalej boli popísané SQLite databázy, ich výhody a nevýhody a prečo sa zvolil práve tento typ databázy. Vytvorený databázový systém, ktorý pokrýva nastavenia skúšok bol zadokumentovaný a bolo vysvetlené akým spôsobom bude v aplikácii pracovať. Navrhnuté softvérové riešenie je rozdelené do dvoch aplikácií a má zaistiť hlavnú požiadavku na automatizáciu práce s testovacími súbormi, minimalizáciu chyby užívateľa a automatický zápis údajov o vykonanom teste. Najprv bola naprogramovaná statická knižnica v C++, ktorá komunikuje s SQLite databázovým systémom a testovacími súbormi a stará sa o obojsmerný prenos dát medzi nimi. Drvivú väčšinu zadávaných nastavení sa podarilo implementovať s výnimkou nastavení `Notching` u sínusových vibrácií. Táto funkcionalita nie je skúšobným technikom takmer využívaná a nie je súčasťou implementácie. Je to v súlade s tým, na čom sme sa s vedúcim práce dohodli, že bude v aplikácii implementované. Vytvorené funkcie knižnice sú v práci riadne dokumentované, ak by sa mali využívať v iných aplikáciách. Samotné testovacie súbory sú staršieho typu MDB a na prácu s nimi je použitý 32 bitový ODBC ovládač.

Ďalej v rámci praktickej časti boli vytvorené dve samostatné aplikácie naprogramované v jazyku C++ pomocou knižnice Qt určené na platformu Windows. Výhodou C++ je hlavne rýchlosť vykonávania programu a tá sa prejavila najmä pri načítavaní veľkých tabuliek. Prvá aplikácia slúži na zadávanie a správu profilov vibračných skúšok a ich následné priradzovanie ponukám. Druhá aplikácia sa týka samotnej konfigurácie parametrov testovacieho súboru, ich zadávanie je kompletne automatizované s využitím naprogramovanej knižnice a databázového systému. Užívateľské rozhranie na kartách, kde sa zadávajú nastavenia skúšky je vzhľadom približne podobné programu SWR1200. Užívateľ má na nich možnosť využiť nastavení existujúcich súborov uložených v databáze, ktoré sa automaticky vyplnia. Všetka evidencia a manipulácia s testovacími súbormi prebieha na vyhradenej karte `Export`.

Obe aplikácie sú v práci dokumentované hlavne z užívateľského pohľadu, v dokumentácii sú popísané jednotlivé kroky pri prechode oboma rozhraniami a k nim je uvedený popis back-end časti aplikácie. V poslednej kapitole sú uvedené niektoré limitácie a chybové stavy v aplikáciách.

LITERATÚRA

[1] Zkušební laboratoř CVVOZE | Ústav automatizace a měřicí techniky. *Ústav automatizace a měřicí techniky | Vysoké učení technické v Brně* [online]. [cit. 8. 10. 2020]. Dostupné z: https://www.uamt.feec.vutbr.cz/laborator/zkusebni-laborator-cvvoze

[2] RMS Dynamic Testsystems [elektronický dokument]. *TEST MANAGER SWR 1200 Users Manual*. ©2009 [cit. 20. 10. 2020]. Dostupné z: https://rms-testsystems.de/?lang=en

[3] NOVÁK, Adam. *Softwarový interface pro práci s databázovými soubory*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky

[4] OWENS, Michael. *The Definitive Guide to SQLite*. United States of America: Apress, 2006. ISBN 978-1-59059-673-9.

[5] KREIBICH, Jay A. *Using SQLite*. Sebastopol, California: O'Reilly, 2010. ISBN 978-0-596-52118-9.

[6] Appropriate Uses For SQLite. *SQLite Home Page* [online]. [cit. 14. 4. 2021]. Dostupné z: https://sqlite.org/whentouse.html

[7] SQLite. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 15. 4. 2021]. Dostupné z: https://en.wikipedia.org/wiki/SQLite

[8] Introduction. *SQLite Home Page* [online]. [cit. 16. 4. 2021]. Dostupné z: https://sqlite.org/c3ref/intro.html

[9] Database Connection Handle. *SQLite Home Page* [online]. [cit. 19. 4. 2021]. Dostupné z: https://sqlite.org/c3ref/sqlite3.html

[10] Opening a New Database Connection. *SQLite Home Page* [online]. [cit. 19. 4. 2021]. Dostupné z: https://sqlite.org/c3ref/open.html

[11] One-Step Query Execution Interface. *SQLite Home Page* [online]. [cit. 19. 4. 2021]. Dostupné z: https://sqlite.org/c3ref/exec.html

[12] THELIN, Johan. *Foundations of Qt Development*. United States of America: Apress, 2007. ISBN 978-1-59059-831-3.

ZOZNAM PRÍLOH

Příloha A - Obsah CD

A.1 Databázy

Tento adresár obsahuje SQLite databázy, ktoré sú využívané aplikáciou.

A.2 Zdrojový kód knižnice

Nachádzajú sa tu `cpp` a hlavičkové súbory vytvorenej knižnice.

A.3 Zdrojové kódy aplikácií

V týchto priečinkoch sú všetky zdrojové súbory oboch aplikácií.

A.4 Správa profilov

Priečinok prvej aplikácie. Nachádzajú sa tu všetky závislosti a dynamické Qt knižnice potrebné pre spustenie aplikácie.

A.5 Konfigurácia testu

Priečinok druhej aplikácie, nachádza sa tu všetko potrebné pre spustenie aplikácie a jej použitie.

A.6 Readme

Textový súbor, ktorý popisuje čo je treba nastaviť, aby aplikácie bežali a správne fungovali. Týka sa to hlavne prepísania konfiguračného súboru.