Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Analýza použití vestavěných relačních databází na platformě Android

Plzeň 2015 Lukáš Havlíček

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2015

Lukáš Havlíček

# Poděkování

Rád bych poděkoval panu Ing. Martinu Zímovi, Ph.D., za ochotu, vstřícnost, odborné rady a připomínky, které mi pomáhaly při vytváření této bakalářské práce.

# Abstract

Title: Analysis of use of embedded database systems on Android platform

Goal of this bachelor thesis is to do a research of embedded relational database systems which are available for free and evaluate which of these are capable of being deployed on Android platform. Based on discovered facts a database system corresponding with mentioned criterions will be chosen. Next step is to develop a simple application for Android platform using the chosen database system. Application will illustrate basic capabilities, functions and potential restrictions of the database system and Android platform.
Key Words: embedded relational database system, Android platform

# Abstrakt

Název: Analýza použití vestavěných relačních databází na platformě Android

Cílem této bakalářské práce je provést rešerši volně šiřitelných vestavěných relačních databázových systémů a zhodnotit, které je možné reálně použít na platformě Android. Následně na základě zjištěných skutečností bude zvolen databázový systém odpovídající zmíněným kritériím, pro který bude vyvinuta jednoduchá aplikace pro platformu Android. Aplikace bude demonstrovat základní možnosti, funkčnost a případná omezení databázového systému i platformy Android.
Klíčová slova: vestavěnný relační databázový systém, platforma Android

# Obsah

# 1 Úvod

Motivací této práce je skutečnost, že mnoho mobilních aplikací, které pracují se vzdálenou databází, jsou bez připojení k internetu nepoužitelné. Bylo by tedy vhodné, kdyby tyto aplikace měly možnost lokálního uložení databáze, se kterou by následně bylo možné pracovat i v okamžiku, kdy není možnost připojení k internetu.

Cílem práce je provést rešerši volně šiřitelných relačních systémů řízení báze dat, které jsou dostupné ve vestavěné verzi, a prozkoumat, které z nich je možné reálně použít na platformě Android.

Na základě dosažených zjištění bude vybrán databázový systém, pro nějž bude vyvinuta aplikace pro platformu Android demonstrující jeho základní možnosti, funkčnost a případná omezení databázového systému a platformy Android. Výsledná aplikace bude pracovat výhradně bez přístupu k internetu.

# 2 Android

## 2.1 Platforma

Android je operační systém založený na Linuxovém jádře, který je v současnosti vyvíjen firmou Google. Systém byl původně vyvinut pro chytré telefony, následně s příchodem tabletů rozšířil své zaměření na přenosná dotyková zařízení obecně a v současnosti nabízí i specializované distribuce pro televize (Android TV), auta (Android Auto) a hodinky (Android Wear).

Zdrojový kód systému je firmou Google uvolněn pod tzv. open source licencí, což znamená, že zdrojový kód je otevřený, volně dostupný a při dodržení daných podmínek je možné ho využívat zdarma [27]. Výhodou této otevřenosti jsou rozsáhle možnosti úprav, přizpůsobení a využití systému. Součástí licence není samotná ochranná známka, takže výrobci zařízení, která používají Android, jsou povinni k použití systému zakoupit licence. Ovšem díky otevřenosti systému je tato cena velmi nízká, což se následně odráží i na prodejních cenách zařízení s tímto systémem.

## 2.2 Historie

Přestože v současnosti je hlavní tváří operačního systému Android společnost Google, autorem systému je jiná firma. Jedná se o společnost Android, podle které operační systém získal své jméno. Společnost vznikla v roce 2003 v USA a původní plán byl vývoj softwaru pro digitální kamery, který by vylepšil propojení se stolními počítači a podporoval aplikace od třetích stran. Brzy ovšem došlo ke zjištění, že trh s digitálními kamerami není dostatečně velký. Ve stejné době docházelo k rozmachu chytrých telefonů a tak se společnost rozhodla zaměření systému přesunout právě tímto směrem [32].

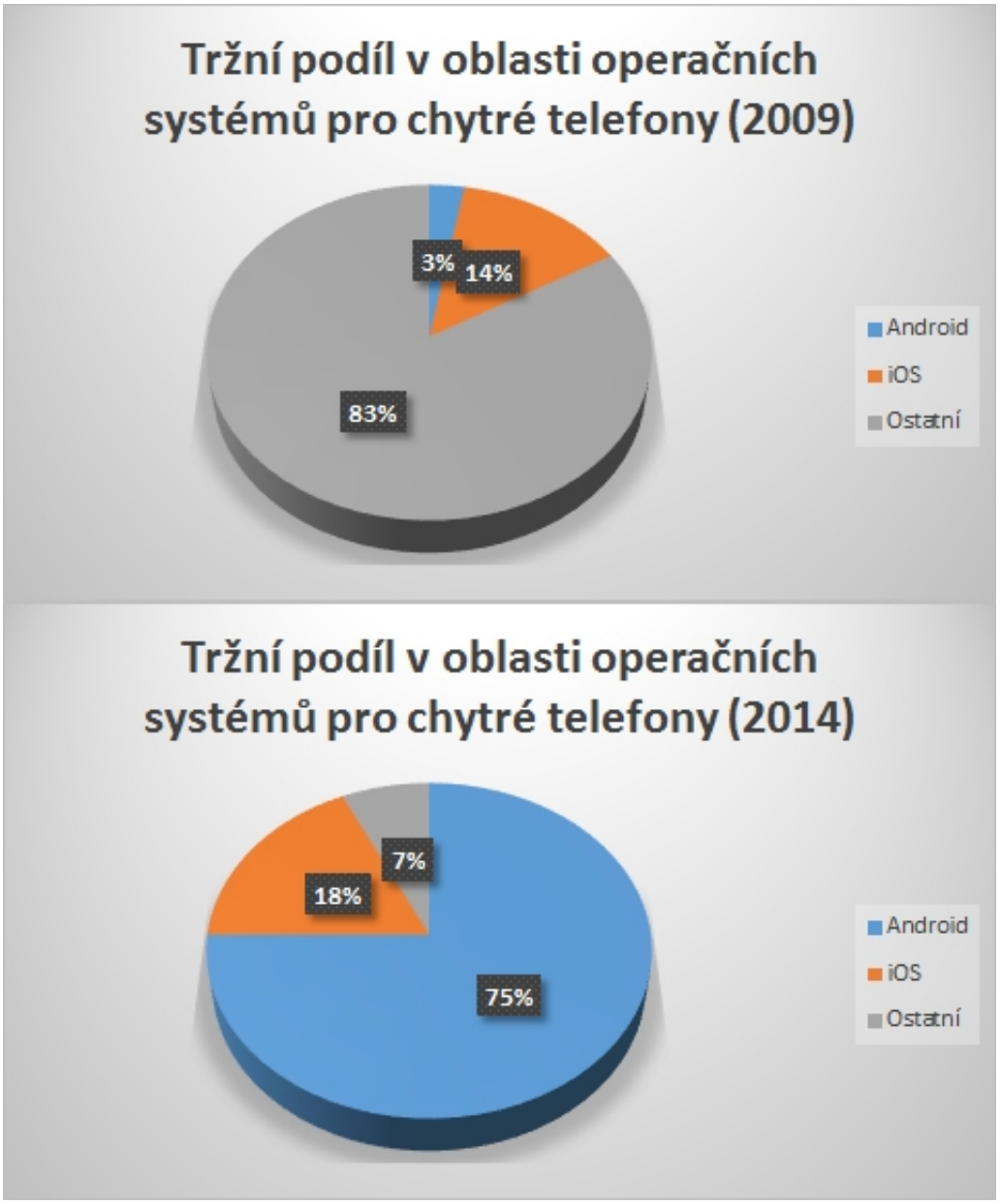
V roce 2005 došlo k odkoupení firmy společností Google, která následně vytvořila tým vedený jedním ze zakladatelů firmy Android, který měl za úkol vytvořit operační systém pro mobilní zařízení. V roce 2007 Google získal několik patentů a v témže roce také došlo k oficiálnímu představení platformy Android. Ve stejný den, kdy došlo k představení platformy, došlo rovněž k představení prvního produktu založeného na této platformě a vzniklo sdru-

žení firem nazvané Open Handset Alliance, které spojovalo mobilní operátory, softwarové společnosti a firmy vyrábějící hardware s cílem vytvořit otevřené standardy pro mobilní zařízení. První oficiální verze systému Android byla uvolněna v roce 2008 [27].

Zajímavostí je, že Android pro označování verzí systému zvolil kódová označení podle názvů zákusků v abecedním pořadí (Donut, Eclair, Froyo, Gingerbread a další) [1].

## 2.3 Podíl na trhu

Operační systém Android během své relativně krátké doby života zcela ovládl trh s chytrými telefony. První oficiální verze systému byla uvolněna v roce 2008, v následujícím roce byl tržní podíl systému na mobilních zařízeních podle průzkumu pouhých 2,8% [33]. Pokud se přesuneme do současnosti, v roce 2014 měl podle průzkumu Android tržní podíl na mobilních zařízeních 75% [34], což je o 25% více, než měl v roce 2009 tehdy nejpoužívanější systém, Symbian. Pro znázornění zde uvedu graf (viz obr. 2.1), který porovná tržní podíly ve zmíněných letech u systému Android a u systému iOS, který si ve sledovaném období také polepšil.




Obrázek 2.1: Porovnání tržního podílu v oblasti operačních systémů pro chytré telefony v letech 2009 a 2014

## 2.4 Architektura

Nejčastěji se architektura operačního systému Android dělí do 4 úrovní (viz obr. 2.2), přičemž na nejnižší úrovni se nachází Linuxové jádro, nad nímž je postavena vrstva s knihovnami a běhovým prostředím, následně aplikační rozhraní a na nejvyšší úrovni jsou aplikace.

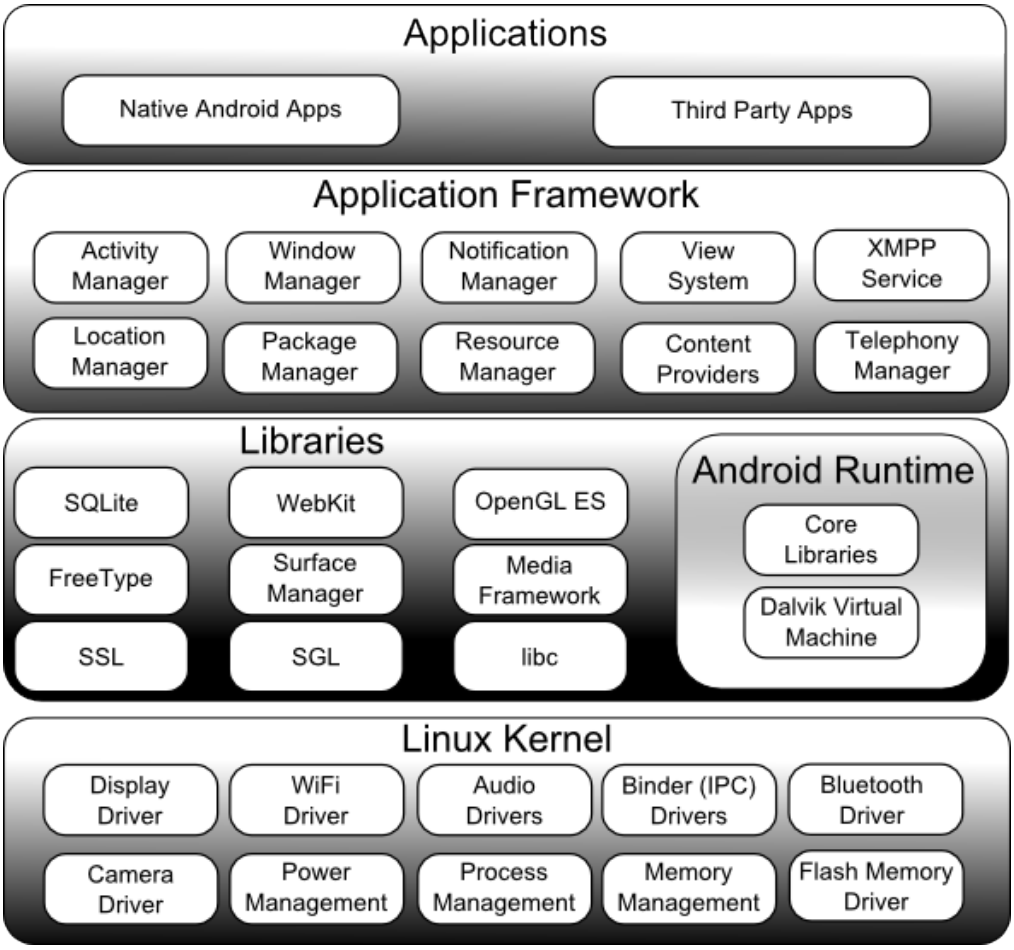
Jádro systému Android má Linuxový základ, ale Google pro potřeby systému jádro upravil a rozšířil o několik funkcí specifických pouze pro Android. Jedná se převážně o funkce pro správu paměti, napájení, komunikaci mezi procesy a mechanismus zámků [28]. I přes tyto rozdíly jsou snahy zakomponovat úpravy jádra do hlavní vývojové větve Linuxu [29] a jeden z autorů Linuxu předpokládá, že postupem času obě větve splynou do jedné [30].

Nad Linuxovým jádrem se nachází vrstva, která se skládá z knihoven a běhového prostředí. Nacházejí se zde volně dostupné podpůrné knihovny, které slouží např. k práci s databázemi (SQLite), zobrazení textu (FreeType) a dalším funkcím. Běhové prostředí se dělí na 2 části, a to na knihovny jádra a Dalvik Virtual Machine (dále jen Dalvik VM). Dalvik VM bude dále rozebrán v sekci 2.7.

Knihovny jádra se dají dále rozdělit na 3 hlavní části – knihovny pro interakci s Dalvik VM, knihovny pro zpracování jazyka Java a C/C++ knihovny. Knihovny pro jazyk Java obsahují pouze podmnožinu Java Standard Edition (dále jen Java SE) knihoven, nenajdeme zde tedy veškeré knihovny, které obsahuje Java standardně. Velkou část knihoven pro jazyk Java tvoří knihovny vyvinuté specificky pro potřeby systému Android. Přestože se může na první pohled zdát, že systém je naprogramován v jazyce Java, ve skutečnosti zde Java a její knihovny plní hlavně funkci rozhraní pro programátory a základ systému tvoří C/C++ knihovny, s kterými Java zprostředkovává komunikaci.

Následující vrstva je aplikační rozhraní, která obsahuje služby pro správu aplikací a jejich podporu. Zároveň poskytuje programátorům přístup k těmto službám.

Poslední vrstvu architektury tvoří již samotné spustitelné aplikace [31].




Obrázek 2.2: Softwarová architektura operačního systému Android [31]

## 2.5 Možnosti úprav

Kolem Androidu postupem času vyrostla rozsáhlá komunita, díky které je pro systém k dispozici velké množství aplikací a dokonce i alternativní systémy postavené na Androidu. Otevřenost se ovšem týká pouze samotného systému, Android je v praxi většinou výrobců zařízení dodáván s předinstalovaným proprietárním softwarem, jehož zdrojový kód je uzavřený a není volně šiřitelný. Přístup společnosti Google k těmto alternativním systémům je pragmatický, zpočátku byly snahy je zakázat, ovšem po nevoli ze strany uživatelů došlo ke kompromisu [26]. Společnost Google tyto alternativní sys-

témy (minimálně některé) uznává za dodržení podmínek, že systémy budou obsahovat pouze Android, nikoliv proprietární software, ovšem výrobci zařízení deklarují ztrátu záruky při jejich instalaci.

Další úpravou, která může zařízení připravit o záruku a která vzešla z komunity kolem Androidu, je tzv. root telefonu. Díky tomu, že je Android postavený na Linuxovém jádře, dá se s ním pomocí různých nástrojů, např. Android Debug Bridge (dále jen ADB) dodávanému spolu s Android Software Development Kit (dále jen Android SDK), pracovat velmi podobně jako s klasickým Linuxem. Z pochopitelných důvodu zde ovšem uživatel nemá právo tzv. superuživatele nebo-li root, což je obdoba administrátora ze systému Windows. Pokud uživatel z nějakého důvodu vyžaduje zvýšená práva, tato úprava mu to umožňuje.

## 2.6 Aplikace

Primárním jazykem pro vývoj aplikací pro platformu Android je jazyk Java upravený pro specifické potřeby platformy. K vývoji je potřeba využít Android SDK, který obsahuje knihovny a nástroje potřebné pro vývoj aplikací v jazyce Java pro tuto platformu.

Aplikace na Androidu neběží přímo v prostředí systému, ani ve virtuálním prostředí Java Virtual Machine (dále jen JVM), jak je zvykem u běžných distribucí Javy, ale specificky pro systém Android byl vyvinut nový virtuální stroj. Hlavní rozdíl v architektuře těchto virtuálních strojů je ten, že JVM využívá zásobníky, zatímco virtuální stroj Androidu využívá registry. Virtuální stroj Androidu by rovněž měl být optimalizován pro nižší požadavky na paměť zařízení.

### 2.6.1 Překlad

Program napsaný v jazyce Java je přeložen do Java bytekódu, který je následně transformován do formátu Dalvik Executable (dále jen dex), což je formát spustitelný virtuálním strojem Androidu. Pro distribuci a instalaci aplikací byl vyvinut formát Android Application Package (dále jen apk), archiv, který obsahuje zdrojový kód aplikace přeložený do formátu dex a pomocné soubory potřebné ke spuštění aplikace [37].

## 2.7 Virtuální stroj

Do verze Android 5.0 byl virtuálním strojem pro Javu na platformě Android Dalvik Virtual Machine (dále jen Dalvik VM), od verze 5.0 je Dalvik VM nahrazen Android Runtime (dále jen ART) [38]. Dalvik VM postupem času získal funkci Just-in-time překladu, což je metoda, kdy se bytekód dex souboru překládá do optimalizovaného strojového kódu ve chvíli, kdy je spuštěna aplikace [39]. Nástupce Dalvik VM, virtální stroj ART, přinesl kromě různých vylepšení, jako např. vylepšený garbage collector, i novou metodu překladu, zvanou Ahead-of-time. Princip této metody spočívá v tom, že překlad probíhá již při instalaci aplikace. Oba virtuální stroje jsou kompatibilní a aplikace vyvíjené pro Dalvik VM by měli fungovat rovněž na ART, ovšem některé techniky, které fungovaly na Dalvik VM, na ART již nefungují [40].

Skutečnost, že společnost Google pro svou platformu Android využívá jako primární programovací jazyk Javu, ale místo standardního virtuálního stroje JVM vyvinula vlastní virtuální stroj, vedla k žalobě ze strany společnosti Oracle. Jazyk Java je sice volně dostupný, ale JVM již nikoliv a práva na něj vlastní právě společnost Oracle. Google by tudíž v případě jeho použití byl nucen platit licenční poplatky, což dle tvrzení společnosti Oracle vedlo k vývoji vlastního virtuálního stroje, který ovšem podle ní porušuje její licenční práva [36].

## 2.8 Aktivity

Každá aplikace obsahuje kromě klasických tříd i tzv. aktivity, které jsou základním stavebním kamenem aplikací pro Android. Aktivita se dá popsat jako zobrazení, které právě vidíme na displeji zařízení při běhu aplikace, a obslužná logika tohoto zobrazení. Pokud aplikace obsahuje více aktivit, jedna z nich musí být určena jako hlavní aktivita, která bude vyvolána po spuštění aplikace. Každá aktivita má buď přiřazený xml soubor s definicí vzhledu, tzv. layout, nebo je tento vzhled nadefinován programově uvnitř aktivity.

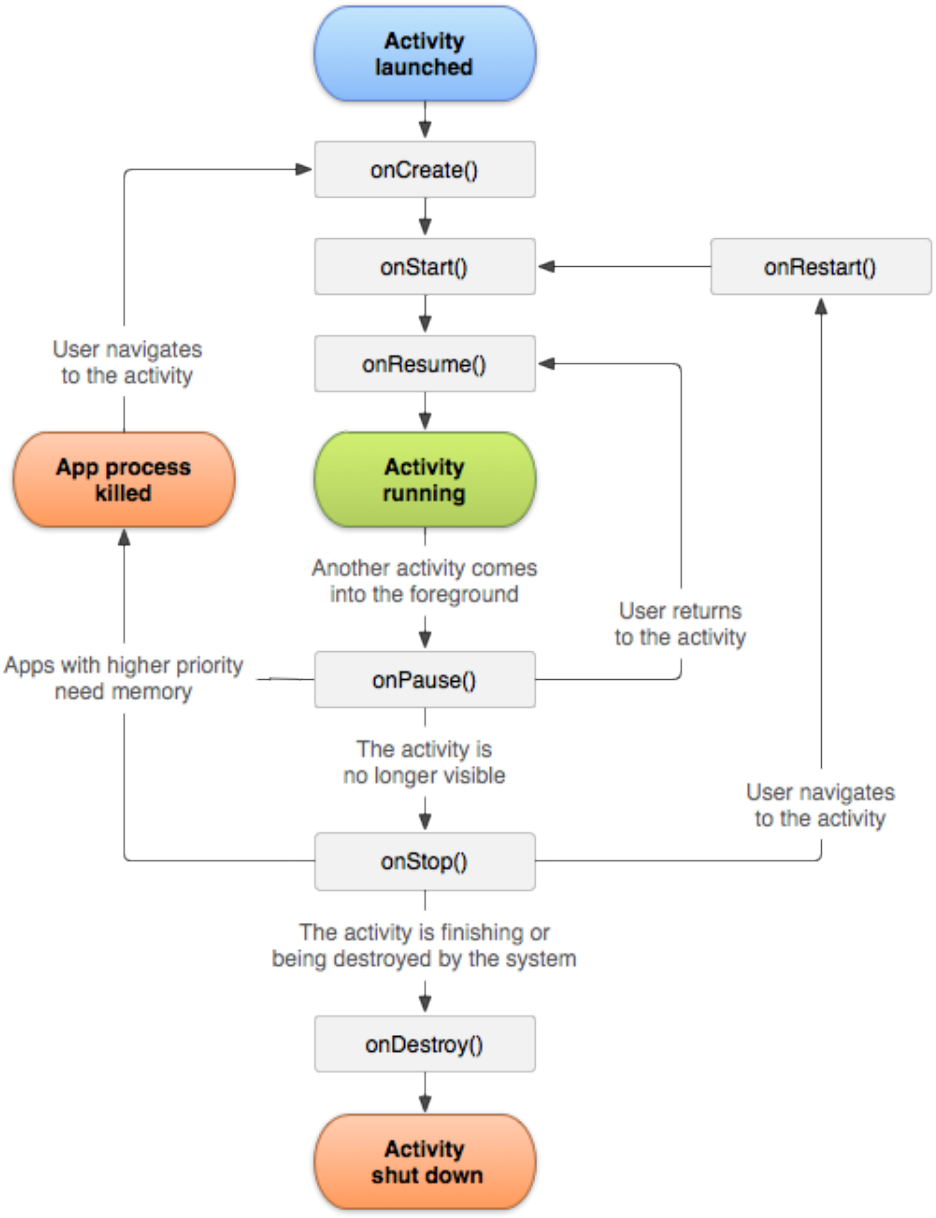
Aktivity fungují jako samostatné komponenty, které mezi sebou ovšem mohou komunikovat a předávat si parametry. Pokud aktivita vyžaduje nějaké zvláštní oprávnění, například přístup k externímu úložišti, tato skutečnost musí být zaznamenána v souboru `AndroidManifest.xml`, který je standardní součástí každé aplikace a obsahuje metadata aplikace, jako jsou oprávnění,

aktivity, jejich struktura, preferované úložiště, název aplikačního balíčku a podobně.

### 2.8.1 Životní cyklus aktivity

Aplikace běžící pod operačním systémem Android nemá kontrolu nad svým životním cyklem a je v tomto směru odkázána na plánovač systému a aktivitu uživatele. Je důležité si uvědomit, že v jednu chvíli může být aktivní jen jedna aktivita a to ta, která je právě zobrazená uživateli. Pokud jsou v systému prováděny nějaké další operace na pozadí, jedná se o služby, nikoliv aktivity. V systému tedy běží jedno hlavní vlákno, které obsluhuje právě zobrazenou aktivitu a předchozí aktivita a všechny ostatní aktivity, které byly spuštěny předtím, jsou buď pozastaveny, nebo v případě vyčerpání paměti zastaveny a vyjmuty z paměti. Operační systém pro tyto účely uchovává aktivity v zásobníku typu Last in, first out, který funguje na stejném principu, jako např. zásobník do zbraně, tedy poslední vložený prvek je jako první vyjmut. Pokud je tedy zavolána jiná aktivita, ta, která je právě v popředí, je pozastavena, vložena na vrchol zásobníku a místo ní je zobrazena volaná aktivita. V případě ukončení aktivity, např. stisknutím tlačítka zpět, je tato aktivita vyjmuta z paměti a je zavolána aktivita nacházející se na vrcholu zásobníku, tedy ta, která byla aktivní předtím. K ukončení aktivity může dojít 2 způsoby - buď je ukončena uživatelem, nebo je ukončena programově.

Vzhledem ke skutečnosti, že aktivita nemá nad svým životním cyklem kontrolu, je žádoucí adekvátně reagovat na různé událostí, které mohou v jejím životním cyklu nastat, např. uložit proměnné při pozastavení nebo uvolnit prostředky při ukončení aktivity. K tomuto účelu jsou systémem prostřednictvím programovacího jazyka poskytnuty mechanismy. Aktivita se v průběhu svého životního cyklu může nacházet v několika fázích (viz obr. 2.3), přičemž průchod těmito fázemi je obousměrný a záleží na konkrétní situaci, která právě nastala [27]. S každou fází životního cyklu se váže odpovídající metoda nesoucí stejné jméno jako fáze cyklu s předložkou „on“. Metoda reagující na stav Resume je tedy např. pojmenována `onResume()`.




Obrázek 2.3: Životní cyklus aktivity [41]

Následuje výběr metod obsluhujících hlavní fáze životního cyklu aplikace: [41]

- `onCreate()` – volána při vytvoření aktivity
- `onResume()` – volána při návratu do aktivity, uživatel s ní v tuto chvíli již může komunikovat
- `onPause()` – volána, pokud je aktivita pozastavena, např. při zavolání jiné aktivity
- `onStop()` – volána, pokud aktivita není viditelná
- `onRestart()` – volána při návratu do aktivity, pokud byla aktivita zastavena (nebyla viditelná) a je potřeba ji obnovit
- `onDestroy()` – volána při ukončení aktivity

Životní cyklus se dá rozdělit na úplný životní cyklus, který zahrnuje všechny fáze mezi *Create* a *Destroy*, a životní cyklus aktivity v popředí, který se odehrává mezi *Resume* a *Pause* a zahrnuje pouze ty fáze cyklu, které jsou uživateli viditelné [27].

Zde bych rád upozornil na skutečnost, že aplikace může v jistých případech disponovat 2 tlačítky s funkcí zpět a obě se chovají jinak. Standardně má každý chytrý telefon vlastní tlačítko s touto funkcí, které při stisknutí v aktivitě zavolá metodu `onResume()` předchozí aktivity v zásobníku, pokud se tam nějaká nachází. Při definici vzhledu aktivity je ovšem možné jí přiřadit i ovládací menu, které je obvykle umístěné v horní části obrazovky, tzv. `ActionBar`. Toto menu má mnoho možností a jednou z nich je zobrazení tlačítka zpět, které po stisknutí zavolá rodičovskou aktivitu definovanou v souboru `AndroidManifest.xml`. Pokud ovšem uživatel využije tohoto tlačítka, nevolá se metoda `onResume()`, ale namísto ní je volána metoda `onCreate()`, nezávisle na tom, zda volaná aktivita již byla vytvořena a nachází se v zásobníku, nebo nikoliv. Jedná se pravděpodobně o ochranný mechanismus, jelikož nikde není určeno, že aktivita, která je v souboru `AndroidManifest.xml` určena jako rodičovská, je rovněž aktivita, přes kterou se do této aktivity dostaneme.

## 2.9 Podpora programovacích jazyků

Základní knihovny jazyka Java obsažené v systému Android se podobají platformě Java SE, jedná se ale pouze o podmnožinu této platformy, nenalezneme zde tedy všechny knihovny obsažené v Java SE a u některých z těch, které se zde vyskytují, jsou jisté rozdíly. Nejpatrnější rozdíl je v absenci Java SE knihoven pro uživatelské rozhraní, které byly nahrazeny vlastními knihovnami [1].

Platforma Android nepodporuje pouze programovací jazyk Java, ale postupem času přibývaly knihovny pro podporu dalších jazyků a v současnosti už je výčet podporovaných jazyků poměrně rozsáhlý.

Následuje výčet podporovaných programovacích jazyků rozdělených podle druhu.

### 2.9.1 Kompilované jazyky

Následuje výčet kompilovaných jazyků, které jsou podporovány platformou Android.

- Java (s využitím Android SDK)
- C, C++ (s využitím Android Native Development Kit - dále jen Android NDK)
- C# (s využitím platformy Xamarin) [35]
- Scala [21]
- RenderScript
- Simple [22]
- Frink [23]

### 2.9.2 Skriptovací jazyky

Následuje výčet skriptovacích jazyků, které jsou podporovány platformou Android s využitím Android Scripting Environment (dále jen ASE)

- perl
- JRuby
- Python
- LUA
- BeanShell

## 2.10 Java knihovny pro práci s databází

Pro práci s databázemi na Java SE platformě slouží knihovna `javax.sql`, zde se ale liší knihovny pro Android a pro Java SE. Android sice obsahuje tuto knihovnu, ale chybí v ní následující rozhraní: [3, 4]

- `XAConnection`
- `XADataSource`

Tato rozhraní poskytují podporu pro distribuované transakce [4]. Z toho se dá usoudit, že absence těchto rozhraní může být překážkou při snaze o konverzi aplikací ze stolních počítačů na platformu Android.

Toto omezení se týká pouze práce se vzdálenými databázemi s využitím knihovny `javax.sql`. Pokud není potřeba využívat vzdáleného přístupu a databáze postačí uložit lokálně, pro tyto účely je v Androidu standardně integrován databázový systém SQLite, který využívají aplikace pro lokální ukládání dat. Knihovny pro práci s databázovým systémem nalezneme v Java knihovnách pro Android v balíčku `android.database.sqlite`.

# 3 Vestavěné databázové systémy

Vestavěné databázové systémy jsou takové systémy řízení báze dat, které jsou integrovány do softwaru, ať už se jedná o operační systém, nebo software konkrétní aplikace. Tyto systémy ke své funkčnosti většinou nevyžadují připojení k internetu a žádnou nebo velmi jednoduchou konfiguraci.

Vestavěných databázových systémů, jak komerčních, tak volně šiřitelných, je velké množství, následující seznam proto obsahuje pouze výběr některých zástupců [5]. Seznam je řazen abecedně.

Advantage Database Server, Amazon SimpleDB, Berkeley DB, CSQL, EffiPRoz, ElevateDB, Empress Embedded Database, Extensible Storage Engine, eXtremeDB , Firebird, HSQLDB, H2 Database, InfinityDB, InnoDB, InterBase, ITTIA DB SQL, Oracle Berkeley DB, Perst, RDM Embedded , solidDB, SQLite, SQL Server Compact, TurboDB, UnQLite, Valentina DB, VistaDB

## 3.1 Vestavěné databázové systémy pro Android

Po analyzování vestavěných databázových systémů z hlediska jejich možného nasazení na platformě Android se počet kandidátů razantně snížil.

Následující výčet obsahuje jak databázové systémy, které jsou již funkční na platformě Android, tak i systémy, na jejichž konverzi se teprve pracuje, nebo je jen teoreticky možná. Systémy jsou rozděleny podle typu na relační, nerelační a objektové. Podle zadání práce se dále budu věnovat jen relačním databázovým systémům.

### 3.1.1 Relační

V relačním databázovém systému jsou data ukládána ve formě tabulek. Tabulka obsahuje buňky, přičemž každá buňka patří určitému řádku a sloupci. Řádek určuje jeden konkrétní záznam tabulky, sloupec určuje jeden z atri-

butů záznamu. Pro zachování integrity by každá tabulka měla mít primární klíč[1], což je sloupec tabulky, který jednoznačně identifikuje záznam. Primární klíč tedy jednoznačně určuje záznam tabulky a zároveň brání tomu, aby byl vložen duplicitní záznam, a umožňuje vytvářet vztahy (relace) mezi tabulkami pomocí tzv. cizích klíčů. Cizí klíč je primární klíč jedné tabulky použitý jako sloupec v jiné tabulce. Díky tomuto je možné se mezi tabulkami odkazovat a uchovávat informace o vzájemných vztazích. Relace mezi tabulkami mohou být jednoduché i mnohonásobné

Následující výčet obsahuje jak relační vestavěné databázové systémy, které jsou již dostupné na platformě Android, tak i systémy, jejichž použití je na této platformě poze teoreticky možné.

- Empress Embedded Database
- Firebird
- H2 Database
- InterBase
- SQLite

### 3.1.2 Nerelační

Nerelační databázové systémy nepoužívají metodu ukládání dat ve formě tabulek, ale nahrazují tuto metodu vlastním způsobem uložení. Pojem nerelační databáze je velmi široký a zahrnuje různé způsoby ukládání dat [42]. Jako nejpoužívanější zástupci těchto databází se dají označit databáze typu klíč-hodnota a dokumentově orientované databáze. V obou případech je princip takový, že libovolné množině záznamů je přiřazen jedinečný klíč. Dokumentově orientované databáze se liší v tom, že jednotlivé záznamy jsou strukturovaná data.

Důvodem pro použití nerelační databáze je především mnohonásobně větší rychlost než u relačních databází, čemuž je ovšem obětována datová konzistence, jelikož data nejsou synchronizována okamžitě [43].

[1]Primární klíč může být tvořen množinou sloupců tabulky.

Nerelační databáze se někdy označují jako NoSQL databáze, což je zkratka pro Not only SQL a odkazuje se na Structured Query Language (dále jen SQL), což je dotazovací jazyk používaný pro práci s daty v relačních databázových systémech.

Následující výčet obsahuje nerelační databázové systémy, které jsou již funkční na platformě Android.

- Amazon SimpleDB
- Oracle Berkeley DB

### 3.1.3 Objektové

Objektové databázové systémy jsou specifický druh nerelačních databázových systémů, kde jsou data reprezentována objekty. Princip je stejný jako v objektově orientovaném programování (dále jen OOP), díky čemuž je zde návaznost na objektově orientované jazyky a objektové databáze využívají všech výhod OOP. V syntaxi zvoleného jazyka jsou vytvořeny objekty, jejich struktura, vlastnosti, objektové vztahy a operace mezi nimi, které jsou následně uloženy v databázi. Dotazování probíhá opět v syntaxi zvoleného jazyka prostřednictvím definovaných operací [44].

Ze zjištěných skutečností vyplynulo, že v současnosti je na platformě Android dostupný jediným objektový databázový systém.

- Perst

# 4 Vestavěné relační databázové systémy pro Android

## 4.1 Empress Embedded Database

Jedná se o komerční aplikaci s možností dočasného vyzkoušení zdarma. Jelikož aplikace není volně šiřitelná, je zde uvedena pouze jako zástupce ve sledované kategorii.

Databázový systém splňuje kritéria Atomicity, Consistency, Isolation, Durability (dále jen ACID), obsahuje Application Programming Interface (dále jen API) pro C/C++, Open Database Connectivity (dále jen ODBC) a Java Database Connectivity (dále jen JDBC), umí šifrovat databázi, podporuje triggery, uložené procedury, referenční integritu a má mnoho dalších funkcí.

V komerční sféře je systém poměrně úspěšný, využívá se například pro předpověď počasí, průzkum vesmíru nebo v jaderných elektrárnách. První verze pochází z roku 1979, první komerční verze z roku 1981 [6].

## 4.2 Firebird

Firebird je volně šiřitelný databázový systém, který běží na systémech Linux, Windows a různých Unixových platformách. Jeho vznik má původ v databázovém systému InterBase. Zdrojový kód je napsán v C/C++ a vývoj probíhá komunitně [7].

Vlastnosti: [7]

- plná podpora pro uložené procedury a triggery (spouště)
- plně kompatibilní transakce ACID
- referenční integrita
- multigenerační architektura
- kompaktní objem serveru

- podpora externích uživatelsky definovaných funkcí
- nástroje třetích stran včetně grafických administračních nástrojů a replikačních nástrojů
- mnoho přístupových metod: nativní API, ODBC, .NET, JDBC ovladač verze 4, modul pro Python, PHP, Perl a další
- inkrementální zálohy

Firebird má zatím pouze částečný port pro Android. Není možné použít vestavěnou verzi databáze ani server, ale lze použít klienta pro přístup ke vzdálené databázi. Jedná se o knihovnu pro C/C++/Delphi/Pascal projekty, tudíž je třeba využít Android NDK [8].

Podle vyjádření jednoho z autorů systému se pracuje na konverzi pro Android [9]. Z interní diskuze mezi programátory podílejícími se na projektu vyplývá, že problematické je hlavně ošetření kritických sekcí a komunikace mezi procesy [10].

Kromě zmíněné konverze klienta existuje řešení i pro Javu. Firebird využívá pro přístup k databázi svůj vlastní JDBC ovladač zvaný Jaybird [11]. Pro ten už rovněž existuje port [12], ovšem neoficiální. Tento port byl navíc naposledy aktualizován 25. 12. 2013 a od té doby je ve fázi betatestu, tudíž se zdá, že na projektu se dále nepracuje a nejspíše by nebylo příliš vhodné se na něj spoléhat v tak citlivé záležitosti, jako je práce s databází.

Jeden z vývojářů Jaybird se vyjádřil v tom smyslu, že není v plánu oficiální port Jaybird pro Android, jelikož by podle jejich zjištění bylo nutné významně přepsat kód kvůli některým knihovnám, které na platformě Android chybí [13].

## 4.3 H2 Database

Volně šiřitelný databázový systém napsaný v jazyce Java. Je dostupný ve vestavěné verzi nebo v klasické verzi klient-server [14].

Podle informací dostupných na oficiálních stránkách by mělo být možné použít systém na platformě Android, přičemž jediné omezení se týká nemožnosti použít *Connection pool*. Proběhlo už dokonce několik testů, které

ukázaly, že čtení z databáze je rychlejší než u SQLite, zápis, otevření a zavření databáze je ovšem pomalejší než u zmíněného konkurenta [15].

Pokusil jsem se tedy o konverzi jar souboru do formátu dex, což je byte kód pro Dalvik VM [1]. K tomuto účelu slouží soubor `dx.bat`, který je standardní součástí Android SDK.

Pro překlad je nutné se v příkazové řádce přesunout do složky obsahující soubor, který bude konvertován, a následně použít příkaz v tomto tvaru: [16]

```
> dx --dex --output="c:\temp\app.dex" "c:\temp\in.jar"
```

Vestavěnou verzi databázového systému získáme tak, že si z oficiálních stránek stáhneme platform independent verzi [14] a spustíme soubor `build.bat` s parametrem `jarSmall` [15]. Tím získáme spustitelný jar soubor.

Po přeložení jar souboru do formátu dex je nutné využít Dalvik VM k jeho spuštění. Jedním ze způsobů je připojit se k němu pomocí konzole a zkopírovat do něj daný soubor [17].

Databázi se mi tímto způsobem bohužel nepodařilo zprovoznit, po připojení do Android shellu a zavolání Dalvik VM program skončil chybou (viz obr. 4.3).

Podle zjištěných informací by mělo být možné databázi použít jako součást Android projektu při programování aplikací, ovšem podle dostupných informací je běh aplikace využívající H2 Database na Androidu řádově až dvakrát pomalejší v porovnání s SQLite při spouštění jednoduchých dotazů [18]. Pravděpodobně se tedy předpokládá využití databázového systému primárně pro rozsáhlé databáze a složité dotazy.

## 4.4 InterBase

InterBase je komerční databázový systém dostupný pro platformy Android, iOS, Windows, OS X, Linux a Solaris. Systém vyžaduje minimální administraci, umožňuje ukládat do cloudu, podporuje šifrování a vývojářské nástroje Java, C, C++, .NET, Delphi, PHP a Ruby [19]. V roce 2000 byl uvolněn zdrojový kód systému, díky čemuž vznikl databázový systém Firebird [20].

IBLite je podle internetových stránek výrobce zdarma dostupný vestavěný databázový systém pro platformy Android, iOS, Windows a OS X. Umožňuje vytváření aplikací využívajících InterBase engine bez nutnosti jej instalovat, stačí jen zahrnout požadované soubory do projektu s aplikací, která bude služeb IBlite využívat.

Tvrzení na stránkách výrobce, že je IBLite volně dostupný, je ovšem zavádějící, jelikož je dostupný pouze jako součást programů RAD Studio XE8 a Delphi XE8 Pro, přičemž oba programy jsou zpoplatněny. První jmenovaný pouze nabízí 30-denní zkušební verzi, takže přinejlepším se dá říci, že IBLite je volně dostupný po dobu 30 dní [19].

## 4.5 SQLite

SQLite je volně šiřitelný databázový systém, který nevyžaduje žádné nastavení nebo instalaci (nepoužívá konfigurační soubory), nevyužívá servery, vyžaduje jen minimální podporu od externích knihoven nebo operačního systému a podporuje full-textové vyhledávání. Transakce implementuje na úrovni *serializable* a dodržuje ACID kritéria. Systém je napsán v jazyce C, z jehož standardních knihoven ale vyžaduje pouze následující funkce: `memset()`, `memcpy()`, `memcmp()`, `strcmp()`, `malloc()`, `free()`, `realloc()`. Kompletní databáze je uložena v jediném souboru na disku, jehož formát je multiplatformní.

SQLite je standardní součástí operačního systému Android [1].

Implementuje většinu z SQL-92 standardu, ale má například jen částečnou podporu triggerů, nemůže zapisovat do pohledů (tuto funkcionalitu ale nabízí INSTEAD OF triggery, které podporuje), má omezenou podporu ALTER TABLE (lze pouze přejmenovat tabulku nebo přidat nový sloupec do již existující tabulky) a nezaručuje doménovou integritu (dá se ovšem vynutit pomocí *constraints*). K databázi může přistupovat několik procesů nebo vláken. Přístup ke čtení může být zpracován paralelně. Přístup k zapisování mohl být dříve proveden jen tehdy, pokud v danou chvíli nebyl v databázi obsluhován jiný přístup. Od verze 3.7 je však možné současně číst i zapisovat.

Možnost současně číst i zapisovat je zde díky funkci Write-Ahead Logging (WAL), kterou je před použitím potřeba zapnout. WAL funguje tak, že vlastně obrátí tradiční transakční postup, takže v tabulce zůstávají původní

hodnoty a nové hodnoty se ukládají do WAL souboru. Jeden tento soubor může obsahovat několik transakcí [24].

Mezi další omezení systému patří, že neobsahuje správu uživatelů a je špatně škálovatelný. Jinak ale najde široké využití, je využíván např. internetovým prohlížečem Firefox, firma Apple ho využila u některých aplikací pro mobilní telefon iPhone a program yum pro Linux byl přepracován k jeho použití [25].

Omezení týkající se zápisu do pohledů, správy uživatelů a škálovatelnosti jsou zde uvedeny pouze informativně, vzhledem k zadání nemají na funkčnost výsledné aplikace žádný vliv.

SQLite nepoužívá klasické statické datové typy, místo toho používá dynamické datové typy v podobě úložných tříd, tzv. *Storage Classes*, a systém typové příbuznosti, tzv. *Type Affinity*.

Úložné třídy plní úlohu optimalizace využití úložného prostoru. Například úložná třída INTEGER navenek funguje stejně jako stejnojmenný datový typ, ale pro ukládání má dostupných 6 různých datových typů různé délky. Po načtení do paměti jsou pak tyto datové typy konvertovány do obecného datového typu, v případě INTEGER se jedná o signed integer délky 8 bytů.

Úložné třídy mohou být typu NULL, INTEGER, REAL, TEXT a Binary Large Object (dále jen BLOB).

Typová příbuznost plní převážně úlohu kompatibility s ostatními databázovými systémy. Pokud je sloupci nastavena typová příbuznost, určuje se tím doporučený datový typ pro daný sloupec. Jedná se ovšem pouze o doporučení, jak již bylo uvedeno výše, SQLite nezaručuje doménovou integritu, takže do sloupce bude i nadále možné vložit jakýkoliv datový typ. Určením typové příbuznosti ale může sloupec upřednostnit specifikovanou úložnou třídou před jinou. Pokud tedy sloupci nastavíme např. typovou příbuznost TEXT, data budou uložena do úložné třídy NULL, TEXT nebo BLOB. V případě, že uživatel zadá číselný vstup, tento vstup je konvertován na text a až posléze uložen do jedné z určených úložných tříd.

Typová příbuznost poskytuje výběr mezi TEXT, NUMERIC, INTEGER, REAL a NONE, přičemž NONE znamená, že nemáme žádný preferovaný typ a nechceme tím pádem využít typovou příbuznost.

Typovou příbuznost neurčujeme sloupci explicitně. Sloupci je definován

klasický datový typ a v závislosti na zvoleném datovém typu je následně vybrána odpovídající typová příslušnost. Mapování datových typů na typovou příbuznost a úložné třídy je uvedeno v tab. 4.1.

| Typová příbuznost | Úložná třída | Datový typ |
|---|---|---|
| INTEGER | INTEGER/REAL/TEXT | INT<br>TINYINT<br>SMALLINT<br>MEDIUMINT<br>BIGINT |
| TEXT | NULL/TEXT/BLOB | CHARACTER<br>VARCHAR<br>NCHAR<br>NVARCHAR<br>TEXT<br>CLOB |
| NONE | BLOB | BLOB |
| REAL | REAL | REAL<br>DOUBLE<br>DOUBLE PRECISION<br>FLOAT |
| NUMERIC | INTEGER/REAL/TEXT | NUMERIC<br>DECIMAL<br>BOOLEAN<br>DATE<br>DATETIME |

Tabulka 4.1: Mapování datových typů na třídy Type Affinity a Storage Classes [24]

Na stránkách projektu je dostupný návod, kdy je vhodné použít SQLite a kdy nikoliv [24].

```
1|root@vbox86tp:/sdcard/Download # dalvikvm -cp h2small-1.3.176.dex h2
dalvikvm -cp h2small-1.3.176.dex h2
Unable to locate class 'h2'
java.lang.NoClassDefFoundError: h2
        at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.ClassNotFoundException: Didn't find class "h2" on path: Dex
PathList[[dex file "dalvik.system.DexFile@5b015990"],nativeLibraryDirectories=[/
system/lib]]
        at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:56
)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:497)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:457)
        ... 1 more
java.lang.NoClassDefFoundError: h2
        at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.ClassNotFoundException: Didn't find class "h2" on path: Dex
PathList[[dex file "dalvik.system.DexFile@5b015990"],nativeLibraryDirectories=[/
system/lib]]
        at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:56
)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:497)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:457)
        ... 1 more
```

Obrázek 4.1: Chybová hláška při pokusu o zkopírování souboru do Dalvik VM.

# 5 Shrnutí analýzy

Ze zjištěných skutečností vyplývá, že ideálním a v současné době bohužel i téměř jediným kandidátem na volně šiřitelný vestavěný relační databázový systém pro operační systém Android je SQLite.

Analýza vestavěných relačních databázových systémů s potenciálem fungování na platformě Android ukázala, že v současnosti jsou k dispozici 2 funkční systémy, které ale nesplňují kritérium volného šíření, jelikož jsou vyvíjeny pod komerční licencí (InterBase a Empress Embedded Database), dále systém, který je volně dostupný i funkční (SQLite) a nakonec systémy, které jsou volně dostupné, ale na jejich konverzi se teprve pracuje a zatím nejsou pro platformu Android k dispozici (Firebird a H2 Database). V posledně jmenované skupině je zástupce (H2 Database), kterého by dle dostupných zjištění již mělo být možné použít jako součást projektu aplikace vyvíjené pro Android, ale tento databázový systém zatím není dostatečně optimalizovaný a dosahuje horších výkonnostních výsledků, než databázový systém SQLite, který je standardní součástí operačního systému Android.

Po důkladném zhodnocení všech zjištěných skutečností jsem se tedy rozhodl zvolit databázový systém SQLite jako systém využívaný vytvořenou aplikací.

# 6 Návrh aplikace

Aplikace vyvinutá pro potřeby tohoto zadání splňuje následující parametry:

- Aplikace pracuje s databázovým systémem SQLite a demonstruje jeho základní možnosti a funkce
- Vývoj je zaměřen čistě na platformu Android
- Aplikace musí pracovat výhradně bez přístupu k internetu

Na základě výsledné aplikace dojde ke zhodnocení možností využití a omezení databázového systému SQLite na platformě Android. Omezení se týkají jak samotného databázového systému, tak operačního systému, pro který je aplikace vyvíjena.

Jelikož je databázový systém SQLite standardní součástí každé distribuce operačního systému Android, není třeba knihovny systému dodatečně integrovat do projektu vyvíjené aplikace. K API databázového systému se dostaneme pouhou implementací systémové knihovny *android.database.sqlite*.

Vývoj aplikace probíhal ve vývojovém prostředí Android Studio, které bylo v době práce na aplikaci jediným oficiálně podporovaným vývojovým prostředím pro platformu Android. Většina testování aplikace probíhala na virtuálních zařízeních s pomocí nástroje Android Virtual Device (dále jen AVD) Manager, který je integrovaný do vývojového prostředí Android Studio. Použil jsem různá virtuální zařízení s verzemi SDK 19 a 21. Finální testování aplikace probíhalo na fyzickém zařízení LG L90 s verzí operačního systému 4.4.2 a Android SDK ve verzi 19.

Jako název aplikace jsem zvolil „SimpleDBManager“, jelikož tento název podle mého názoru dostatečně vystihuje zaměření a rozsah aplikace.

## 6.1 Omezení

Operační systém Android je dynamický a neustále prochází vývojem a spolu s ním i Android SDK. V době psaní této práce je aktuální verzí Android

SDK verze 22 s kódovým označením Lollipop MR1. Novější SDK s sebou samozřejmě často přinášejí změny a nové funkce, které ve starších SDK nebyly dostupné. Při vytváření aplikace pro tuto platformu je tedy nutné udělat rozhodnutí, jaká verze Android SDK bude zvolená jako nejnižší požadovaná k instalaci. Tento parametr je následně uveden v souboru `AndroidManifest.xml`, který uchovává metadata o aplikaci. Jedná se o parametr nepovinný, není tedy nutné zvolit minimální podporovanou verzi Android SDK, v takovém případě se ale implicitně uvažuje nejnižší možná verze a nebude možné využít jakýchkoliv úprav a novinek v novějších verzích. U některých elementů týkajících se vzhledu aplikace je dále možné v xml souborech specifikovat, jak se bude daný element chovat v závislosti na verzi Android SDK. Pokud uvedu příklad z této aplikace, tak např. od Android SDK verze 16 (odpovídá verzi operačního systému 4.1 s označením Jelly Bean) se v celé aplikaci používá styl písma zvaný *Roboto*. Funkčnost aplikace na nižších verzích operačního systému tím není nijak omezena, aplikace pouze bude používat standardní styl písma. Kromě nejnižší požadované verze SDK je možné specifikovat i cílovou verzi a maximální verzi SDK, přičemž v případě neuvedení těchto hodnot se za cílovou verzi považuje minimální verze a maximální verze není nijak omezena.

### 6.1.1 Minimální verze Android SDK

Při hledání optima mezi kompatibilitou pro co největší rozsah zařízení a dostatečně odladěným systémem, který není zastaralý a poskytuje dostatek možností, jsem se rozhodl, že aplikace bude mít jako minimální verzi Android SDK nutnou k instalaci určenu verzi 14. Tato verze zajišťuje, že rozsah zařízení, na kterých bude možné aplikaci nainstalovat, bude dostatečně velký a zároveň nebude hrozit, že aplikace se stane brzy zastaralou nebo se v případě jejího dalšího vývoje narazí na nedostatečné možnosti daného SDK. Minimální verze Android SDK nastavená na hodnotu 14 znamená, že aplikaci budou moci nainstalovat všichni uživatelé s verzí operačního systému Android 4.0 a vyšší. Jak je patrné ze statistik dostupných na oficiálních stránkách Androidu, takto zvolená minimální hodnota pokrývá drtivou většinu provozovaných zařízení, v současnosti více než 93%.

### 6.1.2 Cílová verze Android SDK

Kromě minimální verze Android SDK je v aplikaci stanovena i cílová verze SDK, která udává, že aplikace byla vyvíjená a testována na této verzi a říká systému, že v případě, že běží na verzi operačního systému spojené s tímto SDK, není nutné aktivovat žádné mechanismy pro zajištění kompatibility. Při rozhodování o cílové verzi SDK hrály roli 2 faktory, které spolu naštěstí korespondovaly a rozhodování tím pádem bylo velmi snadné. Prvním faktorem byla možnost otestovat výslednou aplikaci na reálném zařízení. Po většinu vývoje probíhalo testování na virtuálním systému, ovšem reálná zařízení se v určitých ohledech od těch virtuálních liší. Druhým faktorem byl opět co největší rozsah podporovaných zařízení. Chytrý telefon, který v době práce na aplikaci vlastním, běží na verzi operačního systému 4.4 s označením Kitkat, která odpovídá Android SDK verzi 19. Shodou okolností je tato verze podle oficiálních statistik v současnosti nejrozšířenější verzí s více než 41% podílem na zařízeních provozujících systém Android. Jako cílová verze Android SDK tedy byla zvolena verze 19 [45][46].

### 6.1.3 Externí úložiště

Pokud zařízení, na kterém je provozována aplikace, nedisponuje externím úložištěm, aplikace tím přichází o 2 své funkce, a sice import a export. Důvodem je to, že při odinstalování aplikace jsou smazána veškerá data, která si aplikace uložila na lokální úložiště. Zároveň práva čtení a zápisu v interním úložišti mimo aplikační prostor jsou vesměs nulová. Není tedy možné číst databáze z libovolné složky na interním úložišti a ani ukládat mimo aplikační prostor. I pokud by bylo ukládání možné kamkoliv, bylo by to zbytečné, jelikož možnost exportu je zde hlavně z důvodu zálohy a takováto záloha by byla zbytečná, protože po odinstalování aplikace by stejně zmizela spolu s ní [49].

Externí úložiště má také jistá omezení ohledně možnosti zápisu, nelze zapisovat od určitých složek vyhrazených pro konkrétní účely nebo aplikace, ale pokud tomu nebrání nějaké další nastavení, lze bez problému vytvořit složku v kořenovém adresáři externího uložiště a tu využívat pro import a export databází.

# 7 Architektura aplikace

Aplikace pro Android se skládají z různých částí, které mezi sebou mají různé vztahy. Obecně se dají soubory v aplikaci rozdělit do následujících tříd:

- Android Manifest
- Aktivity
- Třídy
- XML soubory
- Moduly

V případě xml souborů se tato třída dá rozdělit na další podtřídy, což bude znázorněno dále, zde se jedná o globální rozdělení. Následuje popis jednotlivých tříd z obecného hlediska a jejich konkrétní zpracování v aplikaci.

## 7.1 Android Manifest

Android Manifest je specifický druh xml souboru. Jak je uvedeno v kapitole 6.1, jde o soubor obsahující metadata o aplikaci. Určuje se zde např. minimální, cílová a maximální verze Android SDK, práva, která aplikace vyžaduje pro svůj běh, název aplikačního balíčku (standardem je uvádět ve tvaru „zkratka země.společnost.název aplikace“) [58], preferované úložiště, jednotlivým aktivitám se dá přiřadit název, popisek zobrazený při běhu, rodičovská aktivita a další věci. Jedná se o velmi důležitou část celé aplikace a v případě chybného nastavení může zamezit jejímu fungování.

### 7.1.1 Verze Android SDK

V aplikaci jsem využil možnosti nastavení minimální a cílové verze Android SDK. Minimální verze SDK nutná pro instalaci aplikace byla nastavena na 14, cílová verze na 19. Tyto hodnoty by dle oficiálních statistik měly zajistit maximální míru kompatibility při zachování dostatečného množství funkcí.

### 7.1.2 Preferované úložiště

Aplikaci je také možné nastavit preferované úložiště pro instalaci. Na výběr je mezi interním úložištěm, externím úložištěm a automatickou volbou. V případě volby interního úložiště nebude po instalaci možné aplikaci přesunout na externí úložiště. Pokud je zvoleno externí úložiště, aplikace sem bude nainstalována pouze v případě, že je tato možnost povolena v operačním systému a dané zařízení disponuje externím úložištěm. V případě automatické volby se aplikace standardně instaluje na interní úložiště, ovšem v případě, že tam již není dostatek místa a zároveň zařízení disponuje externím úložištěm a v systému je povolena možnost instalace na externí úložiště, aplikace je nainstalována sem. Aplikace se implicitně instalují na interní úložiště zařízení a pokud uživatel toto nastavení nezmění, operační systém u většiny zařízení implicitně nedovoluje výběr externího úložiště [48].

Pro aplikaci jsem zvolil automatickou volbu úložiště, jelikož z vlastní zkušenosti vím, že mnoho zařízení s operačním systémem Android má problém s malým interním úložištěm. Velkou část aplikací lze navíc instalovat pouze na interní úložiště, pravděpodobně proto, že pokud vývojář nezvolí explicitně jinou možnost, aplikace se automaticky instalují do interního úložiště. Tímto se problém s interním úložištěm ještě prohlubuje a pokud bych tento atribut v souboru `AndroidManifest.xml` neuvedl, nebo zvolil možnost interního úložiště, v případě, že by bylo plné, aplikaci by nebylo možné nainstalovat ani v případě, že na externím úložišti by byl dostatek místa.

### 7.1.3 Oprávnění

Pokud aplikace potřebuje během svého běhu provádět operace se zvýšeným oprávněním, nebo pokud k této potřebě pouze může v určitých případech dojít, je potřeba tato oprávnění uvést v souboru `AndroidManifest.xml`. Mezi zvýšená oprávnění patří různá volání systémových služeb, jako zjištění stavu zařízení nebo přístup k SMS zprávám a hovorům, ale i například zapisování do souborů. Tato oprávnění jsou při instalaci aplikace zobrazena uživateli a ten se na základě toho může rozhodnout, zda aplikaci tato práva udělí, nebo instalaci raději zruší.

Aplikace vyžaduje při instalaci udělení pouze 2 práv, a sice čtení stavu zařízení, které slouží ke zjištění, zda zařízení disponuje externím úložištěm, a právo zápisu na externí zařízení, které slouží k zálohování databáze, což

bude vysvětleno dále.

### 7.1.4 Aktivity, moduly

Dále `AndroidManifest.xml` obsahuje výčet všech aktivit, jejich název, popisek a rodičovskou aktivitu. Pokud projekt obsahuje externí knihovny v podobě modulů, je potřeba zde tyto moduly též uvést jako aktivity. V opačném případě o nich aplikace nebude vědět a nebude možné se na ně odkázat. V aplikaci je využita jedna externí knihovna, která bude popsána dále.

### 7.1.5 Název aplikace a balíčku

Dalším podstatným atributem je název aplikace a aplikační balíček. Název aplikace jsem zvolil takový, aby pokud možno odpovídal zadání práce, rozhodl jsem se tedy pro tento: „Simple DB Manager“. Pod tímto názvem je aplikace zobrazena v systému po její instalaci. Název aplikačního balíčku jsem zvolil podle zaběhnutých standardů, takže vypadá následovně: `cs.zcu.simpledbmanager`.

Zde bych rád podotkl, že změna aplikačního balíčku u projektu vytvořeného v programu Android Studio se ukázala jako poměrně složitá operace. Nejdříve je nutné pomocí funkce zvané Refactoring přejmenovat všechny složky v projektu. Při přejmenování je na výběr mezi přejmenováním složky nebo balíčku, zde je nutné zvolit přejmenování balíčku. Pro zobrazení skrytých složek je potřeba kliknout na ikonu ozubeného kola nad oknem s projektem a zrušit možnost „Compact Empty Middle Packages“. Dále je nutné v souboru `AndroidManifest.xml` přepsat atribut `package` na požadovaný název. Ovšem ani po splnění těchto kroků stále není balíček přejmenován. Android Studio pro sestavení a překlad projektů používá nástroj na automatizaci zvaný *Gradle*. Pro změnu názvu balíčku je nutné otevřít soubor `build.gradle`, který se nachází ve složce Gradle Scripts. Pokud má projekt více modulů, nachází se zde několik souborů s tímto názvem. K přejmenování balíčku je třeba otevřít ten soubor, který má za svým názvem závorku a v ní text v tomto tvaru: „Module: název složky s hlavním modulem“. Standardně je název složky s hlavním modulem app. V souboru se nachází sekce `defaultConfig` a v ní atribut `applicationId`. Zde je opět nutné přejmenovat atribut na požadovaný název a až po splnění tohoto kroku a překompilování projektu je změna názvu balíčku kompletní.

Uvádím zde Android Studio z toho důvodu, že v době psaní této práce je jediným oficiálně podporovaným vývojovým prostředím pro aplikace cílené pro Android. Do nedávné doby zde byla alternativa v podobě vývojového prostředí Eclipse s nástrojem Android Developer Tools (dále jen ADT), ovšem nyní již Eclipse ADT není oficiálně dostupný ke stažení a jeho používání se nedoporučuje, jelikož byl ukončen vývoj nástroje ADT a vývojáři by tím pádem riskovali zastaralost vývojového prostředí. Od jeho užívání odrazuje i sama společnost Google na oficiálních stránkách Androidu [47].

## 7.2 Aktivity

Aktivita je specifický druh třídy, který zobrazuje grafický výstup, zpracovává vstupy a události a řídí logiku zobrazeného výstupu. Důležitou vlastností aktivity je, že v jednu chvílí může být v systému aktivní pouze jedna aktivita. Pokud bych měl provést přirovnání k některé části standardu softwarové architektury Model-View-Controller (dále jen MVC), byl by to Controller. Stejně jako v případě Controlleru v architektuře MVC spočívá hlavní úloha aktivity v tom, aby zpracovala logiku aplikace a výsledek zobrazila uživateli. Jde tedy o jakéhosi prostředníka mezi logikou aplikace a grafickým uživatelským rozhraním (dále jen GUI). Stejně jako v MVC standardu zde ovšem tato role není vynucena, jedná se pouze o doporučení, je tedy bez problému možné v aktivitě zpracovávat logiku aplikace a zároveň grafický výstup. Pokud se tomu dá ovšem vyhnout, doporučuje se jednotlivé vrstvy oddělit. Detailněji jsem se aktivitám věnoval v kapitole 2.8.

Aktivita se řídí svým životním cyklem, přičemž minimálně metodu `onCreate()` je nutné v drtivé většině případů definovat. V aktivitě je možné vyvolávat dialogová okna a přiřadit jí různé druhy menu a panelů, např. kontextové menu nebo horní panel zvaný `ActionBar`. Přidané prvky mohou obsahovat další ovládací prvky, vše je ovšem stále svázáno s danou aktivitou, ve které se také provádí obsluha těchto prvků. Grafický výstup aplikace je možné definovat 2 způsoby, programově a pomocí xml souboru. Tyto 2 způsoby lze libovolně kombinovat, přičemž platí, že programově nastavené atributy mají vyšší váhu a přepisují atributy definované v xml souboru. Nastavení grafického rozhraní se netýká pouze aktivity jako celku, ale týká se i některých dílčích částí, jako např. dialogového okna nebo horního panelu.

Aktivity se mohou navzájem volat a předávat si parametry. K tomuto účelu je zde třída zvaná `Intent`, nebo-li záměr. V záměru definujeme, jakou

aktivitu chceme volat, a v případě potřeby lze specifikovat další parametry, k čemuž slouží tzv. `Bundle`. `Bundle` uchovává předávané parametry a ve volané aktivitě z něj lze následně tyto parametry přečíst. Předávání parametrů probíhá metodou klíč-hodnota, přičemž klíč je reprezentován textovým řetězcem. Aktivita může být volána běžným způsobem, nebo lze určit, že od dané aktivity očekáváme výsledek, na který budeme v pozastavené aktivitě čekat a který bude vrácen v číselné reprezentaci pomocí tzv. `resultCode`. V takovém případě je možné aktivitě určit i tzv. `requestCode`, který umožňuje identifikovat volající aktivitu.

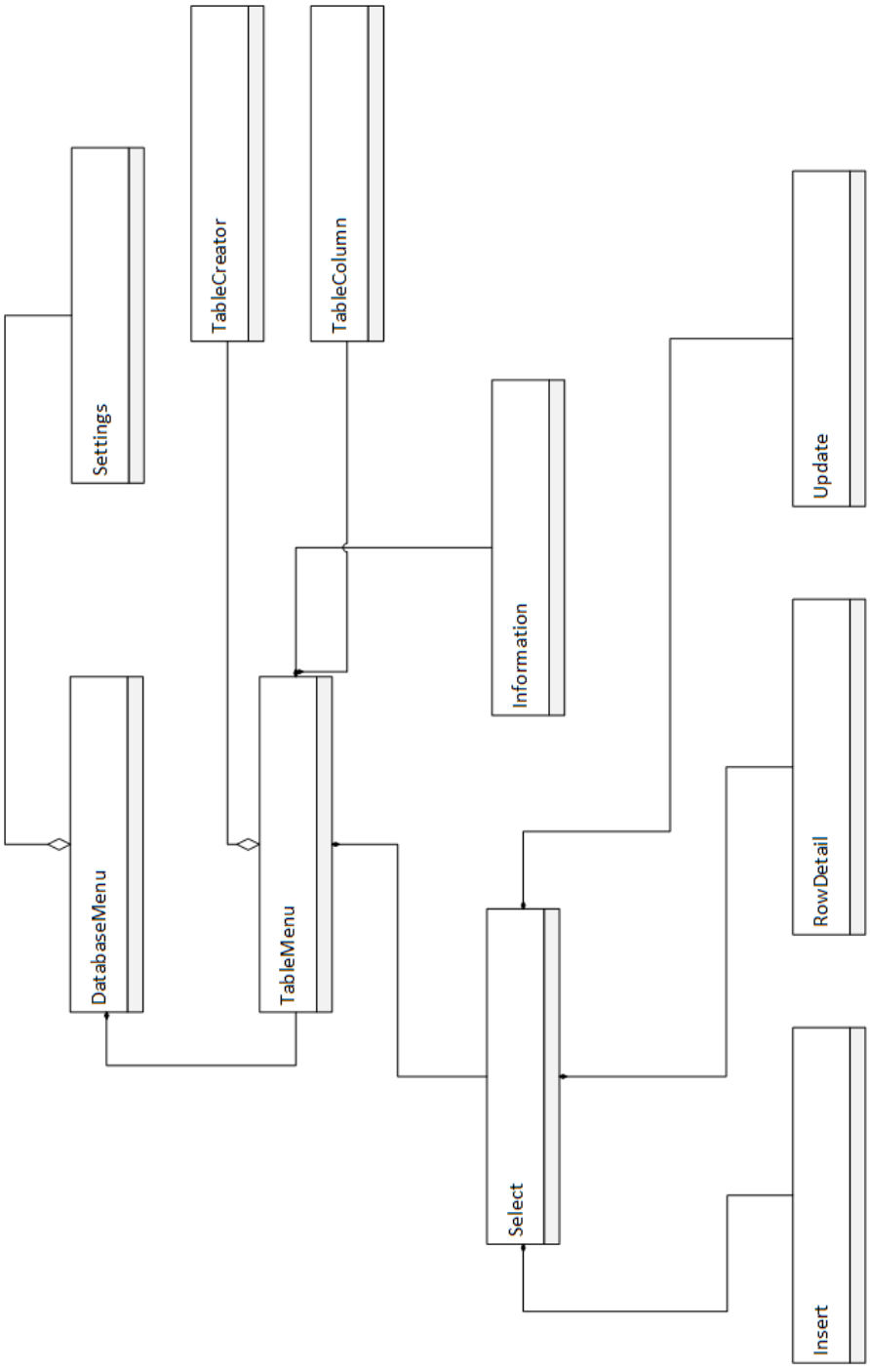
Hierarchii aktivit v aplikaci a vztahy mezi nimi jsem vyjádřil pomocí Unified Modeling Language (dále jen UML) diagramu tříd, viz obr. 7.1.

Následuje detailnější popis každé aktivity, jejich účel v aplikaci a funkce.

### 7.2.1 DatabaseMenu

Jedná se o jednu z nejrozsáhlejších částí celého programu a zároveň úvodní aktivitu, která je uživateli zobrazena po spuštění aplikace. Hlavní funkcí je, že zobrazuje seznam s databázemi dostupnými v datovém prostoru aplikace, pomocí kterého se lze následně dostat ke konkrétním tabulkám v jednotlivých databázích. Seznam je řazen podle času vytvoření, od nejstarších po nejnovější. Cesta k datovému prostoru aplikace je obvykle ve tvaru „/data/data/název balíčku/“, přičemž databáze se nacházejí ve složce `databases`.

Princip, na kterém funguje seznam v této i dalších aktivitách, je založen na tzv. adaptérech. Seznam je zobrazen pomocí komponenty zvané `ListView`. Této komponentě je možné přiřadit různé druhy komponent zvaných `Adapter`. Jednou z možností je využít některý ze standardních adapterů, které jsou součástí knihoven, jako je např. `CursorAdapter`, který slouží k práci s instancí třídy `Cursor`, což je objekt sloužící ke zpracování dotazů a příkazů do databáze a případně vracení výsledků těchto dotazů. Další možností je vytvořit si vlastní adaptér, čehož lze využít v případech, kdy standardní adaptéry neposkytují dostatečné funkce. Pro vytvoření vlastního adaptéru je třeba příslušnou třídu zdědit od nadřazeného adaptéru. Celý řetězec se tedy skládá z 3 částí, `ListView` pro zobrazení dat, `Adapter` jako prostředník mezi objektem uchovávajícím data a seznam zobrazujícím tyto data a nakonec samotný objekt, který data uchovává.




Obrázek 7.1: Schéma aktivit aplikace znázorněné pomocí UML diagramu tříd

Seznam v této aktivitě využívá `ArrayAdapter`, což je jeden ze standardních adaptéru pro práci s polem a kolekcemi. K ukládání dat se využívá kolekce `ArrayList` datového typu `String`.

Jelikož byla aplikace od počátku koncipována tak, aby nebyla omezená jen na databáze v ní vytvořené, respektive aby neplnila pouze úlohu vytváření databází, ale fungovala také rovněž jako jakási čtečka databází, bylo nutné zvolit příponu pro soubory s databázemi, aby bylo možné určit, které druhy souborů bude aplikace vyhledávat. Po prozkoumání datového prostoru ostatních aplikací, které jsou implicitně nainstalovány ve virtuálním prostředí, se zdá, že většina aplikací používá příponu `db`. V zájmu umožnění co nejširší potenciální kompatibility jsem se rozhodl zvolit tuto příponu i pro databáze podporované aplikací.

Zvolení přípony `db` úzce souvisí s dalšími funkcemi této aktivity. Kromě zobrazení databází s danou příponou uložených ve složce `databases` se týká také importu a exportu databáze a vytvoření nové databáze.

Import a export fungují na stejném principu, zajišťuje je tedy i stejná metoda, které je pouze předán parametr s informací, zda se jedná o export, či import. Princip je takový, že je zjištěno, zda existuje vybraná složka a pokud tomu tak není, je vytvořena. Po jejím vytvoření je v ní vytvořen soubor s názvem, jaký uživatel zvolil, k němuž je přidána přípona `db`. Po vytvoření souboru dojde k zkopírování dat ze zdrojového do cílového souboru. V případě exportu je zdrojovou složkou složka `databases` v datovém prostoru aplikace a cílová složka je určená v nastavení. V případě importu je stav opačný. Jak již bylo zmíněno v kapitole 6.1.3, nutnou podmínkou pro využití těchto funkcí je, že zařízení disponuje externím úložištěm. V opačném případě je uživatel o této podmínce informován.

SQLite databáze vytvářejí při změnách databáze transakční žurnál, což je soubor, jehož název je ve stejném tvaru, jako název databáze, jen na konec přidána přípona `-journal`. V rámci minimalizace rizika možné ztráty konzistence dat jsem se rozhodl, že součástí importu a exportu bude automaticky i žurnál dané databáze. Tyto soubory tedy uživateli nejsou zobrazeny, jelikož vše probíhá automaticky a uživatel v tomto směru nemá možnost volby.

Zde jsem narazil na zajímavý problém. Globální proměnná, která uchovává textový řetězec s cestou ke složce pro import a export, totiž dokázala svou hodnotu uchovat pouze v případě, že jí byla předána z nastavení. V momentě, kdy nastavení ještě nebylo inicializováno a tuto hodnotu jsem tedy

předával pouhým přiřazením textového řetězce, aktivita tuto hodnotu následně, když byla volána ve vnořených třídách, ztratila a globální proměnná se tvářila jako prázdná, a to i v případě, že byla nastavená jako statická proměnná. Tento problém jsem tedy vyřešil opětovným přiřazením hodnoty proměnné na vybraných místech kódu.

Další funkcí je vytvoření nové databáze a přejmenování stávající databáze. V případě vytvoření nové databáze se jedná pouze o jednoduché vytvoření souboru ve složce `databases` v datovém prostoru aplikace. Uživatel zvolí název databáze, za který je automaticky přidána přípona `db` a v případě, že se ve složce nenachází soubor se stejným názvem, databáze je vytvořena. V případě, že se tam soubor se stejným názvem nachází, uživateli je nabídnuta možnost tento soubor přepsat, nebo zrušit vytváření a zvolit jiný název. Stejným způsobem jsou hlídány duplicitní názvy i v případě importu. V případě přejmenování stávající databáze je uživateli zobrazeno dialogové okno s předvyplněným původním názvem. Pokud uživatel zadá název, který je již používán jinou databází, je na tuto skutečnost upozorněn a k přejmenování nedojde.

Poslední funkcí je smazání zálohy. Zde je načten obsah složky určené v nastavení a uživatel má možnost smazat obsah celé složky, nebo si vybrat konkrétní databázi. Opět platí, že spolu s databází je automaticky smazán i její žurnál a tento soubor tedy není uživateli zobrazen.

Pokud je do aplikace importována databáze nebo je vytvořena nová databáze, seznam s databázemi v aktivitě je okamžitě aktualizován. Není tedy potřeba aplikaci vypnout a znovu zapnout, případně se přesouvat mezi aktivitami pro aktualizování.

Z této aktivity se lze přesunout do 2 dalších aktivit – `Settings` a `TableMenu`. Obě aktivity budu popisovat dále.

### 7.2.2 Settings

Možnosti této aktivity jsou poměrně skromné, je zde pouze možnost nastavit složku, z které bude probíhat import a do které bude prováděn export, a dále možnost zrušit provedené změny, čímž se nastaví cesta ke složce, která je nastavená jako implicitní.

Implicitně jsem zvolil jako název složky `simpledbmanager_backup` a cesta

k této složce je nastavena do kořenového adresáře externího úložiště.

S každým nastavením je spojen druh tohoto nastavení, který určuje, jak bude dané nastavení zpracováno, uloženo a předáno. Základní druhy se ovšem ukázaly jako nedostačující, jelikož jsou zaměřený příliš konkrétně a není zde možnost pouhého předání textového řetězce, což je ovšem v této aplikaci nutné, jelikož právě touto formou je předávána cesta ke složce se zálohou. Bylo tedy nutné vytvořit novou třídu specificky pro nastavení cesty ke složce. Název této třídy je `LocationPreference` a budu se jí věnovat dále v kapitole 7.3.3.

Během práce na aplikaci jsem narazil na problém, kdy mnou vytvořená třída pro nastavení cesty nevracela žádnou hodnotu, dokud nebyla vytvořena její instance tím, že se v nastavení provedl výběr cesty. Ve třídě obsluhující nastavení ovšem byla zadána implicitní hodnota. Tento problém jsem se tedy pokusil vyřešit tím způsobem, že předtím, než je vytvořena instance nastavení, v aktivitě s nastavením je do popisku nastavení, který zobrazuje cestu a z kterého je také tato cesta předávána dále, napevno přiřazena implicitní hodnota. Tím bylo zajištěno, že i pokud není vyvoláno dialogové okno s volbou složky, v popisku nastavení bude zobrazena implicitní cesta. Zde se projevil jeden z rozdílů mezi virtuálním prostředím a během aplikace na reálném zařízení. Při zjišťování, zda již byla vytvořena instance nastavení, což je zajištěno zavoláním metody pro zjištění popisku nastavení a porovnání tohoto popisku s určitými hodnotami, jsem dospěl ke zjištění, že zatímco na reálném zařízení volaná metoda vrátí prázdný řetězec, na virtuálním zařízení tento řetězec není prázdný, ale obsahuje jakési hodnoty. Ke správnému fungování v obou případech tedy nestačí jen zjištění, zda je řetězec prázdný. V případě této aplikace jsem tento problém vyřešil tím způsobem, že první znak v řetězci je porovnáván se znakem „/“ a pokud tomuto znaku neodpovídá, je do popisku přiřazena implicitní hodnota.

Jak se ovšem ukázalo, pokud uživatel nevyvolá aktivitu s nastavením, nedojde k inicializaci nastavení a popisek s implicitní hodnotou tedy není možné přečíst. Tento problém tedy bylo nutné vyřešit tím, že v aktivitách vyžadujících k některé ze svých funkcí cestu ke složce se zálohami byla tato cesta v případě, že ještě nedošlo k její inicializaci, nastavena přiřazením textového řetězce s implicitní cestou.

### 7.2.3 TableMenu

Aktivita zobrazující tabulky a pohledy v dané databázi. Pohledy jsou od tabulek odlišeny tím způsobem, že za jejich název je přidáno „(view)“. Seznam s tabulkami a pohledy je řazen podle času vytvoření, od nejstarších po nejnovější. Stejně jako v případě nadřazené aktivity `DatabaseMenu` je zde použit adaptér `ArrayAdapter` a data jsou uložena v kolekci `ArrayList` typu `String`. Pokud je v seznamu zvolena některá z tabulek, je zobrazeno dialogové okno s nabídkou, co s tabulkou dále provést.

Na výběr jsou následující položky:

- Zobrazit záznamy – Spustí aktivitu `Select`, která zobrazí záznamy v dané tabulce. Této aktivitě se věnuji v kapitole 7.2.6.
- Editovat – Zde je na výběr mezi 2 možnostmi, a sice přejmenováním tabulky a přidáním sloupce. Přejmenování tabulky funguje naprosto stejně, jako v případě přejmenování databáze, opět je předvyplněný původní název a kontroluje se, zda název není duplicitní i v porovnání s pohledy. Druhou možností je přidání sloupce do tabulky. Tato volba spustí aktivitu `TableColumn`, které se věnuji v kapitole 7.2.5.
- Smazat – Jednoduše smaže tabulku. Nejdříve je zobrazeno okno s varováním a upozornění na následky.
- Informace – Poskytne detailnější informace o tabulce. Tato volba zavolá aktivitu `Information`, které se věnuji v kapitole 7.2.7.

Pokud je v seznamu zvolen pohled, v nabídce chybí možnost editace.

Další z funkcí aktivity je možnost vytvořit novou tabulku. Zvolení této možnosti spustí aktivitu `TableCreator`, které se věnuji v kapitole 7.2.4.

Poslední funkcí aktivity, která je ovšem považována za experimentální a při jejím zvolení je na to uživatel také upozorněn, je možnost ručního zadání SQL příkazu. Tato funkce je momentálně jediný způsob, jak lze v databázi vytvořit pohled. Možnost vytvořit pohled skrze GUI jsem bohužel nestihl ve stanoveném čase vytvořit, v tomto je tedy potenciál pro rozšíření do budoucna. Experimentální je funkce z toho důvodu, že např. momentálně nepodporuje možnost zadání příkazu `SELECT`, jelikož by bylo nutné ošetřit zobrazení záznamů z více různých tabulek. Dále není zaručeno, že zadáním

nevhodného příkazu nedojde k narušení integrity tabulky nebo celé databáze. K pádu aplikace při zadání nevhodného příkazu by ale dojít nemělo, to je ošetřeno pomocí mechanismu výjimek.

Z výše uvedeného je tedy patrné, že kromě zobrazení tabulek a pohledů aktivita slouží jako prostředník pro většinu operací, které je možné s tabulkami provádět.

### 7.2.4 TableCreator

Jediným účelem této aktivity je vytvoření nové tabulky. Předtím, než je uživateli dovoleno cokoliv dalšího, je vyzván k zadání názvu vytvářené tabulky. Po zadání názvu dojde ke kontrole toho, zda je název unikátní v dané databázi. Pokud není, uživatel může název upravit, v opačném případě je umožněno začít vytvářet tabulku. Na displeji jsou zobrazeny 3 atributy, které je možné zadat – název sloupce, datový typ `NOT NULL` v případě, že nechceme, aby bylo možné zadat sloupec prázdný.

Princip fungování aktivity je takový, že aktivita pro zadání dalšího sloupce zavolá sama sebe a předá si atributy, které doposud byly zvoleny, ve formě SQL příkazu v textovém řetězci. Po vyplnění všech požadovaných sloupců je SQL příkaz uzavřen a zpracován. Následně dojde k návratu do aktivity `TableMenu`, kde dojde k aktualizaci seznamu a pokud byla tabulka vytvořena v pořádku, je zde zobrazena jako poslední v pořadí. V opačném případě je zobrazena zpráva o neúspěšném pokusu o vytvoření tabulky.

### 7.2.5 TableColumn

Aktivita sloužící k přidání nového sloupce do existující tabulky. Zobrazení je podobné, jako u aktivity `TableCreator`, pouze s tím rozdílem, že zde chybí možnost přidat sloupci atribut `NOT NULL`. Tento atribut byl odstraněn z toho důvodu, že pokud jsou v tabulce uložené záznamy, nelze přidat nový sloupec, od kterého vyžadujeme, aby byl neprázdný, jelikož po jeho vytvoření by samozřejmě byl u všech stávajících záznamů implicitně prázdný. Pokud v tabulce žádné záznamy nejsou, lze jednoduše využít možnosti založit novou tabulku i s požadovaným sloupcem. Po přidání sloupce do tabulky je tento sloupec přidán na poslední místo v tabulce.

### 7.2.6 Select

Hlavní část aplikace z pohledu uživatele, jelikož poskytuje nejvíce funkcí spojených s prací se záznamy v tabulce. Aktivita zobrazuje záznamy v tabulce, počet záznamů a názvy jednotlivých sloupců, které zároveň slouží jako tlačítko pro řazení záznamů podle daného sloupce. Z tohoto důvodu je zde zobrazen i sloupec s primárním klíčem, aby bylo uživateli umožněno řadit záznamy podle stáří. Řazení podle sloupců se v aktivitě ukládá, takže pokud se do ní uživatel vrátí např. po vložení nového záznamu, seznam se aktualizuje a zároveň zůstane zachováno předešlé řazení. Grafické rozvržení aktivity, tzv. layout, je zde z většiny dynamický, jelikož je závislý na počtu sloupců tabulky a jejich názvech. Z toho důvodu je také layout vytvořen téměř celý programově, nikoliv definicí v `xml` souboru. Rozvržení jsem zvolil takové, že každý sloupec má definovanou stejnou váhu, čímž je zajištěna i stejná šířka všech sloupců, která se s přibývajícím počtem sloupců rovnoměrně snižuje.

Seznam tentokrát nevyužívá žádný ze standardních adaptérů, jelikož z důvodu dynamického layoutu žádný nevyhovoval. Vytvořil jsem proto vlastní adaptér s názvem `DatabaseCursorAdapter`, kterému se dále věnuji v kapitole 7.3.1. Jak vyplývá z názvu adaptéru, data pro tuto aktivitu jsou uchovávána v objektu typu `Cursor`, který slouží ke zpracování dotazů a příkazů do databáze a případně vracení výsledků těchto dotazů. Jedná se tedy o třídu pro komunikaci s databází.

V této aktivitě je dostupná funkce vložení nového záznamu, která spustí aktivitu `Insert`, které se věnuji v kapitole 7.2.8. Tato funkce není dostupná v případě, že se uživatel nachází v pohledu.

Po vybrání záznamu ze seznamu jsou nabídnuty možnosti záznam otevřít, editovat nebo smazat. Otevření zavolá aktivitu `RowDetail` a slouží k přehlednému zobrazení záznamu v případě, že je příliš dlouhý. Této aktivitě se věnuji v kapitole 7.2.10. Účel editace je patrný už ze samotného názvu a tato funkce spustí aktivitu `Update`, které se věnuji v kapitole 7.2.9. Editace není dostupná pro pohledy. Poslední možnost, smazat, zobrazí dialogové okno s upozorněním a pokud uživatel souhlasí, záznam je smazán a seznam se záznamy je aktualizován, opět se zachováním řazení, pokud bylo zvoleno.

V horní části aktivity je rozbalovací nabídka, která slouží k plynulému pohybu mezi tabulkami i pohledy. Není tedy nutné vrátit se o krok zpět do aktivity `TableMenu` a zde následně vybrat hledanou tabulku nebo pohled.

V této aktivitě jsem narazil na zatím nejvážnější problém a omezení týkající se samotného jazyka, konkrétně jeho knihoven pro Android. `ListView` zobrazující data v seznamu totiž poskytuje možnost pouze vertikálního posunu, nikoliv horizontálního. Pro tabulky, které obsahují velké množství sloupců, tato skutečnost představuje velmi významnou překážku. Nabízí se možnost využít třídu zvanou `ScrollView`, která umožňuje i horizontální posun, ovšem tato třída neposkytuje možnost připojení adaptéru. Zkusil jsem tedy vnořit ListView dovnitř horizontálního `ScrollView`, ale jak se ukázalo, toto řešení umožní pouze horizontální posun a `ListView` v tomto případě není možné ovládat. Tento problém ovšem bylo nutné vyřešit, jelikož vzhledem ke zvolenému rozvržení aktivity, kdy je šířka displeje rovnoměrně rozdělená podle počtu sloupců a každý sloupec má tak stejnou šířku, by v případě velkého počtu sloupců došlo k situaci, kdy by každý sloupec byl tak úzký, že by téměř nebyl vidět. V extrémním případě by pravděpodobně mohlo dojít i k pádu aplikace, jelikož sloupce by byly tak úzké, že by je nebylo možné vykreslit.

Řešení tohoto problému jsem zajistil konstantou, která určuje minimální přípustnou šířku sloupce. V průběhu vykreslování layoutu je kontrolována šířka sloupce, což zajištěno tak, že je zjištěna šířka displeje, která je vydělena počtem sloupců tabulky a porovnána s minimální šířkou sloupce. Pokud by tato šířka byla překročena, vykreslí se jen povolený počet sloupců a uživateli je zobrazena zpráva o tom, že ostatní sloupce nebylo možné vykreslit. Pokud uživatel v takovém případě potřebuje zobrazit kompletní data, lze k tomu využít funkci otevření konkrétního záznamu, případně pokud je potřeba pouze zjistit názvy všech sloupců, lze k tomu využít funkci zobrazení informací o tabulce dostupné z aktivity `TableMenu`.

Při řešení problému jsem narazil na uživatelské knihovny, které se pokoušejí toto omezení řešit a přidávají možnost horizontálního ScrollView. Bohužel se mi nepodařilo tyto knihovny zprovoznit, což ovšem nutně neznamená, že knihovny nejsou funkční. Proto je sem pro úplnost přidávám. Jedná se o knihovny `HorizontalVariableListView` [51] a `HorizontalAutoScrollListView` [52].

Jak se ovšem ukázalo, potvrzuje se, že Android je velmi dynamickou platformou, jak jsem popsal v kapitole 6.1. Během psaní tohoto dokumentu, tedy v době, kdy je již vytvářená aplikace dokončená, jsem zjistil, že jednou z nově přidaných funkcí je tzv. `RecyclerView`. S touto třídou nemám žádnou zkušenost, ale jak vyplývá z několika zdrojů [51, 53], mohla by poskytovat řešení problému s horizontálním posunem u `ListView`.

### 7.2.7 Information

Velmi jednoduchá aktivita sloužící k přehlednému zobrazení struktury tabulky. Jsou zde uvedeny názvy sloupců, jejich datový typ (ve skutečnosti se jedná o tzv. *Type Affinity*, jak jsem popsal v kapitole 4.5) a informaci o tom, zda je sloupec typu `NOT NULL` či nikoliv. Není zde uveden sloupec s primárním klíčem v případě, že se jedná o standardní primární klíč s názvem sloupce `_id`, jelikož ten je z principu fungování třídy `Cursor` pevně daný a uživatel pro práci s aplikací nepotřebuje znát. Třída `Cursor` v aplikaci slouží k obsluze databáze a detailněji se jí věnuji v kapitole 7.3.2.

Informace jsou zobrazené ve formě tabulky. I v této aktivitě je možné se pomocí rozbalovací nabídky v horní části GUI přesouvat mezi tabulkami a pohledy.

### 7.2.8 Insert

Aktivita sloužící k vytváření nových záznamů. Nadřazenou aktivitou je `Select`, kde je zajištěno, že přístup k této aktivitě bude umožněn pouze tabulkám, nikoliv pohledům. V horní části GUI se opět nachází rozbalovací nabídka pro plynulý přesun mezi tabulkami, zde ovšem s tím omezením, že se v nabídce nenacházejí pohledy, jelikož pro ně tato aktivita není dostupná.

Aktivita zobrazuje ve vertikálním rozvržení pro každý sloupec jeho název, datový typ, atribut `NOT NULL`, pokud jím sloupec disponuje, a textové pole pro vložení položky do daného sloupce. Sloupec s primárním klíčem není zobrazen v případě, že je název sloupce `_id`. Vysvětlení této skutečnosti se věnuji v kapitole 7.3.1.

Pokud se uživatel pokusí vložit záznam, který má prázdný sloupec s atributem `NOT NULL`, je na tuto skutečnost upozorněn a vložení záznamu není umožněno. Toto omezení bylo nutné ošetřit programově, jelikož databázový systém SQLite podle mých zjištění umožňuje vkládat prázdné sloupce i v případě, že mají definovaný atribut NOT NULL. Je tedy možné, že se stejně jako v případě datových typů jedná pouze o doporučení, nikoliv nutnou podmínku. Je ovšem zajímavé, že pokud je existující tabulce přidán nový sloupec, k čemuž v aplikaci slouží aktivita `TableColumn` popsaná v kapitole 7.2.5, tomuto sloupci není možné nastavit atribut `NOT NULL`, jelikož databázový systém v tomto případě kontroluje, aby sloupce nebyly prázdné. Zdá se tedy, že da-

tabázový systém kontroluje tento atribut pouze při přidání nového sloupce, nikoliv při přidání nového záznamu.

V případě, že tabulka nemá sloupec nazvaný `_id`, předpokládá se, že nedisponuje standardním primárním klíčem. V tomto případě je jako primární klíč považován první sloupec tabulky a před vložením nového záznamu je kontrolováno, zda by nedošlo k duplicitě. V případě, že ano, vložení není umožněno.

### 7.2.9 Update

Aktivita zajišťující editaci záznamu zvoleného v aktivitě `Select`, která je její nadřazenou aktivitou. Aktivita je téměř identická s aktivitou `Insert`, na první pohled jsou zde jen 2 rozdíly. V horní části GUI aktivity chybí rozbalovací menu pro pohyb mezi tabulkami, což je pochopitelné, jelikož aktivita je spojená s konkrétním záznamem, nikoliv celou tabulkou. Další rozdíl je ten, že textová pole pro jednotlivé sloupce zde mají implicitně vyplněné původní hodnoty zvoleného záznamu.

Stejně jako v případě aktivity `Insert` zde v případě, že tabulka nemá sloupec s primárním klíčem nazvaný `_id`, dochází ke kontrole, zda první sloupec tabulky, považovaný za primární klíč, po editaci nezpůsobí duplicitu. V případě, že ano, editace není umožněna.

### 7.2.10 RowDetail

Aktivita, která je stejně jako aktivita `Update` spojená s konkrétním záznamem tabulky. Impulsem k jejímu vzniku byl hlavně problém s absencí možnosti horizontálního posunu v nadřazené aktivitě `Select`, kterému se věnuji v kapitole 7.2.6. Aktivita je velmi jednoduchá, zobrazuje vždy název sloupce a pod ním položku v daném sloupci.

## 7.3 Třídy

Třídy zde fungují na stejném principu, jaký známe z programovacího jazyka Java. Od aktivit se liší v tom, že se neřídí životním cyklem aktivit a nevy-

tvářejí žádný grafický výstup, jedná se tedy čistě o logickou vrstvu aplikace. Pokud bych měl opět provést srovnání s některou částí standardu softwarové architektury MVC, byla by to část zvaná Model. Třídy v této aplikaci tedy zjednodušeně slouží k poskytování dat aktivitám.

Do této sekce jsem zahrnul i třídu `DatabaseHelper`, která není klasickou třídou, ale ani aktivitou. Tato třída se řídí svým vlastním životním cyklem, ale jelikož je tento cyklus odlišný od běžných aktivit a zároveň třída neposkytuje žádný grafický výstup a jedná se o čistě logickou vrstvu aplikace, odpovídá tedy části Model architektury MVC a rozhodl jsem se jí proto zařadit mezi třídy.

### 7.3.1 DatabaseCursorAdapter

Třída sloužící k příjmu dat z databáze a jejich promítnutí do grafických prvků, které jsou následně svázány se seznamem. K příjmu dat z databáze slouží objekt typu `Cursor`, což je objekt sloužící ke zpracování dotazů a příkazů do databáze a případně vracení výsledků těchto dotazů.

Kromě konstruktoru se zde nachází 2 metody, `newView()` a `bindView()`. První jmenovaná slouží k vytvoření grafického rozvržení řádku seznamu, druhá jmenovaná následně zajišťuje vložení správných dat do konkrétního řádku. GUI zde tedy není vytvářeno pro celý seznam `ListView`, s kterým je následně instance třídy svázána, ale pro jednotlivé řádky seznamu.

Třída dědí od třídy `CursorAdapter` sloužící opět k provázání dat přijatých z databáze ve formě objektu typu `Cursor`. Funkčnost obou tříd je tedy v principu stejná, třída `CursorAdapter` se ovšem ukázala nedostačující v tom směru, že neposkytuje možnost dynamického mapování dat. V jejím případě je tedy nutné předem určit pevný počet prvků řádku, v případě této aplikace počet sloupců v tabulce, což samozřejmě není možné, jelikož každá tabulka má jiný počet sloupců, které navíc mohou v průběhu času přibývat.

Z důvodu omezení seznamu `ListView` z hlediska horizontálního posunu, kterému se věnuji v kapitole 7.2.6, probíhá v této třídě kontrola minimální šířky sloupce a pokud by tato šířka měla být překročena, další sloupce nejsou zobrazeny. Pro zobrazení kompletních dat určitého řádku lze využít aktivitu `RowDetail`, které se věnuji v kapitole 7.2.10.

Kontrola minimální šířky probíhá tím způsobem, že je konstantou nasta-

vena minimální šířka sloupce. Tato konstanta se musí shodovat se stejnou konstantou definovanou v aktivitě `Select`. Následně je zjištěna šířka displeje zařízení, která je vydělená počtem sloupců v tabulce a výsledná hodnota je porovnána se zmíněnou konstantou.

Specifickou vlastností třídy `CursorAdapter`, od které tato třída dědí, je, že objekt typu Cursor, který vrací data z databáze, musí obsahovat sloupec s názvem `_id`, jinak tato třída nebude fungovat [54]. Na tuto skutečnost jsem narazil během testování kompatibility s databázemi, které nebyly vytvořené v této aplikace a z nichž některé tento sloupec neobsahovaly. Třída se tím pravděpodobně snaží zajistit, že daná tabulka bude disponovat sloupcem s primárním klíčem a název `_id` byl zvolen jako standard. Jak jsem následně zjistil [55], není nutné tento sloupec pevně definovat tabulce, která jim nedisponuje, ale databázový systém SQLite má vlastní mechanismus, jak tyto situace řešit. V případě, že tabulce chybí sloupec `_id`, je jí přidán skrytý sloupec s názvem `rowid`, který následně plní funkci primárního klíče. Tento sloupec není zobrazen při volání metody pro zjištění sloupců tabulky, jelikož je skrytý, ale lze k němu běžným způsobem přistupovat.

### 7.3.2 DatabaseHelper

Třída, která zprostředkovává veškerou komunikaci s databází. Dědí od třídy `SQLiteOpenHelper`, která poskytuje základní metody pro obsluhu databáze, jako např. získání reference k dané databázi. Nejedná se o klasickou třídu známou z jazyka Java, tato třída se nachází spíše kdesi na pomezí mezi třídou a aktivitou. Má svůj vlastní životní cyklus, který je ovšem odlišný od běžných aktivit, na rozdíl od nich navíc neposkytuje žádný grafický výstup a soustředí se pouze na operace s databázemi a tabulkami. Zaměřuje se tedy čistě na logickou vrstvu aplikace. Pojem životní cyklus u této třídy není úplně přesný, spíše by se zde hodil termín životní etapy, jelikož na rozdíl od aktivit zde nedochází k cyklickému průchodu. Tyto životní etapy odpovídají stavům, které mohou v databázi nastat. Následuje výčet metod odpovídajících těmto stavům.

- `onConfigure()` – Volá se před všemi ostatními metodami. Slouží k nastavení databázového spojení, např. k povolení WAL (vysvětleno v kapitole 4.5) nebo podpory cizích klíčů.
- `onCreate()` – Volá se při prvním vytvoření databáze.

- `onDowngrade()` – Volá se, pokud je aktuální verze databáze novější, než je určeno.
- `onOpen()` – Volá se při každém otevření databáze.
- `onUpgrade()` - Volá se, pokud je aktuální verze databáze starší, než je určeno.

Z výše uvedených je nutné vždy definovat `onCreate()` a `onUpgrade()`, jelikož se jedná o abstraktní metody třídy `SQLiteOpenHelper` [56]. Třída kromě těchto dvou metod obsahuje ještě metodu `onDowngrade()` pro zajištění kompatibility s databázemi, které mají vyšší verzi, než je učeno.

Dalším rozdílem oproti běžným aktivitám je, že třída má kromě metod ošetřujících životní etapy i klasický konstruktor. V konstruktoru je volán rodičovský konstruktor, kterému je předán název a verze databáze. Poté následuje získání reference k databázi, k čemuž zde slouží metoda `getWritableDatabase()`. Tato metoda vrací odkaz na databázi, z které lze číst i do ní zapisovat. V případě, že je databáze prázdná, je zavolána metoda `onCreate()`. Na závěr je zkontrolováno, zda je otevřená nějaká tabulka, k čemuž zde slouží jednoduché ověření, zda je proměnná s názvem tabulky prázdná, nebo nikoliv, a v případě, že tomu tak je, jsou zjištěny názvy sloupců tabulky.

Do metody `onCreate()` se obvykle umisťují příkazy pro vytvoření tabulek, případně zadání implicitních hodnot. V případě této aplikace je metoda prázdná, jelikož není předem známá struktura vytvářené tabulky. Zde bych rád upozornil na možný problém, s kterým jsem se sám setkal při testování vkládání implicitních hodnot. Odkaz na databázi je obvykle uchováván ve formě globální proměnné, do které se ukládá z konstruktoru voláním metody `getWritableDatabase()`. Jelikož tato metoda volá při prázdné databázi metodu `onCreate()`, v průběhu jejího volání je globální proměnná uchovávající odkaz na databázi ještě prázdná. Tato proměnná se naplní až po dokončení metody `onCreate()`. Pro operace s databází a tabulkami je tedy nutné využít parametr metody `onCreate()`, ve kterém je předáván lokální odkaz k této databázi.

Další povinnou metodou je `onUpgrade()`, která v aplikaci funguje naprosto stejným způsobem jako `onDowngrade()` a plní zde úlohu zajištění kompatibility. Pokud totiž má být otevřena databáze s jinou verzí, než která byla předána v konstruktoru, zavolá se jedna z těchto dvou metod v závislosti na tom, zda je verze nižší nebo vyšší. Pro tyto účely je proměnná uchovávající

verzi databáze, která je předávána konstruktoru, nastavená jako statická a v případě volání jedné z těchto metod dojde pouze k přepsání verze databáze tak, aby byla shodná s otevíranou databází. Po ukončení metod se totiž verze databází musejí shodovat, v opačném případě dojde k chybě. Obecně se např. metoda `onUpgrade()` používá v případech, kdy je potřeba s aktualizací aplikace provést i úpravy v tabulce. Její součástí jsou tedy často příkazy na změnu struktury tabulek, vytvoření nových tabulek nebo vložení dodatečných dat do tabulek. Mezi další metody, které poskytuje tato třída, patří např. zjištění názvů, typů nebo atributů sloupců, získání odkazu na databázi. Dále jsou zde metody pro každý ze základních příkazů jazyka SQL, tedy `SELECT`, `INSERT`, `UPDATE` a `DELETE`, navíc i metody pro přejmenování tabulky, přidání nového sloupce a smazání tabulky a pohledu. U většiny z těchto metod dochází ke kontrole, zda tabulka obsahuje sloupec `_id` a v závislosti na tom se kód mírně liší. Tento specifický sloupec je popsán v kapitole 7.3.1.

### 7.3.3 LocationPreference

Třída úzce svázaná jak s modulem `aFileDialog` popsaným v kapitole 7.5, tak s aktivitou pro nastavení aplikace `Settings` popsané v kapitole 7.2.2. Třída umožňuje propojení těchto dvou prvků. Jedná se vlastně o datový typ uchovávající hodnotu nastavení, v tomto případě lokace cílové složky.

Součástí třídy jsou konstruktor, který nastaví modulu potřebné parametry, a metoda zpracovávající kliknutí na položku v aktivitě `Settings`, která spustí modul, čímž se vyvolá dialogové okno s průzkumníkem souborů. Dále se zde nachází nastavení a vrácení počáteční hodnoty, přičemž v metodě pro nastavení počáteční hodnoty dochází ke kontrole, zda již byla nějaká počáteční hodnota uložena, nebo jde o první spuštění. Na závěr jsou zde implementovány mechanismy pro ukládání stavů.

## 7.4 XML soubory

Poslední interní částí aplikace jsou `xml` soubory. Jedná se o definiční soubory se syntaxí značkovacího jazyk XML, ve kterých je možné definovat téměř veškeré statické části aplikace. Specifickým typem `xml` souboru je `AndroidManifest.xml`, kterému se věnuji v kapitole 7.1 a proto zde nebude dále zmíněn.

Rozsah zaměření těchto souborů je poměrně široký, patří sem soubory s definicí vzhledu celé aplikace, aktivit nebo jejích dílčích částí, např. `ActionBar` nebo dialogové okno, dále ikony, barvy, rozměry, písma, textové řetězce a další. Jednotlivé části se mohou ještě dále dělit podle minimálních verzí Android SDK, od kterých jsou podporovány. Této možnosti je v aplikaci využito např. pro písmo, jelikož styl písma, který jsem zvolil, je podporovaný až od Android SDK verze 16. Pro nižší verze jsou tedy použity `xml` soubory s implicitním stylem písma.

Pro soubory definující grafické rozvržení, tzv. layout, platí, že aplikace, aktivita nebo dílčí část aktivity mohou mít přidělený pouze jeden takový soubor. Na druhou stranu každý soubor je možné přidělit neomezenému počtu subjektů. Dále je možné některé atributy v `xml` souborech definovat rovněž programově. V takovém případě platí, že programová definice překrývá statickou definici v `xml` souboru.

V aplikaci jsem pro vzhled tlačítek zvolil definici převzatou z návrhu, na který jsem narazil a který mě svým vzhledem zaujal [57]. Jelikož se nejedná o část aplikace ovlivňující funkčnost, nepovažoval jsem za zásadní přijít s vlastní definicí, která by ve výsledku mohla být vzhledově nezajímavá.

## 7.5 Moduly

Pokud potřebujeme v aplikaci využít služeb externí knihovny nebo aplikace, je možné je importovat do projektu ve formě modulu. Nutnosti uvést tyto moduly v souboru `AndroidManifest` jsem se již věnoval v kapitole 7.1.4. Ovšem pokud chceme modul využívat v naší aplikaci, je nutné provést další úpravy. Podobně jako v případě přejmenování balíčku, což jsem popsal v kapitole 7.1.5, i zde je potřeba upravit soubor `build.gradle` týkající se přímo našeho aplikačního modulu. V tomto souboru je sekce `dependencies`, v níž je uvedeno, co je potřeba zkompilovat před sestavením modulu. Zde je nutné uvést modul, který jsme do projektu importovali, jinak o něm pravděpodobně náš aplikační modul nebude vědět a nebude možné se na něj odkázat. Definice modulu se uvádí v tomto tvaru:

```
compile project(':X')
```

přičemž `X` označuje název modulu, který jsme importovali.

Standardní knihovny jazyka Java dostupné pro systém Android bohužel nedisponují nástrojem pro efektivní pohyb v úložišti, vytváření složek a volbu souborů a složek. Je zde pouze možnost z aplikace zavolat externí aplikaci, která tomuto účelu slouží, to ovšem nepřichází v úvahu v případě, kdy na zařízení není nainstalovaný žádný průzkumník souborů.

Jelikož jsem chtěl udělat volbu složky pro import a export co nejjednodušší, uživatelsky přívětivou a zároveň multiplatformní, tedy zahrnující i zařízení, kde není dostupný průzkumník souborů, byl jsem nucen využít pro tento účel externí knihovnu. V případě, že bych tuto funkci měl vytvořit sám, nebyl bych pravděpodobně ve stanoveném čase schopný vytvořit dostatečně kvalitní řešení. Hlavním zaměřením aplikace má navíc být práce s databázemi, proto jsem se rozhodl, že bude nejlepším řešením využít externí knihovnu.

Pro tento účel jsem se rozhodl použít knihovnu zvanou `aFileDialog` [50], která splňuje veškerá kritéria, která jsem při výběru stanovil. Je malá, v mém projektu zabírá jen něco málo přes 700 kB, takže zbytečně nezatěžuje úložiště zařízení. Uživatelské rozhraní i API je navíc velice jednoduché a intuitivní. Knihovna se dá použít ve dvou režimech, jako samostatná aktivita, nebo ve formě dialogového okna. Rozhodl jsem se pro formu dialogového okna, aby se usnadnila orientace v aplikaci a nepoužívala se zbytečně další aktivita, když lze její funkci provést i v dialogovém okně. Dále je možné nastavit mnoho parametrů upřesňujících, co bude knihovně umožněno a co nikoliv. Všechny nastavitelné parametry, které jsem od knihovny vyžadoval, jsem zde našel, což je další důvod, proč jsem se nakonec rozhodl právě pro tuto knihovnu.

Při používání knihovny je možné narazit na skutečnost, že mnoho složek, které jsou v úložišti, není možné zobrazit a pokud ano, není vidět jejich obsah. Nejde ovšem o problém knihovny, je to z toho důvodu, že aplikace nemá systémem uděleno právo čtení těchto složek.

# 8 Ověření funkčnosti

K ověření funkčnosti aplikace došlo jak na virtuálním, tak na fyzickém zařízení s operačním systémem Android různých verzí. Ve všech případech aplikace bezproblémově fungovala.

Fyzické zařízení, na kterém byla aplikace testována, je LG L90 s verzí operačního systému Android 4.4.2.

Následuje výčet virtuálních zařízení, na kterých byla aplikace testována.

- Nexus S API 21 (Android 5.0)
- Nexus S API 21 bez externího úložiště (Android 5.0)
- Nexus One API 21 (Android 5.0)
- Nexus 5 API 21 (Android 5.0)
- Nexus 4 API 19 (Android 4.4)

Testování na virtuálních zařízeních probíhalo s pomocí nástroje AVD Manager, který je standardní součástí vývojového prostředí Android Studio. Ovládání aplikace a její funkčnost se na virtuálních a fyzických zařízeních nijak neliší.

K ověření funkčnosti aplikace jsem navrhl 3 testovací scénáře. Tyto scénáře byly provedeny na všech zmíněných testovacích zařízeních, s výjimkou virtuálního zařízení bez externího úložiště, na němž nebyly prováděny operace související s importem a exportem.

První testovací scénář začíná první instalací aplikace. Aplikaci je možné nainstalovat několika způsoby, které jsou popsány v uživatelské příručce. Po nainstalování aplikace je vytvořena první databáze. Nejdříve je otestováno, zda aplikace umožní zadat nevhodný název databáze, který by následně mohl způsobit pád. Aplikace tomuto kroku zabrání a zobrazí upozornění o tom, že se databázi nepodařilo vytvořit. Před otevřením korektně vytvořené databáze jsem se pokusil vytvořit další databázi se stejným názvem, jako má první vytvořená databáze. Aplikace tomuto kroku zabrání s upozorněním, že zadaný název se shoduje s názvem jiné databáze. Nová databáze je tedy vytvořená

s odlišným názvem a následně jsem se pokusil jí pomocí přejmenování opět přiřadit název shodný s první vytvořenou databází. Aplikace tomuto kroku opět zabránila.

Po otestování hlídání duplicitních názvu databází je otevřena první vytvořená databáze a v ní jsou provedeny stejné kroky, jako v případě duplicitních názvů databází, tentokrát ovšem pro názvy tabulek. Stejně jako v případě databází, i zde aplikace zabrání zadání duplicitních názvů jak při vytváření, tak při editaci. Dále je testována ochrana proti duplicitním názvům sloupců v dané tabulce. Nejdříve jsem se pokusil vytvořené tabulce přidat nový sloupec s názvem, který již některý ze sloupců tabulky obsahuje. Tato operace nebyla dokončena se zprávou o duplicitním názvu sloupce. Následně jsem se pokusil vytvořit novou tabulku, které jsem zadal pro několik sloupců shodné názvy. Aplikace tomuto kroku opět zabrání, zde ovšem následuje pouze obecné upozornění, že tabulku nebylo možné vytvořit. Mechanismus, který tomuto kroku brání, totiž zamezuje všem případným chybám při vytváření tabulky, ne pouze duplicitním názvům.

Tímto tedy bylo otestováno, zda aplikace zabrání vložení duplicitních názvů u databází, tabulek a sloupců v tabulkách. Další fází testovacího scénáře je import databáze, která obsahuje tabulku s nestandardním primárním klíčem, což znamená, že sloupec s primárním klíčem nemá název `_id`. K tomuto účelu jsem využil databáze, kterou lze nalézt ve virtuálních zařízeních pod následující cestou: „/data/data/com.android.documentsui/databases/recents.db“. Tuto databázi jsem pomocí nástroje Android Device Monitor (dále jen ADM), který je standardní součástí Android Studia, zkopíroval do osobního počítače a následně zpět do zařízení. Databáze byla zkopírována do implicitní složky pro zálohu, která má název `simpledbmanager_backup` a nachází se v kořenovém adresáři externího úložiště. Následně jsem provedl import databáze do aplikace, přičemž během importu jsem zkusil zadat název shodný s některou z databází nacházejících se v aplikaci. Duplicitní název nebyl povolen, ponechal jsem proto původní název a dokončil import. Po importování databáze jsem databázi otevřel, což proběhlo úspěšně. V databázi jsem otevřel tabulku nazvanou `recent` a následně vložil několik záznamů, přičemž při vkládání každého z těchto záznamů jsem do sloupce `key`, který slouží jako primární klíč, zadával duplicitní primární klíč. Aplikace tomuto zabránila a upozornila na duplicitu. Při zadání unikátního primárního klíče proběhlo vložení v pořádku. Stejný scénář jsem provedl při editaci záznamu, kde se aplikace zachovala stejně a vložení zabránila.

Druhý testovací scénář pracuje s již vytvořenou databází, která obsahuje

tabulky. Nejprve je otestována funkce ručního zadání SQL příkazu. Během testování této funkce se nepodařilo způsobit havárii aplikace a potvrdilo se, že neumožňuje zadání příkazu SELECT. Uživatel je ovšem před použitím této funkce varován, že se jedná o funkci experimentální a může způsobit havárii nebo poškození databáze. Po otestování stability funkce pro ruční zadání SQL příkazu byl s její pomocí vytvořen pohled vytvořený ze sloupců několika tabulek. Vytvoření pohledu proběhlo úspěšně, byl zobrazen v seznamu s tabulkami a za názvem pohledu se nacházela přípona `(view)`. Následné otevření pohledu proběhlo úspěšně. Rovněž bylo ověřeno, že dialogové okno při výběru pohledu ze seznamu tabulek nenabízí možnost editace, stejně tak po otevření pohledu nebyla přístupná možnost vkládat záznamy.

Po ověření těchto funkcí došlo k otestování rozbalovacího menu pro přesun mezi tabulkami a pohledy. Přesun probíhal v pořádku, za názvem pohledu se po rozbalení seznamu opět nacházela přípona `(view)` a při přesunu do pohledu bylo znemožněno vkládání pohledů. Při zvolení funkce pro vkládání záznamu z jedné tabulky a následném přesunu do jiné tabulky rovněž vše proběhlo v pořádku.

Třetí testovací scénář opět pracuje s již vytvořenou databází, která obsahuje tabulky. Nejprve se provede záloha databází do složky, která byla vytvořena na externím úložišti pomocí nastavení. Následně bylo několik exportovaných databází smazáno jednotlivě, poté zbylé databáze hromadně a nakonec byly smazány i některé databáze přímo z aplikace, přičemž po každém smazání jsem pomocí nástroje ADM kontroloval, zda došlo i k fyzickému smazání souborů s databázemi. Zde se potvrdilo, že mazání funguje v pořádku. Další testovanou funkcí bylo mazání tabulek, které také fungovalo v pořádku.

Po otestování mazání databází a tabulek bylo několik zbylých databází exportováno do složky pro zálohu a následně byla aplikace odinstalována a zařízení restartováno. Po opětovném spuštění zařízení byla aplikace znovu nainstalována, v nastavení byla zvolena složka, do které byl proveden export databází a následně pomocí funkce pro import byly tyto databáze zkopírovány do aplikace. Při žádném z těchto kroků nedošlo k problému a importované databáze obsahovaly všechna původní data. Na závěr byla otestována minimální šířka sloupce. Byla vytvořena nová tabulka, v níž bylo vytvořeno 15 sloupců. Po otevření této tabulky se zobrazil jen určitý počet sloupců a bylo zobrazeno upozornění, že zbylé sloupce nebylo možné vykreslit. Při vkládání testovacích záznamů se všechny sloupce tabulky zobrazily korektně, stejně tak se vložené záznamy zobrazily korektně při jejich otevření a editaci.

# 9 Možnosti rozšíření

Možnosti rozšíření aplikace jsou poměrně rozsáhlé a věřím, že v případě dalších úprav by aplikace mohla sloužit jako kvalitní databázový manažer. První volbou by dle mého názoru měla být analýza třídy `RecyclerView` a možnost jejího uplatnění místo třídy `ListView`. Tuto možnost jsem zmiňoval již v kapitole 7.2.6. Další z funkcí, kterou by bylo možné doplnit, je vytvoření pohledu pomocí GUI. Aplikace v současné podobě nabízí pouze možnost vytvoření pohledu pomocí ručního zadání SQL příkazu. Dále v aplikaci není implementována podpora cizích klíčů. S cizími klíči se váže i možnost graficky zobrazit vztahy mezi tabulkami. Nabízí se např. i možnost editovat strukturu tabulky, v současné době je možné pouze přidat sloupec. Další funkcí, kterou by nemělo být příliš náročné implementovat, je hromadné mazání záznamů v tabulkách. Je tedy patrné, že možností rozšíření aplikace je celá řada.

# 10 Závěr

Ze zjištění popsaných v této práci vyplývá, že výběr databázových systémů odpovídajících vstupním kritériím, tedy volně šiřitelných vestavěných relačních databázových systémů, je v době psaní této práce velmi omezený. Vstupními kritérii prošly 2 databázové systémy, a to H2 Database a SQLite. Z důvodu udávané lepší výkonnosti, lepší podpory pro cílovou platformu a v neposlední řadě integrace do cílové platformy jsem pro vyvíjenou aplikaci zvolil databázový systém SQLite.

Podle očekávání tento systém disponuje jistými omezeními v porovnání se standardními robustními databázovými systémy používanými např. na osobních počítačích, ale tato omezení nejsou tak rozsáhlá, jak jsem původně čekal a některá z nich ani nehrají roli z podstaty zadání, ve kterém se předpokládá, že databáze je vestavěná a přistupuje k ní současně jen jeden uživatel. Z mého pohledu se nejzásadnějším omezením zdá absence podpory doménové integrity. Datové typy sloupců zde fungují jen jako doporučení, ale databázový systém nedisponuje mechanismem k jejich vynucení. Citelným omezením také může být velmi omezená podpora SQL příkazu `ALTER TABLE`, který umožňuje pouze přejmenovat tabulku nebo přidat nový sloupec do tabulky, nebo nemožnost zapisovat do pohledů. Některá z těchto omezení by ovšem mohla jít vyřešit programově.

Z pohledu platformy Android je jediné zásadní omezení, na které jsem narazil, nemožnost horizontálního posunu, respektive nemožnost pohybu v zobrazených seznamech horizontálně a vertikálně zároveň. Osobně ale tuto skutečnost nepovažuji za zásadní překážku do budoucna, jelikož platforma Android je stále relativně mladá a velmi dynamická a předpokládám, že tento problém bude odstraněn. Už v době psaní této práce jsem narazil na novou funkcionalitu Android SDK, která by tento problém mohla řešit. Tuto funkci jsem neměl možnost vyzkoušet, ale je zmíněna v dokumentu. Kolem platformy je navíc velká komunita a narazil jsem i na několik uživatelských knihoven, které se snaží problém alespoň částečně řešit.

Závěrem bych rád vyjádřil názor, že i přes výše zmíněná omezení je SQLite na platformě Android solidním databázovým systémem, který má široké uplatnění, jak vyplývá mimo jiné i ze skutečnosti, že slouží k lokálnímu ukládání dat aplikací na dané platformě.

# Přehled zkratek

**ACID** Atomicity, Consistency, Isolation, Durability
: Souhrn pravidel, která zajišťují spolehlivost databázových transakcí.

**ADB** Android Debug Bridge
: Nástroj pro komunikaci s připojeným zařízením běžícím na platformě Android.

**ADM** Android Device Monitor
: Nástroj pro práci s připojeným zařízením běžícím na platformě Android. Umožňuje například procházet adresářovou strukturu, stahovat a vkládat soubory nebo simulovat příchod textové zprávy.

**ADT** Android Developer Tools
: Vývojářské nástroje pro Android.

**Android NDK** Android Native Development Kit
: Souhrn nástrojů umožňující při vývoji pro platformu Android používat jazyk C/C++.

**Android SDK** Android Software Development Kit
: Souhrn nástrojů umožňující vývoj aplikací pro platformu Android v jazyce Java.

**API** Application Programming Interface
: Rozhraní pro programování aplikací.

**ART** Android Runtime
: Běhové prostředí pro platformu Android.

**ASE** Android Scripting Environment
: Souhrn nástrojů umožňující při vývoji pro platformu Android používat skriptovací jazyky.

**AVD** Android Virtual Device
Virtuální zařízení běžící na platformě Android.

**BLOB** Binary Large Object
Datový typ binárních dat tvořících jeden objekt.

**Dalvik VM** Dalvik Virtual Machine
Běhové prostředí pro platformu Android.

**dex** Dalvik Executable
Formát bytekódu spustitelný běhovým prostředím platformy Android.

**GUI** Graphical User Interface
Grafické uživatelské rozhraní.

**Java SE** Java Standard Edition
Standardní edice jazyka Java.

**JDBC** Java Database Connectivity
API pro komunikaci s databází pomocí jazyka Java.

**JVM** Java Virtual Machine
Běhové prostředí pro jazyk Java.

**MVC** Model-View-Controller
Standard softwarové architektury oddělující od sebe jednotlivé vrstvy softwaru.

**ODBC** Open Database Connectivity
API pro komunikaci s databází.

**OOP** Object-oriented programming
Objektově orientované programování.

**SQL** Structured Query Language
Dotazovací jazyk používaný pro práci s relačními databázemi.

**UML** Unified Modeling Language
Jazyk pro vizualizaci návrhů softwaru.

**WAL** Write-Ahead Logging
Funkce SQLite, která umožňuje současně číst i zapisovat data.

# Literatura


[1] *Android* [online]. [cit. 2014-12-7]. <https://www.android.com/>

[2] *Which programming languages can be used to develop in Android?* [online]. Paresh Mayani. 4.8.2010 [cit. 2014-11-23]. <http://stackoverflow.com/questions/3316801/which-programming-languages-can-be-used-to-develop-in-android>

[3] *javax.sql | Android Developers* [online]. 20.11.2014 [cit. 2014-11-23]. <http://developer.android.com/reference/javax/sql/package-summary.html>

[4] *javax.sql (Java Platform SE 7)* [online]. c1993 [cit. 2014-11-23]. <https://docs.oracle.com/javase/7/docs/api/javax/sql/package-summary.html>

[5] *Embedded Database* [online]. 18.1.2008, poslední revize 10.11.2014 [cit. 2014-11-18]. <http://en.wikipedia.org/wiki/Embedded_database>

[6] *Empress Software Inc. ∼ Empress Embedded Database* [online]. c2014 [cit. 2014-11-23]. <http://www.empress.com/products/products.html>

[7] *Firebird* [online]. c2012, poslední revize 1.12.2014 [cit. 2014-12-7]. <http://www.firebirdsql.org/>

[8] mariuz. *Firebird News > Firebird Client 3.0 Alpha 2 for Android Download* [online]. 20.3.2014 [cit. 2014-11-23]. <http://www.firebirdnews.org/firebird-client-3-0-alpha-2-for-android-download/>

[9] mariuz. *Firebird News > It is time for Firebird on Android and IOS ?* [online]. 25.9.2013 [cit. 2014-11-23]. <http://www.firebirdnews.org/it-is-time-for-firebird-on-android-and-ios/>

[10] *firebird-devel - libfbembed.so static for android* [online]. Alex Peshkoff. 26.3.2014 [cit. 2014-11-23]. <http://firebird.1100200.n4.nabble.com/libfbembed-so-static-for-android-td4635596.html>

[11] *Firebird: JDBC Driver* [online]. c2000, poslední revize 27.4.2014 [cit. 2014-11-24]. <http://www.firebirdsql.org/en/jdbc-driver/>

[12] cmayer0087. *Android Firebird JDBC Driver Port | SourceForge.net* [online]. 27.1.2013, poslední revize 25.12.2013 [cit. 2014-11-24]. <http://sourceforge.net/projects/androidjaybird/>

[13] *how to connect to firebird DB on android* [online]. Mark Rotteveel. 7.5.2012 [cit. 2014-11-24]. <http://stackoverflow.com/questions/10473779/how-to-connect-to-firebird-db-on-android>

[14] *H2 Database Engine* [online]. [cit. 2014-11-24]. <http://www.h2database.com/html/main.html>

[15] *Tutorial* [online]. [cit. 2014-11-24]. <http://www.h2database.com/html/tutorial.html#android>

[16] *How to convert .jar or .class to .dex file?* [online]. Pavel Netesa. 31.12.2012 [cit. 2014-11-24]. <http://stackoverflow.com/questions/8348144/how-to-convert-jar-or-class-to-dex-file>

[17] YANG, Herong. *"dalvikvm"Command to Run Java Application* [online]. c2012, poslední revize 2012 [cit. 2014-11-24]. <http://www.herongyang.com/Android/shell-dalvikvm-Command-to-Run-Java-Application.html>

[18] *Android sample using H2* [online]. Ted. 16.12.2010, poslední revize 22.12.2010 [cit. 2014-11-24]. <http://stackoverflow.com/questions/4465571/android-sample-using-h2>

[19] *InterBase Database | Embeddable Database* [online]. c2014 [cit. 2014-11-24]. <http://www.embarcadero.com/products/interbase>

[20] *InterBase* [online]. 25.10.2006, poslední revize 16.2.2014 [cit. 2014-11-24]. <http://cs.wikipedia.org/wiki/InterBase>

[21] ODERSKY, Martin. *The Scala Language Specification Version 2.9* [online]. 11.6.2014 [cit. 2014-11-24]. <http://www.scala-lang.org/files/archive/nightly/pdfs/ScalaReference.pdf>

[22] CLABURN, Thomas. *Google Releases 'Simple' Android Programming Language* [online]. 28.7.2009 [cit. 2014-11-24]. <http://www.informationweek.com/applications/google-releases-simple-android-programming-language/d/d-id/1081735?>

[23] ELIASEN, Alan. *Frink* [online]. 21.12.2001 [cit. 2014-11-25]. <http://futureboy.us/frinkdocs/>

[24] *SQLite HomePage* [online]. [cit. 2014-11-30]. <https://www.sqlite.org/index.html>

[25] *SQLite vs MySQL* [online]. 6.8.2008, poslední revize 28.2.2013 [cit. 2014-11-30]. <http://stackoverflow.com/questions/3630/sqlite-vs-mysql>

[26] TÝŘ, Jiří. *Google zakázal upravenou Android CyanogenMod ROM* [online]. 1.10.2009 [cit. 2015-04-10]. <http://www.root.cz/clanky/google-zakazal-upravenou-android-cyanogenmod-rom/>

[27] UJBÁNYAI, Miroslav. *Programujeme pro Android*. 1. vyd. Praha: Grada Publishing a.s., 2012. 192 s. ISBN 978-80-247-7983-6.

[28] *Android Kernel Features* [online]. 9.9.2009, poslední revize 10.11.2013 [cit. 2015-04-10] <http://elinux.org/Android_Kernel_Features>

[29] VON EITZEN, Chris. *Android drivers to be included in Linux 3.3 kernel* [online]. 23.12.2011 [cit. 2015-04-10]. <http://www.h-online.com/open/news/item/Android-drivers-to-be-included-in-Linux-3-3-kernel-1400996.html>

[30] VAUGHAN-NICHOLS, Steven. *Linus Torvalds on Android, the Linux fork* [online]. 18.8.2011 [cit. 2015-04-10]. <http://www.zdnet.com/article/linus-torvalds-on-android-the-linux-fork/>

[31] *An overview of the Android Architecture* [online]. poslední revize 3.7.2014 [cit. 2015-04-10]. <http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture>

[32] WELCH, Chris. *Before it took over smartphones, Android was originally destined for cameras* [online]. 16.4.2013 [cit. 2015-04-10]. <http://www.theverge.com/2013/4/16/4230468/android-originally-designed-for-cameras-before-smartphones>

[33] MCLEAN, Prince. *Canalys: iPhone outsold all Windows Mobile phones in Q2 2009* [online]. 21.8.2009 [cit. 2015-04-10]. <http://appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html>

[34] *Market share of smartphone OS of total smartphone installed base in 2013 and 2014* [online]. c2015 [cit. 2015-04-10]. <http://www.statista.com/statistics/385022/smartphone-worldwide-installed-base-operating-systems-share/>

[35] *Mobile App Development & App Creation Software - Xamarin* [online]. c2015 [cit. 2015-04-11]. <http://xamarin.com/>

[36] GILBERT, Ben. *What you need to know about Google's battle with Oracle over Android* [online]. 14.9.2014 [cit. 2015-04-11]. <http://www.engadget.com/2014/10/14/googles-oracle-explainer/>

[37] *What is... the Dalvik virtual machine?* [online]. 25.10.2011 [cit. 2015-04-11]. <http://www.electronicsweekly.com/eyes-on-android/what-is/the-dalvik-virtual-machine-2011-10/>

[38] LINDER, Brad. *What's new in Android 5.0 Lollipop?* [online]. 15.10.2014 [cit. 2015-04-11]. <http://liliputing.com/2014/10/whats-new-android-5-0-lollipop.html>

[39] CHENG Ben, BUZBEE Bill. *A JIT Compiler for Android's Dalvik VM* [online]. 5.2010 [cit. 2015-04-11]. <http://www.android-app-developer.co.uk/android-app-development-docs/android-jit-compiler-androids-dalvik-vm.pdf>

[40] *ART and Dalvik | Android Developers* [online]. [cit. 2015-04-11]. <https://source.android.com/devices/tech/dalvik/index.html>

[41] *Activity | Android Developers* [online]. 7.4.2015 [cit. 2015-04-11]. <http://developer.android.com/reference/android/app/Activity.html>

[42] ANDLINGER, Paul. *RDBMS dominate the database market, but NoSQL systems are catching up* [online]. 21.11.2013 [cit. 2015-04-12]. <http://db-engines.com/en/blog_post/23>

[43] MALÝ, Martin. *NOSQL* [online]. c2011 [cit. 2015-04-12]. <http://vyuka.maly.cz/download/21-nosql.pdf>

[44] BATKO, Michal. *Relační vs. objektově-relační vs. objektové databáze* [online]. [cit. 2015-04-12]. <http://www.fi.muni.cz/~xbatko/oracle/compare.html>

[45] *Dashboards | Android Developers* [online]. [cit. 2015-04-13]. <https://developer.android.com/about/dashboards/index.html>

[46] <*uses-sdk*> | *Android Developers* [online]. [cit. 2015-04-13]. <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html>

[47] Android Developer Tools | Android Developers [online]. [cit. 2015-04-13]. <http://developer.android.com/tools/help/adt.html>

[48] <manifest> | Android Developers [online]. [cit. 2015-04-13]. <http://developer.android.com/guide/topics/manifest/manifest-element.html>

[49] *Saving Files | Android Developers* [online]. [cit. 2015-04-15]. <http://developer.android.com/training/basics/data-storage/files.html>

[50] jfmdev. *afiledialog - An Android library which provides a simple file chooser - Google Project Hosting* [online]. 23.4.2013, poslední revize 11.2.2015 [cit. 2015-04-15]. <https://code.google.com/p/afiledialog/>

[51] Alessandro Crugnola. *sephiroth74/HorizontalVariableListView GitHub* [online]. 12.11.2012, poslední revize 26.3.2015 [cit. 2015-04-16]. <https://github.com/sephiroth74/HorizontalVariableListView>

[52] Huy Le Tran. *ListView auto scroll horizontally - CodeProject* [online]. 5.9.2013, poslední revize 6.9.2013 [cit. 2015-04-16]. <http://www.codeproject.com/Articles/648221/ListView-auto-scroll-horizontally>

[53] Andre Perkins. *How to build a Horizontal ListView with RecyclerView?* [online]. 11.2.2015, poslední revize 18.3.2015 [cit. 2015-04-16]. <http://stackoverflow.com/questions/28460300/how-to-build-a-horizontal-listview-with-recyclerview>

[54] *CursorAdapter | Android Developers* [online]. poslední revize 14.4.2015 [cit. 2015-04-18]. <http://developer.android.com/reference/android/widget/CursorAdapter.html>

[55] Tim Wu. *Android column '_id' does not exist?* [online]. 21.9.2011 [cit. 2015-04-18]. <http://stackoverflow.com/questions/3359414/android-column-id-does-not-exist/>

[56] *SQLiteOpenHelper | Android Developers* [online]. poslední revize 14.4.2015 [cit. 2015-04-18]. <http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

[57] Folkert. *Gradient buttons for android* [online]. 18.2.2011 [cit. 2015-04-19]. <http://www.dibbus.com/2011/02/gradient-buttons-for-android/>

[58] Jimmy Huch. *Android Package Name convention* [online]. 8.6.2011 [cit. 2015-04-25]. <http://stackoverflow.com/questions/6273892/android-package-name-convention>

# A Uživatelská příručka

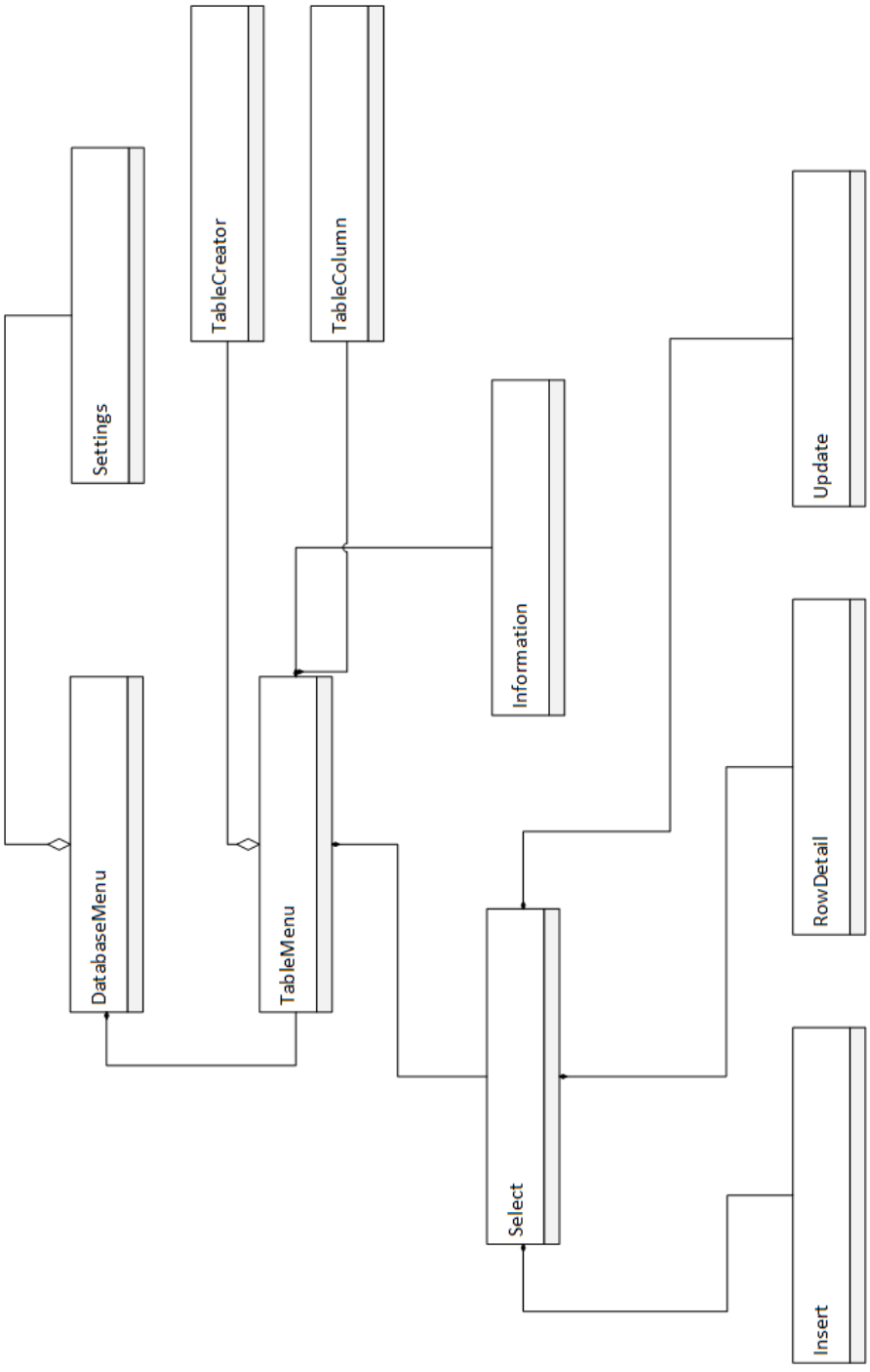
Uživatelská příručka má za cíl usnadnit uživateli instalaci a používání aplikace. Bude zde vysvětleno jak se dostat k různým funkcím aplikace, co tyto funkce nabízejí, jaká jsou jejich omezení a jak je využívat. Přiložené obrázky pocházejí z virtuálního zařízení Nexus S s verzí systému Android 5.0. Ovládání aplikace na tomto zařízení se nijak neliší od testovaného fyzického zařízení.

Pro lepší pochopení struktury aplikace a pohybu mezi aktivitami zde přikládám diagram aktivit, viz. obr. A.1.

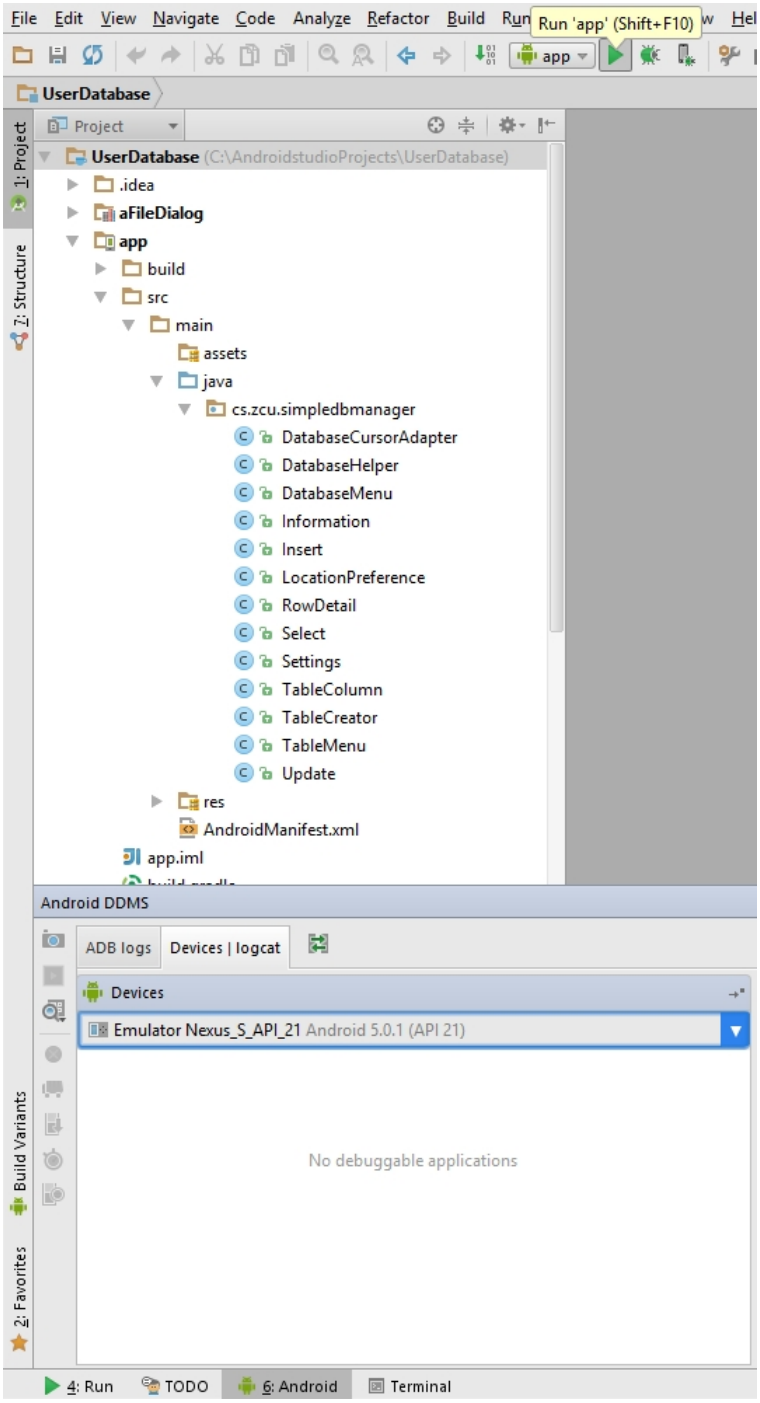
## A.1 Instalace

Instalaci lze provést 2 způsoby. První možností je využít vývojové prostředí Android Studio, ve kterém lze po otevření projektu s aplikací provést kompilaci a nainstalování aplikace na připojené zařízení, viz. obr. A.2. K tomu je ovšem nutné nejdříve povolit na zařízení tzv. Možnosti vývojáře a v nich následně zapnout Ladění USB (anglicky Developer options a USB debugging). Možnosti vývojáře jsou implicitně skryty a k jejich zobrazení je nutné přejít do „/Nastavení/Info o telefonu“ (anglicky „/Settings/About phone“) a zde následně 7x kliknout na volbu Číslo sestavení (anglicky Build number). U některých zařízení se číslo sestavení nachází v sekci Softwarové informace. Následně lze přejít do Nastavení a zde zapnout Ladění USB, viz. obr. A.3.

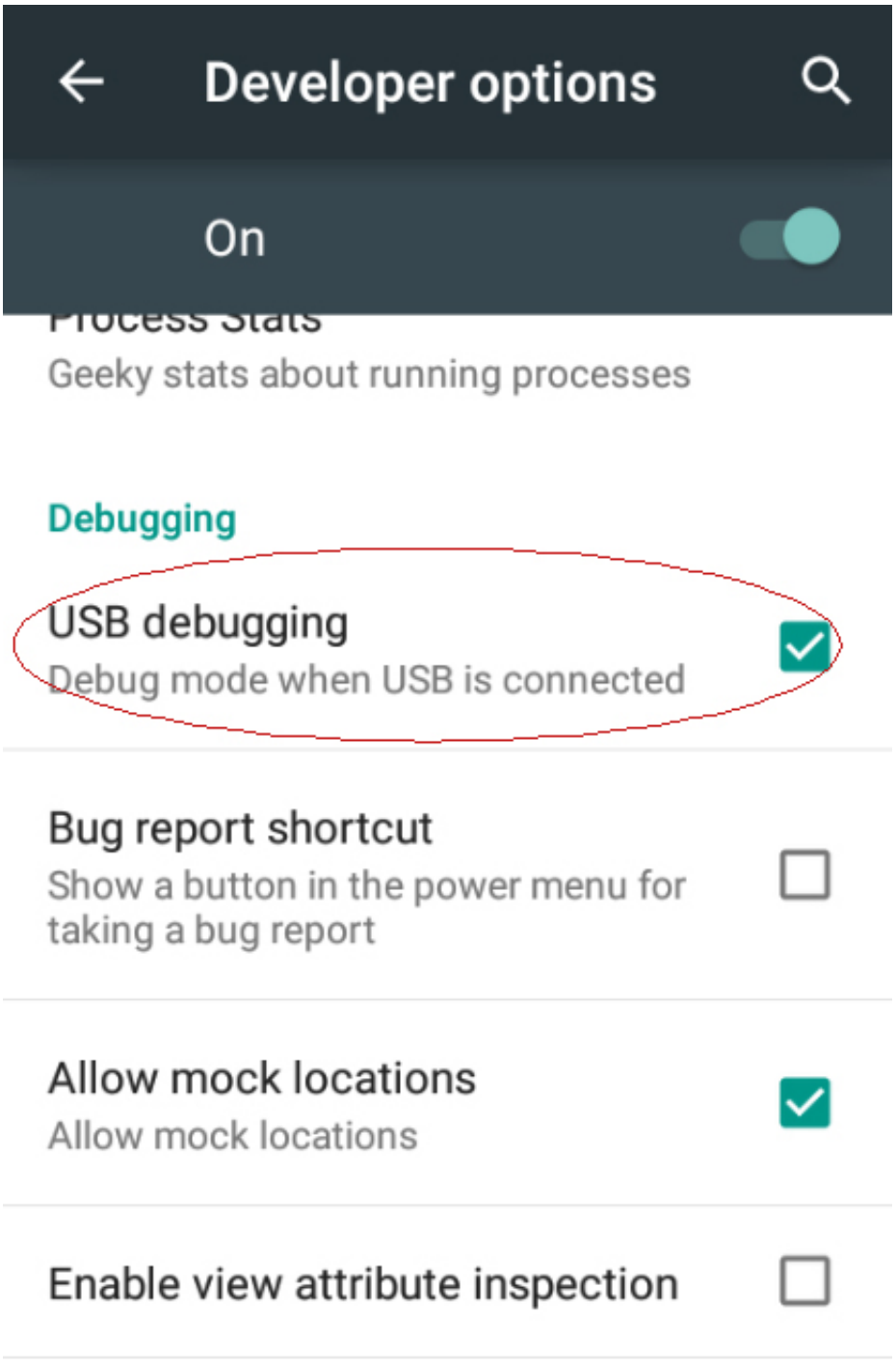
Druhou možností, jak nainstalovat aplikaci, je zkopírování souboru `SimpleDBManager.apk` do zařízení a jeho následné spuštění. Před instalací aplikace je nejprve nutné povolit v nastavení zařízení instalaci z neznámých zdrojů. Tuto volbu lze nalézt na následující cestě: „/Nastavení/Zabezpečení/Neznámé zdroje“ (v anglické verzi „/Settings/Security/Unknown sources“), viz. obr. A.4. Ve většině případů o této nutnosti systém uživatele sám informuje a umožní mu přesunout se přímo ke konkrétnímu nastavení.



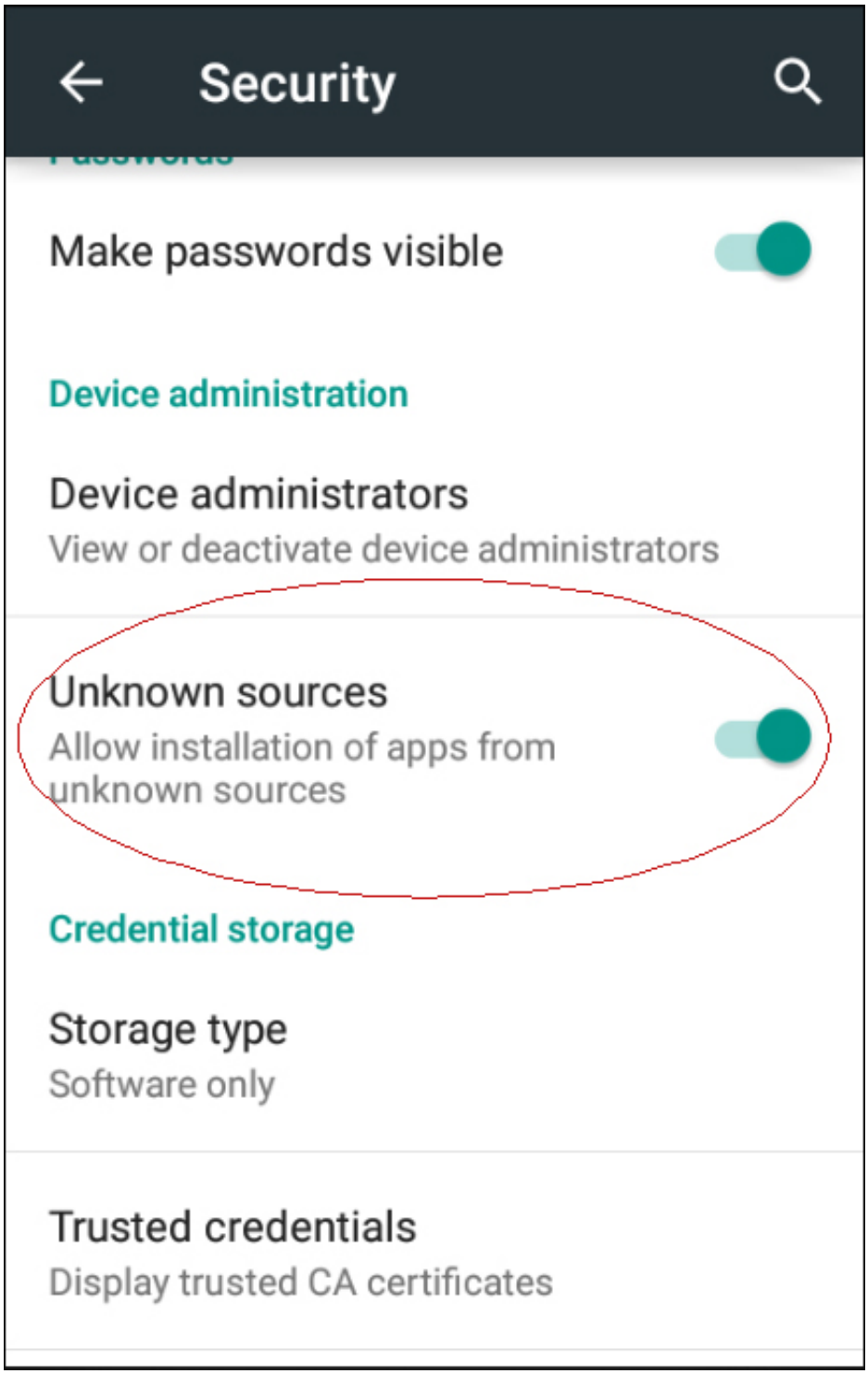

Obrázek A.1: Schéma aktivit aplikace znázorněné pomocí UML diagramu tříd




Obrázek A.2: Instalace aplikace z programu Android Studio.




Obrázek A.3: Povolení ladění přes USB.




Obrázek A.4: Povolení instalace aplikací z neznámých zdrojů.

## A.2 Ovládání

Snahou bylo udělat ovládání aplikaci co nejvíce intuitivní a zároveň se držet zvoleného způsobu způsobu u všech aktivit aplikace. Pohyb v aplikaci a ovládání jednotlivých funkcí probíhá primárně pomocí dialogových oken.

Po spuštění aplikace je uživateli zobrazena úvodní aktivita zobrazující seznam s dostupnými databázemi a několik dalších funkcí, viz. obr. A.5. Obrázek obsahuje očíslované funkce tak, jak se na ně nyní budu odkazovat.

1. Nastavení. Spustí aktivitu, ve které probíhá výběr složky, pro kterou se provádí import a export databází.
2. Nová databáze. Vyvolá dialogové okno s výzvou k zadání názvu databáze.
3. Záloha. Vyvolá dialogové okno s možností volby mezi importem a exportem databází a poskytuje možnost smazat zálohované databáze, jak jednotlivě, tak hromadně.
4. Seznam databází. Po kliknutí na položku v seznamu je vyvoláno dialogové okno s možnostmi, jak dále postupovat. Tyto možnosti jsou Otevřít, Editovat a Smazat. Otevření spustí další aktivitu, která zobrazuje tabulky a pohledy v dané databázi. Editace umožňuje přejmenovat databázi. Smazání dovoluje smazat databázi.

Další aktivitou, do které se lze přesunout, je nastavení. Zde je možné zvolit umístění zálohy, případně vrátit nastavení do původního stavu. Aktuální umístění zálohy je zobrazené v popisku pod volbou Umístění zálohy. Po kliknutí na tuto volbu je vyvoláno dialogové okno pro volbu složky s možností vytvořit novou složku.


Obrázek A.5: Úvodní aktivita se seznamem databází.

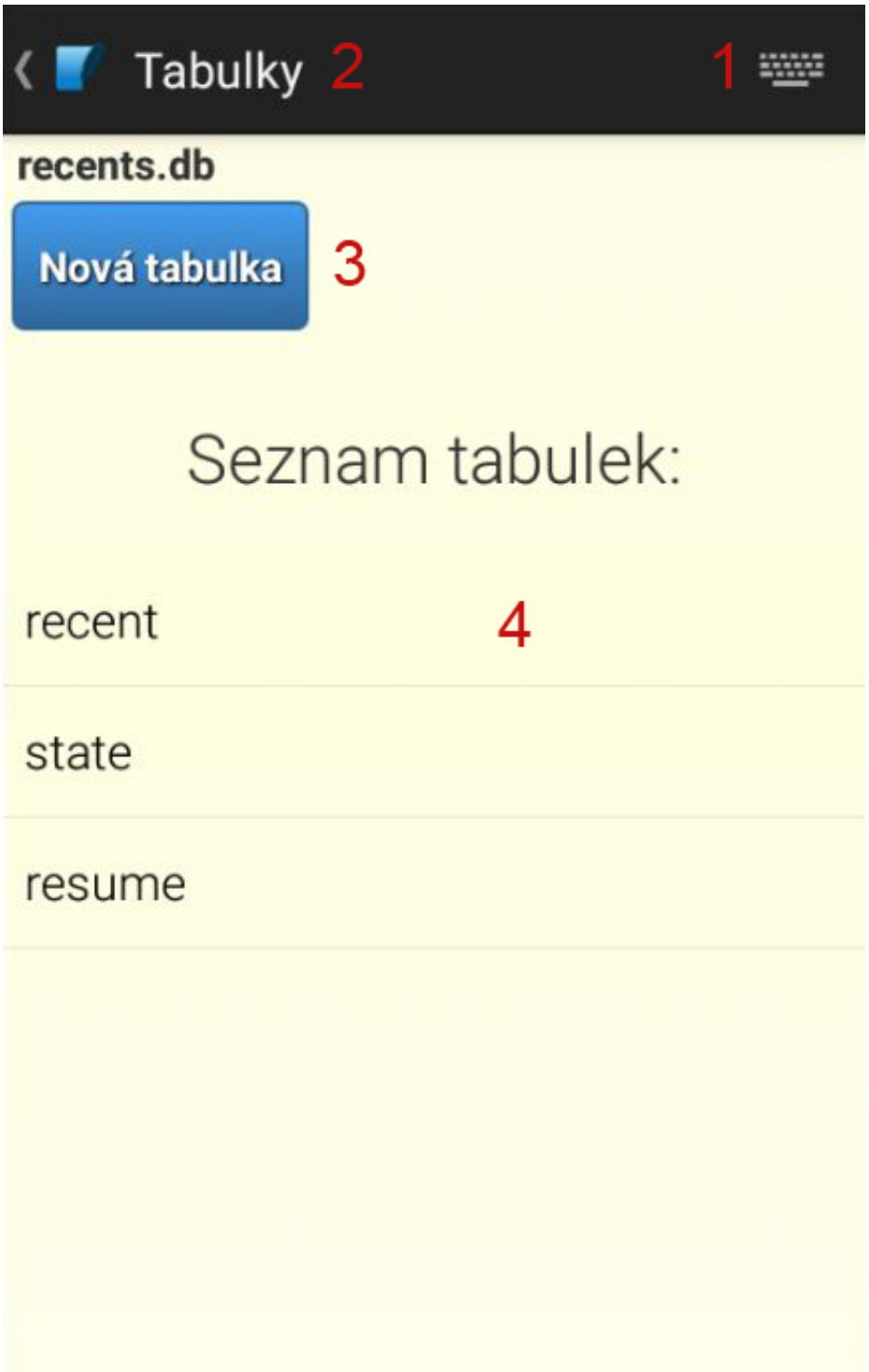
Po otevření databáze je zobrazena aktivita se seznamem tabulek a pohledů dostupných v dané databázi, viz. obr. A.6. V horní části aktivity uživatel vidí, v jaké databázi se právě nachází. Následuje popis funkcí této aktivity podle jejich očíslování na zmíněném obrázku.

1. Ruční zadání SQL příkazu. Jedná se o experimentální funkci, která může způsobit pád aplikace, na což je uživatel před jejím použitím upozorněn. Jediným omezením je, že neumožňuje zadání SQL příkazu `SELECT`, jelikož by bylo potřeba ošetřit vykreslení pro výběr z více tabulek. Jedná se zároveň o jediný způsob, jakým je možné vytvořit pohled.

2. Tlačítko Zpět. Uživatel nemusí využívat tlačítka Zpět na zařízení, ale má možnost využít tohoto tlačítka, které zároveň zobrazuje krátký popis aktivity, v které se právě nachází. Toto tlačítko se nachází ve většina aktivit v aplikaci.

3. Nová tabulka. Tlačítko zavolá aktivitu, která slouží k vytvoření tabulky. Tato aktivita bude popsána dále.

4. Seznam tabulek. Stejně jako v případě seznamu s databázemi, i zde je po kliknutí na záznam v seznamu vyvoláno dialogové okno s možnostmi, jak dále postupovat. Tyto možnosti jsou Zobrazit záznamy, Editovat, Smazat a Informace. V případě, že je vybrán pohled, možnost editace zde chybí. Editace dále nabízí výběr mezi přejmenováním tabulky a přidáním nového sloupce. Po zvolení možnosti Informace je zavolána aktivita zobrazující tabulku s názvy sloupců tabulky, jejich datovými typy a informací o tom, zda má sloupec atribut `NOT NULL`. Funkce Zobrazit záznamy otevře tabulku a zobrazí záznamy, pokud nějaké obsahuje. Tato aktivita má více funkcí a budu se jí věnovat dále.




Obrázek A.6: Aktivita se seznamem tabulek.

Aktivita pro vytvoření nové tabulky, která se uživateli zobrazí po kliknutí na tlačítko Nová tabulka v seznamu tabulek, je zobrazena na obr. A.7. Před stavem, který je viditelný na obrázku, je uživatel nejdříve vyzván k zadání názvu tabulky a předtím, než je umožněno pokračovat ve vytváření, je provedena kontrola duplicity názvu. Po zadání unikátního názvu v dané databázi se uživatel dostává k samotnému vytváření. Sloupci je možné nastavit 3 atributy, a to název, datový typ a případně `NOT NULL`. Název sloupce nesmí být prázdný a zároveň musí být unikátní v rámci tabulky. Datový typ zde plní pouze roli doporučení, aplikace nevynucuje jeho dodržování při následném vkládání záznamů. Následuje popis tlačítek v této aktivitě.

- Vytvořit – Vytvoří tabulku z dosud zadaných sloupců včetně aktuálně zadaného.
- Hotovo – Vytvoří tabulku z dosud zadaných sloupců vyjma aktuálně zadaného.
- Další – Uloží aktuální sloupec a umožní zadání dalšího sloupce.
- Zrušit – Zruší vytváření tabulky a vrátí uživatele do aktivity se seznamem tabulek.


Obrázek A.7: Aktivita pro vytvoření tabulky.

Poslední aktivitou, která zde bude popsána, je zobrazení záznamů, viz. obr. A.8. Aktivita zobrazuje tabulku se záznamy v dané tabulce nebo pohledu, včetně nadřazené databáze a počtu záznamů v tabulce. Následuje popis funkcí aktivity podle jejich očíslování na obrázku.

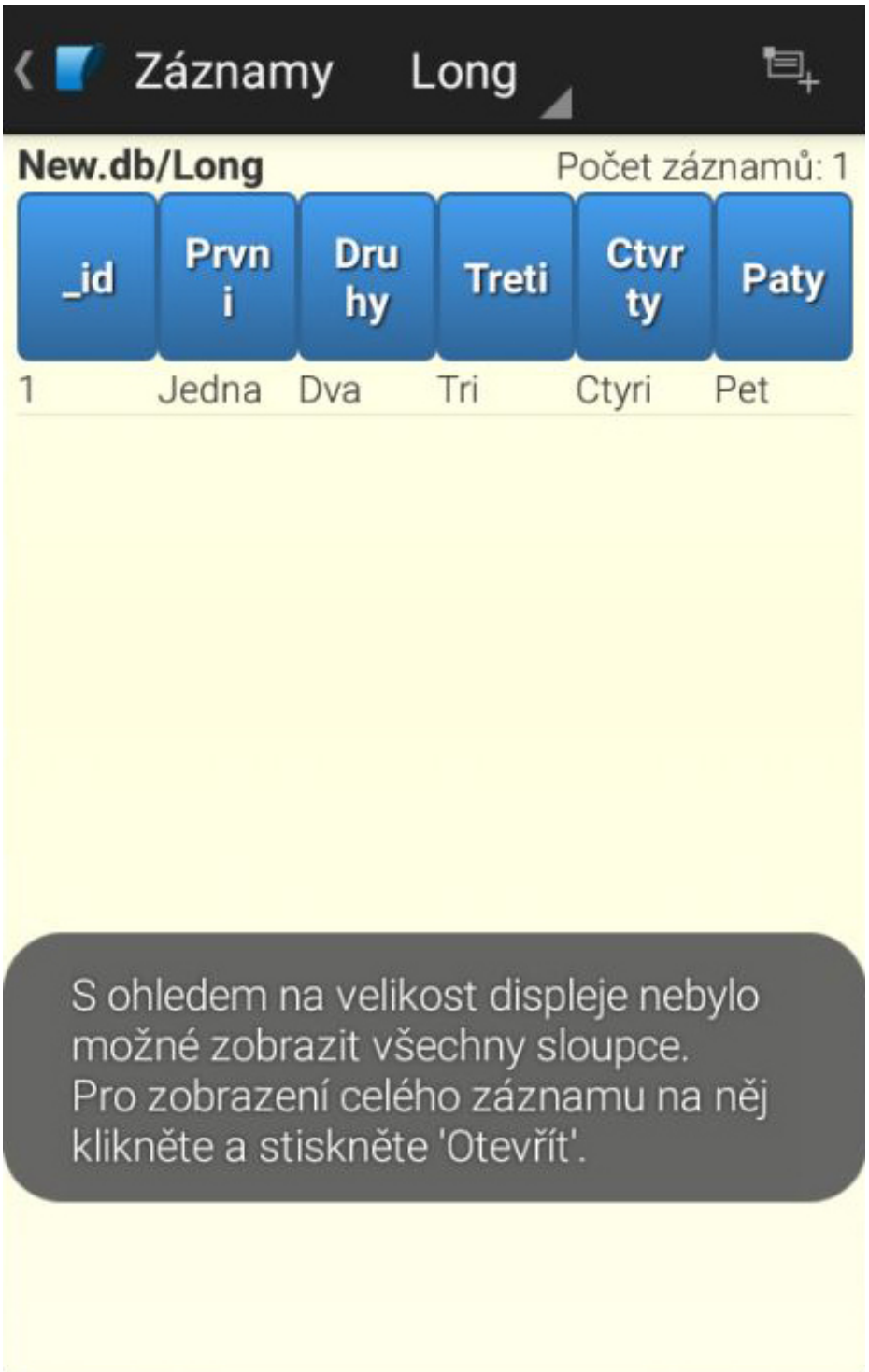
1. Vložení nového záznamu. Vyvolá aktivitu, ve které jsou textová pole nadepsaná názvem sloupce, jeho datovým typem a případně atributem `NOT NULL`, který je aplikací vynucován v případě, že je sloupci nastaven. Tato funkce není dostupná pro pohledy. V případě vkládání záznamu do tabulky, která neobsahuje standardní sloupec s primárním klíčem nazvaný `_id`, k čemuž může dojít např. při importu cizí databáze do aplikace, je aplikací hlídána unikátnost položky vkládané do prvního sloupce tabulky, který je považován za primární klíč.

2. Rozbalovací seznam se seznamem tabulek a pohledů v databázi. Slouží k plynulému přechodu bez nutnosti vracet se o krok zpět a zde následně provádět výběr. Tento seznam je dostupný i v případě vkládání nového záznamu.

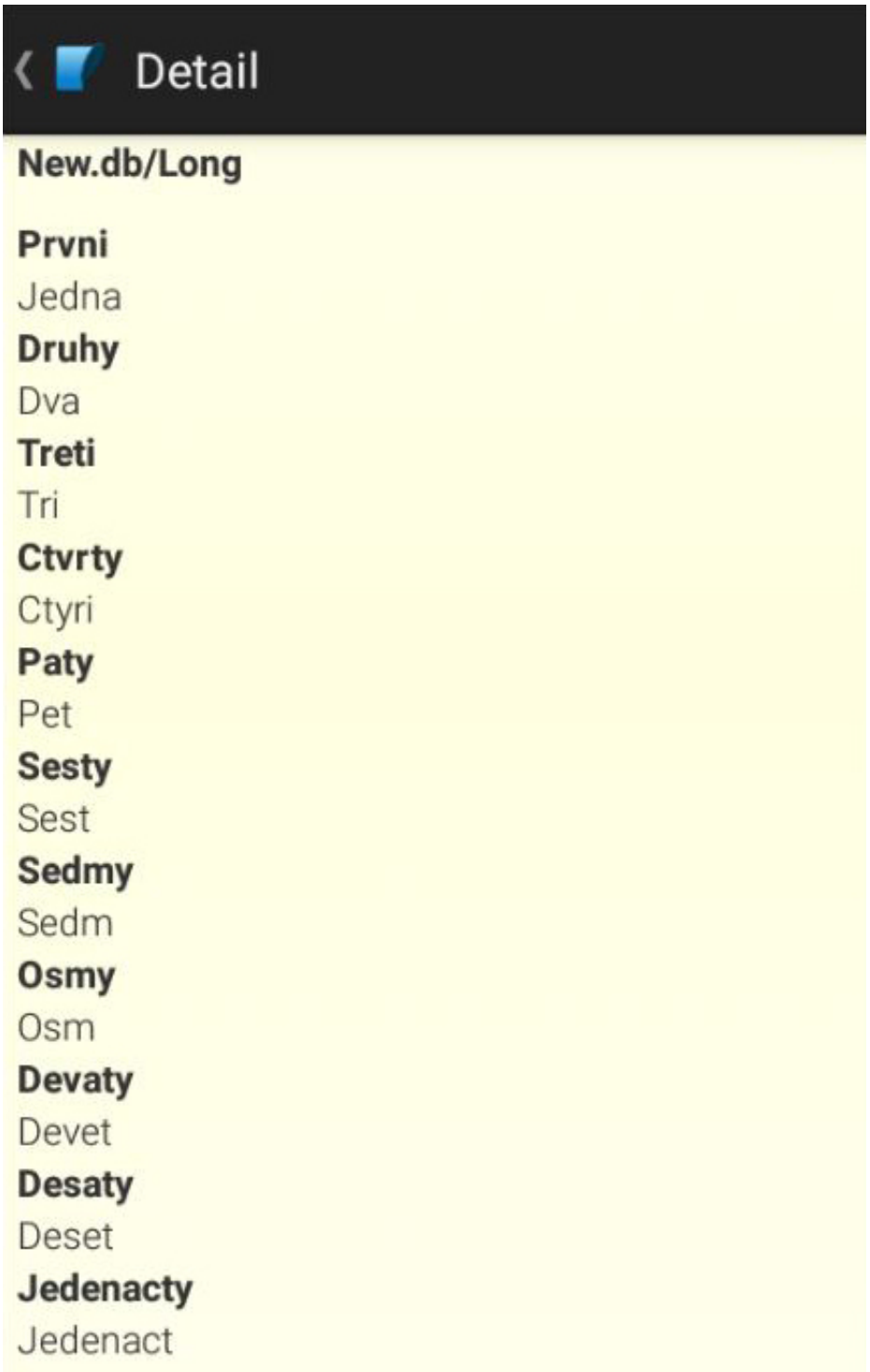
3. Názvy sloupců ve formě tlačítek. Slouží k řazení záznamů podle daného sloupce. Pokud je aktivováno řazení, za název sloupce je přidána šipka znázorňující, zda je řazení prováděno vzestupně nebo sestupně. Způsob řazení lze změnit opětovným kliknutím na tlačítko.

4. Záznamy v tabulce. Po vybrání záznamu je vyvoláno dialogové okno s možnostmi, jak dále postupovat. Na výběr jsou možnosti Otevřít, Editovat a Smazat. Funkce Otevřít vyvolá aktivitu, ve které je záznam zobrazen přehledně podle jednotlivých položek, kde každé položce a názvu sloupce patří vlastní řádek. Slouží pro situace, kdy je příliš velký počet sloupců a tyto sloupce není možné všechny vykreslit na displej zařízení, viz. obr. A.9 a A.10. Funkce Editovat vyvolá stejnou aktivitu jako v případě vkládání nového záznamu, pouze s tím rozdílem, že jsou zde vyplněny původní hodnoty zvoleného záznamu.


Obrázek A.8: Aktivita zobrazující záznamy tabulky.



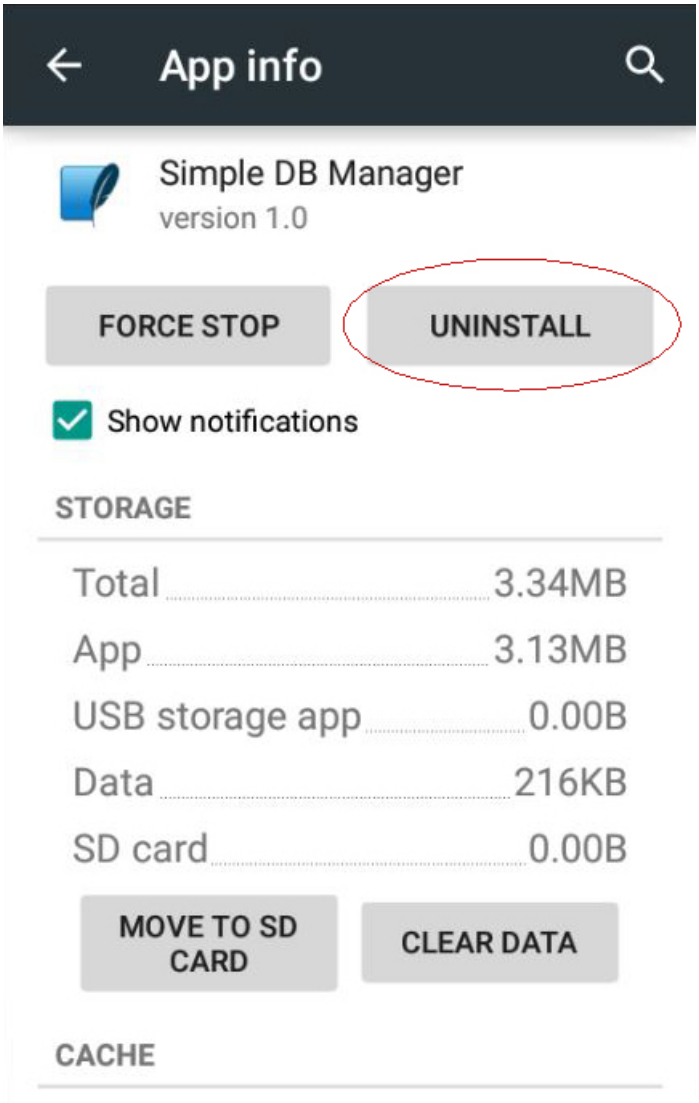

Obrázek A.9: Nevykreslení všech sloupců tabulky.




Obrázek A.10: Detail záznamu.

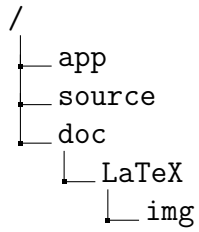
## A.3 Odinstalace

Smazání aplikace je velice jednoduché a probíhá naprosto stejným způsobem, jako u ostatních aplikací na platformě Android. V menu „/Nastavení/ Aplikace/“ (anglicky „/Settings/Apps/“) stačí vybrat aplikaci s názvem `Simple DB Manager` a zde následně zvolit možnost Odinstalovat (anglicky Uninstall), viz obr. A.11.




Obrázek A.11: Odinstalování aplikace.

# B Obsah CD




- `app` - Složka s instalačním souborem aplikace ve formátu `apk`.
- `source` - Složka obsahující zdrojové kódy včetně použitých externích knihoven a nástroje k sestavení ve formě projektu z vývojového prostředí Android Studio.
- `doc` - Složka obsahující dokument bakalářské práce ve formátu `pdf` a složku se zdrojovým kódem, šablonou a obrázky k tomuto dokumentu.
- `LaTeX` - Složka obsahuje soubor se zdrojovým kódem dokumentu bakalářské práce ve formátu `tex`, soubor se šablonou pro dokument ve formátu `cls` a složku s obrázky potřebnými k přeložení zdrojového kódu dokumentu.
- `img` - Složka s obrázky použitými v dokumentu bakalářské práce.

V kořenovém adresáři CD se nachází textový soubor s popisem struktury a obsahu CD s názvem `ReadMe.txt`