DAIS – Semestrální projekt – Doplňující materiál

Petr Lukáš

2. dubna 2017

Abstrakt

Tento dokument obsahuje upřesňující informace k semestrálnímu projektu do předmětu Databázové a informační systémy. Jsou zde uvedeny příklady typických chyb, které se v projektech vyskytují, a postupy, jak se těmto chybám vyvarovat. Součástí dokumentu je také několik ukázkových minispecifikací, ve kterých se chyby dělají nejčastěji. Kompletní zadání včetně pokynů je umístěno na webu `dbedu.cs.vsb.cz`.

Obsah

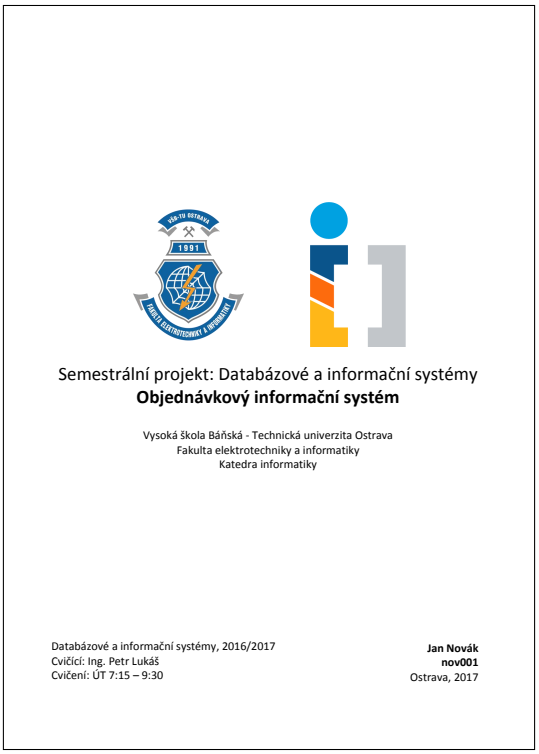
1 Časté chyby v analýze semestrálního projektu

V této kapitole je uveden a popsán seznam nejčastějších chyb, které se v projektech vyskytují. Před odevzdáním projektu si tuto kapitolu prostudujte a na každý bod, který je formulován jako otázka, si sami zkuste odpovědět „ano“ nebo „ne“. Odpovíte-li s nejlepším svědomím vždy „ano“, je pravděpodobné, že máte projekt v pořádku.

1.1 Obecné zásady

1.1.1 Je součástí analýzy hlavička (jméno, příjmení, login, předmět, ročník, cvičící, fakulta, katedra)?

Častým problémem je, že součástí dokumentu není hlavička. Hlavička není pouze na okrasu, ale usnadňuje orientaci v projektech. Projekt, jehož součástí není hlavička vypadá na první pohled neúplně a obvykle naznačuje, že i obsahová kvalita bude na nízké úrovni. Ukázkovou hlavičku můžeme vidět např. na Obrázku 1. Je doporučeno hlavičku umístit na samostatnou stránku.



Obrázek 1: Ukázková hlavička projektu

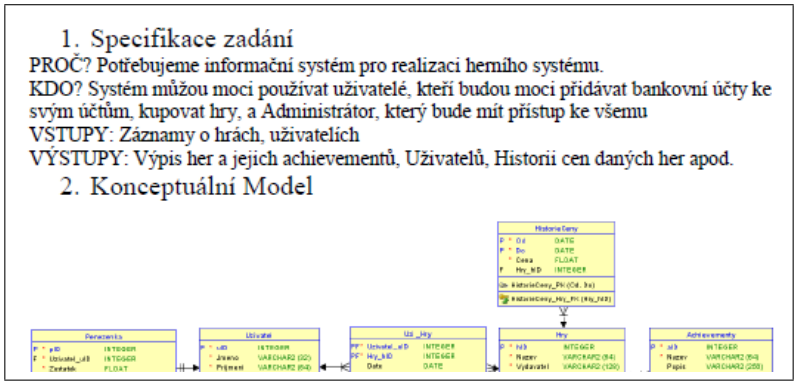
1.1.2 Je součástí analýzy obsah (seznam kapitol + odkazy na stránky)?

Letmým pohledem na obsah je možé jednoduše zkontrolovat, zda je analýza kompletní. Obsah by měl být automaticky vygenerovaný tak, ať skutečně koresponduje s tím, co se v textu nachází dál. Obsah by se měl také nacházet na samostatné stránce. Pokud je analýza kompletní, pak by se obsah měl vemi blížit tomu, co je na Obrázku 2.

Obrázek 2: Obsah analýzy

1.1.3 Je analýza dobře strukturovaná? Tj. používají se číslované nadpisy?

Častým problémem analýz je nevhodné nebo nejednoznačné strukturování textu. Ukázku vidíme na Obrázku 3. Autor evidentně nepoužil číslované nadpisy, ale nadpisy udělal formou číslovaných odrážek. Chybí mezery, které by nadpisy oddělily od zbytku textu. Kromě toho je popis zadání až příliš stručný, ale to nesouvisí se strukturováním. Existují i horší případy, kdy se nadpisy neočíslují vůbec.


1. Specifikace zadání

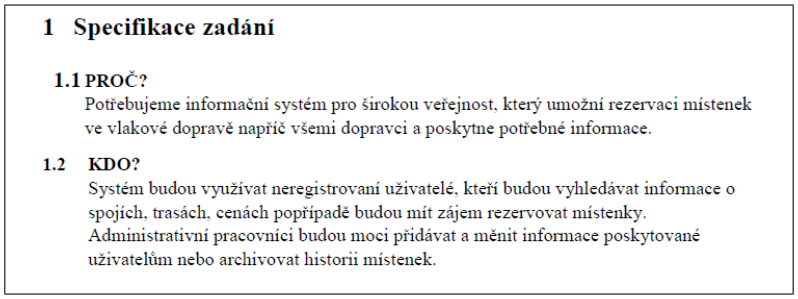
PROČ? Potřebujeme informační systém pro realizaci herního systému.
KDO? Systém můžou moci používat uživatelé, kteří budou moci přidávat bankovní účty ke svým účtům, kupovat hry, a Administrátor, který bude mít přístup ke všemu
VSTUPY: Záznamy o hrách, uživatelích
VÝSTUPY: Výpis her a jejich achievementů, Uživatelů, Historii cen daných her apod.

2. Konceptuální Model

Obrázek 3: Nevhodné strukturování

1.1.4 Je analýza přehledná? Tj. používá se konzistentně určitý typ fontu v určité velikosti pro text a nadpisy na různých úrovních?

Opět viz Obrázek 3. Takováto analýza se nedá považovat za přehlednou. Jinou ukázku vidíme na Obrázku 4, kde je na první pohled jinak zarovnaný nadpis 1.1 a používá se pro něj jiná velikost písma než pro nadpis 1.2.



1 Specifikace zadání

1.1 PROČ?

Potřebujeme informační systém pro širokou veřejnost, který umožní rezervaci místenek ve vlakové dopravě napříč všemi dopravci a poskytne potřebné informace.

1.2 KDO?

Systém budou využívat neregistrovaní uživatelé, kteří budou vyhledávat informace o spojích, trasách, cenách popřípadě budou mít zájem rezervovat místenky. Administrativní pracovníci budou moci přidávat a měnit informace poskytované uživatelům nebo archivovat historii místenek.

Obrázek 4: Nepřehledná analýza

1.1.5 Je analýza srozumitelná? Jsou srozumitelné a čitelné jednotlivé věty, neobsahují množství pravopisných a stylystických chyb?

Jak vypadají pravopisné chyby snad není nutné komentovat. Problémem ale bývají nesrozumitelné formulace. Na Obrázku 5 je ukázka části analýzy, která nedává smysl. V první větě chybí zvratné zájmeno „se“ a kromě toho je dost těžké předstvit si, jak se na klientovi zobrazuje nějaká logika. Konec druhé věty také vyznívá poměrně komicky a o systému neříká vlastně nic.

> *Tato aplikace bude fungovat tak, že na serveru bude zpracovávat logika hry, která se následně bude zobrazovat na klientovi. Na klientovi tedy bude probíhat veškerá interakce s tímto systémem, tato interakce bude fungovat tak, že uživatel se nejdříve přihlásí na server a následně přihlásí.*

Obrázek 5: Nesrozumitelná analýza

1.1.6 Obsahuje analýza všechny části: specifikace zadání, datový model, stavová analýza, funkční analýza, analýza uživatelského rozhraní?

Kompletní analýza musí obsahovat všechny uvedené části. Diskutovat lze pouze o části Stavová analýza, která je povinná „pouze“ pokud lze u entit stavy definovat. Stavy lze ale definovat téměř vždy, takže berme stavovou analýzu jako povinnou.

1.1.7 Je možné dle analýzy implementovat systém tak, aby splňoval požadavky zadavatele?

Toto je jedna z nejdůležitějších otázek, které si musíme položit. Účelem analýzy je, aby podle ní bylo možné systém naimplementovat. Nejjednodušší kontrola, zda máte analýzu správně je ta, že si jí prohodíte např. s kolegou a pokusíte se ji částečně implementovat nebo alespoň rozmyslet, jak byste při implementaci postupovali. Pokud budete muset moc přemýšlet např. jak přesně máte vytvořit tabulky nebo jak máte naimplementovat určitou proceduru, pravděpodobně bude analýza neúplná.

1.2 Specifikace zadání

1.2.1 Je součástí popisu motivace, proč systém vzniká?

Součástí popisu by měl být alespoň nějaký fiktivní důvod, který vysvětluje, proč je vůbec potřeba informační systém vytvořit. Důvodem obvykle je, že určité firmě přestanou stačit jednoduché metody evidence jako např. používání excelovských tabulek.

1.2.2 Vyplývají z popisu uživatelské role?

Se systémem nejčastěji pracuje více typů uživatelů jako např. účetní, dělník, ředitel, administrátor apod. Specifikace zadání musí alespoň stručně naznačit, kteří uživatelé budou se systémem pracovat a jakou funkcionalitu budou využívat. Z toho potom vyplývá, jaké budeme v systému definovat role. Role a to, jaké části systému využívají, je možné zachytit pomocí Use-Case diagramu, který ale v tomto projektu není povinný. Na role se ale budeme odkazovat v části Funkční analýza. Je potřeba dát si pozor, aby to, co nadefinujeme v úvodu, korespondovalo s tím, co budeme používat ve funkční analýze. Tzn. pokud z úvodu vyplývá, že ze systémem pracuje např. skladník a účetní a potom ve funkční analýze řekneme, že určitou funkci bude používat ředitel, je někde něco špatně.

1.2.3 Mám v popisu napsáno, co bude do systému vstupovat?

Měli bychom stručně naznačit, že do systému budeme vkládat informace např. o studentech, zaměstnancích a katedrách.

1.2.4 Mám v popisu uvedeno, jaké budou výstupy, tj. co od systému očekávám?

Měli bychom stručně naznačit, co od systému očekáváme. Detailněji to potom rozebereme v části Funkční analzýa. Tzn. např. můžeme napsat, že systém bude poskytovat přehledy o měsíčních uzávěrkách objednávek. Funkční analýza pak bude obsahovat např. funkci „Uzávěrky“, která bude detailně popsána minispecifikací.

1.2.5 Mám v popisu uveden stručný popis netriviální funkcionality, který bude dále rozebrán v kapitole Funkční analýza (viz bod 1.5)?

Netriviální funkce představují obvykle hromadné transakce. Tzn. najednou manipulujeme s větším množstvím záznamů. Např. hromadně posíláme notifikační e-mail uživatelům, kteří už se dlouho nepřihlásili do systému. Detailněji funkce rozebíráme až v části Funkční analýza, nicméně už v úvodu by se stručný nástin toho, jaké funkce bude systém poskytovat, měl objevit.

1.3 Datový model

1.3.1 Obsahuje datový model relační E-R diagram (s cizími klíči)?

Ve cvičeních DAIS pracujeme pouze s relačním datovým modelem. Na projektu tedy vyžadujeme pouze E-R diagram relačního datového modelu, tj. ten, který obsahuje cizí klíče. Pro vytvoření takového E-R diagramu doporučujeme nástroj Oracle SQL Developer Data Modeler nebo přímo Microsoft SQL Management Studio. Konceptuální model v projekdu DAIS je nadbytečný.

1.3.2 Obsahuje datový model minimálně 7 tabulek, kde minimálně 4 nejsou číselníky?

Navrhnout model s minimálním počtem tabulek obvykle nebývá problém. Na minimálním počtu tabulek je ale často obžížné vymyslet netriviální funkce. Doporučeno je tedy spíše větší množství tabulek, např. 10. Číselníkem se rozumí tabulka, ve které předpokládáme spíše menší počet záznamů, přičemž obsah tabulky se v průběhu času příliš nemění. Číselník typicky obsahuje pouze atributy jako *id* a *název*. Jde např. o seznam škol, seznam skladů, seznam typů výrobků apod. Naopak např. tabulka objednávek číselník není.

1.3.3 Je datový model správně navržen?

Správnému návrhu datového modelu se věnuje předmět UDBS. Musíme si zkontrolovat, zda máme správně navržené tabulky (relace) a vztahy mezi nimi. U vztahů si musíme zkontrolovat kardinalitu (1:1, 1:N, M:N) a povinnost členství.

1.4 Stavová analýza

1.4.1 Je součástí analýzy stavová analýza, pokud se v databázi vyskytují entity, které to vyžadují?

V zadání projektu je uvedeno, že stavová analýza je vyžadována, pokud se v databázi vyskytují entity, které to vyžadují. Problémem ale je, že téměř vždy lze identifikovat nějaké stavy. Tzn. berme stavovou analýzu jako povinnou a snažme se vyspecifikovat stavy alespoň u dvou entit.

1.4.2 Je z popisu jasné, jak jednotlivé stavy souvisí s obsahem tabulek?

Častou chybou je, že ve stavové analýze jsou stavy pouze vyjmenované. Např. v analýze je uvedeno, že pro entitu *Objednávka* existují stavy: *vytvořená*, *odeslaná*, *zaplacená* a *stornovaná*. Už ale chybí to, jak jednotlivé stavy objednávky v databázi poznáme. Jsou dvě možnosti:

1. Stav entity je popsán určitým atributem v tabulce. Tzn. např. stav objednávky poznáme dle atributu `Objednavka.stav`, kde hodnota 1 představuje stav *vytvořená*, hodnota 2 stav *odeslaná* atd.
2. Stav entity může být dán stavem určitých atributů. Tzn. např. stav *zaplacená* znamená, že je vyplněn atribut `Objednavka.datum_uhrazeni`, stav *stornovaná* představuje atribut `Objednavka.storno` = 1.

Ukázku stavové analýzy najdeme v dokumentu *Ukázkový projekt do předmětu DAIS* na `dbedu.cs.vsb.cz`.

1.5 Funkční analýza

1.5.1 Je funkční analýza rozdělena na dvě podkapitoly – seznam funkcí a detailní popis funkcí?

Kapitola Funkční analýza bude jednoznačně rozělena na dvě podkapitoly: Seznam funkcí a Detailní popis funkcí. Nebude se tedy jednat o jednu kapitolu, kde bude seznam kombinovaný s popisem.

1.5.2 Jsou v seznamu funkcí uvedeny všechny funkce (včetně CRUD operací)?

Seznam funkcí musí obsahovat kompletní seznam všech funkcí v systému a to včetně CRUD (create, read, update, delete) operací. Funkce budou seskupené podle toho, s jakými tabulkami pracují. Funkce samozřejmě může využívat i více tabulek, pak ji začleníme do samostatné skupiny, nebo ji dle uvážení přidáme k některé z tabulek. Zjednodušeně řečeno, pokud budu mít v systému 10 tabulek, pak tento seznam bude obsahovat $10 \times 4 + 5$ funkcí. Tzn. pro každou tabulku 4 CRUD operace + 5 netriviálních funkcí. Je doporučeno netriviální funkce v seznamu zvýraznit (např. tučně), usnadňuje to pak kontrolu.

1.5.3 Je ze seznamu funkcí zjevné, které uživatelské role (viz bod 1.2.2) budou využívat které funkce?

Pro každou funkci v seznamu musí být jednoznačně vidět, jaká uživatelská role může tuto funkci využívat. Role můžeme uvést na úrovni tabulky (tzn. neuvádíme pak zvlášť pro každou CRUD operaci nebo netriviální funkci), výjimky (např. určitou funkci může navíc používat určitá role) dopíšeme k jednotlivým funkcím. Příklad vidíme na Obrázku 6.

1.5.4 Je ze seznamu funkcí u operací DELETE zjevné, co se má stát se souvisejícími záznamy?

Mějme v systému např. tabulky *Zákazník* a *Nákup* ve vztahu 1:N. U operací DELETE (tzn. např. funkce „Odstranění zákazníka“) musíme zvážit, jak ošetříme situaci, kdy zákazník již provedl nějaké nákupy. Většinou nastane jedna z možností:

2. Zaměstnanci
Role: *ředitel, administrátor, zaměstnanec*

2.1 Přidání zaměstnance
2.2 Zobrazení zaměstnance
2.3 Úprava zaměstnance (pouze role: *ředitel*, *zaměstnanec* pouze svůj záznam)
2.4 Odstranění zaměstnance (pouze role *administrátor*)

Obrázek 6: Příklad zápisu rolí v seznamu funkcí

1. **Kaskádové mazání** – Vazbě mezi tabulkami je možné nastatit tzv. kaskádové mazání (`ALTER TABLE Zamestnanec ADD FOREIGN KEY (...) REFERENCES (...) ON DELETE CASCADE`). Je to standardní funkcionalita relačního SŘBD. Kaskádové mazání znamená, že odstraníme-li zákazníka, SŘBD automaticky odstraní i všechny související nákupy. Pokud nám tato možnost vyhovuje, napíšeme do analýzy např.: „Při odstranění záznamu o zákazníkovi se pomocí kaskádového mazání odstraní také související záznamy o nákupu."

2. **Zamezení smazání** – Pokud to považujeme za vhodné, můžeme v analýze napst, že: „Jestliže zákazník již provedl nákup, nebude tato funkce dostupná."

3. **Nastavení příznaku o neaktivním záznamu** – V reálných systémech běžní uživatelé operaci DELETE obvykle neprovádějí. Systém musí být „blbuvzdorný". Nechtěným smazáním můžeme způsobit nenávratnou škodu. Proto se obvykle mazání nahrazuje nastavením nějakého atributu typu BIT (např. *Zákazník.aktivní*) na určitou hodnotu, např. 0. Tím určíme, že záznam už se nikde nebude zobrazovat (musíme podle toho samozřejmě patřičně upravit SELECT dotazy nad příslušnou tabulkou).

1.5.5 Je v seznamu funkcí alespoň 5 netriviálních funkcí, kde alespoň 3 jsou implementovány jako procedura (popř. trigger)?

Ideální stav je, když projekt obsahuje 3 netriviální funkce (viz bod 1.5.7) řešené formou uložené procedury a 2 netriviální funkce řešení formou komplexního dotazu. Pokud se rozhodneme pro netriviální funkci, která bude implementována triggerem, budeme ji v analýze popisovat jako proceduru, přičemž uvedeme, že procedura se bude automaticky spouštět např. při vkládání záznamu.

1.5.6 Neobsahuje seznam funkcí další slovní popis (např. zbytečný popis CRUD operací)?

Seznam funkcí bude obsahovat pouze:

1. hierarchický seznam všech funkcí v systému (vč. CRUD operací),
2. specifikaci, které uživatelské role mohou využívat které funkce,
3. co se má dít se souvisejícími záznamy u DELETE operací,
4. velmi stručný popis (max. 2 věty) u funkcí, kde je to potřeba.

Nebude zde žádný další popis. Pro snadnější kontrolu projektů je v seznamu funkcí doporučeno zvýraznit (např. tučným písmem) netriviální funkce.

1.5.7 Obsahují netriviální funkce implementované procedurou alespoň 3 body, které nelze dále zredukovat?

Za netriviální funkci se bude považovat procedura (případně trigger), která **obsahuje alespoň 3 body (příkazy), které už není dále možno redukovat**. Příklad triviální funkce vidíme na Obrázku 7.

Funkce vytvoří zaměstnance s daným jménem.
vstup: *$jmeno*, *$prijmeni*

1. Do tabulky vložíme záznam o zaměstnanci pomocí následujícího SQL příkazu:

   ```
   INSERT INTO Zamestnanec (jmeno, prijmeni) VALUES ($jmeno, $prijmeni)
   ```

Obrázek 7: Příklad triviální funkce

1.5.8 Nepředstavuje některá z funkcí kaskádový DELETE?

Jak již bylo uvedeno v bodě 1.5.4, relační SŘBD jako MS SQL Server nebo Oracle Database nabízí možnost kaskádového mazání. Mějme např. opět tabulky *Zákazník* a *Nákup*, mezi kterými je vazba 1:N. Někoho by mohlo napadnout vytvořit proceduru s parametrem *idZákazníka*, která nejprve odstraní všechny nákupy zákazníka a poté zákazníka samého. Takováto procedura ale nebude na projektu akceptována, protože jde jen o nahrazení kaskádového delete, který můžeme jednoduše aktivovat jako vlastnost vazby mezi zmíněnými tabulkami.

1.5.9 Jsou funkce implementované procedurou popsány v bodech minispecifikací?

Na minispecifikaci se v této analýze můžeme dívat jako na další procedurální jazyk po T-SQL a PL/SQL. Každý bod popisu téměž vždy povede k jednomu příkazu v T-SQL nebo PL/SQL. V minispecifikaci se ale **vyheme implementačním detailům** jako např.: deklarace proměnné, otevírání kurzoru nebo ošetřování výjimek. Body v popisu budou číslované. Z popisu bodu by mělo být naprosto jasné, o jaký příkaz půjde. Ukázku špatného popisu vidíme na Obrázku 8. Z popisu není zřejmé, co se myslí „načtením“. Pokud chceme např. uložit jméno a příjmení zaměstnance do určitých proměnných, pak správny popis bude vypadat např.: „Do proměnných *$jmeno* a *$prijmeni* uložíme výsledek dotazu `SELECT jmeno, prijmeni FROM ...`.“

1. Procedura načte zaměstnance pomocí příkazu:
   `SELECT * FROM Zamestnanec WHERE idZamestnance = $idZam`

Obrázek 8: Příklad nejasného bodu popisu

1.5.10 Je každý bod, se kterým souvisí nějaká operace (SELECT, INSERT, UPDATE, DELETE) doplněn o příslušný SQL příkaz?

V naší analýze je požadováno, abychom všude, kde je to možné, psali standardní SQL příkazy – SELECT, INSERT, UPDATE a DELETE (tedy to, co jsme se naučili v UDBS). Příklad, kde toto není splněno, vidíme na Obrázku 9. S prvním i druhým bodem evidentně souvisí SQL příkazy SELECT, které v popisu chybí.

1. Do proměnné *$jmeno* uložíme jméno zaměstnance s dle proměnné *$idZam*.
2. Do proměnné *$idNakupu* uložíme poslední ID nákupu.

Obrázek 9: Chybějící SQL příkazy

1.5.11 Nevyskytuje se v popisu žádná funkce popsáná pouze kódem T-SQL nebo PL/SQL?

Toto snad není potřeba popisovat. Pokud popíšeme funkci pouze kódem T–SQL nebo PL/SQL, tzn. zkopírujeme hotovou proceduru jako popis, je to špatně.

1.5.12 Nevyskytují se v popisu funkcí rysy specifické pro T-SQL nebo PL/SQL?

Anaýza musí být napsána tak, aby neobsahovala rysy specifické pro jazyk T-SQL nebo PL/SQL. Idea je taková, že teprve poté, co je analýza hotová, se rozhodneme, který systém budeme používat. Příklad této chyby vidíme na Obrázku 10. Chyb je zde několik:

1. `TOP 1` nám nebude fungovat na Oracle Database, jde o specifickou funkcionalitu MS SQL Serveru. Abychom se této chybě vyhli, můžeme jednoduše napsat: „Do proměnné *$idNakupu* uložíme první výsledek následujícího dotazu ...“
2. Funkce `GetDate()` je opět specifická pro MS SQL Server. Tento problém můžeme vyřešit tak, že místo `GetDate()` napíšeme např. *aktuální_datum*. Tak bude každému jasné, co chceme v analýze říci a vyhneme se specifickým rysům konkrétních jazyků.
3. Ve druhém bodě popisu je nesmyslně použitá konstrukce `SELECT ... INTO`, která je specifická pro PL/SQL.
4. V bodě č. 3 je nesprávně uveden příkaz `SET`. Nejen, že je tento příkaz specifický pro T-SQL, ale navíc nejde o žádný SQL příkaz (SELECT, INSERT, UPDATE nebo DELETE).

1. Do proměnné *$idNakupu* vložíme ID posledního nákupu, který byl vytvořen nejpozději před 30-ti dny, pomocí následujícího příkazu:
   ```
   SELECT TOP 1 idNakup FROM Nakup WHERE datum <= GetDate() - 30
   ```
2. Do proměnné *$v_idZamestnance* uložíme výsledek dotazu:
   ```
   SELECT idZamestnance INTO v_idZamestnance FROM Zamestnanec WHERE ...
   ```
3. Do proměnné *$v_pocet* uložíme číslo 0.
   ```
   SET $v_pocet = 0
   ```

Obrázek 10: Nesprávné použití specifických rysů PL/SQL nebo T-SQL

1.5.13 U funkcí implementovaných pomocí procedur, jsou součástí popisu vstupní (popř. výstupní) parametry?

Jelikož v popisu funkci pomocí minispecifikace používáme symboly proměnných, měli bychom proměnné používat i pro vstupní nebo výstupní parametry procedury. Ukázka problému je na Obrázku 11. Problémem zde je, že není vůbec jasné, co je to ten „daný“ zaměstnanec. V popisu v bodě 1 v příkazu INSERT pak vidíme použití nedefinovaných proměnných. Vše se vyřeší tím, že proměnné uvedeme jako vstupy jako to vidíme např. na Obrázku 7.

Funkce 4.2 Přidání zaměstnance
Funkce přidá daného zaměstnance a zkontroluje, zda nebyla porušena podmínka ...

1. Přidáme daného zaměstnance pomocí příkazu:
   `INSERT INTO Zamestnanec (jmeno, prijmeni) VALUES (`*`$jmeno`*`, `*`$prijmeni`*`)`
   ...

Obrázek 11: Chybějící parametry procedury

1.5.14 Jsou všechny body popisu funkcí nedělitelné, tj. nevyskytují se v popisu body, které obsahují třeba dvě SQL operace?

S jedním bodem popisu souvisí vždy maximálně jeden SQL příkaz SELECT, INSERT, UPDATE nebo DELETE nebo jedna konstrukce jako např. podmínka, cyklus apod. Ukázku chyby vidíme na Obrázku 12. V popisu na obrázku chybí SQL příkazy, což už je samo o sobě špatně (viz bod 5.10). Kdybychom SQL příkazy ale dopsali, budou určitě dva: jeden pro zjištění výšky daného zaměstnance *$v_idZam* a jeden pro zjištění průměrné výšky všech zaměstnanců. Správně bychom tedy takový bod měli rozepsat na 3 body: zjištění výšky daného zaměstnance (1 SQL dotaz), zjištění průměrné výšky všech zaměstnanců (1 SQL dotaz) a porovnání (podmínka).

1. Jestliže výška zaměstnance *$v_idZam* je menší než průměrná výška všech zaměstnanců, provedeme následující kroky:
   ...

Obrázek 12: Dále dělitelný bod popisu

1.5.15 Nevyskytuje se ve funkcích zbytečné použití kurzoru?

Častým problémem při implementaci procedur v T-SQL nebo PL/SQL je zbytečné použití kurzoru. Pravidlo je takové, že bychom měli vždy přednostně používat standardní SQL operace (SELECT, INSERT, UPDATE, DELETE). Z toho vyplývá mimo jiné, že pokud je možné vyhnout se použití kurzoru, tak se mu vyhneme. Kurzor zbytečně plýtvá systémovými prostředky. Ukázku této chyby vidíme na Obrázku 13. Situace by se velmi jednoduše dala nahradit pomocí `UPDATE Zamestnanec SET body = body + 1 WHERE rok > 1990`.
U operace INSERT nezapomeňme, že existuje zápis, kdy místo části `VALUES` napíšeme dotaz:

1. Připravíme kurzor procházející zaměstnance narozené po roce 1990.
   `SELECT idZam FROM Zamestnanec WHERE rok > 1990`
   Jednotlivá ID zaměstnanců postupně načítáme do proměnné *$v_idZam* a provádíme následující kroky:

   (a) Zaměstnanci zvýšíme bonusové body:
   `UPDATE Zamestnanec SET body = body + 1 WHERE idZam = $v_idZam`

Obrázek 13: Dále dělitelný bod popisu

```
INSERT INTO Zamestnanec (jmeno, prijmeni)
SELECT jmeno, prijmeni FROM Zakaznik
```

Uvedený příklad zkopíruje všechny zákazníky do zaměstnanců. Opět není nutné používat kurzor.

1.5.16 Jsou SQL příkazy zapsány bez syntaktických a sémantických chyb?

SQL příkazy související s body v popisu sice kombinujeme s pseudokódem, abychom se vyhli specifickým rysům T-SQL nebo PL/SQL, nicméně i tak by měly být zapsány v rámci možností syntakticky a sémanticky správně. Typickým problémem např. bývá, že není uvedena spojovací podmínka při spojení tabulek pomocí `JOIN`.

1.5.17 Je jasná vazba mezi seznamem funkcí a detailním popisem funkcí?

Části specifikace Seznam funkcí a Detailní popis funkcí se na sebe musí odkazovat. Detailní popis funkcí pouze upřesňuje Seznam funkcí. Ukázku problému vidíme na Obrázku 14. V Seznamu funkcí máme nadefinovanou funkci „2.3 Zvýšení platu zasloužilým zaměstnancům“. V části Detailní popis funkcí se pak nachází nějaká funkce „4.2.3 Odměna zasloužilých zaměstnanců“, která má zřejmě detailně popisovat funkci 2.3. Kvůli odlišenému očíslování a navíc odlišnému názvu je návaznost zjistitelná velice obtížně. Správně by tedy bylo, aby se i v části Detailní popis funkcí jmenovala funkce stejně, tj. „2.3 Zvýšení platu zasloužilým zaměstnancům“.

4.1 Seznam funkcí
...

2. Zaměstnanci

2.1 Přidání zaměstnance
2.2 Zobrazení zaměstnance
2.3 Zvýšení platu zasloužilým zaměstnancům
...

4.2 Detailní popis funkcí
...

4.2.3 Odměna zasloužilých zaměstnanců

Obrázek 14: Nejasná návaznost mezi seznamem funkcí a detailním popisem

1.5.18 Popisuje každá funkce pouze operace na straně SŘBD, tj. nejde o popis funkce v uživatelském rozhraní?

V této analýze se soustředíme na transakce probíhající na straně databáze. Nepopisujeme akce uživatelského rozhraní. Jednotlivé funkce ve funkční analýze pak vedou buď na CRUD operace nebo na uložené procedury a triggery. Příklad chyby vidíme na Obrázku 15. Popis v příkladu je sice srozumitelný a jasný, ale není to to, co nás v této analýze zajímá. Popis představuje interakci uživatele se systémem, ale nepopisuje databázovou transakci, která povede např. k implementaci uložené procedury.

1. Uživatel vyplní přihlašovací jméno a heslo do proměnných *$v_login* a *$v_heslo*.
2. Do proměnné *$v_hash* uložíme MD5 hash pro zadané heslo *$v_heslo*.
3. Zavoláme uloženou proceduru `spLogin` s parametry *$v_login* a *$v_hash*.
4. Jestliže procedura vrátí hodnotu 0, zobrazíme hlášku: „Nesprávné jméno nebo heslo.“

Obrázek 15: Nejasná návaznost mezi seznamem funkcí a detailním popisem

1.5.19 Jsou SQL příkazy přehledně odděleny od zbytku textu?

Kód SQL příkazů (připomínám, že v této analýze se objeví pouze kód SQL příkazů SELECT, INSERT, UPDATE nebo DELETE a žádný jiný) musí být viditelně odlišený od zbytku textu. Ukázka problému se nachází na Obrázku 16. Takto napsaná analýza je čitelná velmi obtážně. Příkazy SELECT a DELETE by měly být jednoznačně odlišeny neproporcionálním písmem (v LaTeXu pomocí \texttt{}, ve Wordu např. písmem Courier New nebo Consolas). Kromě toho je doporučeno SQL příkazy umístit na samostatný řádek. Podobná chyba se týká také zápisu proměnných, které by bylo také vhodné jednoznačně odlišit např. kurzívou nebo \emph{} a doplnit o sybmol např. $.

1. Do proměnné v_jmeno uložíme jméno zaměstnance pomocí SELECT jmeno FROM Zamestnanec WHERE login = v_login.
2. Smažeme všechny nákupy, které vyřizoval daný zaměstnanec pomocí DELETE FROM Zamestnanec WHERE login = v_login.

Obrázek 16: Příklad, kdy SQL příkazy nejsou odděleny od textu

1.5.20 Neobsahuje popis funkcí konstrukce smyšleného pseudokódu?

Minispecifikace je něco mezi slovním popisem a pseudokódem. Na jednu stranu bychom se měli vyvarovat zdlouhavému vykládání, na druhou stranu by popis neměl vypadat jako zdrojový kód. Příklad chyby je uveden na Obrázku 17. Je nežádoucí vymýšlet nějaké konstrukce pseudokódu jako `IF`, `WRITELINE` apod.
Obrázek 18 ukazuje, jak by se dala daná situace správně vyřešit.

1. Zvýšíme plat o *$v_prirustek* v případě *$v_prom* > 10. Jinak vypíšeme chybové hlášení.

```
IF $v_prom > 10 THEN
    UPDATE Zamestnanec SET plat = plat + $v_prirustek WHERE login = $v_login
ELSE
    WRITELINE 'Neocekavana chyba'
```

Obrázek 17: Příklad smyšleného pseudokódu

1. Pokud *$v_prom* > 10, aktualizujeme plat zaměstnance pomocí update:
   `UPDATE Zamestnanec SET plat = plat + $v_prirustek WHERE login = $v_login`
2. Pokud není splněn předchozí bod, vypíšeme hlášení: „Neočekávaná chyba“.

Obrázek 18: Příklad smyšleného pseudokódu

1.6 Analýza uživatelského rozhraní

1.6.1 Obsahuje analýza užiavatelského rozhraní dvě podkapitoly – Struktura menu a Návrh formulářů?

Stejně jako v bodě 1.5.1, kapitolu Analýza uživatelského rozhraní rozdělíme na dvě podkapitoly: Struktura menu a Návrh formulářů.

1.6.2 Je jasná vazba mezi strukturou menu a seznamem funkcí?

Funkce definované v kapitole Seznam funkcí můžeme rozdělit do 3 kategorií podle toho, odkud bude uživatel dané funkce spouštět:

1. **Funkce spouštěné přímo z menu** – jde především o CRUD operace jako např. *Seznam zaměstnanců*, *Vyhledání uživatele*, *Nová objednávka*.
2. **Funkce spouštěné z formulářů** – některé funkce jako např. *Ohodnocení zaměstnance* nebo *Přihození k aukci* nemusí být dostupné přímo z hlavní nabídky, ale až z nějakého dalšího netriviálního formuláře, např. pomocí tlačítka.
3. **Automaticky spouštěné funkce** – tyto funkce nespouští uživatel přímo, tyto funkce se spouští automaticky např. při vkládání záznamu (triggery) nebo podle nějakého plánu (např. vždy v noci v 1:00 hod.).

V podkapitole Struktura menu analyzujeme strukturu hlavní nabídky pomocí hierarchického číslovaného seznamu, přičemž bude jasné, která položka menu spouští kterou funkci z první kategorie ve výše uvedeném seznamu. Nestačí tedy pouze vypsat položky menu. Příklad vidíme na Obrázku 19.

7.1 Menu

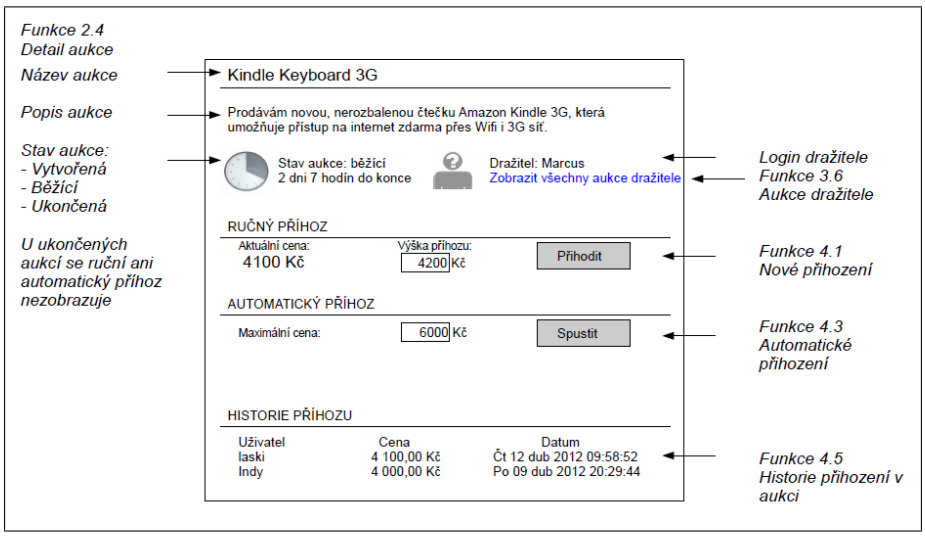
1. **Přehled aukcí** (*zodpovědnost*: Admin, Dražitel, Uživatel)
   (a) **Neskončené aukce** – akce: 3.1 Neskončené aukce
   (b) **Moje aukce** – akce: 3.6 Aukce dražitele
   (c) **Sledované aukce** – akce: 3.2 Sledované aukce
   (d) **Aukce s mým příhozem** – akce: 3.3 Aukce s příhozem
   (e) **Seznam vítězných aukcí bez komentáře** – akce: 3.4 Seznam vítězných aukcí bez komentáře
   (f) **Seznam vítězných aukcí** – akce: 3.5 Seznam vítězných aukcí

Obrázek z dokumentu *Ukázkový projekt do předmětu DAIS*, `dbedu.cs.vsb.cz`

Obrázek 19: Návrh uživatelského rozhraní – menu

1.6.3 Je jasná vazba mezi ovládacími prvky formuláře a seznamem funkcí?

Podobně jako v předchozím bodě – u návrhu jednotlivých netriviálních formulářů bude jasné, které funkce systému které ovládací prvky spouští. Tzn. nestačí pouze formulář nakreslit, nutné je také jej přehledně provázat s nadefinovanými funkcemi, jako to můžeme vidět např. na Obrázku



Obrázek z dokumentu *Ukázkový projekt do předmětu DAIS*, `dbedu.cs.vsb.cz`

Obrázek 20: Ukázka návrhu formuláře

1.6.4 Je součástí analýzy návrh alespoň dvou netriviálních formulářů?

Ukázku netriviálního formuláře vidíme na Obrázku 20. Netriviální formulář je formulář, který bude spouštět jednu nebo více netriviálních funkcí např. pomocí tlačítka. Návrh formuláře musí být přehledný. Účelem je,

abychom mohli formulář prezentovat zadavateli ještě před tím, než jej budeme implementovat.

2 Ukázky minispecifikace

Tato kapitola obsahuje několik vzorových minispecifikací dle kterých se můžete při tvorbě své analýzy inspirovat. Je pro připomenutí, minispecifikaci používáme k popisu funkcí, které budou implementovány jako procedury. V projektu by se měly objevit minispecifikace celkem 3. Další 2 netriviální funkce budou řešeny formou netriviálního dotazu.

2.1 Procedura *IsTall*

Tuto proceduru jsme tvořili v PL/SQL v rámci 2. cvičení. Pozor, funkce porušuje bod 1.5.15, protože celká funkce by se dala poměrně snadno napsat pomocí dvou příkazů UPDATE, kurzor není potřeba. Všimněte si, jakým způsobem je popsána transakce. Minispecifikace sama o sobě nebude obsahovat např. ošetření výjimky – to je implementační detail. Stačí do slovního popisu uvést, že funkce bude představovat transakci.

Funkce X.Y Nastavení příznaku *IsTall*

Vstup: *(bez vstupních parametrů)*
Funkce projde seznam studentů a aktualizuje příznak *IsTall* v závislosti na výšce studenta. Funkce bude představovat jednu transakci. Funkce bude spouštěna automaticky každý den v 01:00 hod.

1. Do proměnné *$v_avgTallness* uložíme průměrnou výšku studentů pomocí dotazu:
   `SELECT AVG(tallness) FROM Student`
2. Procházíme jednotlivé studenty pomocí kurzoru pro dotaz:
   `SELECT login, tallness FROM Student`
   Login a výšku postupně načítáme do proměnných *$v_login* a *$v_tallness* a provádíme následující kroky:
   (a) Jestliže *$v_tallness* < *$v_avgTallness* aktualizujeme příznak *IsTall* pomocí následujícího update:
   `UPDATE Student SET isTall = 0 WHERE login = $v_login`
   (b) V opačném případě provedeme update:
   `UPDATE Student SET isTall = 1 WHERE login = $v_login`

2.2 Reklamace výrobku zákazníkem

Tato procedura byla jedním z příkladů, co se objevily na realtime testu. Ve tředím bodě popisu si všimněme, jak se lze vyhnout zápisu `TOP 1`, který je specifický pro T-SQL.

Funkce X.Y Reklamace výrobku zákazníkem

Vstup: *$p_oznaceni, $p_jmeno*
Funkce nalezne poslední nákup daného produktu daným zákazníkem a na tento nákup naváže novou reklamaci, jejíž cena bude nastavena na cenu posledního nákupu.

1. Do proměnné *$v_pID* uložíme ID produktu dle *$p_oznaceni*:
   `SELECT pID FROM test.Produkt WHERE oznaceni = $p_oznaceni`
2. Do proměnné *$v_zID* uložíme ID zákazníka dle *$p_jmeno*:
   `SELECT zID FROM test.Zakaznik WHERE jmeno = $p_jmeno`
3. Do proměnných *$v_nID* a *$v_cena* uložíme ID a cenu posledního nákupu, tj. první výsledek následujícího dotazu:
   ```
   SELECT nID, cena
   FROM test.Nakup
   WHERE pID = $v_pID and zID = $v_zID
   ORDER BY den DESC
   ```
4. Pokud dotaz v předchozím bodě vrátí prázdný výsledek, procedura vypíše: „Nákup nelze nalézt“ a bude ukončena.
5. Do proměnné *$v_poradi* uložíme pořadí následující reklamace pomocí dotazu:
   `SELECT MAX(poradi) + 1 FROM test.Reklamace WHERE nID = $v_nID`
6. Pokud bude *$v_poradi* prázdné (tj. dotaz v předchozím bodě vrátí prázdný výsledek), nastavíme *$v_poradi* na hodnotu 1.
7. Vložíme nový záznam o reklamaci:
   ```
   INSERT INTO test.Reklamace (nID, poradi, cena)
   VALUES ($v_nID, $v_poradi, $v_cena);
   ```

2.3 Kontrola skladové zásoby

Opět jde o zpracování minispecifikace k jedné z testových úloh. Tentokrát jde o trigger. Trigger v analýze popíšeme jako proceduru, která se bude automaticky spouštět při určité akci, např. aktualizaci záznamu. V popisu si všimněme, jak je možné odkazovat se na vkládaný záznam a přitom nepoužívat tabulky `INSERTED` nebo `DELETED`, které jsou specifické pro T-SQL, nebo `:new` a `:old`, které jsou naopak specifické pro Oracle Database.

Funkce X.Y Kontrola skladové zásoby

Vstup: *$p_pID, $p_kusu*
Funkce zkontroluje, zda při vkládání nového záznamu o nákupu nedojde k přečerpání skladové zásoby produktu. Pokud ne, poníží skladovou zásobu produktu o daný počet kusů. Jinak vypíše chybové hlášení a k samotnému vložení záznamu o nákupu nedojde. Produkt a počet kusů jsou uloženy v parametrech *$p_pID* a *$p_kusu*.

1. Do proměnné *$v_skladem* uložíme aktuální skladovou zásobu produktu:
   `SELECT skladem FROM test.Produkt WHERE pID = $p_pID`
2. Pokud *$p_kusu* > *$v_skladem*, vypíšeme hlášku „Nízká skladová zásoba“, vyvoláme výjimku, která zamezí vložení záznamu, a proceduru ukončíme.
3. Jinak snížíme skladovou zásobu produktu:
   ```
   UPDATE test.Produkt
   SET skladem = skladem - $v_kusu
   WHERE pID = $p_pID
   ```

2.4 Tisková sestava nejoblíbenějších produktů

Jde o další minispecifikaci k jedné z testových úloh. Tentokrát budeme popisovat generování tiskové sestavy. Vytváření tiskové sestavy sice není typický úkol, který bychom řešili v procedurální nádstavbě SQL (tiskové sestavy se vytváří obvykle na straně klientské aplikace v systémech jako např. Crystal Reports nebo Microsoft Reports), nicméně v rámci procvičení minispecifikace je to dobrý příklad. U funkcí, kde očekáváme nějaký složitější (např. formátovaný) výstup, je vhodné uvést i příklad takového výstupu.

Funkce X.Y Nejoblíbenější produkty

Vstup: *(bez vstupních parametrů)*
Funkce vypíše tiskovou sestavu zákazníků, kde u každého zákazníka budou vypsány 3 jeho nejoblíbenější produkty, tj. produkty, které si koupil dohromady v největším množství.

1. Pomocí kurzoru pro dotaz:
   `SELECT zID, jmeno FROM test.Zakaznik`
   budeme procházet jednotlivé zákazníky. Jejich ID a jméno uložíme do proměnných *$v_zID* a *$v_jmeno* a postupně provedeme následující kroky:
   (a) Vypíšeme jméno zákazníka *$v_jmeno*.
   (b) Pomocí vnořeného kurzoru pro dotaz:
   ```
   SELECT test.Produkt.oznaceni, SUM(test.Nakup.kusu)
   FROM test.Produkt JOIN test.Nakup ON test.Produkt.pID = test.Nakup.pID
   WHERE test.Nakup.zID = $v_zID
   GROUP BY test.Produkt.pID, test.Produkt.oznaceni
   ORDER BY SUM(test.Nakup.kusu) DESC, test.Produkt.oznaceni
   ```
   procházíme produkty nakoupené daným zákazníkem. Projdeme pouze první 3 výsledky dotazu. Označení produktu a celkový počet nakoupených kusů ukládáme do proměnných *$v_oznaceni* a *$v_kusu* a provedeme tyto kroky:
      i. Vytiskneme řetězec dle následujícího vzoru:
      `"__$v_oznaceni_($v_kusu)"`

Výstup bude vypadat např. takto:

```
pepik
  WOS-50-K2 (10)
  WOS-10-K80 (8)
  Bongo Ultra 256 (5)
vinetu
  GEL-0006-7G (17)
  WAP 26 (11)
  HUP (8)
```

2.5 Nastavení prémiových bodů zákazníka

Pro změnu opět testová úloha. Minispecifikace porušuje bod 1.5.15 – bylo by možné vyhnout se použití kurzoru, nicméně, pro příklad popisu kurzoru zde tuto minispecifikaci uvádíme.

Funkce X.Y Nastavení prémiových bodů zákazníkovi

Vstup: *(bez vstupních parametrů)*
Výstup: *$p_aktualizovano*
Funkce nastaví jednotlivým zákazníkům prémiové body v závislosti na tom, kolik nákupů provedli. Výstupem bude proměnná *$p_aktualizovano* určující, u kolika záznamů funkce provedla změnu.

1. Proměnnou *$p_aktualizovano* nastavíme na hodnotu 0.
2. Pomocí kurzoru procházíme výsledky následujícího dotazu:
   ```
   SELECT zID
   FROM test.Zakaznik
   ```
   Jednová ID zákazníků načítáme postupně do proměnné *$v_zID* a provedeme následující kroky:
   (a) Do proměnné *$v_pocet* uložíme počet nákupů daného zákazníka:
   `SELECT COUNT(*) FROM test.Nakup WHERE zID = $v_zID`
   (b) Do proměnné *$v_aktualni_body* uložíme současný počet prémiových bodů:
   `SELECT premium FROM test.Zakaznik WHERE zID = $v_zID`
   (c) Pokud $\$v\_pocet < 3$, nastavíme proměnnou *v_body* na hodnotu 0.
   (d) Pokud $\$v\_pocet \in \{3, 4, 5\}$, nastavíme proměnnou *v_body* na hodnotu 1.
   (e) Pokud $\$v\_pocet > 5$, nastavíme proměnnou *v_body* na hodnotu 2.
   (f) Pokud došlo ke změně, tj. $\$v\_body \neq \$v\_aktualni\_body$, potom:
      i. Navýšíme hodnotu proměnné *$p_aktualizovano*.
      ii. Aktualizujeme body zákazníkovi:
      ```
      UPDATE test.Zakaznik SET premium = $v_body
      WHERE zID = $v_zID
      ```