VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

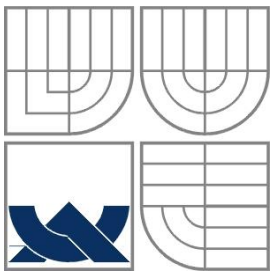
OBJEKTOVĚ RELAČNÍ DATABÁZE

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

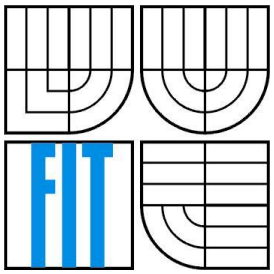
AUTOR PRÁCE
AUTHOR

ZDENKO FRANEK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OBJEKTOVĚ RELAČNÍ DATABÁZE

OBJECT RELATIONAL DATABASES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — ZDENKO FRANEK
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA CSc.
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Franek Zdenko**
Obor: Informační technologie
Téma: **Objektově relační databáze**
Kategorie: Databáze

Pokyny:

1. Seznamte se s problematikou objektově relačních databázových systémů.
2. Seznamte se s objektovými rozšířeními poskytovanými databázovým serverem Oracle a s podporou pro objektově relační modelování u nástroje IBM Rational Rose.
3. Po dohodě s vedoucím bakalářské práce navrhněte ukázkovou aplikaci, na které budete ilustrovat v co největší míře použití jednotlivých objektových rozšíření serveru Oracle 10g a podpory v IBM Rational Rose.
4. Ukázkovou aplikaci implementujte a ověřte.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Price, J.: Oracle Database 10g SQL. Oracle Press 2004.
- Application Developer's Guide - Object-Relational Features. 10g Release 2 (10.2). Oracle documentation. Available at http://download-uk.oracle.com/docs/cd/B19306_01/appdev.102/b14260.pdf.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Zdenko Franek**
Id studenta: 84399
Bytem: Rakovo 21, 038 42 Príbovce
Narozen: 19. 10. 1985, Martin
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP: Objektově relační databáze
Vedoucí/školitel VŠKP: Zendulka Jaroslav, doc. Ing., CSc.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Táto bakalárska práca sa zaoberá problematikou objektovo relačných databázových systémov. Popisuje objektovo relačný model definovaný podľa štandardov SQL, a následne využitie týchto štandardov v produkte databázy Oracle 10g. Zaoberá sa aj návrhom a implementáciou ukážkovej databázovej aplikácie, v ktorej by boli ilustrované jednotlivé objektové rozšírenia databázového servera Oracle 10g. Pre návrh databázy bol zvolený produkt Rational Rose Enterprise Edition, ktorý obsahuje podporu pre objektovo relačné modelovanie. Práca ďalej analyzuje, aké objektovo relačné rozšírenia obsahuje zvolený implementačný nástroj Oracle Forms 10g a využitie jazyka PL/SQL pri implementácii databázovej aplikácie.

Kľúčové slová

Objektovo relačný model, kolekcia, pole, vnorená tabuľka, OID, REF, PL/SQL, Oracle Forms.

Abstract

This bachelor thesis focuses on object-relational database systems. It describes object relational model defined according to standards SQL and utilization of these standards in Oracle 10g product. It also contains information about database analysis, design and implementation of sample application, in which specific aspects of object-relational model of Oracle 10g server would be illustrated. Rational Rose was chosen for database design, because it involves support for object-relational design. This thesis also analyzes, which object-relational extensions are included in implementation tool Oracle Forms 10g and utilization of PL/SQL language for implementation of database application.

Keywords

Object relational model, collection, array, nested table, OID, REF, PL/SQL, Oracle Forms.

Citácie

Franek Zdenko: Objektově relační databáze. Brno, 2008, bakalárska práca, FIT VUT v Brně.

Objektově relační databáze

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Doc. Ing. Jaroslava Zendulky CSc.
Ďalšie informácie mi poskytol Ing. Jaroslav Ráb.
Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Zdenko Franek
14.05.2008

Poďakovanie

Rád by som sa poďakoval vedúcemu práce Doc. Ing. Jaroslavovi Zendulkovi CSc. za jeho odborné rady a postrehy, ktoré mi pomohli pri tvorbe tejto práce.

Obsah

Úvod

Objektovo-relačný model vznikol ako alternatíva k tradičnému relačnému modelu v čase, keď sa plne ukazoval prínos objektového prístupu k programovaniu ako veľmi efektívny a prospešný v mnohých ohľadoch.

Cieľom tejto bakalárskej práce bolo vytvoriť databázu a následne aplikáciu založenú práve na objektovo relačnom modeli, v ktorej by som ukázal výhody využitia tohto prístupu od analýzy a návrhu databázy až po implementáciu databázovej aplikácie. V 2. kapitole objasňujem pojmy a kľúčové vlastnosti objektovo-relačného modelu podľa štandardov SQL a databázového systému Oracle 10g, na ktorom je moja aplikácia založená. V 3. kapitole vysvetľujem špecifické vlastnosti objektovo-relačného modelu Oracle 10g. Kapitola 4 sa zaoberá objektovou podporou pre vývojové prostredia Oracle. Analýza a návrh aplikácie v prostredí Rational Rose sú riešené v kapitolách 5 a 6. V záverečnej kapitole 7 sa zaoberám implementáciou aplikácie v prostredí Oracle Forms 10g s využitím programovacieho jazyka PL/SQL.

2 Objektovo relačný (OR) model

2.1 Vlastnosti OR modelu definované štandardmi SQL:1999 a SQL:2003

Štandard SQL z roku 1999 je nazývaný aj *objektovo orientovaný štandard SQL*, pretože prináša možnosti využitia objektovo orientovaných princípov v jazyku SQL. Štandard SQL:2003 obsahuje ďalšie rozšírenia objektovo orientovaných vlastností. Podrobnejšie informácie je možné nájsť v [1] a [2].

2.1.1 Užívateľsky definované dátové typy

Medzi najdôležitejšie objektovo orientované vlastnosti patrí možnosť vytvárania užívateľsky definovaných štruktúrovaných dátových typov (ŠÚDT). ŠÚDT obsahujú atribúty, ktoré reprezentujú vlastnosti typu a metódy, ktoré reprezentujú jeho správanie. SQL:1999 okrem ŠÚDT definuje aj jednoduché užívateľsky definované dátové typy *(distinct types)*, ktoré je však možné vytvárať iba zo vstavaných dátových typov SQL, akými sú napr. NUMBER, CHAR.

Najdôležitejšie vlastnosti ŠÚDT:

- môžu obsahovať jeden alebo viac atribútov buď dátového typu SQL, typu kolekcia, prípadne iného ŠÚDT (ľubovoľnej úrovne zanorenia)
- všetky aspekty správania ŠÚDT sú zabezpečené cez metódy, funkcie a procedúry
- *get* a *set* funkcie nemôžu byť preťažované, ostatné metódy a funkcie však áno
- porovnanie hodnôt ŠÚDT sa deje iba cez užívateľsky definované funkcie
- môžu vytvárať hierarchie typov, kde špecifickejšie typy tzv. podtypy majú všetky atribúty a metódy všeobecnejších typov a navyše môžu obsahovať aj nové
- podtyp môže dediť iba od jedného predka – jednoduchá dedičnosť

Príklad vytvorenia ŠÚDT:

```
CREATE TYPE emp_type UNDER person_type AS  -- špecifikácia nadtypu
(EMP_ID INTEGER,SALARY REAL)        -- atribúty ŠÚDT
INSTANTIABLE NOT FINAL              -- špecifikácia vlastností typu
INSTANCE METHOD GIVE_RAISE          -- definícia metódy ŠÚDT
(ABS_OR_PCT BOOLEAN,AMOUNT REAL)    -- parametre metódy
RETURNS REAL                        -- návratová hodnota metódy
```

Ako príklad je uvedený ŠÚDT *emp_type*, ktorý je vytvorený ako podtyp ŠÚDT *person_type*, obsahuje 2 atribúty - *EMP_ID* dátového typu INTEGER a *SALARY* dátového typu REAL. Ďalej je špecifikovaný ako *INSTANTIABLE*, tzn. je možné vytvárať inštancie daného typu a ako *NOT FINAL*, tzn. že je možné vytvárať podtypy daného typu. Obsahuje aj metódu *GIVE_RAISE* s dvoma parametrami a návratovou hodnotou dátového typu REAL.

2.1.2 Objekty

Napriek istým objektovým vlastnostiam ako napr. zapúzdrenie, nie je možné inštancie ŠÚDT považovať za ekvivalent výrazu objekt ako ho poznáme z objektovo orientovaných jazykov. Inštancia ŠÚDT je napriek svojej väčšej komplexnosti identifikovateľná iba svojou hodnotou podobne ako ostatné vstavané dátové typy.

Aby bolo možné inštancie ŠÚDT identifikovať, a tým pádom využívať referencie na tieto inštancie, štandard SQL:1999 nahrádza túto možnosť definovaním typovanej tabuľky nad daným ŠÚDT. Každý riadok obsahuje jeden stĺpec pre každý atribút daného ŠÚDT. Inštancie daného typu sú vlastne riadkami danej tabuľky – každý riadok má unikátnu hodnotu, ktorá je obdobou OID (objektového identifikátora) z objektovo orientovaných programovacích jazykov. Táto hodnota je unikátna v priestore (v databáze) a v čase (počas životného cyklu databázy).

SQL obsahuje špecifický dátový typ REF, ktorého hodnoty sú unikátne identifikátory. REF je vždy asociovaný k deklarovanému ŠÚDT a umožňuje odkazovať objekty v typovaných tabuľkách daného ŠÚDT.

Hodnota REF buď identifikuje objekt, čiže riadok v typovanej tabuľke (daného ŠÚDT) alebo neidentifikuje nič – napr. z dôvodu, že riadok, na ktorý REF odkazoval bol medzičasom vymazaný. V tomto prípade hovoríme o tzv. *dangling REF*.

2.1.3 Metódy

Niektoré vlastnosti, ktoré boli využité pri definícii objektovej časti štandardu SQL:1999 boli prvýkrát definované už v štandarde SQL/PSM publikovanom na konci roku 1996 – presnejšie išlo o možnosť volania funkcií a procedúr z príkazov jazyka SQL. SQL štandard z roku 1999 túto možnosť rozširuje – umožňuje volanie z SQL aj pre nový typ rutiny – metódy, ktoré sú dôležitou súčasťou objektovo orientovaného programovania.

Všeobecne povedané, metóda je funkcia s niekoľkými obmedzeniami a zlepšeniami:

a) metódy sú naviazané na jeden ŠÚDT, funkcia nie je.
b) ŠÚDT, ku ktorému je metóda naviazaná je dátovým typom význačného argumentu (prvý argument metódy, ktorý sa ale nedeklaruje). U funkcie takýto argument neexistuje.

c) funkcie umožňujú polymorfizmus ale špecifická funkcia je vybraná ešte počas prekladu preskúmaním deklarovaných typov každého argumentu funkcie a vybraním vhodnej funkcie (rovnaký názov a počet parametrov), metódy sú tiež polymorfické ale význačný argument umožňuje výber metódy až pred jej vykonaním za behu programu – ostatné argumenty metódy sú vyhodnotené ešte počas prekladu.
d) metódy musia byť uložené v tej istej schéme, kde aj definícia ich naviazaného ŠÚDT, funkcie nie sú v tomto smere limitované.

Prístup k atribútom ŠÚDT sa uskutočňuje dvoma možnými spôsobmi:

1. pomocou bodkovej notácie – `WHERE emp.salary > 10 000`
2. cez funkcie – `WHERE salary(emp)`

Oba typy zápisov sú štandardom SQL:1999 podporované. Pre volanie metód sa však používa iba prvý spôsob.

2.1.4 Kolekcie

SQL:1999 zachováva princíp ukladania dát do tabuliek, pričom však rešpektovanie 1. normálnej formy vyžadujúcej nedeliteľnosť stĺpcov tabuliek stráca na význame. Je to z dôvodu definície jednak ŠÚDT ale aj kolekcií, ktoré umožňujú ukladať množinu prvkov v rámci jedného stĺpca riadku tabuľky, v ktorej je daná kolekcia uložená.

2.1.4.1 Pole – ARRAY

Ide o zoradenú množinu prvkov rovnakého dátového typu. Vždy je definovaná maximálna kardinalita – buď užívateľom alebo implementačne definovaná. Pole je uložené do jediného stĺpca riadka tabuľky.

2.1.4.2 Multimnožina – MULTISET

Ide o implicitne nezoradenú množinu prvkov rovnakého dátového typu. Dátový typ prvkov kolekcie môže byť ďalšia kolekcia, čo umožňuje vytvárať rôzne úrovne zanorenia. Nie je nutné špecifikovať maximálnu kardinalitu, čo však neznamená, že je do kolekcie možné uložiť ľubovoľný počet prvkov.

2.2 OR model databázového systému Oracle 10g

Základné poznatky o objektoch Oracle:

- sú užívateľsky definované typy, ktoré umožňujú modelovať skutočné entity ako objekty v databáze (napr. entita *Zákazník*).
- objektová technológia Oracle je abstrakčná vrstva vytvorená na relačnej technológii Oracle.
- Nové objekty môžu byť vytvorené z ľubovoľného dátového typu, prípadne z predtým vytvoreného objektového typu, objektovej referencie a kolekcie.

2.2.1 Vlastnosti objektových typov Oracle

Objektový typ použitý v databázach Oracle je podobný mechanizmu tried v C++ a Java.
Triedy a objekty aj tu umožňujú jednoduchšie modelovať komplexné, reálne entity a logika a znovupoužiteľnosť objektov umožňuje vyvíjať databázové aplikácie rýchlejšie a efektívnejšie.
Najdôležitejšie vlastnosti objektových typov Oracle:

- efektívnejšia možnosť organizovania a prístupu k dátam
- dáta majú zmysel a podobu presne ako v reálnom svete, sú interpretované nie ako tabuľky a stĺpce ale ako reálne entity, hoci pod objektovou vrstvou sú predsa takto uložené
- možnosť pracovať s objektmi a objektovými možnosťami už aj v databáze relačnej, pretože sú navzájom kompatibilné

2.2.2 Výhody objektov Oracle

Zapúzdrenie – relačné databázové systémy obsahujú iba dáta. Objekty navyše obsahujú aj operácie, ktoré môžu nad týmito dátami pracovať, tzv. metódy (trieda *Zákazník* obsahuje metódu na výpis histórie nákupov a pod.).
Objekty sú efektívne – metódy sú uložené v databáze spolu s dátami, takže vývojári nemusia nanovo vytvárať podobné štruktúry pre manipuláciu s dátami. Je možné volať a manipulovať s množinou objektov ako s jediným objektom. Jeden príkaz volania objektu z databázy môže vrátiť iné objekty, ktoré sú s ním spojené. Napr. volanie objektu *Zákazník* spôsobí, že objekt nám vráti nielen jeho meno ale aj telefónne číslo a pod. A to všetko v jednom príkaze medzi klientom a serverom. Takisto, keď odkazujeme na stĺpec objektového typu, vráti sa nám celý obsah.
Objekty môžu reprezentovať vzťahy – objekt môže obsahovať iný objekt ako svoj atribút, čo nám umožňuje usporiadať objekty do vzťahov, namiesto zložitého využitia množstva tabuliek a primárnych a cudzích kľúčov.

3 Kľúčové vlastnosti OR modelu Oracle

Oracle implementoval objektový systém ako rozšírenie relačného modelu. Ostali zachované všetky možnosti a vlastnosti relačného systému ale navyše boli doplnené o také, ktoré umožňujú pracovať s objektmi. Viac informácií v [3].

3.1 Objektové typy

Objektové typy sú jedným z druhov dátových typov, to znamená že sa používajú podobne ako napr. natívne dátové typy databázových systémov Oracle. Rozdiel medzi nimi je napr. v tom, že objekty sú vytvárané výlučne vývojárom, nie sú v databáze preddefinované.

3.1.1 Uloženie objektov

Oracle automaticky mapuje komplexnú štruktúru objektových typov na jednoduchú štruktúru tabuliek.

Objektový typ je možné si predstaviť ako stromovú štruktúru, kde vetvy reprezentujú atribúty. Objektové atribúty vytvárajú ďalšie vetvy pre ich vlastné atribúty a tak ďalej.

Každá vetva končí buď natívnym dátovým typom ako napr. NUMBER, VARCHAR alebo typom kolekcia – pole s premenlivou dĺžkou, vnorená tabuľka. Tieto koncové – listové atribúty objektového typu sú uložené v stĺpcoch tabuľky.

Keď chceme získať alebo zmeniť atribút objektu, Oracle vykoná príslušné operácie na danom riadku tabuľky. K objektu sa pristupuje cez kópiu objektu, ktorá sa vytvorí pomocou prednastaveného konštruktora pre daný objektový typ, a to s použitím stĺpcov objektovej tabuľky ako argumentov konštruktora.

Zameniteľné objektové stĺpce (*substitutable columns*) alebo zameniteľné tabuľky (*substitutable object tables*) umožňujú uloženie nielen objektových typov, pre ktoré boli deklarované ale aj ľubovoľných podtypov daného typu. Aby bolo možné túto vlastnosť objektových stĺpcov a tabuliek Oracle využívať, musí zameniteľný stĺpec alebo zameniteľná objektová tabuľka obsahovať skryté stĺpce pre každý atribút nielen daného typu ale aj všetkých jeho podtypov. Po vytvorení podtypu sa skryté stĺpce atribútov pridajú nielen do tabuľky obsahujúcej daný objektový typ ale aj do tabuliek predkov, ktoré obsahujú objektové typy, z ktorých tento podtyp dedil. Keď podtyp odstránime, automaticky sa odstránia aj príslušné skryté stĺpce atribútov daného objektového typu.

3.1.2 NULL objekty a atribúty

Stĺpec tabuľky, atribút objektu, kolekcia alebo prvok kolekcie má hodnotu NULL, ak bol na túto hodnotu inicializovaný alebo nebol inicializovaný vôbec.

Objekt, ktorého hodnota je NULL sa nezhoduje s objektom, ktorého všetky atribúty sú inicializované na NULL, pretože u takéhoto objektu je stále možné tieto atribúty zmeniť.

3.1.3 Zámena typov v typovej hierarchii

Podtypy sú v typovej hierarchii varianty koreňového – základného typu. Niekedy je nutné pracovať s objektovým typom na všeobecnejšej úrovni, inokedy iba so špecifickými objektovými typmi. Polymorfizmus nám umožňuje nahradzovanie objektového typu ľubovoľným typom, ktorý je danému typu nadradený. Nahradzovanie je možné v atribútoch objektových typov, v stĺpcoch a riadkoch objektových tabuliek, či u referencií.

3.2 Metódy objektov

Metódy sú funkcie alebo procedúry, ktoré sa deklarujú v definícii objektového typu na implementáciu správania objektu daného typu.
Principiálna funkcia metód je umožniť prístup k dátam daného objektu. Je možné definovať metódy pre operácie, ktoré aplikácia bude vykonávať na dátach, takže aplikácia už nemusí obsahovať kód, ktorý s dátami takto manipuluje. Na uskutočnenie operácie aplikácia zavolá vhodnú metódu vhodného objektu.

3.2.1 Typy metód

Všeobecne je možné metódy rozdeliť na tieto typy:
Členské metódy (MEMBER) – aplikácia získava prístup k dátam inštancie objektu. Túto metódu definujeme pri definícii objektového typu pre každú operáciu, ktorú chceme, aby objekt daného typu vykonával. Tieto metódy obsahujú preddefinovaný parameter SELF, ktorý zabezpečí volanie metódy nad objektom, z ktorého je volaná.
Metódy pre porovnávanie objektov – hodnoty skalárnych dátových typov ako CHAR alebo REAL majú preddefinované poradie, čo dovoľuje ich vzájomné porovnanie. Naproti tomu, objektové typy nemajú preddefinovaný spôsob porovnávania, pretože obsahujú atribúty rôznych typov. Aby bolo možné tieto objekty porovnávať, je nutné špecifikovať bázu pre ich porovnávanie.
Inštancie objektov prípadne ukazovatele na objekty môžu byť porovnávané iba v tom prípade, ak sú rovnakého objektového typu alebo jeden je podtypom druhého objektového typu. Ak objektový typ nemá špecifikované metódy pre porovnávanie, je možné objekty porovnať iba v SQL výrazoch a iba na rovnosť či nerovnosť objektov. Existujú dva typy porovnávacích metód, pričom v každom objekte môže existovať iba jedna metóda z týchto dvoch typov.

MAP metódy - každý objektový typ môže mať najviac jednu MAP metódu, podtyp ju môže deklarovať, ak ju deklaroval rodičovský objektový typ. Ide o bezparametrovú funkciu typu MEMBER, ktorá vracia hodnoty skalárnych dátových typov ako napr. DATE, NUMBER, VARCHAR2. Táto funkcia je volaná automaticky na výpočet porovnaní objektov, prípadne vtedy keď je porovnanie vyvolané príkazmi DISTINCT, GROUP BY, UNION, ORDER BY. Volaním MAP metód je možné zoradiť ľubovoľný počet objektov.

Príklad deklarácie a definície MAP metódy:

```
CREATE TYPE obdlznik AS OBJECT (
dlzka NUMBER,
sirka NUMBER,
MAP MEMBER FUNCTION obsah RETURN NUMBER); -- deklarácia MAP metódy
/
CREATE TYPE BODY obdlznik AS
MAP MEMBER FUNCTION obsah RETURN NUMBER IS     -- definícia MAP metódy
BEGIN
      RETURN dlzka * sirka;         -- návratová hodnota MAP metódy
END obsah;
END;
/
```

Objektový typ *obdlznik* obsahuje atribúty *dlzka* a *sirka*, z ktorých je vypočítaná hodnota obsah, ktorá sa použije pri porovnávaní objektov daného objektového typu. Táto hodnota sa špecifikuje ako návratová hodnota danej MAP metódy.

ORDER metódy – uskutočňujú priame porovnanie objektov, pričom nevracajú žiadne hodnoty atribútov objektových typov nad ktorými sú volané, iba hovoria, či je daný objekt väčší, menší alebo sa rovná s objektom, s ktorým je porovnávaný. Ako parameter obsahujú objekt rovnakého dátového typu, akého je objektový typ, ktorý danú ORDER metódu špecifikuje. Návratová hodnota nadobúda tri typy hodnôt – záporné číslo, kladné číslo a nula, a to v závislosti na výsledku porovnania. ORDER metódy sú vhodné na použitie, keď sémantika porovnania by bola veľmi komplexná na použitie MAP metódy, napr. porovnávanie binárnych objektov.

Príklad deklarácie a definície ORDER metódy:

```
CREATE TYPE lokalita AS OBJECT (
cislo_domu NUMBER,
mesto VARCHAR2(40),
ORDER MEMBER FUNCTION zhoda (L lokalita) RETURN INTEGER ); -- deklarácia
/                                                          ORDER metódy
CREATE TYPE BODY lokalita AS
ORDER MEMBER FUNCTION zhoda (L lokalita) RETURN INTEGER IS -- definícia
BEGIN                                                      ORDER metódy
IF cislo_domu < L.cislo_domu THEN
      RETURN -1;                    -- záporná návratová hodnota
ELSIF cislo_domu > L.cislo_domu THEN
      RETURN 1;                     -- kladná návratová hodnota
ELSE
      RETURN 0;                     -- nula ako návratová hodnota
END IF;
END;
END;
/
```

Objektový typ *lokalita* obsahuje atribút *cislo_domu* a ORDER metódu *zhoda*, ktorá porovnáva polohu dvoch budov. Návratová hodnota ORDER metódy *zhoda* je záporná, kladná alebo nula, na základe vzájomnej polohy porovnávaných domov.

Statické metódy – sú volané nad objektovým typom, nie jeho inštanciou. Používajú sa pre operácie, ktoré sú globálne a nepotrebujú odkazovať na príslušné dáta konkrétnej inštancie.

Metódy typu konštruktor – slúžia na vytvorenie inštancie objektového typu. Konštruktor je vlastne funkcia, ktorá vracia novú inštanciu užívateľsky definovaného typu a nastaví hodnoty jej atribútov. Každý objektový typ má túto metódu implicitne definovanú ale je možné ju vytvoriť aj užívateľom. Implicitný konštruktor vyžaduje pri svojom volaní, aby všetky atribúty objektu boli inicializované, čo môže spôsobovať problémy, ak sme dodatočne definovali nejaký atribút – implicitný konštruktor, skončí s chybou, pretože s novým atribútom nepočíta a nie je možné ho nastaviť. Užívateľsky definovaný konštruktor nevyžaduje explicitne nastaviť všetky atribúty, v takom prípade sa tieto atribúty nastavia na hodnotu NULL.

Externe implementované metódy – je možné volať podprogramy, ktoré boli napísané v iných programovacích jazykoch, čo umožňuje využiť silné stránky týchto jazykov.

3.2.2 Preťažovanie a redefinícia metód

Preťažovanie sa využíva, ak chceme rôzne interpretovať spôsob vykonávania metód, ktoré majú rovnakú funkciu. V takomto prípade sa využíva možnosť pomenovať takéto metódy rovnakým názvom, pričom sa musia líšiť v type argumentov, počte argumentov alebo type návratovej hodnoty, aby prekladač vedel rozlíšiť, ktorú z daných metód použiť.

Podtyp môže redefinovať metódy ktoré zdedil, prípadne môže pridať metódy nové. Redefinícia znamená implementovať zdedenú metódu iným spôsobom, aby vykonávala odlišnú činnosť ako v rodičovskom type. Ak má objektový typ, ktorý redefinuje metódu ďalšie podtypy, tak v týchto sa v prípade, že už nedošlo k ďalšej redefinícii, vykonáva metóda zdedená od najbližšieho rodiča.

Aby sme zabezpečili redefiníciu metódy, musíme ponechať metóde presne taký istý zápis ako v rodičovskom objektovom type.

Poznáme dva typy redefinície metód:

1. **prekrytie (overriding)** - redefinovaná metóda je typu MEMBER, vyžaduje dynamické typovanie – o type metódy sa rozhoduje za behu programu
2. **skrytie (hiding)** - redefinovaná metóda je typu STATIC, typ metódy sa určí pri preklade

3.3 Dedičnosť

Dedičnosť umožňuje vytvárať hierarchiu typov, potomkovia dedia vlastnosti od predkov a navyše majú svoje vlastné špecifické vlastnosti – nové atribúty a metódy. Taktiež je možné u potomkov redefinovať metódy predka.

Každý objekt je odvodený od nadtypu, ktorý je možné definovať ako základný objektový typ. Typová hierarchia umožňuje vyšší stupeň abstrakcie pre riadenie komplexnosti aplikačného modelu.

Podtypy sú v hierarchii spojené s nadtypmi dedičnosťou, to znamená, že podtypy automaticky získavajú atribúty a metódy rodičovského typu. Taktiež to znamená, že v podtypoch sa prejavia zmeny, ktoré sa udejú v nadtype.

Podtyp je špeciálna verzia rodiča. Vznikne pridaním nových atribútov a metód prípadne redefinovaním existujúcich metód rodiča. Redefinícia metódy dáva podtypu možnosť vykonať metódu svojím spôsobom.

Podtyp môže dediť od nadtypu buď priamo alebo nepriamo cez ďalšie podtypy. Podtyp môže priamo dediť iba od jedného rodiča, nadtyp môže obsahovať viac podtypov – ide o tzv. jednoduchú dedičnosť.

Vzťah dedičnosti medzi nadtypom a jeho podtypmi dáva veľké možnosti pre vývojára – tabuľka alebo stĺpec môžu obsahovať ľubovoľný typ v hierarchii. Je možné obmedziť DML príkazy a dopyty, aby vyberali z typovej hierarchie iba rozsah typov, ktoré potrebujeme. Je nutné však uvažovať, v akých prípadoch danú možnosť využiť.

Inštancia objektu podtypu môže byť vo všeobecnosti nahradená inštanciou objektu ľubovoľného nadtypu – táto vlastnosť sa nazýva polymorfizmus.

3.4 Objektové tabuľky

Ide o špeciálny typ tabuľky, v ktorej každý riadok reprezentuje objekt. Na takúto tabuľku existujú dva pohľady:

1. jednostĺpcová tabuľka, v ktorej každý riadok je objektového typu umožňujúci vykonávať objektovo orientované operácie
2. viacstĺpcová tabuľka, v ktorej každý atribút obsadzuje jeden stĺpec tabuľky a umožňujúca vykonávať relačné operácie

Objekty, ktoré sú uložené v kompletných riadkoch v objektovej tabuľke sú riadkové objekty.
Uložené ako atribúty iných objektov sú stĺpcové objekty.
Indexy je možné definovať na stĺpec koncového atribútu objektu v objektovej tabuľke alebo objektu v úložnej tabuľke vnorenej tabuľky. Na REF je možné vytvoriť index iba keď je obmedzená na jednu tabuľku.

3.5 Objektové pohľady

Objektové pohľady vytvárajú spôsob, ako sprístupniť relačné dáta pomocou objektovo orientovaných možností. Umožňuje vyvíjať objektovo orientované aplikácie bez zmeny relačnej schémy.

3.6 Referencie

REF je logický ukazovateľ na riadkový objekt, ktorý je vytvorený z objektového identifikátora (OID) a jedná sa o preddefinovaný dátový typ Oracle. Referencie vytvárajú jednoduchý mechanizmus pre navigáciu medzi objektmi, pretože redukujú nutnosť použitia cudzích kľúčov. REF je možné použiť na zistenie, na aký objekt ukazovateľ odkazuje, prípadne získať samotný objekt alebo ho zmeniť.

3.6.1 Uloženie OID – objektového identifikátora

Každý riadkový objekt má implicitne pridelenú unikátnu hodnotu – OID, ktorá identifikuje objekt v objektovej tabuľke. Hodnota OID je uložená v skrytom stĺpci tabuľky. OID umožňuje korešpondujúci riadkový objekt odkazovať z iných objektov alebo relačnej tabuľky. OID môže byť systémovo generované alebo založené na primárnom kľúči:

Systémovo generované OID - Ide o 16 bajtovú hodnotu, ktorá je automaticky indexovaná pre efektívny prístup založený na OID – vyhľadanie, dopytovanie a pod. Každý riadkový objekt, ktorý má systémovo generované OID, vyžaduje 16 bajtov pamäťového priestoru pre uloženie OID pre každý riadkový objekt.

OID založené na primárnom kľúči – nie je potrebné vytvárať stĺpec pre systémovo generované OID a tým sa šetrí úložný priestor, pretože stĺpec s hodnotou primárneho kľúča iba zriedka prekročí veľkosť 16 bajtov. Hodnoty OID založené na primárnom kľúči sú lokálne unikátne hodnoty, globálne pre databázu to však platiť nemusí – je nutné ošetriť, prípadne použiť systémovo generované OID. Ak sú hodnoty v stĺpcoch primárneho kľúča väčšie ako 16 bajtov a používame veľké množstvo referencií, použitie OID založeného na primárnom kľúči môže byť pamäťovo náročnejšie ako systémovo generované OID.

3.6.2 Uloženie REF

Keď Oracle vytvára odkaz na riadkový objekt, REF obsahuje OID, ďalej metadáta objektovej tabuľky , v ktorej je uložený odkazovaný objekt, nepovinne aj ROWID. Veľkosť REF v stĺpci objektovej tabuľky závisí na vlastnostiach uloženia daného stĺpca.

Ak je OID uložené v rámci REF na báze primárneho kľúča, Oracle môže vytvoriť jeden alebo viac interných stĺpcov pre uloženie hodnôt primárneho kľúča závisiac na tom, koľko stĺpcov vytvára primárny kľúč.

Dátový typ REF obsahuje 3 komponenty:

1. **OID odkazovaného objektu** – 16 bajtová hodnota pri systémovo generovanom OID, pri OID založenom na primárnom kľúči záleží na veľkosti hodnoty primárneho kľúča
2. **OID tabuľky alebo pohľadu obsahujúcich odkazovaný objekt** – 16 bajtová hodnota
3. ***ROWID hint*** – 10 bajtová hodnota, ak je špecifikované uloženie ROWID v REF, nie je nutné hodnotu ROWID získať zo stĺpca OID ale je dostupná priamo, čo je rýchlejší prístup k vyhľadaniu odkazovaného objektu ale vyžadujúci viac úložného priestoru pre uloženie REF

3.6.3 Obmedzené referencie (SCOPED REF)

REF stĺpec môže byť obmedzený použitím SCOPED – keď je toto obmedzenie deklarované, môže REF ukladať objektové referencie na riadkové objekty uložené práve v jednej objektovej tabuľke, ktorá však musí byť definovaná nad takým istým objektovým typom, na aký je deklarovaná obmedzená referencia.

Tento typ referencie zaberá menej pamäťového priestoru a umožňuje efektívnejší prístup k dátam.

3.6.4 Výkonnosť a uloženie obmedzených referencií

Obmedzené referencie sú uložené efektívnejšie. Kým REF zaberá min. 32 bajtov na uloženie, obmedzená referencia, čiže SCOPED REF, ukladá iba OID cieľového objektu bez nutnosti ukladania OID objektovej tabuľky obsahujúcej odkazovaný objekt, keďže táto tabuľka je špecifikovaná pri definícii obmedzenej referencie. Obmedzená referencia preto môže zaberať 16 bajtov, ak OID, ktoré je jej obsahom je systémovo generované alebo zaberá veľkosť ekvivalentnú veľkosti stĺpca primárneho kľúča, ak je založená na primárnom kľúči. Okrem toho obmedzené referencie umožňujú optimalizátoru optimalizovať dopyty, ktoré dereferencujú tieto referencie, na efektívnejšie operácie typu JOIN. Nezaručujú však, že odkazovaný objekt existuje ale zaručujú, že existuje tabuľka, v ktorej sa objekt nachádza.
Neobmedzené referencie sú užitočné, ak návrh aplikácie vyžaduje odkazovať objekty rozložené vo viacerých tabuľkách.

3.6.5 Indexovanie obmedzených referencií

Index sa vytvára pomocou príkazu CREATE INDEX na stĺpec dátového typu *SCOPED REF*. Indexovanie tohto stĺpca umožňuje využiť index na efektívnejšie vyhodnocovanie dopytov, ktoré dereferencujú obmedzenú referenciu. Takéto dopyty sú zmenené implicitne na operácie typu JOIN.

3.7 Kolekcie

Na modelovanie viachodnotových atribútov a vzťahov N – N, Oracle podporuje 2 typy kolekcií - **vnorené tabuľky** (*nested tables*) – nezoradená kolekcia prvkov a **polia s premenlivou dĺžkou** (*varray*) – zoradená kolekcia prvkov.
Ak potrebujeme uložiť nemenný počet prvkov alebo potrebujeme mať prvky uložené v istom poradí za sebou alebo potrebujeme často manipulovať s celou kolekciou ako s hodnotou – použijeme pole.
Ak chceme spúšťať efektívne dopyty nad kolekciou, spracovávať ľubovoľné hodnoty prvkov alebo využívať operácie pre masové vkladanie, aktualizovanie alebo mazanie údajov – použijeme vnorenú tabuľku.

3.7.1 Vytvorenie kolekcií

Vytvorenie kolekcie prebieha rovnako ako vytvorenie iného objektového typu – volaním príslušného konštruktora. Prvky špecifikujeme ako zoznam jeho argumentov oddelených čiarkami. Volanie metódy konštruktor s prázdnym zoznamom argumentov, vytvorí prázdnu kolekciu daného typu.

3.7.2 Pole s premenlivou dĺžkou (VARRAY)

Pole je usporiadaná množina dátových prvkov – elementov, pričom všetky elementy sú zhodného dátového prípadne objektového typu alebo jeho podtypu. Každý element má index, ktorého hodnota korešponduje s jeho pozíciou v poli. Hodnota indexu sa používa pre prístup ku konkrétnemu elementu uloženému v poli.

Pole môžeme definovať ako:

- dátový typ stĺpca relačnej tabuľky
- atribút objektového typu
- PL/SQL premennú, parameter, návratovú hodnotu funkcie

3.7.2.1 Uloženie polí

Polia sú uložené v jednom stĺpci tabuľky. Pokiaľ ich veľkosť presiahne určitú hodnotu (4000 bajtov), je uložená ako *BLOB – Binary Large Object*.

Veľkosť uloženia poľa záleží len na aktuálnom počte prvkov v poli nie na maximálnom počte prvkov, ktorý môže pole obsahovať.

3.7.2.2 Uloženie viacúrovňových polí

Viacúrovňové polia sú uložené 2 spôsobmi záležiac na tom, či ide o pole polí alebo pole vnorených tabuliek.

Pole polí – celé pole je uložené v stĺpci tabuľky, pokiaľ neprekročí veľkosť 4000 bajtov alebo pokiaľ nie je explicitne špecifikovaný dátový typ *BLOB*

Pole vnorených tabuliek – celé pole je uložené v *BLOB,* v riadku tabuľky je uložený len *LOB locator*

3.7.3 Vnorené tabuľky (NESTED TABLES)

Vnorená tabuľka je neusporiadaná množina dátových elementov, ktoré sú zhodného dátového typu. Pri definícii nie je nutné narozdiel od poľa špecifikovať počet prvkov, ktoré bude vnorená tabuľka obsahovať, čo však neznamená, že môže obsahovať ľubovoľný počet elementov.

Elementy vnorenej tabuľky sú v skutočnosti uložené v oddelenej tabuľke – úložnej tabuľke, ktorá obsahuje stĺpec identifikujúci rodičovský riadok v tabuľke, do ktorého daná vnorená tabuľka patrí. Vnorená tabuľka obsahuje jediný stĺpec, ktorý môže byť skalárneho dátového typu alebo objektového typu. Ak je objektového typu, je možné sa naň dívať ako na viacstĺpcovú tabuľku, v ktorej každý atribút zaberá jeden stĺpec.

Vnorenú tabuľku môžeme definovať ako:

- dátový typ stĺpca relačnej tabuľky – dáta vnorenej tabuľky sú uložené priamo v relačnej tabuľke

- atribút objektového typu – dáta vnorenej tabuľky sú uložené v osobitnej tabuľke asociovanej s objektovým typom
- PL/SQL premennú, parameter, návratovú hodnotu funkcie

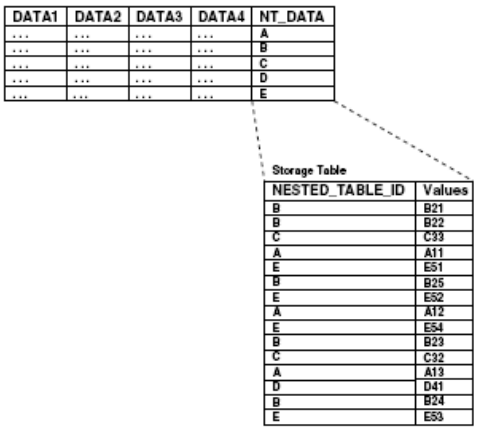
3.7.3.1 Uloženie vnorených tabuliek

Vnorená tabuľka môže obsahovať objekty alebo skalárne hodnoty:

a) **ak sú to objekty** – úložná tabuľka je obdobou objektovej, tzn. že na každý atribút pripadá jeden stĺpec ale pretože riadky vnorenej tabuľky nemajú skrytý stĺpec s hodnotou OID, nie je možné ich odkazovať cez REF

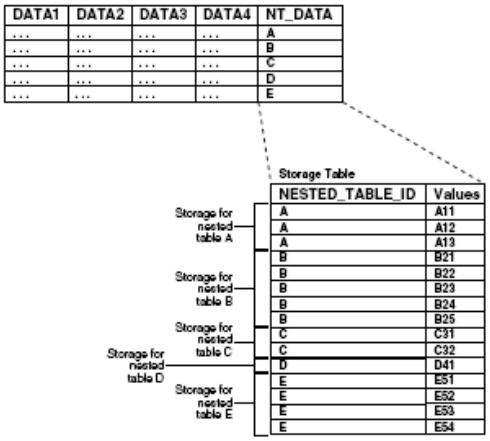
b) **ak sú to skaláry** – úložná tabuľka obsahuje jediný stĺpec, ku ktorému je možné pristupovať cez príkaz COLUMN VALUE, ktorý obsahuje danú skalárnu hodnotu

Oracle ukladá riadky vnorených tabuliek pre jednotlivé objekty objektovej tabuľky v separátnej úložnej tabuľke, ktorá obsahuje stĺpec so systémovo generovanou hodnotou NESTED_TABLE_ID (16 bajtová). Táto hodnota sa používa pre zistenie konkrétnej vnorenej tabuľky, ktorá sa nachádza v rodičovskej objektovej tabuľke.



Obrázok 3.1: Organizácia úložnej tabuľky vnorenej tabuľky

Ak vnorená tabuľka obsahuje stĺpec s primárnym kľúčom, je možné organizovať vnorenú tabuľku ako IOT – *Index Organized Table*. Oracle v tomto prípade združuje hodnoty jednotlivých vnorených tabuliek dokopy, na základe hodnoty NESTED_TABLE_ID, ktorá však musí byť špecifikovaná ako súčasť primárneho kľúča.



Obrázok 3.2: Organizácia úložnej tabuľky vnorenej tabuľky ako IOT

Oracle odporúča uloženie vnorenej tabuľky ako IOT s NESTED_TABLE_ID ako prefixom primárneho kľúča, pretože vtedy je možné aplikovať kompresiu na IOT a zároveň využiť efektívnejší prístup k hodnotám vnorených tabuliek objektov.

3.7.3.2 Uloženie viacúrovňových vnorených tabuliek

Pri viacúrovňových vnorených tabuľkách musíme špecifikovať úložnú tabuľku pre každú úroveň zanorenia. Každá úložná tabuľka obsahuje stĺpec, pomocou ktorého je možné odkazovať na rodičovskú tabuľku – odkazovať sa naň môžeme pomocou NESTED_TABLE_ID. Rodičovská tabuľka obsahuje skrytý stĺpec *associated row*, ktorý umožňuje identifikáciu tejto tabuľky tabuľkou úložnou. Ak je úložná tabuľka vnorenej tabuľky zároveň aj rodičovskou tabuľkou pre inú vnorenú tabuľku, obsahuje oba stĺpce, to znamená stĺpec NESTED_TABLE_ID a aj skrytý stĺpec *associated row*.

Je možné zanorovať kolekcie aj nepriamo pomocou REF – napr. vytvorenie vnorenej tabuľky objektového typu, ktorá má atribút odkazujúci na objekt, ktorý má atribút vnorenú tabuľku alebo pole. Ak v skutočnosti nie je potrebné pristupovať ku všetkým elementom viacúrovňovej kolekcie, potom zanorovanie pomocou referencií môže dávať lepšie výsledky, pretože sa pracuje iba s referenciami a nie so samotnými elementmi, na ktoré tieto referencie odkazujú. Pravé viacúrovňové kolekcie (špeciálne viacúrovňové vnorené tabuľky) dávajú lepšie výsledky pre dopyty, ktoré pristupujú k individuálnym elementom kolekcie.

4 Objektová podpora pre vývojové prostredia Oracle

4.1 SQL

Jazyk SQL obsahuje nasledujúce objektovo-relačné možnosti:

- špecifikuje privilégiá
- špecifikuje tabuľkové stĺpce objektových typov
- umožňuje vytváranie objektových tabuliek
- definuje objektové typy, vnorené tabuľky, polia, referencie
- umožňuje dopytovanie a aktualizáciu objektov a kolekcií

4.2 Oracle call interface

Oracle call interface (OCI) je množina funkcii z knižníc jazyka C, ktoré aplikácie môžu použiť na prácu s dátami a schémami v databáze. OCI podporuje tradičný 3GL ako aj OO prístup k databáze. Dôležitý komponent OCI je množina volaní, ktoré spravujú pracovný priestor nazvaný objektová vyrovnávacia pamäť. Je to pamäťová oblasť na strane klienta, ktorá umožňuje programom uložiť objekty a vyberať spomedzi nich bez ďalších volaní servera. Object cache je pod úplnou kontrolou aplikačného programu, ktorý ho používa. Server Oracle k nej nemá prístup.
OCI zabezpečuje tieto funkcie:

a) sprístupnenie objektov na serveri použitím SQL
b) sprístupnenie, manipulácia a riadenie objektov v object cache s použitím ukazovateľov alebo referencií.
c) konverzia dátumov, reťazcov a čísel do dátových typov jazyka C
d) riadenie veľkosti pamäťových oblastí object cache

OCI programátori môžu použiť prekladač typov na generovanie dátových typov jazyka C korešpondujúcich k dátovým typov Oracle.

4.3 Pro*C/C++

Oracle Pro*C/C++ prekladač dovoľuje programátorom použiť užívateľsky definované dátové typy v C a C++ programoch.

Prístup k objektom na serveri sa deje dvojako:

1. cez SQL príkazy a PL/SQL funkcie a procedúry definované v Pro*C/C++
2. rozhraním do objektovej vyrovnávacej pamäte

4.4 Oracle C++ call interface

Oracle C++ call interface (OCCI) je API, ktoré umožňuje použiť objektovo orientované vlastnosti, natívne triedy a metódy jazyka C++ pre prístup k databáze Oracle. Toto rozhranie takisto umožňuje prístup a modifikáciu objektovo relačných dát vo forme objektov C++ bez použitia SQL.

4.5 Oracle objects for OLE

Oracle objects for OLE (OO4O) zabezpečuje plnú podporu pre prístup a prácu s inštanciami referencií, hodnotovými inštanciami, poliami s premenlivou dĺžkou a vnorenými tabuľkami v databázovom serveri Oracle. Na vytvorenie objektovo orientovaných dátových programov sa môže vo Windows použiť Visual Basic alebo iné prostredia podporujúce COM protokol, ako napr. ActiveX, Excel, Active server pages a pod.

4.6 Java : JDBC, SQLJ

Java je silný, moderný objektovo orientovaný programovací jazyk, ktorý umožňuje implementáciu jednoduchých, efektívnych, prenosných a bezpečných aplikácií. Oracle umožňuje 2 spôsoby ako integrovať jeho objektové vlastnosti do jazyka Java. Ide o JDBC a SQLJ. Tieto 2 rozhrania umožňujú prístup k databáze z Javy a perzistentné uloženie dát v databáze pre Java objekty.

4.6.1 JDBC

JDBC (Java database connectivity) je množina rozhraní pre Oracle server. Oracle vyvinulo úzku integráciu medzi objektmi Oracle a JDBC:

- umožňuje prístup k objektom a kolekciám definovaných v databáze v programoch Java pomocou dynamického SQL
- prekladá objektové typy definované v databáze na triedy jazyka Java pomocou prednastaveného alebo upraveného mapovania

JDBC vo verzii 2.0 podporuje objektovo-relačné konštrukcie ako napr. užívateľsky definované objektové typy. JDBC spracúva objekty Oracle ako inštancie príslušnej triedy Java.

Použitie JDBC zahŕňa 2 prístupy spracovania objektov Oracle:

- nechať JDBC spracovať objekt ako štruktúru, v tomto prípade JDBC vytvorí Java triedy a naplní ich
- manuálne špecifikovať mapovanie medzi triedami Java a objektovými typmi Oracle

4.6.2 SQLJ

SQLJ umožňuje prístup k objektom použitím SQL príkazov včlenených do Java kódu. Je možné použiť užívateľsky definované dátové typy v programoch Java. Na mapovanie Oracle objektov na Java triedy je možné využiť JPublisher.
Takisto je možný aj opačný prístup k údajom, to znamená vytvoriť SQL objektové typy Oracle a tie mapovať do existujúcich tried v Jave. Takéto objektové typy voláme SQLJ objektové typy. Táto vlastnosť umožňuje perzistentné uloženie Java objektov v databáze.

4.7 XML

Pohľady typu XML zabaľujú existujúce relačné a objektovo-relačné dáta v XML formáte. Sú podobné objektovým pohľadom. ID objektu pre unikátnu identifikáciu každého riadku v pohľade môže byť vytvorená ako výraz nad hodnotami typu XML.

4.8 PL/SQL a Oracle Forms

Objektové typy a podtypy môžu byť použité v PL/SQL procedúrach a funkciách na väčšine miest kde sa vyskytujú preddefinované typy. Parametre a premenné funkcií a procedúr v PL/SQL môžu byť objektového typu [4].
Oracle Forms je tradičným vývojovým prostredím pre vývoj databázových aplikácií vytvoreným firmou Oracle. Je súčasťou balíka vývojových prostredí Oracle Developer. Pre programovanie aplikácií využíva jazyky Java a PL/SQL. Obsahuje mnohé funkcie, ktoré uľahčujú programovanie databázovej aplikácie pracujúcej nad databázou Oracle. Aktuálna verzia 10g však neobsahuje takmer žiadnu podporu pre objektovo relačné programovanie. Umožňuje napr. vytvárať pomocou sprievodcov objektové typy, ktoré sa priamo ukladajú do zvolenej databázy alebo pomocou dátových sprievodcov dopytovať údaje z objektových tabuliek. Priama podpora pre manipuláciu s kolekciami, referenciami, či už ide o vkladanie alebo dopytovanie, však v tejto aplikácii chýba. Preto je nutné si takéto funkcie naprogramovať samostatne s využitím jazyka PL/SQL, ktorý poskytuje podporu pre prácu s objektovo relačnými prvkami databázového systému. Na tento účel je možné využiť aplikačné *triggery*, ktoré sú súčasťou Oracle Forms a umožňujú programovať reakcie na udalosti na aplikačnej úrovni a to v jazyku PL/SQL.

Použitie Oracle Forms v súčasnej verzii 10g nám neumožňuje naplno využívať možnosti, ktoré poskytuje objektovo-relačný model, pretože údaje sú v tejto aplikácii v konečnom dôsledku prezentované na úrovni relačného modelu.

5 Špecifikácia a analýza ukážkovej aplikácie

5.1 Neformálna špecifikácia požiadaviek na ukážkovú aplikáciu

Pre ukážkovú aplikáciu, na ktorej sa budem snažiť ukázať využitie objektovo relačných prvkov, predpokladajme nasledujúcu neformálnu špecifikáciu. Aplikácia bude zameraná na správu knižničných jednotiek v malých knižniciach. Ide o knižnice, kde počet knižničných jednotiek už neumožňuje ich správu bez využitia databázovej aplikácie ale na druhej strane nie je potrebná implementácia zložitého informačného systému, ktorý by komplexne pokrýval systém vypožičiavania knižničných jednotiek čitateľmi.

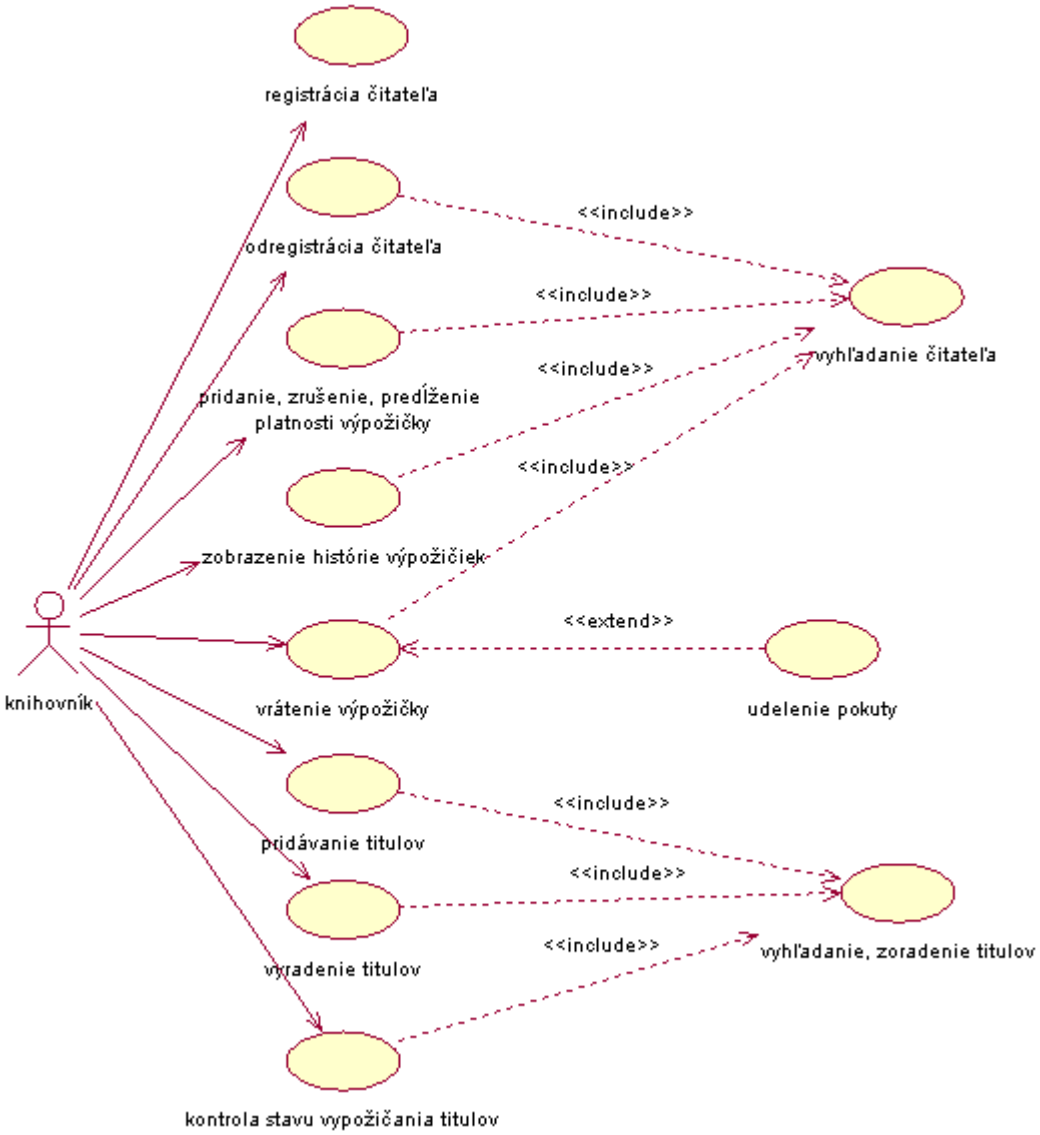
Myšlienka je nasledovná – umožniť jednoduché vkladanie a mazanie databázových údajov v prehľadnej forme, či už ide o pridávanie knižničných titulov alebo čitateľov, ich následné mazanie a pod. Ďalej umožniť užívateľovi aplikácie správu výpožičiek pre čitateľov, ktorí si prichádzajú do knižnice vypožičať knihu prípadne periodikum. S tým súvisí aj poskytovanie informácií o prítomnosti a dostupnosti titulov v knižnici, čo znamená rýchle a prehľadné vyhľadávanie a zoraďovanie v databáze podľa zvolených kritérií. Aplikácia by mala byť schopná spravovať výpožičky jednotlivých čitateľov a v prípade oneskoreného vrátenia vypočítať pokutu, ktorú je nutné čitateľom zaplatiť.

5.2 Požiadavky na funkcionalitu aplikácie

Aplikácia by mala vo všeobecnosti riešiť nasledovné úlohy:

1. **Správa registrovaných čitateľov –** registrácia a odregistrácia čitateľov, vyhľadávanie, pokutovanie za výpožičky
2. **Správa knižničných jednotiek** – pridávanie a vyraďovanie knižničných titulov, vyhľadávanie, zoraďovanie titulov,
3. **Správa výpožičiek –** zobrazovanie aktuálnych výpožičiek čitateľov, pridávanie, vracanie, rušenie výpožičiek pre čitateľov

Funkčné prvky aplikácie je možné rozdeliť na dva vzájomne spolupracujúce celky – ide o správu knižničných titulov, ktorá nevyžaduje interakciu s čitateľom, a správu výpožičiek, ktorá je naopak závislá od komunikácie s čitateľom.



Obrázok 5.1 Diagram prípadov použitia aplikácie

5.3 Analýza a návrh databázy

Keďže databáza, ktorú bude využívať aplikácia bude navrhnutá na objektovo relačnom modeli, je možné jednotlivé reálne entity modelovať ako tzv. objektové typy a nad nimi následne budovať objektové tabuľky. Samozrejmosťou je využitie kolekcií, či už polí prípadne vnorených tabuliek. Vzťahy medzi tabuľkami je možné realizovať pomocou referencií, ktoré sú alternatívou k použitiu cudzích kľúčov u relačného modelu.

5.3.1 Analýza a návrh referencií

Pre potreby našej aplikácie je vhodné využitie obmedzených referencií (*SCOPED REF*), pretože dátové typy REF, ktoré sú atribútmi objektov, odkazujú iba na jednu konkrétnu objektovú tabuľku, čím sa stáva použitie obmedzených referencií pre nás výhodné, keďže umožňujú efektívnejšie dopytovať objekty objektových tabuliek a sú menej pamäťovo náročné.
Implementácia dátového typu REF neumožňuje odkazovať objekt vo vnorenej tabuľke a ani stĺpcový objekt, keďže vnorené tabuľky neuchovávajú OID objektov v nej uložených. Preto je vhodné pre potreby našej aplikácie navrhnúť všetky tabuľky, ktoré chceme odkazovať pomocou referencií ako objektovú tabuľku a nie ako vnorenú tabuľku. pretože by sme stratili možnosť sa na objekty v takejto tabuľke odkazovať pomocou referencií. Implementácia nám ale umožňuje použitie REF vo vnorenej tabuľke pri odkazovaní na inú objektovú tabuľku.
Nad obmedzenými referenciami je možné vytvoriť index. Indexovanie tohto stĺpca umožňuje využiť index na efektívnejšie vyhodnocovanie dopytov, ktoré dereferencujú obmedzenú referenciu.

5.3.2 Analýza a návrh kolekcií

Pri posudzovaní, ktorý typ kolekcie použiť pre naše potreby musíme brať do úvahy vhodnosť ich použitia pre našu aplikáciu. Využitie polí je vhodné v prípade, ak je celá kolekcia spracovávaná ako celok, pretože vtedy polia zabezpečia omnoho lepšie výsledky ako vnorené tabuľky. Ďalej polia sú uložené zbalené a nevyžadujú operácie JOIN na získanie dát ako vnorené tabuľky.
Naopak vnorené tabuľky sú vhodné na použitie, ak potrebujeme z kolekcie získavať iba jednotlivé objekty na základe špecifikovaných kritérií v dopyte nad kolekciou.

5.3.3 Dedičnosť a využitie zameniteľnosti objektov

Implementačnou vlastnosťou objektovo relačného modelu Oracle je typová zameniteľnosť objektov. Ak je tabuľka vytvorená nad určitým objektovým typom, je možné do takejto tabuľky vkladať aj objekty, ktoré sú v hierarchii dedičnosti pod daným objektovým typom. To znamená, že nie je nutné vytvárať pre každý objektový typ osobitnú tabuľku ale stačí vytvoriť tabuľku nad najšpecifickejším objektovým typom a do nej vkladať objekty všetkých synovských typov.

6 Návrh databázy a OR modelovanie v jazyku UML

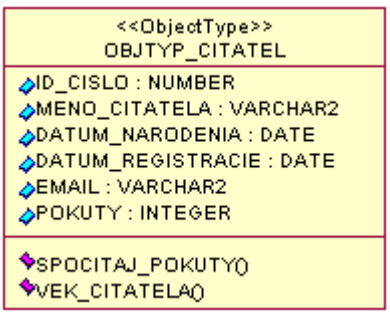
Jedným z najznámejších modelovacích nástrojov v jazyku UML je Rational Rose. Poskytuje podporu pre niekoľko programovacích a dátových prostredí vrátane Oracle.
Nástroj Oracle 8 vo verzii Rational Rose EE v.2002 poskytuje priame aj spätné techniky vývoja objektovo-relačných schém. Ďalšie informácie na [5] a v [6].

6.1 Nástroj Oracle 8

Nástroj Oracle 8 produktu Rational Rose EE v.2002 využíva **stereotypy** (umožňujú vytvoriť nový element odvodený od existujúceho UML meta modelu), **obmedzenia** (vo forme konvencií a tagových hodnôt, špecifikujú podmienky a propozície, ktoré musia byť splnené ako pravdivé inak modelom zobrazený model je neplatný). Tagové hodnoty sú vyjadrené vo forme vlastností generujúcich schému pripojených k projektu, triede, operácii a atribútu – napr. dĺžka poľa.
Vzťahy medzi databázovými elementami sú modelované ako asociácie, agregácie a závislosti. Pomocou asociácií je modelované napr. zanorovanie. Vzťah typu závislosť sa primárne používa na vyjadrenie vzťahov medzi objektovými typmi a tabuľkami, ktoré ukladajú objekty týchto typov. Agregácia sa využíva pri modelovaní referencií.

6.2 Modelovanie objektových typov

Objektové typy sú modelované pomocou stereotypu *<<ObjectType>>*, ktorý je stereotypom triedy. Obsahuje atribúty a metódy.



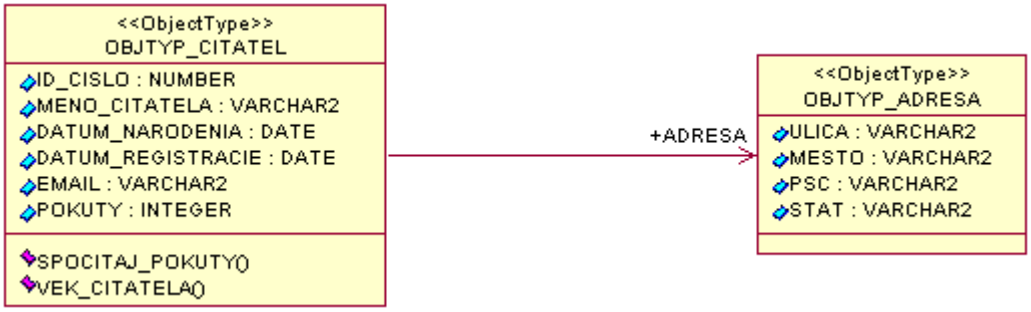
Obrázok 6.1 : Objektový typ ČITATEĽ

6.2.1 Atribúty objektových typov

Atribúty objektových typov sú modelované v Rational Rose ako atribúty tried.

Atribút objektového typu môže byť okrem skalárneho dátového typu aj objektového typu. Vtedy hovoríme o vnorenom objektovom type – je modelovaný pomocou jednosmernej asociácie smerom k vnorenému objektovému typu. Meno role vnoreného typu je meno príslušného atribútu. Takýto typ asociácie vyjadruje vzťah uloženia atribútu ako hodnoty.

Vnorený objektový typ ADRESA objektového typu ČITATEĽ obsahuje vlastné atribúty ako napr. ulica, mesto, PSČ a pod.



Obrázok 6.2: Objektový argument objektového typu ČITATEĽ

Ak však chceme zabezpečiť uloženie atribútu pomocou odkazu, tak takýto typ atribútu modelujeme ako typ asociácie - agregácia.

Obrázok 6.3: Objektový typ REF objektového typu ČITATEĽ

6.2.2 Metódy objektových typov

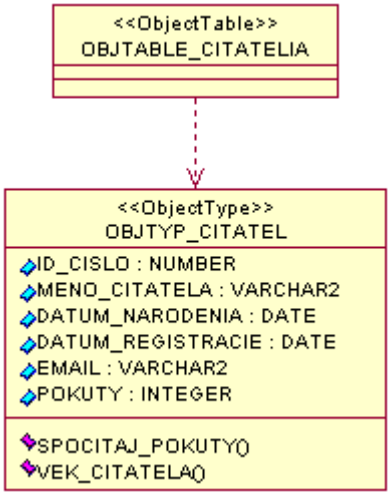
Rational Rose modeluje metódy objektových typov ako operácie triedy. Môžu byť nasledujúcich typov:

- **Spúšťače** čiže *triggery*
- **Procedúry a funkcie** – modelované ako operácie objektového typu
- **Konštruktory**
- **MAP alebo ORDER porovnávacie metódy**

6.3 Modelovanie objektových tabuliek

Modelované ako trieda so stereotypom <<*ObjectTable*>>.

Keďže sú tvorené z objektových typov už vytvorených, vzťah sa modeluje ako závislosť.

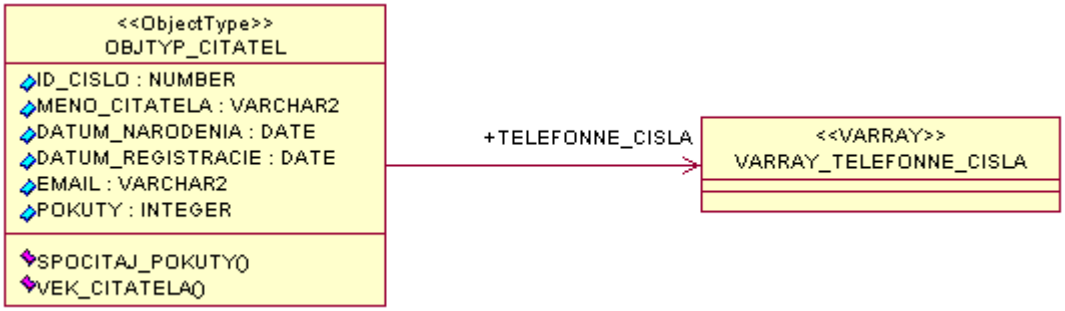


Obrázok 6.4: Objektová tabuľka ČITATELIA

6.4 Modelovanie poľa - VARRAY

Pole je modelované stereotypom <<*VARRAY*>>.

Všetky elementy poľa musia byť rovnakého dátového typu –skalárneho alebo objektového. Ak je pole založené na objektovom type, vzťah je modelovaný ako závislosť VARRAY typu na danom objektovom type. Pole môže byť použité ako atribút objektového typu – modelované asociáciou.

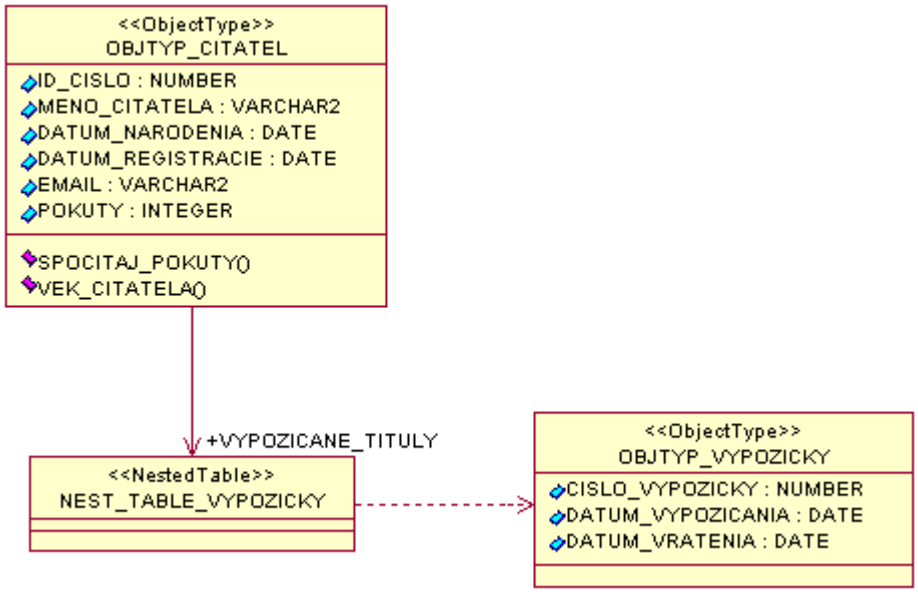


Obrázok 6.5: Pole s premenlivou dĺžkou TELEFÓNNE ČÍSLA

6.5 Modelovanie vnorenej tabuľky - nested table

Vnorené tabuľky sú modelované stereotypom triedy<<*NestedTable*>>. Modelovanie ostatných parametrov funguje podobne ako pri VARRAY.

Ak je dátový typ objektov vnorenej tabuľky objektového typu, je táto situácia modelovaná ako závislosť vnorenej tabuľky na danom objektovom type.



Obrázok 6.6: Závislosť objektového typu VÝPOŽIČKY na vnorenej tabuľke VÝPOŽIČKY

6.6 Kompletná databázová schéma

Databázová schéma a jej modelovanie by malo zahŕňať modelovanie reálnych entít ako objektových typov s vlastnosťami ako atribútmi a operáciami ako metódami, ktoré sú pre náš databázový systém potrebné a podstatné.

Celková databázová schéma aplikácie bude reprezentovaná tromi základnými objektovými typmi – ČITATEĽ, TITUL a KÓPIA TITULU. Tieto tri objektové typy reprezentujú tri základné reálne entity, ktoré pre náš databázový systém budeme modelovať. Ďalšie objektové typy sú odvodené od týchto objektových typov alebo sú priamo ich súčasťou. Objektový typ ČITATEĽ obsahuje vnorený objektový typ ADRESA, ktorý obsahuje vlastné atribúty, ktoré určujú adresu daného čitateľa. Objektový typ TITUL obsahuje takisto vnorený objektový typ – ELEKTRONICKÁ PRÍLOHA, pričom tento objektový typ obsahuje informácie o priloženej elektronickej prílohe k danému titulu. Objektový typ VÝPOŽIČKA je objektovým typom pre vnorené tabuľky VYPOŽIČANÉ TITULY a HISTÓRIA VÝPOŽIČIEK objektového typu ČITATEĽ. Z objektového typu TITUL sú odvodené dva objektové typy – KNIHA a PERIODIKUM, ktoré vznikli dedením z nadtypu TITUL a sú teda podtypmi objektového typu TITUL. Znamená to, že si zachovajú všetky vlastnosti nadtypu, plus navyše obsahujú vlastné špecifické vlastnosti – či už atribúty alebo metódy.

Každý z troch základných objektových typov obsahuje okrem atribútov objektového typu, čo sú vlastne užívateľsky definované dátové typy aj typy skalárne, ktoré sú súčasťou databázového systému Oracle. Spolu všetky atribúty vytvárajú vlastnosti jednotlivých objektových typov, ktoré sú pre našu databázovú schému potrebné.

Každý objektový typ okrem atribútov môže obsahovať aj metódy, čiže operácie, ktoré sa vykonávajú nad dátami, ktoré sú v objektoch daných objektových typov uložené. Objektový typ ČITATEĽ obsahuje metódu SPOČÍTAJ POKUTY, ktorá pracuje nad vnorenou tabuľkou tohto objektového typu a vypočíta výšku pokuty pre výpožičky aktuálne uložené vo vnorenej tabuľke VYPOŽIČANÉ TITULY. Ďalšou metódou je VEK_ČITATEĽA, ktorá z dátumu narodenia čitateľa vypočíta aktuálny vek čitateľa.

Niektoré argumenty objektových typov sú modelované ako kolekcia. Kolekcie sa dajú z výhodou použiť pri modelovaní rôznych vlastností objektových typov. Objektový typ ČITATEĽ obsahuje argument TELEFÓNNE ČÍSLA, ktorý obsahuje tri telefónne kontakty na daného čitateľa. Keďže poradie čísel je v tomto prípade dôležité a s kolekciou budeme pracovať ako celkom, bol zvolený pre tento argument typ kolekcie pole. Ďalšie argumenty objektového typu ČITATEĽ, ktoré sú typu kolekcia, sú VYPOŽIČANÉ TITULY a HISTÓRIA VÝPOŽIČIEK. V tomto prípade je jasné, že s prvkami týchto kolekcii sa bude často pracovať jednotlivo a osobitne, preto môžeme s výhodou využiť typ kolekcie vnorená tabuľka.

Kompletný databázový model je možné nájsť v prílohe 1.

6.7 Nepodporované OR vlastnosti

Podpora pre modelovanie objektovo relačných databázových schém nie je v Rational Rose úplná. Je to z dôvodu, že nástroj Oracle 8 bol vytvorený pre databázu Oracle 8i, a tá v tom období ešte nepodporovala všetky OR vlastnosti ako je to v súčasnom produkte Oracle 10g. Keďže sa však nástroj Oracle 8 už ďalej nevyvíjal, nie je tam podpora nových vlastností zahrnutá. Preto môže nastať situácia, že nie je možné vytvoriť schému, ktorá by plne zodpovedala požiadavkám, ktoré na databázový systém kladieme. Ako najdôležitejšia sa zdá byť absencia modelovania dedičnosti objektových typov. Dedičnosť je totiž významná črta objektového programovania, preto je nutné absenciu modelovania tejto vlastnosti riešiť iným spôsobom. Bolo nutné vytvoriť ďalší model – konceptuálny, ktorý by umožňoval znázorniť dedičnosť ako vlastnosť generalizácia-špecializácia pre entity databázovej schémy, ktoré v tomto modeli vystupujú ako triedy.

Ďalej absentuje nastavenie vlastností objektových typov ako sú INSTANTIABLE a FINAL prípadne tvorba indexov a pod.

Konceptuálny model je možné nájsť v prílohe 1.

6.8 Generovanie Data Definiton Language (DDL) skriptu

Nástroj Oracle 8 produktu Rational Rose obsahuje užitočnú funkciu, ktorá umožňuje generovanie DDL skriptu v jazyku SQL pre nami namodelovaný databázový systém. Vygenerovaný skript obsahuje zdrojový kód v jazyku SQL, ktorý vytvára všetky modelované entity presne podľa špecifikácie v modeli nášho databázového systému. Takisto je možné následne sa pripojiť na databázový server a daný skript okamžite vykonať, čo má za následok vytvorenie databázových objektov v našej databáze.

Keďže však nástroj Oracle 8 neumožňuje modelovať všetky objektovo relačné rysy, je nutné vygenerovaný skript doplniť napr. o vlastnosti objektových typov [NOT] FINAL a INSTANTIABLE a samozrejme o dedičnosť. To zahŕňa doplnenie definície objektových typov, ktoré sú vytvorené pod už modelovaným objektovým typom, s využitím kľúčového slova UNDER, za ktorým špecifikujeme, od ktorého objektového typu bude nový typ dediť vlastnosti. Ďalej je nutné pridať aj definíciu argumentov a metód jednotlivých objektových typov, indexy vytvorené nad referenciami, prípadne doplnenie špecifikácie vnorených tabuliek ako IOT.

Spôsob zápisu jednotlivých vlastností je popisovaný v nasledujúcej kapitole.

7 Implementácia aplikácie v PL/SQL

7.1 Práca s objektovými typmi Oracle

7.1.1 Príkazy pre prácu s objektovými typmi

CREATE TYPE – umožňuje vytvoriť objektový typ v danej tabuľkovej schéme

ALTER TYPE – umožňuje zmeny v objektovom type

DROP TYPE – umožňuje zrušenie objektového typu

EXECUTE TYPE – umožňuje používať a odkazovať sa na pomenované typy

UNDER TYPE – umožňuje vytvoriť podtypy pod ľubovoľným nefinálnym objektovým typom

UNDER VIEW – umožňuje vytvoriť podpohľady pod ľubovoľným objektovým typom

EXECUTE – umožňuje vyvolať metódy objektového typu, vrátane metódy konštruktor. Ďalej umožňuje nad objektovým typom vykonávať nasledujúce operácie s objektovým typom:

a) Definovať tabuľku
b) Definovať stĺpec v relačnej tabuľke
c) Deklarovať premennú alebo parameter pomenovaného typu

Oracle nepodporuje špecifikáciu integritných obmedzení pri deklarovaní objektového typu, avšak môžeme ich špecifikovať pri vytvorení tabuľky.

Príklad vytvorenia objektového dátového typu s použitím jazyka PL/SQL:

```
CREATE OR REPLACE TYPE OBJTYP_CITATEL AS OBJECT (
     ID_CISLO NUMBER,
     MENO_CITATELA VARCHAR2(50),
     DATUM_REGISTRACIE DATE,
     VEK NUMBER,
     ADRESA OBJTYP_ADRESA,
     TELEFONNE_CISLA VARRAY_TELEFONNE_CISLA,
     EMAIL VARCHAR2(40),
     VYPOZICANE_TITULY NT_OT_VYPOZICKY,
     HISTORIA_VYPOZICIEK NT_OT_HISTORIA,
     MEMBER FUNCTION SPOCITAJ_POKUTY RETURN INTEGER
     )
     INSTANTIABLE FINAL;
/
CREATE OR REPLACE TYPE BODY OBJTYP_CITATEL AS
MEMBER FUNCTION SPOCITAJ_POKUTY RETURN INTEGER IS
prem_POKUTY INTEGER := 0;
```

```
BEGIN
    FOR i in 1..SELF.VYPOZICANE_TITULY.COUNT LOOP
      IF (TRUNC(SYSDATE) > VYPOZICANE_TITULY(i).DATUM_VRATENIA) THEN
         prem_POKUTY := prem_POKUTY + ((TRUNC(SYSDATE) -
         VYPOZICANE_TITULY(i).DATUM_VRATENIA));
      END IF;
    END LOOP;
RETURN prem_POKUTY;
END;
END;
/
```

7.1.2 Modifikácia objektových typov

Použitím príkazu ALTER TYPE je možné modifikovať existujúci objektový typ a robiť nasledujúce zmeny:

a) pridať a mazať atribúty

b) pridať a mazať metódy

c) modifikovať číselný atribút na zväčšenie jeho veľkosti, presnosti alebo rozmeru

d) modifikovať premennú na zväčšenie jej veľkosti

e) zmeniť FINAL a INSTANTIABLE vlastnosti

7.2 Práca s objektovými tabuľkami

Tabuľku objektového typu deklarujeme nasledovne:

```
CREATE TABLE OBJTABLE_TITULY OF OBJTYP_TITUL;
```

Ak tabuľka obsahuje stĺpce vnorenej tabuľky, je nutné pri deklarácii objektovej tabuľky, ktorá túto vnorenú tabuľku obsahuje, špecifikovať jej úložnú tabuľku a spôsob organizácie dát v tejto tabuľke. Meno úložnej tabuľky sa používa napr. pre definíciu indexu vnorenej tabuľky.

Nasledujúci zápis ukazuje vytvorenie objektovej tabuľky s dvoma vnorenými tabuľkami, ktorých úložná tabuľka je organizovaná ako *IOT* s NESTED_TABLE_ID ako prefixom primárneho kľúča a využitím kompresie pre zníženie pamäťových nárokov:

```
CREATE TABLE OBJTABLE_CITATELIA OF OBJTYP_CITATEL
NESTED TABLE VYPOZICANE_TITULY STORE AS NEST_TABLE_VYPOZICKY ((
PRIMARY KEY(NESTED_TABLE_ID,CISLO_VYPOZICANIA)) ORGANIZATION INDEX
COMPRESS),
NESTED TABLE HISTORIA_VYPOZICIEK STORE AS NEST_TABLE_HISTORIA ((
PRIMARY KEY(NESTED_TABLE_ID,CISLO_VYPOZICANIA)) ORGANIZATION INDEX
COMPRESS);
```

Ak obsahuje objektová tabuľka dátový typ referencia, je možné ju nastaviť dodatočným príkazom ako obmedzenú, ak to návrh aplikácie umožňuje:

```
ALTER TABLE OBJTABLE_KOPIE ADD (SCOPE FOR(REF_TITUL) IS OBJTABLE_TITULY);
```

Následne je výhodne nad takouto referenciou vytvoriť index, a to napr. pomocou príkazu:

```
CREATE INDEX REF_TITUL_INDEX ON  OBJTABLE_KOPIE (REF_TITUL);
```

Po vytvorení premennej objektového typu je možné z nej vytvoriť inštanciu, ktorá je objektom. Tento objekt má atribúty a metódy pre svoj typ. Pretože objektová inštancia je konkrétny objekt, je možné k atribútom priradiť konkrétne hodnoty a volať jeho metódy. Takéto objekty sú súčasťou objektových tabuliek. Pre vytvorenie inštancie objektového typu sa využíva metóda zvaná konštruktor.
Nasledujúci zápis ukazuje využitie implicitného konštruktora:

```
INSERT INTO OBJTABLE_TITULY VALUES( OBJTYP_KNIHA ('Pán prsteňov 2','J.R.R.
Tolkien','K','Ikar','SK','Tolkienovská sága',OBJTYP_ELEKTRONICKA_PRILOHA
('',''),'978-80-353-424-66','scifi',''));
```

7.2.1 Príkazy pre prístup k objektom

Kým objektové typy využívajú iba príkaz EXECUTE, objektové tabuľky používajú všetky príkazy aké sa používajú aj nad relačnými tabuľkami:
SELECT – umožňuje prístup k objektom a jeho atribútom uloženým v tabuľke
UPDATE – umožňuje modifikáciu atribútov objektov v tabuľke
INSERT – umožňuje pridávať nové objekty do tabuľky
DELETE – umožňuje vymazávať objekty z tabuľky

7.2.2 Funkcie a operátory užitočné pre prácu s objektmi

IS OF - testuje na akej úrovni špecializácie v hierarchii dedičnosti sa daný typ nachádza.
VALUE - ako argument má funkcia alias pre objektovú tabuľku alebo pohľad a vracia inštanciu objektu korešpondujúcu k riadkom tabuľky alebo pohľadu. Môže vrátiť inštanciu deklarovaného typu alebo ľubovoľného podtypu.
TREAT - funkcia sa pokúša vnútiť inštancii rodičovského typu typ synovský. Ak nie je možné danú konverziu uskutočniť, funkcia vracia NULL.

Význam funkcie:

- nastaviť hodnotu podtypu v nadradenom objektovom type
- umožniť prístup nadradeného typu k atribútom a metódam jeho podtypov
- zameniteľná objektová tabuľka alebo stĺpec má skrytý stĺpec, ktorý obsahuje typy atribútov každého podtypu k danému typu –touto funkciou je možné k danému stĺpcu pristupovať

Príklad ukazuje použitie funkcie TREAT pri pretypovaní objektu na špecifickejší objektový typ, čo umožňuje dopytovať argumenty z daného objektu. Objektové inštancie získame funkciou VALUE s parametrom, ktorý obsahuje *alias* na objekty objektového typu špecifikovaného pomocou funkcie IS OF:

```
SELECT TREAT (VALUE(c) AS OBJTYP_KNIHA).SPOLUAUTORI_KNIHY FROM
OBJTABLE_TITULY c WHERE VALUE(c) IS OF (OBJTYP_KNIHA);
```

DEREF - vracia inštanciu objektu korešpondujúcu k REF, táto inštancia môže byť deklarovaného typu REF alebo jeho ľubovoľného podtypu.

Nasledujúci zápis vráti inštanciu objektu z vnorenej tabuľky do premennej objektového typu, ktorý je zhodný s typom, na ktorý odkazuje referencia:

```
SELECT DEREF(REF_KOPIA_TITULU) INTO prem_id_kopie FROM TABLE (
SELECT VYPOZICANE_TITULY FROM OBJTABLE_CITATELIA WHERE ID_CISLO = 1001) p
WHERE p.CISLO_VYPOZICKY = 5001;
```

REF - ako argument obsahuje alias pre objektovú tabuľku, pohľad alebo ich podtyp a vracia odkaz na inštancie objektov z danej tabuľky, pohľadu. Môže vracať odkazy na objekty deklarovaného typu alebo jeho podtypu.

Príklad ukazuje využitie funkcie REF pri vkladaní odkazu na objekt do objektovej tabuľky:

```
INSERT INTO OBJTABLE_KOPIE SELECT KOPIE_SEQ.nextval,'ÁNO',REF(t) FROM
OBJTABLE_TITULY t WHERE t.NAZOV_TITULU = 'Pán prsteňov 2';
```

TABLE - táto funkcia vytvára kolekciu riadkov – vnorenú tabuľku alebo pole, ktorá môže byť použitá ako fyzická dátová tabuľka alebo premenná v jazyku PL/SQL typu kolekcia.

Nasledujúci zápis ukazuje využitie funkcie TABLE pre získanie vnorenej tabuľky:

```
SELECT p.CISLO_VYPOZICKY,p.DATUM_VYPOZICANIA,p.DATUM_VRATENIA FROM TABLE (
SELECT VYPOZICANE_TITULY FROM OBJTABLE_CITATELIA  WHERE ID_CISLO = 1001 )
p );
```

7.2.3 Operácie nad kolekciami

Existujú dva základné spôsoby ako dopytovať tabuľku, ktorá obsahuje stĺpec alebo atribút typu kolekcia:

- vrátiť kolekciu vnorenú vo výslednom riadku, ktorý ju obsahuje
- rozložiť kolekciu tak, že každý jej prvok vystupuje sám ako riadok

Rozložiť kolekciu je možné s použitím kľúčového slova TABLE nad riadkom tabuľky, ktorý obsahuje vnorenú tabuľku alebo pole:

```
SELECT p.DATUM_VYPOZICANIA, p.DATUM_VRATENIA FROM TABLE (
SELECT VYPOZICANE_TITULY FROM OBJTABLE_CITATELIA  WHERE ID_CISLO = 1001)p;
```

Niektoré obmedzenia pre použitie dopytu v argumente:

- TABLE musí vrátiť typ kolekcia
- musí vyberať len jeden prvok
- môže vrátiť iba kolekciu uloženú v rámci jedného riadku

Nezanorený dopyt na kolekciu umožňuje vidieť dáta v relačnej – priamej forme. Dopytovať je možné jednoduché aj viacúrovňové kolekcie oboch typov.
Ak príkaz SELECT odkazuje iba na stĺpce vnorenej tabuľky, dopyt je optimalizovaný pre vykonanie nad úložnou tabuľkou vnorenej tabuľky. Nezanorený dopyt je zhodný pre oba typy kolekcií.

Záver

Cieľom mojej práce bolo ukázať objektovo relačný model pri vývoji aplikácií ako alternatívu k tradičnému, stále prevládajúcemu relačnému modelu. Snažil som sa o to vo všetkých etapách vývoja aplikácie, či už pri analýze, návrhu, či samotnej implementácií ukážkovej aplikácie. Objektovo relačný model sa ukázal ako plnohodnotné riešenie pri návrhu databáz, ktoré môže ponúknuť oproti relačnému modelu aj niečo navyše - či už ide o využitie kolekcií, referencií ale v prvom rade o jednoduchšie modelovanie reálnych entít ako databázových objektov.

Pri analýze požiadaviek na ukážkovú aplikáciu som sa snažil využiť čo najviac vlastností, ktoré ponúka objektovo relačný model databázového systému Oracle. Preto bola potrebná dobrá teoretická znalosť implementácie objektovo relačného modelu Oracle, ktorý vychádza zo štandardov SQL 1999 a 2003. Návrh aplikácie som uskutočnil pomocou nástroja Oracle 8 aplikácie Rational Rose EE. Ide o nástroj, ktorý umožňuje modelovanie objektovo relačnej databázovej schémy konkrétne pre riešenie od Oracle. Tento nástroj sa ukázal ako vhodný pre modelovanie základných databázových entít ale zďaleka sa nedá hovoriť ako o kompletnom nástroji pre modelovanie objektovo relačných databáz, keďže absentuje modelovanie dôležitých vlastností ako napr. dedičnosť. Pre implementáciu aplikácie bolo vybrané vývojové prostredie Oracle Forms 10g, ktoré je tradičným nástrojom firmy Oracle pri tvorbe relačných databáz. Cieľom práce bolo zistiť, či existuje v tejto aplikácii aj objektovo relačná podpora, a pokúsiť sa ju využiť pri implementácii. Zistil som však, že takáto podpora v tejto aplikácii je veľmi slabá až na pár výnimiek - vytváranie objektových typov a tabuliek, dopytovanie objektových tabuliek pomocou sprievodcov. Podpora pre ďalšie vlastnosti však chýba, preto bolo nutné tento problém riešiť pomocou jazyka PL/SQL, ktorý som využíval pri implementácii. Jazyk PL/SQL ako úplný programovací jazyk vytvorený firmou Oracle, poskytol výborné možnosti pre prácu s objektovo relačnou databázou. Z dôvodu absencie podpory u Oracle Forms, bolo nutné využiť princíp, ktorý umožňuje prezentovať objektovo relačné dáta uložené v objektových tabuľkách na relačnej úrovni. Z tohto dôvodu by som zatiaľ neodporúčal využitie Oracle Forms pre implementáciu objektovo relačných databáz, pretože absentujúca podpora pre tento model nám neumožňuje efektívne pracovať s týmto modelom. V práci som však dokázal, že je možné takúto aplikáciu v Oracle Forms vytvoriť ale je nutné použiť implementačné triky.

Prínosom práce pre mňa bola možnosť zoznámiť sa s iným pohľadom na riešenie problematiky databáz. Tieto poznatky som potom využil pri implementácii ukážkovej aplikácie. Keďže nástroje, ktoré som musel pri vývoji aplikácie používať sa ukázali zastarané, prípadne nepodporovali potrebné vlastnosti, ktoré som očakával, odporúčal by som do budúcnosti pracovať pri návrhu databázy s novšími nástrojmi a pre implementáciu aplikácie zatiaľ použiť iné vývojové prostredie.

Literatúra

[1] Melton, J., Simon, A.: SQL: 1999, formerly known as SQL3. SIGMOD Record, 1, 1999, s.131-138.

[2] Eisenberg A., Melton, J., Kulkarni, K., Michels, J.E., Zemke, F.: SQL:2003 has been published. ACM SIGMOD Record, 1, 2004, s. 119-126.

[3] Oracle Database Application Developer's Guide - Object-Relational Features 10g Release 2. (2005). Dokument dostupný na URL http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14260.pdf (máj 2008).

[4] PL/SQL User's Guide and Reference 10g Release 1. (2003). Dokument dostupný na URL http://download.oracle.com/docs/cd/B14117_01/appdev.101/b10807.pdf (máj 2008).

[5] Rational Rose 2000e Using Rose Oracle8. Santa Clara CA, Rational Software Corporation, (2000). Dokument dostupný na URL ftp://ftp.software.ibm.com/software/rational/docs/documentation/manuals/docset149/Rational_Rose/Documentation/Rose_oracle8.pdf (máj 2008).

[6] Zendulka, J.: Object-Relational Modeling. In Ferraggine, V.E., Doorn, J.H., Rivero, L.C. (Editors): Encyclopedia of Database Technologies and Applications. 2nd Edition. IGI Global.(v tisku).

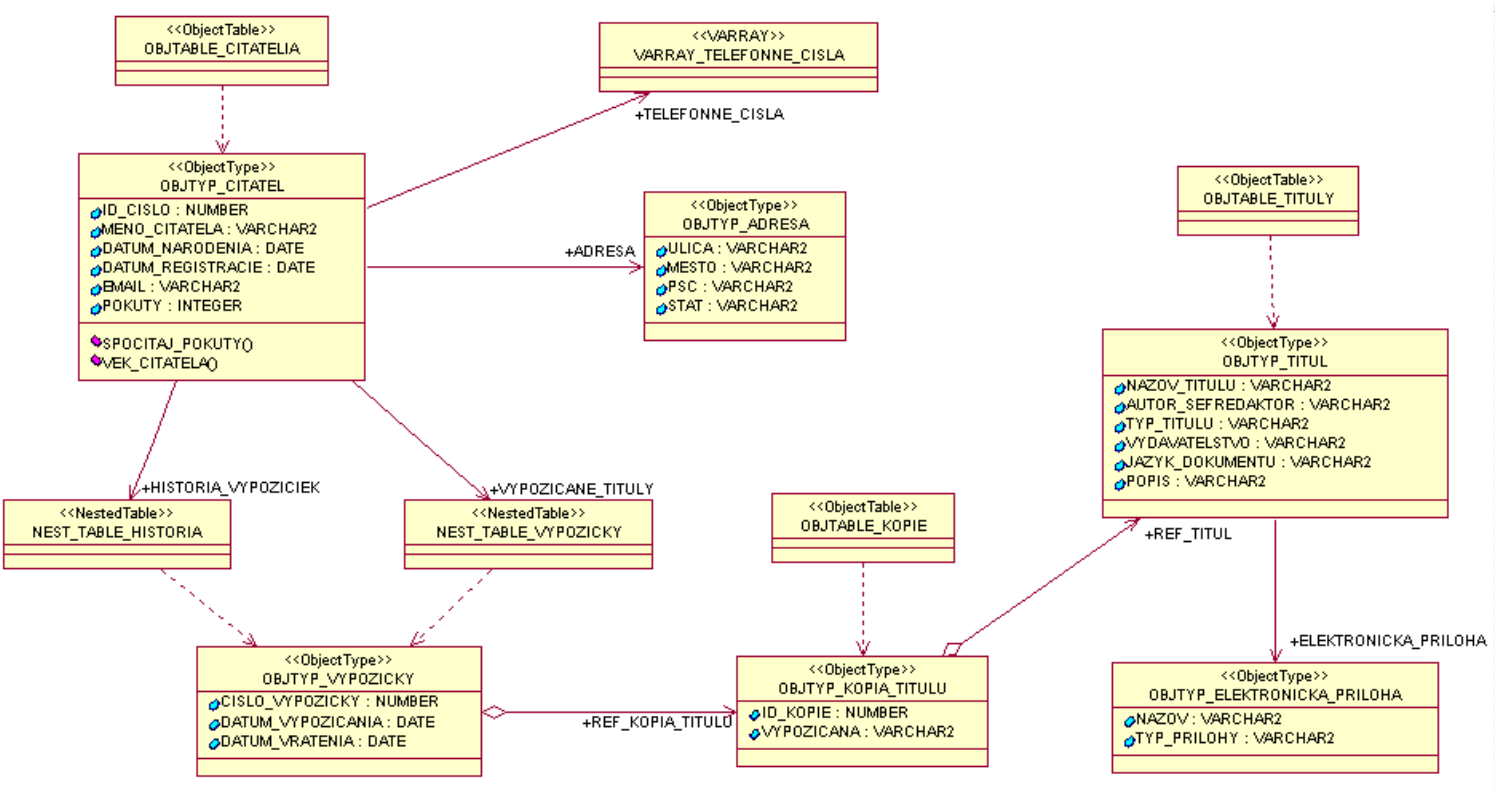
Zoznam príloh

Príloha 1. Modely databázy

Príloha 2. CD so zdrojovými súbormi a aplikáciou

Príloha č.1 - Modely databázy

Databázový model



Konceptuálny model