1 Objektové konceptuální modelování

Konceptuální schéma popisuje metadata systému, je v drtivé většině objektové, může mít grafickou syntax (ER, UML) nebo textovou syntax (CDL – Concept Definition Language). Schéma se používá jak pro vývojáře tak pro uživatele.

Relační konceptuální schéma je velmi odlišné od objektového, v tomto ohledu jsou výhodné objektové databázové systémy, není potřeba složité transformace. Ale relační systémy jsou mnohem používanější v praxi (často s doplněním objektových vlastností).

Struktura je uspořádaná n-tice s pevným počtem pojmenovaných hodnot obecně různých typů. Pojmenované hodnoty označujeme jako *vlastnosti*. Synonyma jsou záznam nebo strukturovaný datový typ.

Kolekce je uspořádaná multimnožina (může jeden prvek obsahovat vícekrát), která je tvořena předem neomezeným počtem prvků stejného typu. Synonyma jsou řetězec, soubor, seznam, posloupnost.

Operace nad kolekcí jsou vkládání, odebrání, získání prvku a počtu prvků. Je možné je provést nad všemi prvky operací `forall`. Kolekce poskytují tzv. kurzor, tedy ukazatel/iterátor. Nad kolekcemi může existovat několik uspořádání, je možné požadovat unikátnost vlastnosti.

Vlastnosti kolekce náleží celé kolekci a nikoli jejím prvkům, dají se definovat uživatelské vlastnosti, většinou agregační údaje (suma, průměr apod.).

Objekt je struktura s jednoznačnou identifikací OID (první a systémová vlastnost struktury). Objekt je tak odkazovatelný a může figurovat ve vztazích.

Prostá struktura je struktura bez identifikace. V CDL se nastaví hodnotou `Data=Value`. `Data=Ref` znamená objekt.

Rejstřík obsahuje uživatelem definované identifikátory objektů vázané na jejich OID. Existenci objektů lze vynutit ihned po spuštění.

Výčtový typ je definován pomocí `Data=Enum`. Má pevný počet identifikátorů hodnot, jeho aktuální hodnota není strukturovaná.

Vazby mohou být definovány jak mezi výskyty struktur – *vztahy*, tak mezi definicemi struktur – *dědičnost*.

Vlastnosti typu struktura jsou pak *členové* nadřazené struktury – *vlastníka*. Jedná se právě o vztah.

Typy vztahů:

1. Členem je prostá struktura – jedná se o hierarchické členění struktur
2. Členem je kolekce prostých struktur – opět pouze hierarchické zanoření
3. Členem je jediný objekt – vztah 1:1 realizovaný pomocí OID
4. Členem je kolekce objektů – vztah 1:N, kolekce obsahuje reference na objekty

Inverzní vztahy se mohou automaticky dotvořit při vzniku vztahu (atribut `inverse`), velmi usnadňují vývoj IS (automatické udržování integrity apod.).

Dědičnost umožňuje definici typu pomocí typu jiného.

- Pojmy *předek* a *následník*.

- Vlastnosti se mohou přidávat, modifikovat i ubírat.
- Existuje vícenásobná dědičnost.
- Relace dědičnosti je tranzitivní uzávěr relace přímé dedičnosti.
- *Generalizace* je postup vytváření hierarchie dědičnosti hledáním společných vlastností a vytvářením předků.
- *Specializace* je opačný postup ke generalizaci, kdy se rozrůzňují typy a vznikají tak následníci.
- *Abstraktní typy* existují pouze kvůli hierarchii dědičnosti, ale nelze vytvářet jejich instance.

Databázový model rozšiřuje konceptuální o implementačně závislé rysy. Řeší integritní omezení, zvláštní a povinné hodnoty, implementace extentů apod.

Identické struktury (objekty) jsou struktury (objekty) se stejným obsahem (počet vlastností a jejich hodnoty).

Totožné objekty jsou objekty se stejným OID.

Extent je kolekce objektů, kam je objekt zařazen při vzniku a odstraněn při zániku. Problémem je udržování extentů také pro předky, jelikož následník je také typem předka a při vyšší hierarchii je udržování velmi složité. Povolení udržování extentu je značeno `Extent` u jména struktury.

2 Transakce a jejich modely

Integrita dat znamená, že některá data nemohou nabývat libovolných hodnot. Zajištění integrity dat zapisujeme *integritními omezeními*.

Integritní omezení lze formalizovat, specifikovat zdrojový a cílový objekt, predikátovou závislost a podmínky, kdy se má pravidlo uplatnit a také algoritmus, který integritu zajistí.

Transakce je skupina operací provedených atomicky jako celek nebo vůbec. Tímto zajišťují konzistenci a integritu dat i celého systému.

Požadavky na transakce:

- Vypořádání se s výskytem poruch.
- Možnost masivního paralelismu (řádově stovky transakcí najednou).
- ACID
  1. *Atomicity* – buď zcela nebo vůbec
  2. *Consistency* – databáze zůstane konzistentní po provední nebo zrušení transakce
  3. *Isolation* – souběžné provádění má stejný efekt jako sekvenční
  4. *Durability* – po dokončení jsou změny trvalé

Databáze je konzistentní, pokud splňuje integritní omezení a transakce jsou konzistentní, pokud po jejich skončení zůstane databáze v konzistentním stavu.

Atomická transakce skončí buďto úspěšně (commit) nebo neúspěchem (abort, rollback), úspěšná je pouze tehdy, pokud systém commit akceptuje, ostatní ukončení se provedou vždy. Commit nemusí být proveden z důvodu porušení různých podmínek.

Izolovanost transakcí vyžaduje sestavení vykonávacího plánu (sekvenční plán provede jednu po druhé, ale to je nevýkonné). Nekteré části lze provést paralelně, jiné sekvenčně (např. kvůli sdíleným prostředům).

Modely transakcí strukturují transakce na jejich části:

- *Distrubuované transakce* – jejich části jsou prováděny na více místech (v různých systémech)
- *Podtransakce* – části transakce vzniklé dekompozicí

Model plochých transakcí modeluje transakce jako posloupnosti proměnných a SQL příkazů, která je dokončena úspěšně nebo neúspěšně (abort nebo chyba). Není schopna částečného návratu, pouze celek nebo nic (neefektivní).

Model strukturovaných transakcí zavádí *milníky* (body návratu, savepoints). Je možné použít podmínky a příkazy pro částečný návrat a vytvoření milníku. Částečný návrat mění pouze stav databáze, nikoli lokální proměnné.

Model distribuovaných transakcí přináší volání podtransakcí (které se mohou vykonávat i na jiném místě v jiném systému). Hlavní (volající) transakce využívá příkaz `execute`. Jednotlivé podtransakce splňují ACID, globální ACID je složitější, pokud se data čerpají z více systémů.

Model zanořených transakcí jsou vytvořeny pro chod na jednom serveru, oproti distribuovaným jde o model skládání shora dolu, tedy dekompozice (distribuované vlastně skládaly jednotlivé transakce). Jedním z mnoha modelů je *Mossův model*.

Model zřetězených transakcí dekomponuje transakce hlavně kvůli výkonu (u dlouhých transakcí se ztratí návratem více dat). Umožňují po potvrzení jedné transakce ihnet pustit další (omezení režie manuálního volání podtransakce). Zřetězené transakce jako celek *nepodporují atomicitu ani izolovanost*. Jde obejít alternativním použitím příkazu `chain`, ale na úkor výkonnosti.

Kompenzující transakce jsou speciálním případem transakce, který zajišťuje obnovu důsledků spuštění jiné transakce.

Model zotavitelných front implementuje řetězec transakcí, ale mohou mezi nimi být časové prodlevy, jediný požadavek je, aby se po dané transakci někdy provedla další. Vhodné pro plánování transakcí.

Řízení toku (workflow) jsou dalším stupněm transakčního zpracování. ACID vlastnosti jsou volnější, podtransakcemi chápeme nejen transakce, ale i akce uživatele a podobně. Snaží se definovat celkové změny, které se v systému provedou (např. fakturace zboží). Dá se chápat jako systém s agenty. Častou reprezentací je And-or graf.

3 Internetová rozhraní informačních systémů

Internetové portály byly původně "rozcestníky", tedy stránky, které odkazovaly na různé oblasti, o kterou se návštěvníci zajímali, postupně vznikaly více profilované oborové portály.

Portálové řešení informačního systému znamená IS typu klient-server, u něhož klient potřebuje pouze nemodifikovaný webový prohlížeč s podporou JavaScriptu a DOM, a kde server generuje stránky jazykem pro generování HTML/XML. Jedná se tedy o model s tenkým klientem.

Multilingválnost je typickým požadavkem pro portálové řešení. Požaduje se, aby systém dokázal komunikovat libovolným jazykem, všechny řetězce tedy nesmějí být pevné, ale musejí být nahraditelné, většinou pomocí resurce-manageru (nutnost skladu textů).

Snadná modifikovatelnost je dalším pilířem správného IS, jelikož je potřeba jej udržovat dlouhou dobu. Důležitými aspekty jsou dokumentace, existence kódu pouze jednou (a ne vícekrát na více místech), jednoduchost modelu a konceptuální modelování.

Uživatelská přívětivost je pak požadavkem pro zisk pomocí vyrobeného systému (cena, rychlost, spolehlivost apod.).

4 Serverová část portálového informačního systému

Databázový systém modeluje pouze data (nikoli procesy), je v drtivé většině relační (MySQL, ORACLE apod.).

Hlavním úkolem serveru je generovat stránky, které se pošlou uživateli. Používá se HTML nebo XML.

Klient stránky buďto pouze prezentuje (ovlivňuje styly) – HTML, nebo přímo dostane jen data a naloží s nimi podle předpisu – XML a XSLT.

Programovací jazyk je hlavně generátor textu, generuje HTML/XML stránky. Musí umět pracovat s databází, aby mohl pracovat jako prostředník mezi klientem (HTTP komunikace) a databázovým serverem (SQL komunikace).

Typické jazyky jsou PHP, ASP, JSP, CSP apod.

Protokol HTTP je *bezestavový*. Ale IS potřebuje znát stav (kontext), jelikož stránky se podle něj generují.

Jednoduché a "nečisté" držení kontextu – posílá se s každým dotazem i odpovědí, neefektivní:

- Kontext součástí URL, většinou metoda GET s parametry – omezení délky a přímá viditelnost.
- Hidden form fields – omezení velikosti a formátu textu, nepřímá viditelnost.

Cookies jsou data odeslaná serverem, uložená u klienta. Nejsou přímo omezena velikostí nebo formátem.

- Bezpečnostní problém v tom, že server může u klienta zapisovat.
- Omezení je dáno politikou (např. maximálně 20 po 4kB na jednu doménu).
- HTTP položky `Set-Cookie:` a `Cookie:`. Útočníkovi nic neřeknou, je to jen identifikační klíč dat.
- Z jedné domény se vše ukládá do jediného souboru.

Session je propracovanější metoda udržování kontextu. Každému uživateli je vygenerována identifikace SESSID, poté lze vždy uživatele poznat.

- Data jsou na serveru, klient obdrží jen klíč.
- K uložení klíče u klienta lze využít cookies, ale mohou být zakázány.
- SESSID lze předávat s každým požadavkem a odpovědí.
- Dá se zrušit a tím vymazat (odhlášení apod.), má danou dobu života.

Formuláře jsou hlavním prostředkem zadání dat uživatelem serveru. Využívají dvě metody odeslání dat:

1. `GET` – transparentní, ale viditelné, omezená délka
2. `POST` – skryté, neomezená délka (i upload souboru), složitější manipulace (např. Zpět nefunguje jednoduše).

Informační systém je založen na databázi, proto je nutné, aby programovací jazyk serveru měl prostředky pro přístup k databázi.

Komunikace s databází se musí provést celá v jednom skriptu, jelikož skript je prováděn při každém načtení stránky a pak skončí a ztratí stav.

Práce s databází: připojení a autentizace u serveru, připojení databáze, zasílání dotazů a převzetí výsledků, ukončení spojení.

Výsledky dotazů jsou ve tvaru kolekce struktur, ale jazyky tyto typy nepodporují. Je potřeba výsledky převést na známé typy. PHP: po `mysql_query()` se volá `mysql_fetch_row()`, která zpřístupní strukturu jako pole jednotlivých hodnot.

Abstraktní databázová vrstva (ADL) je obalová vrstva nad konkrétní implementací komunikace s databází. Výhodou je, že jakákoli změna komunikace (nové verze, nebo změna typu serveru – z MySQL na ORACLE) se provede centrálně na jediném místě. Navíc takto napsané systémy jsou nezávislé na použitém databázovém řešení, což je velice žádoucí.

Používané implementace ADL:

- ADOdb – hodně přenositelná, ale velká a těžkopádná
- PEAR DB
- PDO – přímo v PHP od verze 5.1
- Velmi často stačí vlastní řešení pomocí obalových funkcí.

5 Klientská část portálového informačního systému

Klientský JavaScript je JavaScript implementovaný ve webovém prohlížeči, který využívá kromě jiného hlavně DOM. Takto dokáže ovlivňovat vzhled a chování webových stránek, vzniká dynamické chování.

Skript ošetřuje události, oproti serveru, který je dávkovým programem. JavaScript si je musí registrovat pro příjem a uvést ošetřující funkci.

Kompatibilita je u klienta podstatná, server je nasazen a provozován v daném prostředí, ale uživatelé mají prostředí různorodá. Existuje několik prohlížečů a to v několika verzích.

DOM (Document Object Model) je standard reprezentace HTML a XML objektovým modelem. Je nezávislý na platformě a prohlížeči (i když prohlížeče jej neimplementují stejně). Dokument transformovaný z HTML/XML do DOM lze jednoduše číst a upravovat pomocí JavaScriptu a tím tvořit dynamické stránky u klienta.

Globální objekt DOM `window` obsahuje odkazy na okolní okna, rámce, prohlížeče, historii apod., hlavně objekt `document`. Všechny globální proměnné pod `window` spadají.

Hlavní objekt DOM z pohledu dynamických stránek je `document`. Obsahuje seznamy odkazů, tabulek, obrázků, kotev, apletů i skriptů, také přímý přístup k položkám formuláře a v neposlední řadě ke stromu HTML/XML dokumentu.

Dynamický zápis do stránky se provádí pomocí `document.write()`, řetězec se vypíše za blok, ve kterém je kód skriptu.

6 Metodika a technologie návrhu a vytváření informačního systému v internetovém prostředí s relační databází

Modelujeme vždy nejsložitější případ protože vždy se může najít jediný člověk/zboží/..., které se vymyká všem ostatním a je potřeba jej ošetřit. Například zaměstnanec s více typy mezd, rezident v jiném státě apod.

Fáze vzniku informačního systému:

- Formulace požadavků na úrovni fyzického systému
- Datová a funkční analýza (oddělení obecného od zvláštního - abstrakce)
- Návrh informačního systému (implementační dokumentace)
- Implementace informačního systému

Projektová dokumentace není samoúčelná, je také důležitá pro zaznamenání dohodnutých kroků (obrana výrobce proti uživateli).

Prohlášení o cílech je základním dokumentem, který specifikuje uživatel. Není příliš přesný, ale neformálně popisuje požadovanou funkčnost systému.

Například: Systém musí umět autorizaci uživatelů, výběr zboží, platby, ...

Požadavky na systém jsou opět uživatelské dokumenty, které blíže specifikují funkce systému (stále ještě neříkají, jak se mají implementovat). Bývají hierarchické.

Například: Systém při autorizaci musí provádět autentizaci jménem a heslem, obsahovat registraci a odeslání zprávy s přihlašovacími údaji. Hierarchicky podříztený dokument pak může obsahovat například výčet možných položek při registraci.

Požadavky jsou číslovány, jejich označení se později využívá např. v *plánu testování.*

Plán testování v požadavcích na systém specifikuje přesné požadavky na některé podmínky – systém musí obsahovat, musí otestovat, může nabývat hodnot apod. Testování musí probíhat průběžně v celém procesu návrhu a implementace.

Integritní omezení jsou také součástí uživatelských požadavků na systém. Specifikuje unikátnost, obory hodnot, vazby apod.

Interakce se systémem v požadavcích na systém uvádí *sezení*, které popisují komunikaci:

1. *Účel:* (k čemu je toto sezení)
2. *Uživatel:* (kdo může provádět)
3. *Vstup:* (co zadá)
4. *Výstup:* (co získá)
5. *Chyby:* (podmínky a průběh nestandardního ukončení)

6. *Poznámka:*

Systémové požadavky v požadavcích na systém přímo specifikují architekturu a použité metody/nástroje.

Dodávky uživateli obsahují požadavky uživatele na jednotlivé body, ve kterých bude dodána část systému a bude uhrazena (dokumenty, plány, implementace, manuály, ...).

Analýza požadavků na systém hledá konflikty a nepřesnosti v požadavcích. Provádějí ji vývojáři a konflikty musejí vyřešit s uživateli. Vzniká tak revidovaný dokument požadavků na systém a první část specifikačního dokumentu.

Specifikační dokument je již tvořen vývojáři a obsahuje ještě větší zpřesnění požadavků na systém. Vývojář tak přesně říká, co hodlá vytvořit a je již jasné, co přesně má IS dělat. Přesnost je na úrovni uživatelského manuálu, často se vytváří souběžně.

Například: Autentizační obrazovka obsahuje tyto prvky, mají takovýto význam a tímto tlačítkem se ...; Když se tohle nepovede, pak se tohle stane apod.; Specifikuje také datové typy, použité nástroje, jazyky atd.

Návrhový dokument již přechází na specifikaci toho, jak systém implementuje funkce. Specifikuje datové struktury, návrh databáze, chování modulů, objektů a metod. Tento dokument již není příliš důležitý pro uživatele, spíše vnitřně pro tým vývojářů.

Struktura návrhového dokumentu: titulek, datum, verze, úvod; související dokumenty (odkazy na předchozí dokumenty a sekce), vyšší úroveň návrhu (dekompozice a diagramy – ER) a nakonec specifikace schématu, transakcí, procedur.

Oponentura návrhu (design review) je prováděna na konci procesu návrhu. Realizační tým formálně kontroluje popř. verifikuje návrh. Hledají se inkonzistence se specifikačními dokumenty, víceznačnosti a nekompletnosti, místa, která lze zefektivnit, rizika.

Kódování, předávací testy bývají mnohem kratší nežli návrh (v oblasti IS), zvláště při využití knihoven nebo generátorů aplikací.