32. Postrelační a rozšířené relační databáze (objektový a objektově relační databázový model - struktura a operace; podpora práce s XML a JSON dokumenty v databázích). + vlastnosti objekt. databází

Na počátku vývoje databází se nejúspěšnějším modelem stal relační model. Jeho výhodou jsou matematické formální základy, jednoduché modelování, vysoká míra standardizace a datový model, který jde dobře uplatnit na mnoho běžných problémů. Postupně však vznikala další paradigmata, např. objektově orientované programování, která si vyžádala rozšíření/úpravu základního relačního modelu o další funkcionalitu.

Zopakujme krátce základní rysy relačního modelu. Datovým modelem jsou tabulky (relace) s atomickými datovými typy. Tabulky mají primární klíče, které slouží k jednoznačné identifikaci jednotlivých řádků. Jednotlivé záznamy v tabulkách mohou být propojeny prostřednictvím cizích klíčů. Vztahy M:N jsou modelovány prostřednictvím vazebních tabulek. Výpočetní model je založen na hodnotách ve sloupcích tabulky, v případě vyhledávání více záznamů je využíván tzv. kurzor na řádky, pomocí kterého je možné tabulkami procházet. Relační datový model je standardizován jazykem SQL.

Objekty v databázích

Stejně jako data mohou být ukládána v relačních databázích, mohou být také ukládána v objektech. Datový model se poměrně zásadně liší – máme třídy popisující atributy a operace a každý objekt má své unikátní OID (objektové ID). Atributy nemusí být pouze atomické, skalární, ale může se jednat o složitější datové struktury. Vztahy objektů se reprezentují s pomocí OID (podobný princip jako cizí klíče), ovšem vztahy M:N je možné modelovat přímo (objekt má v sobě seznam/pole dalších objektů). Při výpočtech pak je možné na základě OID navigovat přes propojené objekty. Byly pokusy tento model standardizovat v jazycích ODL (object data language) a OQL (object query language), ovšem to se neuchytilo, protože tento přístup nedává příliš smysl – objekty/třídy jsou definovány přímo v programu/programovacím jazyce a je s nimi také přímo manipulováno.

Jsou v zásadě 3 možnosti, jak s objekty nakládat, co se týče perzistence dat:

- nativní objektové databáze
- využití mapování do jiného typu databáze, např. relační (object-relational mapping apod.)
- přímá podpora pro objekty v relačních databázích (tzv. objektově-relační model)

Objektově-relační model

Podpora pro objekty v relačních databázích byla zavedena v SQL-1999 (implementováno ve většině dnešních databází – Oracle, Postgres, ...). Tato verze umožnila zadefinovat uživatelsky definované datové typy (abstraktní datový typ), které fungují podobně jako třídy v klasickém OOP, tzn. mají atributy, metody, a je možné dědit. ADT může být použit jako typ sloupce tabulky, nebo jako typ celé tabulky. V prvním případě vznikne vnořená struktura, která nemá unikátní OID. Ve druhém případě jsou řádky tabulky objekty, které mají unikátní OID. Navigace a výpočet v modelu pak může kombinovat jak cizí/primární klíče, tak navigaci přes OID.

K vytvoření typu se využívá příkaz CREATE TYPE, pro dědičnost se využívá klauzule UNDER. Vztahy přes OID jsou pak realizovány pomocí klíčového slova REF:

```
CREATE TYPE t_person AS (
  id_no INTEGER, name VARCHAR(64)
) INSTANTIABLE NOT FINAL REF (id_no);

CREATE TYPE t_employee UNDER t_person AS (
  salary REAL
) INSTANTIABLE NOT FINAL
INSTANCE METHOD annual_salary()
  RETURNS REAL CONTAINS SQL;

-- instance method is dynamic
CREATE INSTANCE METHOD annual_salary()
  RETURNS REAL
FOR t_employee
BEGIN
  RETURN SELF.salary * 12;
END;

-- inheritence in typed tables
CREATE TABLE person OF t_person;
CREATE TABLE employee OF t_employee UNDER person;

-- UDT as a column type in relational tables
CREATE TABLE faculty_employee (
  id_faculty INTEGER PRIMARY KEY,
  employee t_employee,
  dept CHAR(3) FOREIGN KEY REFERENCES dept,
  superior REF(t_employee)
);

SELECT e.name, e.annual_salary() FROM employee e;

SELECT fe.id_faculty, fe.employee.name,
  fe.employee.annual_salary(), fe.superior -> name
FROM faculty_employee fe
WHERE fe.dept = 'CSD' AND fe.employee.salary > 20000;
```

Ukládání aplikačních objektů do databází

V OOP paradigmatu se aplikace skládá z objektů. Každý objekt má nějaký stav (hodnotu atributů) a chování (metody). Objekty jsou typicky instancí nějaké třídy. Pokud dva objekty mají stejný vnitřní stav, říkáme, že jsou si rovny, ale nejsou si identické (aby byly identické, musely by mít shodné OID). Rozlišujeme objekty:

1. tranzientní (uložené v paměti (paměti aplikace), typicky spravované garbage collectorem)
2. permanentní – i pokud už nejsou na ně odkazy, jsou ponechány a garbage collector je nesmaže. Tyto objekty chceme zachovat v nějakém trvalém uložišti pro zajištění perzistence.

Na perzistentní uložiště objektů máme tzv. **CCSP požadavky**:

- **Continuity** – dvě instance stejného objektu načteného z úložiště musí být identické
- **Cohesion** – skupina propojených objektů (přes atributy) musí být společně uložena
- **Spatio-temporal priority** – dva objekty reprezentující stejnou reálnou entitu (na stejném místě a ve stejném čase) musí být identické

Máme dva přístupy k perzistenci objektů

1. **Nativní objektová databáze** (např. ObjectBox)
   - nutnost ji integrovat do aplikačního kódu
   - jednoduchá na použití, ovšem není to znovupoužitelný přístup (DB nejde použít na nic jiného než objekty)
2. **Mapování objektů na neobjekty**
   - **Object-relational mapping (ORM)** – mapování objektů na relační tabulky, velmi populární (Hibernate, EF Core)
   - **Object-document mapping (ODM)** – mapování objektů do NoSQL dokumentů v dokumentových databázích (např. MongoDB). Tyto databáze jsou založeny na vzoru klíč-hodnota, OID se tedy stává klíčem a atributy hodnotou.

XML

Kromě požadavků na integraci OO do relačních databází vznikl v průběhu let také požadavek na ukládání a efektivní manipulaci dokumentů v běžně využívaných formátech. Základním příkladem takového formátu je XML. XML dokument se skládá z:

1. deklarace – verze, kódování apod,.
2. elementů – ~~obsah mezi tagy~~ logická část se značkami, atributy a obsahem (mezi start a end tagem)
3. tagů – markup konstrukty začínající < a končící > udávající strukturu dokumentu
4. atributů – jméno-hodnota dvojice v rámci tagů

Níže můžeme vidět příklad XML dokumentu. Na začátku je deklarace, tag qanda má atribut seq="1", obsahem je pak například "William Jefferson Clinton".

- značkovací jazyk
- text-based file format

deklarace

tag

atribut

element = logická část se jménem, atributy a obsahem mezi tagy

```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML

Korektní XML dokument musí být tzv. **well-formed** (syntakticky správný) a zároveň **validní** (sémanticky správný obsah). Pro ověřování sémantické správnosti se zavádí tzv. **XML schema**, které umožňuje limitovat obsah a tvar XML dokumentu, tj. určit, které dokumenty jsou validní. XML schéma se typicky popisuje externím XML dokumentem. Máme několik typů XML schema, např. Document Type Definition (DTD) nebo W3C XML Schema (schéma je definováno v tzv. XSD dokumentech). Níže můžeme vidět příklad XSD dokumentu, který říká, že na nejvyšší úrovni je tag breakfastMenu, ve kterém je vnořena sekvence food tagů, kdy každý se skládá z jména, ceny, popisu a kalorií, přičemž tento obsah musí být korektního typu, aby byl XML dokument validní.

- XML schema (XSD)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified"
  targetNamespace="http://mypub.com/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="breakfastMenu" type="breakfast-menu-type"/
  <xs:complexType name="breakfast-menu-type">
    <xs:sequence>
      <xs:element type="food-type" name="food"
        maxOccurs="unbounded" minOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="food-type">
    <xs:all>
      <xs:element type="xs:string" name="name"/>
      <xs:element type="xs:string" name="price"/>
      <xs:element type="xs:decimal" name="description"/>
      <xs:element type="xs:integer" name="calories"/>
    </xs:all>
  </xs:complexType>
</xs:schema>
```

Rozlišujeme datově zaměřené a dokumentově zaměřené XML dokumenty. První zmíněné mají pravidelnou strukturu a jsou vhodné pro ukládání. Ty druhé jsou vhodné spíše pro čtení lidmi, vizualizaci výstupu (např. HTML).

Pro **získávání informací z XML** je možné využít dva standardy – **XPath** a **XML Query**. XPath umožňuje procházet obsahem podle zadané cesty od kořenového uzlu:

```
(: get food names with the price below 100 :)
/breakfastMenu/food[price < 100]/name

(: get food with the name staring with P letter :)
/*/food[starts-with(name,'P')]

(: get a string of all names separated by the semicolon :)
string-join(//name/text(),'; ')
```

XML Query je funkcionální programovací jazyk pro dotazování a transformaci XML dokumentů:

```
for
  $i in fn:doc("m.xml")/breakfastMenu/food[name = 'Pilsner Urquell'],
  $j in fn:doc("m.xml")/breakfastMenu/food[price < $i/price],
  order by $j/price, $j/name
return
  <cheaper-than-pilsner>{ $j/name, $j/price }</cheaper-than-pilsner>
```

XML v databázích

Moderní databáze typicky umožňují přímo ukládat XML a dotazovat se nad ním. Existují však také nativní XML databáze, které poskytují velmi rychlé operace nad dokumenty. Jsou různé způsoby, jak je podpora pro XML implementována, např:

1. Wrapper na straně databáze, který při SQL příkazu zpracuje fyzicky uložený XML dokument
2. Middleware
3. Navázání XML dokumentů na objekty

V relačních databázích se využívají 3 přístupy:

1. XML data uložena jako CLOB/BLOB (čistá binární/textová data)
   - neposkytuje jednoduchý způsob úprav nebo dotazování
2. XML data jsou naparsována a uložena do relační tabulky
   - jednoduché dotazování a úpravy
   - ztrácíme ovšem původní XML dokument
3. XML data jsou uložena ve specializovaném SQL XML datovém typu
   - opět praktické dotazování a úpravy

uloží původní XML dokument a rozumí mu (umí se nad ním dotazovat a stavět indexy)

Specializovaný XML datový typ byl zaveden v SQL 2003. Umožňuje dotazování nad XML dokumenty s pomocí jazyka XMLQuery i XPath:

```
CREATE TABLE resolutions_xml (
  id number PRIMARY KEY, legis_num NUMBER,
  resolution XMLType
);

SELECT id,
  extract(resolution,
    '/resolution[@public-private="public"]/action')
  AS action,
  extractValue(resolution,
    '/resolution[congress="108"]/official-title')
  AS title
FROM resolutions_xml
WHERE existsNode(resolution,
 '/resolution[legis-num="558"]') = 1;

SELECT xmlquery('
for␣$r␣in␣$res/resolution
let␣$a␣:=␣$r/action
where␣$a/action-date="20040311"
order␣by␣$r/legis-num␣ascending
return
<all-sponsors>
{$a/action-desc/sponsor}
{$a/action-desc/cosponsor}
</all-sponsors>'
  passing resolution as "res"
  returning content
)
FROM resolutions_xml;
```

JSON - JavaScript Object Notation

JSON je jazykově nezávislý datový formát pro přenos dat vycházející z Javascriptu (každý JSON soubor je zároveň i validní Javascript soubor). Pro vyjádření stejného obsahu je typicky stručnější než XML, je více lidsky přívětivý. Na druhou stranu nemá např. příliš rozšířenou standardizovanou validaci podle schématu. SQL přidalo podporu pro JSON v roce 2016. Nejedná se o specializovaný typ jako v případě XML, ale pouze o funkce pracující nad textovými datovými typy:

- json_exists, json_value a json_query pro dotazování
- json_table pro transformaci na SQL tabulku
- json_object a json_array pro tvorbu JSON data

problém s indexy → většina databází dnes již má specializovaný typ a podporuje indexy (nejlépe na tom PostgreSQL)

```
CREATE TABLE t (
  jcol CLOB CHECK (jcol IS JSON)
);

INSERT INTO t (jcol) VALUES (json_array(
  json_object('id' value 10, 'name' value 'Anna'),
  json_object('id' value 20, 'name' value 'Bob')
));

SELECT jt.* FROM t, JSON_TABLE (
  jcol, '$[*]' COLUMNS (id NUMERIC PATH '$.id',
    name VARCHAR(255) PATH '$.name')
) jt
WHERE json_exists(jcol, '$.id')
  AND json_exists(jcol, '$.name');
```

V NoSQL databázích je podpora pro JSON značně rozšířenější, protože převod JSONu do key-value formátu je velmi přímočará (viz např. MongoDB).

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele Fifinas.

- méně verbose oproti XML (menší overhead při přenosu)
- lépe čitelný člověkem
- JSON Schema
- JSON path language
  - $ - root object or array
  - . - child operator ($.store.books)
  - [ ] - array indexing or filters ($.store.books[0].name)