

OO a OR databáze

slajdy k přednášce NDBI001

Jaroslav Pokorný
MFF UK, Praha
pokorny@ksi.mff.cuni.cz



Obsah

1. Úvod - proč více databázových technologií
2. Objektově orientované databáze (ODMG 93)
3. Objektově relační databáze
 - 3.1 Rozšiřitelnost, uživatelsky definované typy a funkce
 - 3.2 Opravdové ORSŘBD (SQL:1999, 2003 a další)
4. Závěr

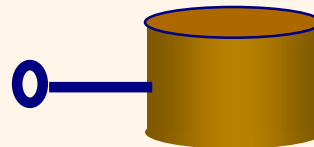
Proč více db technologií

Požadavky nových aplikací:

- nové typy objektů a funkcí
- OO analýza a návrh vs. relační db

"Relační databáze je podobná garáži, která vás nutí rozmontovat vaše auto a uložit díly do malých zásuvek..."

SŘBD



Databáze



Spreadsheet



Fotky



Mail



Mapy



Dokumenty

Cíl: integrace a správa dat
v jednom systému

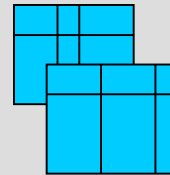
Objektově orientované databáze

objektový datový model:

- je v souladu s viděním světa (entita \Rightarrow objekt)
- definice složitých objektů a jejich manipulace

RSŘBD

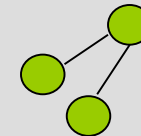
Výkonné OLTP
Dostupnost dat
Utajení
Prostředky pro správu dat
Standardní jazykové rozhraní
Řízení paměti
Souběžné zpracování dat
Integrita



Funkcionalita relačních a OO
SŘBD

Operace na složitých objektech
Rekurzivní struktury
Abstraktní datové typy
Rozhraní k OO jazyku
Složitě transakce

OOSŘBD





Objektově orientované databáze

1993: konsorcium vůdčích výrobců OOSŘBD ⇒
návrh standardu ODMG-93.

- nadmnožina obecnějšího modelu Common Object Model (COM) vytvořeného skupinou OMG. Převzat byl jeho definiční jazyk IDL.
- dotazovací část Object Query Language (OQL), která souvisí s koncepcí dotazovací části standardu SQL92.
- rozhraní k OO PJ C++, Smalltalk (k Java: nahrazeno Java Data Objects (JDO))

2001: skupina rozpuštěna (verze ODMG 3.0)

OOPJ + SŘBD = OOSŘBD

Základní koncepty ODMG-93

- *třída (nebo typ), instance (nebo objekt), atribut, metoda a integritní omezení*
 - třída - šablona pro instance (objekty), které mohou sdílet atributy a metody.
 - ◆ doména atributů: primitivní typ dat, abstraktní typ dat (ADT), nebo odkaz na třídu.
 - ◆ metoda je funkce (její implementace je skryta) aplikovatelná na instance třídy (výpočet založený na hodnotách atributů).
- *identifikátor objektu (OID)*
 - každý objekt má jednoznačný identifikátor, prostřednictvím kterého lze z databáze získat odpovídající objekt.



Základní koncepty ODMG-93

- *zapouzdření*

- data jsou “zabalena” spolu s metodami. Jednotkou zapouzdření je objekt. Metody jsou platné pouze na příslušných objektech, se kterými jsou zapouzdřeny.

- *hierarchie tříd, dědění*

- podtřída \Rightarrow hierarchie
- dědění je proces znamenající pro podtřidu osvojení všech atributů a metod z nadtřidy.
- vícenásobné dědění (\Rightarrow problémy např. řešení konfliktů stejných jmen zděděných atributů a metod).



Nástup OO db technologie

Zdroje: OO programování, OO analýza a návrh, relační SRBD

- Objektově relační mapování (ORM)

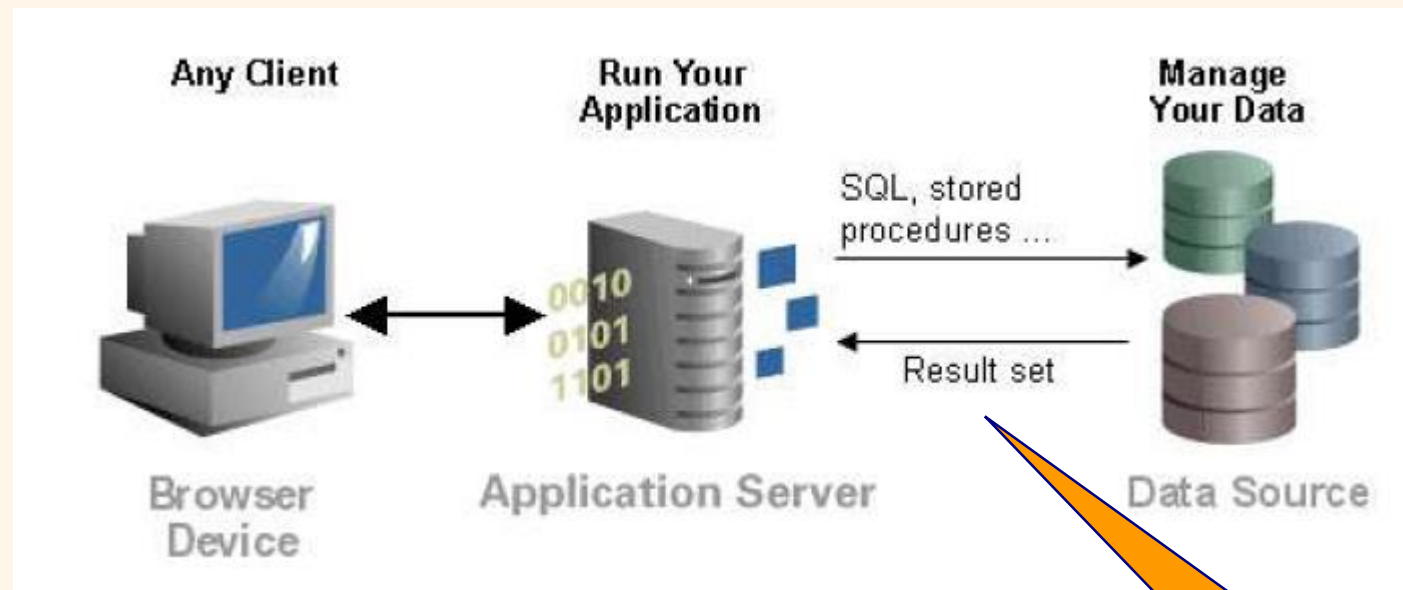
Př.: Hibernate (rozhraní: Session, Transaction, Query)

TopLink (ORM aplikace vlastněná Oracle, Inc.)

Přístup k objektům: např. Hibernate Query Language → SQL; programovací jazyk + metody realizující vstup do SQL databáze

- problém: méně sémantiky v relacích, impedance mismatch v přístupu k objektům

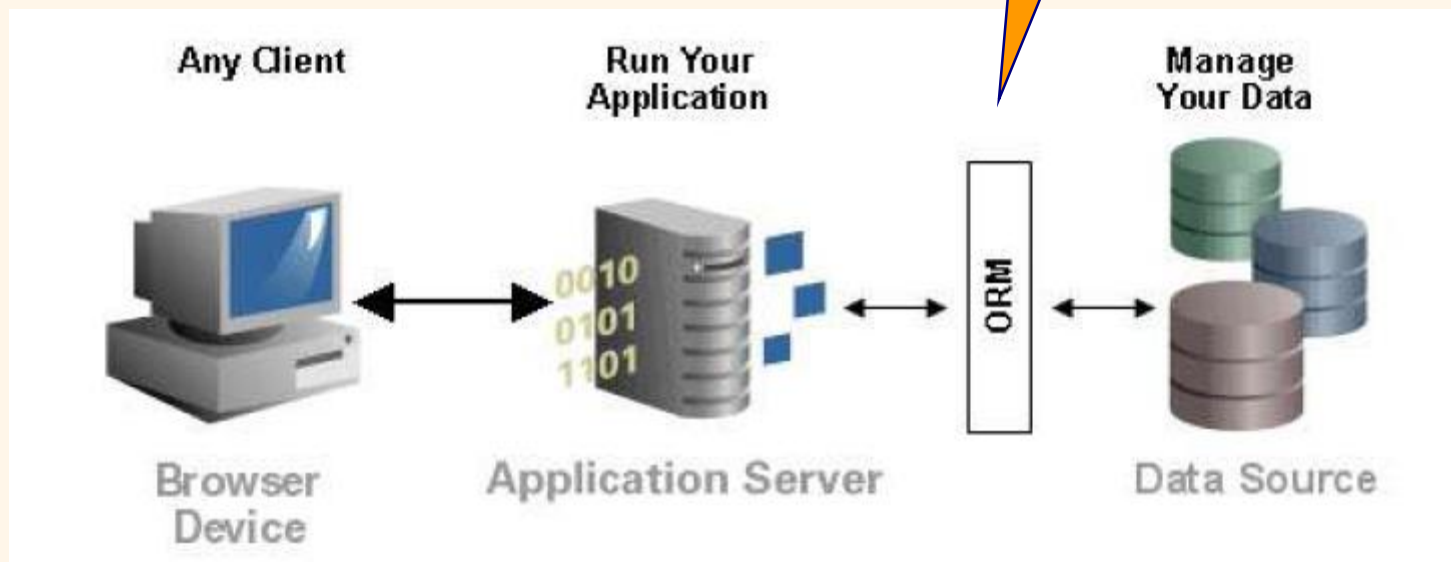
Provoz bez ORM



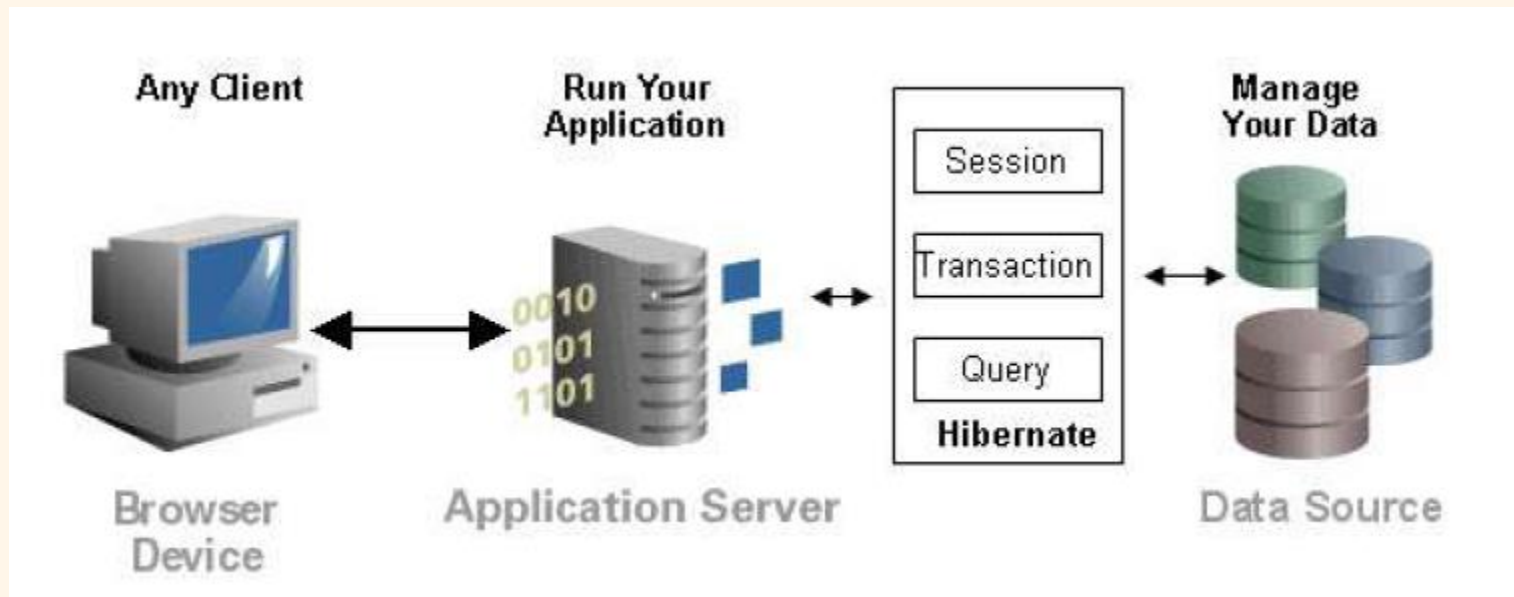
prostřednictvím
API JDBC

Provoz s ORM

15%-20%
pomalejší než
s JDBC



Příklad: Hibernate





Nástup OO db technologie

- 2. pol. 80. let - OO databáze
Př.: O2 (→ Unidata → Informix → IBM)
ObjectStore (dnes Execlon)
Versant Object Database, Objectivity/DB,
GemStone, GemSTone/J, DB4Objects,
Jasmine
Dotazování: OQL – neúplný, jinak: např.
Objectivity/SQL++
- částečný neúspěch: nenabídly pružnost a
výkonnost relačních SŘBD



Nástup OR db technologie

Důvody:

- obdržet maximum z rozsáhlých investic do relační technologie (data, nabyté zkušenosti),
- využít výhody v pružnosti, produktivitě a provozních přínosů OO modelování,
- integrovat databázové služby do systémů výroby a dalších aplikací.



Nástup OR db technologie

- 90. léta - OR přístup - ORSŘBD
 - kombinace OO a relačních SŘBD
 - Př.: 1992: UniSQL/X, dále: HP - OpenODB (později Oadapter)
 - 1993: Montage Systems (později Illustra) - komerční verze Postgres
- dnes: DB/2, INFORMIX, ORACLE, Sybase Adaptive Server+Java, OSMOS, Unidata



Nástup OR db technologie

- počátky: univerzální servery
 - rozšiřitelnost relačního přístupu
 - Příklady: DB2 Common Server (1995!), později DB2 Universal Database, Informix, ORACLE 7.3

Princip: **zdola nahoru**

Použití: pro již existující data v relační DB



Objektově relační databáze

Dva přístupy:

- *univerzální paměť*, kdy všechny druhy dat jsou řízeny SŘBD), jde o integraci (různými způsoby!)
 - ⇒ univerzální servery
- *univerzální přístup*, kdy všechna data jsou ve svých původních (autonomních) zdrojích

Technika: middleware

- ◆ brány (min. dva nezávislé servery)
- ◆ zobrazení schémat, transformace dotazů
- ◆ objektové obálky: Persistence Software, Ontologic, HP, Next, ... (problémy: výkon)
- ◆ DB založené na Web

Rozšiřitelnost, uživatelsky definované typy a funkce

*Možnosti ADT:
black box
white box*

Požadavek: manipulace BLOB (v RSŘBD atomický)

Rozšiřitelnost: možnost přidávání nových datových typů + programů (funkce) „zabalенých“ do speciálního modulu

⇒ UDT (*uživatelsky definované typy*)

UDF (*uživatelsky definované funkce*)

Problém: zapojení do relačního SŘBD (včetně SQL !)


DB/2: relační extendery

Informix: DataBlades

ORACLE: cartridges

Sybase: Component Integration Layer.

univerzální servery




Rozšiřitelnost, uživatelsky definované typy a funkce

Př.: DB/2 v r. 2006:

- MapInfo
- NetOwl (přirozený jazyk v bussiness intelligence)
- EcoWin (časové řady, makroekonomická časové řady, ...)
- GIS a prostorové objekty
- SQL expander (matematické, finanční, konverzní, ... funkce)
- VideoCharger (audio a video objekty v reálném čase)
- text, XML, audio, video, obrázky
- FormidaFire (integrace heterogenních dat)
- ...

Př.: Informix v r. 2006:

- C-ISAM, Excallibur Text Search, Geodetic, Image Foundations, Spatial, TimeSeries, Video Foundation, Web



Rozšiřitelnost, uživatelsky definované typy a funkce

Standardizace: SQL/MM

(Př.: Full-Text - řeší ADT + odpovídající funkce)

Implementace: technologie „plug in“ pomocí
různých technik:

Př.: DataBlades - přímý přístup k databázovému
jádro

ORACLE 7.3 - architektura více serverů a API



Příklad - textový extender

```
SELECT časopis, datum, titul
FROM ČLÁNKY
WHERE CONTAINS(text_článku, ('databáze' AND
                             ('SQL' | 'SQL92') AND NOT 'dBASE')) = 1;
```

Další funkce: NO_OF_MATCHES (kolikrát se zadaný vzorek vyskytoval v textu), RANK (hodnota pořadí v odpovědi na základě nějaké míry).

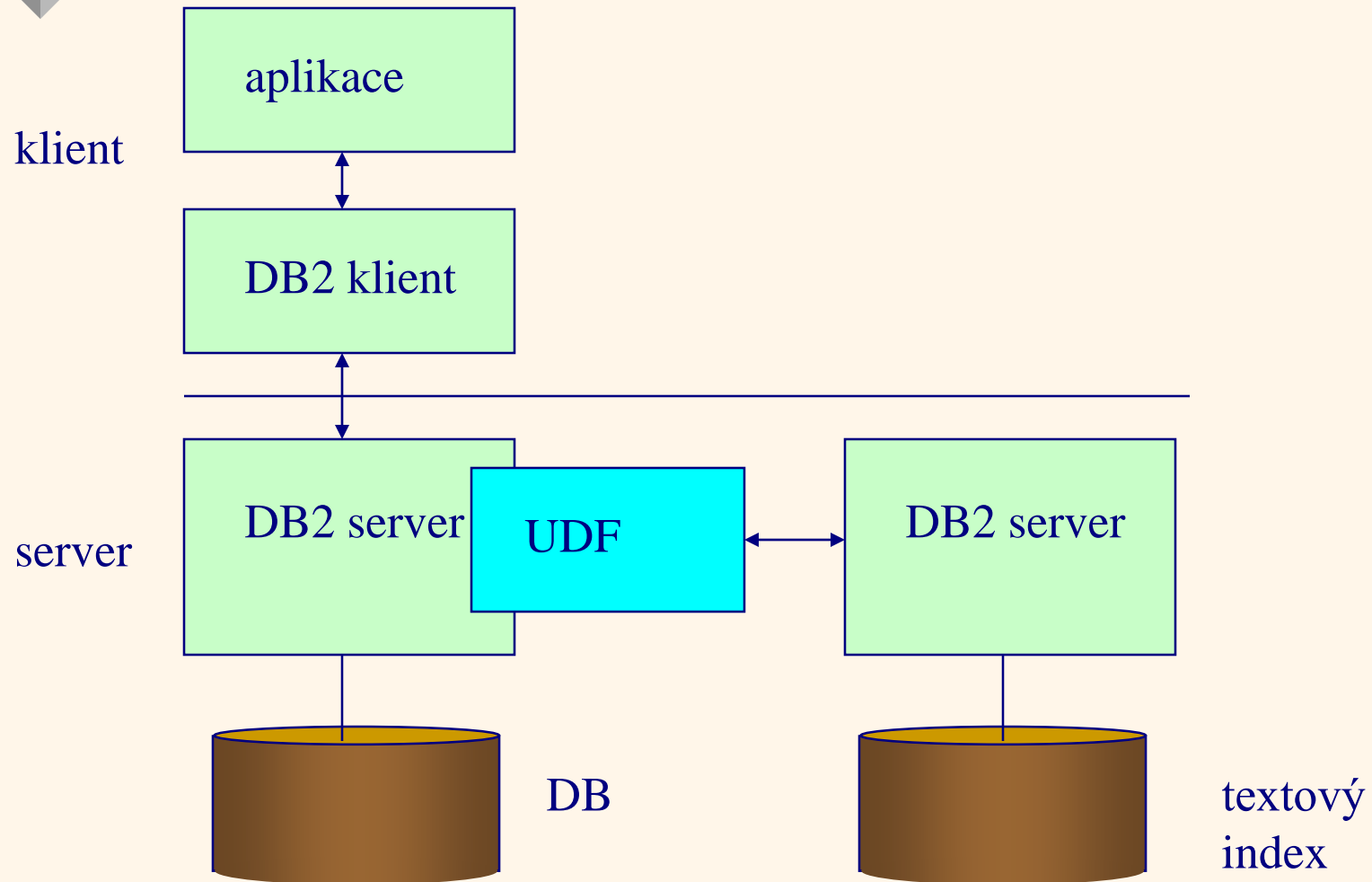
```
SELECT časopis, titul
FROM ČLÁNKY
WHERE NO_OF_MATCHES (text_článku, 'databáze') > 10;
SELECT časopis, datum, titul, RANK(text_článku, ('databáze'
                                                AND ('SQL' | 'SQL92') )) AS relevantni
FROM ČLÁNKY
ORDER BY relevantni DESC;
```



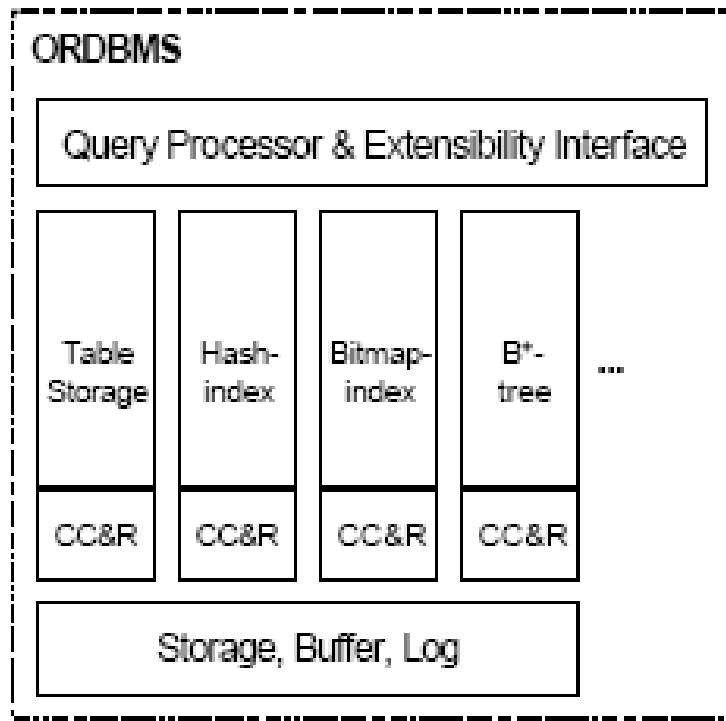
Architektura známých produktů

- přidání zvláštního aplikačního rozhraní (API) a speciálních serverů (také ORACLE 7.3 – viz např. CONTEXT, Media Server, OLAP),
- simulace OR na úrovni middleware (také ORACLE 7.3 - viz např. část Spatial Data Option),
- úplné přepracování databázového stroje (např. Illustra Information Technology),
- přidání OO vrstvy k relačnímu stroji (např. INFORMIX Universal Server, IBM D2/6000 Common Server, Sybase Adaptive Server + Java).

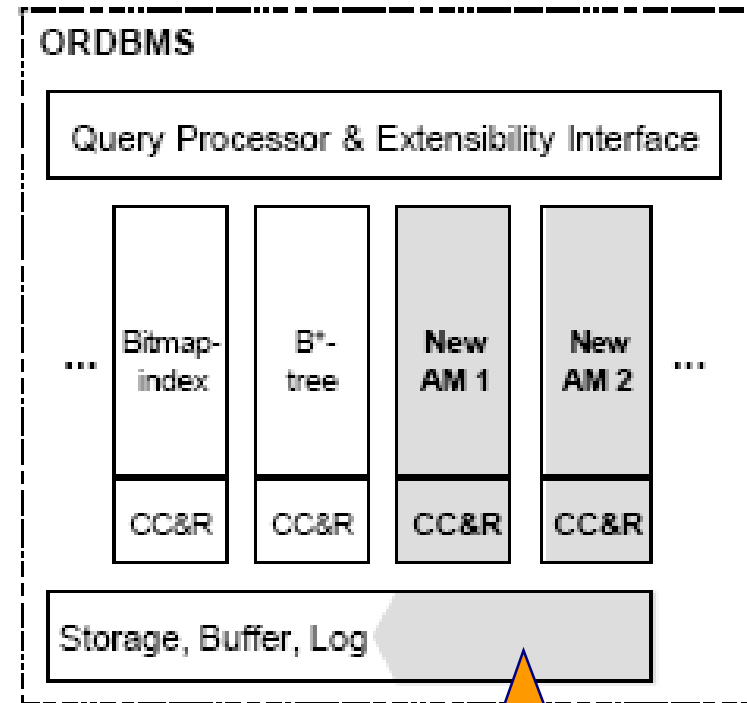
Interakce DB a textového extenderu v DB2



Architektury rozšiřitelnosti (1)



a) Standard ORDBMS kernel

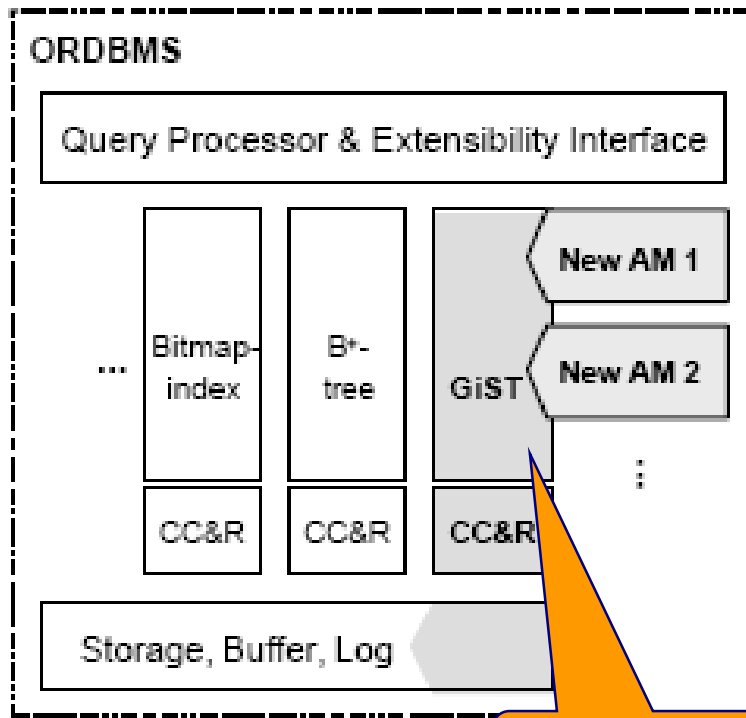


b) Integrating approach

zásah hluboko do jádra

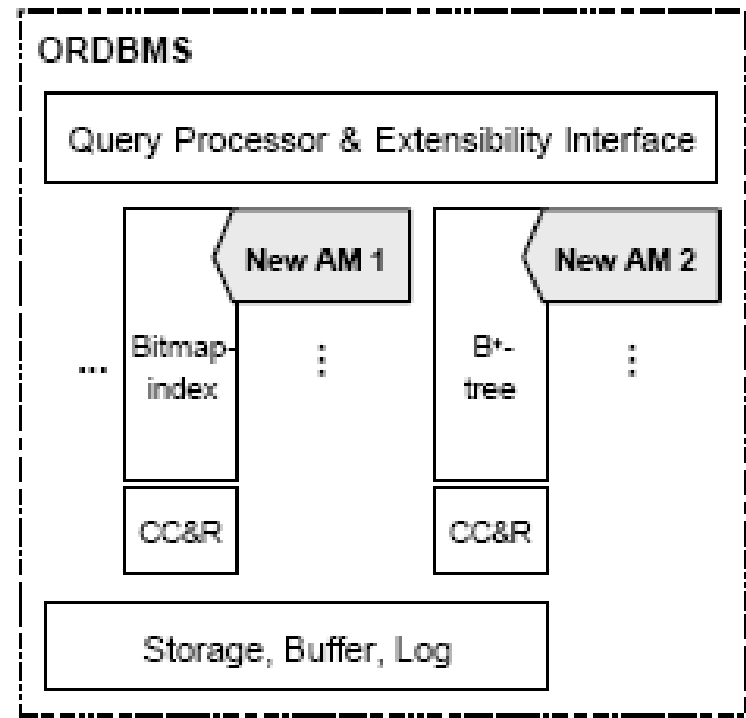
CC&R: concurrency control and recovery
AM: access method

Architektury rozšiřitelnosti (2)



c) Generic approach


je jen jeden



d) Relational approach

GiST: Generalized Search Tree

AM na vrcholu relačního SŘBD



„Opravdové“ ORSŘBD

Won Kim: „rozšiřitelnost je pouze druhotný, i když užitečný rys, jde o důsledek OO přístupu“

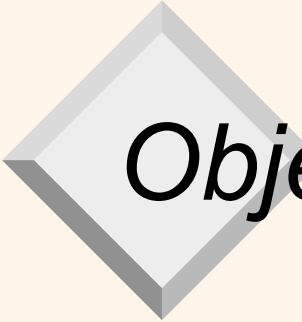
Stonebraker: „rozšiřitelnost typu “plug-in” (např. ORACLE) jako sice vhodnou pro konektivitu aplikace-aplikace, nicméně nemá nic do činění s databázovou “plug-in”. Jde o pouhý middleware, který nezakládá OR technologii“.

Požadavky:

- datový model s hlavními rysy ODMG-93
- odpovídající objektový jazyk vyšší úrovně

Řešení: OO rozšíření SQL naplňuje (přibl.) tyto požadavky

Dnes: standard SQL:1999, SQL:2003 + další vývoj



Objektově relační modelování

- Rozšíření relačního modelu o objekty a konstrukty pro manipulaci nových datových typů,
- atributy n-tic jsou složité typy, včetně hnížděných relací,
- zachovány jsou relační základy včetně deklarativního přístupu k datům,
- kompatibilita s existujícími relačními jazyky (tvoří podmnožinu).

Příklad: hnízděná vs. normalizovaná relace

časopis	titul	autoři	klíčová_slova	datum		
				den	měsíc	rok
CW	OLAP	{Kusý, Klas}	{hvězda, dimenze}	23	duben	1998
SN	Databáze	{Novák, Fic}	{RDM, schéma}	15	květen	1998

časopis	titul	autor	klíčové_slovo	den	měsíc	rok
CW	OLAP	Kusý	hvězda	23	duben	1998
CW	OLAP	Kusý	dimenze	23	duben	1998
CW	OLAP	Klas	hvězda	23	duben	1998
CW	OLAP	Klas	dimenze	23	duben	1998
SN	Databáze	Novák	RDM	15	květen	1998
SN	Databáze	Novák	schéma	15	květen	1998
SN	Databáze	Fic	RDM	15	květen	1998
SN	Databáze	Fic	schéma	15	květen	1998

Normalizace do 4NF

časopis	titul
CW	OLAP
SN	Databáze

titul	autor
OLAP	Kusý
OLAP	Klas
Databáze	Novák
Databáze	Fic

titul	klíčové slovo
OLAP	hvězda
OLAP	dimenze
Databáze	schéma
Databáze	RDM

titul	den	měsíc	rok
OLAP	23	duben	1998
Databáze	15	květen	1998

Nevýhody 4NF:

- spojení v dotazech

Nevýhody pouhé 1NF

- ztráta vztahu řádek = 1 objekt

Normalizace do 4NF

časopis	titul
CW	OLAP
SN	Databáze

titul	autor
OLAP	Kusý
OLAP	Klas
Databáze	Novák
Databáze	Fic

titul	klíčové slovo
OLAP	hvězda
OLAP	dimenze
Databáze	schéma
Databáze	RDM

eventuálně



titul	den	měsíc	rok
OLAP	23	duben	1998
Databáze	15	květen	1998

Nevýhody 4NF:

- spojení v dotazech

Nevýhody pouhé 1NF

- ztráta vztahu řádek = 1 objekt

SQL:1999

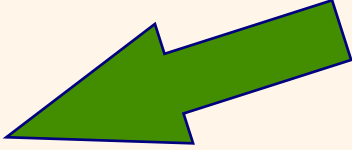
Pět částí:

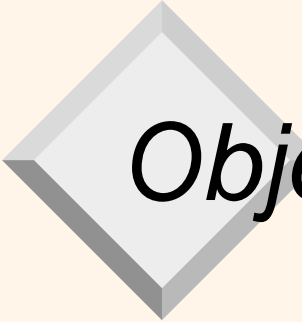
- SQL/Framework 75 str.
- SQL/Foundations 1100 str.
- SQL/CLI (Call Level Interface*) 400 str.
- SQL/PSM (Persistent Store Modules**) 160 str.
- SQL/Bindings 250 str.
(SQL Embedded, Dynamic SQL, Direct invocation)

* alternativa k volání SQL z aplikačních programů (implementace: ODBC)

** procedurální jazyk pro psaní transakcí

SQL:1999

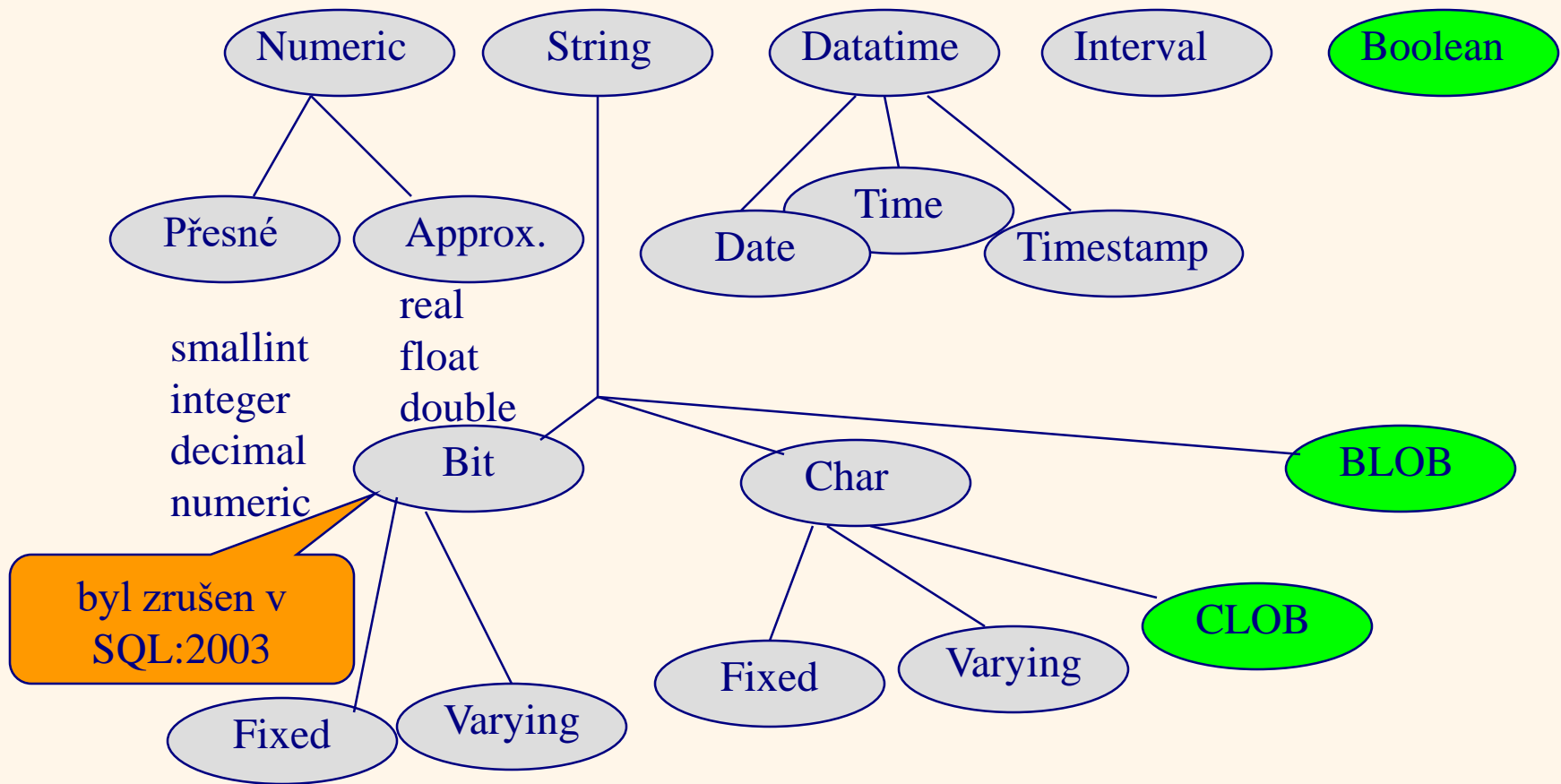
- podpora objektů 
- uložené procedury
- triggery
- rekurzivní dotazy
- rozšíření pro OLAP
- procedurální konstrukty
- výrazy za ORDER BY
- savepoints
- update prostřednictvím sjednocení a spojení



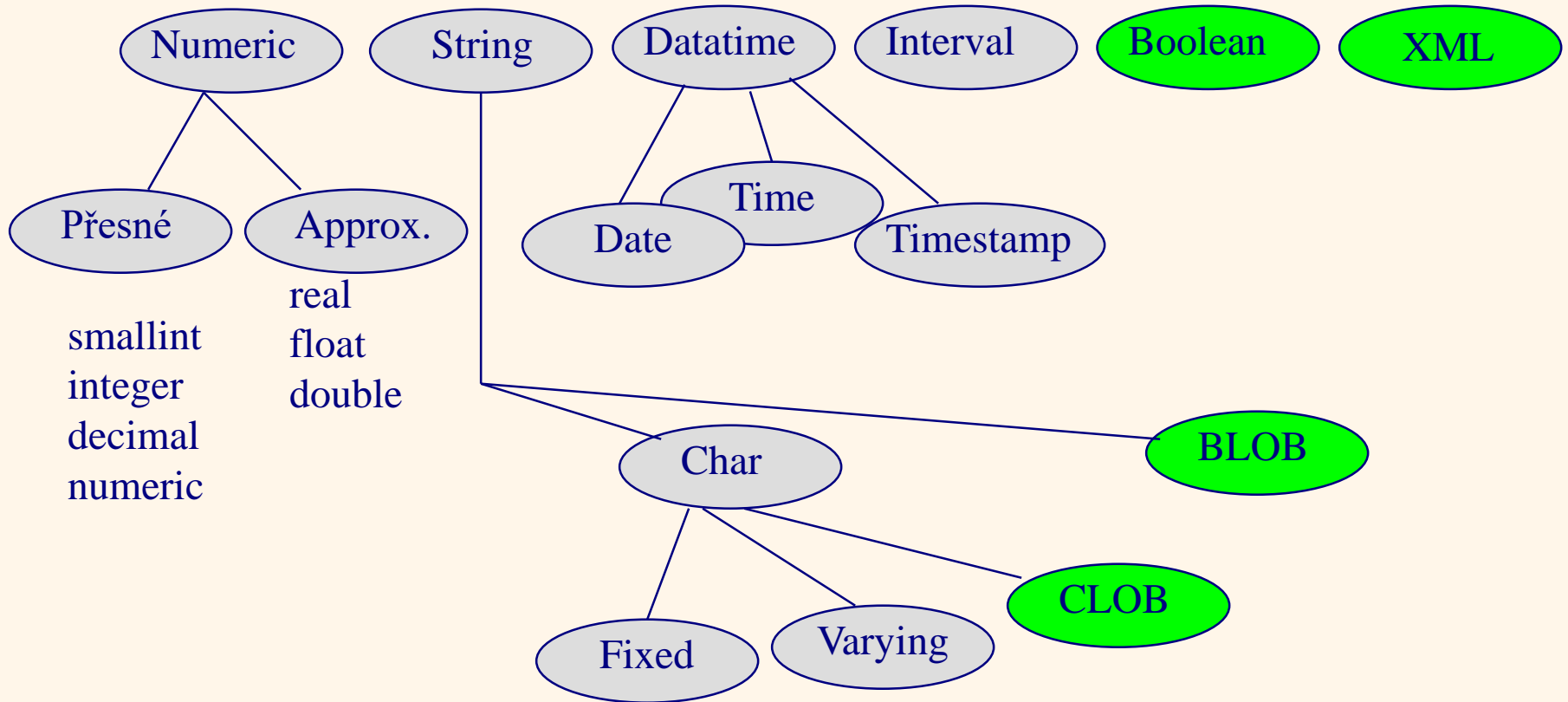
Objekty: od SQL3 k SQL:1999

- SQL3 pro podporu objektů používá:
 - *uživatелеm definované typy (ADT, pojmenované typy řádků a odlišující typy),*
 - *konstruktory typů pro typy řádků a typy odkazů,*
 - *konstruktory typů pro typy kolekcí (množiny, seznamy a multimnožiny),*
 - *uživatелеm definované funkce (UDF) a procedury (UDP),*
 - *velké objekty (Large Objects neboli LOB).*
- Standard SQL:1999 - podmnožina celkové koncepce

Předdefinované typy v SQL:1999



Předdefinované typy v SQL:2003



Typ Boolean

```
SELECT č_odd, EVERY(plat > 20000) AS  
    všichni_bohatí, SOME(plat > 20000) AS  
    někteří_bohatí  
FROM zam  
GROUP BY č_odd;
```

Výsledek:

č_odd	všichni_bohatí	někteří_bohatí
A35	FALSE	FALSE
J48	TRUE	TRUE
Z52	FALSE	TRUE



Další typy v SQL:1999

Konstruované atomické typy:

- reference

Konstruované kompozitní typy:

- array /* podtyp collection */
uspořádaný seznam dané maximální délky
nejsou povolena žádná pole polí nebo
vícedimensionální pole
- row

Pz.: v SQL4 je více podtypů kolekcí (v implementacích rovněž)

Pz.: k typům existují nové funkce (BIT_LENGTH, POSITION, SUBSTRING, ...)

Typ pole

```
CREATE TABLE zpravy(  
  ID INTEGER  
  autoři    VARCHAR(15) ARRAY[20]  
  titul     VARCHAR(100)  
  abstrakt  FULLTEXT
```

- přístup ke složkám pole poziční (pořadovým číslem), např. autoři[3],
- funkce **CARDINALITY**, porovnání =, <>, zřetězení ||, **CAST**
- **UNNEST** (odhnízdění),
- možnost **WITH ORDINALITY** (lze generovat sloupec offset odpovídající pořadovému číslu prvku v poli)

```
SELECT z.ID, a.jméno  
FROM zpravy AS z, UNNEST(z.autoři) as a(jméno)
```



Další typy v SQL:1999

UDT:

- *odlišující typy* (jsou zatím budované pouze na předdefinovaných typech)
- *strukturované typy* (mohou být definované s více atributy, které jsou předdefinovaných typů, typu ARRAY, nebo dalšího strukturovaného typu)
 - ◆ ADT
 - chování je realizováno pomocí funkcí, procedur a metod
 - ADT mohou být organizovány do hierarchií s děděním
 - ◆ pojmenované typy řádků

Odlišující typy

Princip: přejmenování (rozlišení) předdefinovaných typů + odlišné chování

```
CREATE TYPE TYP_MÍSTNOSTI  
AS CHAR(10) FINAL;
```

```
CREATE TYPE METRY  
AS INTEGER FINAL;
```

```
CREATE TYPE KV_METRY  
AS INTEGER FINAL;
```

```
CREATE TABLE místnosti(  
m_id          TYP_MÍSTNOSTI  
m_délka      METRY  
m_šířka      METRY  
m_obvod      METRY  
m_plocha     KV_METRY);
```

OK

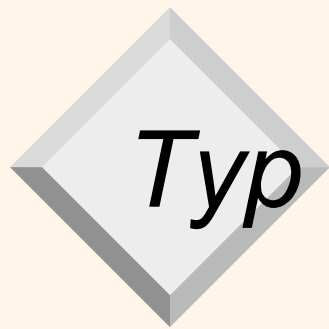
hlásí chybu

```
UPDATE místnosti  
SET m_plocha = m_délka
```

```
UPDATE místnosti  
SET m_šířka = m_délka
```

Pozor: srovnej s pojmem DOMAIN!

Pz.: slabá sémantika: na METRY (KV_METRY) není definováno +.

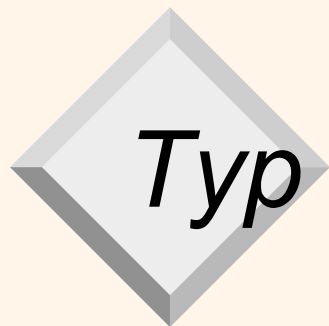


Typ řádku - nepojmenovaný

```
CREATE TABLE osoby (  
    jméno VARCHAR(20),  
    adresa ROW(ulice CHAR(30),  
               č_domu CHAR(6),  
               město CHAR(20),  
               PSČ CHAR(5)),  
    datum_narození DATE);
```

```
INSERT INTO osoby  
VALUES('J.Pokorný', ('Svojetická', '2401/2', Praha 10,  
                    10000), 1948-04-23);
```

```
SELECT o.adresa.město  
FROM osoby o
```

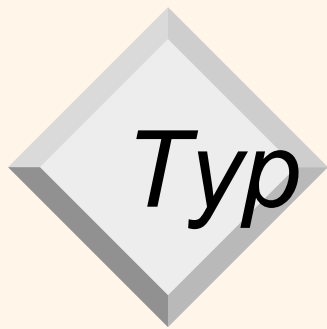
Typ řádku – pojmenovaný

- na rozdíl od ADT není zapouzdřený.

```
CREATE ROW TYPE účet_t (  
    č_účtu      INT,  
    klient      REF(zákazník_t),  
    typ CHAR(1),  
    otevřen     DATE,  
    úrok        DOUBLE PRECISION,  
    zůstatek    DOUBLE PRECISION,  
);
```

```
CREATE TABLE účty OF účet_t  
    (PRIMARY KEY č_účtu );
```

- příkaz není součástí standardu SQL. Lze ho nalézt např. v DB/2.



Typ řádku – pojmenovaný ADT

v podstatě
definice třídy

- datová struktura (+ metody)
- vhodné pro modelování entit a jejich chování

Př.: osoba, student, oddělení, ...

```
CREATE TYPE zaměstnanec_t AS(
  č_zam      INTEGER
  jméno      VARCHAR(20));
```

film	role	herec
Evita	sluha	(23, Kepka)
...

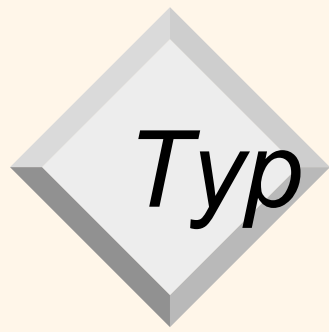
jako typ sloupce

Použití

id	č_zam	jméno
23712	23	Kepka
...

jako typ řádku

id připomíná
OID v OO



Typ řádku – pojmenovaný ADT

```
CREATE TABLE zaměstnanci OF zaměstnanec_t  
(PRIMARY KEY č_zam);
```

Co je vlastně potom výslednou tabulkou?

- unární relace, jejíž n-tice jsou objekty se dvěma komponentami.
- IO je funkcí tabulek a nikoliv typů



Uživatelsky definované procedury a funkce

programy vyvolatelné v SQL: *procedury a funkce*

- procedury mají parametry typu IN, OUT, INOUT
- funkce mají parametry jen typu IN, vracejí hodnotu

konstrukce programů:

- hlava i tělo v SQL (buď 1 SQL příkaz nebo BEGIN...END)
- hlava v SQL, tělo externě definované

v SQL/PSM

volání programů:

- procedura: CALL jméno_procedury(p1,p2,...,pn)
- funkce: funkcionálně f(x,y)
- *uložená procedura (stored procedure)*: CALL statement z
klientského program, který se volá pod řízením databázového
managera.

v UDT přibudou *metody*



Uživatelsky definované procedury a funkce

Př.: DB2 UDB/OSF White Box ADT

```
CREATE TYPE bod AS (  
    x DOUBLE,  
    y DOUBLE,  
);
```

```
CREATE FUNCTION distance(p1 BOD, p2 BOD) RETURNS INTEGER  
LANGUAGE SQL INLINE NOT VARIANT  
RETURN sqrt((p2..y-p1..y)*(p2..y-p1..y) + (p2..x-p1..x)*(p2..x-p1..x));
```

```
SELECT Z.jméno  
FROM zam Z, město M  
WHERE M.název = 'Ostrava'  
    AND distance(Z.bydliště, M.střed) < 25;
```



(Uživatelsky definované) metody

SQL:1999 přidává metody

Rozdíly metod a funkcí:

- metody jsou vždy svázány s typem, funkce nikoliv,
- daný datový typ je vždy typem prvního (nedeklarovaného) argumentu metody,
- metody jsou uloženy vždy ve stejném schématu, ve kterém je uložen typ, ke kterému mají nejbližší. Funkce nejsou omezeny na specifické schéma.
- funkce i metody mohou být polymorfické, liší se v mechanismu volby konkrétní metody v run time,
- signatura a tělo metody jsou specifikovány odděleně,
- volání metody (tečková notace + argumenty v závorkách).



ADT - plány v SQL3

```
CREATE TYPE zaměstnanec_t AS
```

```
(PUBLIC
```

```
    č_zam          INTEGER  
    jméno         VARCHAR(20),  
    adresa        adresa_t,  
    vedoucí       zaměstnanec_t,  
    datum_nástupu DATE,
```

```
PRIVATE
```


```
    základní_plat DECIMAL(7,2),  
    příspěvek     DECIMAL(7,2),
```

```
PUBLIC
```

```
    FUNCTION odpr_léta(p zaměstnanec_t) RETURNS INTEGER  
        <zdrojový kód pro výpočet počtu odpracovaných let> ,
```

```
PUBLIC
```

```
    FUNCTION mzda(p zaměstnanec_t) RETURNS DECIMAL  
        < zdrojový kód pro výpočet mzdy> );
```




ADT - skutečnost v SQL:1999

```
CREATE TYPE zaměstnanec_t AS(  
  č_zam          INTEGER  
  jméno          CHAR(20),  
  adresa         adresa_t,  
  vedoucí        zaměstnanec_t,  
  datum_nástupu  DATE,  
  základní_plat  DECIMAL(7,2),  
  příplatek      DECIMAL(7,2))  
INSTANTIABLE  
NOT FINAL  
REF č_zam  
METHOD odpr_léta() RETURNS INTEGER  
METHOD mzda() RETURNS DECIMAL);
```

```
CREATE METHOD odpr_léta  
FOR zaměstnanec_t  
BEGIN ... END;
```

```
CREATE METHOD mzda  
FOR zaměstnanec_t  
BEGIN ... END;
```

ADT - skutečnost v SQL:1999

NOT FINAL ... může mít další podtyp

v SQL:1999 strukturované typy musí být **NOT FINAL** a odlišující typy musí být **FINAL** (v SQL:2003 uvolněno)

REF umožňuje chápat data (řádky) v tabulkách daného typu jako objekty. V definici tabulky pak lze tento „identifikační“ atribut pojmenovat

ADT - skutečnost v SQL:1999

Možnosti specifikace:

- generované systémem

REF IS SYSTEM GENERATED

nebo

jeho hodnoty lze „vidět“

REF IS PID SYSTEM GENERATED

- generované uživatelem

REF USING <předdefinovaný typ>

- odvozené

REF(<seznam atributů>)

Zde: odkaz pomocí č_zam

Podtypy

```
CREATE TYPE osoba_t AS(
    jméno          CHAR(20),
    adresa         adresa_t,
NOT FINAL
CREATE TYPE zaměstnanec_t UNDER osoba_t(
    č_zam          INTEGER
    vedoucí        zaměstnanec_t,          /*zaměstnanec_t je
    datum_nástupu  DATE,                  podtypem osoba_t */
    základní_plat  DECIMAL(7,2),
    příplatek      DECIMAL(7,2))
NOT FINAL
REF č_zam
METHOD odpr_léta() RETURNS INTEGER
METHOD mzda() RETURNS DECIMAL);
```

Podtypy

podtypy

```
CREATE TYPE úředník_t UNDER zaměstnanec_t
```

...

```
CREATE TYPE dělník_t UNDER zaměstnanec_t
```

...

- strukturované typy mohou být podtypem dalšího UDT
- UDT dědí strukturu (atributy) a chování (metody) ze svých nadtypů
 - povolena je jednoduchá dědičnost (vícenásobná odložena, nevyskytuje se ani v SQL:2003)
- *substituovatelnost*: na místě daného typu může být hodnota podtypu

Podtabulky

*zaměstnanec_t
musí být podtypem
osoby_t*

- aparát závislý na aparátu typů

```
CREATE TABLE osoby OF osoby_t
```

```
CREATE TABLE zaměstnanci OF  
zaměstnanec_t
```

```
UNDER osoby;
```

- dědí sloupce, IO, trigger, ... dané nadtabulkou

Podtabulky

*zaměstnanec_t
musí být podtypem
osoby_t*

- Požadavky konsistence pro podtabulky a nadtabulky
 - každá n-tice v nadtabulce (např. **osoby**) může korespondovat nejvýše k jedné n-tici v podtabulkách (např. **zaměstnanci** a **OON**)
 - tj. každá entita musí být nespecifičtější typ
- Výběr omezený na tabulku X pomocí **FROM ONLY (X)**
 - jinak i z podtabulek X.

Přístup k hodnotám atributů

Každý atribut má automaticky metody *generátor* a *mutátor*

- výběr hodnot

```
SELECT z.jméno()  
FROM zaměstnanci z
```

aplikace metody
generátor

- aktualizace ve 3 krocích

```
SET novýZam = zaměstnanec_t()  
novýZam.č_zam('7897890')  
novýZam.jméno('Jarda')  
INSERT INTO zaměstnanci(novýZam)
```

generuje novou
instanci

Reference a dereference

```
CREATE TYPE účet_t AS (  
  č_úctu    INT,  
  klient    REF(zákazník_t), reference  
  typ       CHAR(1),  
  otevřen   DATE,  
  úrok      DOUBLE PRECISION,  
  zůstatek  DOUBLE PRECISION,  
)
```

FINAL REF IS SYSTEM GENERATED;

```
CREATE TABLE účty OF účet_t  
  (PRIMARY KEY č_úctu );
```

tabulka účty má zvláštní
atribut podobný oid

tzv. **self-referencing**
column

tabulka

Reference a dereference

Co se děje, je-li odstraněn odkazovaný objekt:

- Nic – implicitně REFERENCES ARE NOT CHECKED
- Možnost akce, je-li REFERENCES ARE CHECKED ON DELETE (pak SET DEFAULT, SET NULL, CASCADE, NO ACTION, RESTRICT)

Dereference

- lze dělat pouze tehdy, je-li definováno umístění objektů typu REF (v SQL:1999 je to jedna tabulka)

```
CREATE TABLE zákazníci OF zákazník_t;  
CREATE TABLE účty OF účet_t  
  (PRIMARY KEY č_účtu,  
  klient WITH OPTIONS SCOPE zákazníci  
  );
```

Pz.: připomíná referenční integritu

```
SELECT u.klient -> jméno  
FROM účty u  
WHERE u.klient->adresa.město = "Suchdol" AND u.zůstatek > 100000;
```

alokace

*dereference,
cesta*



Reference a dereference

- dereference cestou a/nebo funkcí Deref

srovnaj

```
SELECT u.otevřen, u.klient  
FROM účty u;
```

a

```
SELECT u.otevřen, Deref(u.klient)  
FROM účty u;
```

**Deref vrací
n-tici**



Reference a dereference

- **výhody používání REF:**
 - sdílení objektů
 - ◆ nejsou zbytečně kopírována data
 - ◆ změna se provádí na jednom místě

- **odkaz na metodu:**

SELECT u.klient() -> jméno

FROM účty u

WHERE u.klient() -> mzda() > 10000;

Pz.: metody bez parametrů nevyžadují ()

Za SQL:1999, 2003

```
CREATE TABLE zaměstnanci  
(id INTEGER PRIMARY KEY,  
jméno VARCHAR(30),  
adresa ROW( uliceCHAR(30),  
číslo_d CHAR(6),  
město CHAR(20),  
PSČ CHAR(5)),  
projekty INTEGER SET,  
děti osoba,  
odměny MONEY MULTISSET
```

kolekce

je v SQL:2003

Multimnožiny

nt1 MULTISET EXCEPT [DISTINCT] *nt2*

nt1 MULTISET INTERSECT [DISTINCT] *nt2*

nt1 MULTISET UNION [DISTINCT] *nt2*

CARDINALITY(*nt*)

nt IS [NOT] EMPTY

nt IS [NOT] A SET

SET(*nt*)

nt1 = *nt2*

nt1 IN (*nt2*, *nt3*, ...)

nt1 [NOT] SUBMULTISET OF *nt2*

r [NOT] MEMBER OF *nt*

CAST(COLLECT(*col*))

POWERMULTISET(*nt*)

POWERMULTISET_BY_CARDINALITY(*nt*,*c*) množina všech neprázdných

nt1 - *nt2*

$nt1 \cap nt2$

$nt1 \cup nt2$

$|nt|$

odstraň duplicity z *nt*

rovnost multimnožin

být v seznamu multimnožin

porovnání multimnožin

$r \in nt?$

hnížděná tabulka založená na *col*

množina všech neprázdných pomnožin *nt*

pomnožin *nt* s kardinalitou *c*

Pz.: SQL:2003: MULTISET bez omezení kardinality

ARRAY bez uvedení maximální kardinality – je implementačně daná



O-R v komerčních produktech

- INFORMIX: kolekce set, multiset, list (bez omezení délky)
- Oracle od verze 8i (r. 1999):
 - místo ADT -- *typ objektů*
 - notace: **CREATE TYPE ... AS OBJECT(...);**
 - kolekce
 - ◆ **VARRAY** (ekvivalentní **ARRAY** z SQL:1999), avšak neumožňuje DELETE pro daný prvek pole
 - ◆ **NESTED TABLE** (neuspořádaná neohraničená kolekce prvků)
Pz.: hníždění do 1 úrovně, obecněji v Oracle 9i

CREATE TYPE Kde_všude AS VARRAY(4) OF Adresa



O-R v komerčních produktech

– viditelnost id

```
SELECT REF(o) INTO reftoosoba  
FROM osoby AS o  
WHERE o.jméno = 'Novák, J.'
```

O-R v komerčních produktech

```
CREATE TYPE Auta AS TABLE OF Auto_t
```

```
CREATE TABLE FIRMY (
```

```
    vozový_park Auta
```

```
    ...)
```

```
    NESTED TABLE vozový_park STORE AS vozy;
```

Pz.: lze zadat, kde mají být „podtabulky“ Auta uloženy

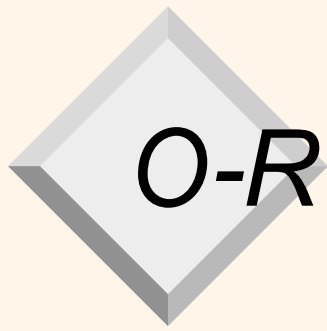
```
SELECT *
```

```
FROM FIRMY AS f, f.vozový_park AS vp
```

```
WHERE 'Buick' IN (SELECT vp.značka FROM vp);
```



sleduj polohu
středníku



O-R v komerčních produktech

Dotazy na hnížděnou tabulku:

- pomocí **THE**
- lze s ní zacházet jako s jinými relacemi

```
SELECT vp.SPZ  
FROM THE (  
    SELECT vozový_park FROM FIRMY  
    WHERE jméno_f= 'Komix')  
) vp  
WHERE vp.ZNAČKA='Buick';
```

D.: Najdi SPZ všech Buicků firmy Komix.



O-R v komerčních produktech

Metody v ORACLE

- Specifikace v **CREATE TYPE** pomocí **MEMBER FUNCTION**, **MEMBER PROCEDURE**
- Tělo v příkazu **CREATE TYPE BODY**
- Přístup

```
SELECT u.klient.jméno  
FROM účty u  
WHERE u.klient.mzda > 10000;
```



Problémy s OO v SQL

- tabulky jsou jediné pojmenované entity
- typ REF aplikovatelný pouze na objekty dané řádkem
- UDT je prvním krokem k OO
 - umožnit perzistenci, musí být objekt v tabulce
 - individuální instanci nelze přiřadit jméno
 - nelze použít dotazy na všechny instance ADT

Návrh OR DB: Transformace E-R → OR

- 1. Fáze: typy

- typy entit → strukturované typy
- složené atributy → pojmenované typy řádků, nepojmenované typy řádků v strukturovaném typu, také strukturovaný typ je možný
- vícehodnotové atributy → pole typovaných hodnot (odhad max je někde důležitý)
- odvozené atributy → přidej metodu do definice strukturovaného typu

odstraněno v SQL:2003
pomocí SET i ARRAY

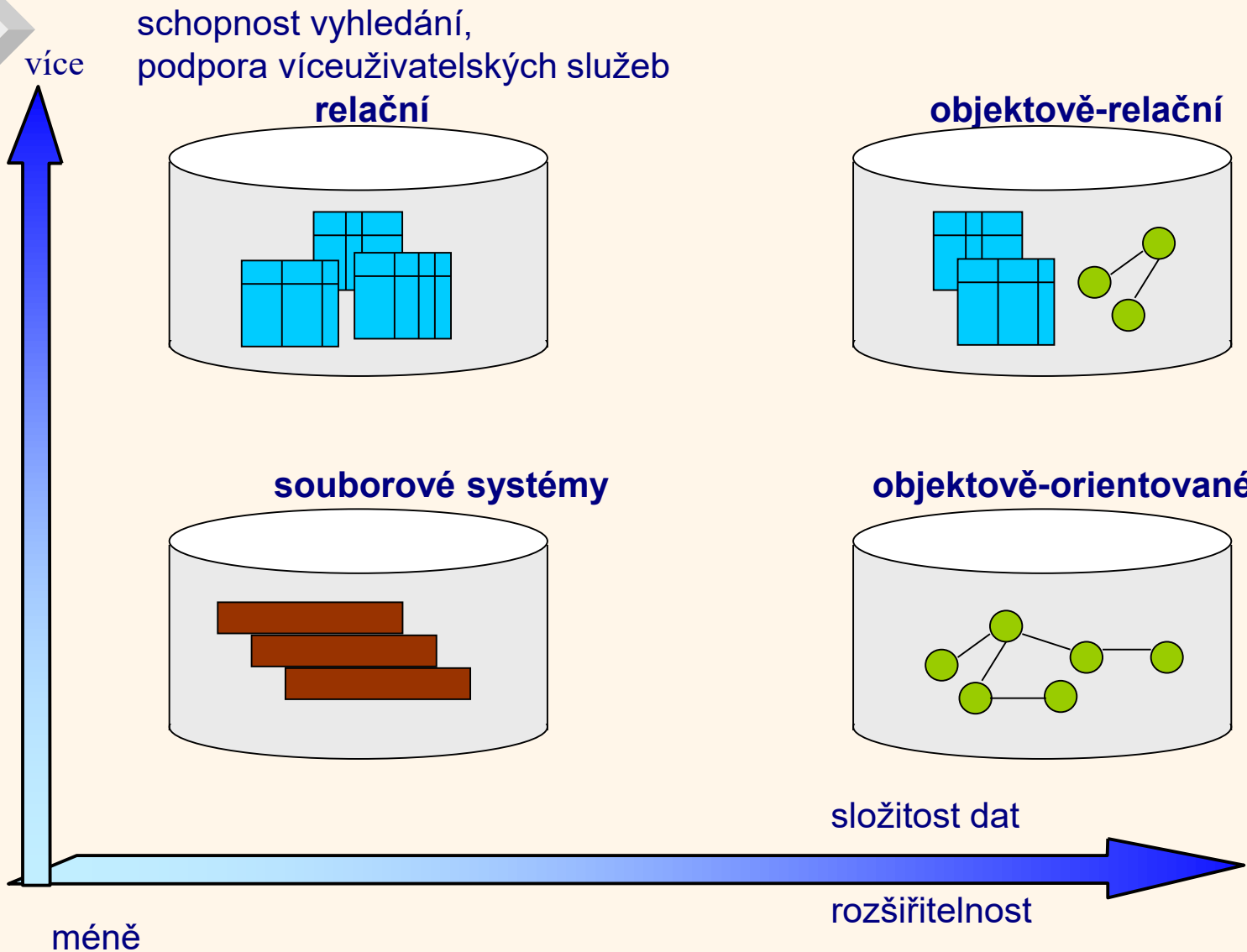
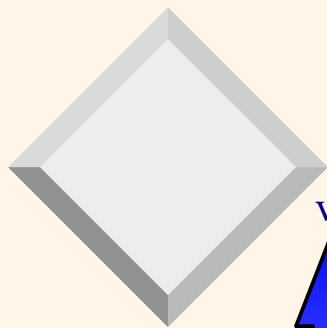


Návrh OR DB: Transformace E-R → OR

- typy vztahů – dvousměrně nebo jednosměrně
 - N:1 → pomocí REF + pole typovaných hodnot (jestliže dvousměrně)
 - M:N → pomocí jednoho nebo dvou polí obsahujících typované hodnoty (jestliže jedno nebo dvousměrně).
- ISA hierarchie → hierarchie typů
- 2. fáze: typované tabulky

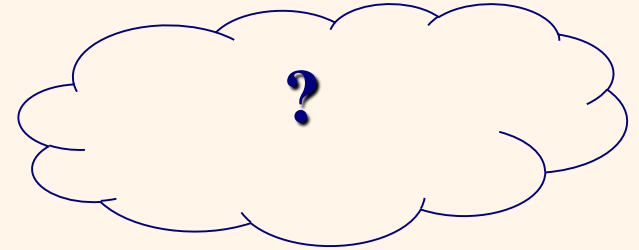
Závěr

- současné implementace:
 - výtky OOSŘBD - málo databázové
 - výtky ORSŘBD - málo objektové
- chybí vývojové prostředky, nové metodologie
- největší problém a současně výhoda: univerzálnost
- vývoj jde dál: XML DB, NoSQL DB, webové, cloudové, NewSQL, ..., obecněji: nerelační, distribuované, open-source a horizontálně škálované



Svět databázových technologií

Závěr



Technologie	70. léta	80. léta	90. léta	2000-10	○> 2010
výzkum	relační	OO,OR	XML	NoSQL	○ ○
komerce	hierarchické síťové	relační	OO,OR	XML	NoSQL
zdeděné		hierarchické síťové	relační	OO,OR	XML

Poučení z historie