Query languages 1 (NDBI001) part 2

J. Pokorný
MFF UK

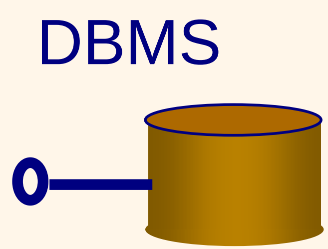
Content

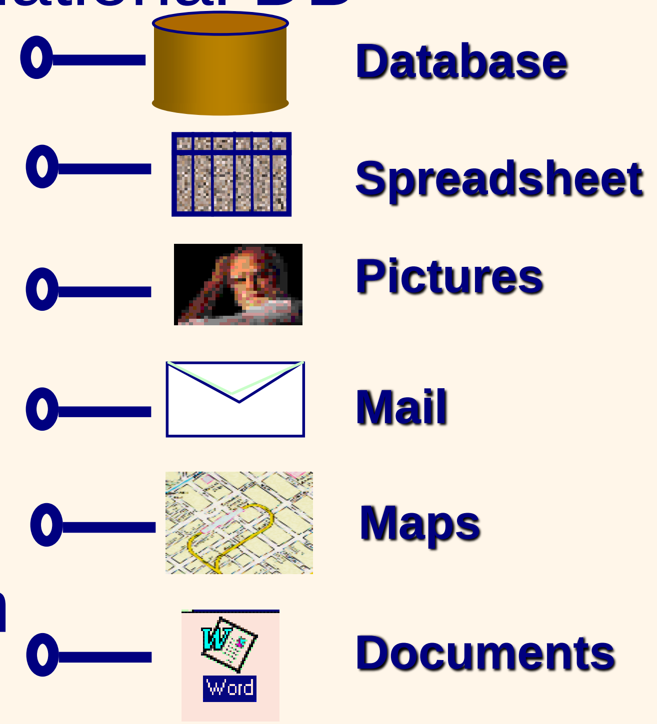
Why more DB technologies

New Application Requirements:

- new types objects and functions
- OO analysis and design vs. relational DB

”Relational database reminds a garage, that forces you to disassemble your car and save parts to the drawers..."





Goal: integration and data management in one system

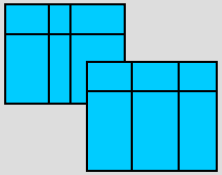
Object-oriented databases

object data model

- in accordance with the natural world view (entity ⇒ object)
- definition of complex objects and their manipulation

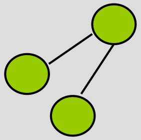
RDBMS

powerful OLTP
data availabiity
confidentiality of data
tools for data management
standard language interface
memory management
parallel data processing
integrity



operations on complex objects
recursive structures
abstract data types
interface to OO language
complex transactions

OODBMS



Functionality of relational and OO DBMSs

Object-oriented databases

- 1993: consortium ODMG (Object Data Management Group) of leading OODBMS vendors ⇒ design of the ODMG-93 standard.
  - superset of the more general Common Object Model (COM) created by the Object Management Group (OMG). Its definition language IDL was adopted.
  - a query part Object Query Language (OQL), which is related to the concept of the query part of the SQL92 standard.
  - interface k OO PL C++, Smalltalk (to Java: replaced by Java Data Objects (JDO))
- 2001: the ODMG disbanded (version ODMG 3.0)

OOPL + DBMS = OODBMS

Basic concepts of ODMG-93

- class (or type), instance (or object), attribute, method and integrity constraint
  - class - template for instances (objects), which can share attributes and methods.
    - attribute domain: primitive data type, abstract data type (ADT), or reference to a class.
    - method is a function (its implementation is hidden) applicable to class instances (calculation is based on the values of attributes).
- object identifier (OID)
  - each object has a unique identifier, through which a corresponding object can be obtained from DB.

Basic concepts of ODMG-93

- encapsulation
  - data is “captured” with methods. An encapsulation unit is an object. Methods hold only on objects, with which they are encapsulated.
- class hierarchy, inheritance
  - subclass $\Rightarrow$ hierarchy
  - inheritance is a process that means for a subclass to use all attributes and methods from its superclass.
  - multiple-inheritance ($\Rightarrow$ problems, e.g., conflict resolution the same names inherited attributes and methods).

Emergence of OO DB technology

Sources: OO programming, OO analysis and design, relational DBMS

- object-relational mapping (ORM)

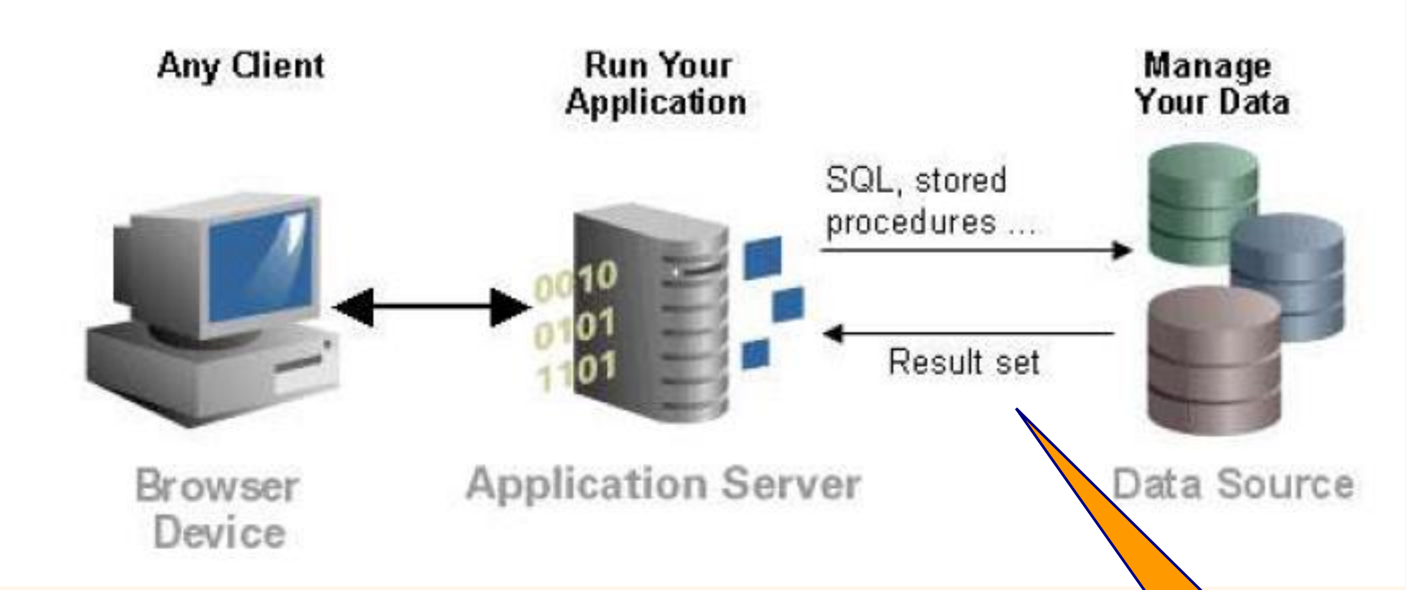
  Ex.: Hibernate (interface: Session, Transaction, Query)

  TopLink (ORM application owned by Oracle, Inc.)

  Access to objects: e.g. Hibernate Query Language $\rightarrow$ SQL; programming language + methods for entering the SQL database

  Problem: less semantics in relations, impedance mismatch in an access to objects.

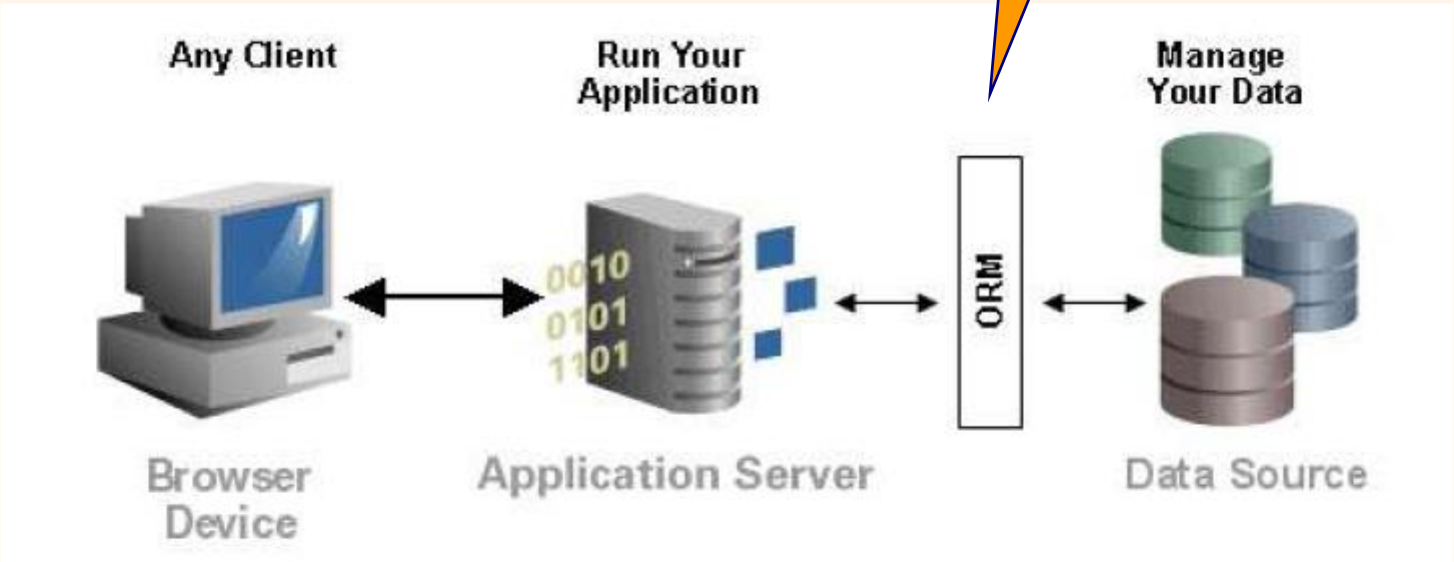
Running without ORM



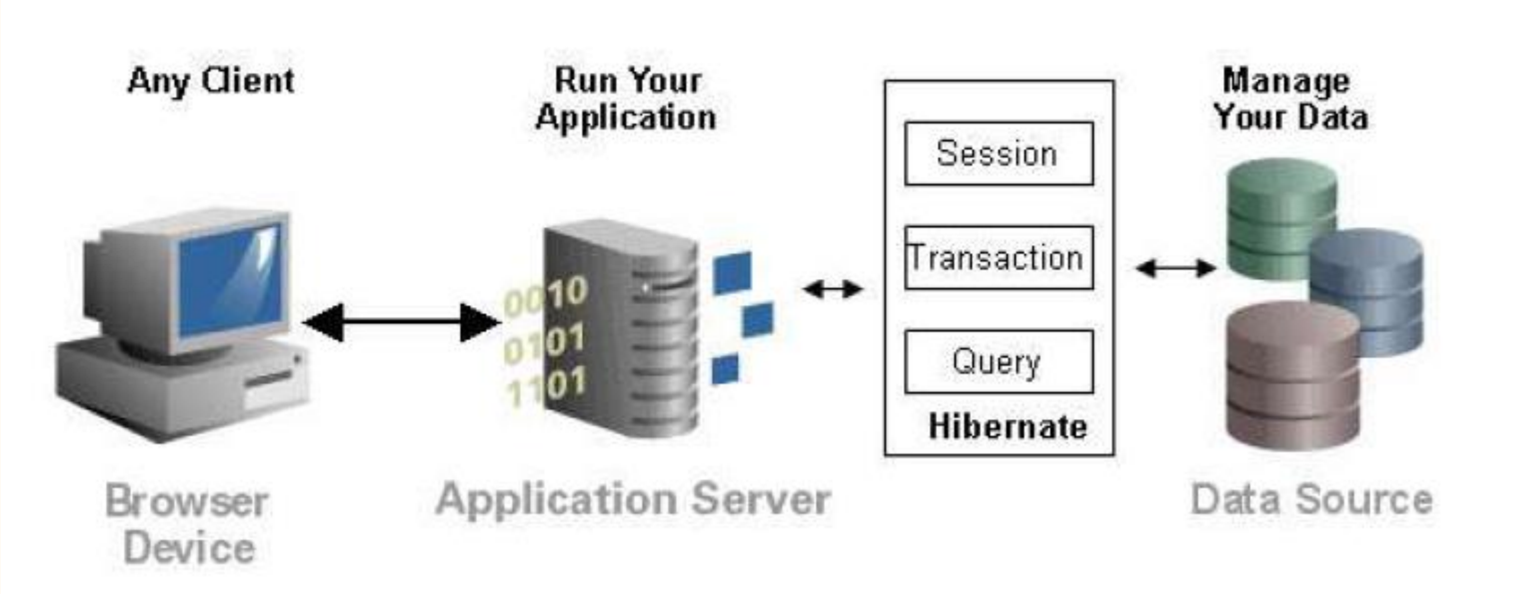
through API JDBC

Running with ORM

15%-20% slower than with JDBC



Example: Hibernate



Emergence of new technologies

- 2nd half of the 1980s -- OO databases
  - O2 (→ Unidata → Informix → IBM)
  - ObjectStore (Ignite Technologies from 2015)
  - Versant Object Database (version 9.2 in 2016),
  - Objectivity/DB, InterSystems Caché (version 2016.1.1),
  - GemStone (GemTalk Systems from 2013),
  - Jasmine
  - newer: db4o (database for objects),
- Principle: top-down (from application to data)
- Querying: OQL – not complete, others: e.g., Objectivity/SQL++


Relational Model
R.I.P
1970-1999
www.wiscorp.com
Copyright 1999, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved
1

OR DB Technologies

Reasons:

- partial failure: OO technologies did not offer the flexibility and performance of relational DBMS
- even later descent - e.g. db4o ceased to be supported in 2014
- DBMS manufacturers target:
  - get the most out of large investments in relational technology (data, experience gained)
  - take advantage of the flexibility, productivity and running benefits of OO modeling,
  - integrate database services into production systems and other applications.

OR DB Technologies

- 90ies - OR databases (OR DBMS)
  - combination of OO and relational DBMS
  - 1992: UniSQL/X, then: HP - OpenODB (later Odapter)
  - 1993: Montage Systems (later Illustra) –
  - the commercial version of Postgres
  - 2000+: DB/2, INFORMIX, ORACLE, Sybase Anywhere (integrated to SAP, 2012), Unidata, Microsoft SQL Sever

OR DB Technologies

Two approaches:

universal memory, all kinds of data is

managed in DBMS; integration (in many different ways!)

⇒ universal servers

- universal approach, all data is in their original (autonomous) systems

  Technique: middleware

  - gateways (at least two independent servers)
  - schemes mapping, query transformations
  - object envelops: Persistence Software, Ontologic, HP, Next, ... (problems: efficiency)
  - DB based on Web

Extensibility, user defined types and functions

ADT possibilities :
black box
white box

Requirement: manipulation of BLOBs (atomic in RDBMS)

Extensibility: possibility to add new data types + programs (functions) „wrapped“ into and special module

$\Rightarrow$ UDT (user defined types)

UDF (user defined functions)

Problem: integration into relational DBMS (including SQL!)

DB/2: relational extenders

Informix: DataBlades

ORACLE: cartridges

Sybase: Component Integration Layer.

universal servers

Extensibility, user defined types and functions

Ex.: DB/2 in r. 2006:

- MapInfo
- NetOwl (business intelligence language)
- EcoWin (time sequences, macroeconomical time sequences, …)
- GIS and spatial objects
- SQL expander (mathematical, financial, conversion and other functions)
- VideoCharger (audio and video objects in real-time)
- text, XML, audio, video, pictures
- FormidaFire (heterogeneous data integration)
- …

Ex.: Informix in r. 2006:

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

Extensibility, user defined types and functions

- Implementation: technology „plug in“ using various techniques:
  - DataBlades - direct access to the database engine
  - ORACLE 7.3 - more servers and API
- Today: direct part of database engines DB/2, INFORMIX, ORACLE, Sybase Adaptive Server+Java, OSMOS, Unidata

Extensibility, user defined types and functions

- partial standardization:
  - SQL/MM (e.g., Full-Text - provides ADT + appropriate functions)
  - more generally: SQL99, SQL:2003 enable to build rather complex data types based on several built-in basic data types
  - extending apparatus of built-in basic types – XML (2003 …), JSON (2016)

Example - text extender

```
SELECT journal, date, title
FROM Articles
WHERE CONTAINS(article_text, '("database" AND
                    ("SQL" | "SQL92") AND NOT "dBASE")') = 1;
```

Other functions: NO_OF_MATCHES ((how many times a sample appears in the text), RANK (ranking of values of the order in the result based on a measure).

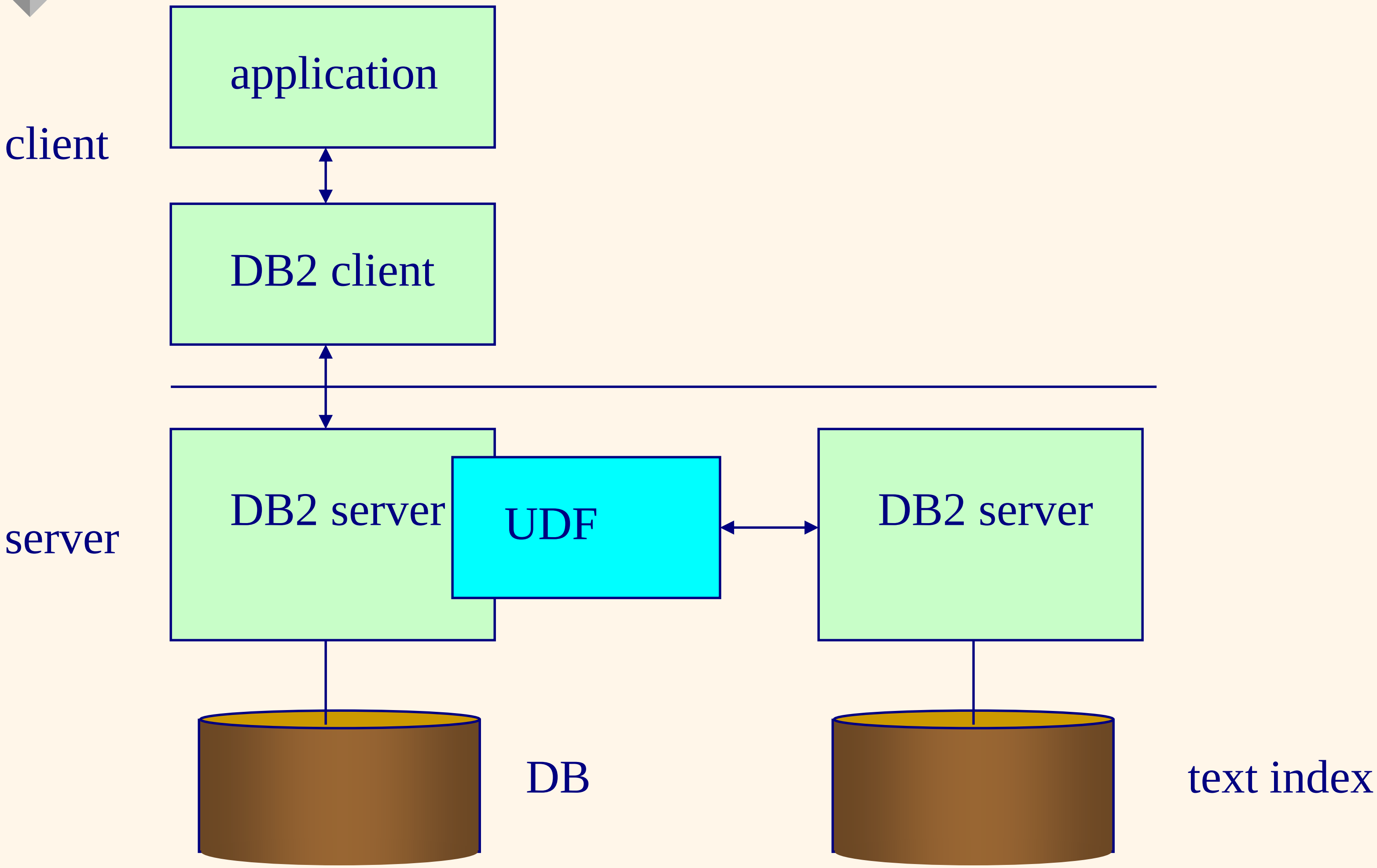
```
SELECT journal, title
FROM Articles
WHERE NO_OF_MATCHES (article_text, 'database') > 10;
```

```
SELECT journal, date, RANK(article_text, '("database" AND
("SQL" | "SQL92") )') AS relevance
FROM Article
ORDER BY relevance DESC;
```

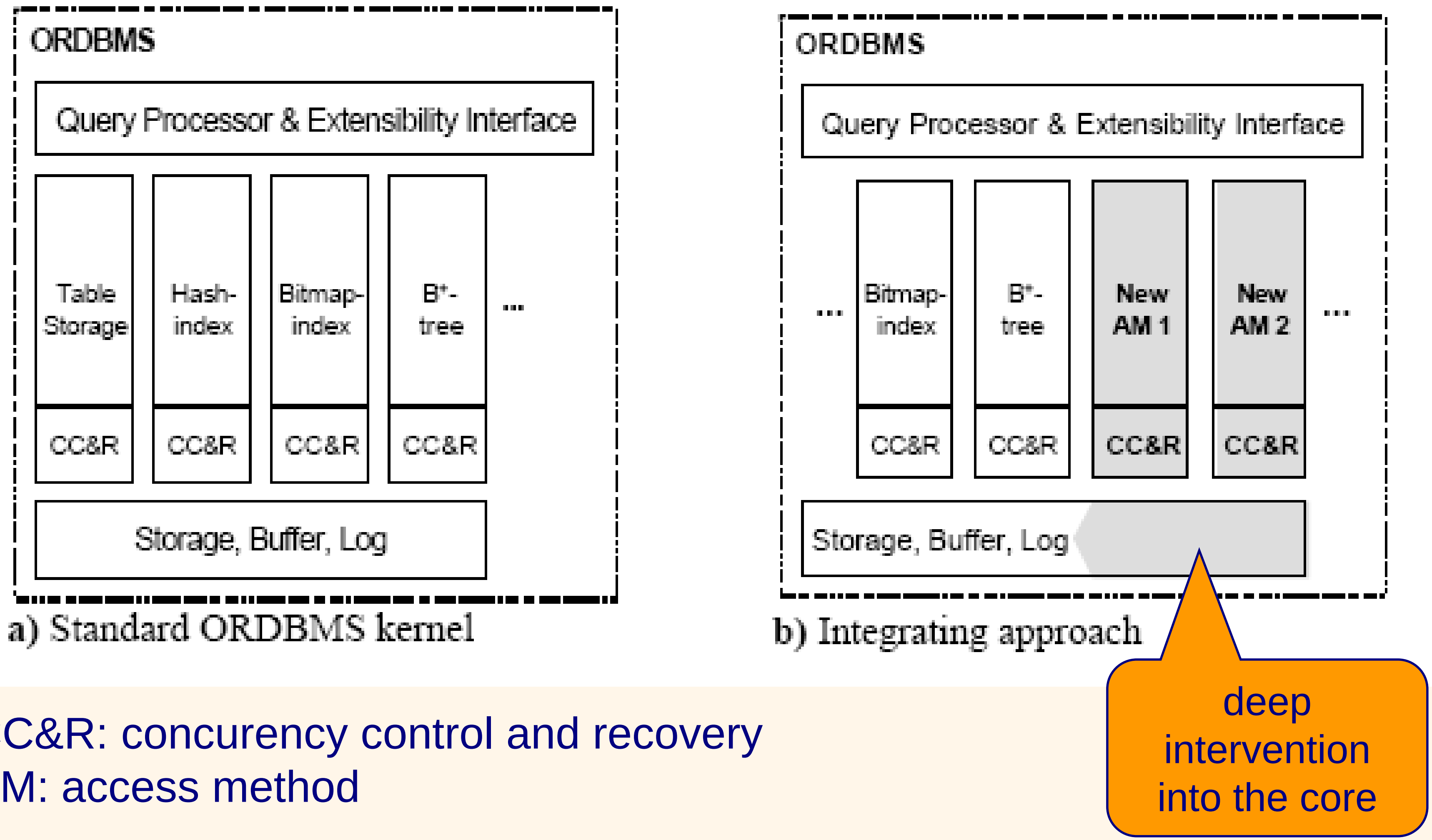
Architecture of known products

- adding a special application interface (API) and special servers (also ORACLE 7.3 – see ,e.g., CONTEXT, Media Server, OLAP),
- simulation OR on the middleware level (also ORACLE 7.3 – see, e.g., the part Spatial Data Option),
- total remaking a database machine (e.g., Illustra Information Technology),
- adding an OO layer to a relational machine (e.g. INFORMIX Universal Server, IBM D2/6000 Common Server, Sybase Adaptive Server + Java).

Interaction of DB and the text extender in DB2



Architectures of extendibility (1)



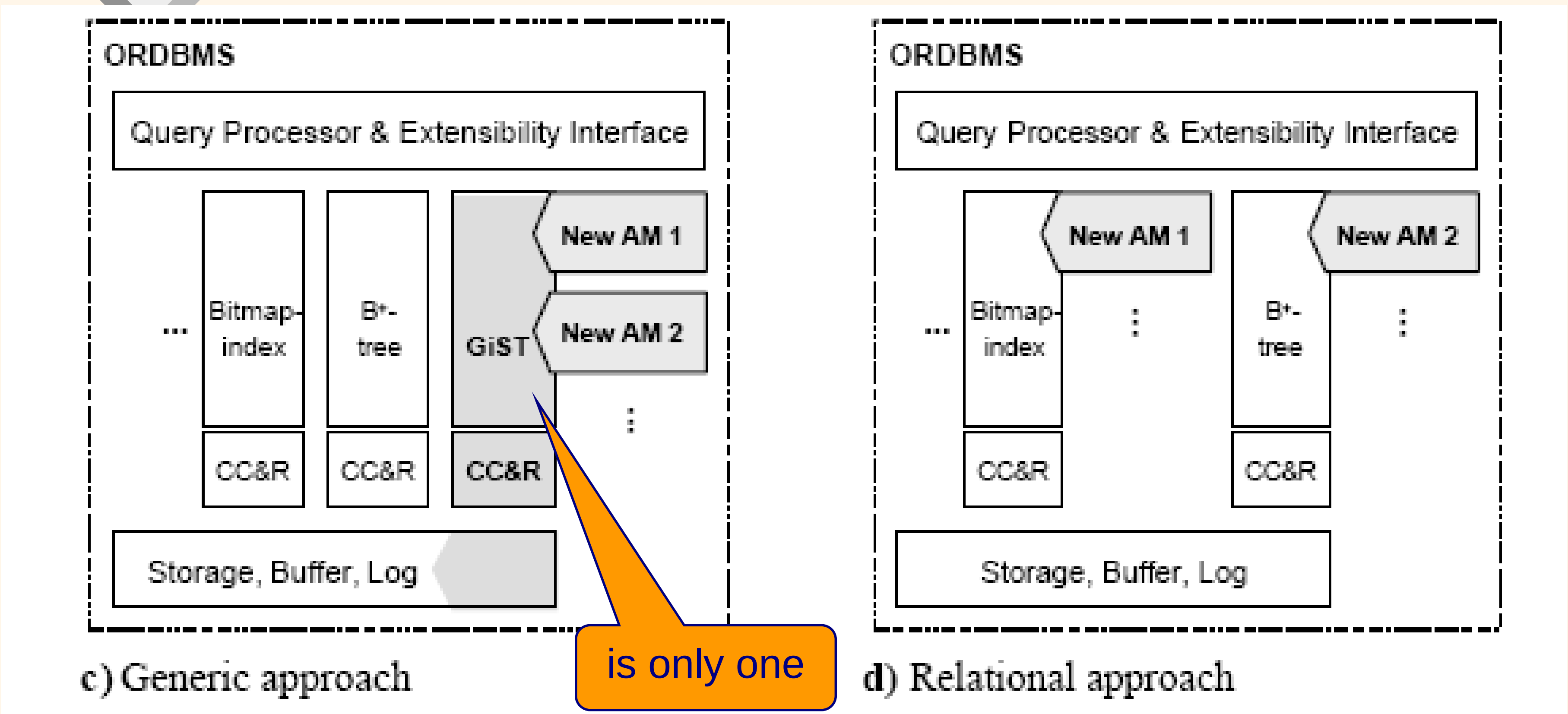
a) Standard ORDBMS kernel

b) Integrating approach

CC&R: concurency control and recovery

AM: access method

Architectures of extendibility (2)



GiST: Generalized Search Tree

AM on the top of relational DBMS

Problems with ORDBMS

- implementation of VITA data types (video, image, text, audio)
- integration of data types
  - how to optimize queries?
- old solutions:
  - modification of the DBMS core (expensive, demanding)
  - functionality available only for some types of requirements
- newer solutions:
  - specialized servers

„Real“ ORDBMS

Won Kim: „extendibility is only a secondary, although useful, feature of the OO approach“

Stonebraker: „extendibility of type “plug-in” (e.g., in ORACLE) is suitable for connectivity application-application, nevertheless has nothing to do with database “plug-in”. It is only a middleware, which does not establish OR technology“.

Requirements:

- data model with the features of ODMG-93
- associated high-level object language

Solution: OO extension of SQL fulfills (approx.) these requirements

Today: standards SQL:1999, SQL:2003 + further development

Object-relational modelling

- Extension of the relational model by type definition, objects and relevant constructs for their manipulation,
- attributes of tuples are of complex types, including nested relations,
- relational background is preserved (including a declarative approach to data processing,
- compatibility with relational languages (they form a subset).

Example: nested vs. normalized relation

journal	title	authors	keywords	date		
				day	month	year
CW	OLAP	{Kusý, Klas}	{star, dimension}	23	April	1998
SN	Database	{Novák, Fic}	{RDM, schema}	15	May	1998

journal	title	authors	keyword	day	month	year
CW	OLAP	Kusý	star	23	April	1998
CW	OLAP	Kusý	dimension	23	April	1998
CW	OLAP	Klas	star	23	April	1998
CW	OLAP	Klas	dimension	23	April	1998
SN	Database	Novák	RDM	15	May	1998
SN	Database	Novák	schema	15	May	1998
SN	Database	Fic	RDM	15	May	1998
SN	Database	Fic	schema	15	May	1998

Normalization into 4NF

journal	title
CW	OLAP
SN	Database

title	author
OLAP	Kusý
OLAP	Klas
Database	Novák
Database	Fic

title	keyword
OLAP	star
OLAP	dimension
Database	schema
Database	RDM

title	day	month	year
OLAP	23	April	1998
Database	15	May	1998

Negatives of 4NF

- joins in queries

Negatives of only 1NF

- the loss of relation 1 row = 1 object

Normalization into 4NF

journal	title
CW	OLAP
SN	Database

eventually *

title	day	month	year
OLAP	23	April	1998
Database	15	May	1998

title	author
OLAP	Kusý
OLAP	Klas
Database	Novák
Database	Fic

title	keyword
OLAP	star
OLAP	dimension
Database	schema
Database	RDM

Negatives of 4NF:

- joins in queries

Negatives of simple 1NF:

- the loss of relation 1 row = 1 object

SQL:1999

5 parts:

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

* alternative to SQL calling from application programs (implementations: ODBC, JDBC)

** procedural language for transactions managements

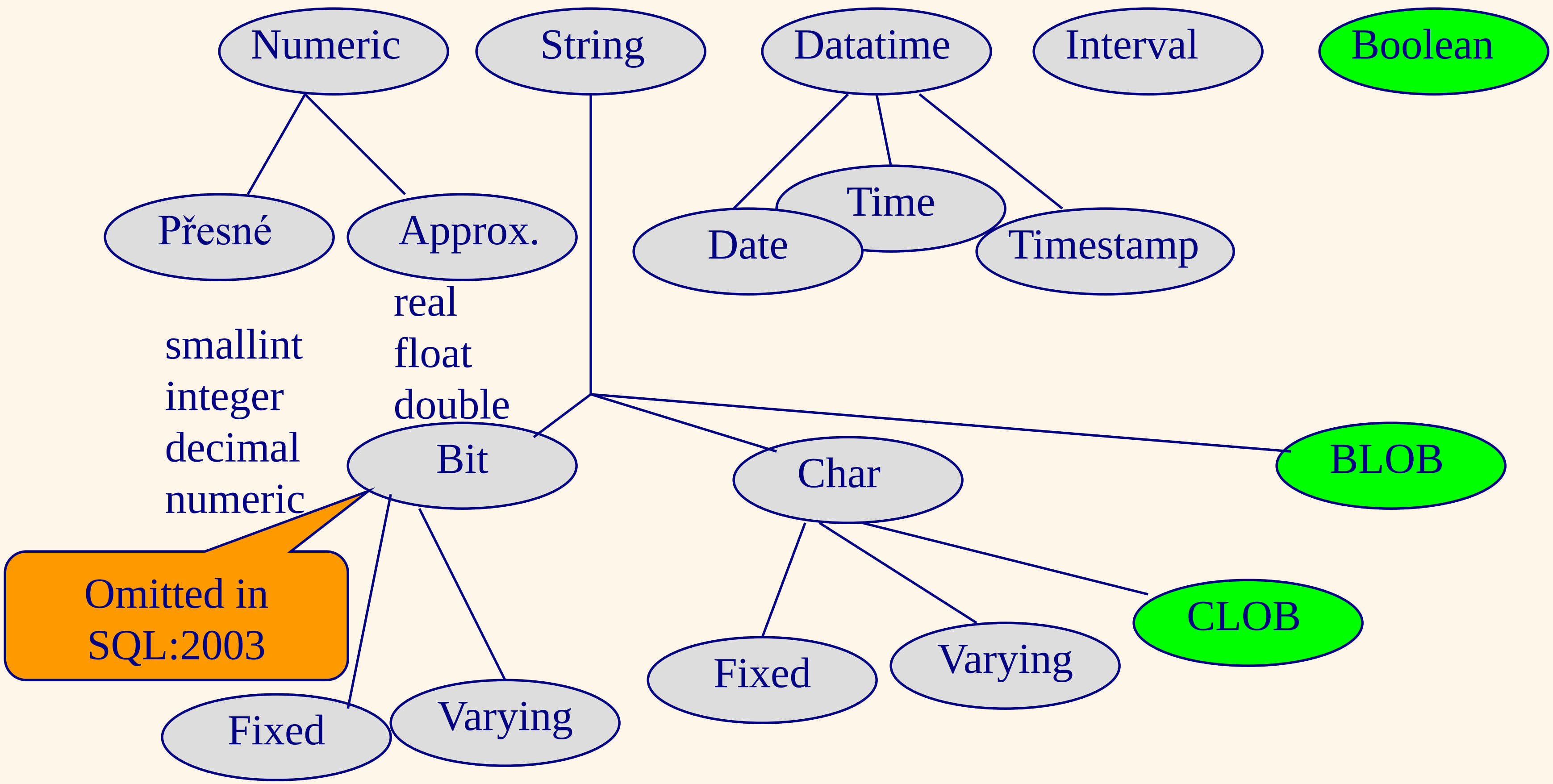
SQL:1999

- object support
- stored procedures
- triggers
- recursive queries
- extension for OLAP
- procedural constructs
- expressions in ORDER BY
- save points
- updates and inserts using join operations

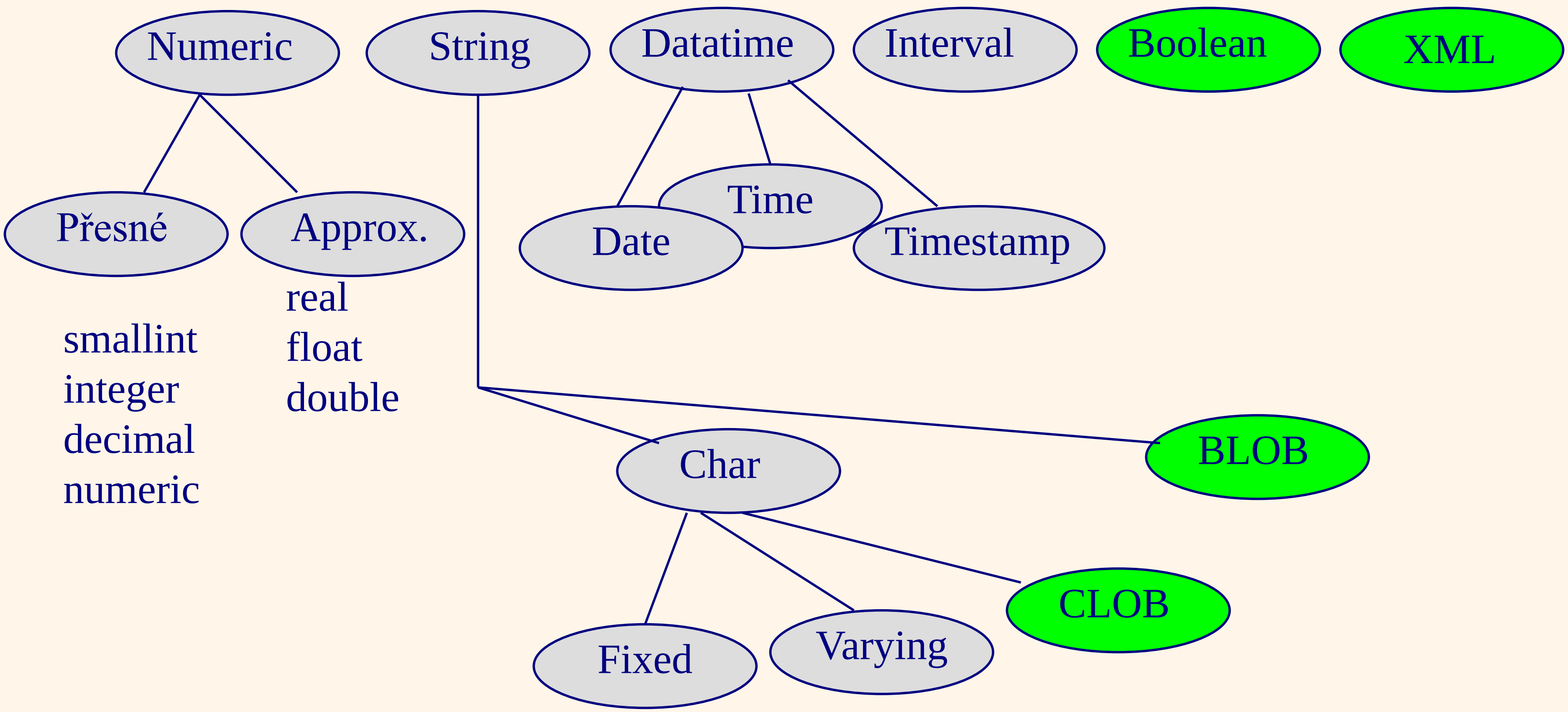
Objects: from SQL3 to SQL:1999

- The original SQL3 used for object support:
  - user defined types (UDT): ADT, named row types and differentiate types,
  - type constructors for row types and references,
  - type constructors for collections (sets, list and multisets),
  - user defined functions (UDF) and procedures (UDP),
  - large objects (LOB).
- Standard SQL:1999 - subset of original concept

Predefined types in SQL:1999



Predefined types in SQL:2003



Type Boolean

```
SELECT dep_n, EVERY(salary > 20000) AS all_rich,
   SOME(salary > 20000) AS some_rich
FROM emp
GROUP BY dep_n;
```

Result:

dep_n	all_rich	some_rich
A35	FALSE	FALSE
J48	TRUE	TRUE
Z52	FALSE	TRUE

Other types in SQL:1999

constructed atomic types:

- reference

constructed composite types:

- array /* subtype collection */
  ordered list with the maximum cardinality;
  arrays of arrays or multidimensional arrays are not allowed
- row

Note: originally considered more subtype collections (in implementation as well)

Note: there are new functions to types (BIT_LENGHT, POSITION, SUBSTRING, …)

Array type

```
CREATE TABLE messages(
    ID          INTEGER
    authors     VARCHAR(15) ARRAY[20]
    title       VARCHAR(100)
    abstract    FULLTEXT
```

- accessing elements of array by subscript numbers, e.g. authors[3],
- function CARDINALITY, comparison =, <>, concatenation ||, CAST
- UNNEST,
- possibility WITH ORDINALITY (to generate the offset column corresponding to subscript numbers of the elements in array)

```
SELECT m.ID, a. name
FROM messages AS m, UNNEST(m.authors) as a(name)
```

Other types in SQL:1999

UDT:

- distinct types (are formed by a single predefined type)
- structured types (can be defined with more attributes, which are of predefined types, type ARRAY, or of another structured type)
  - ADT
    - behavior is specified by functions, procedures and methods
    - ADTs can be organized into hierarchies with inheritance
  - named row types

Distinct types

Principle: renaming (distinguishing) predefined types + different behavior

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

CREATE TYPE METERS
AS INTEGER FINAL;

CREATE TYPE Q_METERS
AS INTEGER FINAL;

CREATE TABLE rooms(
m_id            ROOM_TYPE
m_lenght        METERS
m_width         METERS
m_perimeter     METERS
m_area          Q_METERS);
```

```
UPDATE rooms
SET m_area = m_lenght
```

reports an error

```
UPDATE rooms
SET m_width = m_lenght
```

OK

Attention: compare with the DOMAIN notion!

Note: weak semantics: addition operator is not defined on METERS (Q_METERS)

Row type - unnamed

```
CREATE TABLE persons (
        name VARCHAR(20),
        address ROW(street        CHAR(30),
                    house_n       CHAR(6),
                    town          CHAR(20),
                    zip_code      CHAR(5)),
        birthdate DATE);

INSERT INTO persons
VALUES('J. Novák', ('Svojetická’, '2401/2', Praha 10,
   10000), 1948-04-23);

SELECT p.address.town
FROM persons p
```

Row type – named

- Comparing to ADT, it is not encapsulated.

```
CREATE ROW TYPE account_t (
    account_n   INT,
    client      REF(client_t),
    type        CHAR(1),
    opened      DATE,
    interest    DOUBLE PRECISION,
    balance     DOUBLE PRECISION,
    );
CREATE TABLE accounts OF account_t
  (PRIMARY KEY account_n );
```

- statement is not a part of the SQL standard. There is, e.g., in DB/2.

Row type – named ADT

really a class definition

- data structure (+ methods)
- suitable for entity modelling and their behavior

  Ex.: person, student, departure, …

```
CREATE TYPE employee_t AS(
empID          INTEGER
name           VARCHAR(20));
```

movie	role	actor
Evita	servant	(23, Kepka)
…	…	...

Use

as a type column

as a row type

id	empID	name
23712	23	Kepka
…	…	...

id reminds OID in OO

Row type – named ADT

```
CREATE TABLE employees OF employee_t
  (PRIMARY KEY empID);
```

What is actually the resulting table??

- Unary relation, whose tuples are objects with two components.
- ICs are functions of tables and not of types

User defined procedures and functions

- programs called in SQL: procedures and functions
  - procedures have parameters of type IN, OUT, INOUT
  - functions have parameters only of type IN, they return a value
- programs construction:
  - head and body in SQL (either 1 SQL statement or BEGIN…END)
  - head in SQL, body externally defined

in SQL/PSM

- programs calling:
  - procedure: CALL name_of_procedure(p1,p2,…,pn)
  - function: functionally as f(x,y)
  - stored procedure: CALL statement from a client program, which is called under direction of a database manager.

In UDT methods will be added.

User defined procedures and functions

Ex.: DB2 UDB/OSF White Box ADT

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

CREATE FUNCTION distance(p1 point, p2 point) 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 E.name
FROM emp E, town T
WHERE T.name = ‘Ostrava’
            AND distance(E.residence, T.centre) < 25;
```

(User defined) methods

- SQL:1999 adds methods.
- Differences in methods and functions:
  - methods are always tied to a type, functions do not,
  - the given data type is always the type of the first (undeclared) method argument,
  - methods are always stored in the same schema, in which the type, to which they are closed, is stored. Functions are not restricted to a specific schema.
  - functions and methods can be polymorphic, they differ in mechanism of choosing specific methods in run time,
  - signature and body of methods are specified separately,
  - methods calls (dot notation + arguments in brackets).

ADT in SQL:1999

```
CREATE TYPE employee_t AS(
    empID           INTEGER
    name            CHAR(20),
    address         address_t,
    manager         employee_t,
    hire_date       DATE,
    basic_salary    DECIMAL(7,2),
    supplement      DECIMAL(7,2))
INSTANTIABLE
NOT FINAL
REF empID
METHOD worked_years() RETURNS INTEGER
METHOD salary() RETURNS DECIMAL);
```

```
CREATE METHOD
worked_years FOR
employee_t
BEGIN … END;

CREATE METHOD salary
FOR employee_t
BEGIN … END;
```

ADT in SQL:1999

NOT FINAL … can have another subtype

in SQL:1999 structured types have to be NOT FINAL, distinct types have to be FINAL (in SQL:2003 released)

REF enables to understand data (rows) in tables of given type as objects. In the table definition, it is possible to name this „identification“ attribute.

ADT in SQL:1999

Possibilities of specification:

- system-generated

  REF IS SYSTEM GENERATED

  or

  its values are „visible“

  REF IS PID SYSTEM GENERATED
- user-generated

  REF USING <predefined type>
- derived

  REF(<list of attributes>)

Here: reference with empID

Subtypes

```
CREATE TYPE person_t AS(
    name              CHAR(20),
    address           address_t,
NOT FINAL
CREATE TYPE employee_t UNDER person_t(
    empID             INTEGER
    manager           employee_t,    /*employee_t  is a
    hire_date         DATE,           subtype of person_t */
    basic_salary      DECIMAL(7,2),
    supplement        DECIMAL(7,2))
NOT FINAL
REF empID
METHOD worked_years() RETURNS INTEGER
METHOD salary() RETURNS DECIMAL);
```

Subtypes

subtypes

CREATE TYPE clerk_t UNDER employee_t …
CREATE TYPE worker_t UNDER employee_t …

- structured type can be a subtype of another ADT
- ADT inherits the structure (attributes) and behaviour (methods) of its supertypes
  - single inheritance is allowed (multiple one is postponed in SQL standard, some special cases, e.g., in ORACLE)
  - can define additional attributes and methods and can override inherited methods.
- substitutability: a value of subtype can occur in the place a given type

Subtables

employee_t has to be a subtype of persons_t

- apparatus dependent the type apparatus

  CREATE TABLE persons OF person_t

  CREATE TABLE employees OF employee_t

  UNDER persons;

- inherits columns, IOs, triggers, … of the given supertable

Subtables

employee_t has to be a subtype of persons_t

- consistence requirements for subtables and supertables
  - each tuples in supertable (e.g. persons) can correspond mostly to one tuple in subtables (e.g. employees and external_users)
  - tj. each entity has to have the most specified type
- selection limited to the X table using

  FROM ONLY (X)

  – otherwise also from the subtables of X.

Access to the values of attributes

Each attribute has automatically methods generator and mutator

- values selection

```
SELECT e.name()
FROM employees e
```

application of the generator method

- update in 3 steps

```
SET newEmp = employee_t()
newEmp.empID('7897890')
newEmp.name('John')
INSERT INTO employees(newEmp)
```

generates a new instance

Reference and dereference

```
CREATE TYPE account_t AS (
    account_n        INT,
    client           REF(client_t),
    type             CHAR(1),
    opened           DATE,
    interest         DOUBLE PRECISION,
    balance          DOUBLE PRECISION,
    )
FINAL REF IS SYSTEM GENERATED;
CREATE TABLE accounts OF account_t
  (PRIMARY KEY account_n );
```

reference

table

accounts table has a special attribute similar to oid so called self-referencing column

Reference and dereference

What happens when referenced object is removed:

- nothing – implicitly REFERENCES ARE NOT CHECKED
- possibility of an action, if REFERENCES ARE CHECKED ON DELETE (then SET DEFAULT, SET NULL, CASCADE, NO ACTION, RESTRICT)

Dereference

- possible only when location of objects of REF type is defined (one table in SQL:1999)

```
CREATE TABLE clients OF client_t;
CREATE TABLE accounts OF account_t
          (PRIMARY KEY account_n,
           client WITH OPTIONS SCOPE clients
          );
```

allocation

Note: reminds a referential integrity

```
SELECT a.client -> name
FROM accounts a
WHERE a.client->address.town = "Suchdol“ AND a.balance > 100000;
```

dereference, path

Reference and dereference

- dereference by a path and/or by function DEREF

compare

```
SELECT a.opened, a.client
FROM accounts a;
```

and

```
SELECT a.opened, DEREF(a.client)
FROM accounts a;
```

DEREF returns a tuple

Reference and dereference

- benefits of using REF:
  - sharing objects
    - unnecessarily duplicated data
    - the change is done in one place
- link to a method:

```
SELECT a.client() -> name
FROM accounts a
WHERE a.client() -> salary() > 10000;
```

Note: methods without parameters do not require ()

Beyond SQL:1999, 2003

```
CREATE TABLE employees
    (id INTEGER PRIMARY KEY,
    name VARCHAR(30),
    address ROW(    street CHAR(30),
                    house_n CHAR(6),
                    town CHAR(20),
                    zip_code CHAR(5)),
    projects INTEGER SET,
    children person,
    benefits MONEY MULTISET
```

collection

is in SQL:2003

Multisets

nt1 MULTISET EXCEPT [DISTINCT] *nt2*	*nt1* - *nt2*
nt1 MULTISET INTERSECT [DISTINCT] *nt2*	*nt1* $\cap$ *nt2*
nt1 MULTISET UNION [DISTINCT] *nt2*	*nt1* $\cup$ *nt2*
CARDINALITY(*nt*)	$\mid nt \mid$
nt IS [NOT] EMPTY	
nt IS [NOT] A SET	
SET(*nt*)	remove duplicates z *nt*
nt1 = *nt2*	equality of multisets
nt1 IN (*nt2*, *nt3*, ...)	to be in a list of multisets
nt1 [NOT] SUBMULTISET OF *nt2*	comparison of multisets
r [NOT] MEMBER OF *nt*	*r* $\in$ *nt*?
CAST(COLLECT(*col*))	nested table based on *col*
POWERMULTISET(*nt*)	set of all non-empty subsets *nt*
POWERMULTISET_BY_CARDINALITY(*nt*,*c*)	set of all non-empty subsets *nt* with cardinality *c*

Note:

- from SQL:2003: MULTISET without constraint cardinality
- ARRAY without specification of maximal cardinality – it is given by implementation

O-R in commercial products

- INFORMIX: collection - set, multiset, list (without constraint length)
- Oracle from 8i version (since 1999):
  - instead of ADT -- object types
  - notation: CREATE TYPE … AS OBJECT(…);
  - collections
    - VARRAY (equivalent to ARRAY from SQL:1999), but it is not allowed DELETE element from an array
    - for a given array NESTED TABLE (unordered, unrestricted collection of elements)

      Note: multilevel nesting, e.g., in Oracle 11g

CREATE TYPE WhereEverywhere AS VARRAY(4) OF Address

O-R in commercial products

– Id visibility

```
SELECT REF(p) INTO reftoperson
 FROM persons AS p
 WHERE p.name = 'Novák, J.'
```

O-R in commercial products

CREATE TYPE Cars AS TABLE OF Car_t

CREATE TABLE COMPANIES (

fleet Cars

…)

NESTED TABLE fleet STORE AS cars;

Follow the position of ;

Note: You can specify where "subtables" Cars are to be stored.

SELECT *

FROM COMPANIES AS c, c.fleet AS f

WHERE ‘Buick’ IN (SELECT f.car_brand FROM f);

O-R in commercial products

Queries a nested table:

- using THE
- can be treated similarly as with other relations

```
SELECT f.licence_plate
FROM THE (
   SELECT fleet FROM COMPANIES
   WHERE c_name= ‘Komix’)
) f
WHERE f.car_brand=‘Buick’;
```

Q.: Find license plates of all Buicks owned by the Komix company.

O-R in commercial products

Methods in ORACLE

- specification in CREATE TYPE with MEMBER FUNCTION, MEMBER PROCEDURE
- body in the statement CREATE TYPE BODY

Access:

```
SELECT a.client.name
FROM accounts a
WHERE a.client.salary > 10000;
```

Problems with OO in SQL

- tables are the only named entities
- REF type is applicable only on objects given by a row
- UDT is the first step to OO
  - to enable persistency, the object has to be in a table,
  - it is not possible to assign a name to individual instance,
  - it is not possible to query for all instances of an ADT

OR DB design : Transformation E-R → OR

- 1. phase: types
  - entity types → structured types
  - composite attributes → named row types, unnamed row types in structured type, also a structured type is possible
  - multivalued attributes → array of typed values (estimation of max is somewhere important)
  - derived attributes → dd a method into the structured type definit

removed in SQL:2003 with SET as well as ARRAY

OR DB design : Transformation E-R → OR

- relationship types – both single- or bidirectional

  N:1 → with REF + array of typed values (if bidirectional)

  M:N → with one or two arays containing typed values (if single or bidirectional).

- ISA hierarchies → hierarchies of types

- 2. phase: typed tables

Conclusions

- Actual implementations of ORDBMS:
  - complaints:

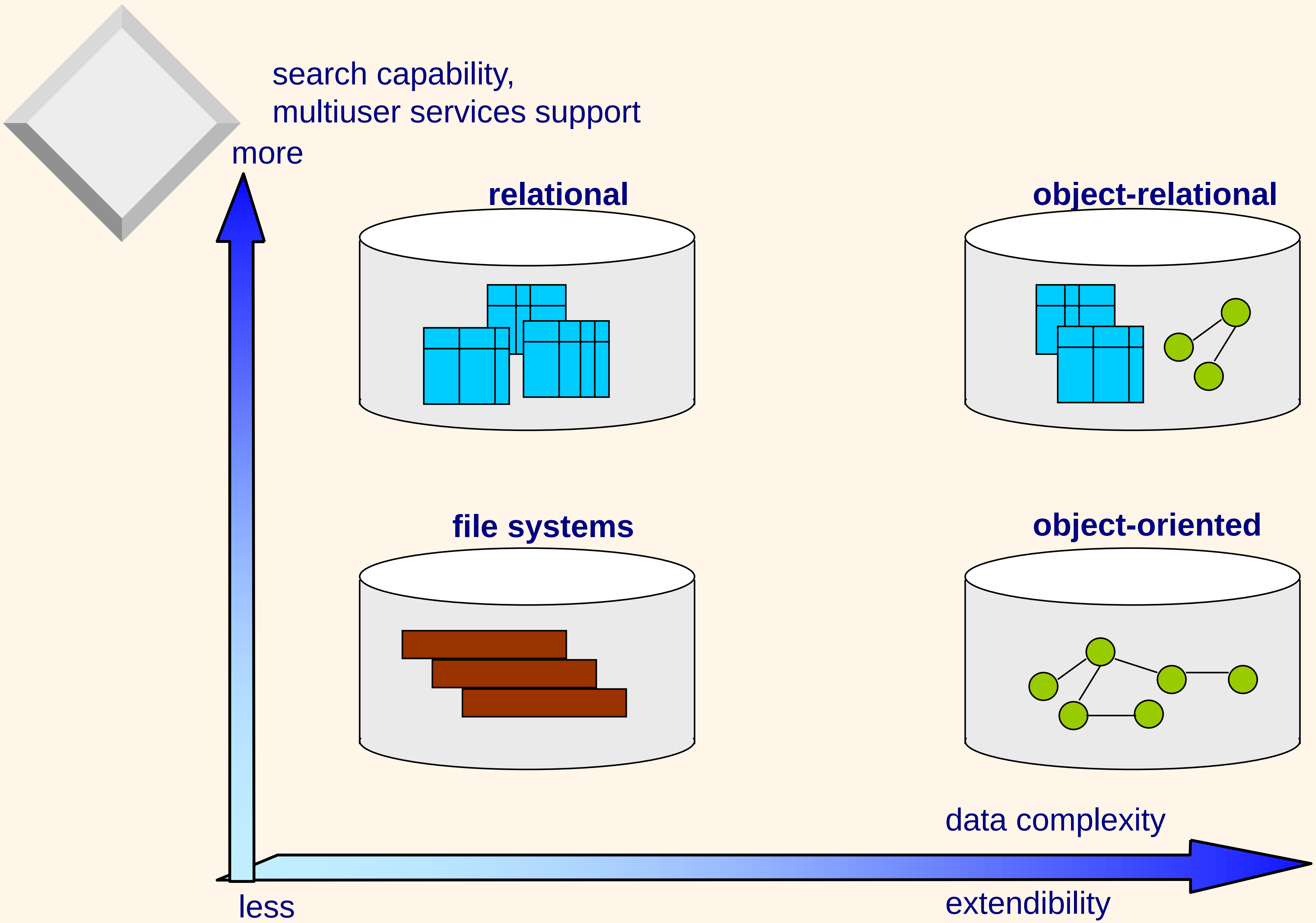
      OODBMS - not database enough

      ORDBMS - not object-oriented enough

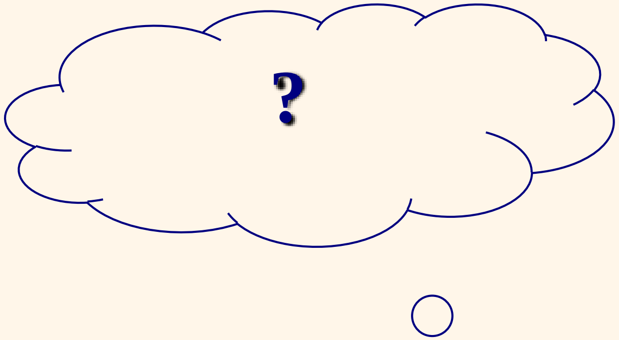
      lack of development tools, new methodologies

      the biggest problem, but also the biggest advantage: universality
- Development goes on: XML DB, NoSQL DB, Web, cloud, NewSQL, …, more generally: non-relational, distributed, open-source and horizontally scalable

The world of DB technologies at the end of the 1st millennium



Conclusions



technologies	70s	80s	90s	2000-10	2010+
research	relational	OO,OR	XML	NoSQL	NewSQL
commerce	hierarchical, network	relational	OO,OR	XML	NoSQL
inherited		hierarchical, network	relational	OO,OR	XML

Lessons learned from history