Objects in Databases

Marek Rychlý
rychly@fit.vut.cz

Brno University of Technology
Faculty of Information Technology
Department of Information Systems

Data Storage and Preparation (UPA)
16 September 2025

Outline

History: 1st half of '80s

Relational databases became the most popular because ...

(instead of network or hierarchical databases)

- easy data model,
- formal-based data model (mathematical relations) and design process (normalization),
- easy and intuitive modeling techniques (ER diagrams),
- significant support of major database vendors,
- standardization in SQL standards.

History: End of ’80s

Emerging object-oriented paradigm and the usage of complex structured data because

- object-oriented programming languages, design and analysis,
- requirements of application domains are beyond relational model.

New database features:

- object databases to implement the persistence of objects,
- relational databases are getting non-relational and object-oriented features to address new requirements,
- object-relational mapping.
  (Hibernate, Java Data Objects/JDO, etc.)

Relational Model

- Relational databases introduced by Codd in 1970, SQL in 1986.
- Relational data model:
  - a collection of tables with atomic/scalar data-types,
  - primary/foreign-key relationships,
  - many-to-many relationships via relationship tables.
- Computation model:
  - based on values in table columns,
  - no native references or pointers,
  - navigation through selected rows in a table is by a cursor.
- Standardization:
  - an SQL non-procedural/declarative query language.

Object Model (1)

- Persistent data can be modeled and stored in objects.
  (DBMS is object-oriented according to OODBS Manifesto, no standardisation)
- Object data model:
  - classes, attributes, operations, a unique OID for each object,
  - abstract data-types (ADTs), encapsulated and polymorphic,
  - attributes can be of complex data-types,
  - OID/reference relationships of objects,
  - many-to-many relationships can be direct.
    (an attribute value is a list of objects in the relationship)
- Computation model:
  - OID plays a significant role,
  - navigation through references/pointers to OID.

Object Model (2)

Standardization:

- ODMG object model, not successful ⇒ no standard available
  (just a proposal of ODL/OQL by ODMG in 1991–2001)
- ODL not needed as the classes/objects are defined at app. level
  (DB drivers integrated into an app. programming languages/libraries)
- OQL not needed as data loaded directly into application objects.
  (querying directly form the app. programming language)

```
class Employee (extent AllEmployees key name) {
  attribute string name;
  attribute Struct Addr { string street, string city } address;
  attribute Enum Job { FULL, PART } job;
  relationship Employe managedBy inverse Employe::manages;
  relationship Set<Employe> manages inverse Employe::managedBy;
}

SELECT e.name, e.managedBy.name
FROM AllEmployees e
WHERE e.name = 'John␣Smith'
```

Object-relational Model

- Advantages of RDBS and OODBS, supported by major vendors.
- Object-relational data model:
  - (nested) relational tables with object-oriented features,
  - persistent data in tables, attributes can be of ADT types,
    (ADT encapsulates data and their operations)
  - relationships can be done by OIDs as well as by PKs/FKs.
- Computation model:
  - the navigation by both cursors and references.
- Standardization:
  - SQL-1999 non-procedural/declarative query language and later.

SQL Versions

1975 Sequel in System R
1986 ANSI standard, ISO standard SQL/86 from SQL in IBM DB2
1989 Integrity Addendum to SQL/89
1992 SQL/92 supported in variants Entry/Intermediate/Full
1996 stored packages in SQL (PSM/96)
1999 SQL-1999 with object-relational features
2002 SQL/MM Framework: Full-text, Spatial, Still image, Data mining
2003 SQL-2003 with OLAP and XML support (SQL/XML)
2006 enhanced XML support in SQL/XML (XQuery)
2008 SQL/XML finished as well as other SQL parts
2011 the support for temporal data
2016 the support for JSON operation in text data
2019 multi-dimensional arrays
2023 JSON data type, graph data and operations

SQL-1999 and Later – Relational Features

- LOB (Large Objects) data-types:
  BLOB (Binary LOB), CLOB (Character LOB)
- BOOLEAN data-type for TRUE, FALSE, UNKNOWN values
- ARRAY data-type of limited number of data items
- MULTISET (2003) w/predicates, set/aggregation operations, etc.
  (predicates IS MEMBER OF, IS A SET, IS A SUBMULTISET OF)
- ROW data-type for structured (non-atomic) data in columns
- SIMILAR predicate for regexp pattern matching (such as LIKE)
- updatable views, recursive queries, transaction save-points, etc.

SQL-1999 and Later – Relational Examples

```
CREATE TABLE test (
  daysofweek VARCHAR(10) ARRAY(7),
  authors VARCHAR(20) MULTISET
);

CREATE TABLE person (
  id INTEGER,
  name ROW (first VARCHAR(20), last VARCHAR(30))
);

SELECT p.name.first FROM person p
WHERE p.name.last SIMILAR 'Smith(y|son)';
```

SQL-1999 and Later – Object Features

Structured User-defined Data Types as ADTs[1]

- analogous to the class concept of object-oriented paradigm,
  (attributes, methods, polymorphic, comparators, inheritance, etc.)
- an ADT can be used as a column data-type of a table,
  (values in such column are not objects but structures, i.e., no OID)
- or an ADT can be used as a type of table,
  (rows of "the typed table" are instances of the ADT with unique OID)
- "REF" can be used create references to the object (to their OIDs).

[1] besides the structured UDTs, there are also "named UDTs" in SQL that are not ADTs; e.g., `CREATE TYPE ssn AS NUMBER(9,0);`

SQL-1999 and Later: Class Definition Examples

```
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;
```

SQL-1999 and Later: Object Definition Examples

```
-- 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;
```

SQL-1999 and Later: Object Manipulation Examples

```
DELETE FROM employee WHERE name = 'John Smith';

UPDATE employee SET name = 'Jane Doe'
  WHERE name = 'John Smith';

-- the ONLY option excepts inherited data from the operation
-- so the following updates persons that are not employee
UPDATE ONLY (persons p) SET p.name = 'Jane Doe'
  WHERE p.name = 'John Smith';
```

Object-relational Database Engines

- Oracle Database (see the documentation)
  (ADT as the type of object table, or of column, i.e., nested table)
- IBM DB2
  (support OR features and non-relational structures like JSON and XML)
- Teradata Database/Vantage Advanced SQL Engine
  (a highly scalable database engine, an ANSI-compliant product)
- Microsoft SQL Server, PostgreSQL, ...

Application Objects

- Object-oriented Programming (OOP) composes an app. from objects.
- Each application object can have its state and behavior.
  (they are represented by the object's attribute values and methods, respectively)
- Each object is usually an instance of a class.
- Two objects with the same state are equal but NOT identical.
  (they can be used in different scopes, have different lifecycles, etc.; the equality is usually determined by their "equal" method)
- Transient objects live in the main memory during app. runtime.
  (the memory is managed by a garbage collector, so the most objects cease to exist when they go out of their scope)
- Permanent objects continue to exist even though they cannot be sensed.
  (e.g., in the case of no references, they are not destroyed by the GC)
- The permanence is an important property in object trees/clusters.
  (a root/cluster object contains references to another/child objects, etc.)

Persistent Objects

- The ability to keep the existence of permanent objects.
  (to keep them stored in a permanent storage when they are not loaded)
- Each persistent object must have an object identifier (OID).
  (OID must provide a global identification of the object)
- With OID, two different but equal objects can be distinguished.
  (missing OID would result into possible errors in storing/loading/using objects)
- Structures that are not objects do not need the OIDs.
  (equal structures are identical, e.g., equal rows in a database table)

Persistent Objects Storage

- To persist the objects, an application needs a persistent storage.
  (usually, provided by an external component, e.g., by a database backend)
- The persistent storage/framework must provide CCSP:
  - **Continuity** so two instances of the same object loaded into an application from the storage must be really identical, e.g., must share their state.
  - **Cohesion** so a group of interconnected objects must persist together (a cohesive group) and the app. can use refs. to go from one to another.
  - **Spatio-temporal Priority** so two objects both representing one entity (that exists in the same place and at the same time in the real-world) must be identical (have the same OID).

Mapping Persistent Objects into the Storage

1. Native mapping
   - The persistence is implemented by an object database.
   - To provide CCSP, the DB must be integrated to an application code.
     (by utilized libraries or by code annotations provided by a developer)
   - Easy and straightforward to use but (maybe) unpopular.
     (the approach requires an object database which is not much reusable)
2. Object-to-(non-object) mapping
   (by a middleware between the application and the storage backend)
   - Object-relational Mapping (ORM)
     (class-to-table/object-to-row mapping&synchronization, data type conversion)
   - Object-document Mapping (ODM)
     (class-to-document-schema/object-to-document mapping&sync., DT conv.)
   - . . . have many disadvantages, yet they are quite popular
     (different granularity, types, life-cycle; difficult to ensure CCSP)

You will use an object database (ObjectBox) in the course lab.

Objects in NoSQL Databases

- An object as a data structure identified by the OID.
  (two equal non-identical objects can be distinguished)
- NoSQL databases are key-value databases.
  (each key uniquely identifies the corresponding value)
- OID as the key, data structure as the value.
- So the question is “How to store the data structure?”.
  1. as an object structure in object NoSQL databases,
     (references, inheritance, methods, etc.)
  2. or as a non-object data; e.g., document structure, property list, etc.
     (this approach would require an object-to-NoSQL-value mapping)

Object Storage in NoSQL

- Native object databases can be implemented as NoSQL.
  (however, there must be a local library managing CCPS properties carefully)
- However, usually it is easier to embed the object DB into app. and use NoSQL database just for synchronization of embedded obj. DBs of multiple app. instances.
  (for example, this approach is utilized in ObjectBox)
- Object-document Mapping (ODM) can be utilized with document NoSQL DBs.
  (e.g., in quite popular MongoDB database)

You will use both ObjectBox and the ODM in MongoDB in the course lab.

Thank you for your attention!

Marek Rychlý

<rychly@fit.vut.cz>