universität wien

Data Engineering

Chapter 1-2

VU Information Management & Systems Engineering

Peter Kalchgruber

Object-Oriented Database Management Systems

universität wien

Object-Oriented Database Management Systems

- Object-Oriented Paradigm: Combines database functions with OO programming techniques.
- Objects and Attributes: Real-world entities with data and methods.
- Classes: Templates for creating objects, defining structure and behavior.
- Encapsulation: Bundles data and methods and restricts direct access.
- Inheritance: New classes inherit features from existing ones.
- Polymorphism: Different classes, common interface; method redefinition in subclasses.
- Object Identity: Unique identifiers for each object, independent of attributes.
- Object Relationships and Associations: Manages complex inter-object relationships efficiently.

Storage and Retrieval

Storage

- **Object Persistence**: Objects exist beyond creation, ensuring data continuity.
- **Complex Structures**: Naturally stores nested and interconnected object structures.
- **Object Identity**: Each object is uniquely identified through persistent OIDs.
- **Direct Storage**: Objects stored as-is, not in rows/columns.

Retrieval

- **Navigational Access**: Retrieve by navigating through object network relationships.
- **Querying**: Uses Object Query Language (OQL) for object-oriented data querying.
- **Indexing**: Complex indexing methods for efficient object retrieval.
- **Characteristics-Based Retrieval**: Search based on attributes, behavior, and relationships.
- **Handling Relationships**: Exploits object associations for intuitive data access.

Query Languages in Object-Oriented Database Management Systems

- **Nature**: Incorporates object-oriented concepts to extend traditional query capabilities for complex data handling.
- **Integration**: Seamlessly integrates with object-oriented programming languages for data manipulation.
- **Object-Oriented Features**: Supports encapsulation, polymorphism, inheritance in database queries.
- **Complex Data Types**: Manages arrays, lists, and custom structures effectively in queries.
- **Complex Queries**: Enables navigating through intricate object relationships and hierarchies.
- **Performance**: Potentially slower for deep hierarchies and complex relational queries.
- **Extensibility**: May offer extensibility for specific application requirements and functions.

Object Query Language: Example I

Task

Identify authors with multiple highly-rated books published in the last 5 years.

OQL

```
SELECT DISTINCT author
FROM Authors author
WHERE (SELECT COUNT(*)
FROM author.books book
WHERE book.publication_year
>= (CURRENT_YEAR - 5) AND
book.averageReviewRating() >
4.0) >= 2
```

SQL

```
SELECT DISTINCT a.author_id, a.name
FROM Authors a
INNER JOIN BooksAuthors ba ON
a.author_id = ba.author_id
INNER JOIN Books b ON b.book_id =
ba.book_id
INNER JOIN Reviews r ON r.book_id =
b.book_id
WHERE b.publication_year >=
(YEAR(GETDATE()) - 5)
GROUP BY a.author_id, a.name
HAVING AVG(r.rating) > 4.0
AND COUNT(DISTINCT b.book_id) >= 2
```

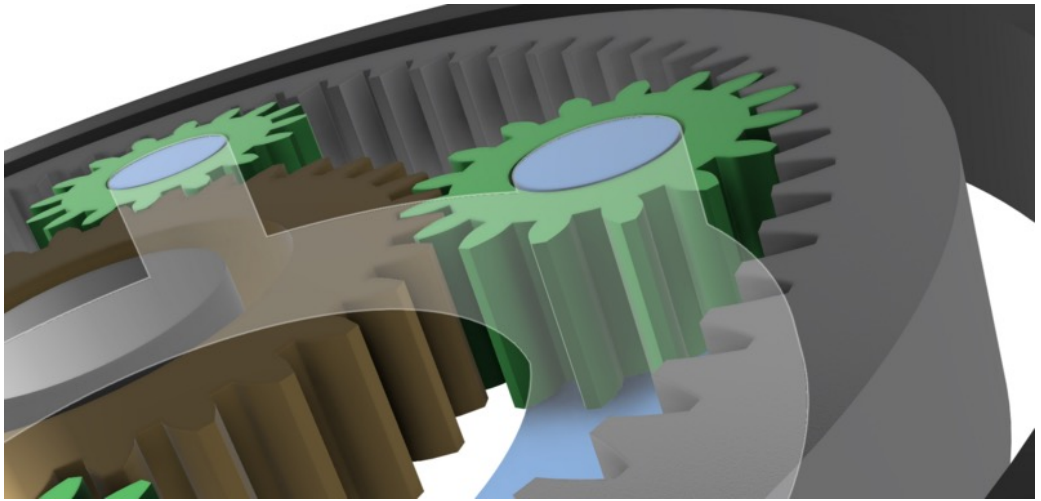
Object Query Language: Example II

Task	OQL	SQL

Find books similar to a reader's favorites based on genre and reviews.	`SELECT book`
	`FROM Books book`
	`WHERE`
	`reader.hasSimilarFavorite(book)`
	`AND book NOT IN reader.favoriteBooks`
		?

OODBMS: Use Cases

- CAD/CAM (Computer-Aided Design/Manufacturing)
- Geographic Information System (GIS)
- Digital Asset Management (DAM)
- Content Management Systems (CMS)
- Scientific Data
- Multimedia Applications



Pros and Cons

Pros

- **Reduced Impedance Mismatch**: Seamlessly maps object-oriented concepts to database representation, simplifying application logic
- **Complex Data Handling**: Natively supports complex data structures, custom types, and rich object relationships
- **Intuitive Queries**: Querying often follows natural object navigation and method calls

Cons

- **Less Mature Technology**: Smaller market share compared to RDBMS solutions
- **Potential Learning Curve**: OODBMS concepts and query languages might be less familiar to developers
- **Performance Considerations**: Some OODBMS can face challenges in scenarios traditionally dominated by highly optimized relational databases
- **Vendor Support**: May have fewer vendor choices or less extensive support networks compared to major RDBMS

Comparison

RDBMS vs. ORDBMS vs. OODBMS

Comparison I

	RDBMS	ORDBMS	OODBMS
Data Model	Relational (tables, rows, columns)	Hybrid (relational + object extensions)	Pure object-oriented
Query Language	SQL	SQL with OO extensions	Object-oriented (OQL)
Navigation	Joins	Blend of joins and object navigation	Object traversal
Complex Data	Requires mapping to tables	Can handle, often less direct	Native support
Persistence	Explicitly managed (saving/loading)	Persistence of OO structures, some mismatch	Transparent object persistence
Use Case Fit	Structured data, strong standardization	Primarily relational with some OO aspects	Complex object structures

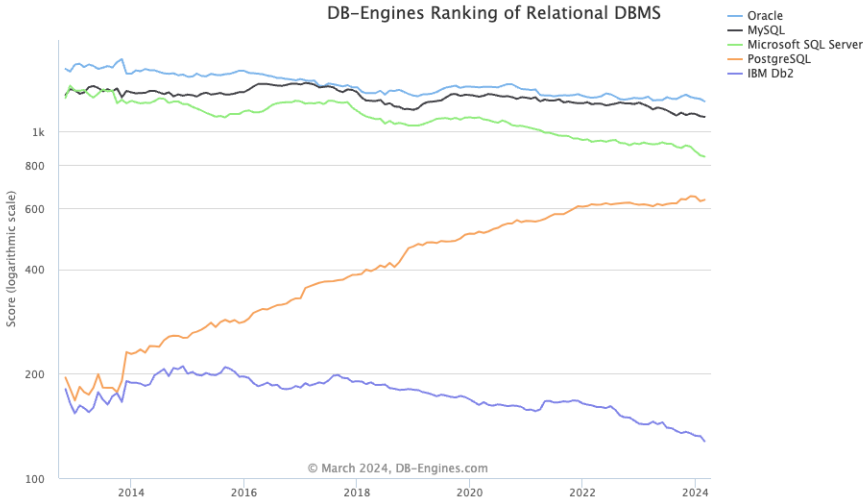
Comparison II

	RDBMS	ORDBMS	OODBMS
Data Model Mismatch	High	Medium	Minimal
Complex Queries	Can be Cumbersome	Potentially Easier	Often Intuitive
Data Structure Flexibility	Limited	Moderate	High
Development Speed (Right Fit)	Slow (Complex Objects)	Moderate	Fast (Complex Objects)
Scalability (Large Datasets)	Highly Optimized	Vendor-Dependent	Varies
Maturity & Support	Extensive	Well-Established	Niche

Popular ORDBMSs

ORDBMS

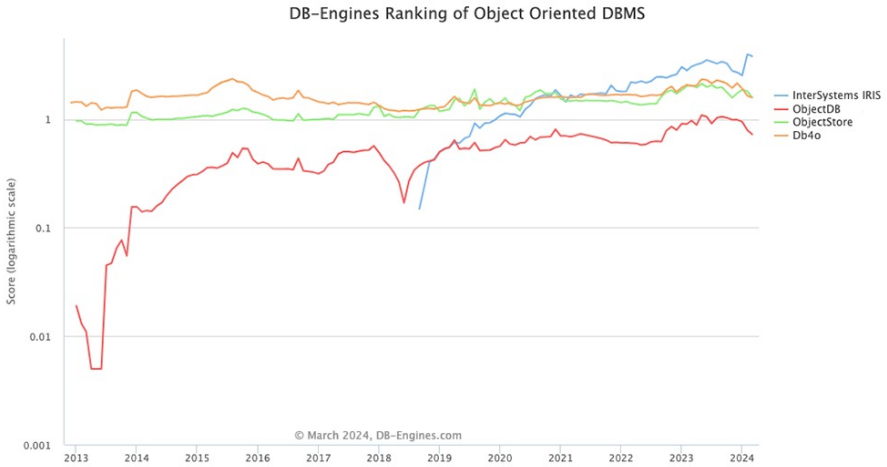
- **PostgreSQL**: An advanced ORDBMS that supports various data types, including custom types, and offers powerful programming and querying features.
- **Oracle Database**: Oracle offers robust object-relational features and is widely used in enterprise environments for its robustness and extensive feature set.
- **IBM DB2**: Offers strong object-relational capabilities, particularly in its advanced editions, and is widely used in enterprise and large-scale systems.
- **Microsoft SQL Server**: While primarily a relational database, it has expanded to include more object-oriented features like user-defined types and functions.



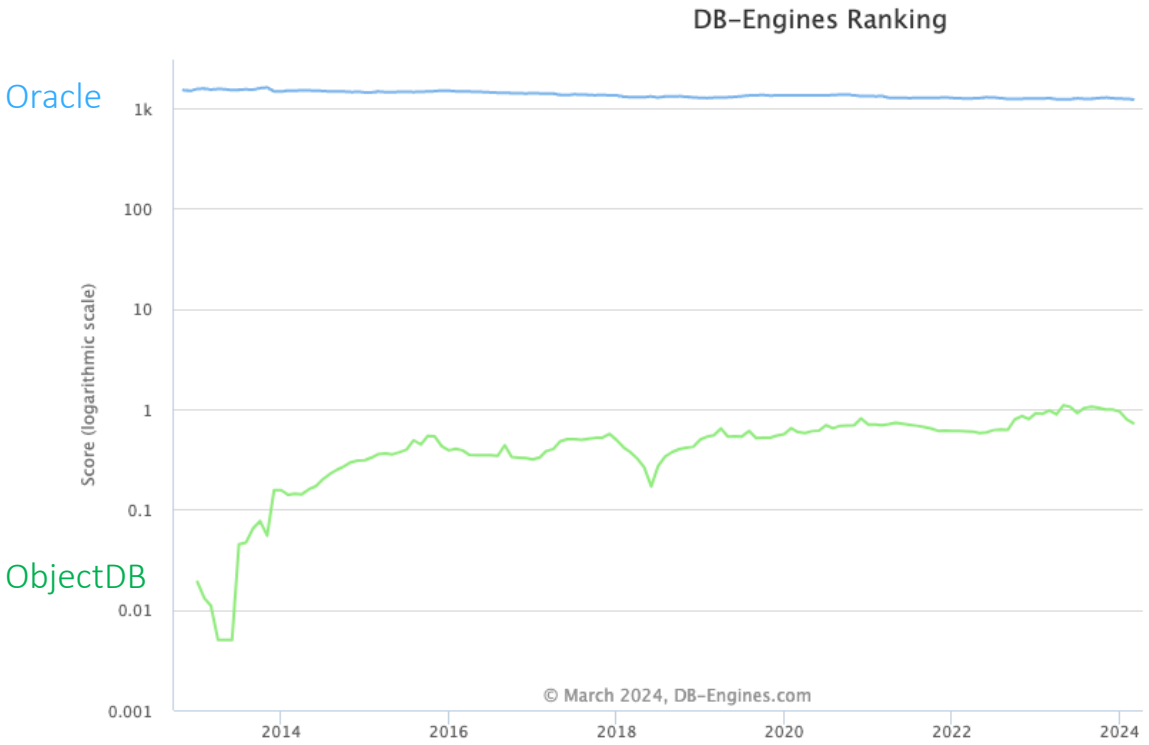
Popular OODBMSs

OODBMS

- **InterSystems Caché**: Known for its high performance, it integrates object database, SQL, and analytics capabilities.
- **db4o** (database for objects): Specifically designed for object-oriented languages like Java and .NET.
- **ObjectDB**: An object database for Java and JPA/JDO. It's very efficient for Java-based applications.
- **ObjectStore**: A mature OODBMS with strong support for commercial applications, offering versions for Java, C++, and other languages.



ORDBMS vs. OODBMS



Data Engineering - Table of Contents

Extend

Object DBMS

- Object–relational mapping
- Object-Relational DBMS
- Object-Oriented DBMS
- Object Query Language

Distribute

Parallel and Distributed DBMS

- Replication
- Fragmentation
- Parallel Execution
- Map-Reduce
- Hadoop Ecosystem

Simplify

Semi-Structured Data and NoSQL DBMS

- XML, JSON
- NoSQL Database Systems
  - Key Value Store
  - Document Store
  - Graph Database
- Mongo DB
  - Data migration

Parallel and Distributed DBMS









Drivers of Database Evolution: From Traditional to Parallel and Distributed Systems

Parallel Databases

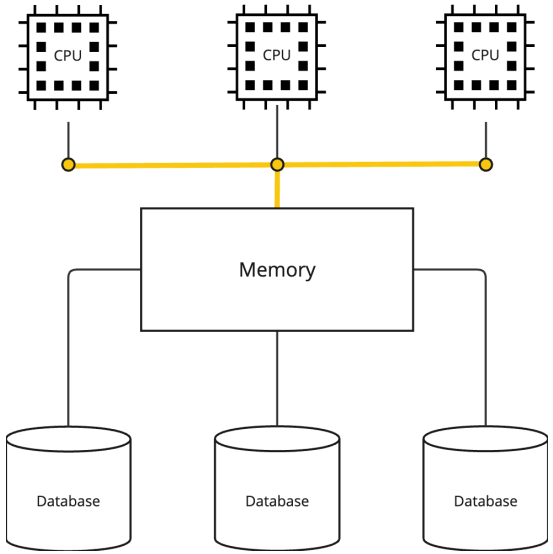
- Increasing Data Volumes
- Need for Faster Query Processing
- Scalability Limitations
- High Availability and Reliability Requirements
- Advancements in Parallel Computing
- Complex Analytical Workloads

Distributed Databases

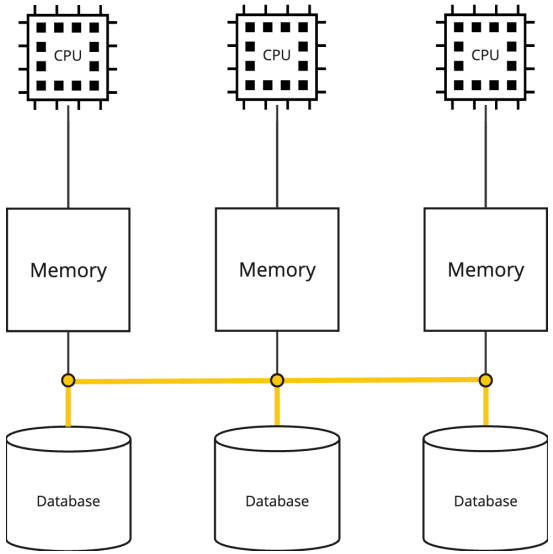
- Geographical Distribution
- Scalabilty Concerns
- High Availability and Disaster Recovery
- Network and Internet Growth
- Diverse Data Sources and Big Data
- Load Balancing
- Local Autonomy
- Cost-Effectiveness and Efficiency

Parallel Database Architectures

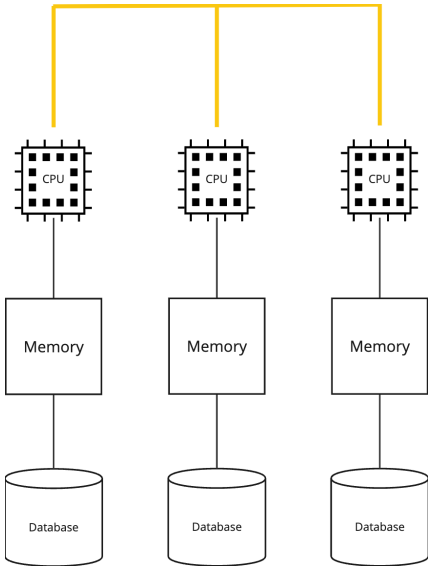
Shared Memory



Shared Disk

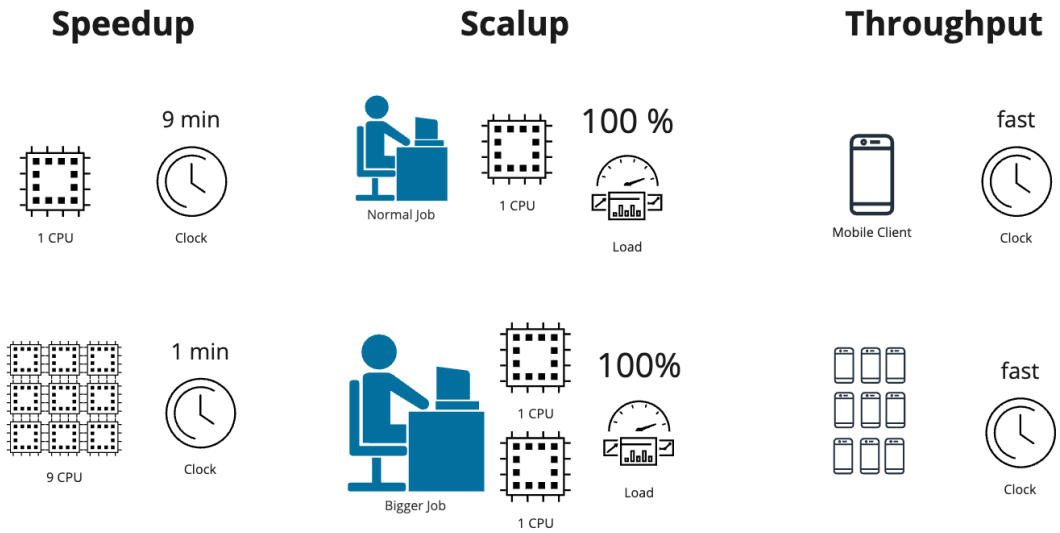


Shared Nothing



Performance Metrics in Database Scalability

- **Speedup**: the same job, more hardware, less time
- **Scaleup**: bigger job, more hardware, same time
- **Throughput**: more clients/servers, same response time



Distributed Database Systems - Data Replication

Multiple copies of data stored on different sites

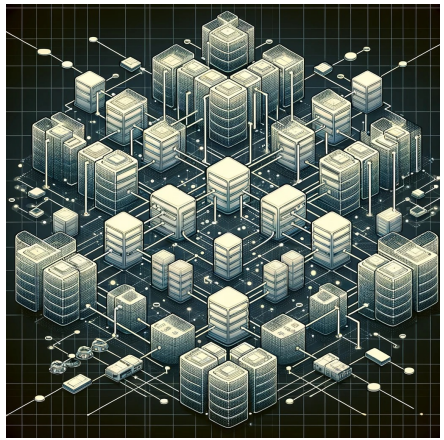
+ Availability
+ Fast (local) access
+ Performance by parallel execution
- High data update cost (each replicate)
- Complex Concurrency-Management

- **Synchronous Replication**
  - Changes are atomically applied to destination DBs
  - Consistency: Strong
  - Latency: Higher
- **Asynchronous Replication**
  - Changes propagated to destination DBs with some delay
  - Consistency: Moderate
  - Latency: Lower

Distributed Database Systems – Design Considerations

Network Structure

- Latency
- Bandwith
- Network Partitioning



Design Considerations

- Homogeneous vs. Heterogeneous Systems
- Client-Server vs. Peer-to-Peer Architecture
- Transparency

Single System Image (SSI) for Distributed Databases

- Provides the illusion of a centralized system despite distributed data.
- Key aspects:
  - Abstraction (hides infrastructure complexity)
  - Unified Interface (consistent way to interact with data)
  - Global Schema (centralized view of all data)
- Enables other transparencies (replication, fragmentation, location)

Transparency in Distributed Systems

Replication transparency

Users view data items as logically unique and are not concerned about which data item is replicated.

Fragmentation transparency

Users have not to know if and how a relation has been fragmented

Location transparency

Users are not required to know the physical location of a data item

Benefits of Transparency in Distributed Systems

- Easier application development (data location agnostic)
- Simpler data management (unified interface)
- Improved scalability (easier to add resources)
- Increased fault tolerance (redundancy hidden)

Examples

- **E-commerce Platforms**: Amazon, ebay Alibaba
- **Content Sharing Services**: Facebook, X (Twitter)
- **Cloud Storage Provider**: Dropbox, Google Drive
- **Travel Booking Services**: Expedia, Booking.com

Distributed Database Systems - Fragmentation

Partitions (fragments) are stored on different sites

+ Fast (local) access
+ Performance by parallel execution
- High cost for de-fragmentation of a relation

- **Horizontal Partitioning**: Data rows are distributed across different sites.
- **Vertical Partitioning**: Different columns of a table are stored at different sites.
- **Full Replication**: Each site holds a complete copy of the database.

Vertical Fragmentation - Example

Books Table

id	title	author	ISBN	publication_year
1	The Lord of the Rings	J.R.R. Tolkien	978-0547928227	1954
2	Harry Potter and the Sorcerer's Stone	J.K. Rowling	978-0590353427	1997
3	The Hitchhiker's Guide to the Galaxy	Douglas Adams	978-0345391803	1979

Books Description Table

id	description	full_text
1	A tale of hobbits, elves, wizards...	(Very long text of the entire book)
2	A young boy discovers he's a wizard...	(Very long text of the entire book)
3	An earthling's comedic journey through space...	(Very long text of the entire book)

Horizontal Fragmentation - Example

Checkouts Table

id	book_id	member_id	checkout_date	due_date
1	2	1001	2023-12-20	2024-01-10
2	1	1005	2023-12-25	2024-01-15
3	3	1002	2024-01-02	2024-01-23

Checkouts Archive Table

id	book_id	member_id	checkout_date	due_date	return_date
4	1	1003	2023-08-12	2023-08-26	2023-08-26
5	3	1001	2023-11-01	2023-11-22	2023-11-20

Exploring Parallelism in Databases
Intra-Query vs. Inter-Query

Intra-Query Parallelism

- Dividing a single query into subtasks
- Parallel execution on a multi-core CPU
- Applicable to operations like scans, joins, aggregations, sorting

Inter-Query Parallelism

- Distributing query tasks across multiple database servers
- Leveraging multiple CPUs and storage devices
- Suitable for complex queries or large datasets

Distributed vs. Parallel Databases

	Parallel Database System	Distributed Database System
Geographic Location	Nodes located nearby, to a data center	Nodes spread across multiple locations, often wide distances
Administration	Typically centralized administration	Potential for separate administration of individual nodes
Network	Utilizes fast interconnects (high-speed LAN within a datacenter)	Relies on slower interconnects (WAN, internet)
Transactions	Primarily focuses on local transactions	Distinguishes local vs. global transactions
Design Focus	Performance, scalability for large datasets	Data sharing, autonomy, availability
Complexity	Comparatively less complex	Generally higher complexity in design and operation
Consistency	Often aims for stronger consistency guarantees	May face eventual consistency challenges across nodes

Literature

- Wiese, Lena. Advanced data management: for SQL, NoSQL, cloud and distributed databases. Walter de Gruyter GmbH & Co KG, 2015. Link
- Özsu, M. Tamer, and Patrick Valduriez. Principles of distributed database systems. Vol. 2. Englewood Cliffs: Prentice Hall, 1999.
- Harrington, Jan L. Object-oriented database design clearly explained. Morgan Kaufmann, 2000.
- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. "Database system concepts." (2011).
- White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.