# Data Engineering

Chapter 1-1

VU Information Management & Systems Engineering

Peter Kalchgruber

Running jobs: 365644
Active CPU cores: 807139
Transfer rate: 21.54 GiB/sec

# CERNs Challenge: Datagrid

- Large Hadron Collider
  - Purpose: Test theories of particle physics
  - Design: 27km, -271,3° C.,
  - Detectors: ATLAS, CMS, LHCb, ALICE
  - Discoveries: Higgs Boson
  - Global Collaboration

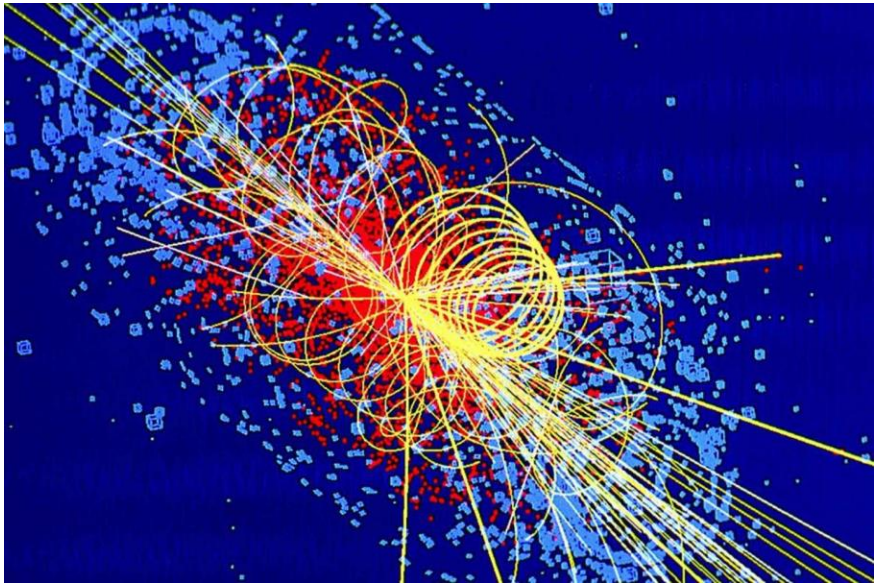
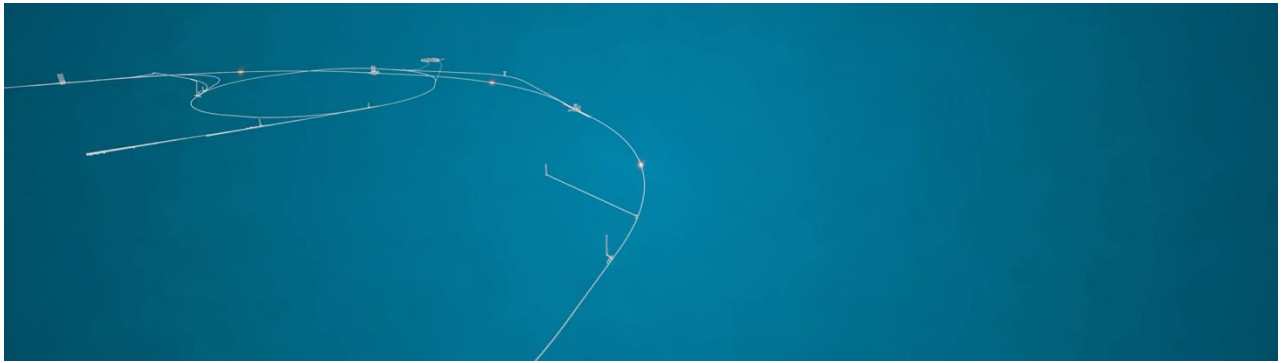


*Image: https://home.cern/news/news/physics/higgs-boson-one-year*

# Worldwide LHC Computing Grid (WLCG) - Datagrid



Source: https://videos.cern.ch/record/1951286

- Decentral system (Grid computing)
  - 1.4 million computer cores
  - 2 exabytes of storage
  - 2 million tasks per day
  - Transfer rates > 260 GB/s

# Worldwide LHC Computing Grid (WLCG) - Datagrid

- Essential component of the LHC's research infrastructure
- Process, store, distribute and analyze petabytes of data
- Global Collaboration (170 computing centers, 42 countries)
- Tiered Structure
  - Tier 0 (CERN)
  - Tier 1 (13 major centers worldwide)
  - Tier 2 (over 150 centers)

Full list: http://wlcg-cric.cern.ch/wlcg/fedrcsite/list/

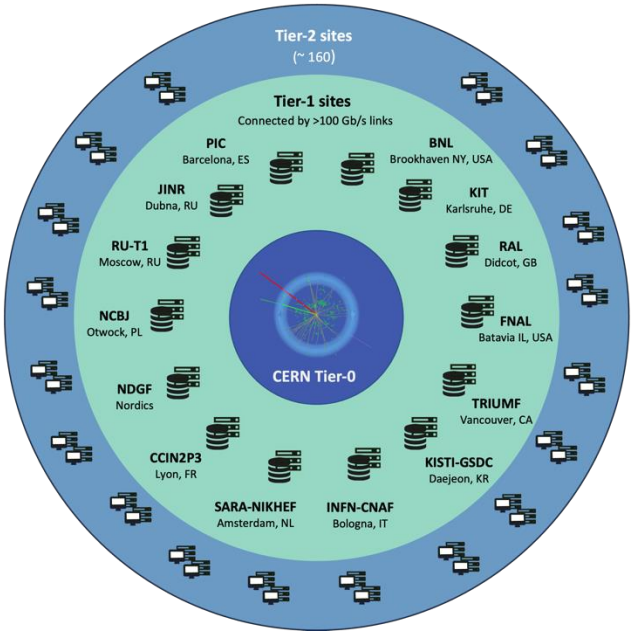



*Image: NCBJ: https://www.ncbj.gov.pl/en/news/cis-achieves-tier-1-status-lhcb-experiment-cern*

# WLCG Challenges

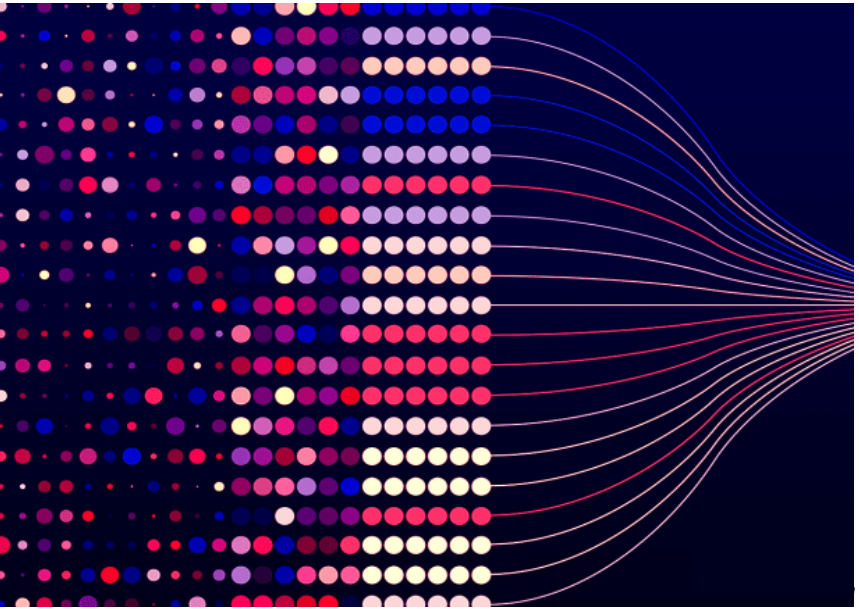


*Image: https://builtin.com/big-data/data-integration*

Data Integration and Interoperability

Data Security and Privacy

Scalability

Data Lifecycle Management

Resource Allocation and Management

Real-time Processing and Streaming Data

Machine Learning and Advanced Analytics Integration

Cost Management

# 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

# Data Engineering

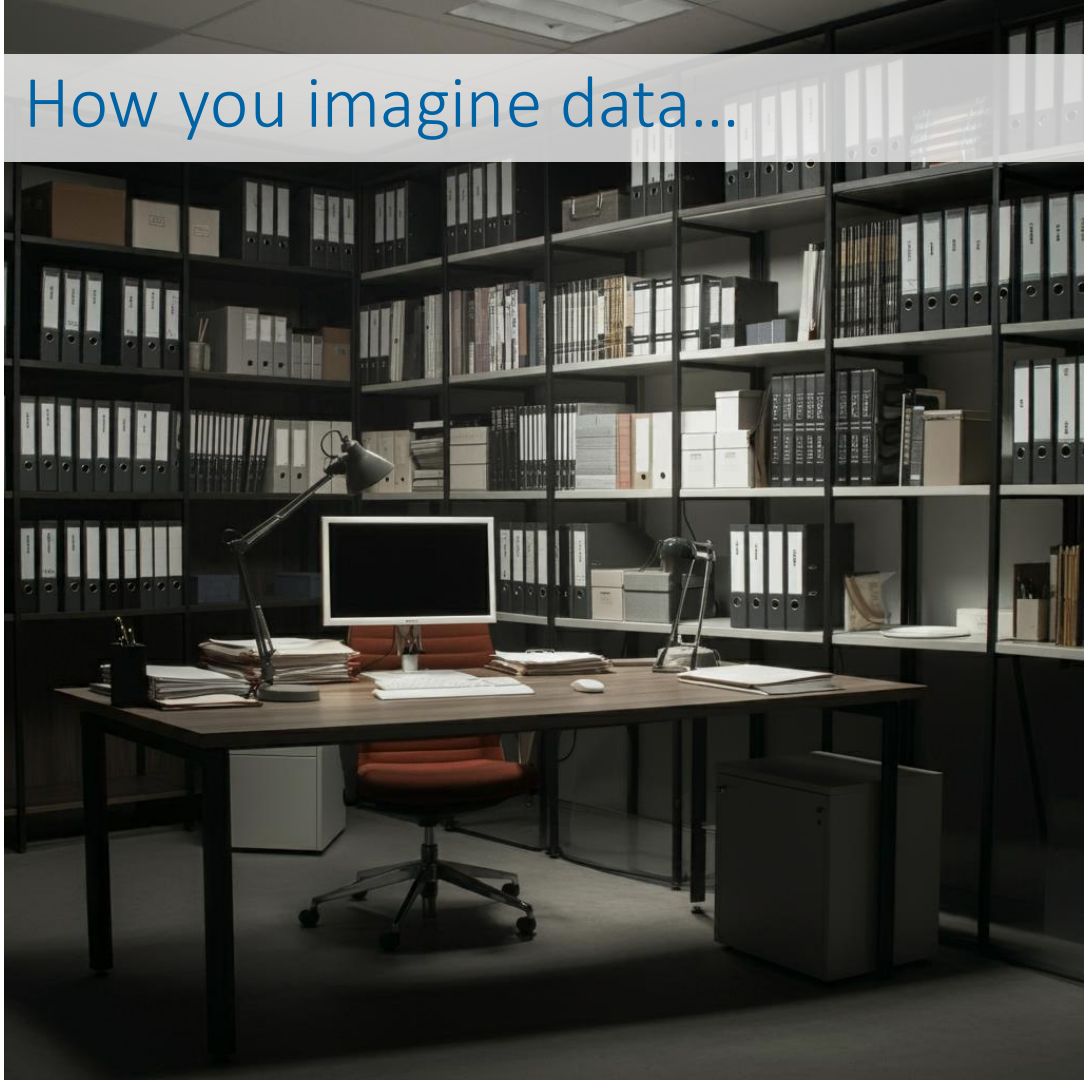
Basics

universität wien

# What is Data Engineering?

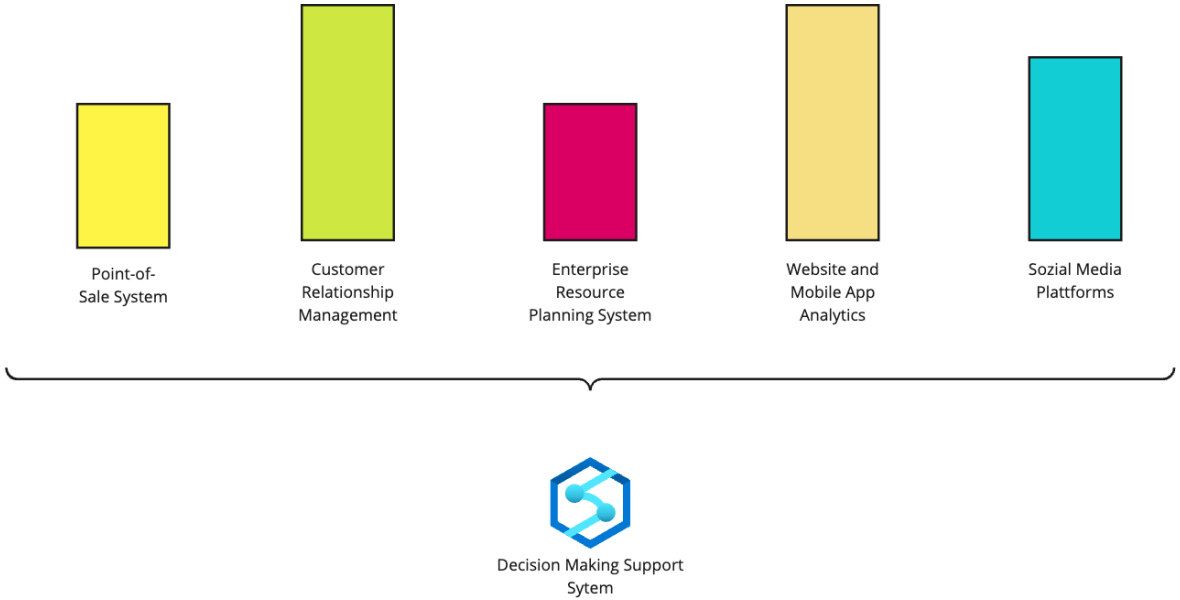
- Data Ingestion: Collect, clean, and validate raw data (Extract)
- Data Transformation: Convert, clean, enrich, and aggregate data (Transform)
- Data Storage: Store and manage data securely and scalably (Load)
- Data Retrieval: Access and extract data efficiently

# How you imagine data…



# How data look like…

# Why are data moved?



# Role of data engineers

- Primary responsibility
  - Design, build, and maintain the infrastructure that enables the seamless flow and it's utilization of data throughout the organization
- Work area
  - Data Pipeline Development
  - Database Management
  - ETL Processes
  - Data Quality
  - Collaboration

# Comparison of…

| Role | Data Engineer | Data Scientist | Data Analyst |
|---|---|---|---|
| Focus | Build data pipelines and infrastructure | Analyze data, build predictive models | Analyze trends, support decision-making |
| Key Skills | SQL, ETL, cloud platforms, data architecture | Stats, ML, Python/R | Data visualization, data analysis tools |
| Tools | Spark, Hadoop, AWS | Python, R, TensorFlow, PyTorch | SQL, Excel, Tableau, Power BI, R, Python |
| Output | Reliable, scalable data infrastructure | Predictive models, insights, experiments | Reports, dashboards, presentations |
| Background | Software engineering, IT | Math/Statistics – Computer Science | Business, Economics, Statistics |

# Comparison of…

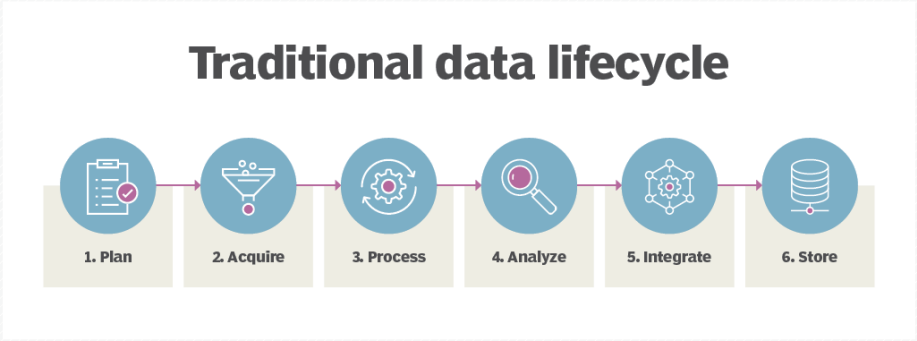
| Role | Data Engineer | Data Scientist | Data Analyst |
|---|---|---|---|
| | data pipelines and ructure | Anal ze data, build pr ictive models | Anal e trends, support deci n-making |
| | , cloud platforms, chitecture | ats, ML, Python/R | Dat alization, data ana ols |
| | park, | ython, R, TensorFlow ch | |
| Output | R | e mode xperi | |
| Background | Sc | atistics - uter Science | |

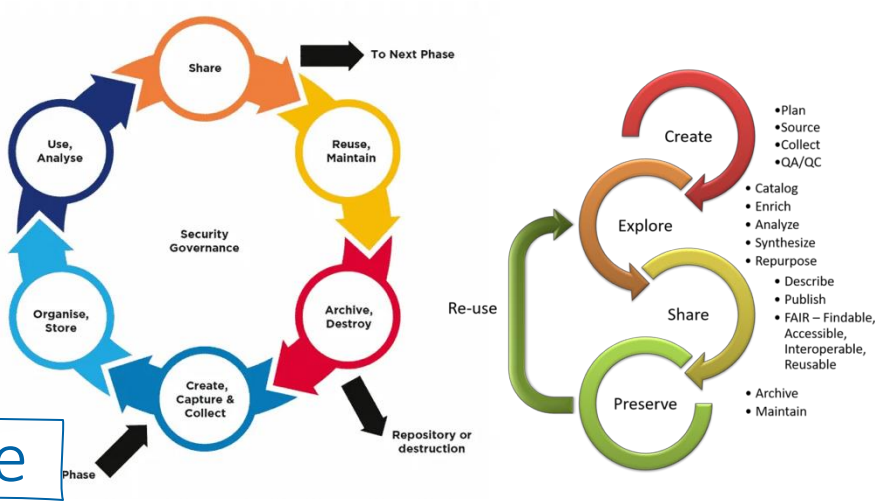
How can we design a data pipeline that ingests real-time streaming data from our website and ensures it is stored in a scalable, fault-tolerant data warehouse?

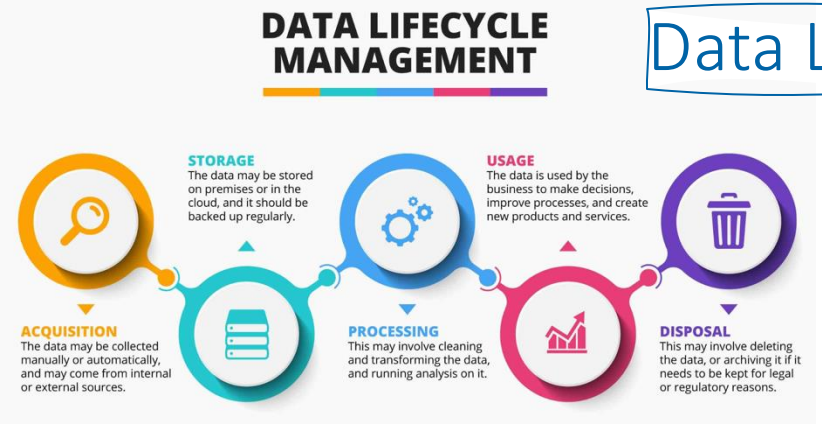
How can we use historical customer data to build a predictive model that forecasts customer churn over the next six months?

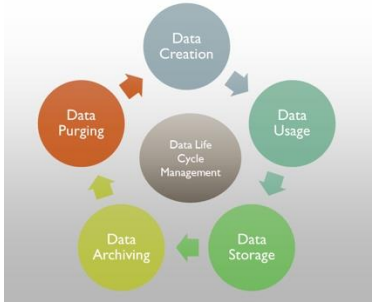
What are the key trends in our sales data over the last quarter, and how can we visualize these to support the marketing team's decisions?

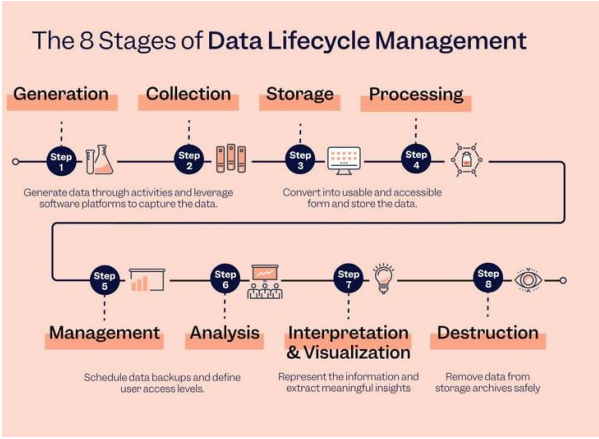
# Data Lifecycle



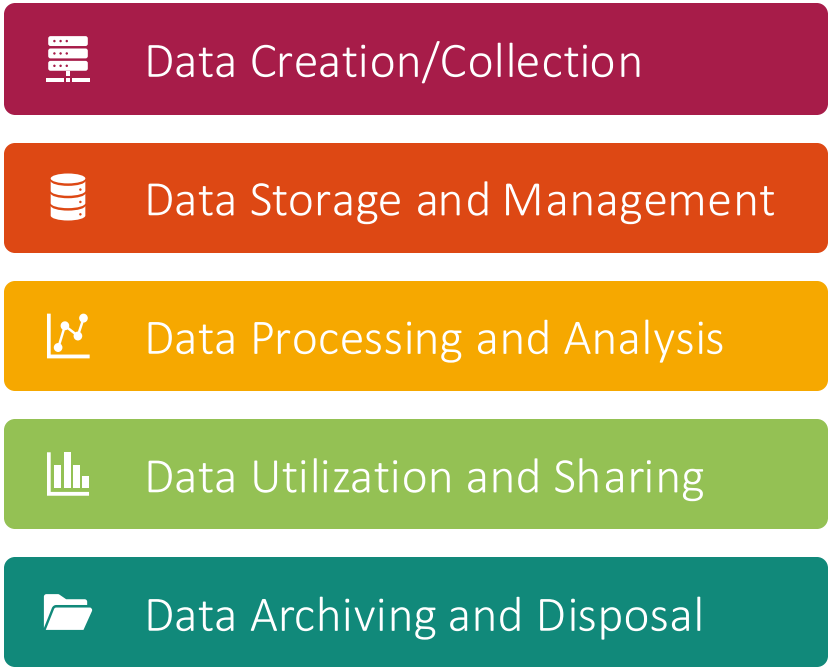




https://www.techtarget.com/searchstorage/definition/data-life-cycle-management
https://formiti.com/the-importance-of-data-lifecycle-management-what-it-is-and-why-you-need-it/
https://www.scilife.io/blog/stages-data-lifecycle-management
https://youngdestinya.ng/data-lifecycle-management-explained/
https://libguides.asu.edu/c.php?g=264208&p=10002198
https://www.linkedin.com/pulse/data-lifecycle-management-muhammad-salahuddin-irbfe/

# Data Lifecycle



# Data Sources and Collection

- Types of data sources
  - Structured
  - Semi-structured
  - Unstructured
- Data collection methods
  - APIs
  - Databases
  - Streaming data
  - Web scraping

- Challenges in data collection
  - Data quality
  - Data volume
  - Data variety

# 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

# Object-Relational Database Technologies

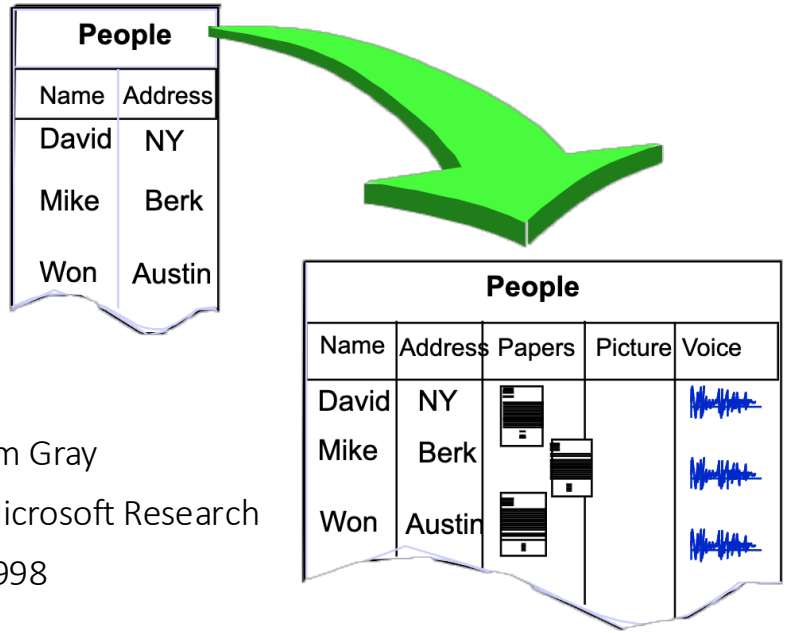
# Can traditional relational databases satisfy the requirements of today's scenarios like WLCG?

## The Old World

- Millions of objects
- 100-byte objects

## The "New" World

- Billions of objects
- Big objects (1MB)
- Objects have behavior (methods)




Jim Gray

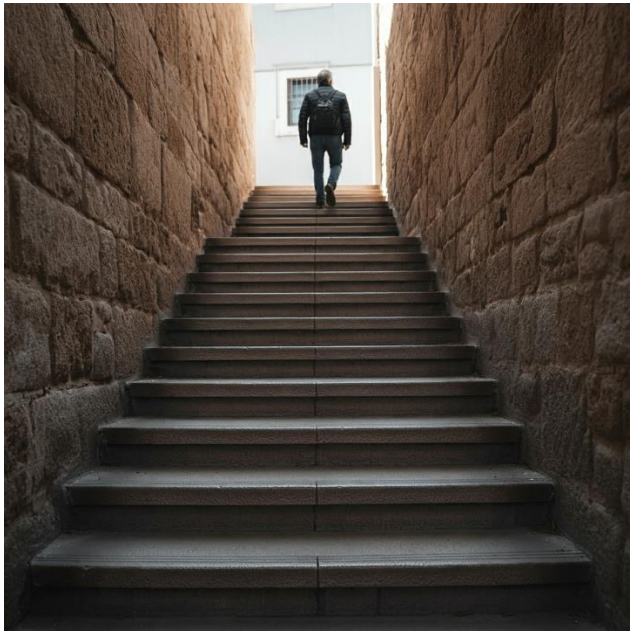
Microsoft Research

1998

# New Requirements for Data Management

- Explosion of unstructured and semi-structured data (text, sensor data, social media feeds, JSON, etc.)
- Complex data types (arrays, maps, nested structures) enable natural modeling of these data forms
- Machine learning algorithms often work with complex data representations (text embeddings, image vectors, geospatial coordinates, etc.)

# Evolutionary Approach



# Revolutionary Approach

# Evolutionary Approach

- Extends existing relational database models with object-oriented features
- Focuses on backward compatibility with traditional SQL and relational schemas
- Adds complex data types and structures, like nested tables or arrays
- Gradual integration ensures a smoother transition for users and developers
- Maintains the robustness and reliability of established relational databases

# Revolutionary Approach

- Builds database systems fundamentally on object-oriented principles
- Supports OO features like inheritance, polymorphism, and encapsulation natively
- Redesigns data storage and access to be object-centric
- Often requires a complete restructuring of existing database systems
- Prioritizes object-oriented design over traditional relational models

# Object-relational impedance mismatch

| | Object-Oriented Programming | Relational Databases |
|---|---|---|
| Data Structure | Classes and objects with fields and methods | Tables with rows and columns |
| Data Manipulation | Through methods in objects | SQL queries and operations |
| Inheritance | Supports inheritance and polymorphism | Lacks direct support for inheritance |
| Identity | Object identity tied to object references | Identity via primary keys in tables |
| Relationships | Object references and associations | Foreign key constraints and joins |
| Schema Evolution | Changes require object model modification | Changes require alteration of tables |
| Transaction Support | Often managed within the application code | Managed by the database system |
| Concurrency Control | Typically, through object-locking mechanisms | Managed through transaction isolation & locking at the DB level |
| Query Language | Object query languages or methods | Structured Query Language (SQL) |
| Data Types | Supports complex, user-defined types | Limited to predefined data types |

# Object-Relational Mapping (ORM)

# Object-Relational Mapping (ORM)

.NET Entity Framework

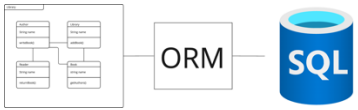
HIBERNATE

django

SQLA

Sequelize

- **Definition**: Technique that bridges the gap between object-oriented programming and relational databases.




- **Functionality**: Automates the translation of objects in code to relational tables in a database, simplifying data manipulation.
- **Benefits**: Simplifies database interactions, enhances productivity and maintains data integrity.
- **Process**: Maps objects to database entities; CRUD operations in code translate to SQL queries.
- **Use Cases**: Widely used in applications requiring database interaction, such as web and enterprise applications.
- **Popular Frameworks**: Hibernate (Java), Entity Framework (.NET), Django ORM (Python).

# ORM Example with Python I

## Pure Python Database Access

```
def create_book(title, author, pub_year):
    conn = sqlite3.connect('library.db')
    cursor = conn.cursor()
    cursor.execute('''INSERT INTO Books (title, author,
    pub_year) VALUES (?, ?, ?)''', (title, author, pub_year))
    conn.commit()
    conn.close()

create_book("1984", "George Orwell", 1949)
```

## ORM-Based Access with Django

```
class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    pub_year = models.IntegerField()

new_book = Book(title="1984", author="George Orwell",
pub_year=1949)
new_book.save()
```

# ORM Example with Python II

## Defining Model Relationships in Django

```
class Genre(models.Model):
    name = models.CharField(max_length=50)

class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author) #def
    published_date = models.DateField()
    genre = models.ManyToManyField(Genre)
```

## Applying Model Changes with Django Migrations

```
python manage.py makemigrations
python manage.py migrate
```

# Object-Relational DBMS vs. Object-Oriented DBMS

## Object-Relational DBMS

- **Hybrid Model**: Combines relational and object-oriented database features.
- **Complex Data Types**: Supports structured types and nested objects.
- **SQL Enhancements**: Extends SQL to handle OO features like inheritance.

## Object-Oriented DBMS

- **OO Principles**: Built entirely on object-oriented concepts like encapsulation, inheritance, and polymorphism.
- **Direct Object Storage**: Stores objects as they are used in OO programming languages.
- **No SQL Required**: Interacts with data using object-oriented languages, bypassing traditional SQL.

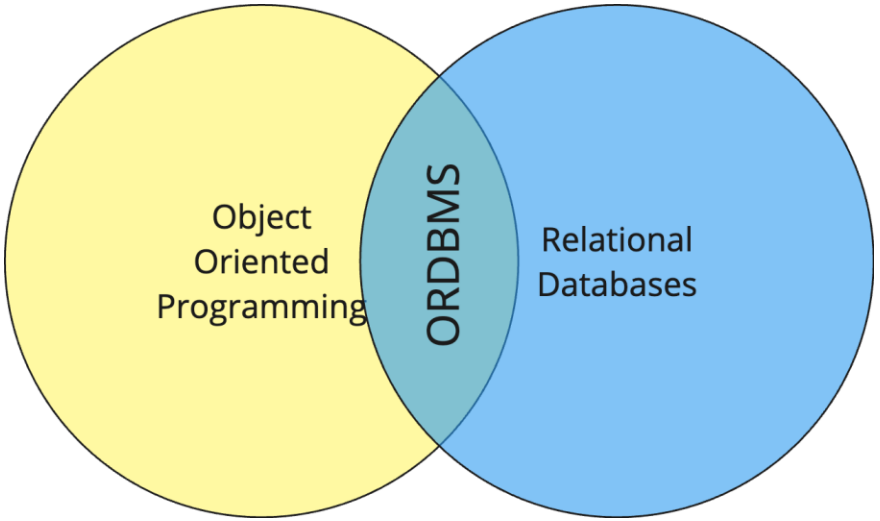
# Object-Relational DBMS

# Object-Relational Database Management Systems (ORDBMS)

## Overview

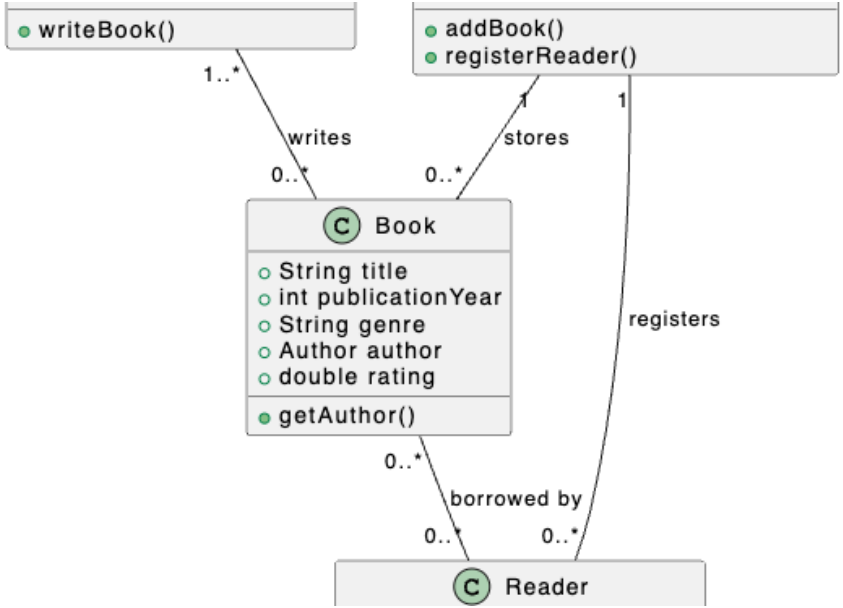
- **Definition**: ORDBMS extends traditional relational models with object orientation
- **Added Features**: Supports objects, classes, and inheritance
- **Significance**: Represents an advancement in database technology by integrating object-oriented concepts

## ORDBMS: Where OOP and Relational Models Meet



# ORDBMS Key Features

- Complex Structured Data-Types
- OO Methods
- Inheritance
- Polymorphism
- Encapsulation
- Database Integrity



# ORDBMS Compatibility

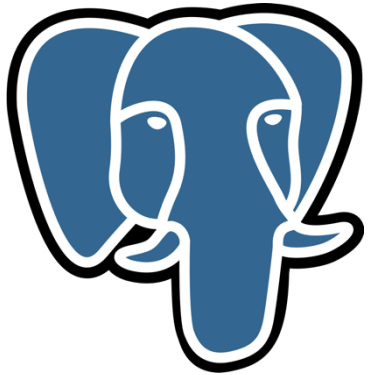
- **Upward Compatibility**: Maintains compatibility with existing SQL-based relational database languages
- **Integration with Existing Applications**: Facilitates the use of object-relational features alongside current applications without disruption
- **Support for Legacy Systems**: Allows for the enhancement of legacy relational databases, preserving previous investments

# SQL:1999

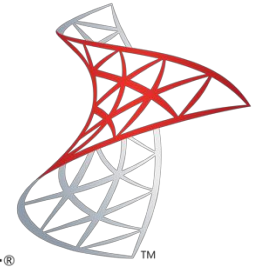
- **SQL Standards Compliance:** Adheres to SQL:1999 (SQL3) introduced several extensions
- **Relational Principles Adherence**: Upholds fundamental relational database concepts, ensuring declarative access and the enforcement of ACID properties
- **Object-Relational Features**: Object-oriented features, more complex data types, user-defined types, and support for object behaviors and hierarchies
- **Recursive Queries**: WITH clause
- **Triggers and Stored Procedures**: More sophisticated data processing and business logic
- **Advanced Querying Capabilities**: New operators and functions
- **Enhanced Data Integrity**: Assertions and referential constraints

# DMBS that support the SQL:1999 standard (OO features)

- Oracle Database
- IBM DB2
- PostgreSQL
- Microsoft SQL Server



ORACLE

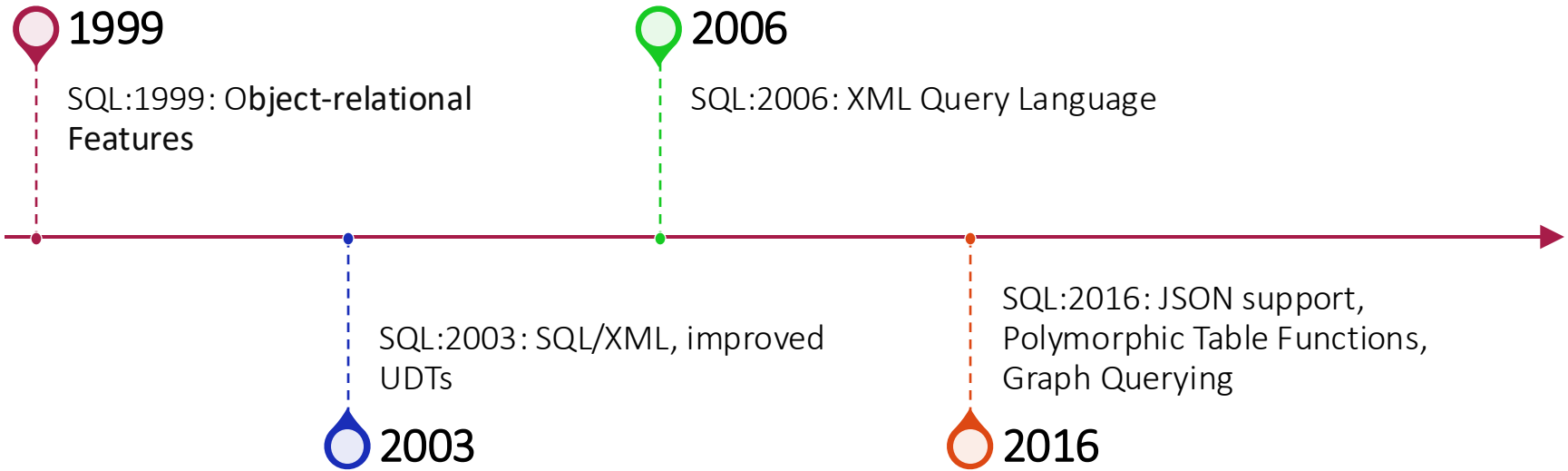


Microsoft® SQL Server®

# SQL:1999 - Selected Extensions for Complex Types

- User-Defined Types (UDTs)
- User Defined Functions (UDF)
- Inheritance
- Reference Types
- Collections
- Large Object Types (LOBs)

# ORDBMS - SQL Improvements

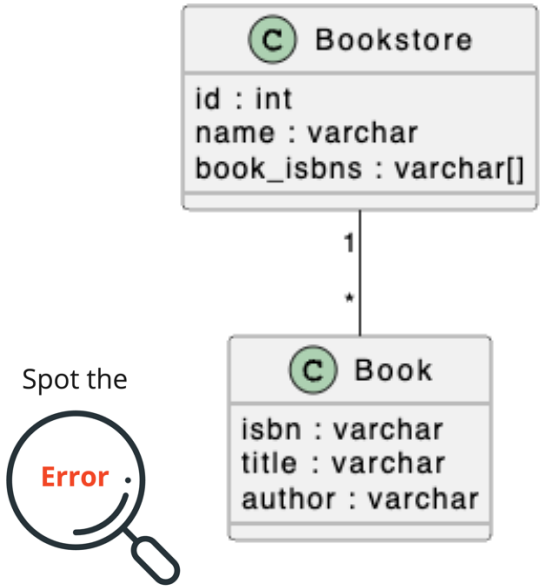


# Collection Type Example

## Collection Type

```
CREATE TABLE bookstore (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    book_isbns VARCHAR(13)[]
);
CREATE TABLE books (
    isbn VARCHAR(13) PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255)
);
```

## Class Diagram




## Select Statement

```
SELECT b.name, bk.title,
  bk.author
FROM bookstore AS b
CROSS JOIN LATERAL
  UNNEST(b.book_isbns) AS
  u(isbn)
JOIN books AS bk ON
  bk.isbn = u.isbn;
```

# Inheritance Example

## Definition

```
CREATE TABLE Item (
    item_id SERIAL PRIMARY KEY,
    title VARCHAR(255),...
);
CREATE TABLE Book (
    isbn VARCHAR(13),...
) INHERITS (Item);
CREATE TABLE Magazine (
    issue_number INT,...
) INHERITS (Item);
```

## Insertion

```
INSERT INTO Book (title, price, isbn,
author, publication_year)
VALUES
('The Great Gatsby', 14.99,
'1234567890123', 'F. Scott
Fitzgerald', 1925);
```

## Selection

Polymorphic Query:

```
SELECT * from Item;
 item_id |     title      | price
---------+----------------+-------
       1 | The Great Gatsby | 14.99
```

Type Specific Query:

```
SELECT isbn, author FROM Book
     isbn     |      author
--------------+------------------
 1234567890123 | F. Scott Fitzgerald
```

# User Type Definition (UDT) Example

## Type Definition

```
CREATE TYPE address_type AS (
    street VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    zip_code VARCHAR(20)
);
CREATE TABLE Author (
    author_id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    birth_date DATE,
    home_address address_type,
    publisher_address address_type
);
```

## Insert Data

```
INSERT INTO Author (name, birth_date,
address) VALUES
('George Orwell', '1903-06-25',
ROW('123 Main St', 'London', 'Greater
London', 'NW1'),
ROW('123 Publisher St', 'London', 'Greater
London', SW1'));
```

## Selection

```
SELECT name FROM Author WHERE
home_address.city = 'London';
```

or

```
SELECT name FROM Author WHERE
home_address.city <>
publisher_address.city;
```

# User Defined Functions (UDF) Example

## Create

```
CREATE TABLE Author (
    author_id SERIAL PRIMARY KEY,
    name VARCHAR(255)
);

CREATE TABLE Books (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(255),
    author_id INT REFERENCES
    Author(author_id)
);
```

## UDF

```
CREATE OR REPLACE FUNCTION
get_books_by_author(author_name VARCHAR)
RETURNS TEXT[] AS $$
BEGIN
RETURN (SELECT array_agg(Books.title)
FROM Books
JOIN Author ON Books.author_id = Author.author_id
WHERE Author.name = author_name);
END;
$$ LANGUAGE plpgsql;
```

## Use Function

```
SELECT get_books_by_author('George
Orwell');
```

**Result:**

```
 get_books_by_author
---------------------------------------
 {1984,"Animal Farm","Coming Up for Air"}
(1 row)
```

# Literature


- ORM: Russell, Craig. "Bridging the Object-Relational Divide: ORM technologies can simplify data access, but be aware of the challenges that come with introducing this new layer of abstraction." Queue 6.3 (2008): 18-28. Link
- 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).