DATABASE SYSTEMS

Introduction to Databases and Data Warehouses, Edition 2.0

APPENDIX 5 - OO DATABASES
APPENDIX 7 – OTHER DATABASE STRUCTURES
CHAPTER 7 - DATA WAREHOUSES
APPENDIX 10 - NO SQL
DATA MODELS

- Hierarchical (1960)
- Network (1969)
- Relational (1970)
- Entity-relationship (ER) model (1976)
- Object-Oriented (1985)
- Data warehouses (1980’s)
- Parallel databases (1990’s)
- NoSQL (2000's)
- Distributed databases (2010's)
HIERARCHICAL MODEL (1960'S)

- Based on an upside-down tree structure in which each record is called a segment.
- Designed in the 1960’s to manage large amounts of data for complex projects such as the Apollo rocket which landed on the moon in 1969.
- Depicts sets of 1:M relationships between parent and child segments.
- A website sitemap or folder structure on a computer are examples.
HIERARCHICAL MODEL (1960'S)

Examples:

- The organizational structure of a college:
NETWORK MODEL (1960'S)

- Created to be more efficient than the hierarchical model to improve database performance.
- One child entity can have more than one parent entity.
- M:N relationships can be represented
- Standard database concepts emerged:
  - The schema
  - The Data Manipulation Language
  - The Data Definition Language
NETWORK MODEL (1960'S)

Example:
RELATIONAL MODEL (1970'S)

- Introduced in 1970 by E. F. Codd of IBM.
- A major breakthrough for designers and users.
- Based on mathematical set theory.
- Represents data as independent relations.
- Development of the SQL language.
OBJECT-ORIENTED DATABASES (1980'S)

- Proposed as a way to implement complex data applications:
  - Multimedia databases
  - Geographical information systems (GIS)
  - Computer-aided design and manufacturing (CAD/CAM)

- Stored data represent objects such as images, maps, video, that don't fit well into the relational concept of rows and columns.
OBJECT-ORIENTED DATABASES (1980'S)

- More complex real-world problems
- Both data and its relationships are contained in a single structure known as an object.
- Subsequent Object-Oriented Database Management (OODM) allows objects to contain all operations that can be performed on it.
- Usually depicted using Unified Modeling Language (UML) class diagrams.
OBJECT-ORIENTED MODELS (1980'S)

Object Representation

**OBJECt Representation**

- **INVOICE**
  - INV_DATE
  - INV_NUMBER
  - INV_SHIP_DATE
  - INV_TOTAL

- **CUSTOMER**
  - INV_GENERATES
  - INV_BELONGS_TO

- **LINE**
  - I

**UML Class Diagram**

**INVOICE**

- +INV_NUMBER : Integer
- +INV_DATE : Date
- +INV_SHIP_DATE : Date
- +INV_TOTAL : Double

**CUSTOMER**

- 1..1

**LINE**

- 1..*

- +has

**ER Model**

**INVOICE**

- INV_NUMBER
- INV_DATE
- INV_SHIP_DATE
- INV_TOTAL

**CUSTOMER**

- generates

**LINE**

- has
An object in an OODB corresponds to a real-world object much like an entity in a relational database.

However, an object has attributes, operations (methods), and an object-identified (OID).

An OID is different from a primary key because it is automatically assigned by the OODB system, is considered to be immutable, and is unique across the database.
# OBJECT-ORIENTED DATABASES (1980'S)

## PRIMARY KEY (Relational Database)

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EID</strong></td>
<td><strong>CId</strong></td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td><strong>EName</strong></td>
<td><strong>CName</strong></td>
</tr>
<tr>
<td>Anne</td>
<td>Adam</td>
</tr>
<tr>
<td>Bob</td>
<td>Betty</td>
</tr>
<tr>
<td>Cliff</td>
<td>Cindy</td>
</tr>
</tbody>
</table>

## OID (Object-oriented Database)

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OID</strong></td>
<td><strong>OID</strong></td>
</tr>
<tr>
<td>100</td>
<td>201</td>
</tr>
<tr>
<td>101</td>
<td>202</td>
</tr>
<tr>
<td>102</td>
<td>203</td>
</tr>
<tr>
<td><strong>EName</strong></td>
<td><strong>CName</strong></td>
</tr>
<tr>
<td>Anne</td>
<td>Adam</td>
</tr>
<tr>
<td>Bob</td>
<td>Betty</td>
</tr>
<tr>
<td>Cliff</td>
<td>Cindy</td>
</tr>
</tbody>
</table>
Data Warehouses (1980's)

- A typical organization maintains and utilizes a number of operational data sources.
  - The operational data sources include the databases and other data repositories which are used to support the organization’s day-to-day operations
- A data warehouse is created within an organization as a separate data store whose primary purpose is data analysis.
Data Warehouses (1980's)

- Two main reasons for the creation of a data warehouse as a separate analytical database
  - The performance of operational day-to-day tasks involving data use can be severely diminished if such tasks have to compete for computing resources with analytical queries
  - It is often impossible to structure a database which can be used in an efficient manner for both operational and analytical purposes
PARALLEL DATABASES (1990'S)

- Multiple processors execute and run queries simultaneously on different slices of data of the same data set.
- Database operations can be parallelized to improve performance such as data loading, evaluating queries, etc.
Examples:

- ORDER BY clause of a query that tries to execute on millions of records can be parallelized on multiple processors and then brought together.

- If we need to join four tables, two can be joined at one processor and the other two can be joined at another processor. Final join can be done later.
BIG DATA AND NOSQL (2000'S)

- Goals of Big Data
  - Find new and better ways to manage large amounts of web and sensor-generated data
  - Provide high performance at a reasonable cost

- Characteristics of Big Data
  - Volume
  - Velocity
  - Variety
BIG DATA AND NOSQL (2000'S)

- Challenges of Big Data
  - Volume doesn’t allow usage of conventional structures
  - Expensive
  - Online Analytical Processing (OLAP) tools proved inconsistent dealing with unstructured data
BIG DATA AND NOSQL (2000'S)

- NoSQL databases (Not Only SQL)
  - Not based on the relational model
  - Support distributed database architectures
  - Provide high scalability, high availability, and fault tolerance
  - Support large amounts of sparse data
  - Geared toward performance rather than transaction consistency
  - Provides a broad umbrella for data storage and manipulation
BIG DATA AND NOSQL (2000'S)

<table>
<thead>
<tr>
<th>Relational DB</th>
<th>NoSQL DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Table (relation)</td>
<td>▪ Collection</td>
</tr>
<tr>
<td>▪ Tables contain rows having the same structure</td>
<td>▪ Collections contain documents which do not need to have the same structure</td>
</tr>
<tr>
<td>▪ Structural metadata is represented by a column</td>
<td>▪ Structural metadata is represented by a field</td>
</tr>
</tbody>
</table>
BIG DATA AND NOSQL (2000'S)

Relational DB
- Tables must be created first and then populated with data
- Must have a primary key
- Focus on how data is stored

NoSQL DB
- Metadata and data for collections can be created at the same time
- System-generated key like an OID
- Focus on how data is used
BIG DATA AND NOSQL (2000'S)

MongoDB example:
```javascript
db.createCollection("hotels")
db.hotels.insert({name: "Metro Blu",
                   address: "Chicago, IL",
                   rating: 3.5})
db.hotels.insert({name: "Experiential",
                   address: "New York, NY",
                   rating: 4})
db.hotels.insert({name: "Zazu Hotel",
                   address: "San Francisco, CA",
                   rating: 4.5})
```
To add additional information about some hotels:

db.hotels.update({ name:"Zazu Hotel" }, { $set: 
{wifi: "free"}})

db.hotels.update({ name:"Zazu Hotel" }, { $set: 
{parking: 45}})
MongoDB creates a unique, system-generated primary key (_id) for each of the documents.

Queries:

db.hotels.find() is equivalent to SELECT *
db.hotels.find({address: "Chicago, IL"})
is equivalent to
SELECT * FROM Hotels
WHERE address = 'San Francisco, CA';
DISTRIBUTED DATABASES (2010’S)

- Distributed among separate computers.
- Allows for sharing of data but also local control and autonomy.
- It mirrors the organizational structure.
- For example, a company that has a headquarters and multiple branches:
DISTRIBUTED DATABASES (2010’S)

- End users should be able to use the DDBS in the same fashion as if it were not distributed: this is called **distribution transparency**.

- Each computer in a DDBS has its own database and the data is distributed among them.

- Performance advantages:
  - Fewer queries per computer allows for faster processing.
  - Increased reliability and availability from data replication.
DISTRIBUTED DATABASES (2010’S)

- DDBS costs:
  - More complex: Additional functions are needed to maintain replicated data
  - A single query might be run on multiple machines.

- Fragmentation: the strategy for distributing the data across different locations
  - Horizontal fragmentation – all columns are stored at each location, but rows are divided.
  - Vertical fragmentation – Attributes are divided at different locations.
## DISTRIBUTED DATABASES

<table>
<thead>
<tr>
<th>EmpID</th>
<th>EmpName</th>
<th>EmpGender</th>
<th>EmpPhone</th>
<th>EmpBdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Joe</td>
<td>M</td>
<td>x-234</td>
<td>1/11/1985</td>
</tr>
<tr>
<td>0002</td>
<td>Sue</td>
<td>F</td>
<td>x-345</td>
<td>2/7/1983</td>
</tr>
<tr>
<td>0003</td>
<td>Amy</td>
<td>F</td>
<td>x-456</td>
<td>8/4/1990</td>
</tr>
<tr>
<td>0004</td>
<td>Pat</td>
<td>F</td>
<td>x-567</td>
<td>3/8/1971</td>
</tr>
<tr>
<td>0005</td>
<td>Mike</td>
<td>M</td>
<td>x-678</td>
<td>5/5/1965</td>
</tr>
</tbody>
</table>

**EMPLOYEE (Horizontally fragmented - Location A)**

<table>
<thead>
<tr>
<th>EmpID</th>
<th>EmpName</th>
<th>EmpGender</th>
<th>EmpPhone</th>
<th>EmpBdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Joe</td>
<td>M</td>
<td>x-234</td>
<td>1/11/1985</td>
</tr>
<tr>
<td>0002</td>
<td>Sue</td>
<td>F</td>
<td>x-345</td>
<td>2/7/1983</td>
</tr>
<tr>
<td>0003</td>
<td>Amy</td>
<td>F</td>
<td>x-456</td>
<td>8/4/1990</td>
</tr>
</tbody>
</table>

**EMPLOYEE (Horizontally fragmented - Location B)**

<table>
<thead>
<tr>
<th>EmpID</th>
<th>EmpName</th>
<th>EmpGender</th>
<th>EmpPhone</th>
<th>EmpBdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Joe</td>
<td>M</td>
<td>x-234</td>
<td>1/11/1985</td>
</tr>
<tr>
<td>0004</td>
<td>Pat</td>
<td>F</td>
<td>x-567</td>
<td>3/8/1971</td>
</tr>
<tr>
<td>0005</td>
<td>Mike</td>
<td>M</td>
<td>x-678</td>
<td>5/5/1965</td>
</tr>
</tbody>
</table>
### DISTRIBUTED DATABASES

#### EMPLOYEE (Not fragmented)

<table>
<thead>
<tr>
<th>EmpID</th>
<th>EmpName</th>
<th>EmpGender</th>
<th>EmpPhone</th>
<th>EmpDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Joe</td>
<td>M</td>
<td>x-234</td>
<td>11/11/1985</td>
</tr>
<tr>
<td>0002</td>
<td>Sue</td>
<td>F</td>
<td>x-345</td>
<td>2/7/1983</td>
</tr>
<tr>
<td>0003</td>
<td>Amy</td>
<td>F</td>
<td>x-456</td>
<td>8/4/1990</td>
</tr>
<tr>
<td>0004</td>
<td>Pat</td>
<td>F</td>
<td>x-567</td>
<td>3/8/1971</td>
</tr>
<tr>
<td>0005</td>
<td>Mike</td>
<td>M</td>
<td>x-678</td>
<td>5/5/1965</td>
</tr>
</tbody>
</table>

#### EMPLOYEE (Vertically fragmented - Location A)

<table>
<thead>
<tr>
<th>EmpID</th>
<th>EmpName</th>
<th>EmpPhone</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Joe</td>
<td>x-234</td>
</tr>
<tr>
<td>0002</td>
<td>Sue</td>
<td>x-345</td>
</tr>
<tr>
<td>0003</td>
<td>Amy</td>
<td>x-456</td>
</tr>
<tr>
<td>0004</td>
<td>Pat</td>
<td>x-567</td>
</tr>
<tr>
<td>0005</td>
<td>Mike</td>
<td>x-678</td>
</tr>
</tbody>
</table>

#### EMPLOYEE Vertically fragmented - Location B)

<table>
<thead>
<tr>
<th>EmpID</th>
<th>EmpGender</th>
<th>EmpBdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>M</td>
<td>1/11/1985</td>
</tr>
<tr>
<td>0002</td>
<td>F</td>
<td>2/7/1983</td>
</tr>
<tr>
<td>0003</td>
<td>F</td>
<td>8/4/1990</td>
</tr>
<tr>
<td>0004</td>
<td>F</td>
<td>3/8/1971</td>
</tr>
<tr>
<td>0005</td>
<td>M</td>
<td>5/5/1965</td>
</tr>
</tbody>
</table>
SUMMARY

- Databases have evolved in recent decades.
- The relational model is still the best model for data that is uniformly structured and not massive - as most operational databases tend to be.
- The NoSQL model is appropriate for massive amounts of data with structure that is not necessarily uniform.