

Study Unit 6

**Development in Relational
Database System**

Learning Outcomes

By the end of this unit, you should be able to:

1. Describe the components of a database system
2. Differentiate between vertical and horizontal scalability
3. Discuss CAP theorem and differentiate between ACID and BASE transaction models
4. Outline how ORM is applied in applications
5. Employ XML data type in databases
6. Summarize the big data and NoSQL movements and the responses of relational databases to these challenges

Overview

In this study unit, we look at the development in relational database systems in response to new data requirements.

First, we describe the four components of a database system, namely, the database, the database management system, the database application and the users. Understanding these components is crucial to understanding the strength and limitation of relational databases.

For example, the differences in the way that data is partitioned in relational databases and NoSQL databases give rise to different transaction models, the two characteristics they can achieve with regards to CAP theorem, and whether vertical and horizontal scalability are possible.

Next, we will discuss the object-relation impedance mismatch. This mismatch arises as data in tables and data in an object-oriented programming are conceptually different. A data access layer must map classes to tables. We also look at how applications can use ORM (Object-Relation Mapping) software to persist application data.

Finally we look at features that have been added to relational databases, namely, the XML and JSON data types, the capability to link to databases with unstructured data and to perform big data analysis.

This study unit covers parts of chapters 11 and 12 of the course text. It is estimated that the student will spend about 6 hours to read the course text chapters in conjunction with the study notes, to work out the activities, and self-assessment questions in the study notes included at suitable junctures to test understanding of the contents covered. It is advisable to use the study notes to guide the reading of the chapter in the textbook, attempt the questions, and then check the text and/or other sources for the accuracy and completeness of your answers.

Chapter 1 Relational Database System

In Study Unit 3, we saw that the database is one of the 4 components of a database system, as shown in Figure 6.1. In this chapter, we will discuss each of the components. Our discussion focuses on traditional databases. We will see how some of these components are changing in the face of new data challenges.

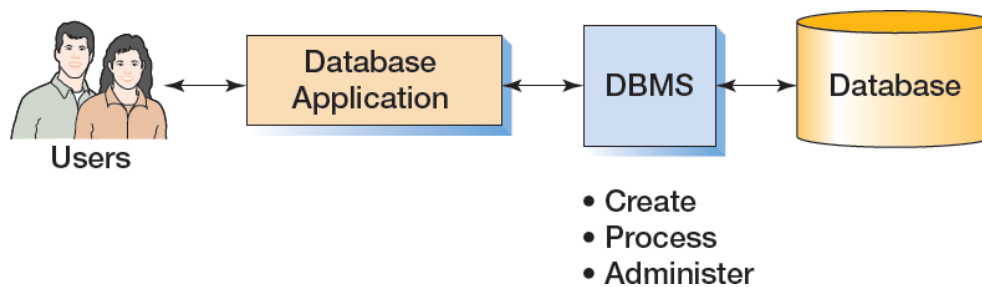


Figure 6.1 Components of a Database System

(Source: Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, Figure 1-8)



Reread

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 43-49.

1.1 Database

A database is a collection of data files and other structures such as index files. The data files store the data records of tables in a relational database. The data records are stored either unsorted or sorted, and indexes can be used to locate them.

If data records are sorted, they can be sorted according to the primary key of the table or according to some clustered index. There can be only one clustered index at most, per table as the data records can be sorted in one order only.

There can be many unclustered indexes to point to data records to speed up search operations. However, the data records are not sorted according to unclustered indexes. This means that if there are many records returned, several disk i/o operations are required to read those records.

Each data record contains fields corresponding to the columns defined in a table when the table is created. The data types and constraints defined for the fields in the data records are the same as those for the columns in the `create table` statement. The physical schema, that is, the implemented database, determines what data can be stored as well as how it is stored, according to the `create table` statements.

To determine the physical scheme, one typically starts from the data requirements, gathered from user interviews, required reports and data entry forms for the application, as well as from the transaction processing that is required on the data. Entities and the relationships between data are then identified, documented in Entity-Relationship diagram (ERD) and verified.

The ERD captures the known data and processing requirements at the time data requirements are gathered. Once the ERD is finalised, it is transformed into a logical schema or a relational model, which is then implemented on a DBMS as a physical schema. Thus, the physical schema can be implemented only after the ERD is finalised.

Once the physical schema is implemented, the columns and their constraints determine how data must be structured to be recorded in a database. Each piece of data must go into a specific column in a table, and must satisfy the data constraints of the column it goes into. Rows are related according to the foreign key constraints. Thus, a relational database is a schema-based database.

Schema-based databases have many advantages such as they provide support for transaction processing such as concurrency control and recovery as well as data security, covered in study units 3 and 4.

Schema-based databases, however, have two major weaknesses. First, when data requirements are fast-changing, growing or unclear, it is not possible to finalise an ERD to derive a physical schema. It is also difficult to make changes to the physical

schema quickly enough to fit new data requirements. During data gathering, it can be expected that certain data requirements are not foreseeable. One such example is the need to harness the great variety of internet data as user profiles, blogs, tweets, clicks, etc. in the advent of big data. Furthermore, the storage architecture of relational database has so far been to reside data of a whole table on one or a fixed number of disks, thus, limiting how data that can be processed.

Second, applications exchange data over the internet in XML and JSON format. The data that is being exchanged is varied in format and the pieces of data cannot fit into the physical schema of DBMS. Thus, data in XML and JSON format are stored as character string and processed using string operations, either in query or in functions and stored procedures.

To address these two weaknesses, many relational DBMSs such as Oracle and MS SQL Server have started to support data in JSON and XML format, as discussed in chapter 2. We will also look how the features of NoSQL are changing relational DBMSs in chapter 2.



Activity 1

[Reproduced from Question 11.1 of the course text.](#)

Describe why the data environment is complicated.

1.2 Database Management System

A DBMS is a piece of software that creates and manages databases. It is made up of many components, e.g., query parser, optimizer, buffer manager, disk space manager, transaction manager, lock manager and recovery manager. These components help ensure that transactions are ACID, that is, atomic, consistent, isolated and durable, as covered in study unit 4.

A relational DBMS is usually run on a single machine called the database server. The database server may have one or more processors with the DBMS configured to access more than one disk. A database server can serve users distributed across the internet, and users access the database server through an application server. Such database server achieves vertical scalability by having better processors, memory and storage devices.

A distributed DBMS runs on several machines or nodes, and has both local and distributed data management components. There are many configurations for how the machines share data resources, and the distributed data management components provide system-wide concurrency control and database recovery. Transactions in a distributed DBMS system is still ACID, requiring more complex concurrency control mechanisms such as three-phase commit. Having ACID transactions is an important consideration for business transactions.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 592-594.



Activity 2

Reproduced from Question 12.40, based on 12.39, both from the course text.

If more than one computer can update a replicated database, then:

- There may be inconsistent updates, and
- One computer may delete a record that another computer is updating, and
- Changes may be made that violate uniqueness constraints.

What solution is used to prevent the problems?

The CAP theorem or Brewer's theorem states that a distributed system can achieve at most two of these characteristics:

- Consistency

Every node always reads the latest value.

- Availability

A node that has not failed must service every request at some point in time.

- Partition tolerance

The system must continue to function even if the network has been partitioned due to delay or failure.

Distributed relational DBMS sacrifice availability to achieve consistency. When there is a delay or network failure, some nodes should not service requests to avoid an inconsistent read. Non-distributed relational DBMS offers consistency and availability, as partition tolerance is not applicable.

NoSQL databases sacrifice consistency to offer availability and partition tolerance. Some NoSQL databases use the BASE (Basically Available, Soft state, Eventual consistency) model. When a database provides eventual consistency, it means that when a transaction commits, its update will eventually be seen in all its replication. Before the data eventually becomes consistent in all its replication, no other transaction is allowed to update that data. Some NoSQL databases are moving from BASE to ACID, as ACID transaction is sometimes a non-negotiable in certain business transactions.

1.3 Database application

Recall that for security reasons, applications that use a database do not run on the machine that hosts the DBMS. As mentioned in Section 1.2, application server run on separate machines and make request to the database server. The application server must make a request for a database connection so that SQL statements to the DBMS can be issued.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 498-503.

1.3.1 ODBC

Open Database Connectivity (ODBC) is a set of APIs (application programmer interface) that applications use to communicate with a DBMS to access a database.

One such example is pyodbc which allows Python programs to connect to a DBMS such as SQL Server. Refer to Figure 6.2 for an example code to access a database.

In this example, a connection to a database server and to a database for a particular user is first obtained. A cursor is next requested to allow the Python program to issue SQL statements. A `select` statement on the Employee table is first requested. The user should have been granted the select permission on the Employee table for the execution to return a resultset. The resultset is then printed.

Next, the cursor object is used to issue an execute command on a stored procedure. The output of the stored procedure is printed.

When changes are made to the database, the program must issue a commit command on the cursor. If there is not further SQL statement to issue, the cursor and the connection are closed.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 503-512.



Activity 3

Reproduced from Question 11.4 of the course text.

Name the components of the ODBC standard.

```
import pyodbc

server = 'SomeServer'
database = 'dbname'
username = 'username'
password = 'password'
cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER='+server+';DATABASE='+
database+';UID='+username+';PWD='+ password)

cursor = cnxn.cursor()
cursor.execute('SELECT * FROM dbname.dbo.Employee')
for col in cursor:
    print(col[0], col[1], col[2], 'No manager' if col[3] is None else col[3])

sql = """\
DECLARE @out int;
EXEC [dbo].[countEmployeeUnder] @mgrID = ?, @count = @out OUTPUT;
SELECT @out AS the_output;
"""
mgr = 2
params = (mgr, )
cursor.execute(sql, params)
rows = cursor.fetchall()
for r in rows: print(r)
cursor.nextset()
cursor.nextset()
the_output = cursor.fetchval()
print(f'There {"are" if the_output > 1 else "is"} {the_output} \
employee{"s" if the_output > 1 else ""} under manager {mgr}')
cursor.commit()
cursor.close()
cnxn.close()
```

Figure 6.2 Using pyodbc

1.3.2 ORM

Newer programming languages such as Python are object-oriented. Programs written in them operate on data as if they are objects that provide service when requested. However, there is no notion of object-orientedness in a relational DBMS as data are simply stored in tables with no capability to service requests. As such, there is a conceptual mismatch between a class definition and a table definition.

A Python program needs to create objects when data are extracted from database tables rows, so that the data can be manipulated as objects. For data in an object to be recorded in the database, the data in the object must be mapped into the corresponding tables, and SQL `INSERT` statements are issued. This mapping is necessary to overcome object-relation impedance mismatch.

The object-relation impedance mismatch requires object-table mapping which can be done in several ways:

- Data access objects (DAO)

Data access objects are created from classes specifically for data access. These classes form the data access layer for object mapping and persistence. These classes have CRUD operations for the various tables in a database.

- Object-relation mapping (ORM)

ORM is a piece of software that provides CRUD operations on tables to applications. An application that uses ORM first defines classes that map to tables.

- DAO and ORM

Although ORM can eliminate the need for a data access layer, it is good design practice to always separate out the different concerns of an application. It is recommended that persistence logic that uses ORM be placed in a data access layer. Data access classes defined in a data access layer can utilize ORM.

One example of an ORM software is `sqlalchemy`. Refer to Figure 6.3 for an example of a Python class `Employee` defined in a module, `mapClasses`. The `Employee` table is mapped to the `Employee` class.



Activity 4

Reproduced from Question 12.43 of the course text.

Explain the meaning of the term object persistence



Activity 5

Explain the object-relation impedance mismatch.

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, String, Float, ForeignKey
Base = declarative_base()
class Employee(Base):
    __tablename__ = 'Employee'
    eno = Column(Integer, primary_key=True, autoincrement=False)
    ename = Column(String)
    salary = Column(Float)
    manager = Column(String, ForeignKey ('Employee.eno'))

    def __init__(self, eno, ename, salary, manager):
        self.eno = eno
        self.ename = ename
        self.salary = salary
        self.manager = manager

    def __repr__(self):
        return f'Employee Number: {self.eno:3} Name:{self.ename:8} Salary:
        ${self.salary:6} Manager: {self.manager if self.manager else "No Manager"}'
```

Figure 6.3 ORM: Mapping class to table

Figure 6.4 shows a Python program that connects to the database server using ODBC, pyodbc and the ORM, sqlalchemy.

```
from sqlalchemy import create_engine, text
from sqlalchemy.orm import sessionmaker
from myPackage.mapClasses import Employee
import urllib

params = urllib.parse.quote_plus("DRIVER={SQL Server Native Client 11.0};"
                                "SERVER=aServer;"
                                "DATABASE=aDatabase;"
                                "UID=aUserId;"
                                "PWD=aPassword")

engine = create_engine("mssql+pyodbc:///odbc_connect={}".format(params))
Session = sessionmaker(bind=engine, autoflush=False)
session = Session()

for emp in session.query(Employee).all():
    print(emp)
print()

# find and delete
upEmp = session.query(Employee).filter_by(eno=20).first()
if upEmp is not None:
    print('Found', upEmp)
    session.delete(upEmp)
else:
    print('Not found')

# update employee salary to 100 if salary is less than 20
listEmp = session.query(Employee).filter(text('salary<20')).all()
for emp in listEmp:
    emp.salary = 100.0

for emp in session.query(Employee).all():
    print(emp)
print()
# insert
e = Employee(20, 'Janet', 760, None)
session.add(e)

e.salary = 500.0
for emp in
    session.query(Employee).filter(text("salary<900")).order_by(text('salary
desc')).all():
    print(emp)
print()

session.commit()
session.close()
```

Figure 6.4 ORM: Application using ORM and mapping class

A session once created, can be used to make query on the Employee class which maps to the Employee table. The method `all()` returns all the rows in the Employee table as a list of Employee objects. The Employee objects are printed and formatted according to the method `__repr__` () in the Employee class.

The method `filter()` implements the `where` clause and the method `order_by()` implement the `order by` clause of `select` statement.

The objects returned from executing a query can be processed using the methods of their classes. Furthermore, their instance variables that correspond to data in table column, can also be accessed. To update a row in the table, the program updates the object. To insert or delete a row on a table, the corresponding `add` or `delete` method is invoked on the session.

For the operations to be persisted on the database, the Python program must issue a commit request on the session.

Note also that the user that the session is for must have already been given the necessary permissions on the table to query, update, insert or delete.

Another framework for handling the object-relation impedance mismatch is the Microsoft .NET Framework and ADO.NET.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 512-522.

1.4 Users

Users of the database applications access the database through the interface of the applications. To successfully run the program, users will need the appropriate permissions for the SQL statements that the program issues through the session and/or through the cursor.

Chapter 2 Emerging Trends in Relational Database System

Many features have been added to relational DBMSs to respond to data format for data exchange between applications becoming standardised, and to the requirements for big data analysis. We look at some of these features.

2.1 XML Support

Traditionally, relational DBMSs simply store XML data simply as character strings. Refer to Figure 6.5 for an example of XML data for customer food orders.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 552-555.



Activity 6

Reproduced from Question 11.87 of the course text.

Explain the phrase “standardized but customizable.”

```

<orders>
<order orderId = "001">
<customer>
  <givenname>Mary</givenname>
  <familyname>Lee</familyname>
</customer>
<address>12 Tampines Drive</address>
<food>
  <name>Apple Pie</name>
  <quantity>3</quantity>
</food>
<food>
  <name>Cheese Burger</name>
  <quantity>5</quantity>
</food>
</order>
<order orderId = "002">
<customer>
  <givenname>John</givenname>
  <familyname>Tan</familyname>
</customer>
<address>27 Simei Avenue 2</address>
<food>
  <name>chichen Burger</name>
  <quantity>2</quantity>
</food>
</order>
</orders>

```

Figure 6.5 XML data for customer food orders

As there is no support for XML data processing, user functions and/or stored procedures are written to perform the required processing. Furthermore, relational database does not provide any means to create index on the XML elements.

Currently, a number of relational DBMSs provide the XML data type with functions to process XML data, and to index the XML elements. For example, SQL Server 2016 has the data type, xml for variables and table columns. Indexes can be created on selected paths to specific elements in the xml document. In addition, an XSD schema can be associated with an xml document so that the structure of the document can be verified.

A subset of the XQuery language is implemented in Transact-SQL. One such method is `query()`. The method, when supplied with an XPATH expression, can access the elements and their values. Another method `value()` allows us to access the individual value for each element.

For example, the following code fragment was run in SQL Server 2014 to extract specific XML elements for the food order data in Figure 6.5.

```
declare @myDoc xml
set @myDoc = '<orders>
<order orderId = "001">
<customer>
  <givenname>Mary</givenname>
  <familyname>Lee</familyname>
</customer>
<address>12 Tampines Drive</address>
<food>
  <name>Apple Pie</name>
  <quantity>3</quantity>
</food>
<food>
  <name>Cheese Burger</name>
  <quantity>5</quantity>
</food>
</order>
<order orderId = "002">
<customer>
  <givenname>John</givenname>
  <familyname>Tan</familyname>
</customer>
<address>27 Simei Avenue 2</address>
<food>
  <name>chichen Burger</name>
  <quantity>2</quantity>
</food>
</order>
</orders>
'

SELECT @myDoc.query('/orders/order/address') [Delivery Addresses]

SELECT o.value('@orderId','int') [order number]
, o.value('customer[1]/familyname[1]/.' , 'varchar(50)') [Surname]
, o.value('customer[1]/givenname[1]/.' , 'varchar(50)') [Name]
, o.value('address[1]', 'varchar(50)') [address]
FROM @myDoc.nodes('/orders/order') AS aTable(o);
```

Output:

Delivery Addresses			
<address>12 Tampines Drive</address><address>27 Simei Avenue 2</address>			

order number	Surname	Name	address
1	Lee	Mary	12 Tampines Drive
2	Tan	John	27 Simei Avenue 2

Elements can be also updated. The many features for XML data added to relational database thus allow XML data to be stored and processed in tables, and XML text to be generated from table for data exchange.

2.2 JSON Support

JSON is the current popular format for data exchange and it looks to overtake XML. JSON is both human and machine readable. Refer to Figure 6.6 for an example of the same XML data in Figure 6.5, in JSON format.

```
{
  "orders": {
    "order": [
      {
        "orderId": "001",
        "customer": {
          "givenname": "Mary",
          "familyname": "Lee"
        },
        "address": "12 Tampines Drive",
        "food": [
          {
            "name": "Apple Pie",
            "quantity": 3
          },
          {
            "name": "Cheese Burger",
            "quantity": 5
          }
        ]
      },
      {
        "orderId": "002",
        "customer": {
          "givenname": "John",
          "familyname": "Tan"
        },
        "address": "27 Simei Avenue 2",
        "food": [
          {
            "name": "Chicken Burger",
            "quantity": 2
          }
        ]
      }
    ]
  }
}
```

Figure 6.6 JSON data for passenger booking

Many relational databases are adding capabilities to handle data in JSON format. For example, SQL Server 2019 now provides methods for JSON text similar to those for XML, such as JSON_VALUE to extract the value for a key.

Value in a JSON text can be updated using the method JSON_MODIFY, and column data in tables can be combined to form a JSON text using methods similar to those for XML. In addition, a JSON text can be parsed and fed into a table as rows and columns.

2.3 NoSQL Features

NoSQL stands for Not Only SQL or No SQL. The NoSQL movement was started in the late 1990s to provide new schema-less or schema-flexible data storage and processing frameworks. Many NoSQL databases represent data in JSON format as documents, stored in nodes that independently process them, thus achieving horizontal scalability.

There are currently four main categories of data models: key-value pair, document (JSON-like), columns and graphs. An example of a NOSQL database is MongoDB. MongoDB is document-based.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 598-601.



Activity 7

Reproduced from [Question 12.52](#) of the course text.


What is the NoSQL movement? What are the four categories of NoSQL databases used in the course text?



Activity 8

Reproduced from Question 12.55 of the course text.

As illustrated in Figure 12-33 of the course text, what is column family database storage, and how are column family database storage systems organized? How do structured storage systems compare to RDBMS systems?

 **Figure 12-33**
A Generalized Structured
Storage System

Name: LastName
Value: Able
Timestamp: 40324081235

(a) A Column

Super Column Name:	CustomerName	
Super Column Values:	Name: FirstName	Name: LastName
	Value: Ralph	Value: Able
	Timestamp: 40324081235	Timestamp: 40324081235

(b) A Super Column

Column Family Name:	Customer				
RowKey001	Name: FirstName	Name: LastName			
	Value: Ralph	Value: Able			
	Timestamp: 40324081235	Timestamp: 40324081235			
RowKey002	Name: FirstName	Name: LastName	Name: Phone	Name: City	
	Value: Nancy	Value: Jacobs	Value: 817-871-8123	Value: Fort Worth	
	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055	
RowKey003	Name: LastName	Name: EmailAddress			
	Value: Baker	Value: Susan.Baker@elswhere.com			
	Timestamp: 40340103518	Timestamp: 40340103518			

(c) A Column Family

The main features of NoSQL are speed, horizontal scaling, and availability. The data in a NoSQL database is partitioned horizontally into shards and distributed to various nodes.

While the strength of relational databases is that their transactions are ACID, NoSQL adopts BASE (Basically Available, Soft State, Eventual consistency) for speed and availability. It is difficult to incorporate ACID transactions in the current NoSQL framework.

Relational databases have started to incorporate many NoSQL features. For example, SQL Server 2017 has added a feature, Polybase, to manipulate data in NoSQL data models. This feature is further extended in SQL Server 2017 so that other types of databases can be linked, for example, Oracle and MongoDB.

Another movement, NewSQL is started to develop relational databases that combine the best features of relational database and NoSQL databases.

2.4 Big Data Cluster

Big data analytics is about analysing massive amount of complex data of different data quality, and that are generated at high frequency. Big data analytics is now a necessity in the business world. To facilitate big data analysis, new framework for storage and processing are required.

One such framework is Apache Hadoop which uses a distributed file system (HDFS). In Hadoop, each file is broken into block, and each file block is duplicated twice. Each copy resides in three different nodes. This replication facilitates data recovery. In addition, multiple copies imply that several nodes may be chosen to process the blocks at different times, based on the workload.

Data analysis uses a MapReduce algorithm. One copy of each block to be analysed undergoes a Map phase to produce intermediate result which are then shuffled to other nodes for Reduce phase. In the Reduce phase, the intermediate result are processed, and finally collected to a final output. MapReduce algorithms can be implemented in various programming languages such as Python and Java.

A newer and faster framework is Apache Spark which processes data cached in memory, and uses stream processing against Apache Hadoop which is batch processing. Apache Spark provides a library of algorithms but it does not have a distributed file system. Therefore, Apache Spark is oftentimes run on Apache Hadoop.

SQL Server 2019 extended its Polybase feature to store data in Apache Hadoop and can make query to access file blocks in HDFS. In addition, part of the query computation can be pushed to nodes in Hadoop. SQL Server runs both Apache Spark and Apache Hadoop.



Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 601-602.



Activity 9

Reproduced from Question 12.50 of the course text.

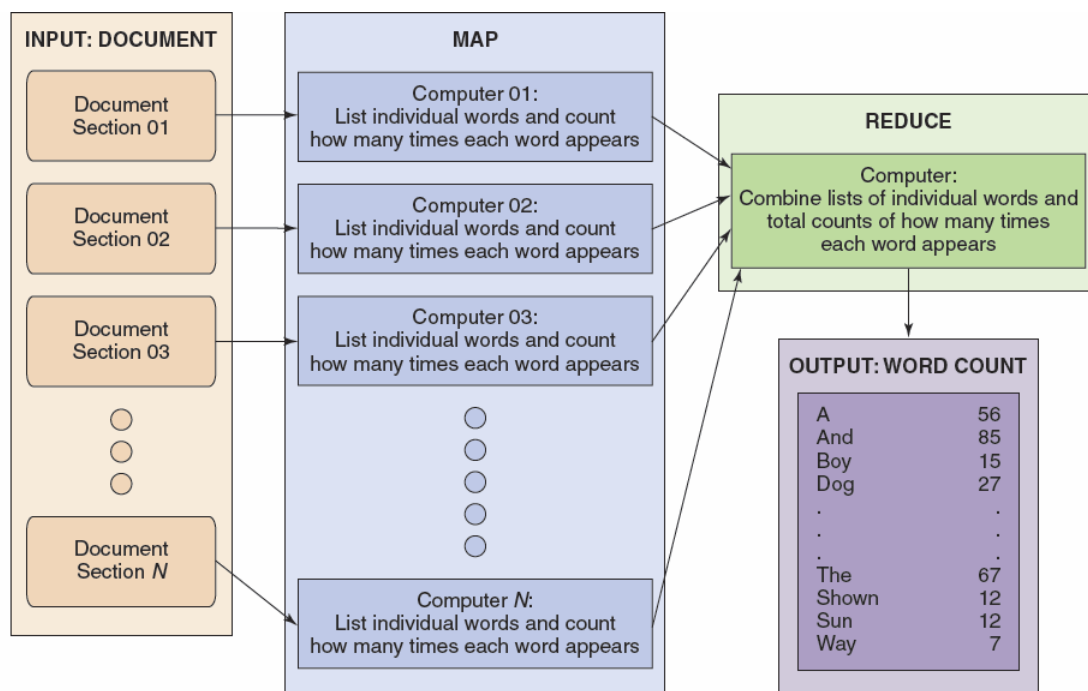
What is Big Data?



Activity 10

Reproduced from Question 12.56 of the course text.

Explain MapReduce processing using Figure 12.34 of the course text.



 **Figure 12-34**
MapReduce

Summary

A relational database is a collection of data files and other structures. Whole tables reside on one or more disks. The physical schema determines how and what data are stored in a database. Thus, a database implements vertical scalability, and has a rigid or fixed schema.

A distributed DBMS runs on several machines or nodes, and has both local and distributed data management components. The CAP theorem or Brewer's theorem states that a distributed system can achieve at most two of these characteristics: consistency, availability and partition tolerance. Relational database sacrifice availability for ACID transactions as consistency is a non-negotiable in certain business transactions.

Object-oriented programming languages such as Python operate on data as if they are objects with services. However, in a relational DBMS, data have no capability to service requests. As such, there is a conceptual mismatch between a class definition and a table definition. The object-relation impedance mismatch requires object-table mapping. Object-table mapping can be done in several ways: Data access objects (DAO), Object-relation mapping (ORM) or DAO with ORM.

To overcome the rigidity of fixed schema and to cater data formats for data exchange, support for XML and JSON data types is now available in many relational databases. XQuery language using XPATH expression can access XML elements and their values.

The NoSQL movement was started to provide new schema-less or schema-flexible data storage and processing frameworks. However, transactions in NoSQL database are not ACID, and at best, are BASE (Basically Available, Soft State, Eventual consistency). Many relational databases are beginning to provide facilities to connect to NoSQL databases to handle their unstructured data. Another movement, NewSQL is developing relational databases to combine the best features of relational database and NoSQL databases.

To facilitate big data analysis, new frameworks for storage and processing such as Apache Hadoop are developed. Apache Hadoop uses a distributed file system (HDFS) and a MapReduce algorithm to analyse data through horizontal partitioning. Apache Spark processes data cached in memory, and uses stream processing. but it does not have a distributed file system.

References

Book

Author(s)	Year	Book Title	Edition	Publisher
Kroenke, D., & Auer, D. J.	2016	<i>Database Processing – Fundamentals, Design, and Implementation</i>	14	Pearson

Quiz

1. Which component performs the function of locating the employee record with employee id 00001 on disk?
 - a. database application
 - b. database
 - c. users
 - *d. database management system

2. Which statement is false?
 - a. ODBC stands for Open Database Connectivity.
 - *b. ODBC has not had practical success, but has shown great potential for future development.
 - c. ODBC works with table-like data sources such as relational databases and spreadsheets.
 - d. With ODBC, a database and the DBMS that processes it are identified by the data source.

3. Which statement is true?
 - a. In the 1990s the database community and the document processing community created a standard called Expandable Markup Language (XML).
 - b. XML provides a standardized, non-customizable way to describe the content of a document.
 - c. XML documents can be generated only from database data with xml data type.
 - *d. XML text can be extracted using XQuery language and XPath.

4. Which statement is false?
 - a. The movement that uses different database methods than the relational model and/or SQL is called the NoSQL movement.
 - b. Most of NoSQL nonrelational database methodologies are known as structured storage.

c. NoSQL can also mean "No SQL."

*d. Structured storage column families are indistinguishable from relational database tables

5. When distributed databases create copies of the database on different servers, this is known as _____.

*a. replication

b. partitioning

c. disbursing

d. distributed two-phase locking

Formative Assessment

1. Which component performs the function of making an SQL request on a database server?

a. database

Incorrect. This is not a function of the database. Refer to Study Unit 6, Section 1.1.

b. database management system

Incorrect. This is not a function of the database management system. Refer to Study Unit 6, Section 1.2.

c. database application

Correct! This is a function of the database application. Refer to Study Unit 6, Section 1.3.

d. user through an interactive session

Incorrect. Users should not make SQL request on a database server through an interactive session. Refer to Study Unit 6, Section 1.3 and Section 1.4.

2. What code must be written by an application programmer using ORM software?

a. code to extract the values in instance variables so that SQL insert statement can be issued to persist the object

Incorrect. ORM already provide code for this function. Refer to Study Unit 6, Section 1.3.2.

b. code to create an object with the values from an SQL select statement

Incorrect. ORM already provide code for this function. Refer to Study Unit 6, Section 1.3.2.

c. code to map a class with its corresponding table

Correct! ORM already provide code for this function. Refer to Study Unit 6, Section 1.3.2.

d. code to map a class with its corresponding table

Incorrect. Application programmers must write the code. Refer to Study Unit 6, Section 1.3.2.

3. Which is not an added feature to the relational database?

a. adding data types for XML data and JSON documents

Incorrect. This is an added feature to the relational database. Refer to Study Unit 6, Section 2.1 and Section 2.2.

b. adding facilities to connect with various types of databases.

Incorrect. This is an added feature to the relational database. Refer to Study Unit 6, Section 2.3.

c. adding new data architecture for big data analysis.

Incorrect. This is a type of distribution channel arrangement. This is an added feature to the relational database. Refer to Study Unit 6, Section 2.4.

d. adding ACID transactions

Correct! This is not an added feature. Rather ACID transactions is foundational to relational databases.

4. Which is not a data representation for NoSQL ?

a. key-value

Incorrect. This is a type of data representation for NoSQL. Refer to textbook page 598.

b. document

Incorrect. This is a type of data representation for NoSQL. Refer to textbook page 598.

c. column family

Incorrect. This is a type of data representation for NoSQL. Refer to textbook page 598.

d. None of the above

Correct! These are data representation for NoSQL. Refer to textbook page 598.

5. In the MapReduce process, the first step is the _____ step.

a. shuffle

Incorrect. This is the second step of the map phase. Refer to Study Unit 6, Section 2.4.

b. reduce

Incorrect. This is the first step of the reduce phase. Refer to Study Unit 6, Section 2.4.

c. map

Correct! This is the first step of the map phase as well as the first step of the MapReduce process. Refer to Study Unit 6, Section 2.4.

d. collect

Incorrect. This is the second step of the reduce phase. Refer to Study Unit 6, Section 2.4.

Solutions or Suggested Answers

Activity 1

Internet technology applications need to publish database applications that involve dozens of different data types; this includes many different types of DBMS data plus nondatabase data. Some data is table-like, but does not conform to the definition of a relation.

Activity 2

If more than one computer can update a replicated database, then the associated problems can be prevented by using distributed two-phase locking.

Activity 3

Application program, driver manager, DBMS drivers, and data source

Activity 4

Object persistence means storing the values of the properties of an object.

Activity 5

Programs in object-oriented programming language operate on data as if they are objects written that provide service when requested. However, there is no notion of object-orientedness in a relational DBMS as data are simply stored in tables with no capability to service requests. Thus, there is a conceptual mismatch between a class definition and a table definition. This mismatch is called the object-relation impedance mismatch.

Activity 6

Standardized means always using the same techniques the same way. Customizable means changeable to fit any situation. As such, XML can be used to describe any database view, but in a standardized way.

Activity 7

The NoSQL movement is now usually referred to as the Not only SQL movement. It is the movement from Big Data databases to the use of non-relational DBMSs, often known as structured storage. A NoSQL DBMS is typically a distributed, replicated database used where this type of a DBMS is needed to support large datasets. For example, both Facebook and Twitter use the Apache Software Foundation's Cassandra database. The four categories of NoSQL databases are key-value, document, column family, and graph DBMSs.

Activity 8

A generalized structured storage system (in this case, a column family database) is shown in Figure 12-33. The column family equivalent of a relational DBMS (RDBMS) table has a very different construction. Although similar terms are used, they do not mean the same thing that they mean in a relational DBMS. The smallest unit of storage is called a column, but it is really the equivalent of an RDBMS table cell (the intersection of an RDBMS row and column). A column consists of three elements: the column name, the column value or datum, and a timestamp to record when the value was stored in the column. This is shown in Figure 12-33(a) by the LastName column, which stores the LastName value Able.

Columns can be grouped into sets referred to as super columns. This is shown in Figure 12-33(b) by the CustomerName super column, which consists of a FirstName column and a LastName column, and which stores the CustomerName value Ralph Able. Columns and super columns are grouped to create column families, which are the structured storage equivalent of RDBMS tables. In a column family we have rows of grouped columns, and each row has RowKey, which is similar to the primary key used in an RDBMS table. However, unlike an RDBMS table, a row in a column family does not have to have the same number of columns as another row in the same column family. This is illustrated in Figure 12-33(c) by the Customer column family, which consists of three rows of data on customers.

Figure 12-33(c) clearly illustrates the difference between structured storage column families and RDBMS tables; column families can have variable columns and data stored in each row in a way that is impossible in an RDBMS table. This storage column structure is definitely not in 1NF as defined in Chapter 3, let alone BCNF!

For example, note that the first row has no Phone or City columns, while the third row not only has no FirstName, Phone, or City columns, but also contains an EmailAddress column that does not exist in the other rows. Finally, all the column families are contained in a keyspace, which provides the set of RowKey values that can be used in the data store. RowKey values from the keyspace are shown being used in Figure 8-20(c) to identify each row in a column family. Although this structure may seem odd at first, in practice it allows for great flexibility because columns to contain new data may be introduced at any time without modifying an existing table structure.

Activity 9

Big Data is the current term for the enormous datasets generated by Web applications such as search tools (for example, Google and Bing) and Web 2.0 social networks (for example, Facebook, LinkedIn, and Twitter). Although these new and very visible Web applications are highlighting the problems of dealing with large datasets, these problems were already present in other areas, such as scientific research and business operations.

Activity 10

The MapReduce process is used to break a large analytical task into smaller tasks, assign each smaller task to a separate computer in the cluster, and then gather the results of each of those tasks and combine them into the final product of the original tasks. The term Map refers to the work done on each individual computer, and the term Reduces refers to the combining of the individual results into the final result.

A commonly used example of the MapReduce process is counting how many times each word is used in a document. This is illustrated in Figure 12-34, where we can see how the original document is broken into sections, and then each section is passed to a separate computer in the cluster for processing by the Map process. The output from each of the Map processes is then passed to one computer, which uses the Reduce process to combine the results from each Map process into the final output, which is the list of the words in the document and how many times each word appears in the document.