Topic 10: Database Architectures;Embedding SQL in Program Code

ICT285 Databases
Dr Danny Toohey

About this topic

- In today’s topic we'll explore some further concepts to do with database implementation, including distributed database architectures, and database programming using embedded SQL.

Topic learning outcomes

- **After completing this topic you should be able to:**
- Explain the drivers of the development of DDBMSs
- Distinguish between distributed processing and distributed databases
- Describe the functions and architecture of a DDBMS
- Describe and give examples of the additional factors involved in distributed database design: partitioning (fragmentation), allocation and replication
- Describe the transparency goals of a DDBMS
- Explain how distribution affects concurrency and how the two-phase commit protocol attempts to address this
- Briefly describe the components of replication services and some replication models /

Topic learning outcomes /cont'd

- /
- Explain why there is a 'mismatch' between SQL and procedural programming languages
- Describe how cursors and variables are used to embed SQL into program code
- Explain the difference between user-defined functions, triggers, and stored procedures, and appropriate uses
- Give an example of a user-defined function
- Be able to create simple triggers in Oracle PL/SQL
- Be able to create simple stored procedures in Oracle PL/SQL
- Explain the advantages of using stored procedures for database applications

Resources for this topic

READING

- Text, Chapter 12 pp 592-594 "Distributed Database Processing"
- Text, Chapter 7 pp 379-386 "Embedding SQL in Program Code"

Other resources:

- Oracle DBA Guide: Distributed Database Concepts https://docs.oracle.com/en/database/oracle/oracle-database/12.2/admin/distributed-database-concepts.html
- Replication publication models in SQL Server: https://docs.microsoft.com/en-us/sql/relational-databases/replication/publish/replication-publishing-model-overview?view=sql-server-2017

And other links in slides

Lab 10 – Oracle PL/SQL

- In today's lab we look at how we can use Oracle's procedural language PL/SQL (Procedural Language/SQL) to embed SQL in program code. This allows us much more flexibility and power in interacting with the database.
- The three types of persistent stored modules that can be created using PL/SQL are *functions*, *triggers*, and *procedures*. We'll look at all of them in today's lab.

Topic Outline

Part 1:

- 1. Introduction
- 2. Distributed database design
- 3. Transparency goals of a DDBMS
- 4. Replication
- **Part 2:**
- 5. Embedding SQL in program code

Introduction

- Why distribute?
- Advantages and challenges
- Types of distributed DBMS
- Functions of a DDBMS

Database distribution

- So far in the unit, we have concentrated mainly on issues associated with single-user, or multi-user stand alone database systems
- In this topic we look at distributed databases – databases that are *stored and processed* on more than one computer

Database distribution

- In a **centralised database**, has a central site where all processing is completed but the database can be accessed in many terminals
- In **distributed processing**, the application logic can be completed at different sites – e.g. various clients in client server system. However the database itself is still at one central site
- In a **distributed database**, parts of the database are physically distributed between different sites

Why distribute?

Organisations tend to be distributed

- Logically, into divisions, departments etc
- Physically such as geographical location

Distributed databases can mirror this distribution and so allow for:

- More Data sharing
- Increased data availability
- Brings data stored closer to where it is most used

Definitions

Distributed Database

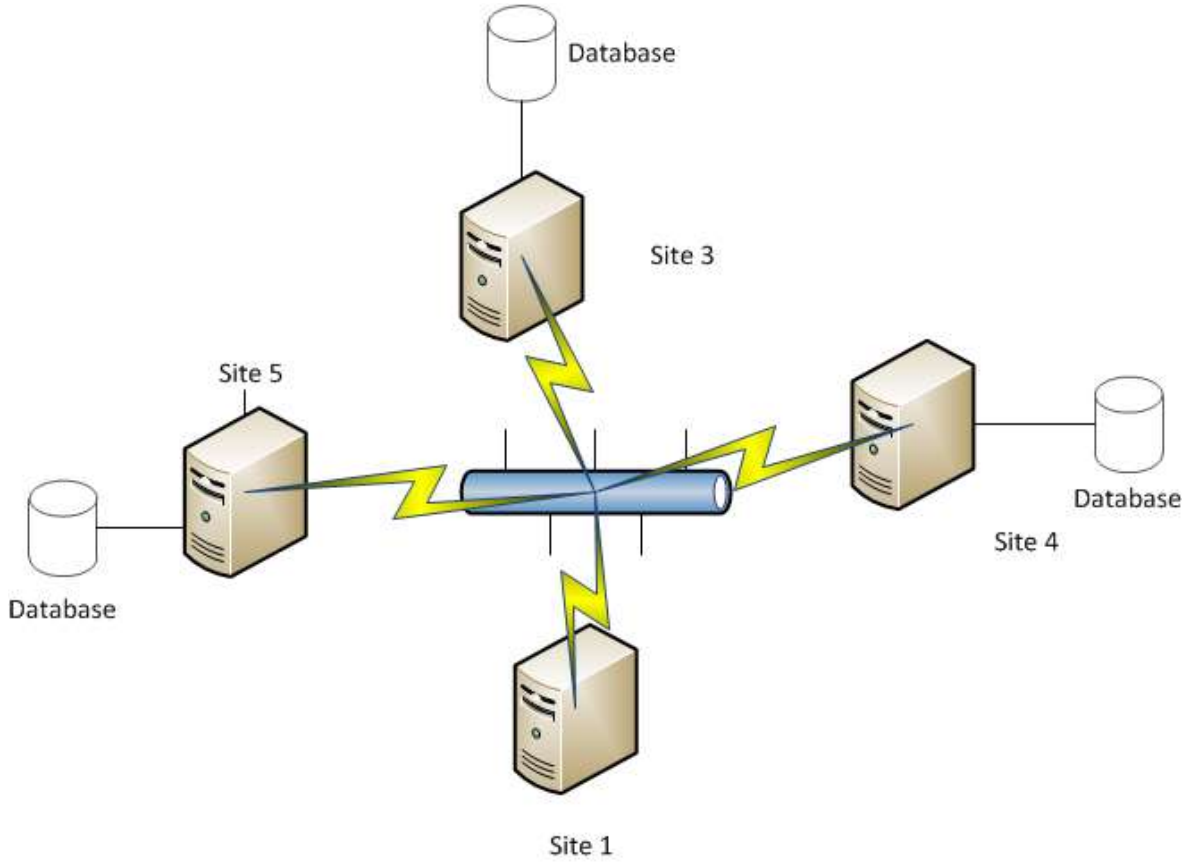
- A logically interrelated collection of shared data (and a description of this data), physically distributed over the network

Distributed DBMS (DDBMS)

- Software system that lets you manage the distributed database and makes that transparent to users

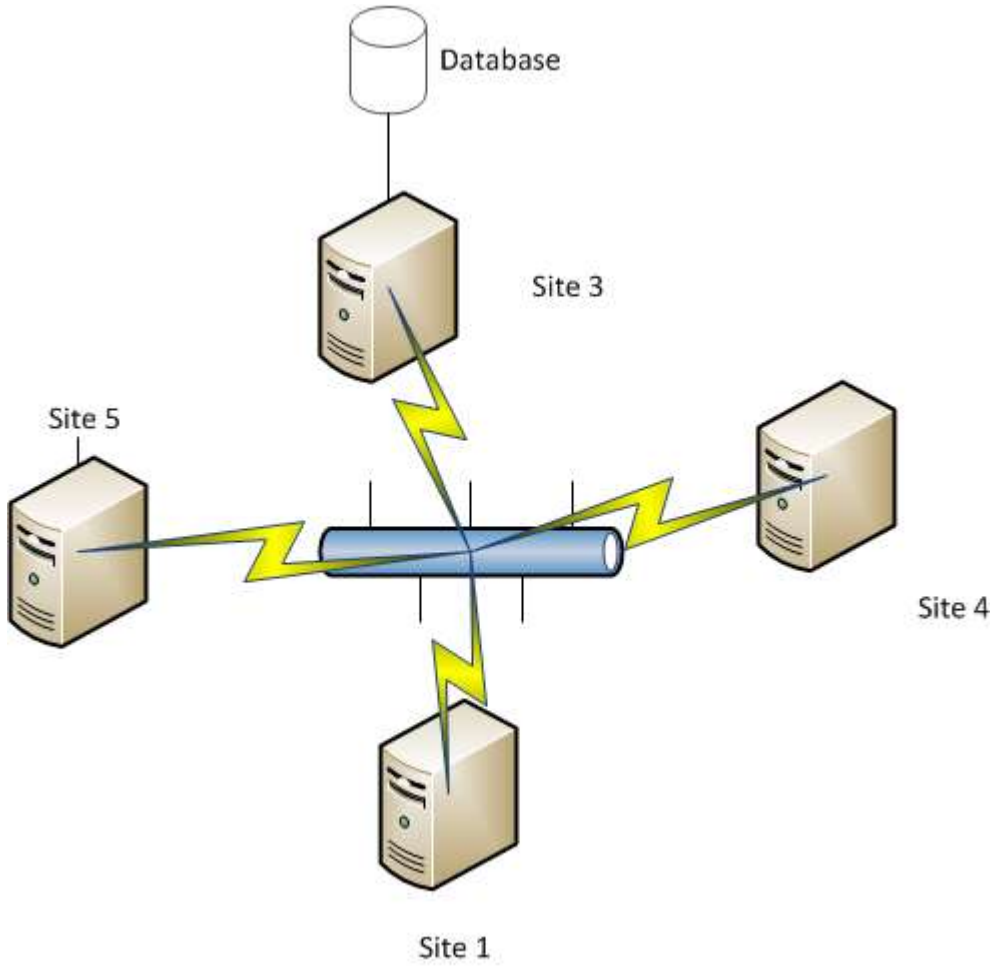
Distributed DBMS

- Database are at various local sites
- Network allows for communication
- Client-server model, and each node can act as both client and server



Contrast DDBMS with Distributed Processing

Single database that can be accessed using a network

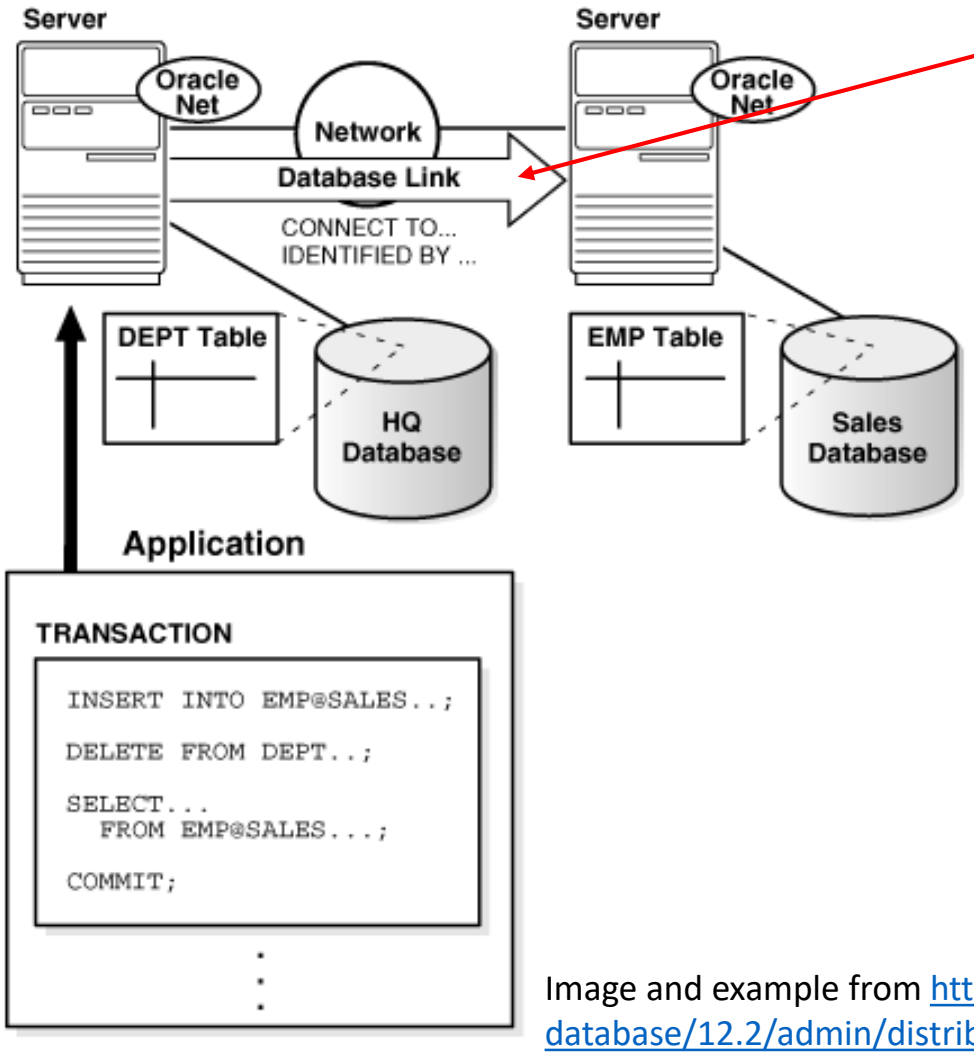


What makes a DDBMS?

- Group of logically-related shared data
- Data are split into partitions (fragments)
    - Fragments may be duplicatred
    - Fragments/replicas given to sites
- Sites joined by a communications network
    - Data at each site is under control of a local DBMS

- Local DBMSs deals with local applications autonomously
    - Each DBMS participates in at least one global application

An Oracle distributed database system



A link defines a one-way communication path from one Oracle database server to another

Links allow local users to access remote databases

Image and example from https://docs.oracle.com/en/database/oracle/oracle-database/12.2/admin/distributed-database-concepts.html

Potential advantages of DDBMSs

Mirrors organisational structure

- Organisations tend to have a distributed structure

Better local autonomy and sharability

- Users at one site can access data at another
- Data can be placed where it is needed giving local users control over data

•(more …)

Potential advantages of DDBMSs cont'd…

Better availability

- If a single node goes down the system shouldn't become inoperable; DDBMS should still function

Better reliability

- Copies of data at various sites results in data being available if a node fails

Improved performance

- Data close to site where it is used the most
- Therefore, reduced contention for CPU

(more …)

Potential advantages of DDBMSs cont'd...

Economics

- Cheaper to build a system of smaller computers than a one large computer
- Decreased network costs because data is stored close to where it is used
- More cost effective to add nodes

Integration

- DDBMS can be utilised to integrate legacy systems

Modular growth

- Easier to add new nodes/increase size of database

Challenges of DDBMSs

Increased Complexity

- Inherently more complex
- Replication – may have to handle updates to duplicated data

Database design more complex

- Where and how are the data to be located
- Which applications can access which nodes

Security

- Replicated data located in multiple locations
- Network needs to be secure

Disadvantages of DDBMS cont'd

Integrity control more difficult

- Consistency across multiple locations

Lack of standards

- Not really any standards in this area

Lack of experience

- Not as high a level of experience in industry in DDBMS as with DBMS

Cost

- Procurement
- Maintenance
- Communications

Types of DDBMS

One distinction is between homogenous and heterogeneous:

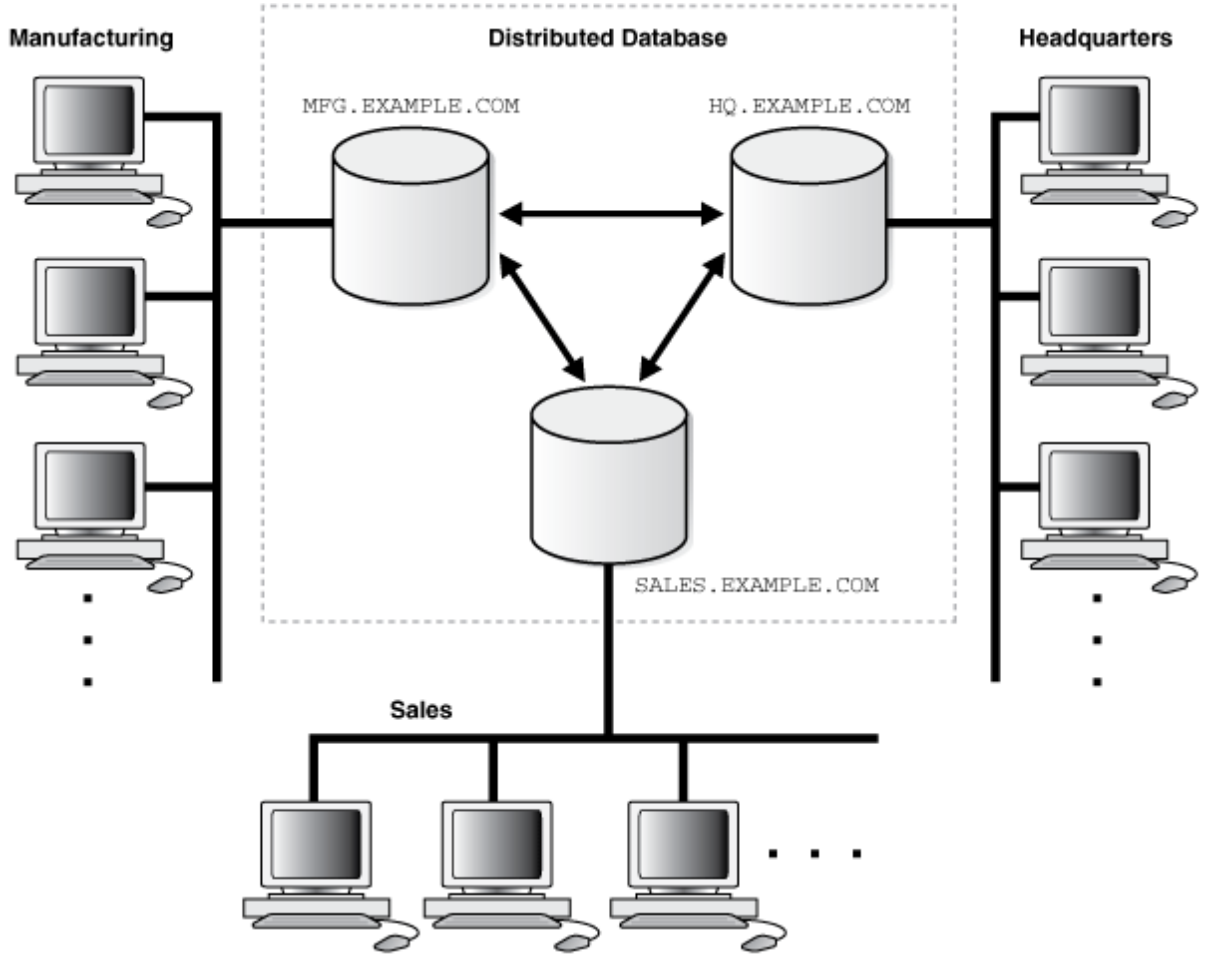
Homogeneous

- All sites use same DBMS product (e.g. Oracle)
- Much easier to design and manage
- Approach provides incremental growth and allows increased performance

Heterogeneous

- At least one site uses a different product

Image and example from https://docs.oracle.com/en/database/oracle/oracle-database/12.2/admin/distributed-database-concepts.html



Three databases in a **homogeneous distributed database system**. Each database is connected to various client systems at three different divisions: Sales, Headquarters, Manufacturing.

Heterogeneous DDBMS

- Sites may run different DBMS products, with possibly different underlying data models
- Occurs when sites have implemented their own databases and integration is considered later
- Translations required to allow for:
  - Different hardware
  - Different DBMS products
  - Different hardware and different DBMS products.
- Typical solution is to use **gateways**
  - Convert language and model of each different DBMS into the language and model of the relational system
  - DBMS-specific (e.g. Oracle's Sybase gateway; or generic connectivity such as ODBC or OLE DB)

Types of DDBMS: Autonomy

Can vary from full local autonomy to full global control

- **Full global control** will occur when there is an homogenous DDBMS which has been designed top-down
- **Federated** DBMSs are where local DBMSs operate independently but participate in a federation to make their local data available (semi-autonomous)
- **Multi database systems** (MDBSs) are where the local DBMSs have full autonomy. For this to work, there needs to be an additional layer of software

Functions of a DBMS - revision

Codd (1982) listed 8 services that should be provided by any full-scale DBMS:

- Data storage, retrieval, and update
- User accessible catalogue
- Transaction support
- Concurrency control services
- Recovery services
- Authorisation services
- Support for data communications
- Integrity services

Functions of a DDBMS

We expect DDBMS to have at least the functionality of a DBMS

Also to have the following functionality:

- Extended communication services
- Extended data dictionary (global)
- Distributed query processing
- Extended security control
- Extended concurrency control
- Extended recovery services

Date's rules for a fully distributed database

1. Each local site can act as an independent DBMS
2. No reliance on a central site
3. The system is not affected by failure at any of its sites - continuous operation
4. Location transparency - user doesn't need to know where data is located
5. Fragmentation transparency - user doesn't need to know how the database is partitioned
6. Replication transparency - user deals with single logical database; DDBMS deals with copies of fragments
7. A query may be executed over several sites, and optimisation is done by the DDBMS
8. A transaction may update data at different sites

9-12 System is independent of hardware, operating system, network and product

Summing up…

- There are many potential advantages to a distributed database, including improved shareability and availability of data across a distributed organisation
- There are also many challenges associated with distributing databases
- There are various types of distribution model, depending on the products used and the degree of autonomy of each site
- An ideal DDBMS would function exactly as a single-site DBMS from the point of view of users and programmers

2. Distributed database design

Principles of distributed design

Types of partitioning

Murdoch UNIVERSITY

Distributed database design

There are three key issues in distributed database design:

- **Partitioning (fragmentation)**
  - Relation may be divided into a number of sub-relations, which are then distributed.
- **Allocation**
  - Each fragment is stored at site with “optimal” distribution.
- **Replication**
  - Copy of fragment may be maintained at several sites.

Data allocation strategies

Centralised

- Consists of single database and DBMS stored at one site with users distributed across the network

•Partitioned (or Fragmented)

- Database partitioned into disjoint fragments, each fragment assigned to one site

•Complete Replication

- Consists of maintaining complete copy of database at each site

•Selective Replication

- Combination of partitioning, replication, and centralization

Comparison of strategies for data allocation

	Locality of Reference	Reliability and availability	Performance	Storage Costs	Communications Costs
Centralised	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Fragmented	High (if good design)	Low for item; high for system	Satisfactory (if good design)	Lowest	Low (if good design)
Complete Replication	Highest	Highest	Best for read	Highest	High for update; low for read
Selective Replication	High (if good design)	Low for item; high for system	Satisfactory (if good design)	Average	Low (if good design)

Partitioning (fragmentation)

Definition and allocation of fragments is carried out strategically to achieve:

- Locality of reference – store data close to where it is used
- Improved reliability and availability
- Improved performance
- Balanced storage capacities and costs
- Minimised communication costs

Involves analysing most important applications/transactions (next slide)

Partitioning decisions

Information used to analyse applications and make decisions about partitioning may include:

- **frequency** with which an application is run
- **site** from which an application is run
- **performance criteria** for transactions and applications
- **transactions** that are executed by application
  - type of access (read or write)
  - predicates of read operations σ (what is in the 'where' condition)

Four reasons for partitioning:

Usage

- Applications work with views rather than entire relations. Therefore makes sense to work with subsets of relations as the unit of distribution

Efficiency

- Data is stored close to where it is most frequently used
- Data that is not needed by local applications is not stored locally

Parallelism

- With fragments as unit of distribution, transaction can be divided into several subqueries that operate on fragments

Security

- Data not required by local applications is not stored and so not available to unauthorized users

Disadvantages of partitioning

- Performance
  - the performance of global applications that require data from several fragments located at different sites may be slower
- Integrity
  - Integrity control may be more difficult if data and attributes involved in functional dependencies are fragmented and located at different sites

Types of partitioning

Four types of partitioning:

- Horizontal
- Vertical
- Mixed
- Derived

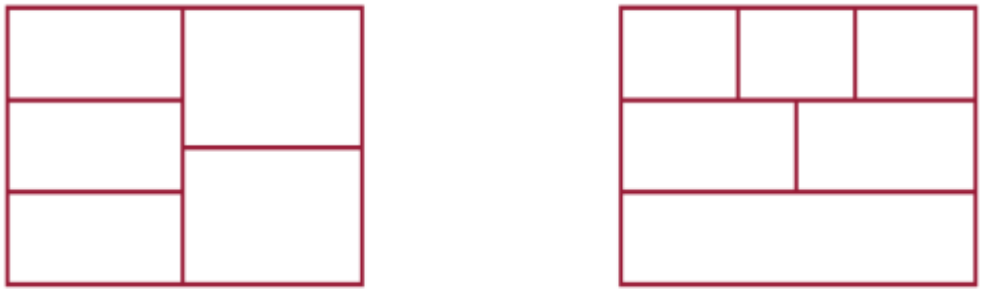
No partitioning is also a possibility:

- If relation is small and not updated frequently, may be better NOT to fragment relation

Horizontal, vertical and mixed partitions

(a)

(b)



(a)

(b)

Horizontal partition

- Consists of a subset of the tuples of a relation
- Defined using *Restrict* operation of relational algebra:

$$\sigma_p(R)$$

For example:

$$P_1 = \sigma_{\text{Nationality='French'}}(\text{ARTIST})$$

$$P_2 = \sigma_{\text{type='American'}}(\text{ARTIST})$$

Vertical partition

- Consists of a subset of *attributes* of a relation
- Defined using *Project* operation of relational algebra:

$$\Pi_{a1, \ldots, an}(R)$$

  - For example:

$$S_1 = \Pi_{\text{CustomerID, FirstName, LastName}} (\text{CUSTOMER})$$
$$S_2 = \Pi_{\text{CustomerID, Country, PostCode}} (\text{CUSTOMER})$$

- Determined by establishing *affinity* of one attribute to another

Mixed partition

- Consists of a horizontal fragment that is vertically fragmented, or a vertical fragment that is horizontally fragmented
- Defined using Restrict and Project operations of relational algebra:

$$\sigma_{p}(\Pi_{a1, \ldots, an}(R)) \quad \text{or}$$
$$\Pi_{a1, \ldots, an}(\sigma_{p}(R))$$

Example - Mixed partition

- $S_2 = \Pi$ CustomerID, Country, ZipCode(CUSTOMER)

$$S_{21} = \sigma_{\text{Country='Australia'}}(S_2)$$
$$S_{22} = \sigma_{\text{Country='Bhutan'}}(S_2)$$
$$S_{23} = \sigma_{\text{Country='Singapore'}}(S_2)$$

Correctness of partitioning

Partitioning can’t be carried out haphazardly – 3 rules must be followed

- Completeness
- Reconstruction
- Disjointness

•These rules follow from what we know about separating and recombining relations according to relational algebra

Correctness of partitioning

Completeness

- If relation R is decomposed into fragments R_1, R_2, ... R_n, each data item that can be found in R must appear in at least one fragment.

Reconstruction

- Must be possible to define a relational operation that will reconstruct R from the fragments:
  - *union* for horizontal partitioning
  - *join* for vertical partitioning

Correctness of partitioning

Disjointness

- If data item d_i appears in fragment R_i, then it should not appear in any other fragment.
  - Exception: vertical partitioning, where primary key attributes must be repeated to allow reconstruction
- For horizontal partitioning, data item is a tuple
- For vertical partitioning, data item is an attribute

Summing up…

- The database can be partitioned into fragments to be located in such a way as to improve reliability, availability and performance for most important transactions
- Partitioning can be vertical (by attributes) or horizontal (by rows), or a combination
- The database must remain correct after partitioning: rules of completeness, reconstruction, and disjointness

3. Transparency goals of a DDBMS

Distribution transparency

Transaction transparency

Performance transparency

Database transparency

Murdoch UNIVERSITY

DDBMS transparency

DDBMS transparency goals have the common property of allowing the end users to think that they are the only user of the database

- Distribution transparency
- Transaction transparency
  - Concurrency transparency
  - Failure transparency
- Performance transparency
- Database (or heterogeneity) transparency

Distribution transparency

Distribution transparency allows us to manage a physically dispersed database as though it were centralised

Three Levels of Distribution Transparency

- **Fragmentation** transparency
- **Location** transparency
- **Local mapping** transparency

TABLE 10.2 A SUMMARY OF TRANSPARENCY FEATURES

LEVEL OF DISTRIBUTION TRANSPARENCY
HIGH ← → LOW

Specify:	*Fragmentation*	*Location*	*Local Mapping*
Fragment?	No	Yes	Yes
Location?	No	No	Yes

Distribution transparency : Example

- Employee data (EMPLOYEE) are distributed over three locations: New York, Atlanta, and Miami.
  - Depending on the level of distribution transparency support, three different cases of queries are possible:

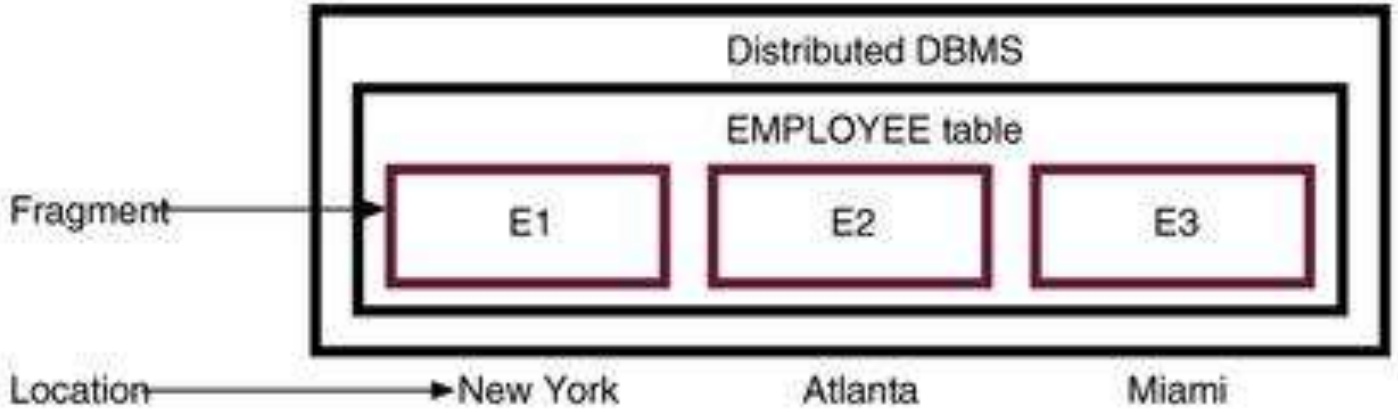
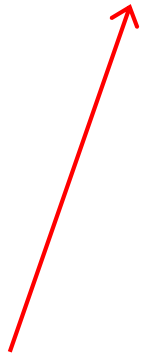


FIGURE 10.9 FRAGMENT LOCATIONS

Distribution transparency

Case 1: DB Supports Fragmentation Transparency

```
SELECT *
  FROM EMPLOYEE
  WHERE EMP_DOB < '01-JAN-1940';
```

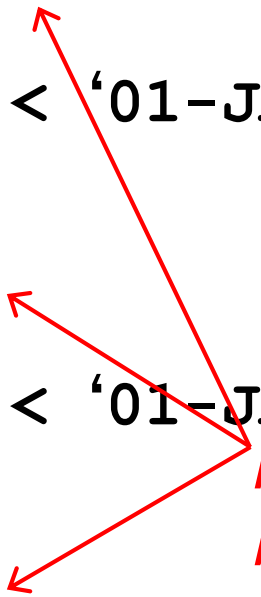


no need to reference fragment or location – just the table

Distribution transparency

Case 2: DB Supports Location Transparency

```
SELECT *
  FROM E1
  WHERE EMP_DOB < '01-JAN-1940';
UNION
SELECT *
  FROM E2
  WHERE EMP_DOB < '01-JAN-1940';
UNION
SELECT *
  FROM E3
  WHERE EMP_DOB < '01-JAN-1940';
```



need to reference fragment but not location

Distribution transparency

Case 3: DB Supports Local Mapping Transparency

```
SELECT *
  FROM E1 NODE NY
  WHERE EMP_DOB < '01-JAN-1940';
UNION
SELECT *
  FROM E2 NODE ATL
  WHERE EMP_DOB < '01-JAN-1940';
UNION
SELECT *
  FROM E3 NODE MIA
  WHERE EMP_DOB < '01-JAN-1940';
```

need to reference both fragment and location

Distribution transparency

- Distribution transparency is supported by **distributed data dictionary** (DDD) or a distributed data catalog (DDC)
- The DDC contains the description of the entire database as seen by the database administrator
- The database description, known as the **distributed global schema**, is the common database schema used locally to translate user requests into subqueries

Global database names

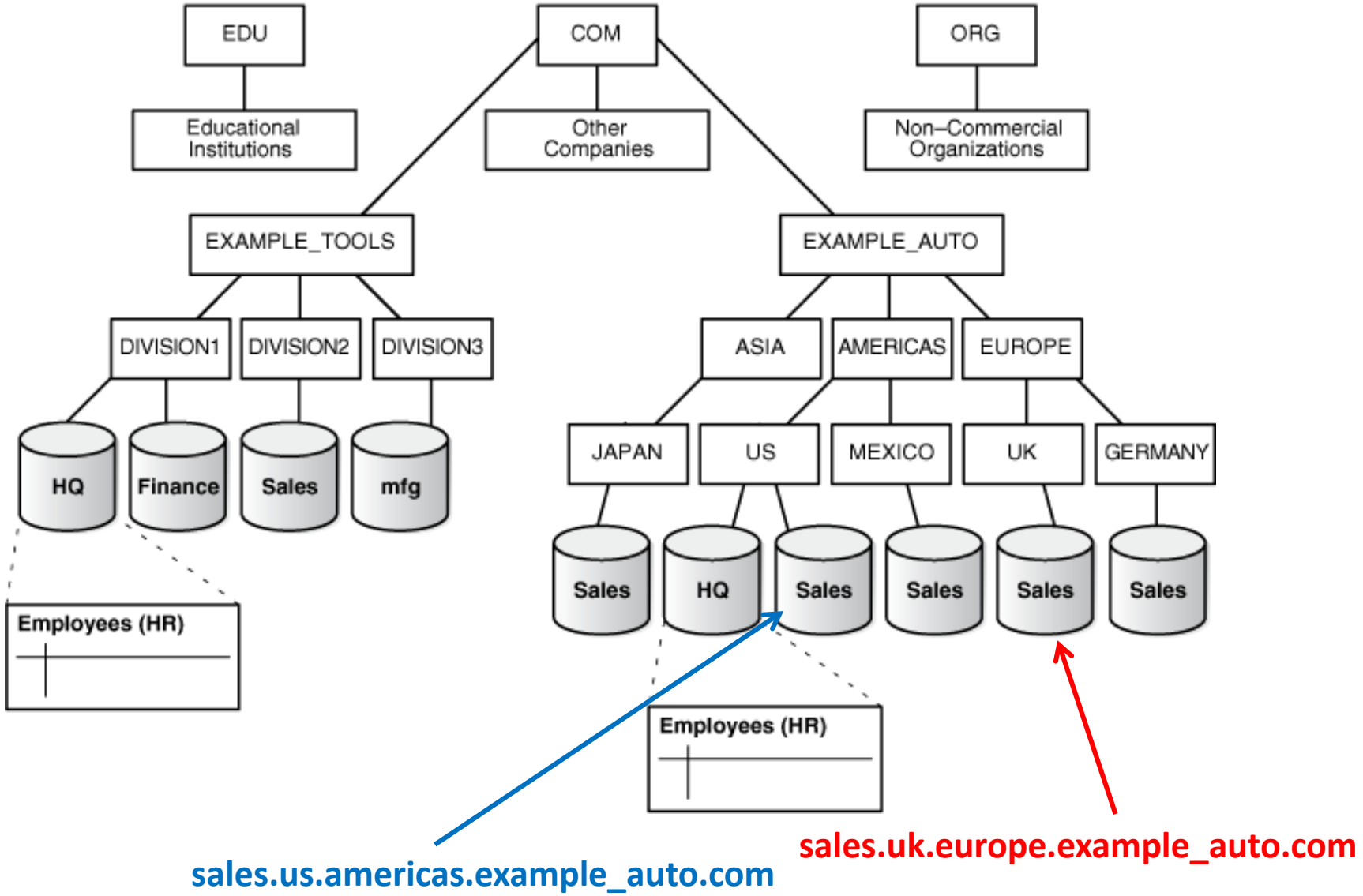


Image and example from https://docs.oracle.com/en/database/oracle/oracle-database/12.2/admin/distributed-database-concepts.html

Transaction transparency

- Transaction transparency ensures that database transactions will maintain the database's integrity and consistency
- Each transaction is divided into number of sub-transactions, one for each site that has to be accessed
- The transaction will be completed **only if all database sites involved in the transaction complete their part of the transaction**
- DDBMS must ensure the indivisibility of both the global transaction AND each of the sub-transactions

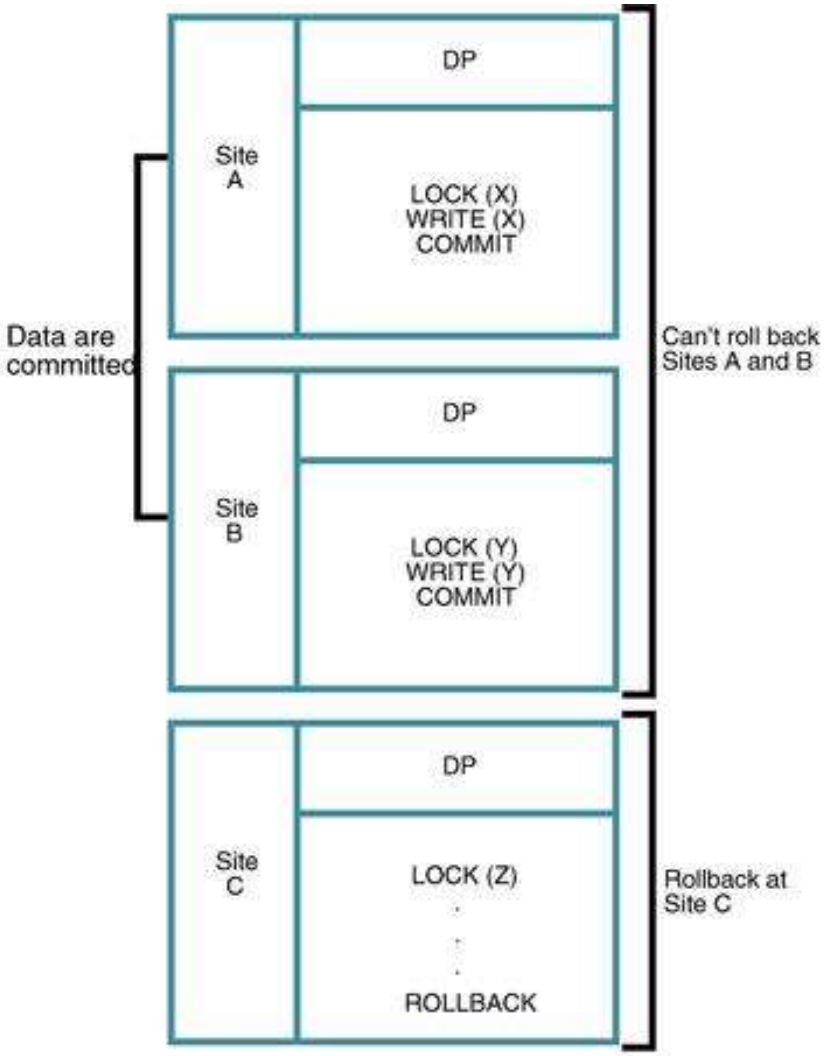


FIGURE 10.15 THE EFFECT OF A PREMATURE COMMIT

Example: transaction operations at sites A and B are committed BUT C cannot commit (for some reason).

What is the result?

Inconsistency in our data – we can't un-commit committed data!

Two-Phase Commit

- The two-phase commit protocol guarantees that, if ANY portion of a transaction operation cannot be committed, ALL changes made at the other sites participating in the transaction will be undone to maintain a consistent database state
- There are several sites involved in the transaction and one site acts as **coordinator**, the others as subordinates
- The two phases are a **voting phase** in which all the participating sites are asked by the coordinator if they are able to commit their sub-transactions, and a **decision phase** in which the coordinator instructs all sites to either commit or abort

Phase 1: Preparation

In the Preparation Phase, the following occurs:

- The coordinator sends a PREPARE TO COMMIT message to all subordinates
- The subordinates receive the message, write the transaction log using the write-ahead protocol, and send an acknowledgement message to the coordinator.
- The coordinator makes sure that ALL nodes are READY TO COMMIT, or it sends the message to abort the transaction

Phase 2: The Final Commit

If all subordinates have voted READY TO COMMIT:

- The coordinator broadcasts a GLOBAL COMMIT message to all subordinates and waits for the replies
- Each subordinate receives the GLOBAL COMMIT message and commits their transaction
- The subordinates reply with an acknowledgement to the coordinator
- The coordinator commits the whole transaction
- If one or more subordinates does not reply, the coordinator re-sends the GLOBAL COMMIT message until has received all replies

Distributed concurrency control

- Multiple-site processing, multiple site data are more likely to create inconsistencies and deadlocks than single-site processing, single site data
  - If there is no replication and all transactions are at a single site, then ordinary two-phase locking can be used
  - However if data is replicated or several sites are involved, things get more complicated
  - Variations of the non-distributed concurrency schemes can be used, including **variations to two-phase locking**

Performance transparency

- Performance transparency requires a DDBMS to perform as if it were a centralised DBMS
  - The system should not suffer any performance degradation due to the distributed architecture
- Performance transparency also requires the DDBMS to determine the most cost-effective strategy to execute a request
- Distributed Query Processor (DQP) maps data request into ordered sequence of operations on local databases. Has to decide:
  - which fragment to access
  - which copy of a fragment to use (if replicated)
  - which location to use

Distributed query optimisation

- The objective of query optimisation is to minimize the total cost associated with the execution of a request
- The costs associated with a request are a function of the:
  - Access time (I/O) cost involved in accessing the physical data stored on disk
  - CPU time cost associated with the processing overhead of managing distributed transactions
  - **Communication cost** associated with the transmission of data among nodes in distributed database systems
- Most query optimisation algorithms are based on two principles:
  - Selection of the **optimum execution order**
  - Selection of sites to be accessed to **minimize communication costs**

Example

PROPERTY (propNo, city)	10000 records in London
CLIENT (clientNo,maxPrice)	100000 records in Glasgow
VIEWING (propNo, clientNo)	1000000 records in London

```
SELECT p.propNo
FROM PROPERTY p INNER JOIN
(CLIENT c INNER JOIN VIEWING v ON c.clientNo = v.clientNo)
        ON p.propNo = v.propNo
WHERE p.city='Aberdeen' AND c.maxPrice > 200000;
```

Example cont'd

Assume:

- Each tuple in each relation is 100 characters long
- 100,000 viewings for properties in Aberdeen
- 10 renters with maximum price greater than $200,000
- 10,000 characters/second data transmission
- Computation time negligible compared to communication time

From Connolly & Begg

Example cont'd

Table 22.4 Comparison of distributed query processing strategies.

	Strategy	Time
(1)	Move Client relation to London and process query there	16.7 minutes
(2)	Move Property and Viewing relations to Glasgow and process query there	28 hours
(3)	Join Property and Viewing relations at London, select tuples for Aberdeen properties, and for each of these in turn, check at Glasgow to determine if associated maxPrice > £200,000	2.3 days
(4)	Select clients with maxPrice > £200,000 at Glasgow and for each one found, check at London for a viewing involving that client and an Aberdeen property	20 seconds
(5)	Join Property and Viewing relations at London, select Aberdeen properties, and project result over propertyNo and clientNo and move this result to Glasgow for matching with maxPrice > £200,000	16.7 minutes
(6)	Select clients with maxPrice > £200,000 at Glasgow and move the result to London for matching with Aberdeen properties	1 second

Database (or heterogeneity) transparency

- Allows for the integration of several different local DBMSs under a common (global) schema
- DDBMS becomes responsible for translating the data requests from the global schema to the local DBMS schema

Summing up…

- **Transparency** means the nature of the distribution should be invisible to the database users/applications
- *Distribution* transparency – no need to reference location or fragment in queries
- *Transaction* transparency – ACID properties still apply when distributed
    - Two-Phase Commit
- *Performance* transparency – distributed query optimiser includes distribution information
- *Database* transparency – independent of different database schemas involved

4. Replication

Replication

A purely distributed DBMS can be very complex!

- Advantages may be outweighed by complexity
- Replication offers similar services to DDBMS but at a lower level of complexity

Simply, replication is the provision of copies of data to multiple sites

- Most vendors support replication of some sort
- Replication enables users to have local access to current data

Functionality

A distributed data replication service should be capable of copying data from one database to another, synchronously or asynchronously

Additional functionality: should be able to:

- Specify replication schema
- Handle both large and small volumes of data
- Handle replication across heterogeneous DBMSs and platforms
- Replicate objects other than data, such as indexes, triggers, etc
- Enable user to specify what is to be replicated
- Provide a subscription mechanism
- Provide easy administration

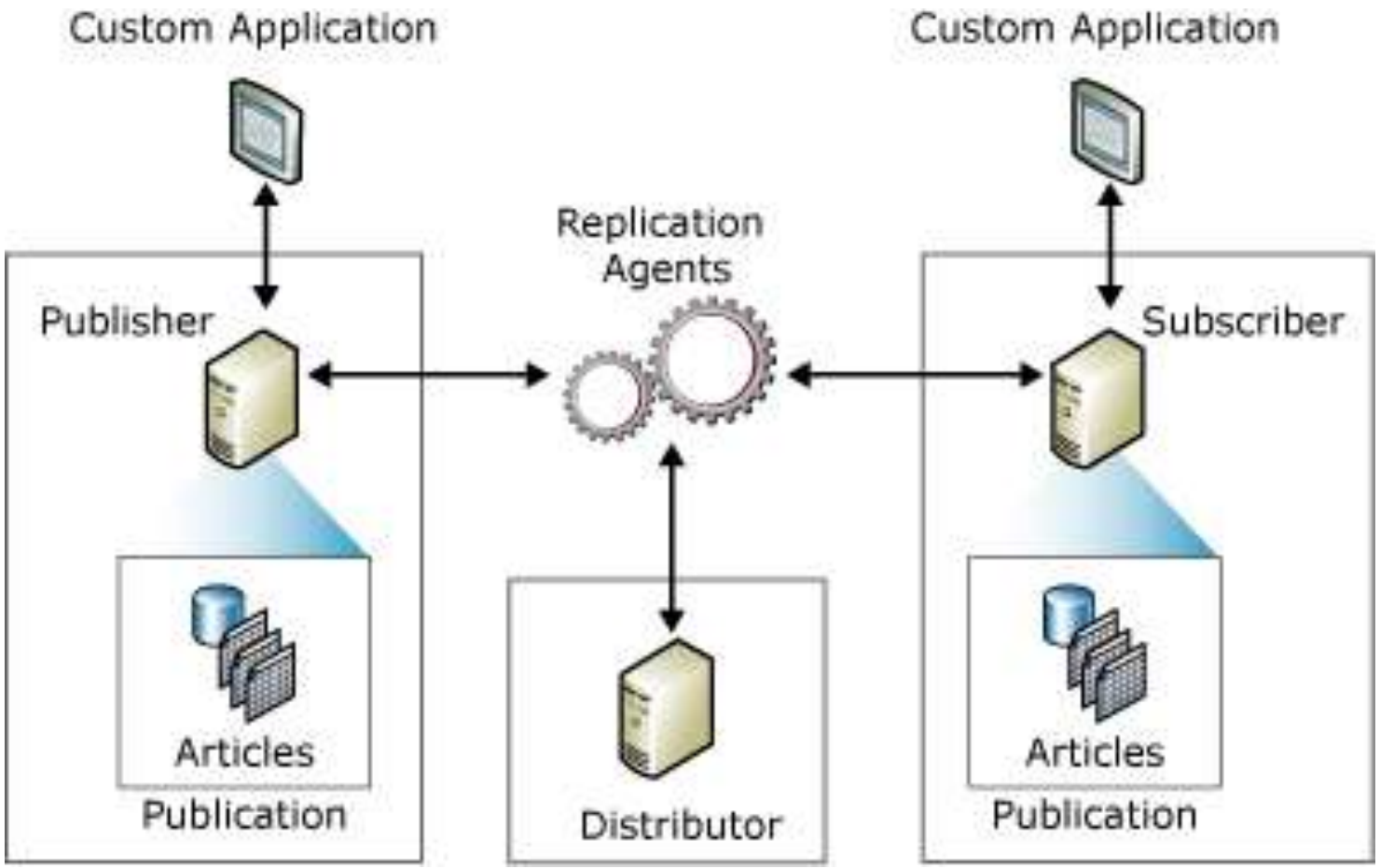
Components of replication

SQL Server uses a publishing metaphor to liken the components in a replication topology to a magazine that is published and distributed to subscribers:

- **Publisher**
  - Server that makes data available for replication
- **Distributor**
  - Servers that distribute replicated data
- **Subscriber**
  - Destination servers that receive data

https://msdn.microsoft.com/en-us/library/ms152567(v=sql.130).aspx

Components of replication



https://docs.microsoft.com/en-us/sql/relational-databases/replication/publish/replication-publishing-model-overview?view=sql-server-2017

Types of replication

Snapshot replication

- Distributes data exactly as it appears at a specific moment in time
- Use when data changes infrequently or it is OK to have copied that are slightly out of date with the ‘publisher’ site

Transactional replication

- Starts with a snapshot
- Subsequent data changes are delivered to the ‘subscriber’ in near real time to maintain transaction consistency

See https://msdn.microsoft.com/en-us/library/ms152531.aspx

Types of replication cont'd

Merge replication

- Starts with a snapshot
- Multiple subscribers make changes to their own data partitions autonomously and later synchronise with the publisher and other subscribers

See https://msdn.microsoft.com/en-us/library/ms152531.aspx

Types of subscription

- A subscription is a request for a copy of the data in a publication

Push subscription

- Publisher propagates changes to a subscriber without request
- Can be continuous, on demand or scheduled
- Use when data needs to be synchronised frequently or continuously

Pull subscription

- Subscriber requests changes from the publisher
- Subscriber determines when changes are synchronised
- Use when subscribers are autonomous or mobile

See https://msdn.microsoft.com/en-us/library/ms151170.aspx

Replication models

Replication components can be organised in various different models:

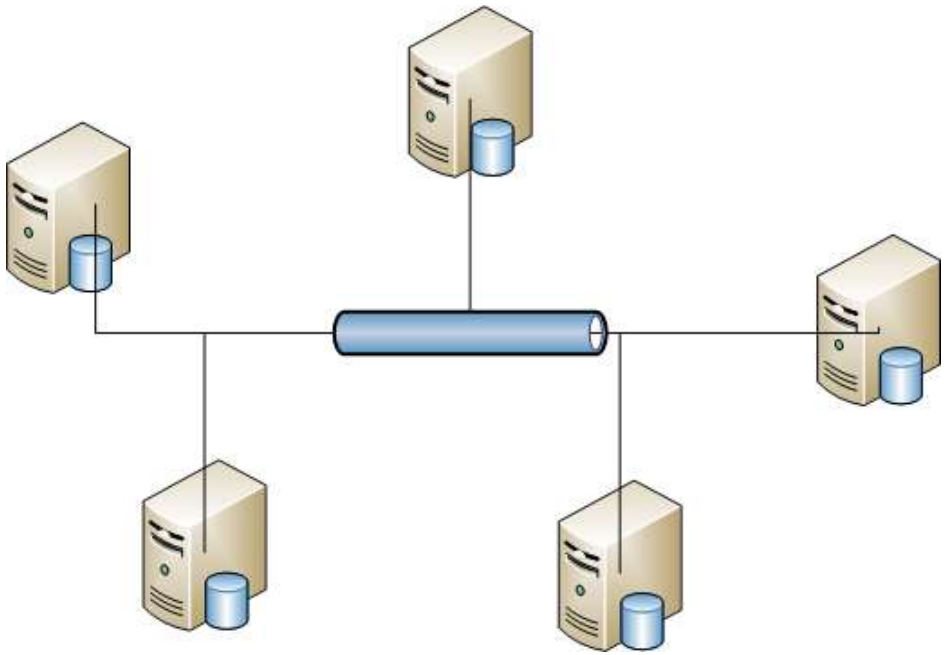
- Peer-to-Peer
- Central Publisher
- Central Publisher with remote distributor
- Central subscriber model
- Publishing subscriber model

•These models are described here for SQLServer 2008: https://technet.microsoft.com/en-us/library/dd323622.aspx

Peer-to-Peer model

Replication between identical participants

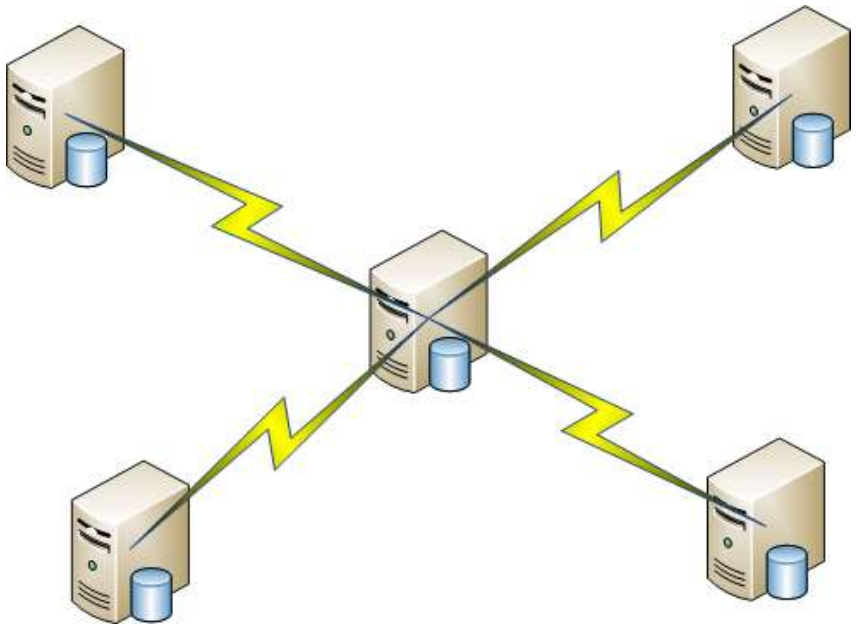
- Allows roles to move between nodes for maintenance
- More complexity involved in management



Central Publisher model

Publisher and distributor are on same server with several subscribers

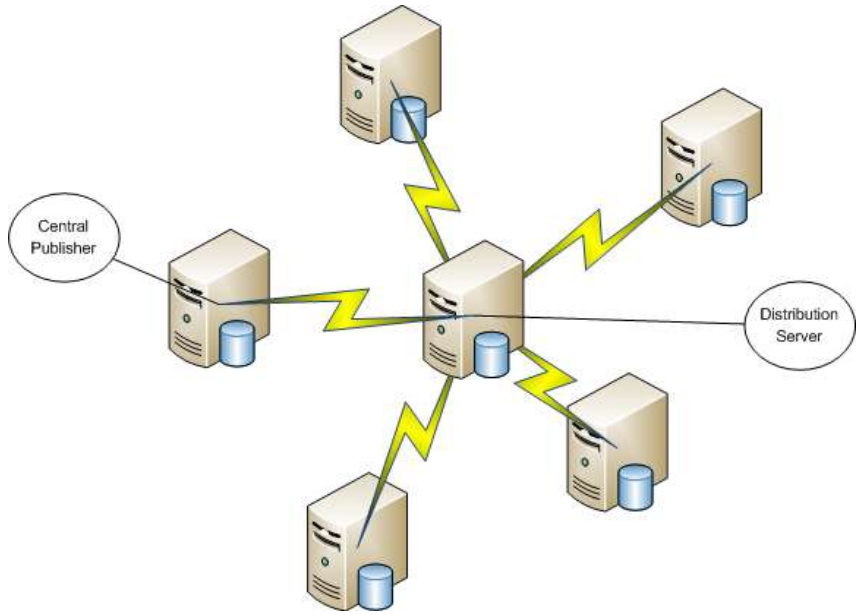
- More manageable and easier to maintain than P2P
- Single point of failure and increased workload on server
- Most commonly used option



Central Publisher with remote distributor model

Publisher and distributor are on different servers with several subscribers

- More even distribution of workload
- Need to manage additional server



Summing up…

- Replication offers many of the benefits of pure distribution with less of the complexity
- There are a number of different replication models, based on the idea of a publisher site and subscription sites
- The optimal configuration in any particular system depends on a number of factors such as frequency of updates and extent of shared data

Topic learning outcomes revisited

- **After completing this topic you should be able to:**
- Explain the drivers of the development of DDBMSs
- Distinguish between distributed processing and distributed databases
- Describe the functions and architecture of a DDBMS
- Describe and give examples of the additional factors involved in distributed database design: partitioning (fragmentation), allocation and replication
- Describe the transparency goals of a DDBMS
- Explain how distribution affects concurrency and how the two-phase commit protocol attempts to address this
- Briefly describe the components of replication services and some replication models /

5. Embedding SQL in program code: user-defined functions, triggers and stored procedures

SQL and programming languages

- You are familiar with entering SQL statements, e.g. to retrieve a set of records according to some criteria
- You are probably also familiar with programming languages such as Java or C++, where you work with program variables and control flow statements (such as loops or if-else)
- Adding this sort of functionality to the DBMS would enable much more flexibility in manipulating the data in the database!
- BUT there is a mismatch between the table-based results of SQL, and record-based programming languages that we need to deal with

SQL and programming languages

To embed SQL into program code, we need:

- A way of assigning results of SQL statements to **program variables**
  - select count(*) into **rowcount** from Artist;
    - -- the program variable *rowcount* can then be used in further processing
- A way of iterating through a result table one row at a time: this is done using a **cursor**
  - A cursor is a pointer to the first row of the result table, which can then be processed a row at a time by moving the cursor

SQL/Persistent Stored Modules

- Each DBMS product has its own variations on SQL, including features that allow it to function similarly to a procedural programming language
- The ANSI/ISO standard calls these SQL/PSM, SQL Persistent Stored Modules
  - **Persistent**: remains available for use over time
  - **Stored**: the module is stored in the database
  - **Module**: a piece of code in the form of a function, trigger or stored procedure
- The Oracle SQL version is PL/SQL (Procedural Language/SQL)

Oracle PL/SQL

SQL code can be written as one of three module types:

- User-defined functions
- Triggers
- Stored procedures

User-defined functions

- Functions are stored sets of SQL statements that can be called by other SQL statements,

  and which may have input parameters and output values
- For example, a stored function could be written to concatenate firstname and lastname, using code similar to that in a simple SELECT
- The function could then be called any time the concatenation was required (next slide)

```
CREATE OR REPLACE FUNCTION FirstNameFirst
    -- These are the input parameters
       (
        varFirstName    IN Char,
        varLastName     IN Char
       )
    -- This is the variable that will hold the returned value
    RETURN          Varchar
    IS varFullName  Varchar(60);

BEGIN

   -- SQL statements to concatenate the names in the proper order
   varFullName := (RTRIM(varFirstName)||' '||RTRIM(varLastName));
   -- Return the concatenated name
   RETURN varFullName;

END;
/
```

```
SELECT      FirstNameFirst(FirstName, LastName) AS CustomerName,
            Street, City, State, ZipPostalCode
FROM        CUSTOMER
ORDER BY    CustomerName;
```

Triggers

- A **trigger** is a stored program that executes whenever a specified event occurs
- A trigger is attached to a table or view, and is invoked when there is an *insert, delete, update* request on that table/view

There are three types of triggers:

- BEFORE: are executed before the request
- AFTER: are executed after the request
- INSTEAD OF: are executed in place of any processing of the request

Uses of triggers

The text describes four uses for triggers:

- Providing default values
- Enforcing data constraints
- Updating SQL views
- Performing referential actions

In this week's lab we will use a trigger to audit activity on a database table, by writing to an audit table whenever a change is made to the data table.

Uses of triggers - Oracle

Oracle describes the following uses for triggers:

- Automatically generate derived column values
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access

https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#g1041679

https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#g1041679

Uses of triggers – Oracle cont'd

- Modify table data when DML statements are issued against views
- Publish information about database events, user events, and SQL statements to subscribing applications
- Restrict DML operations against a table to those issued during regular business hours
- Enforce security authorizations
- Prevent invalid transactions

Oracle recommends triggers should NOT be used for integrity constraints where those features are already available in the DBMS

Stored procedures

A stored procedure is a program that is stored within the database and compiled when used

- Oracle stored procedures can be written in PL/SQL or Java
- Can take input parameters and return results
- Are attached to the database (rather than table/view)
- Can be executed by any user or process that has permission
- Used by DBA for administrative tasks, or called by application programs to supply application logic
- Always reside within the DBMS on the database

Summing up…

- SQL statements can be embedded in program code in functions, triggers and stored procedures
- **User-defined functions** take input parameters from an SQL statement and return a value
- A **trigger** is a stored program that is executed by the DBMS whenever a specified event occurs on a specified table or view
- **Stored procedures** are modules of code that are database-wide and can be used as part of an application

Topic learning outcomes revisited

/

- Explain why there is a ‘mismatch’ between SQL and procedural programming languages
- Describe how cursors and variables are used to embed SQL into program code
- Explain the difference between user-defined functions, triggers, and stored procedures, and appropriate uses
- Give an example of a user-defined function
- Be able to create simple triggers in Oracle PL/SQL
- Be able to create simple stored procedures in Oracle PL/SQL
- Explain the advantages of using stored procedures for database applications

Murdoch UNIVERSITY

What's next?

- In the next topic we look at how databases can be used to add value beyond their role in operational transaction processing, by supporting management analysis, planning, and decision making.
- In the associated lab, we will take a brief look at some of the features in Oracle that extend its application into Business Intelligence.