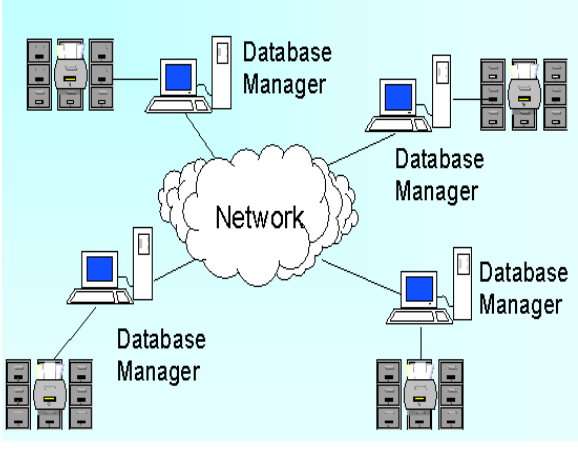
ADVANCED TOPICS IN DATABASES

Distributed Databases

Distributed Database

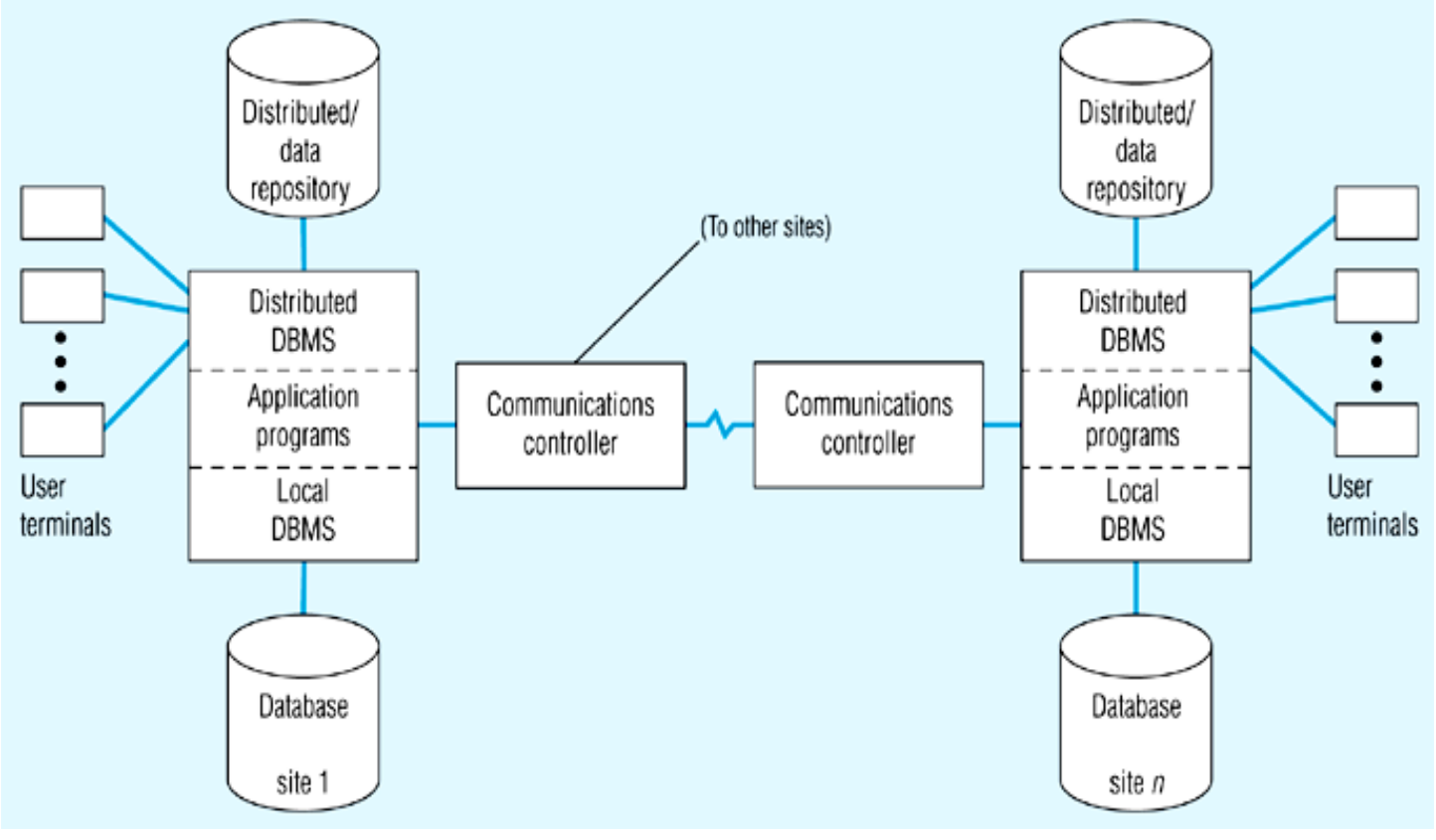
- A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network.

- A distributed database management system (DDBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users



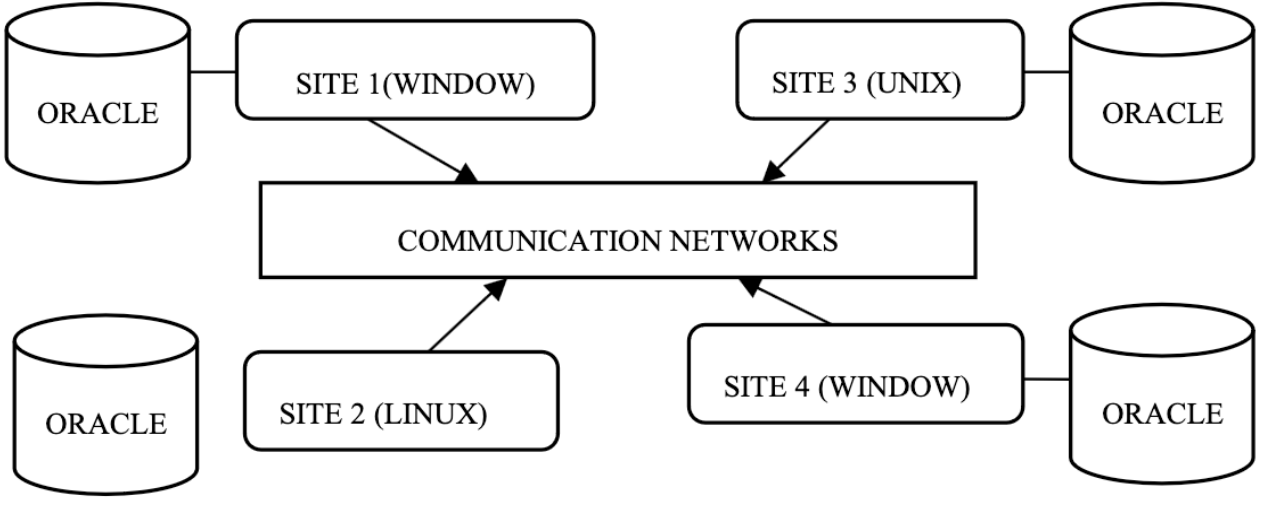
Distributed Database System (DDBS) = DDB + DDBMS

Distributed DBMS Architecture



Distributed Database. System

Homogeneous Distributed Database System.



Autonomous – each DBMS works independently, passing messages back and forth to share data updates.

Nonautonomous – a central, or master, DBMS coordinates database access and updates across the sites.

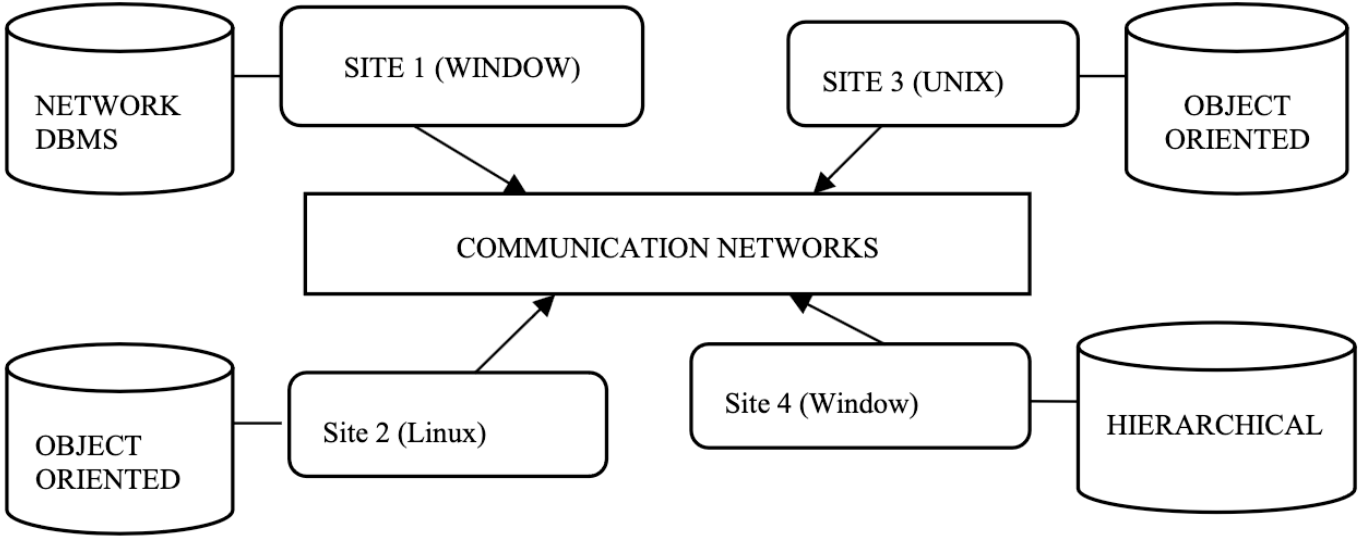
Distributed Database System

Homogeneous Distributed Database System.

- This environment is typically defined by the following characteristics:
  - Data are distributed across all the nodes.
  - The same DBMS is used at each location.
  - All data are managed by the distributed DBMS. There is no exclusively local data.
  - All users access the database through one global schema or database definition.
  - The global schema is simply the union of all the local database schemas.

Distributed Database System

Heterogeneous Distributed Database System



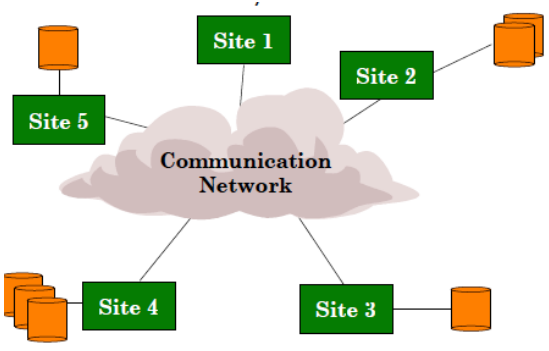
Distributed Database System

Heterogeneous Distributed Database System

- It is difficult in most organizations to force a homogeneous environment, yet heterogeneous environments are much more difficult to manage.
- a typical heterogeneous distributed database environment is defined by the following characteristics:
  - Data are distributed across all the nodes.
  - Different DBMSs may used at each location.
  - Some users require only local access to databases, which can be accomplished using only the local DBMS and schema.
  - A global schema exists, which allows local users to access remote data.

Distributed Database

- The data is physically stored by multiple machines (also referred to by sites or nodes), and each site is usually managed by a DBMS (Database Management System) and is able to run independently of the other sites
- Data is shared by users on the multiple machines.
- The network interconnects the machines, so the communication cross-site interaction is carried out over the network
- Transactions can access data in one or more sites, all at once
- Each node can be a query site, a data site, or both



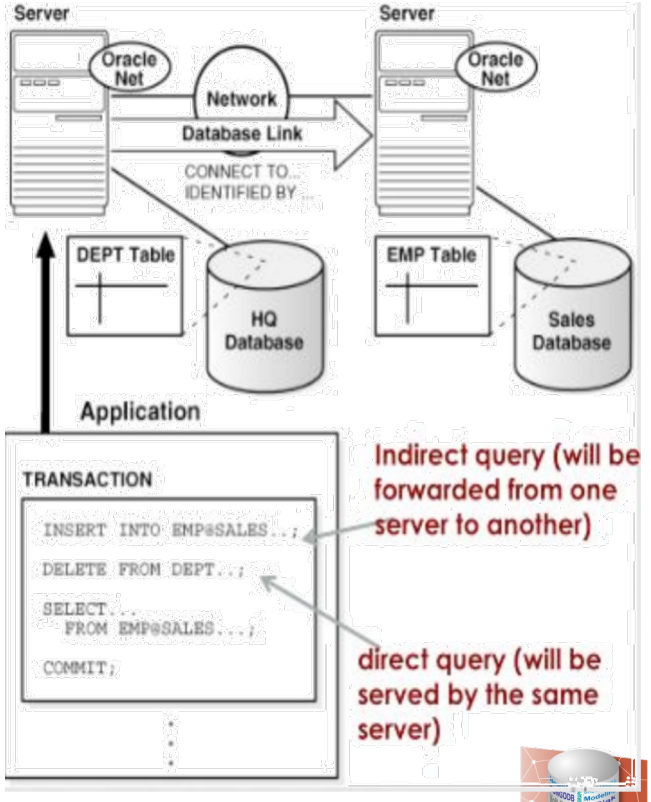
Distributed Database

Client - Server

- Client connects directly to specific server(s) and access only their data
  - The client is the application that requests information from the database data
- Direct queries only

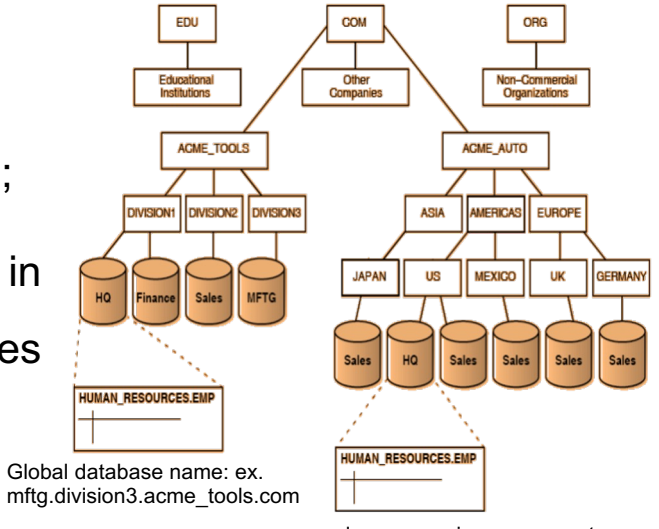
Collaborative Servers

- Servers can serve queries or be clients and query other servers .
- Support indirect queries;



Distributed Database

- Each database in a distributed system is distinct from all other databases and has its **own global database name;**
- Data distribution is managed by factors such **as local administration and** increases availability and performance;
- The system should address the impact of data distribution in a transparent manner. In particular, the following properties are desirable:
  - Independent data distribution
  - Atomicity in data distribution



Global database name: ex. mftg.division3.acme_tools.com

sales.us.americas.acme_auto.com

- It is **easy to expand** either by **adding new nodes** or **by improving hardware features** (disks, processors, etc.)

Distributed Database

Transparency can be understood as the separation of the high-level semantics of a system from the low-level aspects related to its implementation.

TYPES

- Transparency in the location of data relative to the application level (physical distribution of data – logical integration of data)
- Transparency availability
- Transparency fragmentation
- Transparency replication

The transparency features of the distributed database management system (DDBMS) **have** the **common property** of allowing the user to feel that they are the only one using the database

Objetives of a DDB

The fundamental principle of distributed database

To the user, a distributed database system should look exactly like a no distributed database system.

- The fundamental principle of distributed databases gives rise to a set of twelve fundamental objectives. These objectives were defined by C.J. Date in 1990.

1. Local autonomy
2. No reliance on a central site
3. Continuous operation
4. Location transparency
5. Fragmentation transparency
6. Replication transparency
7. Distributed query processing
8. Distributed transaction management
9. Hardware independence
10. Operating system independence
11. Network transparency
12. DBMS independence

Objetives of a DDB

Local autonomy - means that all operations at a given site X are controlled by that site: no site X should depend on some site Y for its successful operation, otherwise if site Y is down, then X cannot run even if there is nothing wrong with site X itself.

No reliance on a central site -There must not be any reliance on a central “master” site for some central service, such as transaction management or query processing.

Continuous operation - An advantage of distributed systems in general is that they can provide greater reliability and greater availability.

Location transparency - is desirable because it simplifies application programs and end-user activities; in particular, it -allows data to migrate from site to site without invalidating any of those programs or activities

Objetives of a DDB

Fragmentation transparency- allows data to be refragmented at any time (and fragments to be redistributed at any time) in response to changing performance requirements.

Replication transparency - allows users to behave – from a logical standpoint – as if the data were not replicated.

Distributed transaction management - There are two major aspects to transaction management, recovery and concurrency, and both require extended treatment in a distributed environment.

Hardware independence - Real-world computer installations typically involve a multiplicity of different machines and hardware which must be configured to integrate the data on all of the systems to present the user with a "single-system image".

Objetives of a DDB

Operating system independence - This objective is a corollary of the previous one. It is obviously desirable to be able to run the same DBMS on different operating systems on either different or the same hardware.

Network transparency - The system should be able to support many different sites, with different hardware and disparate operating systems. It is also desirable to support a variety of different communication networks

DBMS independence - The system should be able to relax any requirements for strict homogeneity amongst the DBMSs.

Forms of distributed data storage

- ***Replication***
  - There are multiple copies of the data, stored on different nodes
  - Great fault tolerance
- ***Fragmentation***
  - Determining distinct shard sets that constitute logical allocation units
  - Each logical unit is stored in a specific node (possibly different, but not necessarily)
  - The existing data in the global relation is mapped across the fragments without any loss
- ***Replication and fragmentation***
  - Relations are divided into distinct fragments, and there are multiple copies of these fragments replicated by different nodes

Data Replication

- A relationship or part of a relationship is replicated if it is stored redundantly in two or more sites;
- **Full replication of a relationship happens when a relationship is saved in all sites;**
- **Fully redundant databases are those where each site has a copy of the entire database.**

Vantagens

- Availability

  If replicas exist, the failure of a node does not imply unavailability

- Parallelism

  Queries can be performed on multiple nodes simultaneously

- Reduction of the amount of data to be transferred

  A relationship is available on each node that has a replica

Data Replication

Disadvantages

- Replica updates
- Each replica needs to be updated
- Failed updates or concurrent updates can imply data inconsistency

Synchronous replication: All replicas are updated before transactions are submitted

Increased response time

Asynchronous replication: Replicas are **updated periodically**

Temporary data inconsistency

Data Fragmentation

- It consists of dividing a relationship into smaller relationships or fragments, and storing the fragments (rather than the relationship itself), possibly at different sites.
- **Horizontal fragmentation**: Each fragment consists of a subset of tuples from the original relationship. **Fragments must be disjoint**.
- **Vertical fragmentation**: Each fragment consists of a subset of columns (attributes) from the original relationship.

TID	eid	name	city	age	sal
t1	53666	Jones	Madras	18	35
t2	53688	Smith	Chicago	18	32
t3	53650	Smith	Chicago	19	48
t4	53831	Madayan	Bombay	11	20
t5	53832	Guldu	Bombay	12	20

Vertical

Horizontal

Data Fragmentation

- **Advantages**
  - Smaller relationships – less data means better response time
  - Parallelism - queries can be performed on multiple nodes simultaneously
  - Availability - allows a relationship to be divided so that the data is where it is most frequently accessed;

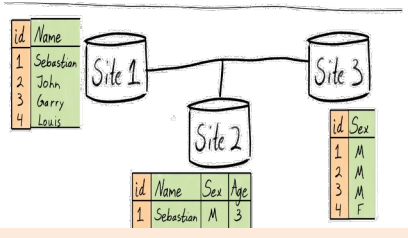
- **Disadvantages**
  - Splitting queries across multiple nodes
  - If relationships are fragmented, it may be necessary to perform join and join operations
  - Compared to replication, it can lead to longer response times when the desired data is on different nodes

Vertical Fragmentation

- Each fragment is a set of attributes of the original relationship
- Each fragment has a smaller schema than the original relation schema
- Each fragment of a relationship must have a common attribute to allow mapping the original relationship without loss, that is, the primary key of

Original table-.

id	Name	Sex	Age
1	Sebastian	M	3
2	John	M	5
3	Garry	M	8
4	Louis	F	7



Its objective is to identify fragments in such a way that several applications can be executed with access to only one partition of the global relationship.

If an application needs access to data present in more than one fragment, vertical fragmentation is not beneficial as it will be necessary to make joins between the fragments to obtain the result

Horizontal Fragmentation

- Each fragment is a set of tuples from the original relation
- Fragments must be disjoint

- ***Techniques used***
  - **Fragmentation by range of values**
    - The fragments are obtained depending on the values of the attributes of the relationship, taking into account the predicates existing in applications

Select name
from Teacher
where age < 70

Logical predicate

suggests creating two partitions:
- one with teachers under the age of 70 and
- another with teachers aged 70 or over.

So,
the horizontal fragmentation functions FH1 and FH2 on the teacher table would be defined by:

FH1= (age < 70) e FH2= (age >= 70).

Horizontal Fragmentation

- ***Techniques use***
  - **Fragmentation by Hashing function**
    - The global relationship is fragmented according to the value of a hashing function applied to one of the attributes, in order to relate each tuple to a fragment. The hash function itself represents the FH function.
  - **Circular fragmentation**
    - It consists of the sequential association of each tuple of the global relation to a fragment without taking into account the value of any attribute.
    - This way, if the relation contains n tuples and the number of fragments to be generated is m, each fragment will receive n/m tuples.
    - This form of fragmentation is quite used in systems that exploit parallel processing.

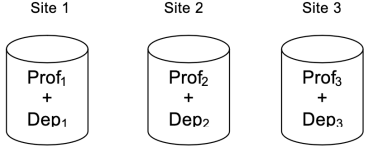
Horizontal Fragmentation

- **Derived fragmentation**
  - The fragmentation derived from a global relationship is not based on the properties of the its own attributes but due to the fragmentation of another relationship. It is used to facilitate the join operation between fragments

Exemplo

- We have Department and Professor;
- The department has n professors, and the Department table has been partitioned in three partitions (Dep1, Dep2, Dep3);
- The FHD was applied to the Teacher table, defining its derived horizontal fragments as follows:

$$\text{Prof}_1 = \text{Professor} \ltimes \text{Dep}_1$$
$$\text{Prof}_2 = \text{Professor} \ltimes \text{Dep}_2$$
$$\text{Prof}_3 = \text{Professor} \ltimes \text{Dep}_3$$



Hybrid Fragmentation

- Combination of horizontal and vertical fragmentation
- It can be done in two ways: fragmenting horizontally the relation and then for each fragment obtained, fragment it vertically, or vice-versa
- Although these operations can be recursively repeated, what happens in practice is a interruption of the process after the second iteration.
- The order in which fragments are applied affects the final result.

Fragmentation in Oracle

Oracle DBMS supports horizontal table sharding in three ways:

- List- Explicitly controls how fragmentation will happen by determining how each record is allocated to a partition.

```
CREATE TABLE sales_by_region (
deptno            NUMBER(10),
deptname          VARCHAR2(20),
quarterly_sales   NUMBER(10,2),
state                    VARCHAR2(2))
PARTITION BY LIST (state) (
PARTITION q1_northwest VALUES ('OR', 'WA'),
PARTITION q1_southwest VALUES ('AZ', 'CA', 'NM'),
PARTITION q1_northeast VALUES ('NY', 'VT', 'NJ'),
PARTITION q1_southeast VALUES ('FL', 'GA'),
PARTITION q1_northcent VALUES ('MN', 'WI'),
PARTITION q1_southcent VALUES ('OK', 'TX'));
```

Fragmentation in Oracle

- **Range -** In this case, the records are mapped to the partitions based on the value ranges of each partition of the key(s) defined for sharding.

```
CREATE TABLE professional_history (
prof_history_id  NUMBER(10),
person_id        NUMBER(10) NOT NULL,
organization_id  NUMBER(10) NOT NULL,
record_date      DATE NOT NULL,
ph_comments      VARCHAR2(2000))
PARTITION BY RANGE (record_date) (
PARTITION yr0
VALUES LESS THAN (TO_DATE('01-JAN-2000','DD-MON-YYYY'))
TABLESPACE <tablespace_name>,
PARTITION yr1
VALUES LESS THAN (TO_DATE('01-JAN-2001','DD-MON-YYYY'))
TABLESPACE <tablespace_name>,
PARTITION yr2
VALUES LESS THAN (TO_DATE('01-JAN-2002','DD-MON-YYYY'))
TABLESPACE <tablespace_name>,
PARTITION yr9
VALUES LESS THAN (MAXVALUE)
TABLESPACE <tablespace_name>);
```

Distribution key with more than one attribute

```
CREATE TABLE sales    (
invoice_no NUMBER,
      sale_year  INT NOT NULL,
      sale_month INT NOT NULL,
      sale_day   INT NOT NULL )
PARTITION BY RANGE (sale_year, sale_month, sale_day)
    ( PARTITION sales_q1 VALUES LESS THAN (1997, 04, 01)
        TABLESPACE <tablespace_name>,
      PARTITION sales_q2 VALUES LESS THAN (1997, 07, 01)
        TABLESPACE <tablespace_name>,
      PARTITION sales_q3 VALUES LESS THAN (1997, 10, 01)
        TABLESPACE <tablespace_name>,
      PARTITION sales_q4 VALUES LESS THAN (1998, 01, 01)
        TABLESPACE <tablespace_name> );
```

Fragmentation in Oracle

- **Hashing -** is generally used when there are no access characteristics that indicate fragmentation by Range or List. In this case, a hash function is used that splits the records between the partitions defined at the time of sharding:

```
CREATE TABLE professional_history (
prof_history_id  NUMBER(10),
person_id        NUMBER(10) NOT NULL,
organization_id  NUMBER(10) NOT NULL,
record_date      DATE NOT NULL,
prof_hist_comments VARCHAR2(2000))
PARTITION BY HASH (prof_history_id)
PARTITIONS 3
STORE IN (tablespace_name1, tablespace_name2, tablespace_name3);
```

Oracle even allows **these techniques to be combined** and **applied together** on a table through Composite Partitioning.

Oracle supports two types of combinations for fragmentation: Range-List and Range-Hash

Selecting a Distribution Strategy

A distributed database can be organized in five unique ways:

Totally centralized – all data resides at one location accessed from many geographically distributed sites.

Partially or totally replicated (snapshot) – data is either partially or totally replicated across geographically distributed sites, with each replica periodically updated with snapshots.

Partially or totally replicated (real-time synchronization) – data is either partially or totally replicated across geographically distributed sites, with near real-time synchronization.

Fragmented (integrated) – data is into segments at different geographically distributed sites, but still within one logical database and one distributed DBMS.

Fragmented (nonintegrated) – data is fragmented into independent, non integrated segments spanning multiple computer systems and database software.

Summary of Distributed Design Strategies

Strategy	Reliability	Expandability	Communications Overhead	Manageability	Data Consistency
Centralized	**POOR:** Highly dependent on central server	**POOR:** Limitations are barriers to performance	**VERY HIGH:** High traffic to one site	**VERY GOOD:** One monolithic site requires little coordination	**EXCELLENT:** All users always have the same data
Replicated with snapshots	**GOOD:** Redundancy and tolerated delays	**VERY GOOD:** Cost of additional copies may be less than linear	**LOW to MEDIUM:** Not constant, but periodic snapshots can cause bursts of network traffic	**VERY GOOD:** Each copy is like every other one	**MEDIUM:** Fine as long as delays are tolerated by business needs
Synchronized replication	**EXCELLENT:** Redundancy and minimal delays	**VERY GOOD:** Cost of additional copies may be low and synchronization work only linear	**MEDIUM:** Messages are constant, but some delays are tolerated	**MEDIUM:** Collisions add some complexity to manageability	**MEDIUM to VERY GOOD:** Close to precise consistency
Integrated partitions	**VERY GOOD:** Effective use of partitioning and redundancy	**VERY GOOD:** New nodes get only data they need without changes in overall database design	**LOW to MEDIUM:** Most queries are local, but queries that require data from multiple sites can cause a temporary load	**DIFFICULT:** Especially difficult for queries that need data from distributed tables, and updates must be tightly coordinated	**VERY POOR:** Considerable effort; and inconsistencies not tolerated
Decentralized with independent partitions	**GOOD:** Depends on only local database availability	**GOOD:** New sites independent of existing ones	**LOW:** Little if any need to pass data or queries across the network (if one exists)	**VERY GOOD:** Easy for each site, until there is a need to share data across sites	**LOW:** No guarantees of consistency; in fact, pretty sure of inconsistency

Distributed Database Issues

- **Distributed Database Design**
  - how to distribute the database
  - replicated & non-replicated database distribution
  - a related problem in directory management

- **Query Processing**
  - convert user transactions to data manipulation instructions
  - Optimization problem
  - min{cost = data transmission + local processing}

Distributed Database Issues

- **Concurrency Control**
  - synchronization of concurrent accesses
  - consistency and isolation of transactions' effects
  - deadlock management

- **Reliability**
  - how to make the system resilient to failures
  - atomicity and durability

References

T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 25 Distributed DBMSs - Advanced Concepts, Pearson Education Ltd, 2015

Principles of Distributed Database Systems, Third Edition. M. Tamer Özsu · Patrick Valduriez. Principles of Distributed Database Systems, Third Edition. Springer ISBN 978-1-4419-8833-1 2010.

CORONEL, Carlos; MORRIS, Steven; ROB, Peter. Database Systems: Design. *Implementation, and Management, Boston. Chapter 12.*