Databases 2023

Darko Bozhinoski,
Ph.D. in Computer Science

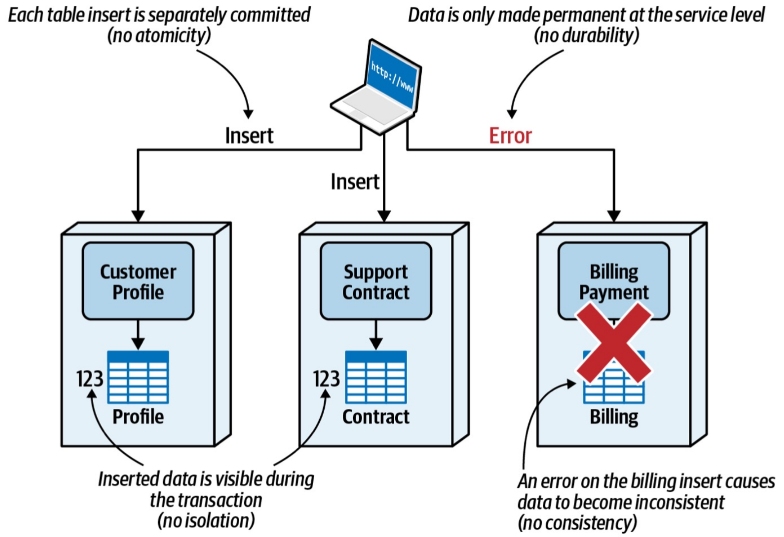
AGENDA

- Distributed Database Concepts
- Distributed Transaction Properties
- Eventual Consistency Patterns
- Data Fragmentation, Replication and Allocation
- Query Processing
- Practical Assignment

Distributed Database Concepts

- A transaction can be executed by multiple networked computers in a unified manner.
- A **distributed database (DDB)** processes units of execution (transactions) in a distributed manner.
- A distributed database (DDB) is a collection of multiple logically related database distributed over a computer network, and a distributed database management system as a software system that manages a distributed database while making the distribution transparent to the user.

Distributed transactions



Each table insert is separately committed (no atomicity)
Data is only made permanent at the service level (no durability)
http://www
Insert
Error
Insert
Customer Profile
123
Profile
Support Contract
123
Contract
Billing Payment
Billing
Inserted data is visible during the transaction (no isolation)
An error on the billing insert causes data to become inconsistent (no consistency)

In a distributed transaction, atomicity is bound to the service, not the business request (such as customer registration)

Consistency is not supported because a failure in one service causes the data between tables

Isolation is not supported because once the Customer Profile service inserts the profile data in the course of a distributed transaction to register a customer, that profile information is available to any other service or request, even though the customer registration process (the current transaction) hasn’t completed.

Durability is not supported: any individual commit of data does not ensure that all data within the scope of the entire business transaction is permanent.

BASE Properties for distributed transactions

- **Basic Availability(BA):** all of the services or systems in the distributed transaction are expected to be available to participate in the distributed transaction.

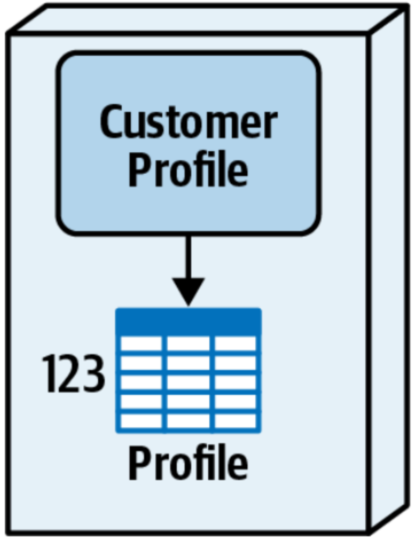
- **Soft state:** describes the situation where a distributed transaction is in progress and the state of the atomic business request is not yet completed (in some cases not even known).

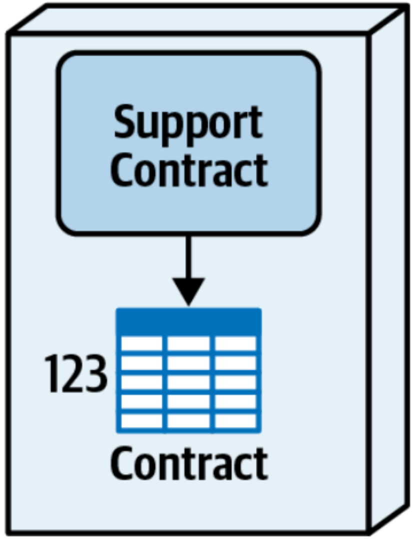
- **Eventual consistency:** given enough time, all parts of the distributed transaction will have completed successfully and all of the data is in sync with one another.--> The type of eventual consistency pattern used and the way errors are handled dictates how long it will take for all of the data sources involved in the distributed transaction to become consistent.

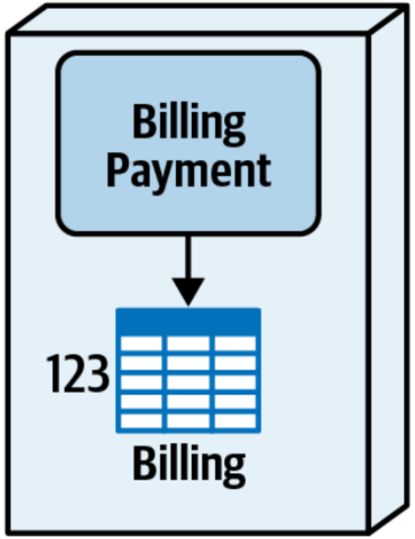
Eventual Consistency Patterns

- Background synchronization pattern
- Orchestrated request-based pattern
- Event-based pattern

Scenario I





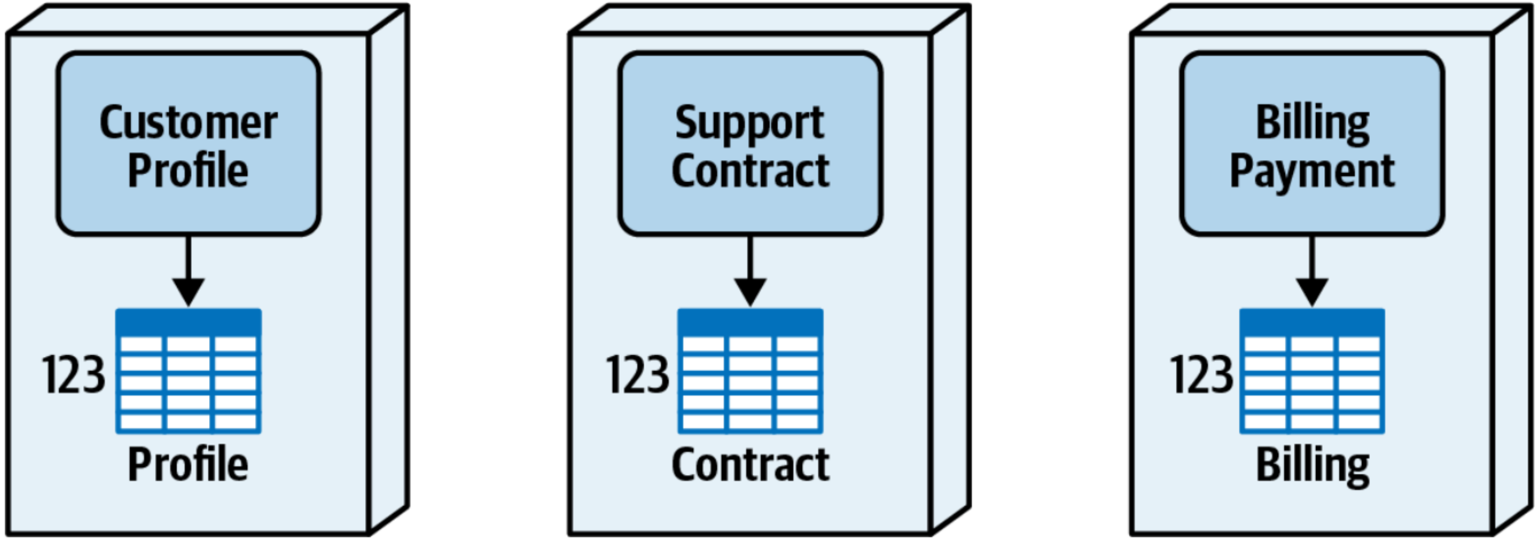


Customer Profile service maintains basic profile information

Support Contract service that maintains products covered under a repair plan for each customer

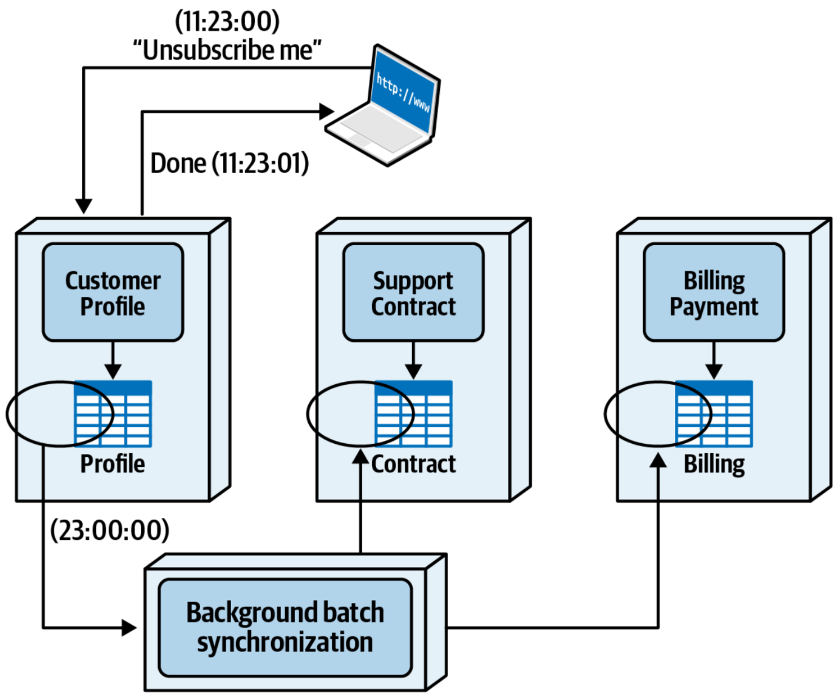
Billing Payment service that charges the customer for the support plan

Scenario I



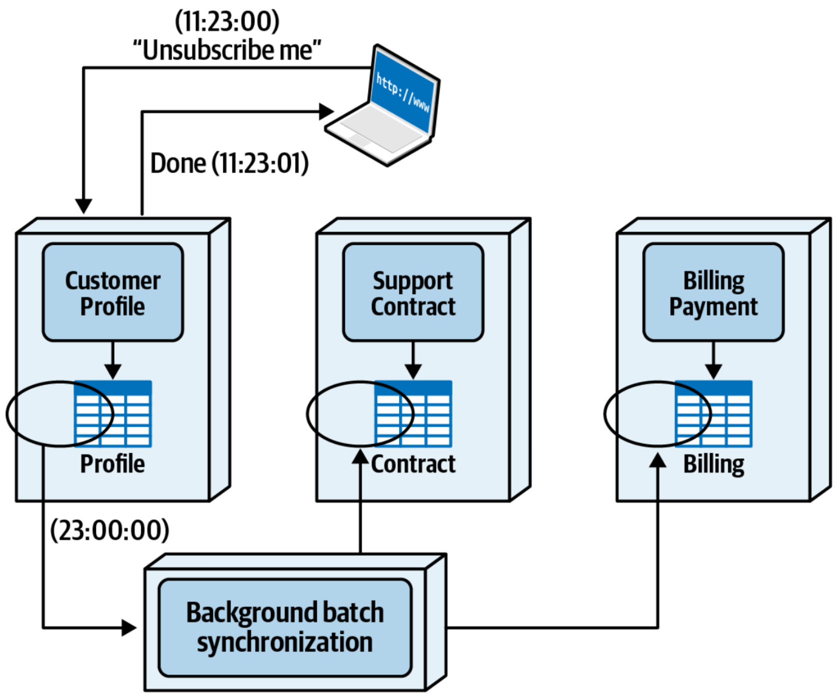
Customer 123 unsubscribes from the service. Customer Profile service receives a request from the user interface, removes the customer from the Profile table, and returns a confirmation to the customer they are successfully unsubscribed and will no longer be billed.
The data for that customer still exists in the Contract and Billing table!

Background synchronization pattern



- A separate external service or process to periodically check data sources and keep them in sync with one other.
- The length of time for data sources to become eventually consistent using this pattern can vary based on the implementation of the background process (a batch job or a service periodically activated.)

Background synchronization pattern

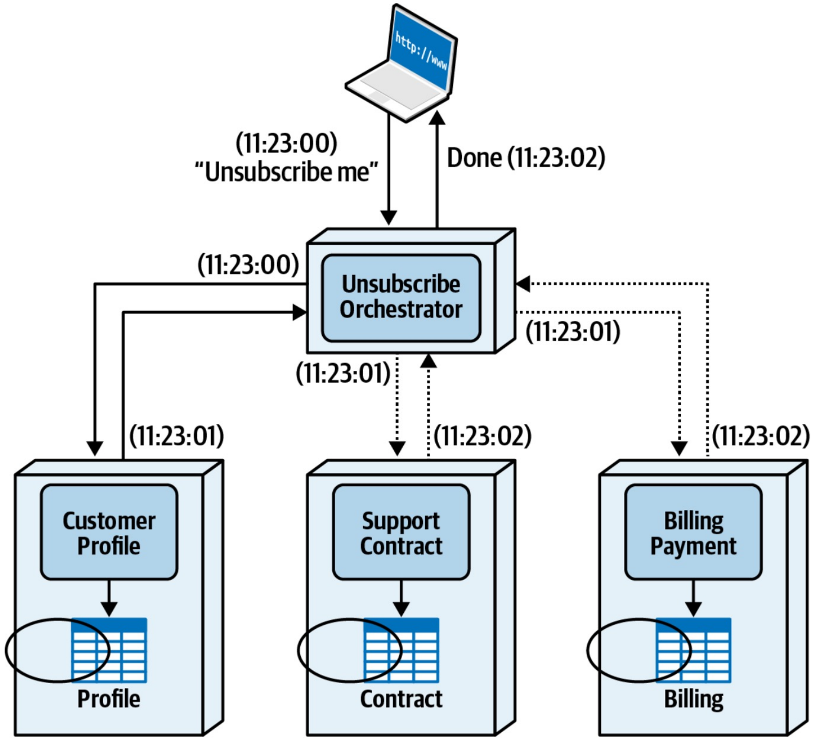


+ Improves the overall responsiveness (the end user doesn't have to wait for the entire business transaction to complete)

- Breaks the bounded context between the data and the services: couples all of the data sources together
    - Structural changes of the schema are difficult
    - Duplicated Business logic
    - slow eventual consistency

This pattern is useful when applied on closed (self-contained) heterogeneous systems that don't communicate with each other or share data.

Orchestrated Request-Based Pattern

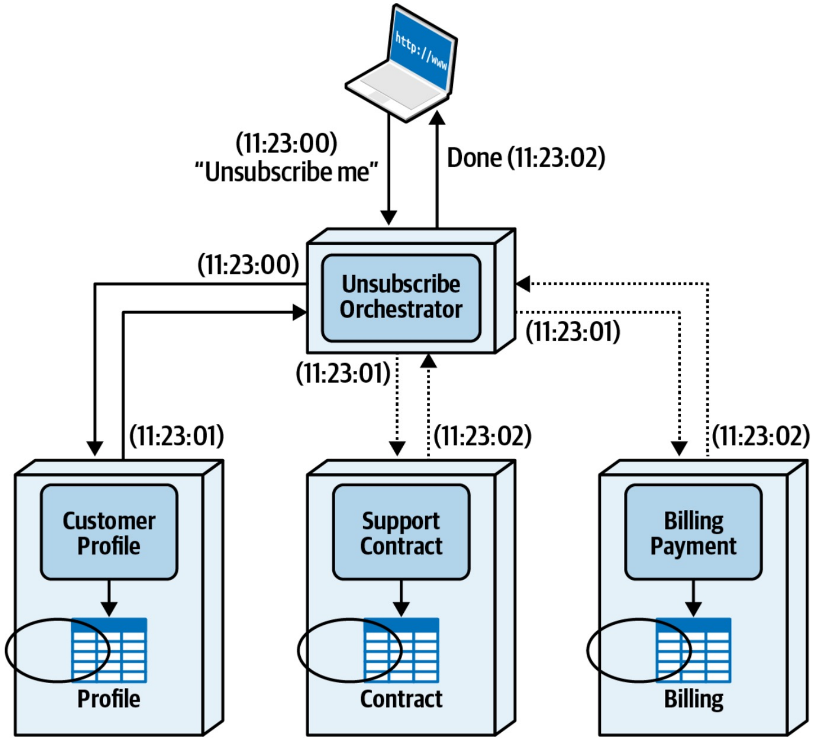


- Orchestrated request-based pattern uses an orchestrator to manage the distributed transaction during the business request

  -> a designated existing service or a new separate service to be orchestrator

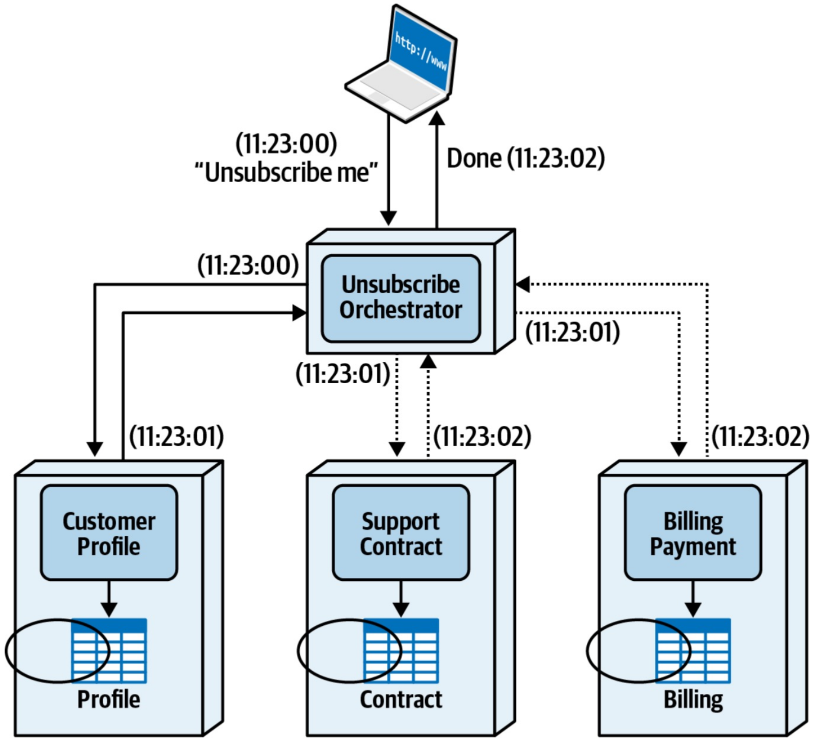
Preferable approach: use a dedicated orchestration service for the business request.

Orchestrated Request-Based Pattern



1. Customer issues a request to unsubscribe from the support plan.
2. The request is received by the Unsubscribe Orchestrator which then forwards the request synchronously to the Customer Profile service to remove the customer from the Profile table.
3. Customer Profile service sends back an acknowledgement to the Unsubscribe Orchestrator
4. The Unsubscribe Orchestrator sends parallel requests to both the Support Contract and Billing Payment services.
5. Both services process the unsubscribe request, and then send an acknowledgement to the Unsubscribe Orchestrator.
6. Unsubscribe Orchestrator responds back to the client"

Orchestrated Request-Based Pattern



+ Favours data consistency over responsiveness

- Additional network hops and service calls
- Complex error handling.

What happens when the customer is removed from the Profile table and Contract table, but an error occurs when trying to remove the billing information from the Billing table (outstanding billing charge)?

The orchestrator must decide what action to take.

- Resend request?
- A compensating transaction and request services to reverse their update operations?
- Respond to the user that an error occurred?
- Ignore the error and wait for other process to notify the customer?

Event-Based Pattern

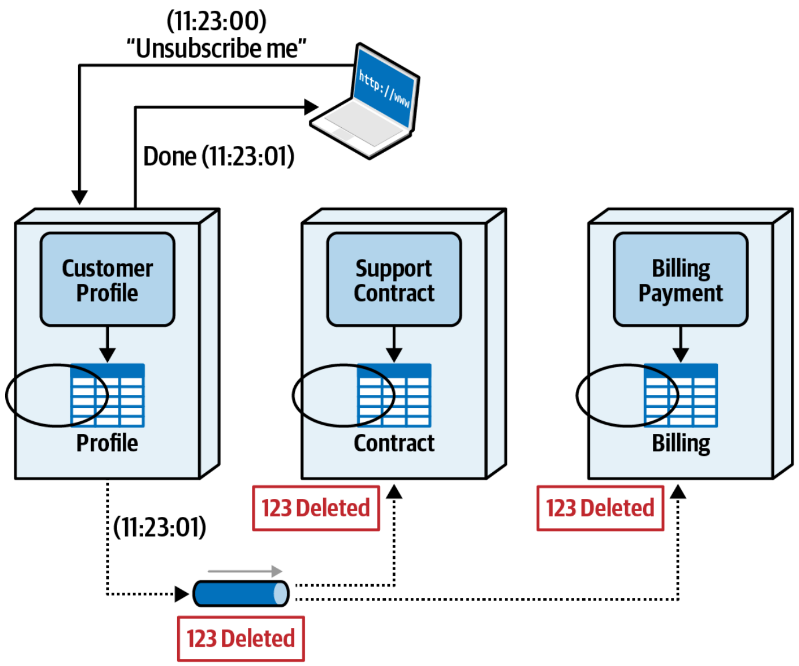
Popular and reliable eventual consistency patterns for most modern distributed architectures (microservices and event-driven architectures)

-> usually events are used in conjunction with an asynchronous (pub/sub) messaging.

+ High decoupling, good responsiveness, eventual consistency time is very short

- Error handling.

Event-Based Pattern



The Customer Profile service receives the request, removes the customer from the Profile table, publishes a message and returns information letting the customer know they were successfully unsubscribed.

At around the same time, both the Support Contract and Billing Payment services receive the unsubscribe event and perform actions to unsubscribe the customer, making all the data sources eventually consistent.

What constitutes a Distributed DataBase System (DDBS)?

- **Connection of database nodes over a computer network.** There are multiple computers, called **sites** or **nodes**. These sites must be connected by an underlying **network** to transmit data and commands among sites.
- **Logical interrelation of the connected databases.** It is essential that the information in the various database nodes be logically related.
- **Possible absence of homogeneity among connected nodes.** It is not necessary that all nodes be identical in terms of data, hardware, and software.

What constitutes a Distributed DataBase System (DDBS)?

❖ **Connection of database nodes over a computer network.** There are multiple computers, called **sites** or **nodes**. These sites must be connected by an underlying **network** to transmit data and commands a

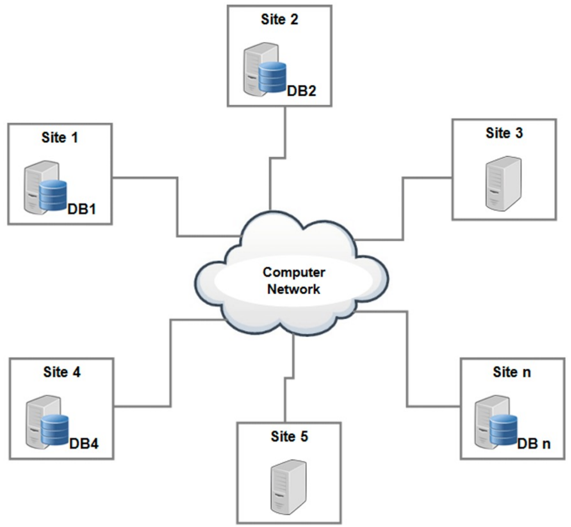
❖ **L** ential th y related.

❖ **P** **es.** It is n dware, a

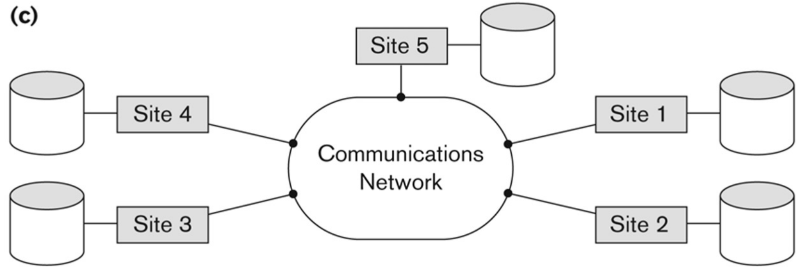
The type and topology of the network used may have a significant impact on the performance and hence on the strategies for distributed query processing and distributed database design.

Distributed Database System

- Management of distributed data with different **levels of transparency**:
  - This refers to the physical placement of data (files, relations, etc.) which is not known to the user (distribution transparency).

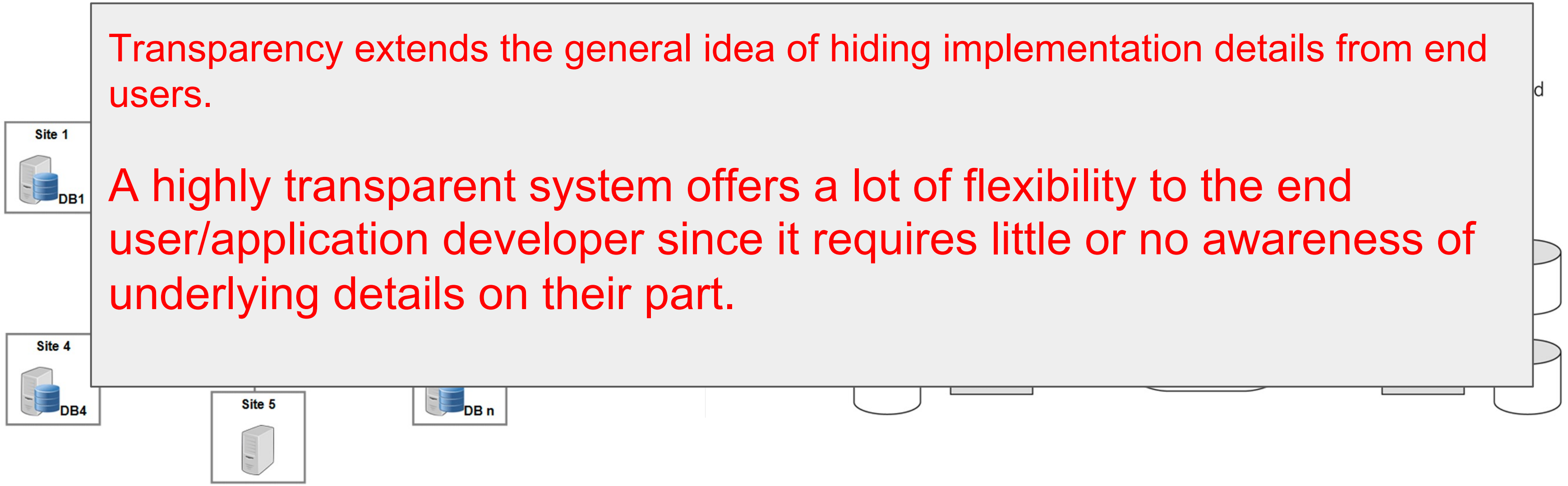


Some different database system architectures. (a) Shared nothing architecture. (b) A networked architecture with a centralized database at one of the sites. (c) A truly distributed database architecture.



Distributed Database System

- Management of distributed data with different **levels of transparency**:
  - This refers to the physical placement of data (files, relations, etc.) which is not known to the user (distribution transparency).



Transparency extends the general idea of hiding implementation details from end users.

A highly transparent system offers a lot of flexibility to the end user/application developer since it requires little or no awareness of underlying details on their part.

Distribution/Network transparency

- Users do not have to worry about operational details of the network.
  - <u>Location transparency</u>, which refers to freedom of issuing command from any location without affecting its working.
  - <u>Naming transparency</u>, which allows access to any names object (files, relations, etc.) from any location.

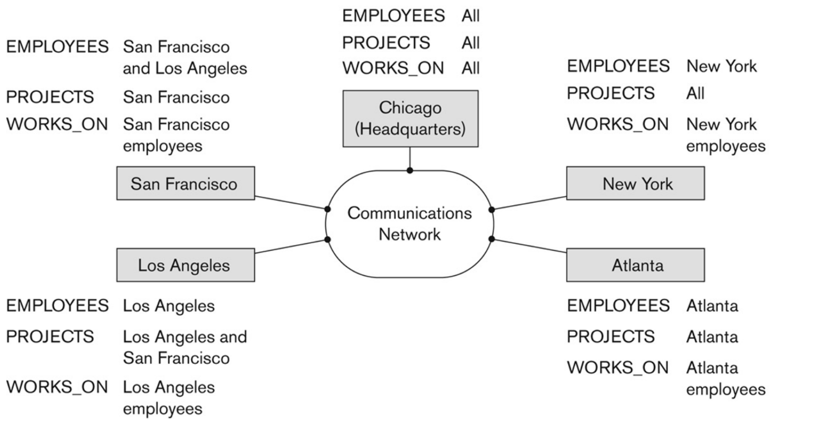
Replication and Fragmentation Transparency

- **Replication transparency**:
  - It allows to store copies of a data at multiple sites.
  - This is done to minimize access time to the required data.
- **Fragmentation transparency**:
  - Allows to fragment a relation horizontally (create a subset of tuples of a relation) or vertically (create a subset of columns of a relation).

Replication and Fragmentation Example

- The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally and stored with possible replication as shown below.

Figure 25.2
Data distribution and replication among distributed databases.



Reliability and Availability

- **Increased reliability and availability**:
  - Reliability refers to the probability that the system is running at a certain moment of time.
  - Availability is the probability that the system is continuously available (usable or accessible) during a time interval.
  - A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job.

Reliability and Availability

- **Increased reliability and availability**:
  - A **failure** is a deviation of a system's behavior from that which is specified in order to ensure correct execution of operations.
  - **Errors** constitute that subset of system states that causes the failure.
  - **Fault** is the cause of an error.
- Different strategies to construct a system that is reliable
  - Stress fault tolerance: designs mechanisms that can detect and remove faults before they can result in a system failure
  - Exhaustive design process with extensive quality control

Performance and Scalability

- **Improved performance**:
  - A distributed DBMS fragments the database to keep data closer to where it is needed most.
  - This reduces data management (access and modification) time significantly.
- **Scalability**:
  - the extent to which the system can expand its capacity while continuing to operate without interruption
  - Allows new nodes (computers) to be added anytime without chaining the entire configuration.

Data Fragmentation, Replication and Allocation

Data Fragmentation, Replication and Allocation

- **Data Fragmentation**
  - Split a relation into logically related and correct parts. A relation can be fragmented in two ways:
    - **Horizontal Fragmentation**
    - **Vertical Fragmentation**
- The information concerning data fragmentation, allocation, and replication is stored in a global directory that is accessed by the DDBS applications as needed.

Horizontal fragmentation

- It is a horizontal subset of a relation which contain those of tuples which satisfy selection conditions.
- Consider the Employee relation with selection condition (DNO = 5). All tuples satisfy this condition will create a subset which will be a horizontal fragment of Employee relation.
- A selection condition may be composed of several conditions connected by AND or OR.
- Derived horizontal fragmentation: It is the partitioning of a primary relation to other secondary relations which are related with Foreign keys.

Horizontal fragmentation (2)

- **Representation**
  - Each horizontal fragment on a relation can be specified by a σ_{Ci} (R) operation in the relational algebra.
  - Complete horizontal fragmentation: A set of horizontal fragments whose conditions C1, C2, …, Cn include all the tuples in R- that is, every tuple in R satisfies (C1 OR C2 OR … OR Cn).
  - Disjoint complete horizontal fragmentation: No tuple in R satisfies (Ci AND Cj) where i ≠ j.
  - To reconstruct R from horizontal fragments a UNION is applied.

Vertical fragmentation

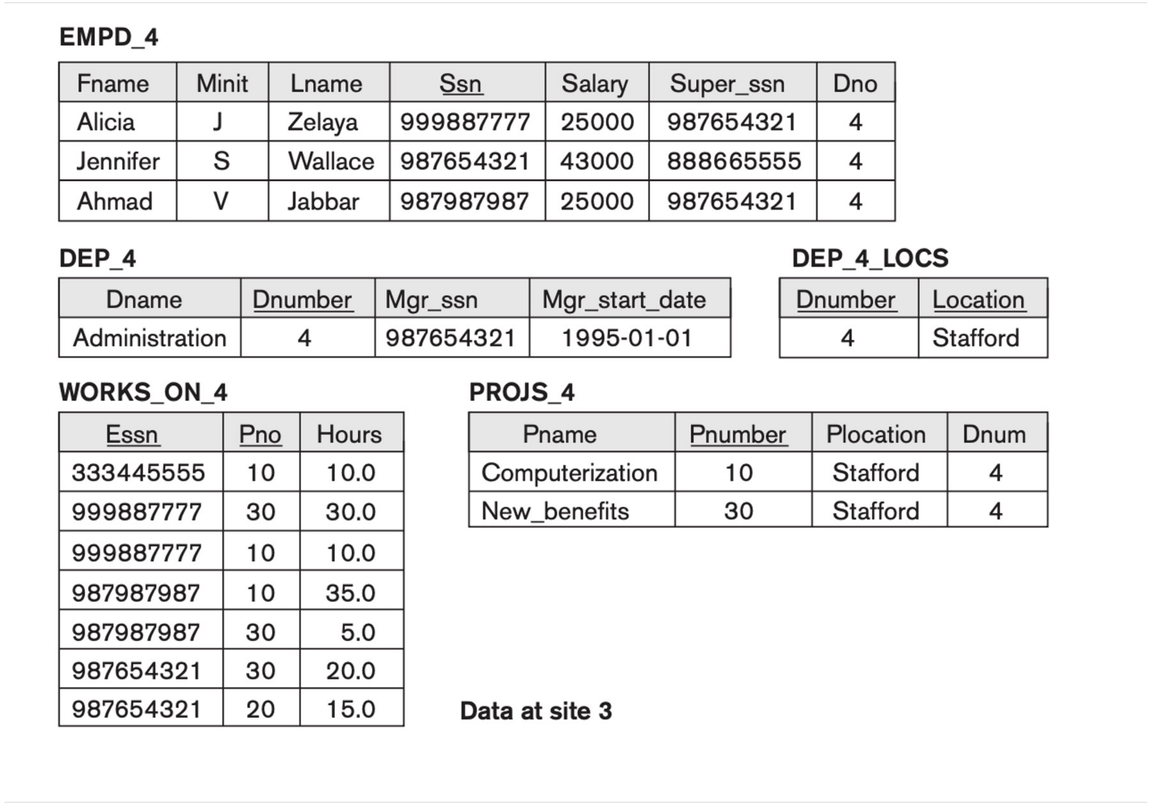
- It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation.
- Consider the Employee relation. A vertical fragment of this relation can be created by keeping the values of Name, Bdate, Address, and Sex in a first fragment and Ssn, Salary, Super_ssn, and Dno work-related information in a second fragment.
- Because there is no condition for creating a vertical fragment, each fragment must include the primary key attribute of the parent relation Employee. In this way all vertical fragments of a relation are connected.

Vertical fragmentation (2)

- **Representation**
  - A vertical fragment on a relation can be specified by a $\Pi_{Li}(R)$ operation in the relational algebra.
  - <u>Complete vertical fragmentation:</u> A set of vertical fragments whose projection lists L1, L2, …, Ln include all the attributes in R but share only the primary key of R. In this case the projection lists satisfy the following two conditions:
  - $L1 \cup L2 \cup ... \cup Ln = ATTRS (R)$
  - $Li \cap Lj = PK(R)$ for any i j, where ATTRS (R) is the set of attributes of R and PK(R) is the primary key of R.
  - To reconstruct R from complete vertical fragments a OUTER UNION is applied.

Mixed (Hybrid) fragmentation

- A combination of Vertical fragmentation and Horizontal fragmentation.
- This is achieved by SELECT-PROJECT operations which is represented by $\Pi_{Li}(\boldsymbol{\sigma}_{Ci}\ (R))$.
- If C = True (Select all tuples) and L ≠ ATTRS(R), we get a vertical fragment, and if C ≠ True and L ≠ ATTRS(R), we get a mixed fragment.
- If C = True and L = ATTRS(R), then R can be considered a fragment.



EMPD_4

Fname	Minit	Lname	Ssn	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	25000	987654321	4
Jennifer	S	Wallace	987654321	43000	888665555	4
Ahmad	V	Jabbar	987987987	25000	987654321	4

DEP_4

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Administration	4	987654321	1995-01-01

DEP_4_LOCS

Dnumber	Location
4	Stafford

WORKS_ON_4

Essn	Pno	Hours
333445555	10	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0

PROJS_4

Pname	Pnumber	Plocation	Dnum
Computerization	10	Stafford	4
New_benefits	30	Stafford	4

Data at site 3

Allocation of fragments to sites.

Decomposing Monolithic data

Decomposing a monolithic database is hard, and requires an architect to collaborate closely with the database team to safely and effectively break apart the data.

Step 1. Analyze Database and Create Data Domains
- data domain is a collection of coupled database artifacts—tables, views, foreign keys, and triggers—that are all related to a particular domain and frequently used together within a limited functional scope.

Step 2. Assign Tables To Data Domains

Decomposing Monolithic data

Decomposing a monolithic database is hard, and requires an architect to collaborate closely with the database team to safely and effectively break apart the data.

Step 1. Analyze

- data domain is eign

keys, and trigger used

together within a

A data domain is an architectural concept, whereas a schema is a database construct that holds the database objects belonging to a particular data domain. While the relationship between a data domain and a schema is usually one-to-one, data domains can be mapped to one or more schemas, particularly when combining data domains due to tightly coupled data relationships.

Step 2. Assign Tables To Data Domains

Decomposing Monolithic data

Step 3. Separate Database Connections To Data Domains

- the database connection logic within each service is refactored to ensure services connect to a specific schema and have read and write access to the tables belonging only to their data domain.
- When data from other domains is needed, do not reach into their databases. Instead, access it using the service that owns the data domain.
- The database is in a state of data sovereignty per service, which occurs when each service owns its own data.

Step 4. Move Schemas To Separate Database Servers
- moving schemas to separate physical databases

Step 5. Switch over to independent database servers:

- remove the connection to old database servers and remove the schemas from the old database servers

Fragmentation and Allocation Schema

- **Fragmentation schema**
  - A definition of a set of fragments (horizontal or vertical or horizontal and vertical) that includes all attributes and tuples in the database that satisfies the condition that the whole database can be reconstructed from the fragments by applying some sequence of UNION (or OUTER JOIN) and UNION operations.
- **Allocation schema**
  - It describes the distribution of fragments to sites of distributed databases. It can be fully or partially replicated or can be partitioned.

Data Replication and Allocation

- **Data Replication**
  - Database is replicated to all sites.
  - In full replication the entire database is replicated and in partial replication some selected part is replicated to some of the sites.
  - Data replication is achieved through a replication schema.
- **Data Distribution (Data Allocation)**
  - This is relevant only in the case of partial replication or partition.
  - The selected portion of the database is distributed to the database sites.

Data Replication and Allocation

- Replication is useful in improving the availability of data: system can continue to operate as long as at least one site is up.
- **Fully replicated distributed database**: advantages and disadvantages. **What is the trade-off?**

Data Replication and Allocation

- Replication is useful in improving the availability of data: system can continue to operate as long as at least one site is up.
- **Fully replicated distributed database**: advantages and disadvantages. **What is the trade-off?**

Advantageous: Improves performance of retrieval (read performance) for global queries because the results of such queries can be obtained locally from any site;

Disadvantageous: it can slow down update operations (write performance) drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent.

Data Replication and Allocation

- **Fully replicated distributed database**:
- **No replication**—that is, each fragment is stored at exactly one site. All fragments must be disjoint, except for the repetition of primary keys among vertical (or mixed) fragments. This is also called non redundant allocation.

Between these two extremes, we have a wide spectrum of partial replication of the data.

The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site!

Query Processing in Distributed Databases

Query Processing in Distributed Databases

A distributed database query is processed in stages as follows:

1. **Query Mapping.** The query on distributed data is translated into an algebraic query on global relations. This translation is done by referring to the global conceptual schema and does not take into account the actual distribution and replication of data.

    - Similar to the one performed in a centralized DBMS.

2. **Localization:** maps the distributed query on the global schema to queries on individual fragments using data distribution and replication information.

Query Processing in Distributed Databases (2)

3. Global Query Optimization: selecting a strategy from a list of candidates

- Generates a distributed execution plan so that least amount of data transfer occurs across the sites. The plan states the location of the fragments, order in which query steps needs to be executed and the processes involved in transferring intermediate results.
- Time is the preferred unit for measuring cost. The total cost is a weighted combination of costs such as CPU cost, I/O costs, and communication costs.
- <u>Communication costs over the network are the most significant.</u>

4. Local Query Optimization: common to all sites in the DDB.

- Similar to those used in centralized systems.

Data Transfer Costs of Distributed Query Processing

- Cost of transferring data (files and results) over the network.
  - This cost is usually high so some optimization is necessary.
  - **Example relations:** Employee at site 1 and Department at Site 2
    - Employee at site 1. 10,000 rows. Row size = 100 bytes. Table size = 10^6 bytes.

Fname	Minit	Lname	<u>SSN</u>	Bdate	Address	Sex	Salary	Superssn	Dno

    - Department at Site 2. 100 rows. Row size = 35 bytes. Table size = 3,500 bytes.

Dname	Dnumber	Mgrssn	Mgrstartdate

  - Q: For each employee, retrieve employee name and department name Where the employee works.
  - Q: $\Pi_{Fname,Lname,Dname}$ (Employee $_{Dno = Dnumber}$ Department)

Data Transfer Costs of Distributed Query Processing (2)

- **Result**
  - The result of this query will have 10,000 tuples, assuming that every employee is related to a department.
  - Suppose each result tuple is 40 bytes long. The query is submitted at site 3 and the result is sent to this site.
  - Problem: Employee and Department relations are not present at site 3.

Query Processing Strategies

- **Strategies:**

1. Transfer Employee and Department to site 3.
   - Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3.
   - Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
3. Transfer Department relation to site 1, execute the join at site 1, and send the result to site 3.
   - Total bytes transferred = 400,000 + 3500 = 403,500 bytes.

- Optimization criteria: minimizing data transfer.

Query Processing Strategies

- **Strategies:**

1. Transfer Employee and Department to site 3.
   - Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3.
   - Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
3. Transfer Department relation to site 1, execute the join at site 1, and send the result to site 3.
   - Total bytes transferred = 400,000 + 3500 = 403,500 bytes.

- Optimization criteria: minimizing data transfer. (**Preferred approach: Strategy 3.**)

Another Example

- Consider the query
  - Q’: For each department, retrieve the department name and the name of the department manager

Result

- The result of this query will have 100 tuples, assuming that every department has a manager.

- Execution strategies:
  1. Transfer Employee and Department to the result site and perform the join at site 3.
     - Total bytes transferred = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = 40 * 100 = 4000 bytes.
     - Total transfer size = 4000 + 1,000,000 = 1,004,000 bytes.
  3. Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
     - Total transfer size = 4000 + 3500 = 7500 bytes.

Result (2)

- The result of this query will have 100 tuples, assuming that every department has a manager.

- Execution strategies:
  1. Transfer Employee and Department to the result site and perform the join at site 3.
     - Total bytes transferred = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = 40 * 100 = 4000 bytes.
     - Total transfer size = 4000 + 1,000,000 = 1,004,000 bytes.
  3. Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
     - Total transfer size = 4000 + 3500 = 7500 bytes.

- **Preferred strategy: Choose strategy 3.**

Modifications in the Scenario.

- Now suppose the result site is 2.

  What are the possible strategies?

Modifications in the Scenario.

- Now suppose the result site is 2. Possible strategies:
  1. Transfer Employee relation to site 2, execute the query and present the result to the user at site 2.
     - Total transfer size = 1,000,000 bytes for both queries Q and Q’.
  2. Transfer Department relation to site 1, execute join at site 1 and send the result back to site 2.
     - Total transfer size for Q = 400,000 + 3500 = 403,500 bytes and for Q’ = 4000 + 3500 = 7500 bytes.

Practical Research Assignment. (30 minutes)

Examine the data stores supported by each of the following Platform as a Service (PaaS) providers:

- **AWS (Amazon Elastic Beanstalk);** (Students with last names starting with letters A-I)
- **Google App Engine.** (Students with last names starting with letters J-R)
- **Microsoft Azure.** (Students with last names starting with letters S-Z)

Extra: Try to provide performance and cost analysis. For cost analysis you can analyse the costs associated with using different data stores on each platform. You can identify pricing models and calculate the total cost of ownership (TCO).

Reading Material

- Fundamentals of Database Systems. Ramez Elmasri and Shamkant B. Navathe. Pearson. **Chapter 23.**

Q & A