University of Windsor

06 Transparencies in a DDBMS

Kalyani Selvarajah
School of Computer Science
University of Windsor

Advanced Database Topics
COMP 8157 01
Fall 2023

Announcement

Milestone I: October 29/30/31

Today’s Agenda

Various type of Transparencies

https://domains.upperlink.ng/elementor-947/

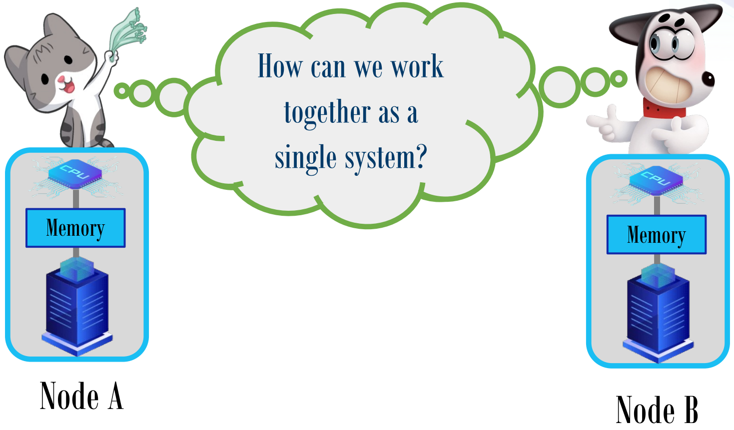
Introductory Questions

What do you mean by transparencies in DDBMS?

What are the different type of transparencies in DDBMS?

Introduction

- From the definition of the DDBMS, the system is expected to make the distribution transparent (invisible) to the user.
  - A distributed database is split into fragments that can be stored on different computers and perhaps replicated, should be hidden from the user.
- The system should make the distribution transparent to the user.
  - The objective of transparency is to make the distributed system appear like a centralized system. This is sometimes referred to as the fundamental principle of distributed DBMSs.



Main types of transparency in a DDBMS

Distribution

Transaction

Performance

DBMS

1. Distribution Transparency

Distribution transparency allows the user to perceive the database as a single, logical entity.

If a DDBMS exhibits distribution transparency, then the user does not need to know the data is fragmented (fragmentation transparency) or the location of data items (location transparency).

Fragmentation Transparency

The end user or programmer does not need to know that a database is partitioned.

Location Transparency

The end user or programmer does not need to know the location of data items.

Replication Transparency

The user is unaware of the replication of fragments.

Local Mapping Transparency

The user needs to specify both fragment names and the location of data items, taking into consideration any replication that may exist.

Naming Transparency

The DDBMS must ensure that no two sites create a database object with the same name.

Distribution Transparency : Example

Consider the following example to illustrate these concepts;

$S_1 = \prod_{\text{staffNo, position, sex, DOB, salary}}(\text{Staff})$ located at site 5

$S_2 = \prod_{\text{staffNo, fName, lName, branchNo}}(\text{Staff})$

$S_{21} = \sigma_{\text{branchNo='B003'}}(S_2)$ located at site 3

$S_{22} = \sigma_{\text{branchNo='B005'}}(S_2)$ located at site 5

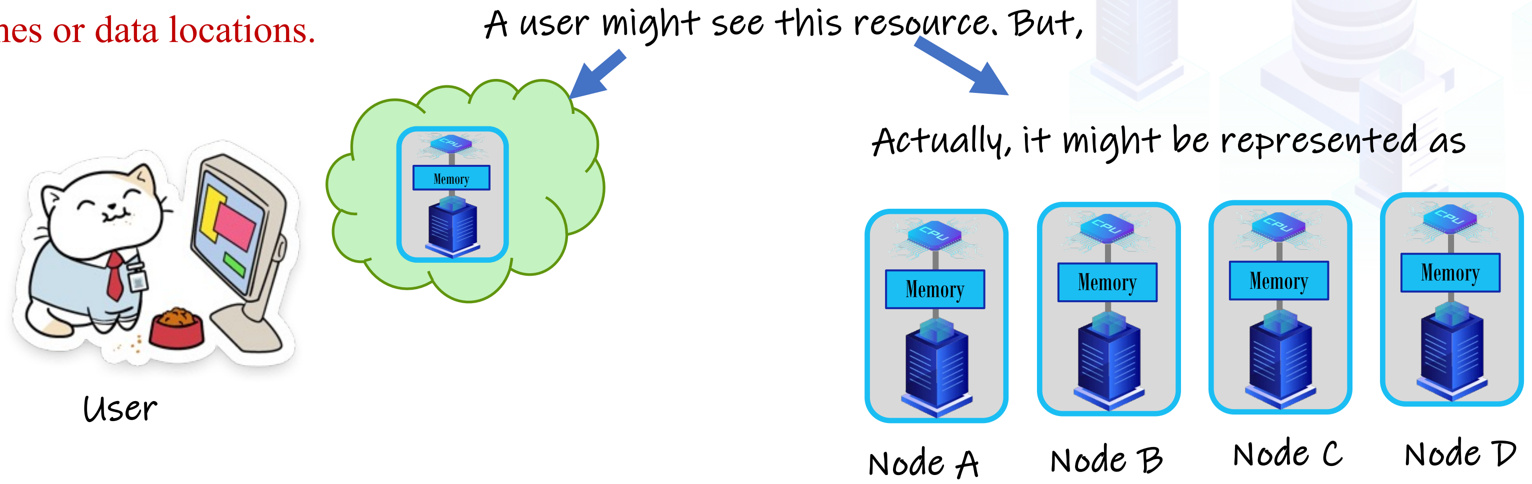
$S_{23} = \sigma_{\text{branchNo='B007'}}(S_2)$ located at site 7

Fragmentation transparency

Fragmentation is the highest level of distribution transparency.

If fragmentation transparency is provided by the DDBMS, then the user does not need to know that the data is fragmented.

As a result, database accesses are based on the global schema, so the user does not need to specify fragment names or data locations.



Fragmentation transparency

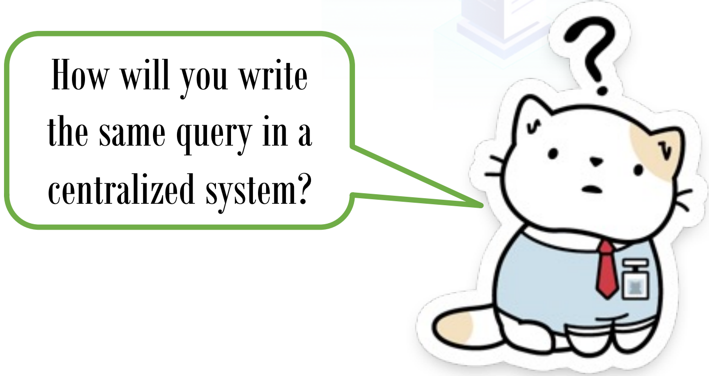
Fragmentation is the highest level of distribution transparency.

If fragmentation transparency is provided by the DDBMS, then the user does not need to know that the data is fragmented.

As a result, database accesses are based on the global schema, so the user does not need to specify fragment names or data locations.

For example, to retrieve the names of all Managers, with fragmentation transparency we could write:

```
SELECT fName, lName
FROM Staff
WHERE position = 'Manager';
```



Location transparency

Location is the middle level of distribution transparency.

With location transparency, the user must know how the data has been fragmented but still does not have to know the location of the data.

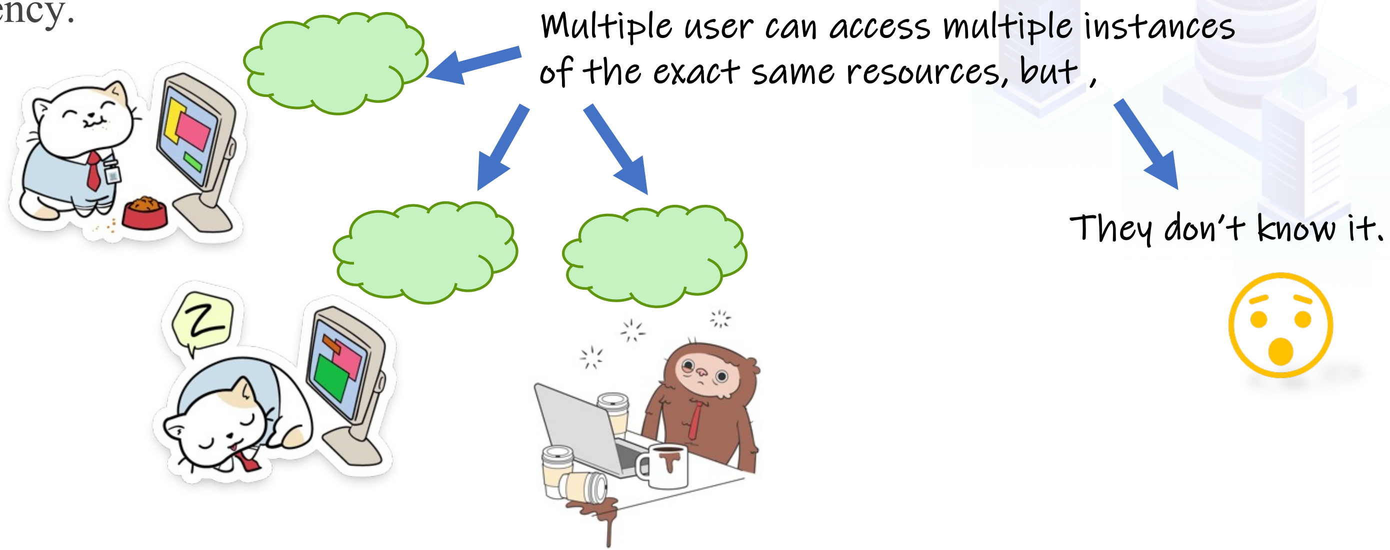
SELECT fName, lName **FROM** S_{21}
WHERE staffNo **IN** (**SELECT** staffNo **FROM** S1 **WHERE** position =‘Manager’)
UNION
SELECT fName, lName **FROM** S_{22}
WHERE staffNo **IN** (**SELECT** staffNo **FROM** S1 **WHERE** position = ‘Manager’)
UNION
SELECT fName, lName **FROM** S_{23}
WHERE staffNo **IN** (**SELECT** staffNo **FROM** S1 **WHERE** position = ‘Manager’);

Replication transparency

The user is unaware of the replication of fragments.

Replication transparency is implied by location transparency.

However, it is possible for a system not to have location transparency but to have replication transparency.



Local mapping transparency

The lowest level of distribution transparency.

The user needs to specify both fragment names and the location of data items, taking into consideration any replication that may exist.

SELECT fName, lName
FROM S_{21} *AT SITE* 3
WHERE staffNo **IN** (**SELECT** staffNo **FROM** S_1 *AT SITE* 5 **WHERE** position = ‘Manager’) **UNION**
SELECT fName, lName
FROM S_{22} *AT SITE* 5
WHERE staffNo **IN** (**SELECT** staffNo **FROM** S_1 *AT SITE* 5 **WHERE** position = ‘Manager’) **UNION**
SELECT fName, lName
FROM S_{23} *AT SITE* 7
WHERE staffNo **IN** (**SELECT** staffNo **FROM** S_1 *AT SITE* 5 **WHERE** position =‘Manager’);

Naming transparency (1/3)

Each item in a DDB must have a unique name.

DDBMS must ensure that no two sites create a database object with same name.

Solution: create a central name server.

However, this results in:

- loss of some local autonomy;
- central site may become a bottleneck;
- low availability; if the central site fails, remaining sites cannot create any new objects.

Naming transparency (2/3)

Alternative solution - prefix object with identifier of site that created it.

For example,

- Branch created at site S1 might be named S1.BRANCH.
- Also need to identify each fragment and its copies.
- Thus, copy 2 of fragment 3 of Branch created at site S1 might be referred to as S1.BRANCH.F3.C2.
- However, this results in loss of distribution transparency.

To resolve the problems with both these solutions, use **aliases** for each database object.

For example,

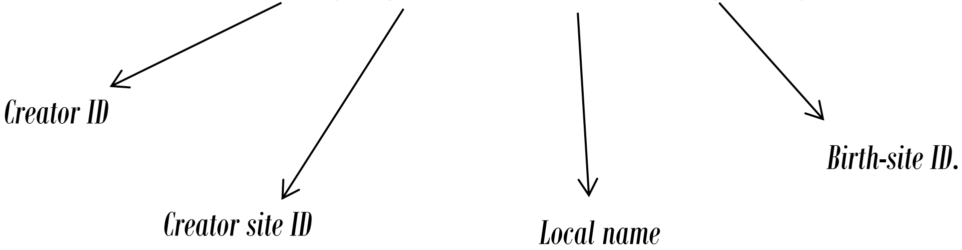
- S1.Branch.F3.C2 might be known as LocalBranch by the user at site S1.
- DDBMS has task of mapping an alias to appropriate database object.

Naming transparency (3/3)

The distributed **system R*** distinguishes between an object's printname and its systemwide name.

- ✓ The printname is the name that the users normally use to refer to the object.
- ✓ The systemwide name is a globally unique internal identifier for the object that is guaranteed never to change.
- ✓ For example, the systemwide name:

Manager@London.LocalBranch@Glasgow



2. Transaction Transparency

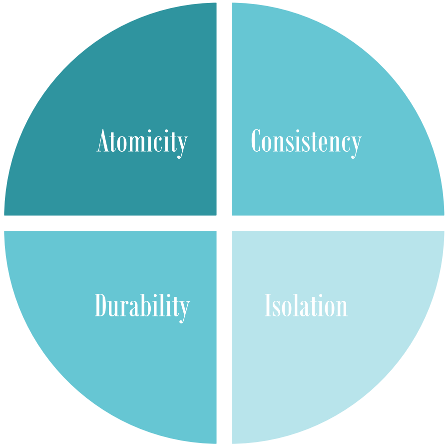
Ensures that all distributed transactions maintain distributed database's integrity and consistency.

Distributed transaction accesses data stored at more than one location.

Each transaction is divided into number of sub-transactions, one for each site that has to be accessed.

DDBMS must ensure the indivisibility of both the global transaction and each of the sub-transactions

Properties of Transactions:



Properties of Transactions

Atomicity: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

Consistency: Data is in a consistent state when a transaction starts and when it ends.

Isolation: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.

Durability: After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

Transaction Transparency: Example

Consider a transaction T that prints out names of all staff, using schema defined above as S_1, S_2, S_{21}, S_{22}, and S_{23}. Define three sub-transactions T_{S3}, T_{S5}, and T_{S7} to represent agents at sites 3, 5, and 7.

Time	T_{S_3}	T_{S_4}	T_{S_7}
t_1	begin_transaction	begin_transaction	begin_transaction
t_2	read(fName, lName)	read(fName, lName)	read(fName, lName)
t_3	print(fName, lName)	print(fName, lName)	print(fName, lName)
t_4	end_transaction	end_transaction	end_transaction

Distributed transaction.

Transaction transparency in a DDBMS is complicated by the fragmentation, allocation, and replication schemas.

We consider two further aspects of transaction transparency: concurrency transparency and failure transparency

Concurrency Transparency

All transactions must execute independently and be logically consistent with results obtained if transactions executed one at a time, in some arbitrary serial order.

Same fundamental principles as for centralized DBMS.

DDBMS must ensure

- ✓ both global and local transactions do not interfere with each other.
- ✓ consistency of all sub-transactions of global transaction.

Replication makes concurrency more complex.

- ✓ If a copy of a replicated data item is updated, update must be propagated to all copies.
- ✓ Could propagate changes as part of original transaction, making it an atomic operation.
- ✓ However, if one site holding copy is not reachable, then transaction is delayed until site is reachable.

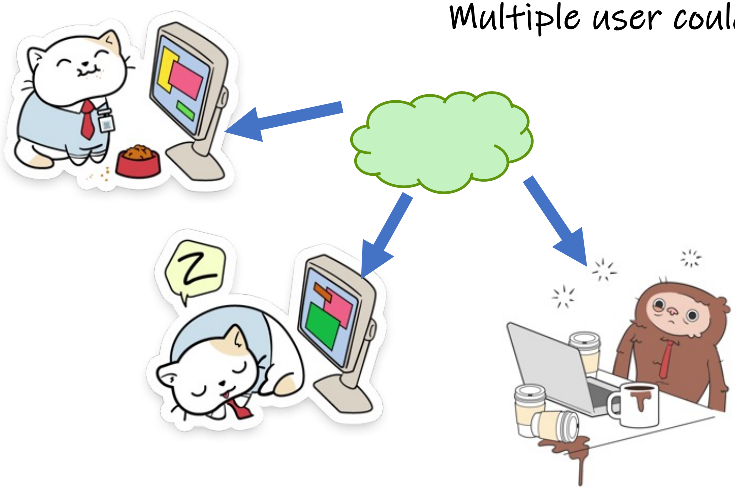
Concurrency Transparency

Solution to handle Replication smoothly:

Could limit update propagation to only those sites currently available. Remaining sites updated when they become available again.

Could allow updates to copy to happen asynchronously, sometime after the original update. Delay in regaining consistency may range from a few seconds to several hours.

Multiple user could use and share a recourse, but ,



They don't know it.

Failure Transparency

In the distributed environment, the DDBMS must also cater for:

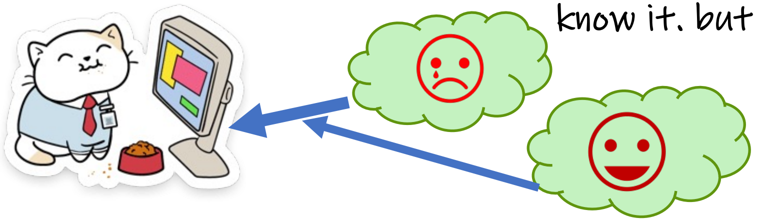
- ✓ the loss of a message;
- ✓ the failure of a communication link;
- ✓ the failure of a site;
- ✓ network partitioning.

DDBMS must ensure atomicity and durability of global transaction.

- ✓ Means ensuring that sub-transactions of global transaction either all commit or all abort.

Thus, DDBMS must synchronize global transaction to ensure that all sub-transactions have completed successfully before recording a final COMMIT for global transaction.

When part of a system fails, user doesn't know it. but ,

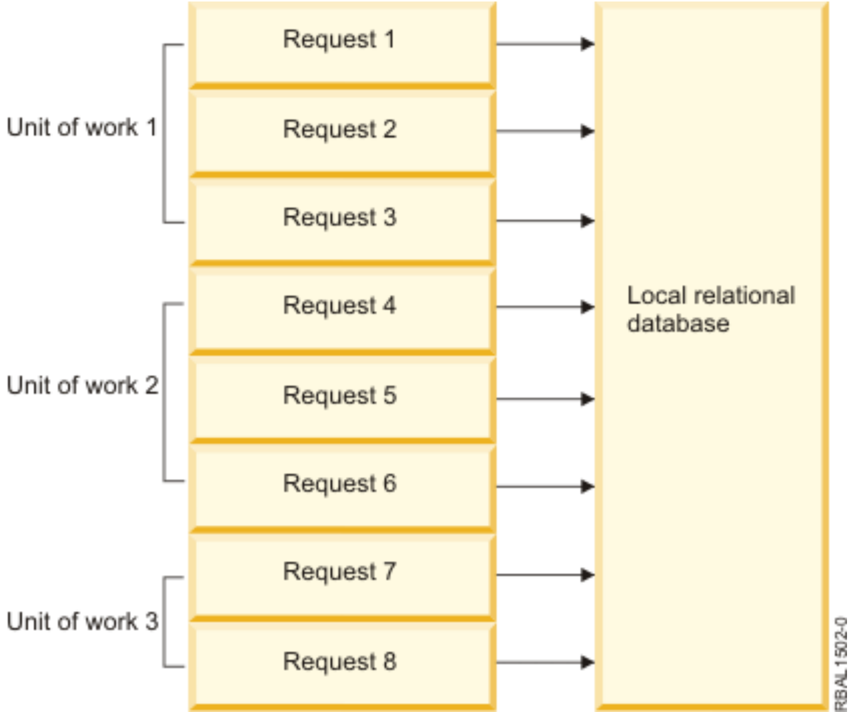


They can still access the resources.

Classification of transactions

IBM's Distributed Relational Database Architecture (DRDA) defined four types of transaction:

Unit of work in a local relational database

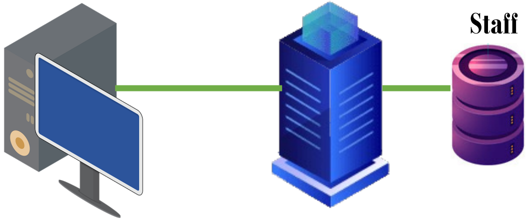


A unit of work (UOW) is a single logical transaction. It consists of a sequence of SQL statements in which either all of the operations are successfully performed or the sequence as a whole is considered unsuccessful.

Classification of transactions

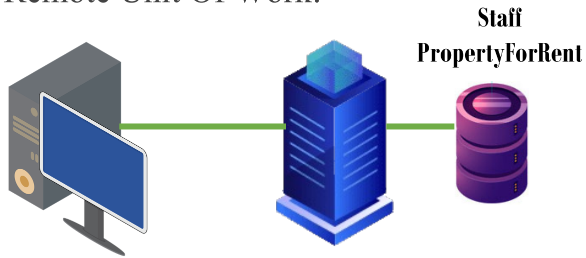
IBM's Distributed Relational Database Architecture (DRDA) defined four types of transaction:

1. Remote Request:



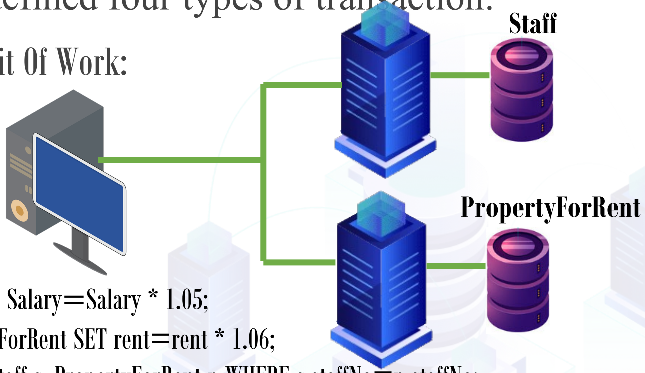
SELECT * FROM Staff WHERE Salary>20000

2. Remote Unit Of Work:



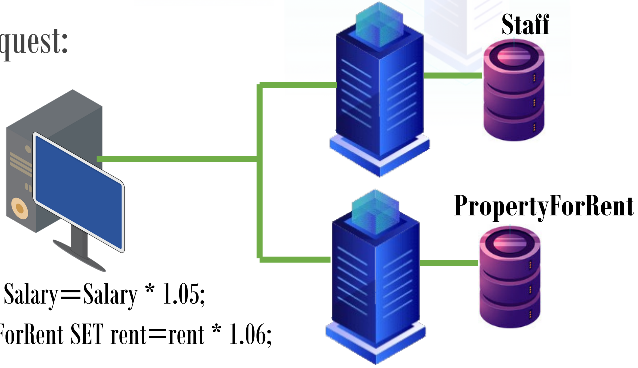
UPDATE Staff SET Salary=Salary * 1.05;
UPDATE PropertyForRent SET rent=rent * 1.06;
COMMIT;

3. Distributed Unit Of Work:



UPDATE Staff SET Salary=Salary * 1.05;
UPDATE PropertyForRent SET rent=rent * 1.06;
SELECT * FROM Staff s, PropertyForRent p WHERE s.staffNo=p.staffNo;
COMMIT;

4. Distributed Request:



UPDATE Staff SET Salary=Salary * 1.05;
UPDATE PropertyForRent SET rent=rent * 1.06;
COMMIT;

Performance Transparency

DDBMS must perform as if it were a centralized DBMS.

- ✓ DDBMS should not suffer any performance degradation due to distributed architecture.
- ✓ DDBMS should determine most cost-effective strategy to execute a request

Distributed Query Processor (DQP) maps data request into ordered sequence of operations on local databases.

Must consider fragmentation, replication, and allocation schemas.

DQP has to decide:

- ✓ which fragment to access;
- ✓ which copy of a fragment to use;
- ✓ which location to use.

Performance Transparency

The DQP produces an execution strategy that is optimized with respect to some cost function.

Typically, the costs associated with a distributed request include:

- ✓ the access time (I/O) cost involved in accessing the physical data on disk;
- ✓ the CPU time cost incurred when performing operations on data in main memory;
- ✓ the communication cost associated with the transmission of data across the network.

Performance Transparency – Example

Property(propNo, city)	10000 records in London
Client(clientNo,maxPrice)	100000 records in Glasgow
Viewing(propNo, clientNo)	1000000 records in London

Query: List the properties in Aberdeen that have been viewed by clients who have a maximum price limit greater than £200,000

```
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;
```

Assume:

Each tuple in each relation is 100 characters long.

10 renters with maximum price greater than £200,000.

100 000 viewings for properties in Aberdeen.

Computation time negligible compared to communication time.

Performance Transparency – Example

Strategy 1:	Move the Client relation to London and process query there: Time = 1+ (100 000 * 100/10 000) ≅ 16.7 minutes
Strategy 2:	Move the Property and Viewing relations to Glasgow and process query there: Time = 2+ [(1 000 000 + 10 000) * 100/10 000] ≅ 28 hours
Strategy 3:	Join the Property and Viewing relations at London, select tuples for Aberdeen properties and then for each of these tuples in turn check at Glasgow to determine whether the associated client's maxPrice > £200,000. The check for each tuple involves two messages: a query and a response. Time = 100 000 * (1 + 100/10 000) + 100 000 * 1 ≅ 2.3 days
Strategy 4:	Select clients with maxPrice > £200,000 at Glasgow and for each one found, check at London to see whether there is a viewing involving that client and an Aberdeen property. Again, two messages are needed: Time = 10 * (1 + 100/10 000) + 10 * 1≅20 seconds
Strategy 5:	Join Property and Viewing relations at London, select Aberdeen properties, project result over propertyNo and clientNo, and move this result to Glasgow for matching with maxPrice > £200,000. For simplicity, we assume that the projected result is still 100 characters long: Time = 1 + (100 000 * 100/10 000) ≅ 16.7 minutes
Strategy 6:	Select clients with maxPrice > £200,000 at Glasgow and move the result to London for matching with Aberdeen properties: Time = 1 + (10 * 100/10 000) ≅ 1 second

4. DBMS Transparency

DBMS transparency hides the knowledge that the local DBMSs may be different and is therefore applicable only to heterogeneous DDBMSs.

It is one of the most difficult transparencies to provide as a generalization.

Summary

Distribution Transparency : Fragmentation, Location, Replication, Local Mapping and Naming

Transaction Transparency: Concurrency Transparency , Failure Transparency

Performance Transparency

DBMS Transparency