# CST3604 Lecture Notes

## Physical Design

## Physical Data Model – Distributed Design

## (Part 1 of ?)

### (Lecture Notes 3?)

Prof. Abel Angel Rodriguez

# Chapter 3 Distributed Database Design

## 3.1 Review of Basic Distributed Database Concepts

### 3.1.1 Review of Centralize and Distributed Model

- In the last lecture, we introduce the concept of Distributed Database Model by comparing it to the Centralized Model.
- Before is a short review of the concept

  - Before we begin, we will use the Manufacturing Company data model use case from Malyuta's book. Using the following tables:

**Title (titleCode, titleDescription, salary)**

**Department (deptCode, deptName, location, deptType)**

**Employee (ID, emplName, emplType, *deptCode*, *titleCode*)**

  - And the following data:

Title

| Title Code | Title Description | salary |
|---|---|---|
| T1 | Accountant | 10000 |
| T2 | Analyst | 20000 |
| T3 | Programmer | 30000 |
| T4 | DBA | 40000 |
| T5 | Manager | 50000 |

Department

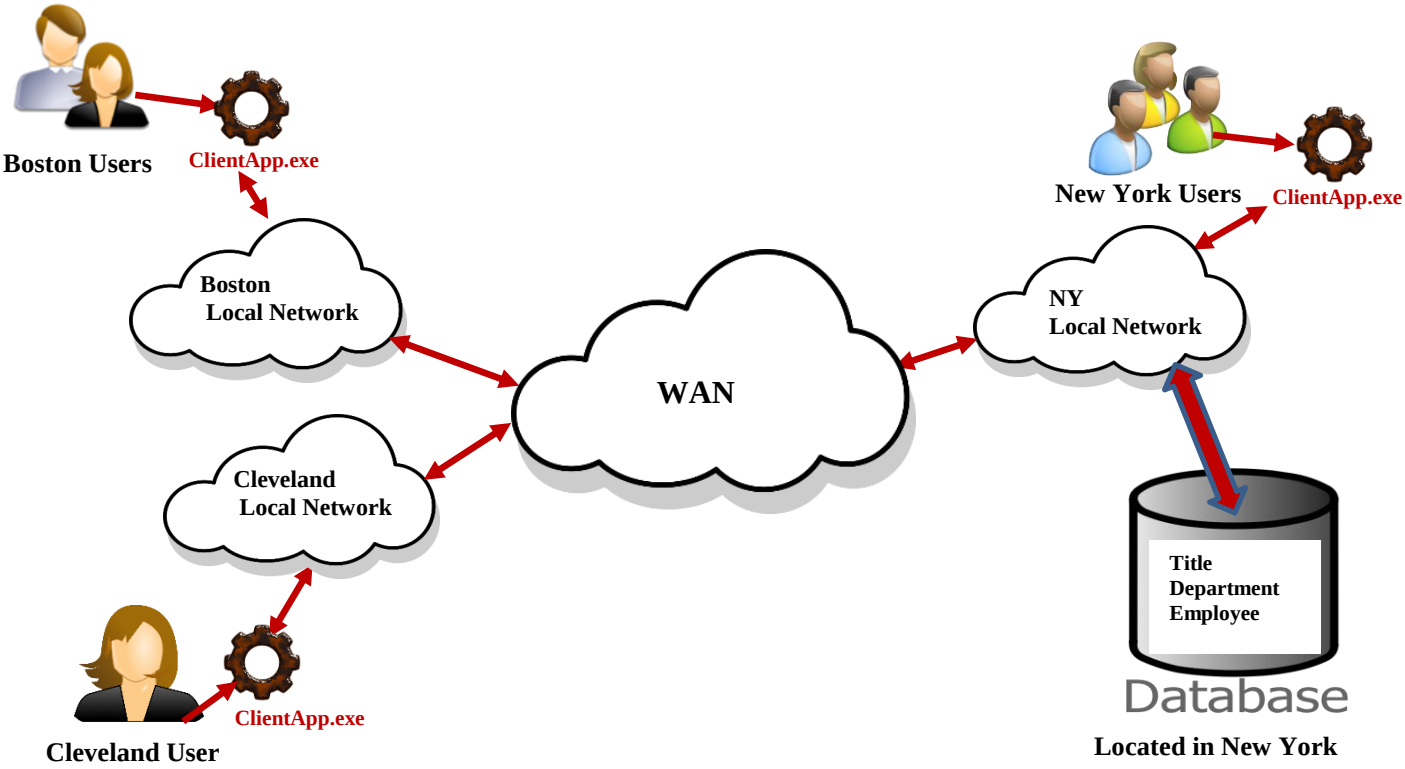
| Dept Code | Dept Name | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 002 | Budget | New York | Business |
| 003 | Marketing | Boston | Marketing |
| 004 | Database Support | Cleveland | IT |
| 005 | Purchasing | New York | Business |

Employee

| ID | Empl Name | Empl Type | Dept Code | Title Code |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 2 | Adam | Consultant | 001 | T3 |
| 3 | Mary | Part-time | 004 | T4 |
| 4 | Peter | Full-time | 003 | T2 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |
| 7 | Alex | Part-time | 004 | T2 |

## Centralized Model

- Diagram below illustrates an example of a possible **centralize model** for this manufacturing company:




- Properties and pros & cons of the centralized model:
  - Database resides in NY, all users that do not reside in the New York office must traverse the network from a greater distance thus may incur latency, e.g.: Boston and Cleveland users.
  - New York users are local or in the same office as the database applications therefore may not experience any latency.
  - Reliability & availability is at risk for Boston & Cleveland users. Any networking issues with the WAN will affect them. Where as for NY users, they may reside in the same local network as the centralized database and won't be affected.

## The Distributed Database Model

- Diagram below illustrates a possible **Distributed Database model** for this manufacturing company from Malyuta's book :

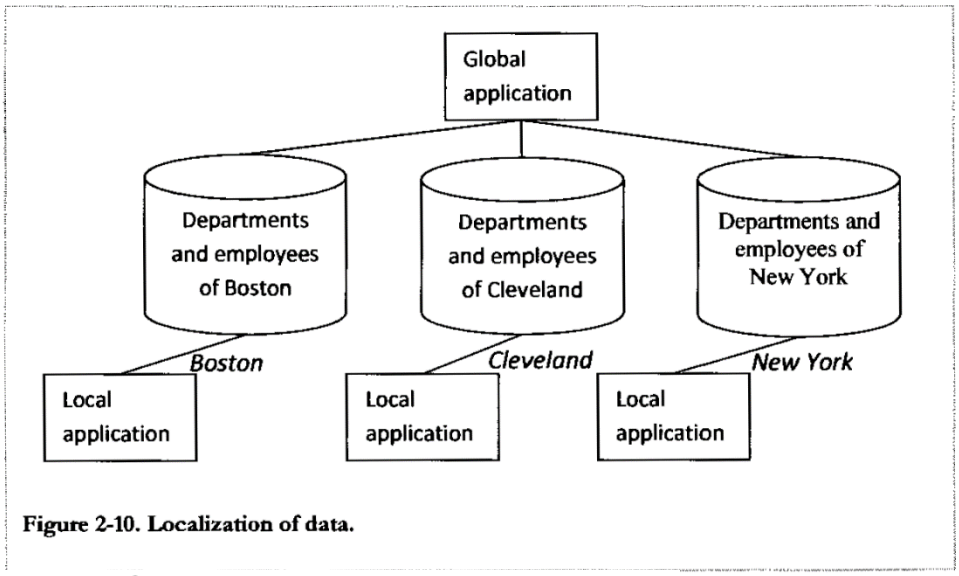
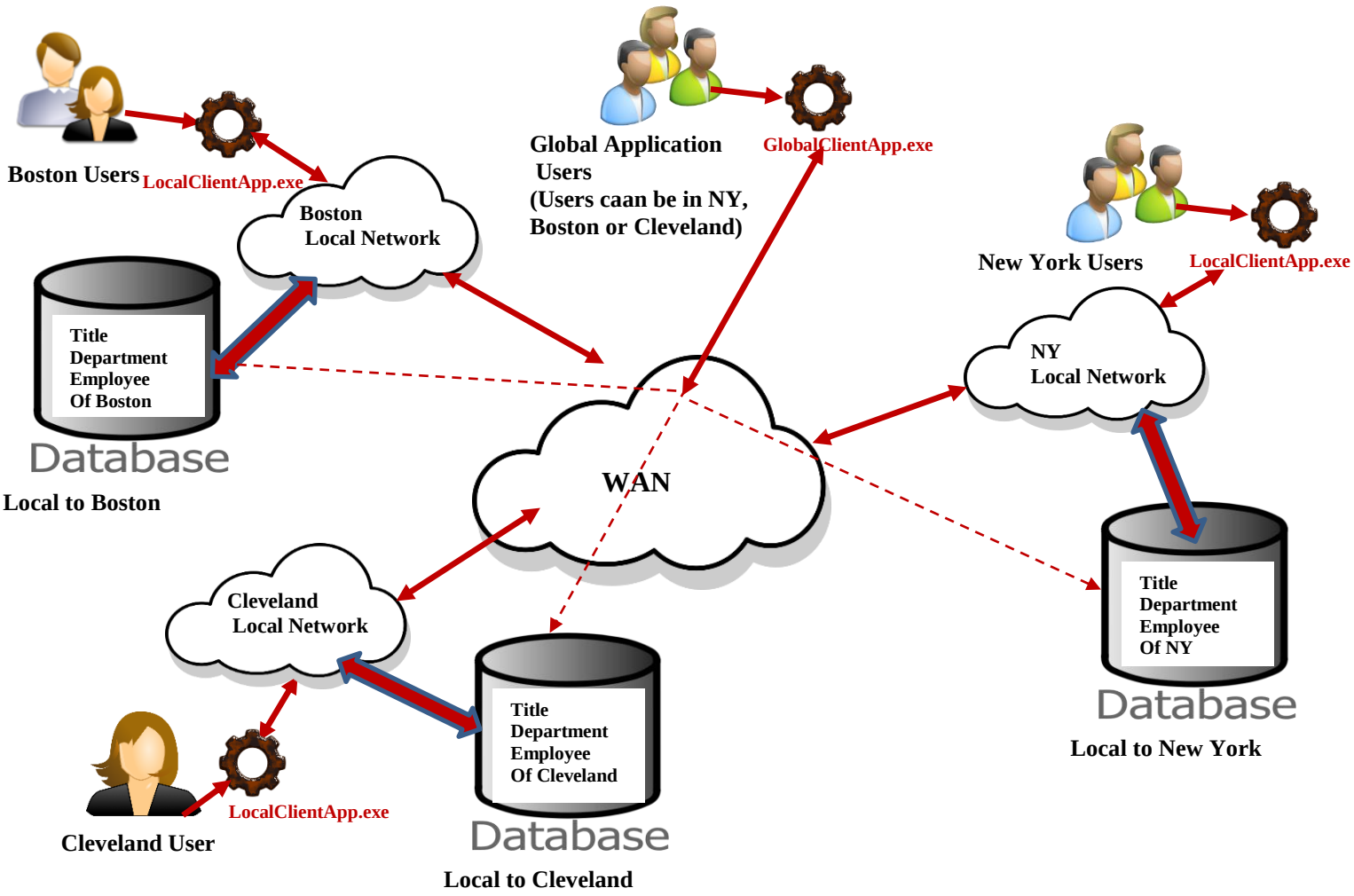



**Figure 2-10. Localization of data.**



- Diagram below illustrates a detailed example of a possible **Distributed Database model** for this manufacturing company:



## Characteristics of the Distributed Data Model

- Characteristics of the distributed model:

  - Data is physically distributed in several databases across different locations.

    - This means you will have database servers in every location with data in each location for only the users in that location.
    - Tables are replicated in each location but only the data required by the users in that location. Database in NY contains NY data only, Boston database, Boston data only an Cleveland database Cleveland data only.
    - **Fragments** – Parts of the tables can be distributed, not just the entire table.
    - **Fragmentation** – Process of splitting the tables into different fragments. We will analyze the various methods to do this in the next chapter

  - **Autonomous** – Each database is autonomous & supported independently of the others.

  - **Homogeneous** – When all databases are using the same DBMS. Usually the case in most implementation.

  - **Heterogeneous** – When all databases are using different DBMS. Complex.

  - **Global Application** – A requirement of the distributed model to include at least one application that uses data from all the different databases.

  - **Local Application** – Local application only work on data from the local database, where data only needed by that location exists.

  - **Connected Physically via a Network** – In order to realize a distributed model, obviously, all databases must be connected via a network.

  - **Look like a centralized database to users** – The distributed database must appear like a centralized database to the users.

## 3.1.2 Review of Distributed Database Model in Oracle

- Oracle provided support for distributed database.
- We will look at this in more details in the next lecture, but now we will look at an overview how is done in Oracle.

### Oracle Database Link

- Let's look at one of the main component in s:
  - **Database Link**:
    - A one-way communication link from one oracle database to another.
    - It is a logical connection between database via the physical network.
- MODIFIED illustration from Malyuta' s book below for our Manufacturing Company example:

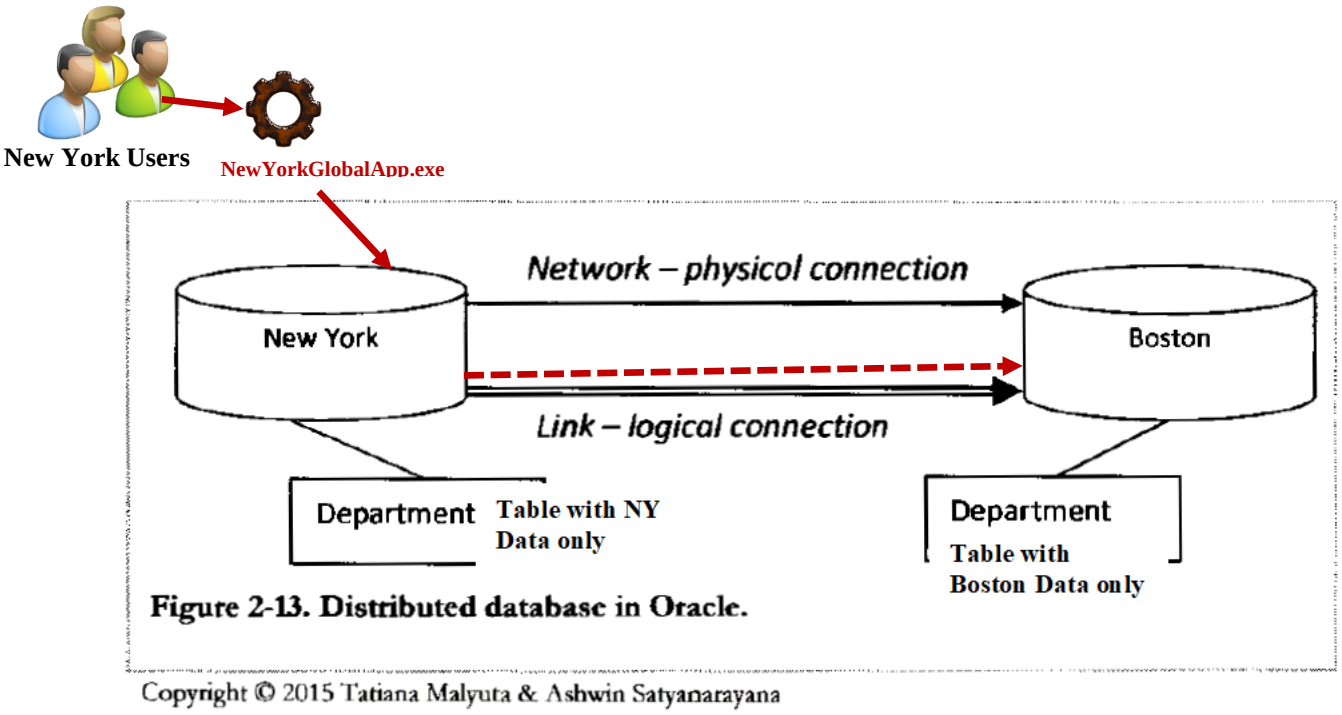



**Figure 2-13. Distributed database in Oracle.**



### Simple Syntax to create Database Link

- A **Database Link** is a **SCHEMA OBJECT** in Oracle:
  - This means when you create a Database Link and Object will exist in your Oracle schema.
  - The syntax to creating a Database Link has many options, I will show the simple option below. The syntax is as follows:

```
CREATE PUBLIC|SHARED DATABASE LINK
dbLink_name|dbLink_name.DomainServiceName
USING
remote_DB_connection_string_name|remote_DB_connection_string_nam
e.DomainServiceName
```

## Using the DATABASE LINK after creation

- Once the **Database Link** is created an a **SCHEMA OBJECT** exists, you can then use it:

  - Create your queries using the name of the link after a TABLE NAME using @ symbol:

```
--Select Query
SELECT *
FROM DEPARTMENT@BostonConn;
```

    - We assume **BostonConn** is the Database Link name based on the following DATABASE LINK created:

```
CREATE PUBLIC DATABASE LINK BostonConn USING
boston.ourcompany.us.com
```

  - You can use any of the SQL STATEMENTS to perform operation on the remote database once the Database Link is created and you use the @ symbol after a TABLE NAME.

# 3.2 General Approaches to Implementing Distributed Model

## 3.2.1 Fragmentation & Types of Fragmentation

- The idea of the distributed model is to **fragment** tables in different location, NOT place entire tables in different databases.
  - **Fragments** – Parts of the tables are distributed, not just the entire table.
  - **Fragmentation** – Process of splitting the tables into different fragments. We will analyze the various methods to do this in the next chapter

- Keep in mind that the main goal of the distributed model is:
  - **Improve performance, availability & reliability through the localization of data or placing ONLY the data that is needed by each set of users in a location**.
  - **Therefore, fragmentation is the basis of the distributed model**
  - **Tables are fragmented and placed in the locations where users most often need the data of each fragment.**

- Types of fragmentation:
  - **Horizontal Fragmentation** – **ROWS** of a table are stored in different databases:

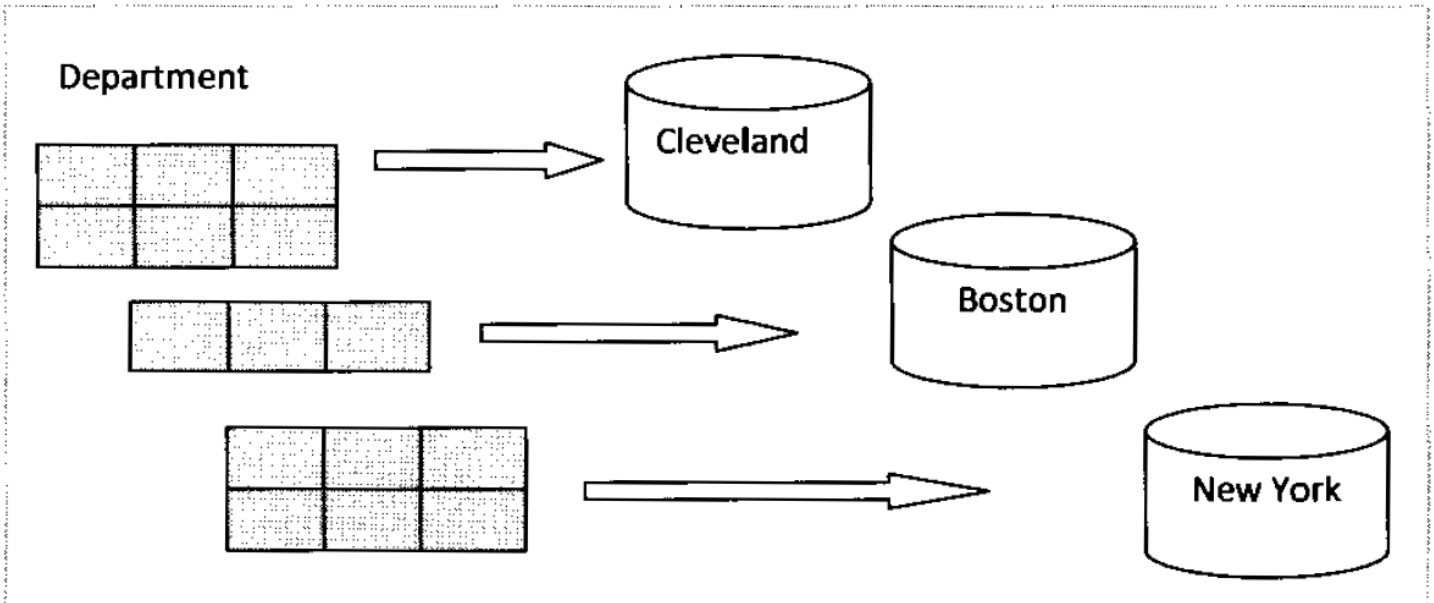



**Figure 3-3. Horizontal fragmentation.**

- **Vertical Fragmentation** – **COLUMNS** of a table are stored in different databases:

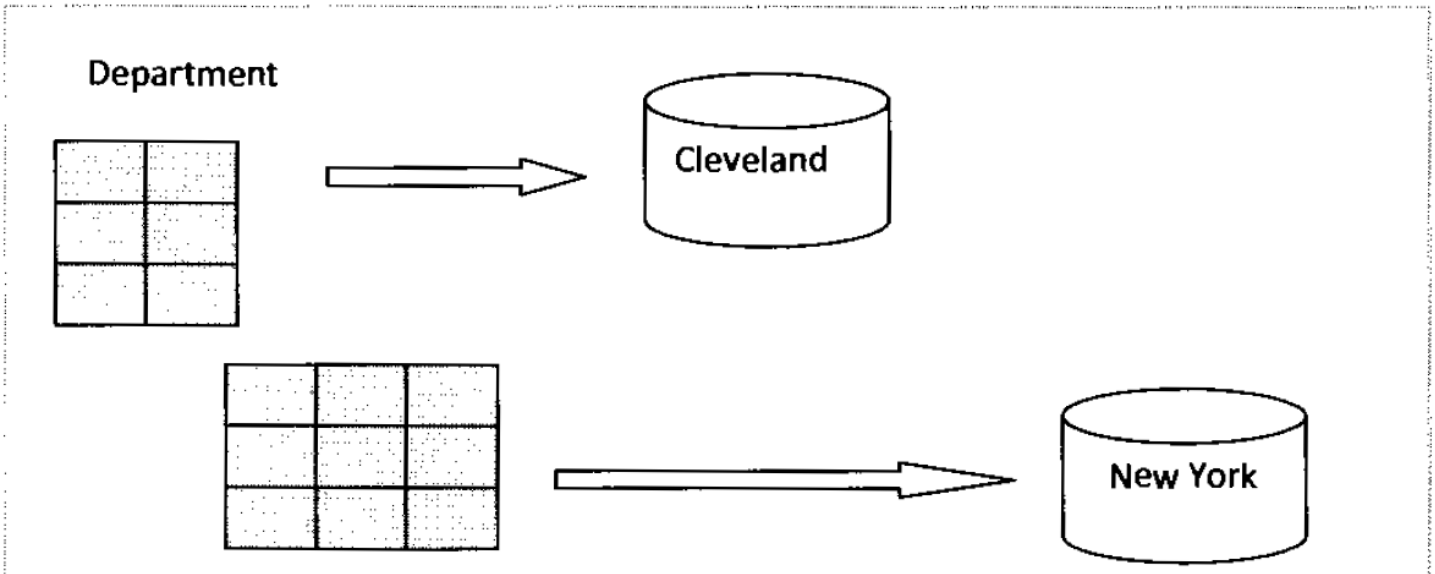



**Figure 3-4. Vertical fragmentation.**



- **Hybrid Fragmentation** – Combination of Horizontal and Vertical.

## Rules of Fragmentation

- Fragmentation must abide by the following rules:

### Completeness Rule:

- Data **cannot** be lost in the process of fragmentation
- **Each data item must be included in ONE & ONLY ONE fragment**.

### Disjoint Rule:

- Data **cannot** be duplicated in the process of fragmentation
- No duplication, so we don't have redundancy and inconsistency.
- **Each data item must be included in ONE & ONLY ONE fragment**

## 3.2.2 Horizontal Fragmentation

- **Horizontal Fragmentation** –:
  - **ROWS of a table are stored in different databases, based on information about what rows of the table are accessed & how frequently and from which location.**

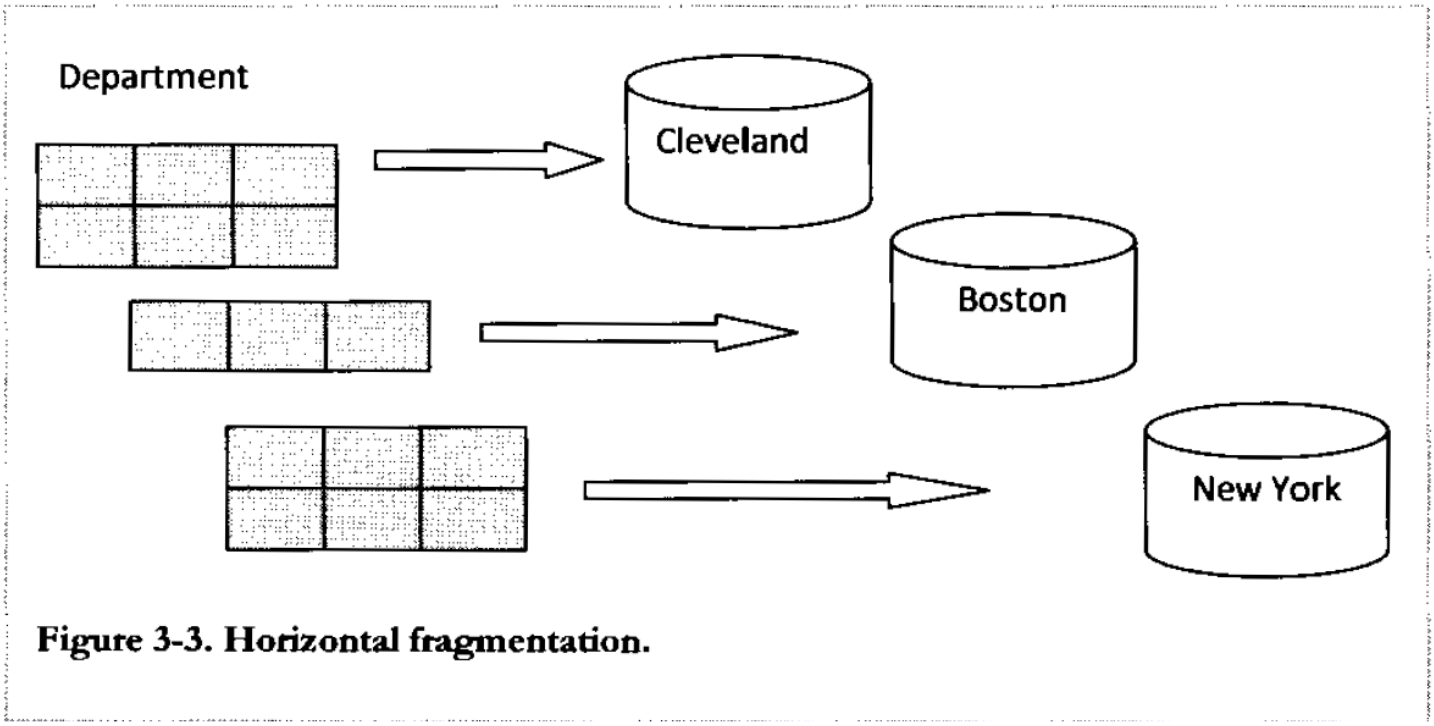



**Figure 3-3. Horizontal fragmentation.**



- Recalling our Manufacturing Company distributed database model and how tables are fragmentation across location:
  - Users in each city only work with data local to that city.

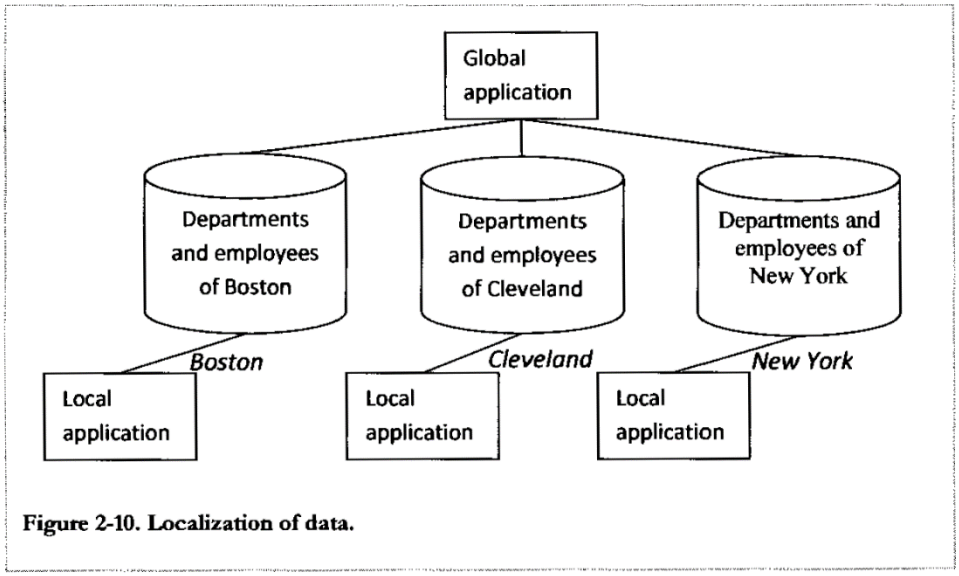



**Figure 2-10. Localization of data.**

- Malyuta's book, explains this concept with 3 user cases:
  - **Use-case #1:**
    - **In the Manufacturing Company case, users of each city work with data about local departments (the department located in that city)**
  - **Use-case #2:**
    - **There is another application in Boston that process data about all IT departments, and there is a similar application in Cleveland which works with data about the Business & Marketing departments)**
- Let's look at both of these user-cases in the next section.

## Horizontal Fragmentation for Use-case #1 & 2

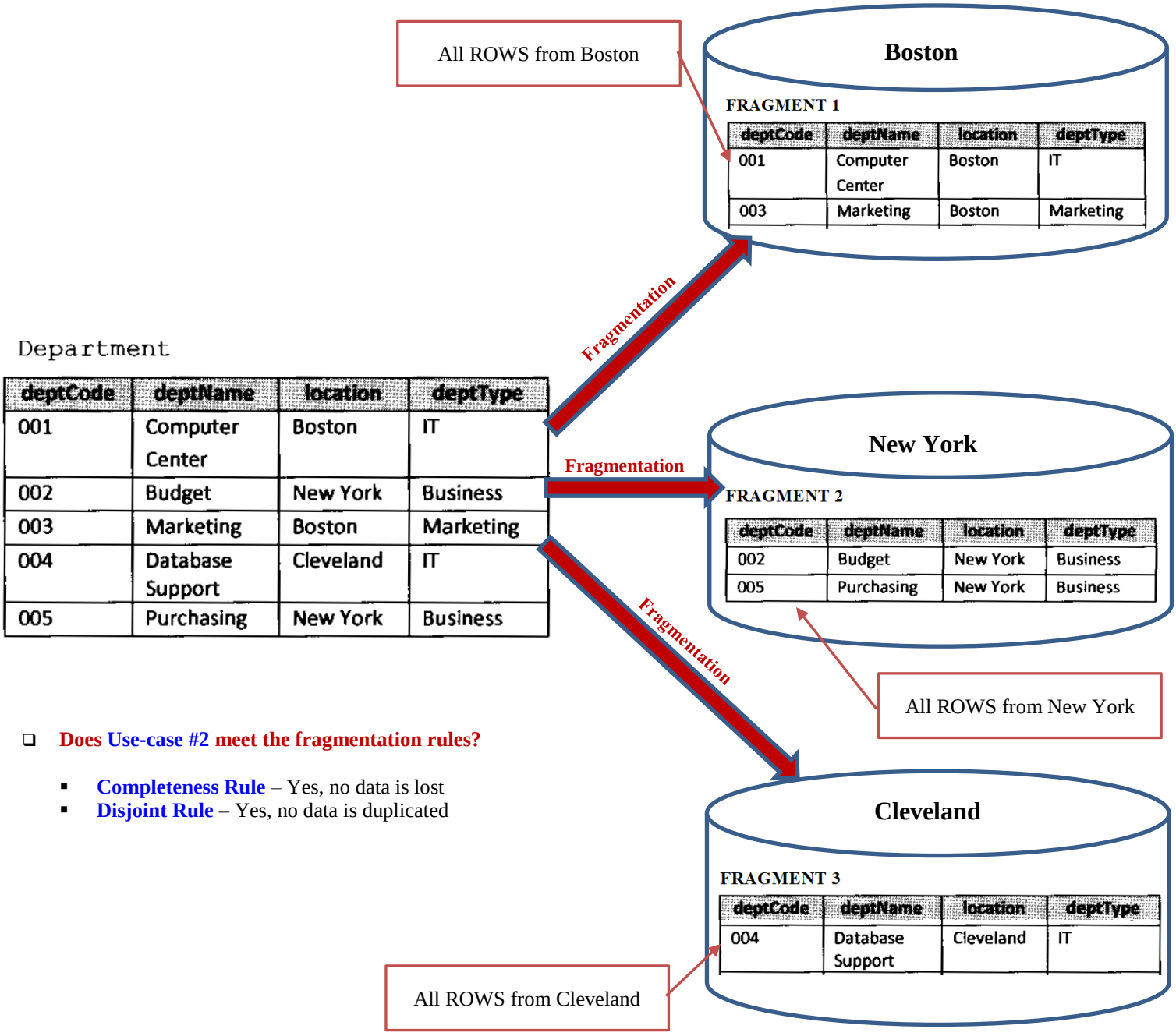
- Looking at the department table of the Manufacturing Company database:

Department

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 002 | Budget | New York | Business |
| 003 | Marketing | Boston | Marketing |
| 004 | Database Support | Cleveland | IT |
| 005 | Purchasing | New York | Business |

## Use-Case #1 Requirements & Implementation

- **Use-case #1** states the following:
  - **In the Manufacturing Company case, users of each city work with data about local departments (the department located in that city) – Assume this access takes place several thousand times per day.**
- After analysis, we reach the following conclusions:
  - We would need to fragment the table by **ROWS** into **three fragments** based on locations: New York, Boston & Cleveland.

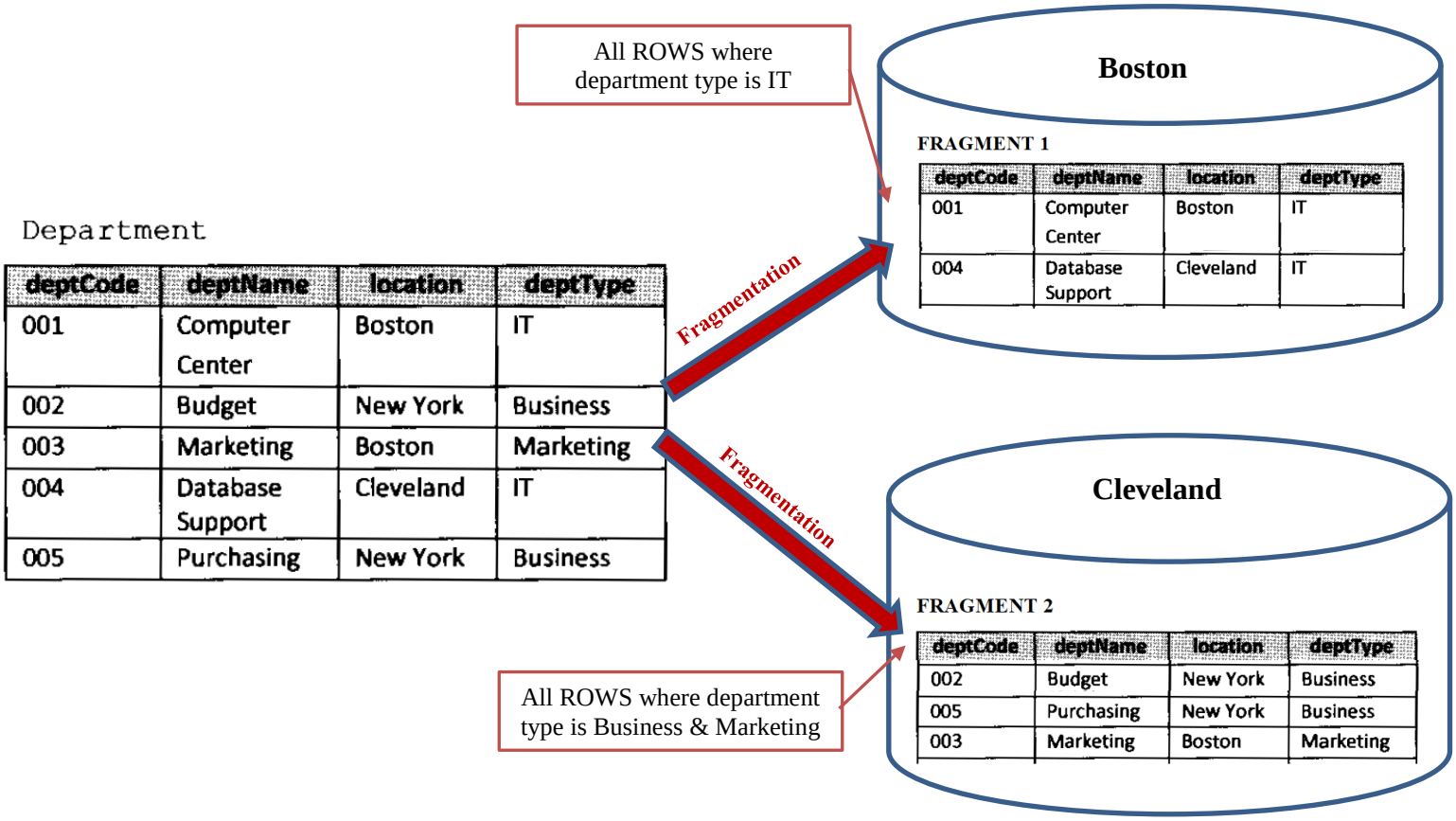



- **Does Use-case #2 meet the fragmentation rules?**
  - **Completeness Rule** – Yes, no data is lost
  - **Disjoint Rule** – Yes, no data is duplicated

## Use-Case #2 Requirements & Implementation

- **Use-case #2** states the following:
  - **There is another application in Boston that process data about all IT departments, and there is a similar application in Cleveland which works with data about the Business & Marketing department– Assume this access takes place several times per month)**

- After analysis of these two requirements in the use-case, we reach the following conclusions:
  - First Requirement – We would need to fragment the table by **ROWS** based on department type = IT department
  - Second Requirement – We would need to fragment the table by **ROWS** based on department type = Business & Marketing




| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 002 | Budget | New York | Business |
| 003 | Marketing | Boston | Marketing |
| 004 | Database Support | Cleveland | IT |
| 005 | Purchasing | New York | Business |

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 004 | Database Support | Cleveland | IT |

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 002 | Budget | New York | Business |
| 005 | Purchasing | New York | Business |
| 003 | Marketing | Boston | Marketing |

- **Does Use-case #2 meet the fragmentation rules?**
  - **Completeness Rule** – Yes, no data is lost
  - **Disjoint Rule** – Yes, no data is duplicated

## Issues with Implementing Use-Case #1 & Use-Case #2

- Use-case #2 presents a challenge:
  - **USE-CASE #1** does not provide the fragmentation needed for **USE-CASE #2** since the breakdown is department by city, where's **USE-CASE #2**needs department by department type.
  - If we fragment based on **USE-CASE #1**, this would mean the Boston & Cleveland applications would have to access REMOTE DATA therefore performance & availability would be degraded.
  - **MAIN POINT #1:**
    - The horizontal fragmentation of **USE-CASE #1** does not provided the localization of data needed for **USE-CASE #2**.
    - The horizontal fragmentation of **USE-CASE #2** does not provided the localization of data needed for **USE-CASE #1**.
  - **MAIN POINT #2:**
    - We have conflicting requirements.
    - This was mentioned as one of the challenges of the distributed model.

## Resolving the Issues with Implementing Use-Case #1 & Use-Case #2

- To resolve this issues?
  - Decide which use-case is more important
  - Follow the guidance in the definition we listed previously on Horizontal Fragmentation:

    **ROWS of a table are stored in different databases, based on information about what rows of the table are accessed & how frequently and from which location.**
  - We have three criteria:
    1. What rows
    2. How frequent
    3. Location
  - In each case, we are given information on frequency of use or how often data is accessed.
  - In this case **USE-CASE #1**, is accessed thousands of times a day, while **USE-CASE #1**, is only a few times a month.
  - Therefore, performance and availability of **USE-CASE #1** is more important and should be localize.
- The popular 80/20 rule:
  - **80% of data processing in an enterprise is done by 20% of the applications.**
  - **Therefore, the needs of the applications within the 20% have priority!**

## Primary Horizontal Fragmentation & Relational Algebra

- **Primary Horizontal Fragmentation** defined:
  - **Fragmentation is based on:**
    - **What rows are accessed**
    - **From which site or location**
    - **And how frequent**
- **Use-case #1** is an example of **Primary Horizontal Fragmentation** since it states the following based on location & frequency:
  - **In the Manufacturing Company case, users of each city work with data about local departments (the department located in that city) – Assume this access takes place several thousand times per day.**
- Access is based on:
  - Location = 3 (NY, Boston & Cleveland)
  - The query **condition** will be based on these 3 locations:

```
--Select Query
SELECT *
FROM DEPARTMENT
WHERE Location = ?;
```

  - This means the CONDITION (WHERE clause), which is relational algebra is known as the PREDICATE (P). So the PREDICATE would have 3 options:
    - $P_1$ = Location = "**Boston**"
    - $P_2$ = Location = "**New York**"
    - $P_3$ = Location = "**Cleveland**"
  - Using RELATIONAL ALGEBRA:
    - Before we get started on Relational Algrebra equation, keep in mind the following:

**The results of a query on a table(s)/Relation(s) is a table (Relation)**

- The expression for a query in Relational Algebra is called a SELECTION & the formula is as follows:

$$R_i = \sigma_F (R)$$

- Where:
  - $R_i$ – The resultant relation from the query (the result of a query is a table).
  - $\sigma$ – Greek letter Sigma or SELECTION SYMBOL: Represents the SELECT clause.
  - $F$– the actual filter or condition (where clause) for each table fragment (i). Note sometimes letter P for predicate is used.
  - $R$ = relation or table selection being applied

- Based on equation above, **HORIZONTAL FRAGMENTATION** is defined as follows:

$$R_i = \sigma_{Pi}(R)$$

Where $1 \leq i \leq N$

  - Where:
    - $R_i$ – HORIZONTAL FRAGMENTATION - The resultant **FRAGMENT i** that results from the SELECT query
    - $\sigma$ – Greek letter Sigma or SELECTION SYMBOL: Represents the SELECT clause.
    - $P_i$ – the predicated or actual filter or condition (where clause) for each table fragment (i)
    - $R$ = Relation or table selection being applied

- For **USE-CASE #1**, the **HORIZONTAL FRAGMENTATION** equations look as follows:
  - **Boston**

$$Department_1 = \sigma_{location = \text{'Boston'}}(Department)$$

All ROWS from Boston →

**FRAGMENT 1**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 003 | Marketing | Boston | Marketing |

  - **New York**

$$Department_2 = \sigma_{location = \text{'New York'}}(Department)$$

All ROWS from New York →

**FRAGMENT 2**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 002 | Budget | New York | Business |
| 005 | Purchasing | New York | Business |

  - **Cleveland**

$$Department_3 = \sigma_{location = \text{'Cleveland'}}(Department)$$

All ROWS from Cleveland →

**FRAGMENT 3**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 004 | Database Support | Cleveland | IT |

- For **USE-CASE #1** if you want to rebuild the original table from all it's fragments the equation is:

$$Department = Department_1 \cup Department_2 \cup Department_3$$

## Horizontal Fragmentation Use-case #3

- Again, using the department table of the Manufacturing Company database:

Department

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 002 | Budget | New York | Business |
| 003 | Marketing | Boston | Marketing |
| 004 | Database Support | Cleveland | IT |
| 005 | Purchasing | New York | Business |



  - **Use-case #3:**
    - **The users in Boston and New York work with data about local departments**
    - **Users in Cleveland work with data about local department and New York departments**

- Based on use cases, here are the three location conditions $P_1$:

$P_1$ : location = 'Boston'

$P_2$ : location = 'New York'

$P_3$ : location = 'Cleveland' OR location = 'New York'

- Based on the conditions $P_1$, $P_2$ & $P_3$ we have the following SELECTION equations and fragments:
  - Boston:

$$Department_1 = \sigma_{location = \text{'Boston'}} (Department)$$

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 003 | Purchasing | Boston | Production |

This may be an error in the book. Does not match Dept table

  - New York:

$$Department_2 = \sigma_{location = \text{'New York'}} (Department)$$

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 002 | Budget | New York | Business |
| 005 | Purchasing | New York | Business |

  - Cleveland:

$$Department_2 = \sigma_{location = \text{'Cleveland' OR location = 'New York'}} (Department)$$

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 002 | Budget | New York | Business |
| 004 | Database Support | Cleveland | IT |
| 005 | Purchasing | New York | Business |

## Violation of Disjoint Rule

- **Does Use-case #3 meet the fragmentation rules?**
  - **Completeness Rule** – Yes, no data is lost
  - **Disjoint Rule** – NO, DATA IS DUPLICATED IN NEW YORK AND CLEVELAND

- DISJOINT RULE is broken, which introduces inconsistency:
  - If you create a report listing the number of departments the results would be incorrect.

- To resolve this issue, we would have to analyze the usage or frequency of usage to determine which location gets preference:
  - If New York users use the data more frequently, then we would have to fragment Cleveland only for Cleveland and Cleveland users would have to traverse the network to access New York data.

## Derived Horizontal Fragmentation & Relational Algebra

- To understand **Derived Horizontal Fragmentation** let's look at the following scenario:
  - We have the following Parent (Department Table) & Child (Employee Table) relationship:
    - Note we assume, the Boston Department Table has been horizontally fragmented as per examples in last section

**Department**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 003 | Marketing | Boston | Production |

Employee

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 2 | Adam | Consultant | 001 | T3 |
| 3 | Mary | Part-time | 004 | T4 |
| 4 | Peter | Full-time | 003 | T2 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |
| 7 | Alex | Part-time | 004 | T2 |



  - Supposed we placed the entire EMPLOYEE table in Boston:
    - Boston users will access data about employees working in Boston. To do this, they will need to JOIN both the employee & department tables as follows:

**Department**

```
--Select Query using Joins with aliases
SELECT e*
FROM Department d, Employee e
WHERE d.deptCode = e.deptCode;
```

    - The JOIN query, will only access those rows of the table EMPLOYEE that match or are JOINED to rows of the DEPARTMENT table.
    - But the DEPARTMENT table is fragmented with only Boston data. So you are joining only to two records and the only records available in DEPARTMENT table.
    - **The POINT IS that it makes no sense to keep the ENTIRE EMPLOYEE TABLE in Boston since other rows of the EMPLOYEE table are NEVER ACCESSED by Boston users**.

- **Derived Horizontal Fragmentation** defined:

  - **When in the same application, you have a RELATIONSHIP between a PARENT table and a CHILD table (very common scenario) and the PARENT table is FRAGMENTED, then it makes sense to FRAGMENT the CHILD table as well by JOINING IT to the PARENT FRAGMENT**
  - **Derived Horizontal Fragmentation -Each CHILD fragment is defined by the SEMI-JOIN operation**

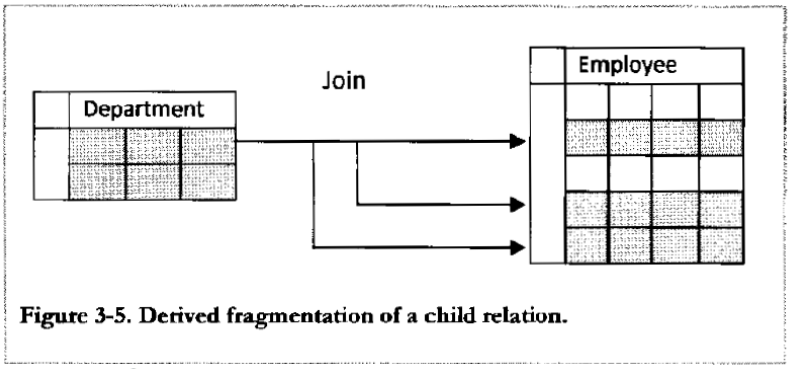



**Figure 3-5. Derived fragmentation of a child relation.**



- Derived Horizontal Fragmentation in Relational Algebra:

$$\text{Child Fragment}_i = \text{Child Relation} \blacktriangleright_F \text{Parent Fragment}_i$$

  - Where:
    - ► Is the symbol for Semi-join
    - F is the condition or predicate P.
    - Child relation: full Child table
    - Parent Fragment: Parent table that is already fragmented.

- A Semi-join, is defined as follows in Relational Algebra:

$$R \blacktriangleright_F S = \Pi_R(R \blacktriangleright\blacktriangleleft_F S)$$

  - Where:
    - **R**: is TABLE 1
    - ► Is the symbol for Semi-join
    - **F**: is the condition or predicate P.
    - **S**: TABLE 2
    - ►◄: The CROSS-PRODUCT symbol
    - $\pi$ : The PROJECTION of R or TABLE 1. Projection means only select the column or columns.

- In our example, this equation is saying the following:

$$\textbf{Semi-Join} = \Pi_R(R \blacktriangleright\blacktriangleleft_F S)$$

$$\textbf{Employee}_i = \pi_{\text{Employee}} (\text{Employee} \blacktriangleright\blacktriangleleft_{\text{deptCode}} \text{Department})$$

$$\textbf{Employee}_i \blacktriangleright_{\text{deptCode}} \textbf{Department}_i = \pi_{\text{Employee}} (\text{Employee} \blacktriangleright\blacktriangleleft_{\text{e.deptCode = d.deptCode}} \text{Department}_i)$$

## Derived Fragmentation Applied to Boston Employees:

- We now apply this Relational Algebra equation to derive the **Boston** employee fragment table.
  - $Employee_1$ = Boston or location 1 Employees derived from fragmented $Department_1$ in **Boston**:

$$\mathbf{Employee_1 = \pi_{Employee}\ (Employee \blacktriangleright\!\blacktriangleleft_{e.deptCode = d.deptCode}\ Department_1)}$$

  - Where: The projection of employee table
    - $\pi_{Employee}$ the projection of the employee table = select all columns of employee table
    - $\blacktriangleright\!\blacktriangleleft_{e.deptCode = d.deptCode}$ **cross product on condition where employee deptCode = Department DeptCode**
    - **$Department_1$ = Department** in location 1 which is Boston.
    - **Employee = the employee table.**
  - Taking the CROSS-PRODUCT between Employee table & FRAGMENT Department table in Boston

$$\mathbf{(Employee \blacktriangleright\!\blacktriangleleft_{e.deptCode = d.deptCode}\ Department_1)}$$

Employee

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 2 | Adam | Consultant | 001 | T3 |
| 3 | Mary | Part-time | 004 | T4 |
| 4 | Peter | Full-time | 003 | T2 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |
| 7 | Alex | Part-time | 004 | T2 |

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 003 | Purchasing | Boston | Production |

  - Results in the following intermediate table:

| ID | empName | emplType | deptCode | titleCode | deptCode | deptName | location | deptType |
|---|---|---|---|---|---|---|---|---|
| 2 | Adam | Consultant | 001 | T3 | 001 | Computer Center | Boston | IT |
| 4 | Peter | Full-time | 003 | T2 | 003 | Purchasing | Boston | Production |

  - Now take the projection of the intermediate table based on the condition of selecting only the columns of employee table:

$\mathbf{Employee_1 = \pi_{Employee}\ (}$

| ID | empName | emplType | deptCode | titleCode | deptCode | deptName | location | deptType |
|---|---|---|---|---|---|---|---|---|
| 2 | Adam | Consultant | 001 | T3 | 001 | Computer Center | Boston | IT |
| 4 | Peter | Full-time | 003 | T2 | 003 | Purchasing | Boston | Production |

$\mathbf{)}$

  - The CROSS-PRODUCT between Employee table & FRAGMENT Department table in Boston and selecting only the columns of Employee table results in the following:

$$Employee_1 = Employee \blacktriangleright_{deptCode} Department_1$$

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 2 | Adam | Consultant | 001 | T3 |
| 4 | Peter | Full-time | 003 | T2 |

## Boston Fragmentation Summary

- Now we summarized the results for the Boston Database.
  - After all Horizontal fragmentation (**Primary** & **Derived**) is applied we end up with the following fragments in Boston:

**Department**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 001 | Computer Center | Boston | IT |
| 003 | Purchasing | Boston | Production |

**Employee**

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 2 | Adam | Consultant | 001 | T3 |
| 4 | Peter | Full-time | 003 | T2 |



- **Does this schema meet the fragmentation rules?**
  - **Completeness Rule** – Yes, no data was lost from the fragmentation of the original Department & Employee tables.
  - **Disjoint Rule** – Yes, it appears no data is duplicated compared to the fragmentation in the other locations, which will be shown in next sections.

- Other benefits:
  - Data is located in proximity where the **Boston** users really need it, in **Boston**.
  - This will improve ***performance***.

## Derived Fragmentation Applied to New York Employees:

- We now apply this Relational Algebra equation to derive the **New York** employee fragment table.
  - $Employee_2$ = New York or location 2 Employees derived from fragmented $Department_2$ in **New York**:

$$\textbf{Semi-Join} = \pi_{\textbf{Employee}} (\textbf{Employee} \blacktriangleright\blacktriangleleft_{\textbf{e.deptCode = d.deptCode}} \textbf{Department}_2)$$

  - Taking the CROSS-PRODUCT between Employee table & FRAGMENT Department table in **New York**

$$(\textbf{Employee} \blacktriangleright\blacktriangleleft_{\textbf{e.deptCode = d.deptCode}} \textbf{Department}_2)$$

Employee

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 2 | Adam | Consultant | 001 | T3 |
| 3 | Mary | Part-time | 004 | T4 |
| 4 | Peter | Full-time | 003 | T2 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |
| 7 | Alex | Part-time | 004 | T2 |

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 002 | Budget | New York | Business |
| 005 | Purchasing | New York | Business |

  - Results in the following intermediate table:

| ID | empName | emplType | deptCode | titleCode | deptCode | deptName | location | deptType |
|---|---|---|---|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 | 002 | Budget | New York | Business |
| 5 | Scott | Consultant | 002 | T1 | 002 | Budget | New York | Business |
| 6 | Susan | Full-time | 005 | T5 | 005 | Purchasing | New York | Business |

  - Now take the projection of the intermediate table based on the condition of selecting only the columns of employee table:

$\textbf{Employee}_2 = \pi_{\textbf{Employee}}$ (

| ID | empName | emplType | deptCode | titleCode | deptCode | deptName | location | deptType |
|---|---|---|---|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 | 002 | Budget | New York | Business |
| 5 | Scott | Consultant | 002 | T1 | 002 | Budget | New York | Business |
| 6 | Susan | Full-time | 005 | T5 | 005 | Purchasing | New York | Business |

)

  - The CROSS-PRODUCT between Employee table & FRAGMENT Department table in New York and selecting only the columns of Employee table results in the following:

$$Employee_2 = Employee \blacktriangleright_{deptCode} Department_2$$

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |

## New York Fragmentation Summary

- Now we summarized the results for the **New York** Database.
  - After all Horizontal fragmentation (**Primary** & **Derived**) is applied we end up with the following fragments in Boston:

**Department**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 002 | Budget | New York | Business |
| 005 | Purchasing | New York | Business |

**Employee**

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |



- **Does this schema meet the fragmentation rules?**
  - **Completeness Rule** – Yes, no data was lost during fragmentation from the original Department & Employee tables.
  - **Disjoint Rule** – Yes, it appears no data is duplicated compared to the fragmentation in **Boston** & **Cleveland**, which are shown in previous and next sections.

- Other benefits:
  - Data is located in proximity where the **New York** users really need it, in **New York**.
  - This will improve ***performance***.

Derived Fragmentation Applied to Cleveland Employees:

- $Employee_3$ = Cleveland or location 3 Employees derived from fragmented $Department_3$ in **Cleveland**:

$$\textbf{Semi-Join} = \pi_{\textbf{Employee}} (\textbf{Employee} \blacktriangleright\blacktriangleleft_{\textbf{e.deptCode = d.deptCode}} \textbf{Department}_3)$$

  - Taking the CROSS-PRODUCT between Employee table & FRAGMENT Department table in **Cleveland**

$$(\textbf{Employee} \blacktriangleright\blacktriangleleft_{\textbf{e.deptCode = d.deptCode}} \textbf{Department}_3)$$

Employee

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 1 | John | Full-time | 002 | T1 |
| 2 | Adam | Consultant | 001 | T3 |
| 3 | Mary | Part-time | 004 | T4 |
| 4 | Peter | Full-time | 003 | T2 |
| 5 | Scott | Consultant | 002 | T1 |
| 6 | Susan | Full-time | 005 | T5 |
| 7 | Alex | Part-time | 004 | T2 |

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 004 | Database Support | Cleveland | IT |

  - Results in the following intermediate table:

| ID | empName | emplType | deptCode | titleCode | deptCode | deptName | location | deptType |
|---|---|---|---|---|---|---|---|---|
| 3 | Mary | Part-time | 004 | T4 | 004 | Database Support | Cleveland | IT |
| 7 | Alex | Part-time | 004 | T2 | 004 | Database Support | Cleveland | IT |

  - Now take the projection of the intermediate table based on the condition of selecting only the columns of employee table:

$\textbf{Employee}_3 = \pi_{\textbf{Employee}}$ (

| ID | empName | emplType | deptCode | titleCode | deptCode | deptName | location | deptType |
|---|---|---|---|---|---|---|---|---|
| 3 | Mary | Part-time | 004 | T4 | 004 | Database Support | Cleveland | IT |
| 7 | Alex | Part-time | 004 | T2 | 004 | Database Support | Cleveland | IT |

)

  - The CROSS-PRODUCT between Employee table & FRAGMENT Department table in **Cleveland** and selecting only the columns of Employee table results in the following:

$$Employee_3 = Employee \blacktriangleright_{deptCode} Department_3$$

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 3 | Mary | Part-time | 004 | T4 |
| 7 | Alex | Part-time | 004 | T2 |

## Cleveland Fragmentation Summary

- Now we summarized the results for the **Cleveland** Database.
  - After all Horizontal fragmentation (**Primary** & **Derived**) is applied we end up with the following fragments in **Cleveland**:

**Department**

| deptCode | deptName | location | deptType |
|---|---|---|---|
| 004 | Database Support | Cleveland | IT |

**Employee**

| ID | empName | emplType | deptCode | titleCode |
|---|---|---|---|---|
| 3 | Mary | Part-time | 004 | T4 |
| 7 | Alex | Part-time | 004 | T2 |



- **Does this schema meet the fragmentation rules?**
  - **Completeness Rule** – Yes, no data was lost from the original Department & Employee tables.
  - **Disjoint Rule** – Yes, it appears no data is duplicated compared to the fragmentation in **Boston** & **New York**, which were shown in previous sections.

- Other benefits:
  - Data is located in proximity where the **Cleveland** users really need it, in **Cleveland**.
  - This will improve ***performance***.

## Final Summary of Horizontal Fragmentation

- The following diagram shows a global view of all the fragmented tables.

  - Each location only has data that users in that location can access, thus improvement in performance.
  - The Global Application which needs data about all departments & employees of the company can execute a global query to all sites. And more importantly can execute each part for each location in PARALLEL.

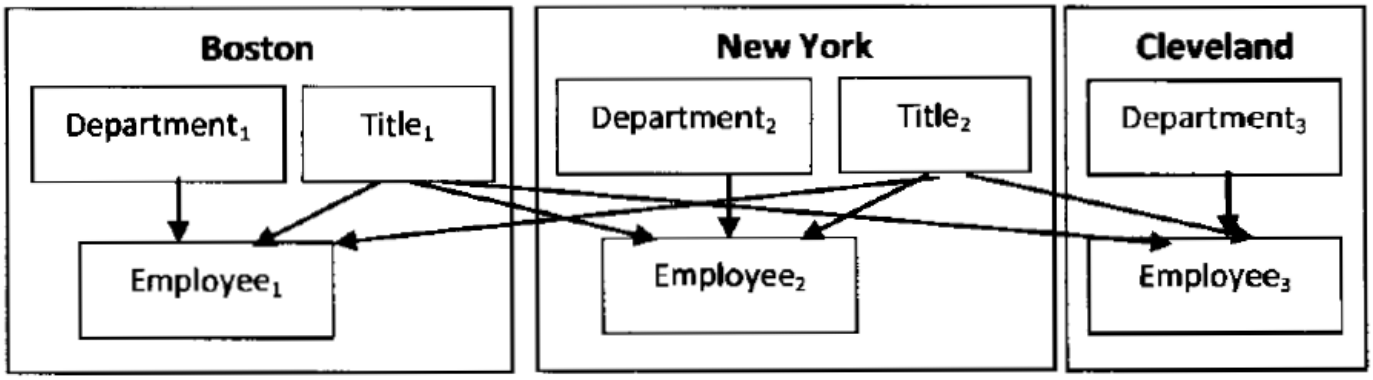



**Figure 3-7. Diagram of derived fragmentation.**

## 3.2.3 Vertical Fragmentation

- **Vertical Fragmentation** – **COLUMNS** of a table are stored in different databases

  - **COLUMNS of a table are stored in different databases.**
  - **Goals of Vertical Fragmentation is to keep attributes or columns needed by an Application close together in the same FRAGMENT.**
  - **Which attributes or columns that are placed closed together is based on the following requirements:**
    - **How the columns are accessed or used**
    - **Frequency of access**
    - **Location**

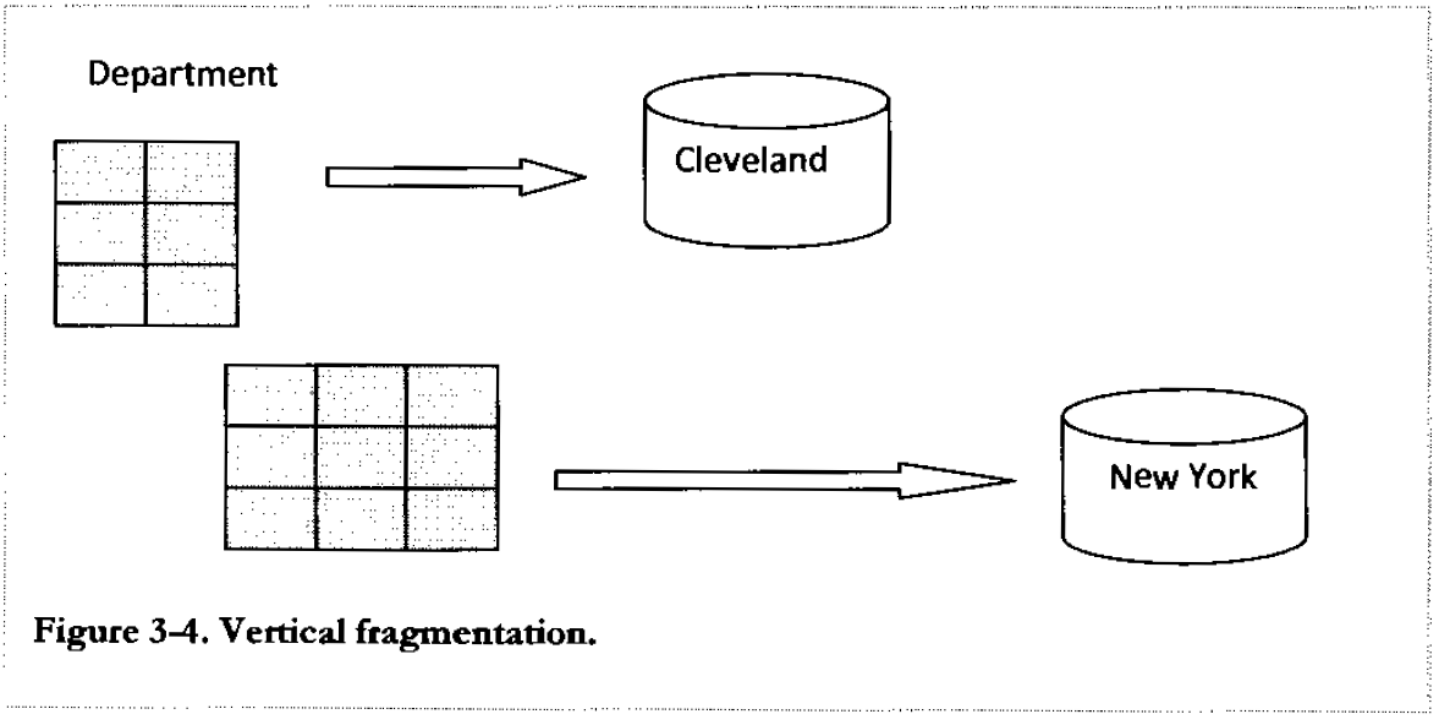



**Figure 3-4. Vertical fragmentation.**



### Rules for Vertical Fragmentation

- Fragmentation must abide by the following rules:

  - **Completeness Rule**:
    - Data **cannot** be lost in the process of fragmentation
    - **Each Attribute MUST be included in AT LEAST ONE fragment**.

  - **Disjoint Rule**:
    - Data **cannot** be duplicated in the process of fragmentation
    - No duplication, so we don't have redundancy and inconsistency.
    - **Each NON-KEY COLUMN MUST be included in ONE & ONLY ONE fragment**
    - **The PRIMARY KEY ATTRIBUTES OR COLUMN ARE INCLUDED IN EACH VERTICAL FRAGMENT**

## Vertical Fragmentation & Relational Algebra

- **Vertical Fragmentation** is defined using RELATIONAL ALGEBRA by the PROJECTION OPERATOR:

  - The expression for a **Vertical Fragmentation** query in Relational Algebra is called a PROJECTION & the formula is as follows:

$$R_i = \pi_{xi} (R)$$

Where $X_i$ is a subset of the attributes or columns in relation or table R

  - Where:

    - $R_i$ – The resultant relation from the query (the result of a query is a table).
    - $\pi$ – Greek letter PI or PROJECTION SYMBOL: Represents the SELECTION of columns.
    - $X_i$– The subset of the columns in the table R.
    - R = relation or table selection being applied

## Use-Case #6 (Malyuta's book) Requirements & Implementation

- **Use-case #6** states the following:

*Case 3.6.*

*The New York users need data about the names of all employees and codes of the departments to which employees are assigned. The Cleveland users are working with data about the types and title codes of all employees.*

- For **USE-CASE #6**, the **VERTICAL FRAGMENTATION equations** & **fragment** for the **New York** users look as follows:

$$Employee_1 = \Pi_{ID,\ empName,\ deptCode}(Employee)$$

**$Employee_1$ (New York):**

| ID | empName | deptCode |
|---|---|---|
| 1 | John | 001 |
| 2 | Adam | 002 |
| 3 | Mary | 003 |
| 4 | Peter | 001 |
| 5 | Scott | 002 |
| 6 | Susan | 005 |
| 7 | Alex | 004 |

- For **USE-CASE #6**, the **VERTICAL FRAGMENTATION equations** & **fragment** for the **Clevelan** users look as follows:

$$Employee_2 = \Pi_{ID,\ emplType,\ titleCode}(Employee)$$

**$Employee_2$ (Cleveland):**

| ID | emplType | titleCode |
|---|---|---|
| 1 | Full-time | T1 |
| 2 | Consultant | T3 |
| 3 | Part-time | T4 |
| 4 | Full-time | T2 |
| 5 | Consultant | T1 |
| 6 | Full-time | T5 |
| 7 | Part-time | T2 |

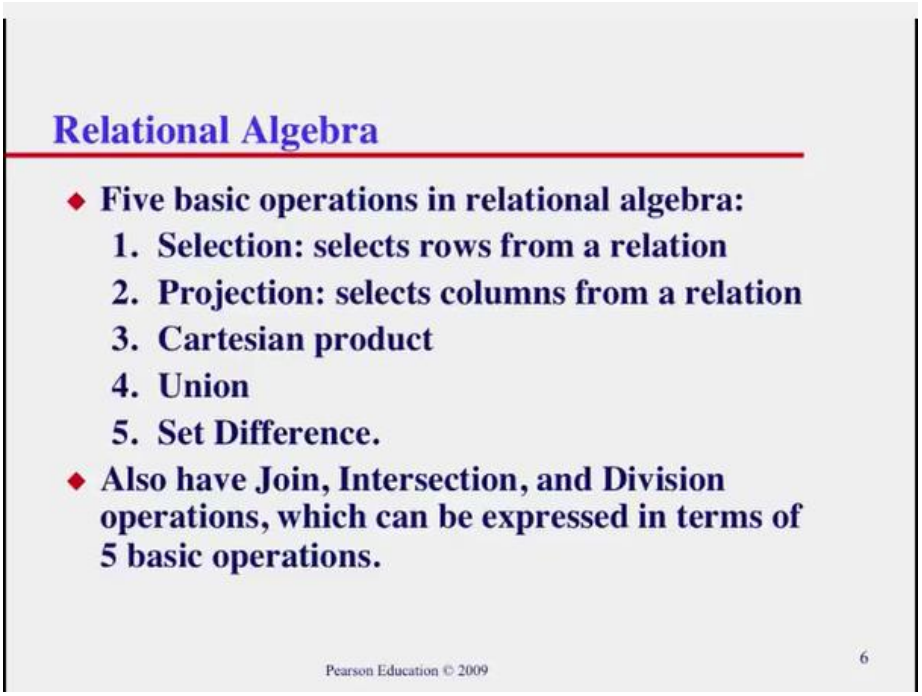
# Appendix – Relational Algebra

- Videos to help you learn Relational Algebra:

05-01-relational-algebra-1.mp4
https://www.youtube.com/watch?v=tii7xcFilOA

05-02-relational-algebra-2.mp4
https://www.youtube.com/watch?v=GkBf2dZAES0

- Database Management Systems use Relations Algebra algorithms to implement the execution of a Query.



## Relational Algebra

- Five basic operations in relational algebra:
  1. Selection: selects rows from a relation – WHERE
  2. Projection: selects columns from a relation – SELECT
  3. Cartesian product
  4. Union
  5. Set Difference.
- Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.

ver → column list FROM

## Relational Algebra

Horizontal slice of a relation

- Five basic operations in relational algebra:
  1. Selection: selects rows from a relation – WHERE
  2. Projection: selects columns from a relation – SELECT
  3. Cartesian product
  4. Union
  5. Set Difference.
- Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.

vertical slice of a relation → column list FROM

## Relational Algebra Operations

1. **Unary Operations** – Operate on one relation.
   - Selection
   - Projections
2. **Binary Operations** or **Set Operations** – Operate on pairs of relations.
   - Union
   - Set Difference
   - Intersection
   - Cartesian Product
   - Join Operations

# Relational Algebra Operators

## SELECT

**Symbol #1 - Sigma**

Select Operation = $\sigma$

- Use to pick rows from a table

Subscript to select operator $\sigma_{condition}$ (Relation or table)

- Condition or filter to filter rows to extract from table

R = Relation or tabler $\sigma_{condition}(R)$

- Condition or filter to filter rows to extract from table

Formula

Subset of relation = $\sigma_F(R)$

Where:

- Sigma = selection
- F = filter or condition (where clause)
- R = relation or table selection being applied.

Subset of relation = $\sigma_F(Expr)$

Where:

- Sigma = selection
- F = filter or condition (where clause)
- Expr = Expression or another Relational algebra statement (Select or Project expression).

**Symbol #1 – Logical AND Operator = ∧**

## Project Operator

**Operator #2 - Project**

Select Operation = $\pi$

- Use to pick columns from a table

Subscript to list of colums to return = $\pi_{column\ list}$ (Relation or table)

- Column list = list of columns to return

R = Relation or tabler $\pi_{A1....An}(R)$

- Column list = list of columns to return

Formula

Subset of relation = $\pi_{A1....An}(R)$

Where:

- Pi = projection
- A1…An = List of columns or attributes.
- R = relation or table selection being applied.

Subset of relation = $\pi_{A1....An}(Expr)$

Where:

- Pi = projection
- A1…An = List of columns or attributes.
- Expr = Expression or another Relational algebra statement (Select or Project expression).

## Duplicate Values

## Cross Product:  Combine Two Tables

# To be continued