CST3604 Lecture Notes

Physical Design

RDBMS Physical Design & Implementation – Physical DB Design

(Part 3 of 3)

(Lecture Notes 2C)

Prof. Abel Angel Rodriguez

Chapter 2 Physical Database Design (Chapter 2 - PART 2)

2.1 3-Level Database Architecture (Level of abstraction)

2.1.1 Commercial DBMSs & the 3-Level Architecture

- Commercial DBMS vendors comply with standards from ANSI (American National Standards Institute) and SPARC (Standards Planning & Requirements Committee)
- These regulatory agencies require DBMS manufactures to comply with a 3-level architecture shown below:

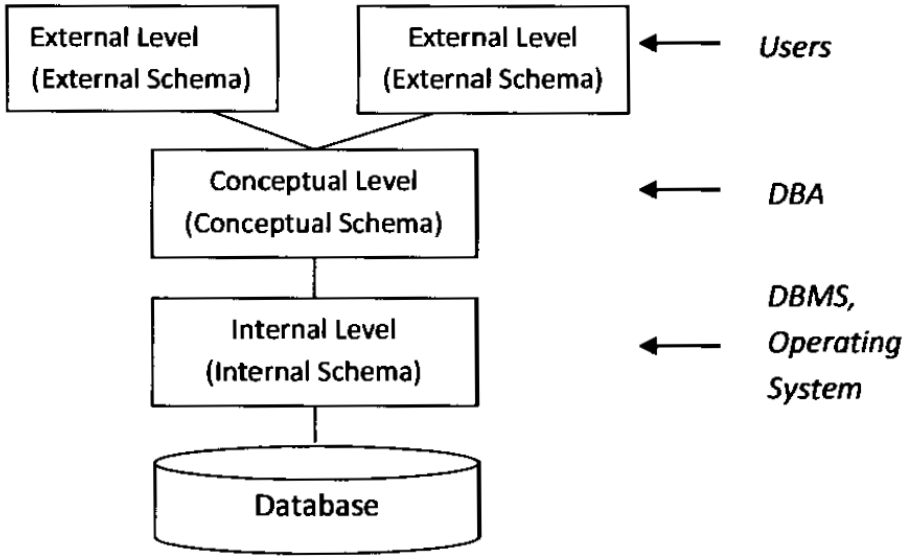


Figure 2-8. Three-level database architecture.

- This 3-level architecture provides a level of separation and abstraction between the USER'S perception of the database, the DEVELOPERS/DBA perception of the database and finally the DBMS & OS perception or physical implementation of data.

- Table below summarizes the **EXTERNAL LEVEL**:

Levels	Description

EXTERNAL LEVEL or External Schema (User view)	▪ How users see data (User View)
	▪ In this abstraction, different users may see different data:
	o For example, in the manufacturing database shown in figure below:
	- User scott a consultant from the Budget department, may only work with the following data:
	ID, Name & Salary of employees
	- This is Scott's view of the database and all he needs to see
	- A second user John also from the Budget department, may only work with the following data:
	Names, Department codes & Department Names
	- This is John's view of the database and all he needs to see. To John this is the database.
	▪ The external schema consists of several views depending on the usage of the database.
	▪ DBMS contain a mechanism called a **VIEW** to assist in creating different user views to implement the external level
	o A VIEW is a table that does not actually exists but is a virtual table.
	o It is a table that is created on the fly by a QUERY joining one or more table.
	o A view is really a STORED QUERY that executes on the fly.
	o To the user, the VIEW is a table, but, it is a Query.
	o Working with VIEWS requires the following steps:
	1) Analyze the business requirements for isolating data from an existing table as the foundation of the VIEW. These could be a business requirement as well as an EXISTING QUERY that needs to be addressed and modified into a VIEW based on those business requirements.
	2) DESIGN & CREATE the VIEW based on your analysis.
	• Syntax to creating a VIEW in ORACLE 11g:
	`CREATE VIEW View_Name AS`
	`SELECT columns`
	`FROM table(s)`
	`[WHERE conditions];`
	3) Execute or COMPILE the VIEW, thus creating a VIEW OBJECT in the database.
	4) Now you are ready to create Queries that query the VIEW or VIRTUAL TABLE.
	• In queries, the VIEW is used instead of the real table name. For example.
	`SELECT columns`
	`FROM Table`
	`[WHERE conditions];`
	- Where the Table is this case is the view ***View_Name***

Example:

- Consider the following Manufacturing company database:

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

- Assuming an analysis is made by the DB Admin and is identified that different user applications are using different data:
  - User scott a consultant from the Budget department, may only work with the following data:

    ID, Name & Salary of employees

    - This is Scott's view of the database and all he needs to see

  - A second user John also from the Budget department, may only work with the following data:

    Names, Department codes & Department Names

    - This is John's view of the database and all he needs to see. To John this is the database.

- We can create a VIEW for Scott and a separate VIEW for John:

Scott's view

```
CREATE VIEW vw_ScottTable AS
SELECT ID, name, salary
FROM Employee e, Title t
WHERE e.TitleCode = t.TitleCode;
```

John's View

```
CREATE VIEW vw_JohnTable AS
SELECT name, DeptCode, DeptName
FROM Employee e, Department d
WHERE e.DeptCode = d.DeptCode;
```

  - Below Scott sees only the **`vw_ScottTable`** when executing queries and John sees on the **`vw_JohnTable`**:

Scott's query

```
SELECT * FROM vw_ScottTable;
```

John's query

```
SELECT * FROM vw_JohnTable;
```

  - Both of these uses don't know where the actual data is, how is organized or what actual tables they are using, since their view of the data is a DATABASE VIEW not the actual tables.

- Table below summarizes the **CONCEPTUAL LEVEL**:

Levels	***Description***
CONCEPTUAL LEVEL (DBA/Developer view)	▪ This is the developers/dba vision of the database. ▪ Contains the table definitions, columns, constraints, etc. ▪ This is where all the Database OBJECTS, such as o Tables, views, stored procedures, indices, etc., are defined. ▪ Each VIEW of the users from the EXTERNAL LEVEL is based on the definitions of these database objects. ▪ It is also the link between the EXTERNAL LEVEL and the INTERNAL LEVEL or operating system level.

- Table below summarizes the **INTERNAL LEVEL**:

Levels	***Description***
INTERNAL LEVEL (DBMS/OS View)	▪ This is the DBMS & Operating System OS sees the data in the database. ▪ It includes description of: o Data Structures o File Organization o Storage Space allocation for objects of the database o Rows placement o Data compression, o Data encryption o Other physical parameters of data

- Table below list some benefits of the 3 – level architecture:

Benefits of the 3-Tier Architcture include:
▪ Users see data according to their needs: o If users needs change: - Wiew of data can change without rebuilding or restructuring the database or the view of others. Just change/update the existing view! - Create new views as needed
▪ Users can see data the way they want to see it, completely unaware of the conceptual or physical/internal aspects of the data o Don't know which actual table the data comes from o What data structure, format, etc. o How the data is processed etc.
▪ Reorganizing data storage does not affect the CONCEPTUAL or EXTERNAL user views o Tables can be moved from disk to disk without user's knowledge
▪ Redesigning the CONCEPTUAL model does not affect the EXTERNAL user views o You can ADD columns to the tables that don't affect the actual VIEW QUERY, without affecting the VIEWS o If adding a column to a table does affect the view, because the column is part of the VIEW QUERY, simply rebuild the view itself and users will be unaware of the change.

2.2 Views in Oracle

2.2.1 Overview

- This section will be an introductory conversation on Views in ORACLE, primarily in preparation for the Project Exam that you will need to implement VIEWS.
- In previous section we introduced VIEWS

  - A VIEW is a table that does not actually exists but is a virtual table.
  - It is a table that is created on the fly by a QUERY joining one or more table.
  - A view is really a STORED QUERY that executes on the fly.
  - To the user, the VIEW is a table, but, it is a Query.
  - Working with VIEWS requires the following steps:
  - Working with VIEWS requires the following steps:

    1) Analyze the business requirements for isolating data from an existing table as the foundation of the VIEW. These could be a business requirement as well as an EXISTING QUERY that needs to be addressed and modified into a VIEW based on those business requirements.
    2) DESIGN & CREATE the VIEW based on your analysis.
       - Syntax to creating a VIEW in ORACLE 11g:

```
CREATE VIEW View_Name AS
  SELECT columns
  FROM table(s)
  [WHERE conditions];
```

    3) Execute or COMPILE the VIEW, thus creating a VIEW OBJECT in the database.
    4) Now you are ready to create Queries that query the VIEW or VIRTUAL TABLE.
       - In queries, the VIEW is used instead of the real table name. For example.

```
  SELECT columns
  FROM Table
  [WHERE conditions];
```

       - Where the Table is this case is the view ***View_Name***.

- We also provided an example of VIEWS for our manufacturing company example.
- In this section we dive a little deeper

2.2.1 Creating Views

- Previous section we listed the steps to creating views:

1) Analyze the business requirements for isolating data from an existing table as the foundation of the VIEW. These could be a business requirement as well as an EXISTING QUERY that needs to be addressed and modified into a VIEW based on those business requirements.
2) DESIGN & CREATE the VIEW based on your analysis.
3) Execute or COMPILE the VIEW, thus creating a VIEW OBJECT in the database.
4) Now you are ready to create Queries that query the VIEW or VIRTUAL TABLE.

- Let's look at each of these steps in details and discuss these steps using another example.

Step 1 – Analyze the business requirements for isolating data from an existing table as the foundation of the VIEW

- The ideas of views to provide an isolation of data from a table to user applications:
  - The idea is only expose data that is needed by the applications not any more:
    - These requirements may come from:
      - The InfoSec Team (Information Security Team at the organization)
      - Regulatory Requirements from Compliance Team of an organization
      - Database Admin looking for ways to properly implement the Physical Model (Performance, Security, etc.), by analyzing the existing queries and looking to properly design the system.
  - The point is that if applications need access to a table:
    - If after analyzing the queries used by the application, you realized that the queries are only accessing only a portion of the table's data, why EXPOSE the application to all the data?
    - Simply create Views for those queries and isolate. We showed and example of this with our manufacturing database, where we realized that the application used by user Scott only was accessing a specific set of data (***ID, Name & Salary of employees***) and the application used by user John was also only accessing a specific subsets of the tables data (***Names, Department codes & Department Names***):

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

- And we ended up creating the following two views which enabled Scott' application to only see the **`vw_ScottTable`** when executing queries and John' application sees on the **`vw_JohnTable`** when executing queries:

Scott's view

```
CREATE VIEW vw_ScottTable AS
SELECT ID, name, salary
FROM Employee e, Title t
WHERE e.TitleCode = t.TitleCode;
```

John's View

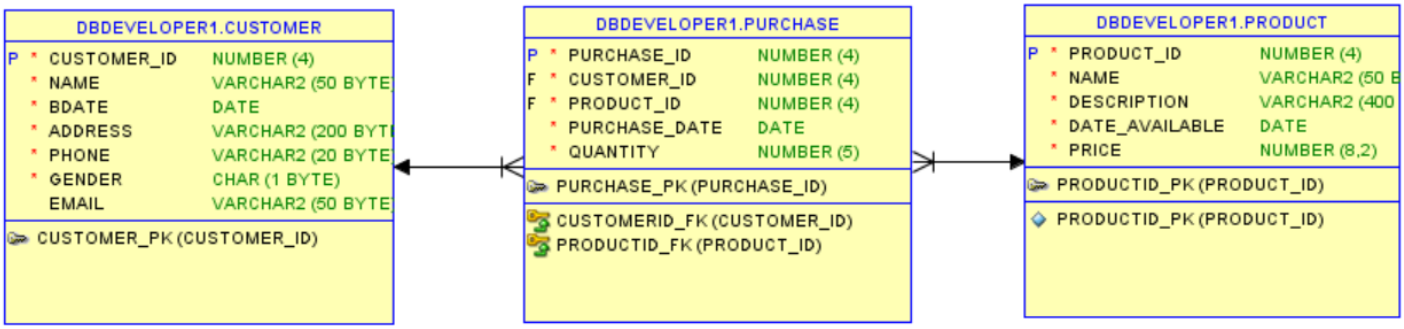
```
CREATE VIEW vw_JohnTable AS
SELECT name, DeptCode, DeptName
FROM Employee e, Department d
WHERE e.DeptCode = d.DeptCode;
```

Ecommerce Website Analysis Example:

- Another example or scenario a for an ecommerce website where customer purchase products and the following requirements have been derived by the security and compliance team of the company:

The InfoSec team & compliance have advised the database admins of new regulations to isolate and secure data better. One method of implementing this security is to isolate user view to data to only what they need and so creating VIEWS is recommended to the Database Team

- The DB Admins get to work on analyzing and looking for opportunities to leverage Views to isolate data in the application
- Currently the application schema & data looks as follows:
  - Schema/table structure



- We also assume we have the following data in the **Customer Table**:

CUSTOMER_ID	NAME	BDATE	ADDRESS	PHONE	GENDER	EMAIL
1111	Joe Smith	01-JAN-71	111 Jay Street, Brooklyn NY 11201	718 111-1111	M	jsmith@xyz.com
2222	Angel Rodriguez	02-FEB-72	222 Flatbush Avenue, Brooklyn NY 11202	718 222-2222	M	arod@xyz.com
3333	Mary Jones	03-MAR-73	333 Flatlands Avenue, Brooklyn NY 11203	718 333-3333	F	mjones@xyz.com
4444	Samuel Adams	04-APR-74	444 Park Avenue, New York NY 11204	212 444-4444	M	(null)
5555	Nancy Rivera	05-MAY-75	555 Metropolitan Avenue, Brooklyn NY 11205	718 555-5555	F	nrivera@xyz.com

- We also assume we have the following data in the **Product Table**:

PRODUCT_ID	NAME	DESCRIPTION	DATE_AVAILABLE	PRICE
1111	Galaxy HD TV	High Definition TV	01-JAN-01	1000
2222	Tornato Washer Delux	High Power Washing Machine	02-FEB-02	2000
3333	Tornato Dryer Delux	High Power Dryer Machine	03-FEB-03	2000
4444	StarBright Computer	Laptop Computer	04-MAR-04	1500

- Finally, we assume we have the following data in the **Purchase Table**:

PURCHASE_ID	CUSTOMER_ID	PRODUCT_ID	PURCHASE_DATE	QUANTITY
1	2222	2222	10-MAY-10	1
2	2222	3333	10-MAY-10	1
3	1111	1111	04-JUL-10	2
4	5555	4444	24-DEC-15	4

- After an assessement of the application, inteviews with the application business users & the Queries by the DB Admins, it is identified that the business or marketing teams of the ecommerce company only require access to to customer & purchase information for reasons of promotion offers, other marketing reasons, etc., and need access only to the following data:

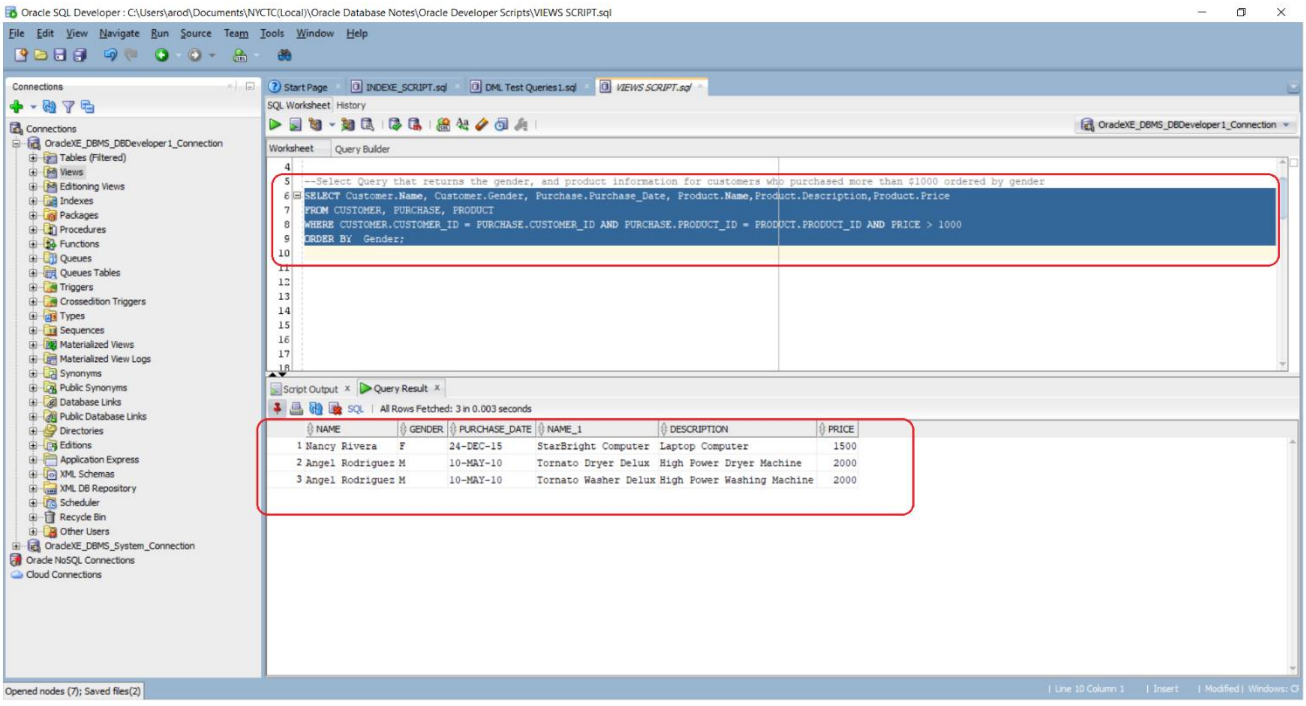
  - **Customer name**
  - **Customer Gender**
  - **Name of Product**
  - **Description of Products customer have purchased**
  - **Date the customer made the purchase**
  - **Finally price paid for the products.**

- In addition, the following Query is identified as a key query often run by this Marketing department in order to identify customers who've made X amount of purchases in order to send them a promotional discount etc. **An example of this query with the data already captured from the webpage UI is as follows**:

```
--Select Query that returns the gender, date of purchase,
description of product & price by customers orderd by gender
SELECT Customer.Name, Customer.Gender, Purchase.Purchase_Date,
Product.Name, Product.Description,Product.Price
FROM CUSTOMER, PURCHASE, PRODUCT
WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID AND PRICE > 1000
ORDER BY  Gender;
```

- Using Oracle SQL Developer, the DB Admin execute the querty to see the results:

  1. Enter the QUERY into the worsheet window
  2. Highlight & execute QUERY
  3. The output window should show the result of the query.



- An analysis of the query shows the following:

```
--Select Query that returns the gender, date of purchase,
description of product & price by customers orderd by gender
SELECT Customer.Name, Customer.Gender, Purchase.Purchase_Date,
Product.Name, Product.Description,Product.Price
FROM CUSTOMER, PURCHASE, PRODUCT
WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID AND PRICE > 1000
ORDER BY  Gender;
```

  - Three tables are required: CUSTOMER, PURCHASE & PRODUCTS
  - The following columns are returned which align with the data they only need to see: Customer.Name, Customer.Gender, Purchase.Purchase_Date, Product.Description & Product.Price columns are returned.
  - Specific search or criteria on Price column
  - Query is ordered by Gender column.
  - Finally & IMPORTANT, we realized there may be a name conflict in the final VIEW because the Customer.Name column and the Product.Name colums are both named NAME and the final VIEW VIRTUAL table will default to the first Customer.Name, so the final VIEW VIRTUAL TABLE will have Customer.Name repeated twice and we won't get the Product name, so we need to fix and address this in the VIEW DESIGN/IMPLEMENTATION:

    - Note that this error may not be obvious at first and you may go ahead and implement and the final product will show the error and you come back to the drawing board! Neverthless, this is the idea of analysis and design, which is to THINK and take everything under consideration.
    - Of course with experience you will be able to pick this up during analysis
    - Nevertheless this was a bad design from the start of the table creation and should have been pick-up when application was being designed based on nameing convention.

- Based on this information we begin to design & create the VIEW. We will show this in the next section.

Step 2 – DESIGN & CREATE the VIEW based on the analysis

- In the Analysis step, we came to the following conclusions for the query in question:

```
--Select Query that returns the gender, date of purchase,
description of product & price by customers orderd by gender
SELECT Customer.Name, Customer.Gender, Purchase.Purchase_Date,
Product.Description,Product.Price
FROM CUSTOMER, PURCHASE, PRODUCT
WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID AND PRICE > 1000
ORDER BY  Gender;
```

  - Three tables are required: CUSTOMER, PURCHASE & PRODUCTS
  - The following columns are returned which align with the data they only need to see: Customer.Name, Customer.Gender, Purchase.Purchase_Date, Product.Description & Product.Price columns are returned.
  - Specific search or criteria on Price column
  - Query is ordered by Gender column.
  - Finally & IMPORTANT, we realized there may be a name conflict in the final VIEW because the Customer.Name column and the Product.Name colums are both named NAME and the final VIEW VIRTUAL table will default to the first Customer.Name, so the final VIEW VIRTUAL TABLE will have Customer.Name repeated twice and we won't get the Product name, so we need to fix and address this in the VIEW DESIGN/IMPLEMENTATION.

- We now begin to anlayze the design and create the views. We will show the WRONG WAY to do this and the CORRECT APPROACH but first let's look at the simple syntax for a VIEW in ORACLE 11g again:

```
CREATE VIEW View_Name AS
  SELECT columns
  FROM table(s)
  [WHERE conditions];
```

- Based on the analysis, we begin our high-level decisions for our VIEW:

  - We decide to name the VIEW Customer_Purchases
  - To address the name conflict we will use the AS clause in the SELECT query to rename each column appropriately in the results of the query as follows:

    - Customer.Name AS Customer_Name
    - Product.Name AS Product_Name

Wrong Approach to Implementing the Views

- You may be tempted to simply take the query as is and create the view:
  - We decide to name the VIEW Customer_Purchases
  - We implement as follows:

```
CREATE VIEW Customer_Purchases AS
  SELECT Customer.Name AS Customer_Name, Customer.Gender,
  Purchase.Purchase_Date, Product.Name AS Product_Name,
  Product.Description,Product.Price
  FROM CUSTOMER, PURCHASE, PRODUCT
  WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
  PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID AND PRICE > 1000
  ORDER BY  Gender;
```

- This of course works and the users of this application will always get the correct results since the VIEW is effectively the same as the QUERY, neverthless a NEW QUERY will query the VIEW thus providing the desire isolation.
- This approach will also make it easy to execute the queries that will use this VIEW in STEP 4, since it will simply be this:

```
SELECT *
FROM CUSTOMER_PURCHASES;
```

- **But THIS APPROACH IS INCORRECT, AND YOU CANNOT DO THIS IN YOUR PROJECT EXAM!!!**

- Here is why:
  - What happens when new requirements for the marketing team is needed using the same data set? In other words, what if there is a new querty that is not just targeting PRICE BY GENDER, but also other criteria?
  - Well, you cannot use this VIEW! You will need to create another VIEW!
  - DOES THIS MEAN WE NEED TO CREATE A VIEW FOR EVERY SCENARIO FOR THIS MARKETING DEPARTMENT?
  - OF COURSE NOT!! Than we end up with an excessive amount of VIEWS and we are now having performance issues etc.

- Solution:
  - **DESIGN & CREATE YOUR VIEW, LIKE YOU CREATED YOUR TABLES, OPEN FOR ANY QUERIES ON THAT DATA SET!!!**
  - DESIGN & CREATE the VIEW so that the Marketing team CAN EXECUTE ANY QUERIES ON THAT DATA SET!!!!

- Next we show how ot do it right AS EXPECTED IN YOUR PROJECT EXAM!

Correct Approach to Implementing the Views

- You CANNOT simply create the VIEW identical to the query:
- You need to analyze and design based on the following guidance:

  - **DESIGN & CREATE YOUR VIEW, LIKE YOU CREATED YOUR TABLES, OPEN FOR ANY QUERIES ON THAT DATA SET!!!**
  - DESIGN & CREATE the VIEW so that the Marketing team CAN EXECUTE ANY QUERIES ON THAT DATA SET!!!!

- With the above guidance we analyze the original query as follows

```
--Select Query that returns the gender, date of purchase,
description of product & price by customers orderd by gender
SELECT Customer.Name, Customer.Gender, Purchase.Purchase_Date,
Product.Description,Product.Price
FROM CUSTOMER, PURCHASE, PRODUCT
WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID AND PRICE > 1000
ORDER BY  Gender;
```

  - We decide to name the VIEW Customer_Purchases
  - Three tables are required: CUSTOMER, PURCHASE & PRODUCTS
  - The following columns are returned which align with the data they only need to see: Customer.Name, Customer.Gender, Purchase.Purchase_Date, Product.Description & Product.Price columns are returned.
  - Specific search or criteria on Price column, **BUT WE WILL NOT INCLUDE PRICE in the VIEW so VIEW is a table that can be searched on PRICE OR ANY COLUMN**!
  - Query is ordered by Gender column. **BUT WE WILL NOT INCLUDE ORDER BY in the VIEW so VIEW is a table that can be contain any type of cluase NOT JUST ORDER BY**!
  - IMPORTANT, we realized there may be a name conflict in the final VIEW because the Customer.Name column and the Product.Name colums are both named NAME and the final VIEW VIRTUAL table will default to the first Customer.Name, so the final VIEW VIRTUAL TABLE will have Customer.Name repeated twice and we won't get the Product name, so we need to fix and address this in the VIEW DESIGN/IMPLEMENTATION.

- We now DESIGN & CREATE our VIEW as follows

```
CREATE VIEW Customer_Purchases AS
 SELECT Customer.Name AS Customer_Name, Customer.Gender,
 Purchase.Purchase_Date, Product.Name AS Product_Name,
 Product.Description,Product.Price
 FROM CUSTOMER, PURCHASE, PRODUCT
 WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
 PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID;
```

- **This works not just for PRICE & GENDER SEARCHING BUT ALL TYPES OF SEARCHING ONLY ON THIS SUBSET OF DATA**.

- **THIS APPROACH IS CORRECT AND SHOULD BE THE METHOD EXECUTED IN YOUR PROJECT EXAM!!!**

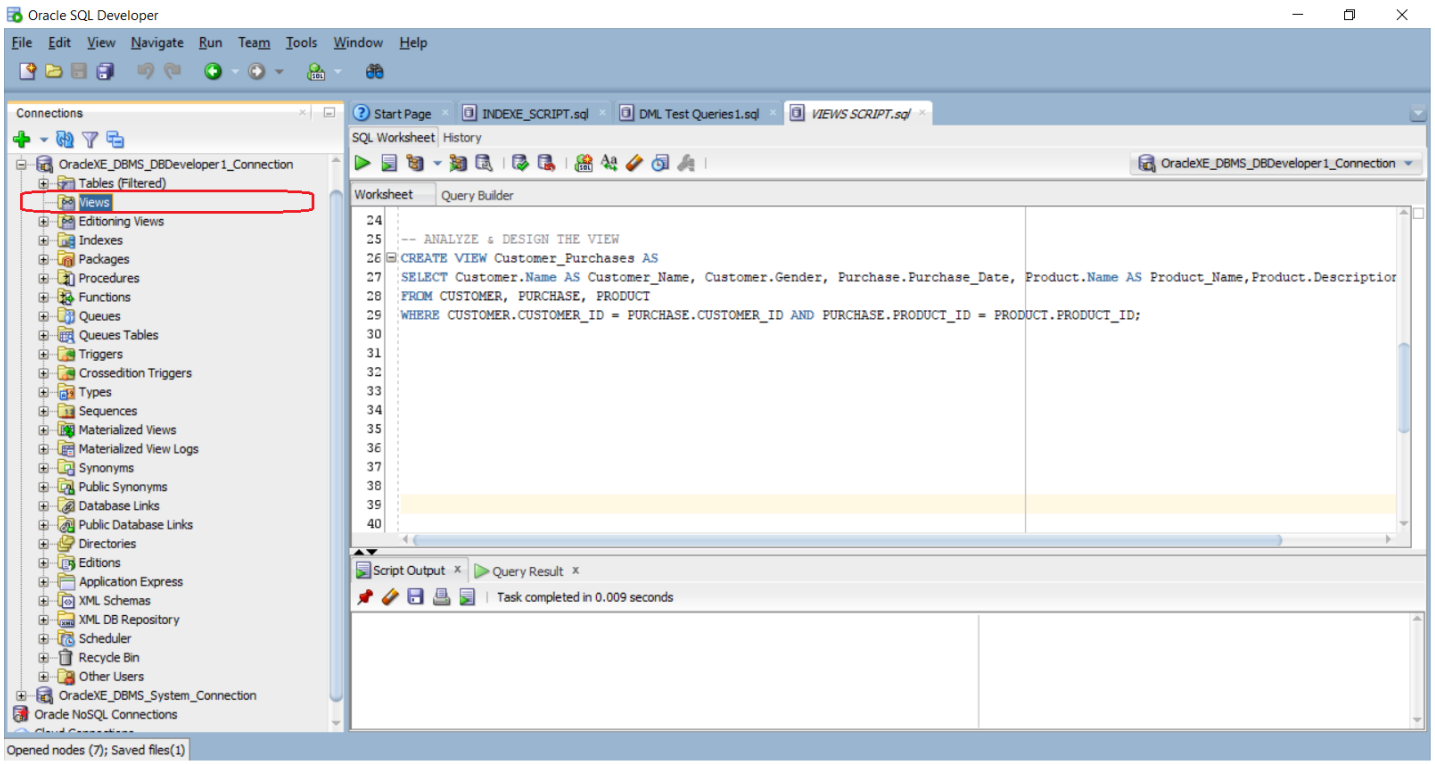
- In the next steps we discuss how to finalize the implementatio of the VIEW and how to use it!

Step 3 – Execute or COMPILE the VIEW, thus creating a VIEW OBJECT in the database

- The DESIGN of our VIEW as follows

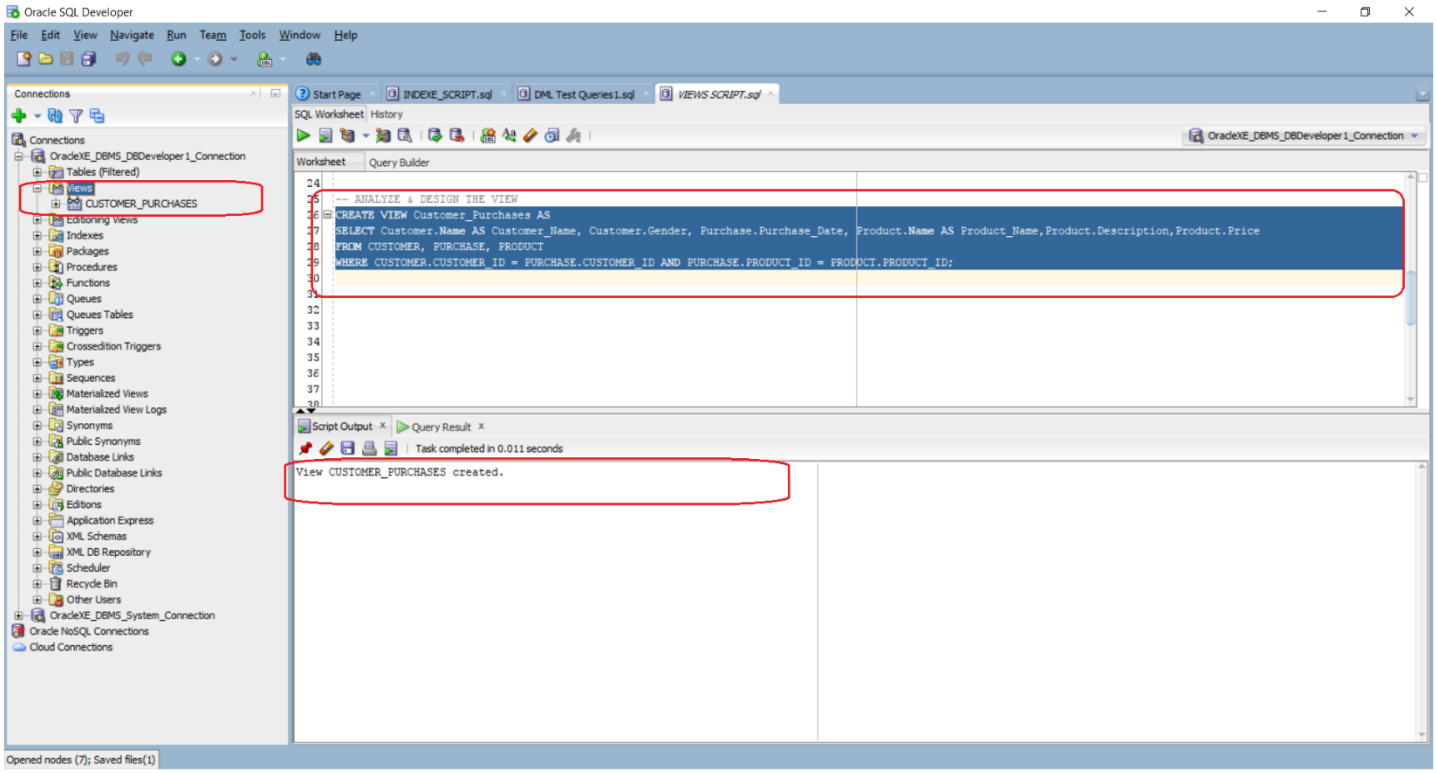
```
CREATE VIEW Customer_Purchases AS
 SELECT Customer.Name AS Customer_Name, Customer.Gender,
 Purchase.Purchase_Date, Product.Name AS Product_Name,
 Product.Description,Product.Price
 FROM CUSTOMER, PURCHASE, PRODUCT
 WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
 PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID;
```

- We now need to EXECUTE & COMPILE this VIEW so is an OBJECT in the database, just like tables are objects.
- In Oracle SQL Developer we create and execute the view and we end up with a VIEW object.
- We open Oracle SQL Developer to begin to implement the VIEW:
  - We see that in Oracle SQL Developer there is NO VIEW OBJECTS



- Next we enter the newly designed VIEW into the worsheet window, highlight & execute:

1. Enter the VIEW into the worsheet window
2. Highlight & execute VIEW
3. The output window should indicate the that VIEW was created.
4. The View Object should now display in the object window. You may need to refresh.



Step 4 – Use the VIEW by Creating the desired Queries that will query the VIEW

- Now you are ready to create Queries that query the VIEW or VIRTUAL TABLE.
  - For our example, the goal was to get the same results as the original query:

```
--Select Query that returns the gender, date of purchase,
description of product & price by customers orderd by gender
SELECT Customer.Name, Customer.Gender, Purchase.Purchase_Date,
Product.Description,Product.Price
FROM CUSTOMER, PURCHASE, PRODUCT
WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID AND PRICE > 1000
ORDER BY  Gender;
```

  - Where the results were as follows:

	NAME	GENDER	PURCHASE_DATE	NAME_1	DESCRIPTION	PRICE
1	Nancy Rivera	F	24-DEC-15	StarBright Computer	Laptop Computer	1500
2	Angel Rodriguez	M	10-MAY-10	Tornato Dryer Delux	High Power Dryer Machine	2000
3	Angel Rodriguez	M	10-MAY-10	Tornato Washer Delux	High Power Washing Machine	2000

- Listing again the view for our convenience in designing a NEW QUERY to give us the results of the original query:

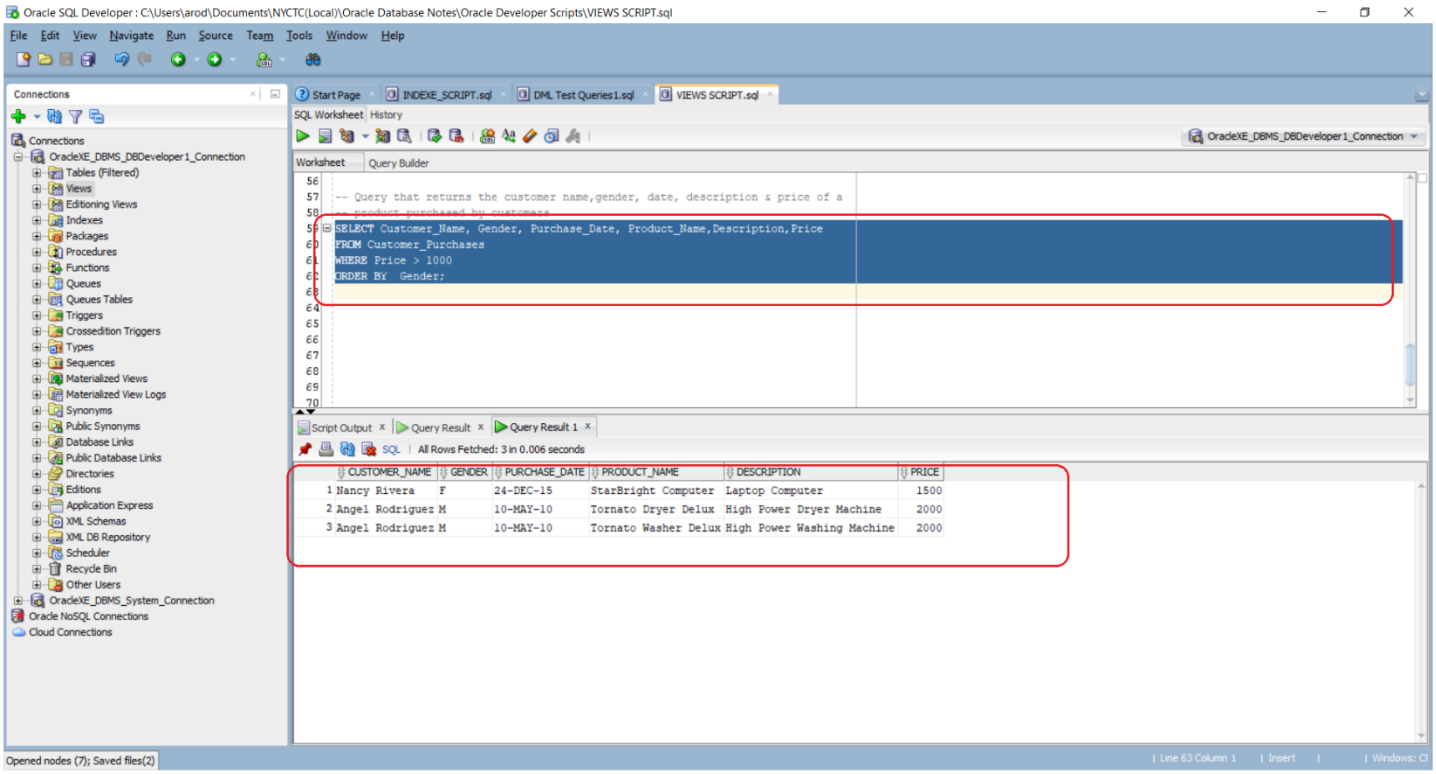
```
CREATE VIEW Customer_Purchases AS
 SELECT Customer.Name AS Customer_Name, Customer.Gender,
 Purchase.Purchase_Date, Product.Name AS Product_Name,
 Product.Description,Product.Price
 FROM CUSTOMER, PURCHASE, PRODUCT
 WHERE CUSTOMER.CUSTOMER_ID = PURCHASE.CUSTOMER_ID AND
 PURCHASE.PRODUCT_ID = PRODUCT.PRODUCT_ID;
```

- Our NEW QUERY should look as follows:

```
SELECT Customer_Name, Gender, Purchase_Date,
Product_Name,Description,Price
FROM Customer_Purchases
WHERE Price > 1000
ORDER BY  Gender;
```

- Using Oracle SQL Developer, the DB Admin execute the querty to see the results:

1. Enter the QUERY into the worsheet window
2. Highlight & execute QUERY
3. The output window should show the result of the query.



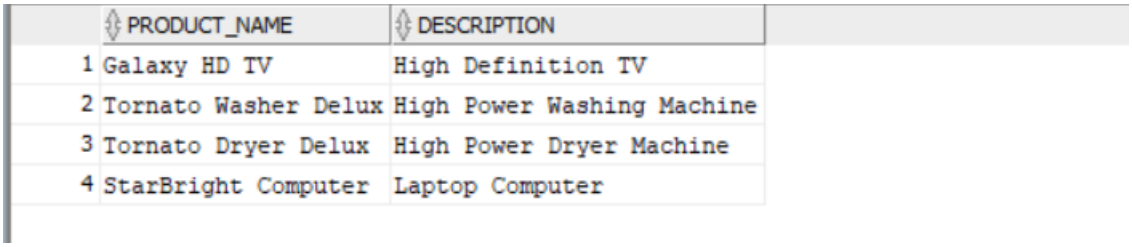
- As you can see the results are identical to the original query:

	CUSTOMER_NAME	GENDER	PURCHASE_DATE	PRODUCT_NAME	DESCRIPTION	PRICE
1	Nancy Rivera	F	24-DEC-15	StarBright Computer	Laptop Computer	1500
2	Angel Rodriguez	M	10-MAY-10	Tornato Dryer Delux	High Power Dryer Machine	2000
3	Angel Rodriguez	M	10-MAY-10	Tornato Washer Delux	High Power Washing Machine	2000

- But nevertheless since we created the VIEW correctly, we can execute any type of queries on this set of data:
- For example:

```
SELECT Product_Name,Description
FROM Customer_Purchases
WHERE Purchase_Date > '02-FEB-72';
```

  - Resulting in the following:

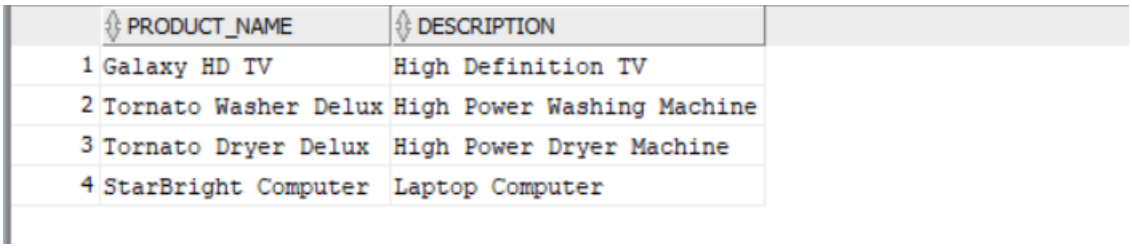


	PRODUCT_NAME	DESCRIPTION
1	Galaxy HD TV	High Definition TV
2	Tornato Washer Delux	High Power Washing Machine
3	Tornato Dryer Delux	High Power Dryer Machine
4	StarBright Computer	Laptop Computer

- For example:

```
SELECT Product_Name,Description
FROM Customer_Purchases
WHERE Purchase_Date > '02-FEB-72';
```

  - Resulting in the following:



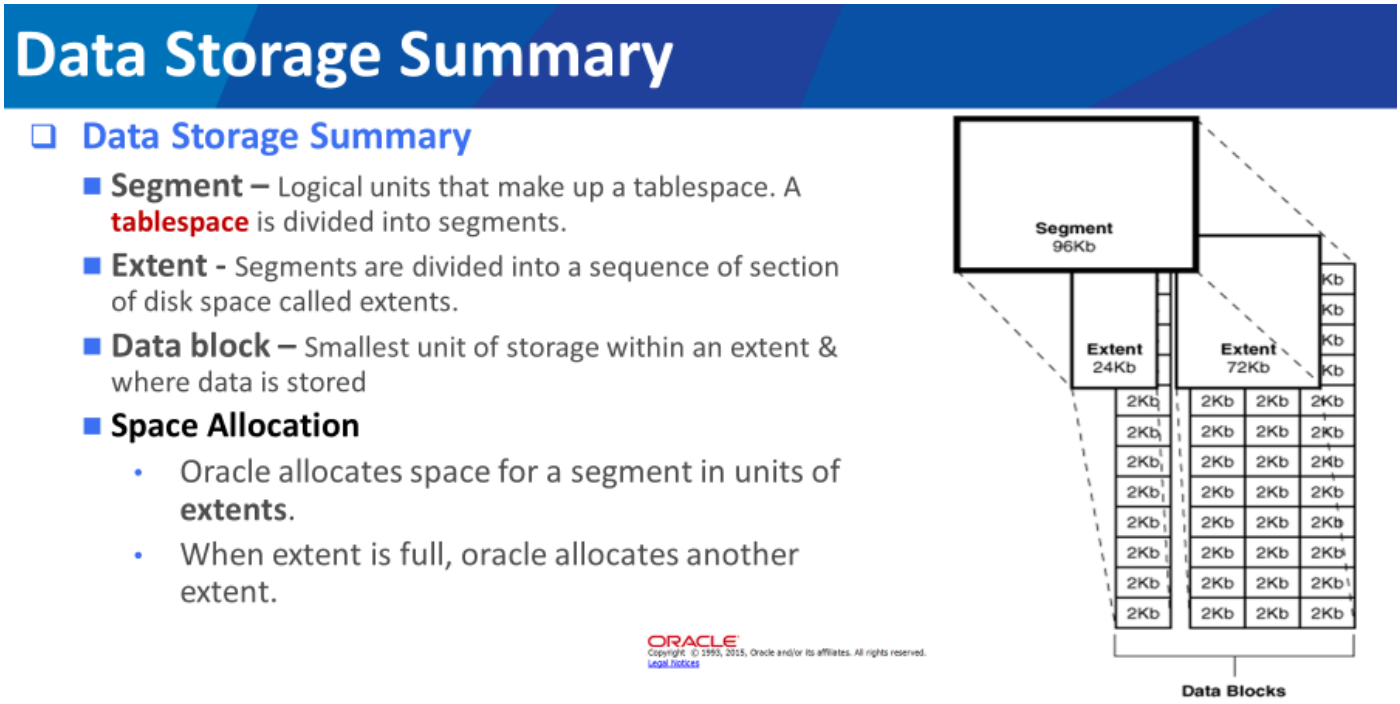
	PRODUCT_NAME	DESCRIPTION
1	Galaxy HD TV	High Definition TV
2	Tornato Washer Delux	High Power Washing Machine
3	Tornato Dryer Delux	High Power Dryer Machine
4	StarBright Computer	Laptop Computer

2.3 Physical Data Model in Oracle

2.2.1 Heap Storage Organization

- First let's define the word HEAP
  - In Oracle, a HEAP is the **default** type of database table:
    - Rows are inserted without any particular order in the first available location within the table.
    - Each row is identified by a unique ROWID, which will change if the row moves.
    - As rows are added and deleted, the row order becomes random.

- Next, we want to summarize how data is stored in Oracle



- Finally, we want to get some hands-on or how a DBA/Developer can allocate space for a table in Oracle.
  - This is done via the CREATE TABLE statement.
  - In the next section we take a look at this statement and how to apply it for storage allocation.

Create Table Statement to implement Storage Paremeters

- In previous lectures and notes we defined the CREATE TABLE STATEMENT in many variations. One of these was as follows:

```
CREATE TABLE table_name
 (
      column_name1 datatype [CONTRAINT|NULL|NOT NULL],

      column_name2 datatype [CONTRAINT|NULL|NOT NULL],

      column_name3 datatype [CONTRAINT|NULL|NOT NULL],
      ....
      ....
      CONSTRAINT constraint_name1
      FOREIGN KEY (column1, colum2, etc.)
      REFERENCES parent_table (column1, colum2, etc.),
      ....
      ....

 );
```

- As stated in the past, the syntax to the CREATE TABLE statement is quite complex and provides many features and functionality which can be implemented.
- One of these features is being able to define STORAGE PARAMETERS for:

  - Target tablespace
  - Storage parameters:

    - Initial table size
    - Expansion size if you need to add storage to a table (***Extents size*** etc.)
    - And the size for each of the expansion extents
    - Manage free storage in a BLOCK.

Syntax for creating a table with storage specifications

```
CREATE TABLE table_name
 (
      column_name1 datatype [CONTRAINT|NULL|NOT NULL],

      column_name2 datatype [CONTRAINT|NULL|NOT NULL],

      column_name3 datatype [CONTRAINT|NULL|NOT NULL],
      ....
      ....
      CONSTRAINT constraint_name1
      FOREIGN KEY (column1, colum2, etc.)
      REFERENCES parent_table (column1, colum2, etc.),
      ....
      ....

      PCTFREE value
      PCTUSED value
      TABLESPACE tablespace_name
      STORAGE (
                INITIAL value
                NEXT value
                MAXEXTENTS value
                PCTINCREASE value );
```

Clause	Description
PCTFREE	▪ The percentage of FREE SPACE in a BLOCK that must remain free to accommodate future updates. ▪ Once block is full (over the percentage of free space), the DBMS removes block from available list of blocks. Block is full, no rows can be added.
PCTUSED	▪ Defines the percentage of space in the block that must become free again for the DBMS to return the block to the list of available blocks that can sustain new rows.
TABLESPACE	▪ Target Tablespace where table will reside
STORAGE	▪ Specifies the initial size value, and values for how the table will expand.
STORAGE (INITIAL)	▪ The initial size of the extent to create (this is the FIRST EXTENT to be created)
STORAGE (NEXT)	▪ The size of the NEXT extent the DBMS system will create to expand the table (if expansion is needed this is the SECOND EXTENT to be created)
STORAGE (MAXEXTENTS)	▪ If expansion is needed, MAXEXTENTS dictates the maximum number of extents the DBMS will allocate for this table.
STORAGE (PCTINCREASE)	▪ After the SECOND EXTENT is created, every extent after that will have the following size = PREVIOUS EXTENT + a percentage dictated by PCTINCREASE value. Therefore, the value of PCTINCREASE is the percentage of increase added to the previous size for every extent created after the SECOND EXTENT.

Example

- Let's look at an example:
  - Supposed we wanted to create the following table:

Title (titleCode, titleDescription, salary)

  - But we also wanted to specify the initial size of storage to allocate and how the table should expand.
  - The syntax would look as follows:

```
CREATE TABLE Title (
   titleCode CHAR(2) PRIMARY KEY,
   titleDescription VARCHAR2(15),
   salary NUMBER CHECK (Salary BETWEEN 30000 AND 90000))
PCTFREE 10
PCTUSED 40
TABLESPACE users
STORAGE (  INITIAL 50K
           NEXT 50K
           MAXEXTENTS 10
           PCTINCREASE 25 );
```


  - The storage created for this table would be as follows:
    - INITIAL size of the table or the FIRST EXTENT = 50K
    - The NEXT or SECOND extent = 50K
    - The maximum number of extent we can have = 10
    - The PCTINCREASE or percentage to increase every extent after the SECOND EXTENT = 25%
    - After the SECOND EXTENT, every extent gets the size = PREVIOUS EXTENT + 25%
    - Therefore, the THIRD EXTENT = 50K + (50K*.25) = 62.5K
    - The FOURTH EXTENT = 62.5K + (62.5K*.25) , the FIRTH EXTENT = PREVIOUS + (PREVIOUS * .25) , and so on… until a maximum of 10 extents created.

  - Block management:
    - PCTFREE or size of free space in a block = 10%
      - This means 90% of block space is available for row inserts and 10% free for updates of existing rows.
    - PCTUSED = 40%, which means NO NEW ROWS can be added until amount of used space falls below 40%

- Characteristic of block size management:
    - The choice of parameter values selected can increase performance of writing and reading operations on tables, indices, etc.
    - The Oracle documentation provides guidance on how to select the values of PCFREE & PCUSED. The choices depend on the type of activities on the table.
    - Guidance:
        - Frequent UPDATES PCTFREE = 20 & PCTUSED = 40 is appropriate
        - For table that does little or no update, insert & delete, PCTFREE = 5 & PCTUSED = 90

2.4 Introduction to the Distributed Database Model

2.4.1 Centralized Database

- Before analyzing the distributed model, let's review what the centralize model looks like and its pro's & cons.
  - Assuming the we have the Manufacturing Company data model use case from Malyuta' book:

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

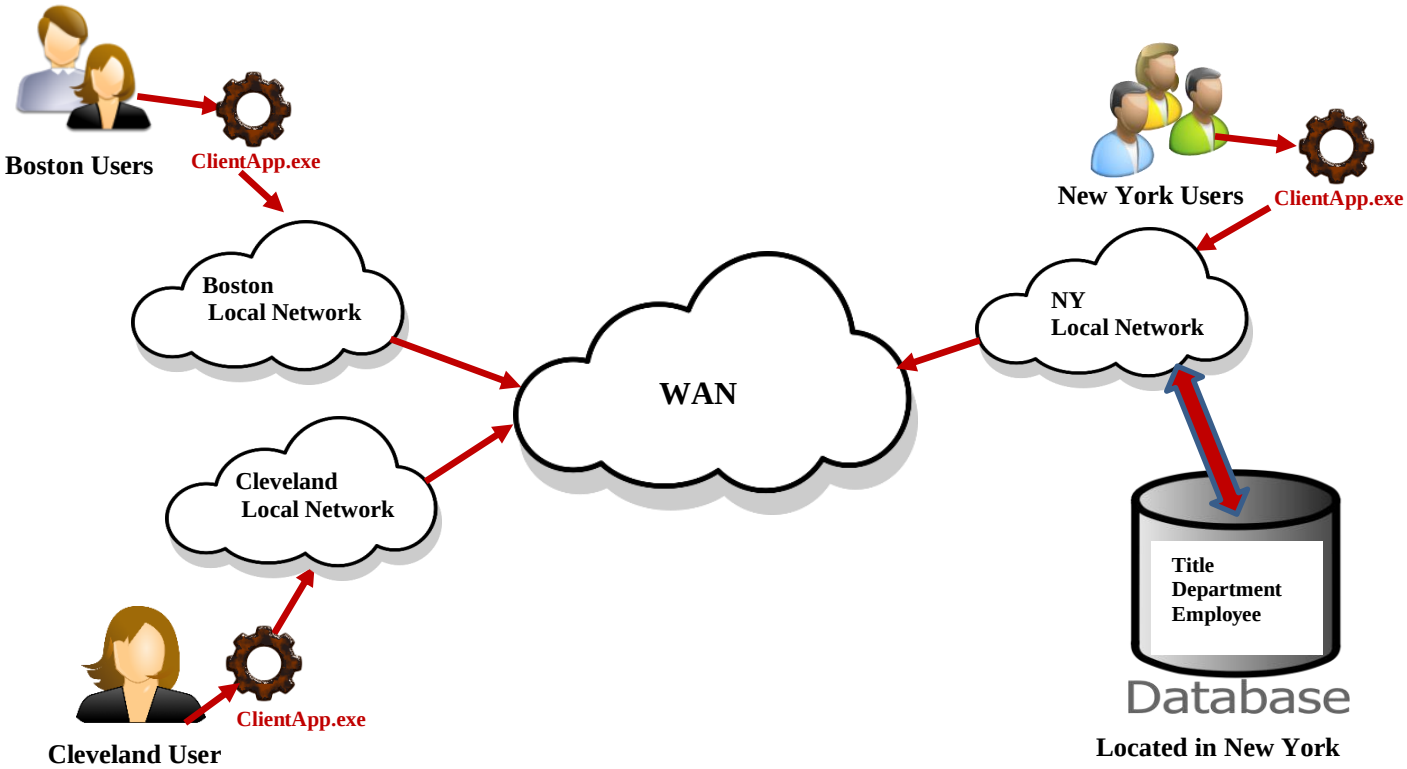
Department

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

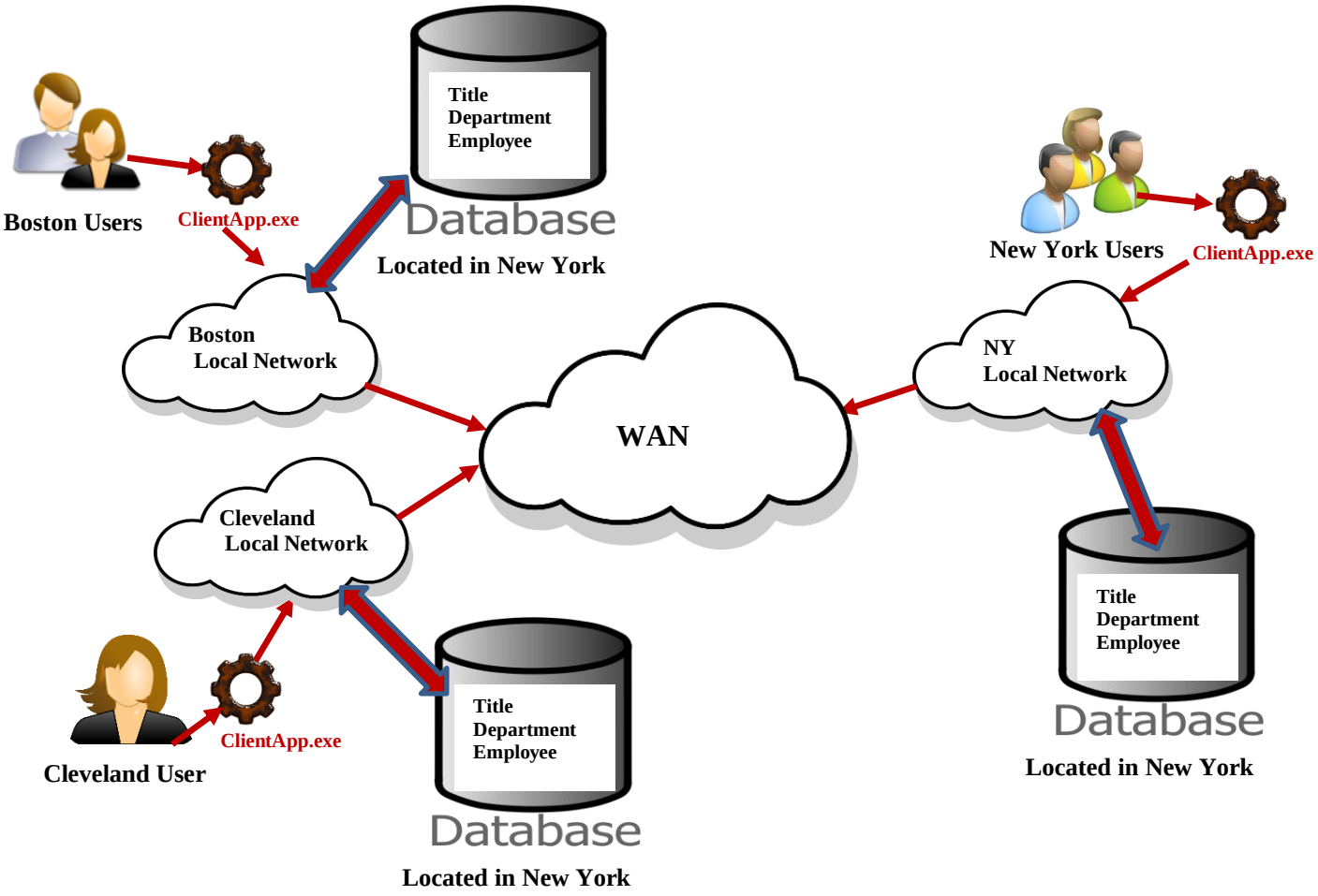
ID	Empl Name	Empl Type	Dept Code	Title Code
1	John	Full-time	002	T1
2	Adam	Consultant	001	T3
3	Mary	Part-time	004	T4
4	Peter	Full-time	003	T2
5	Scott	Consultant	002	T1
6	Susan	Full-time	005	T5
7	Alex	Part-time	004	T2

  - 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. Whereas for NY users, they may reside in the same local network as the centralized database and won't be affected.

- To address these network latency issues, you may be tempted to simply place a COPY of the entire database in each location as follows:



- Nevertheless, this creates other issues of INCONSISTENCY since every database is isolated and separate.
- To solve this INCONSISTENCY, DBMS can replicate their data by creating a LINK between each database.
- This will solve the problem, but now you could have serious NETWORKING PERFORMANCE ISSUES due to the replication traffic between DBMS servers which would have to replicate every change in real-time!!!

2.4.2 Distributed Database Model

- The distributed data model offers a variety of design options & benefits

  - Assuming the we have the Manufacturing Company data model use case from Malyuta' book in the following distributed configuration:

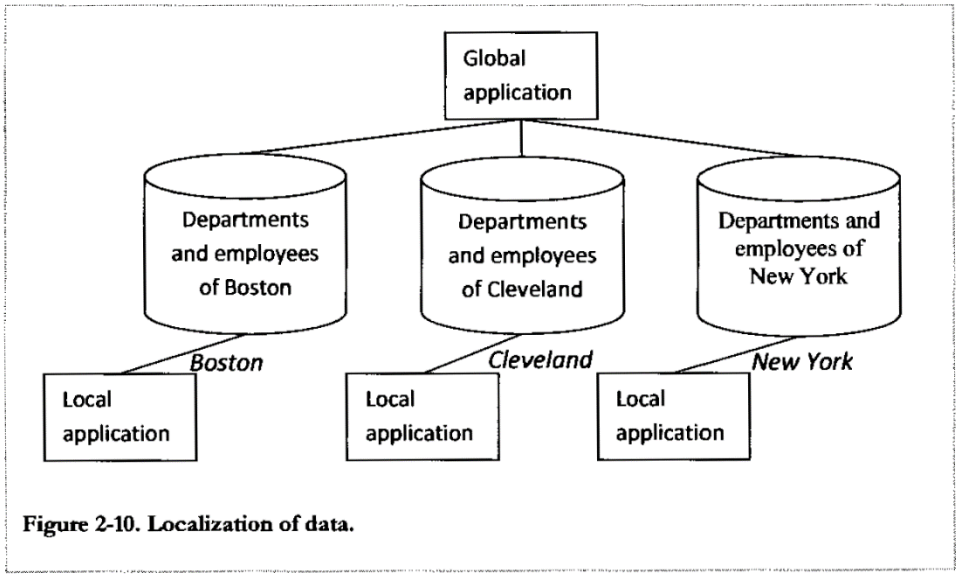


Figure 2-10. Localization of data.

Characteristics of the Distributed Data Model

- Characteristics of the distributed model:

  - Data is physically distributed in several databases across various 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 and 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.

Advantages of the Distributed Data Model

- Advantages of the distributed model include:

  - Better **performance**.:

    - **Local Data** – Data is localized where is needed or in greater demand:
      - Each office in the Commercial Manufacturing company are in separate cities (New York, Boston, Cleveland).
      - We can distribute data for each city locally. Not necessarily the entire tables for just fragments. Now each city can consume only the data pertinent to each department in that location.

    - **Smaller Set of Data** – Each local application searches/processing is done on a smaller amount of data (Table or fragment)
      - Smaller set of data means better performance and faster user-response
      - Managing smaller data is simpler.

    - **Global Applications** – Application that requires data from all distributed database:
      - Global applications can also benefit from distributed databases.
      - For example, if a global application needs data for all EMPLOYEE, in all DEPARTMENT for a TITLE on all 3 cities, then the GLOBAL REQUEST can be made to all three databases in PARALLEL (simultaneous request).
      - This parallel execution is known as INTRA-QUERY PARALLELISM.
      - Parallel processing of smaller data set is faster than a centralized model.

    - **Parallelism** –Data/Databases are parallelized thus balancing the load on the databases

    - **Network Traffic/Load** –Network load is minimized with distributed database since data is localized, thus other applications/systems can benefit from less load on the network.

      - Better **Scalability**: In a distributed solution, expansion of a database or an increase in the number of database users is easier to handle than in a centralized one:

    - **Server Scalability & Maintenance** –
      - New database servers can be added in one city's database without affecting the other cities database. In centralize model, maintenance may require downtime.
      - Easier to maintain & manage the distributed databases unlike centralize.

    - **Reliability and availability** is improved since for example New York was to go down, Boston and Cleveland can still function.

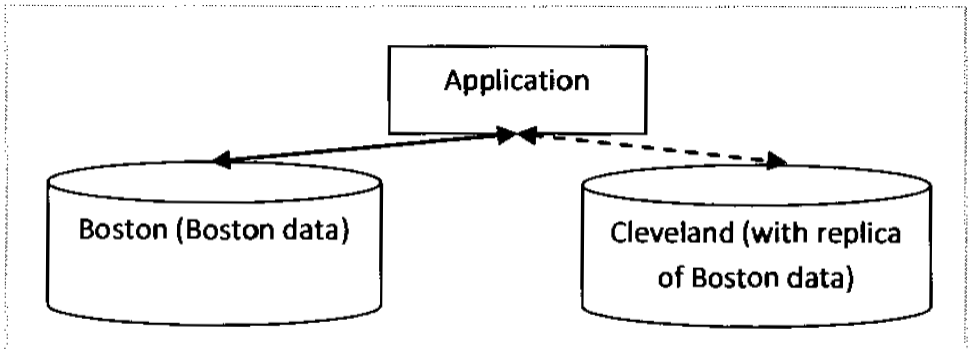


Figure 2-11. Replication of data.

    - If the Boston database goes down or even is destroyed, the application can switch to the Cleveland database that keeps a **synchronized copy** of the Boston data (replica).
    - Again, maintenance is required in a database in Cleveland, it can happen without affecting Boston or New York.

Disadvantages of the Distributed Data Model

- Disadvantages of the distributed model include:

  - **Lack of standards and Methodology**
    - No straightforward rules or methodology on how to design, but trial an error
    - No standards no standards or architecture regulations on how to convert centralized database to distributed

  - **More complex to design**
    - Limited by solution provided by DBMS
    - Complex implementation of integrity constraints etc.
    - No set standards or methodology for distributed design
    - Requires more hardware, thus, **higher cost!**
    - Data transfer (synchronization between databases)

  - **Complexity of Fragmentation**
    - Decision on how to split tables into fragments, allocating fragments, managing fragments are difficult.
    - Fragments replication may be challenging and can decrease performance if not done right.
    - Decision on how to distribute or fragment data can be complex. For example:

      - In our Manufacturing Company example, supposed we have a new application with the following requirements:

        - Boston Users – Users of this application process data about all full-time employees
        - New York Users – Process data about part-time employee
        - Cleveland Users – Process data about consultants.
        - Because each city users must process data of employee of a type must process data across all three cities.
        - As you can see in diagram below, the new application requirements force the users from each city thus availability of data can be at risk due to various locations:

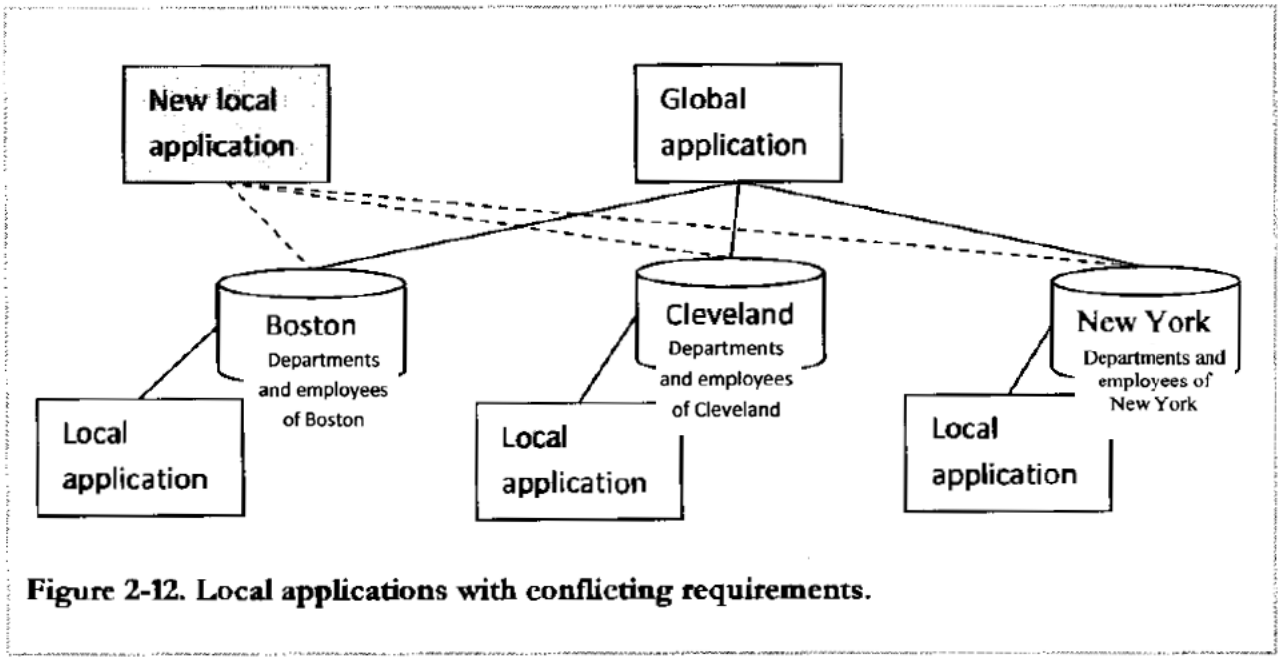


Figure 2-12. Local applications with conflicting requirements.

        - If decision is made to add additional fragmentation by fragmenting the data needed for new APPLICATION 2, then you will affect the first local APPLICATION 1.
        - This complicates matter and decisions need to be made on which application takes priority or reach an agreeable compromise.

2.4.3 Overview of the 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 components 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.
    - The connection specifies the NAME & LOCATION of the remote database.
    - This is how Oracle makes each database in a distributed model visible to one another.
    - Of course, you need a physical network and this Database Link leverages this connection to establish a link between databases.
    - A database link is a SCHEMA OBJECT.

- Look at this MODIFIED illustration from Malyuta' s book below for our Manufacturing Company example:

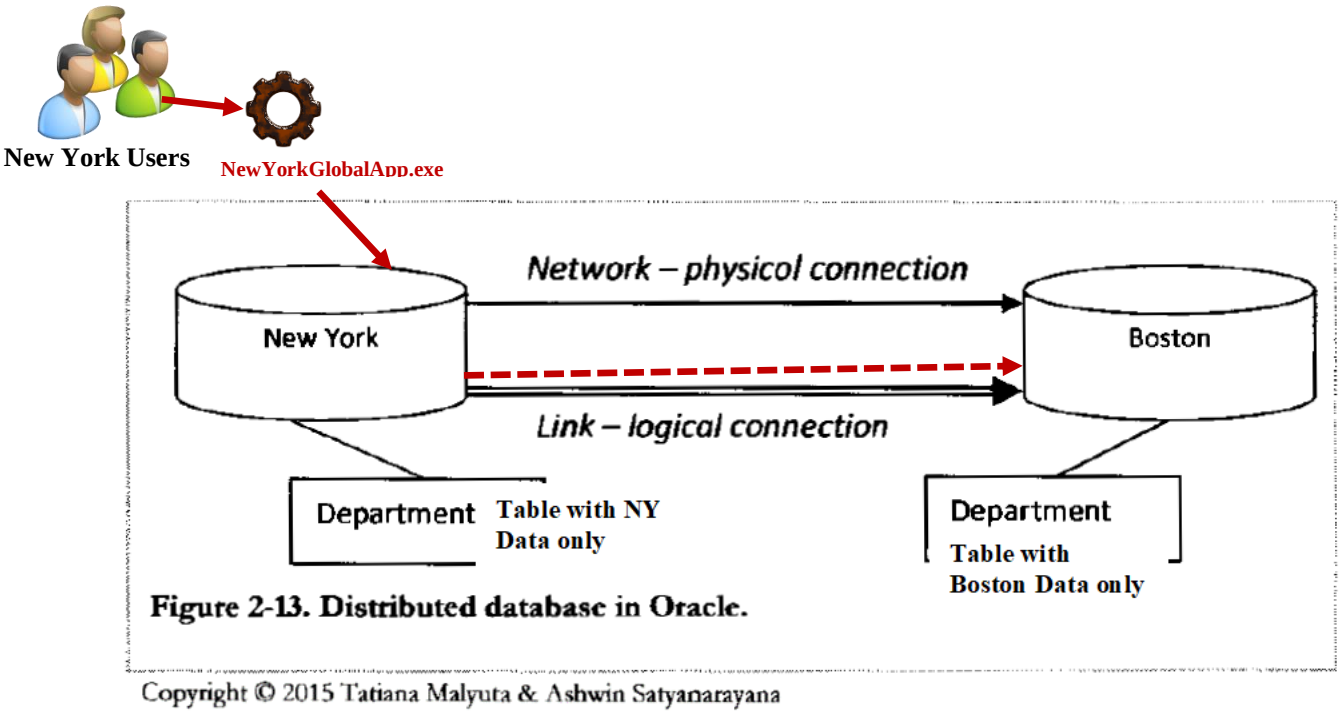


Figure 2-13. Distributed database in Oracle.

- Note the following:

  - We assume, that there is a new global application being executed from NY, that must retrieve data about all departments, therefore it must retrieve data from its local database in NY and must retrieve data from the Boston department table as well.
  - To do so, we need to create a **Database Link** between the New York database and the Boston database via the physical network.

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
```

Clause	Description
CREATE	▪ CREATE keyword.
PUBLIC	▪ A database link public to all users.
SHARED	▪ A database link that can be shared by multiple sessions or connection using one physical network connection from source database to target database. ▪ Once a session is established on the target database, that session is disassociated from the connection, to make the connection available to another session on the source database
dbLink_name ***or*** ***dbLink_name.Domain ServiceName***	▪ Name of Database Link. Domain Service Name. Case-sensitive. Two options: ○ Partial short name –first part of service name or DNS name. Oracle system will complete the full-service name of the partial name provided. For example, if using partial name BostonConn: **CREATE PUBLIC DATABASE LINK BostonConn USING boston.ourcompany.us.com** ○ Full Service (DNS) Name – you explicitly add the full domain service name to Database Link name & Oracle does not have to do it automatically. In following example, you list the full name for Database Link **CREATE PUBLIC DATABASE LINK BostonConn.ourcompany.us.com USING boston.ourcompany.us.com**
DATABASE LINK	▪ DATABASE LINK keyword

Clause	Description
USING dbLink_name ***Or*** ***USING dbLink_name.DomainServiceName***	▪ Name of remote database connection string. Name can contain first part of DNS name or full Domain Service Name. Case-sensitive. Two options: ○ Partial short connection string name –first part of service name or DNS name. Oracle system will complete the full-service name of the partial name provided. For example, using partial name **Boston** for connection string of remote database: **CREATE PUBLIC DATABASE LINK BostonConn USING Boston** ○ Full Service (DNS) Name – you explicitly add the full domain service name to Database Link name & Oracle does not have to do it automatically. In following example, you list the full name for remote database: **CREATE PUBLIC DATABASE LINK BostonConn USING boston.ourcompany.us.com**

Using the DATABASE LINK after creation

❑ Once the **Database Link** is created 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 with WHERE clause
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.