BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DATA WAREHOUSING LAB-1
[Basics of SQL & Working with Multi-dimensional Databases]

DATE: 23/03/2021 **TIME: 02 Hours**

Consider the Northwind database (a sample database used by Microsoft) whose schema is given in Figure 1. The database is about a fictitious company named, Northwind Traders. The database captures all the transactions that occur between the company i.e. Northwind traders and its customers and also, the purchase transactions between Northwind and its suppliers.

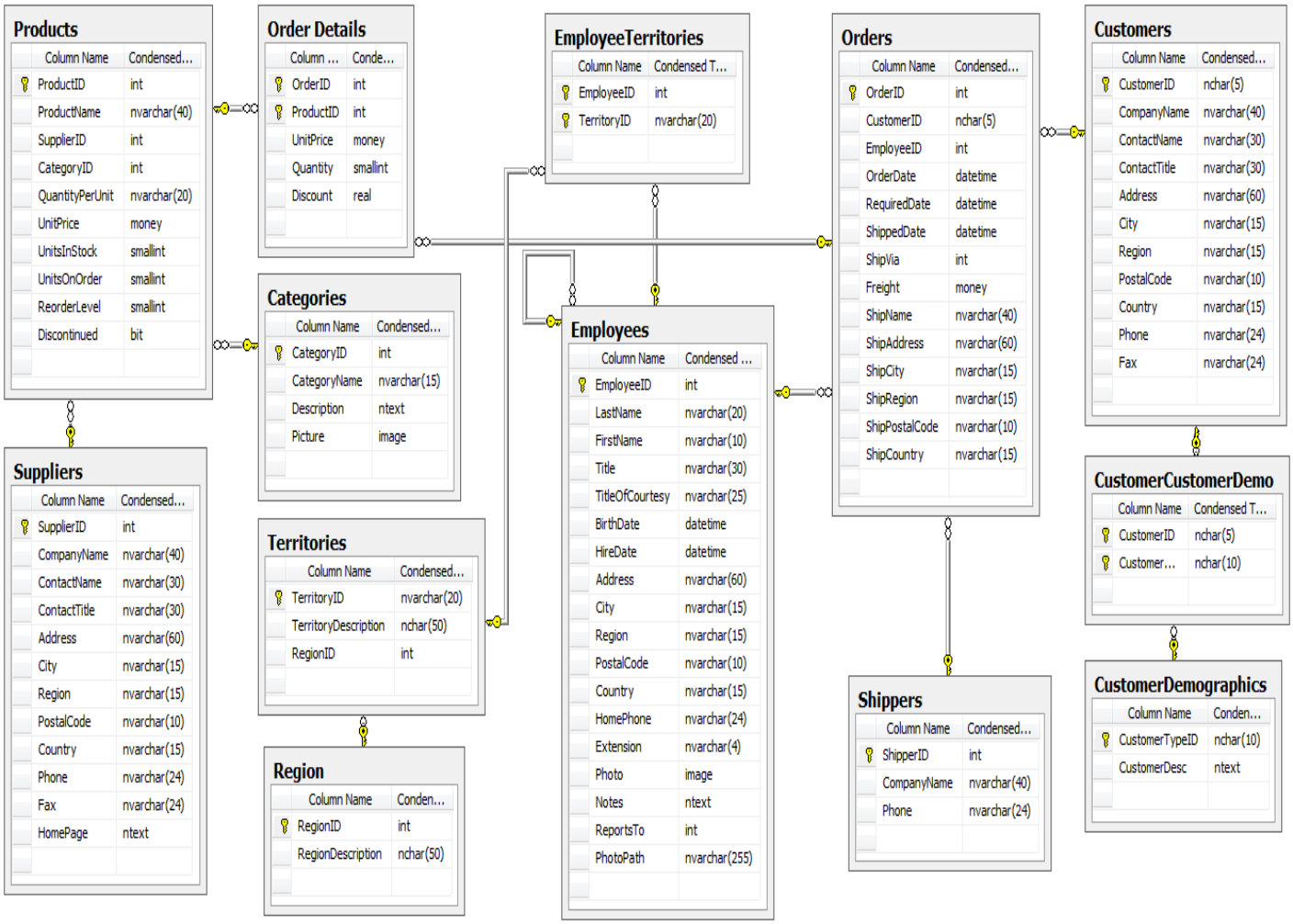


Fig 1. Northwind Database Schema Design

IMPORTING NorthWind DATABASE TO MS SQL SERVER:

Step 1: Right Click on the Databases folder in Object Explorer and select 'Restore Database' option.

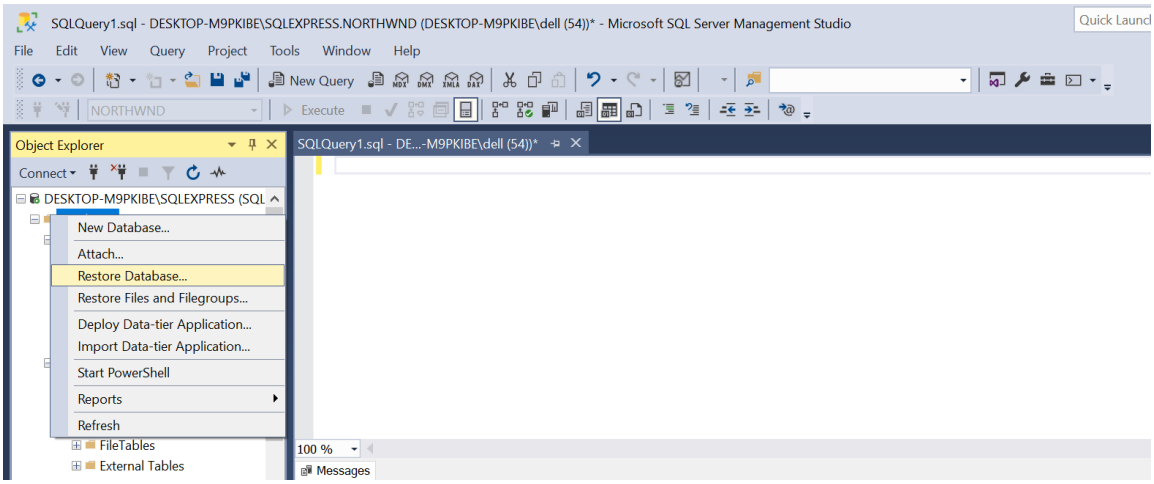
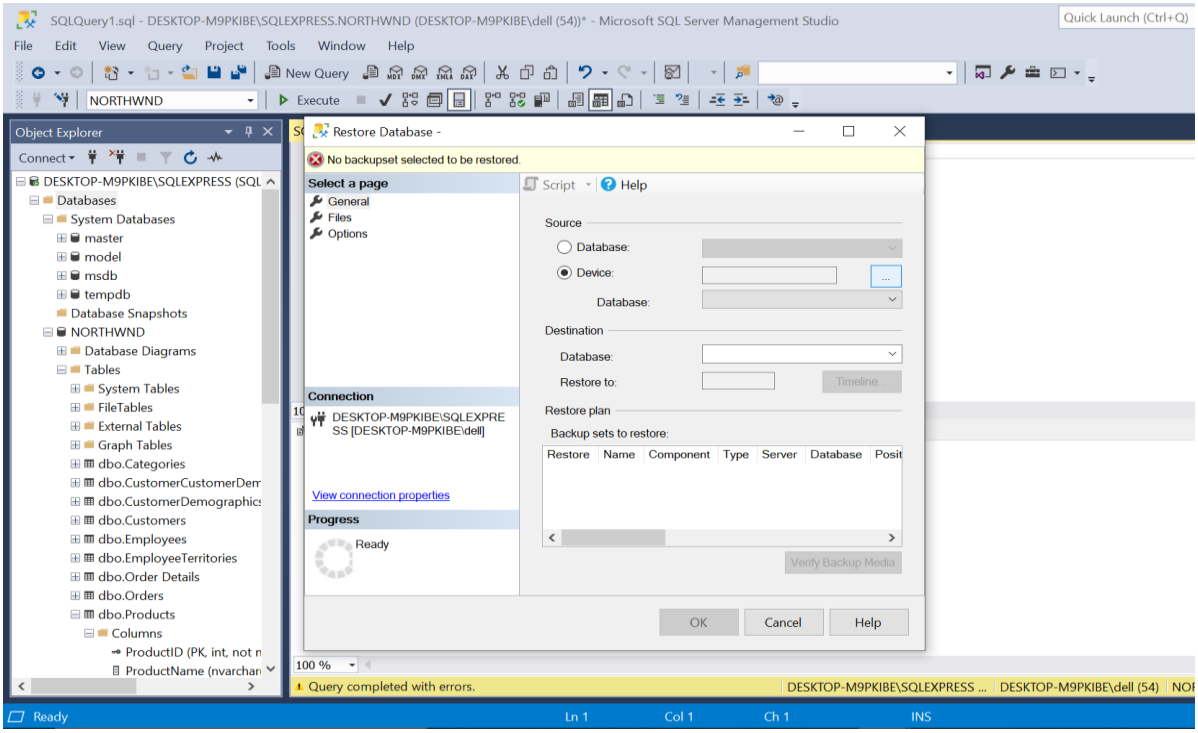
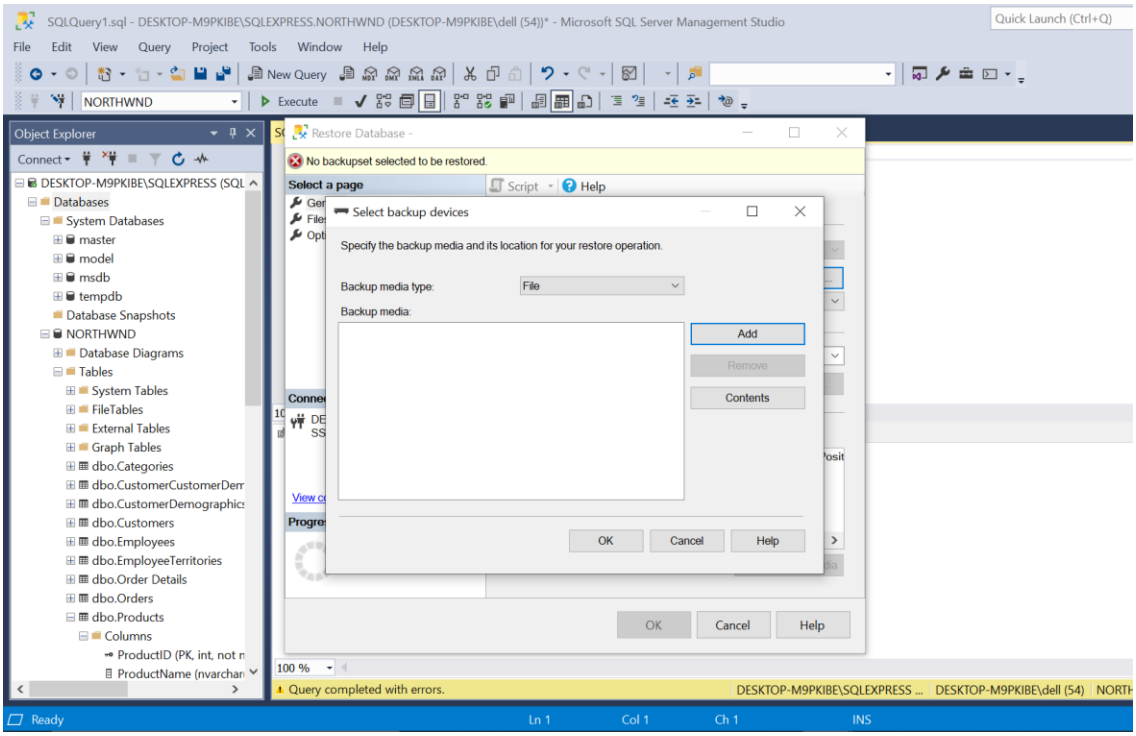


Fig 2. Step-1 to import northwind database

Step 2: A dialog box will open as shown in the figure given below. Select 'Device' option as the database backup file 'northwind.bak' is stored on the device. Click on Add and browse to the location of the 'northwind.bak' file. Click on OK. The Northwind database is now ready to be used.





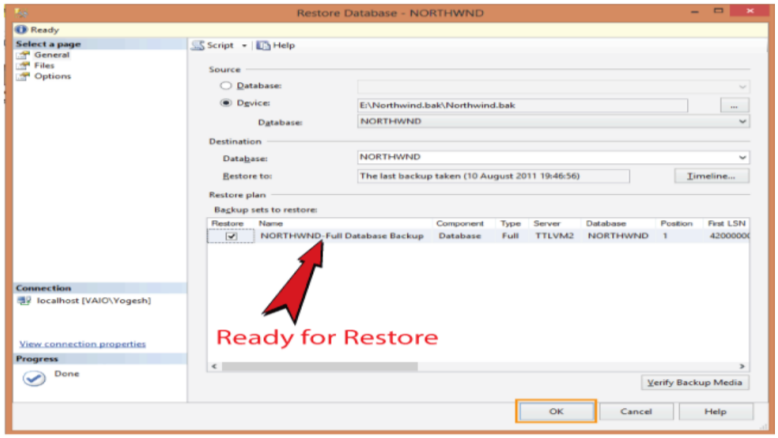


Fig 3. Step-2 to import northwind database

1. Statements in SQL
    a CREATE TABLE: It lets the user to create a table by specifying the names of the columns and their corresponding data type.

```
CREATE TABLE table_name
(column_name1 data_type,
column_name2 data_type,
column_name3 data_type
);
```

    b INSERT: The INSERT statement enables the user to enter records (rows) in the table. It can be done in two ways-

```
INSERT INTO table_name
VALUES (value1, value2, value3)

INSERT INTO table_name (column1,column2)
VALUES (value1,value2)

--example

INSERT INTO Customers(CustomerID, CompanyName,
ContactName, City, Country)
values('DW', 'Bon app',
'DataWarehouse','Pilani','INDIA');
```

    c SELECT: The SELECT * statement retrieves all the rows from a given table. SELECT keyword followed by the column names is used to fetch only the specified column entries.

```
SELECT * FROM table_name
SELECT column_name1,column_name2 FROM table_name

--example
select * from Customers;
select ContactName, City from Customers;
```

    SELECT DISTINCT is used when only the unique column entries are to be fetched from the given table.

```
SELECT DISTINCT column_name FROM table_name
```

```
--example
select distinct Country from Customers;
```

d ALTER: This is used to make modifications to the already existing table in a database. For example, to add an attribute (column) in a table-

```
ALTER TABLE table_name ADD column_name datatype;
```

e DELETE: DELETE is used to delete the existing records (rows) from the table. This does not delete the table from the database. Only those records that satisfy the condition mentioned in 'WHERE' clause will be deleted from the table. If no WHERE clause is specified, all the records will be deleted.

```
DELETE FROM table_name
WHERE column_name=value

--example
DELETE FROM Customers where ContactName='DataWarehouse';
```

f SELECT TOP: The SELECT TOP clause allows you to limit the number of rows or percentage of rows returned in a query result set. Because the order of rows stored in a table is unpredictable, the SELECT TOP statement is always used in conjunction with the ORDER BY clause. As a result, the result set is limited to the first N number of ordered rows.

```
SELECT TOP 3
    ProductName, UnitPrice
    FROM Products
    ORDER BY UnitPrice ASC;
```

2. COLUMN CONSTRAINTS

a PRIMARY KEY and NOT NULL: A primary key is a field that uniquely identifies each record (row) in a table. The value of the primary key must be unique and should be not null. A table can have only one primary key but it may contain one or more number of columns.

```
CREATE TABLE STUDENTS(
      BITS_ID int NOT NULL PRIMARY KEY,
```

```
      FIRST_NAME varchar(100),
      LAST_NAME varchar(100),
      AGE int
);
```

To define primary key constraint on multiple columns,

```
CREATE TABLE STUDENTS(
      BITS_ID int NOT NULL,
      FIRST_NAME varchar(100) NOT NULL,
      LAST_NAME varchar(100),
      AGE int,
      CONSTRAINT PK_Students PRIMARY KEY
(BITS_ID,FIRST_NAME)
);
```

b FOREIGN KEY: A foreign key constraint establishes a connection between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the referenced table.

```
CREATE TABLE STUDENTS_BILL(
      ORDER_ID int NOT NULL PRIMARY KEY,
      ORDER_NUMBER int NOT NULL UNIQUE,
      BITS_ID int FOREIGN KEY REFERENCES
STUDENTS(BITS_ID)
);
```

c UNIQUE: The UNIQUE constraint ensures that all values in a column are different. Both the UNIQUE and PRIMARY KEY constraints ensures the uniqueness for a column or set of columns. However, a table can have multiple columns with UNIQUE constraint but only a single PRIMARY KEY.

d DEFAULT: The DEFAULT constraint assigns a default value to a column when no value has been assigned to it.

```
CREATE TABLE STUDENTS(
      BITS_ID int NOT NULL PRIMARY KEY,
      EMAIL varchar(500) NOT NULL UNIQUE,
      FIRST_NAME varchar(100),
      LAST_NAME varchar(100),
      AGE int,
```

```
      City varchar(100) DEFAULT='Delhi'
);
```

3. CLAUSES
   a. AS: Columns or tables in SQL can be aliased using the AS clause. This allows columns or tables to be specifically renamed in the returned result set.
   NOTE: It requires double quotation marks or a square bracket if the alias name contains spaces.

```
SELECT CustomerID as ID, CustomerName as Customer from
Customers;
```

   b. WHERE: The WHERE clause is used to filter records (rows) that satisfy the specified condition.

```
SELECT * FROM Customers WHERE CustomerID = 'ALFKI';
SELECT * FROM Customers WHERE Country='Mexico';
```

   c. ORDER BY: The ORDER BY clause can be used to sort the result set by a particular column either alphabetically or numerically. It can be ordered in ascending (default) or descending order with ASC/DESC.

```
SELECT * from CUSTOMERS
ORDER BY COUNTRY DESC;
--order by several columns

SELECT * from Customers
ORDER BY Country DESC, ContactName ASC;
```

   d. GROUP BY: The GROUP BY clause will group records in a result set by identical values in one or more columns. It is often used in combination with aggregate functions to query information of similar records. The GROUP BY clause can come after FROM or WHERE but must come before any ORDER BY or LIMIT clause.

   e. HAVING: The HAVING clause is used to further filter the result set groups provided by the GROUP BY clause. HAVING is often used with aggregate functions to filter the result set groups based on an aggregate property.

4. OPERATORS

a AND: It is a logical operator that allows you to combine two Boolean expressions. It returns TRUE only when both expressions evaluate to TRUE .

```
SELECT * FROM Customers
WHERE Country = 'Germany' AND City = 'BERLIN';
```

b OR: OR returns a record if any of the specified statements are true.

```
SELECT * FROM Customers
WHERE City = 'Aachen' OR City = 'BERLIN';
```

c NOT: NOT operator displays a record if the condition is NOT TRUE.

```
SELECT * FROM Customers
WHERE Country <> 'Mexico';
```

d Combining AND, OR, NOT

```
--example1
SELECT * FROM Customers
WHERE Country = 'Germany' AND (City = 'Aachen' OR City =
'BERLIN');

--example2
SELECT * FROM Customers
WHERE Country <> 'Germany' AND Country <> 'USA';
```

e IS NULL/ IS NOT NULL: A field with a NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value. To check for a NULL value, these functions can be used.

```
SELECT * FROM Customers
WHERE ContactName IS NULL;

--NOT NULL
SELECT * FROM Customers
WHERE RegionIS NOT NULL;
```

f IN/NOT IN: IN condition checks if an expression matches any value in a list of specified arguments. It is used in place of multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

g BETWEEN/ NOT BETWEEN: The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT * FROM Products
WHERE UnitPrice BETWEEN 10 AND 20;
```

5. SQL FUNCTIONS

There are two types of SQL functions known as, aggregate functions and scalar (non-aggregate functions). Aggregate functions operate on many records to produce a summary while, scalar functions operate on each record independently.

a Aggregate Functions: These functions can produce a single value for an entire group or table. They operate on sets of rows and return results based on groups of rows.

i MIN() / MAX()

```
SELECT MIN(UnitPrice) as SmallestPrice FROM Products

SELECT MAX(UnitPrice) as LargestPrice FROM Products;
```

i COUNT()

```
SELECT COUNT(ProductID) AS TotalNumberOfProducts FROM
Products;
```

i SUM() / AVG()

```
SELECT SUM(Quantity) AS TotalQuantityOfOrderDetails FROM
[Order Details];

SELECT AVG(UnitPrice) AS AvgPrice FROM Products;
```

b Arithmetic Functions:

i ABS() : It is used to get the absolute value of the number passed as an argument.

i CEILING()/FLOOR(): The ceil() function outputs the smallest integer which is greater than or equal to the input numeric expression. On the

other hand, floor() outputs the integer that is less than or equal to the argument passed in the function.

iii. POWER(): This function returns the value of a number raised to the power of another number. Both the numbers have to be passed as arguments to this function.

iv. SQRT(): This function returns the square root of the given number.

```
SELECT ABS(-17.36) AS ABS_VALUE;

SELECT (CEILING(17.36)) AS CEIL_VALUE;

SELECT (FLOOR(17.36)) AS FLOOR_VALUE;

SELECT POWER(2,3) AS POWER_VALUE;

SELECT SQRT(36) AS SQ_ROOT;
```

c. Character Functions:

i. LOWER(): It converts all the characters of the string passed to the function in lower case.

ii. UPPER(): It converts all the characters of the string passed to the function in upper case.

```
SELECT LOWER('CONVERTING THE STRING TO LOWERCASE') AS
lower_func;

SELECT UPPER('converting the string to upper case') AS
upper_func;
```

6. SUBQUERIES: A subquery is a query embedded in another query. These are generally used when the tables have some kind of relationship. The subquery can contain any valid SELECT statement, but it must return a single column with the expected number of results. For example, if the subquery returns only one result, then the main query can check for equality, inequality, greater than, less than, etc. On the other hand, if the subquery returns more than one record, the main query must check to see if a field value is (or is NOT) IN the set of values returned. For example, in the northwind database, the Orders table has a CustomerID field that refers to the Customers table.

Example1: Retrieve a CustomerID for an order number 10290.

```
SELECT CompanyName FROM Customers
WHERE CustomerID = (SELECT CustomerID FROM Orders
                    WHERE OrderID=10290);
```

Example2: Retrieve all products by name that are in the Seafood category.

```
SELECT  ProductName FROM Products
WHERE CategoryID = (SELECT CategoryID FROM Categories
                    WHERE CategoryName = 'Seafood');
```

Example3: Retrieve all companies by name that sell products in the Seafood category.

```
SELECT CompanyName FROM Suppliers
WHERE SupplierID IN( SELECT SupplierID FROM Products
       WHERE CategoryID = (SELECT CategoryID FROM Categories
                    WHERE CategoryName = 'Seafood'));
```

7. DATE Functions:
    a DATEPART(): Returns the date part, as an integer, of the specified date.
    b MONTH(): Returns the month part (a number from 1 to 12) for the date passed as an argument.
    c YEAR(): Returns the year part for the date passed as an argument.
    d DATEDIFF(): Returns the difference between two dates.
    e ISDATE(): Checks if the argument is a valid date. Returns 1 in case of valid and 0 otherwise.

```
SELECT YEAR(OrderDate) AS year, MONTH(OrderDate) AS Month
       FROM Orders;
```

Exercises:

Q1. Select the number of sales per category and country.

Q2. Select the 3 top-selling categories overall (hint: use "select top 3" construction).

Q3. List the total amount of sales in $ by employee and year (discount in OrderDetails is at UnitPrice level). Which employees have an increase in sales over the three reported years?

Q4. Get an individual sales report by month for employee 9 (Dodsworth) in 1997.

Q5. Get a sales report by country and month.