Database I

SQL (Microsoft T-SQL)

Etelka Szendrői (Phd)
szendroi@mik.pte.hu

Subqueries

Working with subqueries

Subqueries are nested queries or queries within queries

Results from inner query are passed to outer query

- Inner query acts like an expression from perspective of outer query

Subqueries can be self-contained or correlated

- Self-contained subqueries have no dependency on outer query
- Correlated subqueries depend on values from outer query

Subqueries can be scalar, multi-valued, or table-valued

Writing scalar subqueries

Scalar subquery returns single value to outer query

Can be used anywhere single-valued expression can be used: SELECT, WHERE, etc.

Display the last order lines:

```
SELECT OrderID, ProductID, UnitPrice, Quantity
FROM [Order Details]
WHERE OrderID =
(SELECT MAX(OrderID) AS LastOrder
FROM Orders);
```

If inner query returns an empty set, result is converted to NULL

Construction of outer query determines whether inner query must return a single value

Writing multi-valued subqueries

Multi-valued subquery returns multiple values as a single column set to the outer query

Used with IN predicate

If any value in the subquery result matches IN predicate expression, the predicate returns TRUE

Retrieve those orders which was placed by a customer from UK:

```
SELECT CustomerID, OrderId, OrderDate
FROM Orders
WHERE CustomerID IN (
SELECT CustomerID
FROM Customers
WHERE Country = 'UK');
```

Writing queries using EXISTS with subqueries

The keyword EXISTS does not follow a column name or other expression.

The SELECT list of a subquery introduced by EXISTS typically only uses an asterisk (*).

```
SELECT CustomerID, CompanyName
FROM Customers AS Cust
WHERE EXISTS (
SELECT *
FROM Orders AS Ord
WHERE Cust.CustomerID = Ord.CustomerID);
```

Those customers who has at least one order

```
SELECT CustomerID, CompanyName
FROM Customers AS Cust
WHERE NOT EXISTS (
SELECT *
FROM Orders AS Ord
WHERE Cust.CustomerID = Ord.CustomerID);
```

Those customers who has not any orders

Scalar subqueries

Show the cheapest product(s)!

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
WHERE UnitPrice =
    (SELECT MIN(UnitPrice)
    FROM Products);
```

First we need the minimum value

Results | Messages

	ProductID	ProductName	UnitPrice
1	33	Geitost	2,50

7

Multi-valued subquery

List the ProductId and ProductName of products, which supplied by Japan supplier.

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
WHERE SupplierID IN
    (SELECT SupplierID
    FROM Suppliers
    WHERE Country ='Japan');
```

Results | Messages

	ProductID	ProductName	UnitPrice
1	9	Mishi Kobe Niku	97,00
2	10	Ikura	31,00
3	74	Longlife Tofu	10,00
4	13	Konbu	6,00
5	14	Tofu	23,25
6	15	Genen Shouyu	15,50

8

JOIN Statements

Overview of JOIN types

JOIN types in FROM clause specify the operations performed on the virtual table:

Join Type	Description
Cross	Combines all rows in both tables (creates Cartesian product).
Inner	Starts with Cartesian product; applies filter to match rows between tables based on predicate.
Outer	Starts with Cartesian product; all rows from designated table preserved, matching rows from other table retrieved. Additional NULLs inserted as placeholders.

Understanding INNER JOINS

Returns only rows where a match is found in both tables
Matches rows based on attributes supplied in predicate
- ON clause in SQL-92 syntax

Why filter in ON clause?
- Logical separation between filtering for purposes of JOIN and filtering results in WHERE
- Typically no difference to query optimizer

If JOIN predicate operator is =, also known as equi-join

INNER JOIN Syntax

List tables in FROM Clause separated by JOIN operator
Table order does not matter, and aliases are preferred

```
FROM t1 JOIN t2
   ON t1.column = t2.column
```

```
SELECT SOH.SalesOrderID,
       SOH.OrderDate,
       SOD.ProductID,
       SOD.UnitPrice,
       SOD.OrderQty
FROM Sales.SalesOrderHeader AS SOH
JOIN Sales.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID;
```

Emploees table (emp)

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800.00	0.00	20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600.00	300.00	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250.00	500.00	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975.00	0.00	20
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250.00	1400.00	30
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850.00	0.00	30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450.00	0.00	10
8	7788	SCOTT	ANALYST	7566	1987-04-19 00:00:00.000	3000.00	0.00	20
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000.00	0.00	10
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500.00	0.00	30
11	7876	ADAMS	CLERK	7788	1987-05-23 00:00:00.000	1100.00	0.00	20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950.00	0.00	30
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000.00	0.00	20
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300.00	0.00	10

Salgrade table

	GRADE	LOSAL	HISAL
1	1	700.000	1200.000
2	2	1201.000	1400.000
3	3	1401.000	2000.000
4	4	2001.000	3000.000
5	5	3001.000	9999.000

Department (dept)

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	50	Kereskedes	Pecs

Display the name of the employees and the department name they work for!

```
SELECT  emp.ename As név, dept.dname As "Részleg neve"
FROM emp, dept
WHERE emp.deptno=dept.deptno;
```

INNER JOIN operation:

	név	Részleg neve
1	SMITH	RESEARCH
2	ALLEN	SALES
3	WARD	SALES
4	JONES	RESEARCH
5	MARTIN	SALES
6	BLAKE	SALES
7	CLARK	ACCOUNTING
8	SCOTT	RESEARCH
9	KING	ACCOUNTING
10	TURNER	SALES
11	ADAMS	RESEARCH
12	JAMES	SALES
13	FORD	RESEARCH
14	MILLER	ACCOUNTING

```
SELECT  emp.ename As név, dept.dname As "Részleg neve"
FROM emp INNER JOIN dept ON emp.DEPTNO=dept.DEPTNO;
```

List the name, department name and department location of those employees, whose name contains R letter.

```
SELECT  emp.empno As Azonosító, emp.ename As név, dept.dname As "Részleg neve",
dept.loc As telephely
FROM emp INNER JOIN dept ON EMP.DEPTNO=DEPT.DEPTNO
WHERE emp.ename LIKE '%R%';
```

Results | Messages

	Azonosító	név	Részleg neve	telephely
1	7521	WARD	SALES	CHICAGO
2	7654	MARTIN	SALES	CHICAGO
3	7782	CLARK	ACCOUNTING	NEW YORK
4	7844	TURNER	SALES	CHICAGO
5	7902	FORD	RESEARCH	DALLAS
6	7934	MILLER	ACCOUNTING	NEW YORK

List the salgrade categories which belongs to the employees!

```
SELECT   emp.ename As név, emp.sal As fizetes,
    salgrade.grade As "fizetési besorolás"
FROM emp, salgrade
WHERE emp.sal BETWEEN salgrade.losal AND salgrade.hisal;
```

Results | Messages

	név	fizetes	fizetési besorolás
1	SMITH	800.00	1
2	ADAMS	1100.00	1
3	JAMES	950.00	1
4	WARD	1250.00	2
5	MARTIN	1250.00	2
6	MILLER	1300.00	2
7	ALLEN	1600.00	3
8	TURNER	1500.00	3
9	JONES	2975.00	4
10	BLAKE	2850.00	4
11	CLARK	2450.00	4
12	SCOTT	3000.00	4
13	FORD	3000.00	4
14	KING	5000.00	5

With JOIN operation

```
SELECT   emp.ename As név, emp.sal As fizetes,
    salgrade.grade As "fizetési besorolás"
FROM emp CROSS JOIN salgrade
WHERE emp.sal BETWEEN salgrade.losal AND salgrade.hisal;
```

Join more than one table

Display those customers who ordered chocolate. (Northwind)!

```
SELECT c.CompanyName
From Customers c
INNER JOIN Orders o ON c.CustomerID=o.CustomerID
INNER JOIN [Order Details] od ON o.OrderID=od.OrderID
INNER JOIN Products p ON od.ProductID=p.ProductID
WHERE p.ProductName='Chocolade';
```

Results | Messages

	CompanyName
1	Victuailles en stock
2	Queen Cozinha
3	Furia Bacalhau e Frutos do Mar
4	Antonio Moreno Taquería
5	Around the Horn
6	Ernst Handel

17

Display those customers who ordered not only chocolate but vegie-spread also!

```
SELECT DISTINCT c.CompanyName
FROM Customers c
INNER JOIN
    (SELECT CustomerID
    FROM Orders o
    INNER JOIN [Order Details] od ON o.OrderID=od.OrderID
    INNER JOIN Products p ON od.ProductID=p.ProductID
    WHERE p.ProductName='Chocolade') as csoki
    ON c.CustomerID=csoki.CustomerID
  INNER JOIN
    (SELECT CustomerID
    FROM Orders o
    INNER JOIN [Order Details] od ON o.OrderID=od.OrderID
    INNER JOIN Products p ON od.ProductID=p.ProductID
    WHERE p.ProductName='Vegie-spread') as krém
    ON c.CustomerID=krém.CustomerID;
```

Results | Messages

	CompanyName
1	Ernst Handel

Understanding OUTER JOINS

Returns all rows from one table and any matching rows from second table

One table's rows are "preserved"

- Designated with LEFT, RIGHT, FULL keyword
- All rows from preserved table output to result set

Matches from other table retrieved

Additional rows added to results for non-matched rows

- NULLs added in place where attributes do not match

Example: Return all customers and for those who have placed orders, return order information. Customers without matching orders will display NULL for order details.

OUTER JOIN examples

Customers that did not place orders:

```
SELECT CUST.CustomerID, CUST.StoreID,
ORD.SalesOrderID, ORD.OrderDate
FROM Sales.Customer AS CUST
LEFT OUTER JOIN Sales.SalesOrderHeader AS
ORD
ON CUST.CustomerID = ORD.CustomerID
WHERE ORD.SalesOrderID IS NULL;
```

Understanding CROSS JOINS

Combine each row from first table with each row from second table

All possible combinations are displayed

Logical foundation for inner and outer joins

- INNER JOIN starts with Cartesian product, adds filter
- OUTER JOIN takes Cartesian output, filtered, adds back non-matching rows (with NULL placeholders)

Due to Cartesian product output, not typically a desired form of JOIN

Some useful exceptions:

- Generating a table of numbers for testing

CROSS JOIN Example

Create test data by returning all combinations of two inputs:

```
SELECT EMP1.BusinessEntityID, EMP2.JobTitle
FROM HumanResources.Employee AS EMP1
CROSS JOIN HumanResources.Employee AS EMP2;
```

Understanding Self-Joins

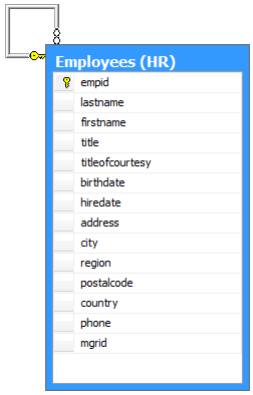
Why use self-joins?

- Compare rows in same table to each other

Create two instances of same table in FROM clause

- At least one alias required

Example: Return all employees and the name of the employee's manager



SELF JOIN operation

List the employee's name and their boss name!

```
SELECT   dolgozo.ename As dolgozó, fonok.ename AS Főnök
FROM emp dolgozo INNER JOIN emp fonok ON dolgozo.MGR=fonok.EMPNO;
```

Results | Messages

	dolgozó	Főnök
1	SMITH	FORD
2	ALLEN	BLAKE
3	WARD	BLAKE
4	JONES	KING
5	MARTIN	BLAKE
6	BLAKE	KING
7	CLARK	KING
8	SCOTT	JONES
9	TURNER	BLAKE
10	ADAMS	SCOTT
11	JAMES	BLAKE
12	FORD	JONES
13	MILLER	CLARK

Self-Join examples

Return all employees with ID of employee's manager when a manager exists (INNER JOIN):

```
SELECT  EMP.EmpID, EMP.LastName,
        EMP.JobTitle, EMP.MgrID, MGR.LastName
FROM    HR.Employees AS EMP
LEFT OUTER JOIN HR.Employees AS MGR
ON EMP.MgrID = MGR.EmpID ;
```

Return all employees with ID of manager (OUTER JOIN). This will return NULL for the CEO:

```
SELECT  EMP.EmpID, EMP.LastName,
        EMP.Title, MGR.MgrID
FROM HumanResources.Employee AS EMP
LEFT OUTER JOIN HumanResources.Employee AS MGR
ON EMP.MgrID = MGR.EmpID;
```

Display the number of employees per departments!

```
SELECT   d.dname As részlegnév, count(e.ename) AS létszám
FROM emp e INNER JOIN dept d ON e.deptno=d.deptno
GROUP BY d.dname;
```

Results | Messages

	részlegnév	létszám
1	ACCOUNTING	3
2	RESEARCH	5
3	SALES	6

Where is the Operations and Kereskedes department?

```
SELECT   d.dname As részlegnév, count(e.ename) AS létszám
FROM dept d LEFT OUTER JOIN emp e
ON e.deptno=d.deptno
GROUP BY d.dname;
```

Results | Messages

	részlegnév	létszám
1	ACCOUNTING	3
2	Kereskedes	0
3	OPERATIONS	0
4	RESEARCH	5
5	SALES	6

Display the name of employees and the departement name per employees!

```
SELECT   e.ename As név, d.deptno As kód,
d.dname As részlegnév
FROM emp e Right OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Results | Messages

	név	k...	részlegnév
1	CLARK	10	ACCOUNTING
2	KING	10	ACCOUNTING
3	MILLER	10	ACCOUNTING
4	SMITH	20	RESEARCH
5	JONES	20	RESEARCH
6	SCOTT	20	RESEARCH
7	ADAMS	20	RESEARCH
8	FORD	20	RESEARCH
9	ALLEN	30	SALES
10	WARD	30	SALES
11	MARTIN	30	SALES
12	BLAKE	30	SALES
13	TURNER	30	SALES
14	JAMES	30	SALES
15	NULL	40	OPERATIONS
16	NULL	50	Kereskedes

FULL OUTER JOIN

```
SELECT e.ENAME, e.DEPTNO, loc
FROM EMP e FULL JOIN DEPT ON e.DEPTNO=DEPT.DEPTNO
ORDER BY e.DEPTNO;
```

Results | Messages

	ENAME	DEPTNO	loc
1	JOYCE	NULL	NULL
2	NULL	NULL	BOSTON
3	MILLER	10	NEW YORK
4	CLARK	10	NEW YORK
5	KING	10	NEW YORK
6	SCOTT	20	DALLAS
7	SMITH	20	DALLAS
8	JONES	20	DALLAS
9	FORD	20	DALLAS
10	ADAMS	20	DALLAS
11	JAMES	30	CHICAGO
12	MARTIN	30	CHICAGO
13	BLAKE	30	CHICAGO
14	ALLEN	30	CHICAGO
15	WARD	30	CHICAGO
16	TURNER	30	CHICAGO

28

Set Operators

Interactions between sets

The results of two input queries may be combined, compared, or operated against each other

Both sets must have the same number of compatible columns

ORDER BY not allowed in input queries, but may be used for result of set operation

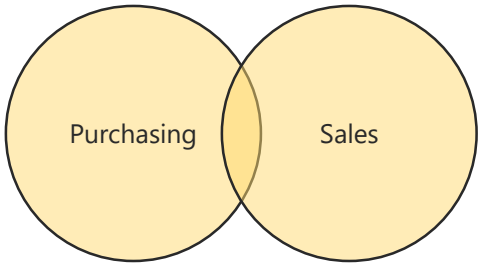
NULLs considered equal when comparing sets

SET operators include UNION, INTERSECT, EXCEPT, and APPLY

```
<SELECT query_1>
<set_operator>
<SELECT query_2>
[ORDER BY <sort_list>]
```

Using the UNION operator

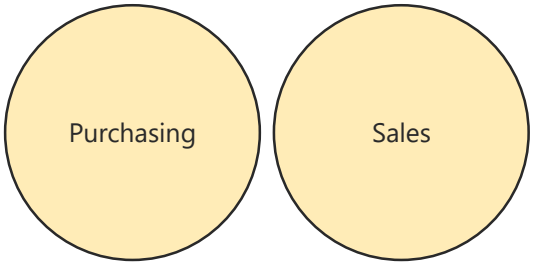
UNION returns a result set of distinct rows combined from both sides
Duplicates removed during query processing (affects performance)




```
-- only distinct rows from both queries are returned
SELECT ProductID, OrderQty, UnitPrice FROM Sales.SalesOrderDetail
UNION
SELECT ProductID, OrderQty, UnitPrice FROM Purchasing.PurchaseOrderDetail
```

Using the UNION ALL operator

UNION ALL returns a result set with all rows from both sets
To avoid performance penalty, use UNION ALL even if you know there are no duplicates




```
-- all rows from both queries are returned
SELECT ProductID, OrderQty, UnitPrice FROM Sales.SalesOrderDetail
UNION ALL
SELECT ProductID, OrderQty, UnitPrice FROM Purchasing.PurchaseOrderDetail
```

UNION ALL and UNION

```sql
USE Northwind
GO
SELECT Country, City FROM Customers
UNION ALL
SELECT Country, City FROM Suppliers;
```

```sql
USE Northwind
GO
SELECT Country, City FROM Customers
UNION
SELECT Country, City FROM Suppliers;
```

Results | Messages

	Country	City
1	Germany	Berlin
2	Mexico	Mexico V.
3	Mexico	Mexico V.
4	UK	London
5	Sweden	Luleå
6	Germany	Mannheim
7	France	Strasbourg
8	Spain	Madrid
9	France	Marseille
10	Canada	Tsawassen
11	UK	London
12	Argentina	Buenos Aires
13	Mexico	Mexico V.
14	Switzerland	Bern
15	Brazil	Sao Paulo
16	UK	London
17	Germany	Aachen

120 rows

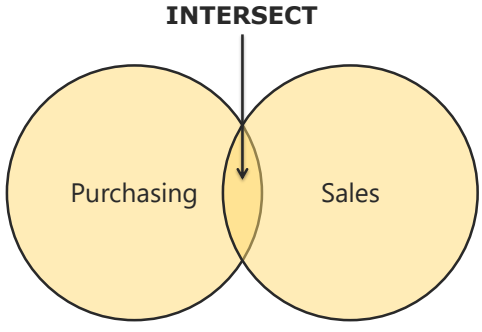
Results | Messages

	Country	City
1	Argentina	Buenos Aires
2	Australia	Melbourne
3	Australia	Sydney
4	Austria	Graz
5	Austria	Salzburg
6	Belgium	Bruxelles
7	Belgium	Charleroi
8	Brazil	Campinas
9	Brazil	Resende
10	Brazil	Rio de Janeiro
11	Brazil	Sao Paulo
12	Canada	Montréal
13	Canada	Ste-Hyacinthe
14	Canada	Tsawassen
15	Canada	Vancouver
16	Denmark	Århus
17	Denmark	Kobenhavn

93 rows

Using the INTERSECT operator

INTERSECT returns only distinct rows that appear in both result sets




```sql
-- only rows that exist in both queries are returned
SELECT ProductID, OrderQty, UnitPrice FROM Sales.SalesOrderDetail
INTERSECT
SELECT ProductID, OrderQty, UnitPrice FROM Purchasing.PurchaseOrderDetail
```

The INTERSECT operator

```
USE Northwind
GO
SELECT Country, Region, City FROM Customers
INTERSECT
SELECT Country, Region, City FROM Suppliers;
```

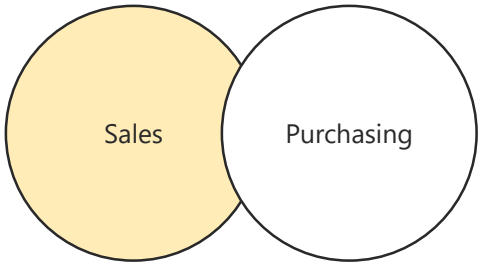
Results | Messages

	Country	Region	City
1	Canada	Québec	Montréal
2	France	NULL	Paris
3	Germany	NULL	Berlin
4	UK	NULL	London

Using the EXCEPT operator

EXCEPT returns only distinct rows that appear in the left set but not the right

- Order in which sets are specified matters




```
-- only rows from Sales are returned
SELECT ProductID, OrderQty, UnitPrice FROM Sales.SalesOrderDetail
EXCEPT
SELECT ProductID, OrderQty, UnitPrice FROM Purchasing.PurchaseOrderDetail
```

Az EXCEPT operator

```
USE Northwind
GO
SELECT Country, Region, City FROM Customers
EXCEPT
SELECT Country, Region, City FROM Suppliers;
```

65 rows

Results | Messages

	Country	Region	City
1	Argentina	NULL	Buenos Aires
2	Austria	NULL	Graz
3	Austria	NULL	Salzburg
4	Belgium	NULL	Bruxelles
5	Belgium	NULL	Charleroi
6	Brazil	RJ	Rio de Janeiro
7	Brazil	SP	Campinas
8	Brazil	SP	Resende
9	Brazil	SP	Sao Paulo
10	Canada	BC	Tsawassen
11	Canada	BC	Vancouver
12	Denmark	NULL	Århus
13	Denmark	NULL	Kobenhavn
14	Finland	NULL	Helsinki
15	Finland	NULL	Oulu
16	France	NULL	Lille
17	France	NULL	Lyon

Az EXCEPT operator

```
USE Northwind
GO
SELECT Country, Region, City FROM Suppliers
EXCEPT
SELECT Country, Region, City FROM Customers;
```

25 rows

Results | Messages

	Country	Region	City
1	Australia	NSW	Sydney
2	Australia	Victoria	Melbourne
3	Brazil	NULL	Sao Paulo
4	Canada	Québec	Ste-Hyacinthe
5	Denmark	NULL	Lyngby
6	Finland	NULL	Lappeenranta
7	France	NULL	Annecy
8	France	NULL	Montceau
9	Germany	NULL	Cuxhaven
10	Germany	NULL	Frankfurt
11	Italy	NULL	Ravenna
12	Italy	NULL	Salerno
13	Japan	NULL	Osaka
14	Japan	NULL	Tokyo
15	Netherlands	NULL	Zaandam
16	Norway	NULL	Sandvika
17	Singapore	NULL	Singapore

Precedence rule

INTERSECT
UNION, EXCEPT

```
USE Northwind
GO
SELECT Country, Region, City FROM Suppliers
EXCEPT
SELECT Country, Region, City FROM Employees
INTERSECT
SELECT Country, Region, City FROM Customers;
```

28 rows

	Country	Region	City
1	Australia	NSW	Sydney
2	Australia	Victoria	Melbourne
3	Brazil	NULL	Sao Paulo
4	Canada	Québec	Montréal
5	Canada	Québec	Ste-Hyacinthe
6	Denmark	NULL	Lyngby
7	Finland	NULL	Lappeenranta
8	France	NULL	Annecy
9	France	NULL	Montceau
10	France	NULL	Paris
11	Germany	NULL	Berlin
12	Germany	NULL	Cuxhaven
13	Germany	NULL	Frankfurt
14	Italy	NULL	Ravenna
15	Italy	NULL	Salerno
16	Japan	NULL	Osaka
17	Japan	NULL	Tokyo

Modify the precedence rule applying parenthesis

```
USE Northwind
GO
(SELECT Country, Region, City FROM Suppliers
EXCEPT
SELECT Country, Region, City FROM Employees)
INTERSECT
SELECT Country, Region, City FROM Customers;
```

100 %

Results | Messages

	Country	Region	City
1	Canada	Québec	Montréal
2	France	NULL	Paris
3	Germany	NULL	Berlin

VIEW

What Is a View?

Stored query. It does not contain any data. You can use it in a SQL statement as a table.

Employee (table)				
EmployeeID	LastName	FirstName	Title	…
287	Mensa-Annan	Tete	Mr.	…
288	Abbas	Syed	Mr.	…
289	Valdez	Rachel	NULL	…

vEmployee (view)	
LastName	FirstName
Mensa-Annan	Tete
Abbas	Syed
Valdez	Rachel

Advantages of Views

- Focus the data for a user
- Mask database complexity
- Simplify management of user permissions
- Improve performance
- Organize data for export to other applications

Syntax for Creating Views

Use CREATE VIEW Transact-SQL statement:

```
CREATE VIEW [ schema_name.] view_name [ (column [ ,...n ] ) ]
   [WITH [ENCRYPTION] [SCHEMABINDING] [VIEW_METADATA] ]
   AS select_statement [ ; ]
   [ WITH CHECK OPTION ]
```

- Cannot contain more than 1,024 columns
- Cannot use COMPUTE, COMPUTE BY, or INTO
- Cannot use ORDER BY without TOP

Syntax for Altering and Dropping Views

Alter by using the ALTER VIEW Transact-SQL statement:

```
ALTER VIEW [ schema_name.]view_name [ (column [ ,...n ] ) ]
   [WITH [ENCRYPTION] [SCHEMABINDING] [VIEW_METADATA] ]
   AS select_statement [ ; ]
   [ WITH CHECK OPTION ]
```

```
DROP VIEW [ schema_name.]view_name [ ...,n ] [ ; ]
```

DML Statements

Using INSERT to add data

The INSERT...VALUES statement inserts a single row by default

```
INSERT INTO Production.UnitMeasure (Name,
UnitMeasureCode, ModifiedDate)
VALUES (N'Square Yards', N'Y2', GETDATE());
GO
```

Table and row constructors add multi-row capability to INSERT...VALUES

```
INSERT INTO Production.UnitMeasure (Name,
UnitMeasureCode, ModifiedDate)
VALUES
        (N'Square Feet', N'F2', GETDATE()),
        (N'Square Inches', N'I2', GETDATE());
```

Using INSERT with SELECT

INSERT...SELECT is used to insert the result set of a query into an existing table

```
INSERT INTO Production.UnitMeasure (Name,
UnitMeasureCode, ModifiedDate)
SELECT Name, UnitMeasureCode, ModifiedDate
FROM Sales.TempUnitTable
WHERE ModifiedDate < '20080101';
```

Using SELECT INTO

SELECT...INTO is similar to INSERT...SELECT but SELECT...INTO creates a new table each time the statement is executed

Copies column names, data types, and nullability

Does not copy constraints or indexes

```
SELECT Name, UnitMeasureCode, ModifiedDate
INTO Production.TempUOMTable
FROM Production.UnitMeasure
WHERE orderdate < '20080101';
```

Using UPDATE to modify data

Updates all rows in a table or view

- Set can be filtered with a WHERE clause
- Set can be defined with a JOIN clause

Only columns specified in the SET clause are modified

Updates the ModifiedDate using a the GETDATE function for the record that has 'M2' in the UnitMeasureCode

```
UPDATE Production.UnitMeasure
    SET ModifiedDate = (GETDATE())
    WHERE UnitMeasureCode = 'M2'
```

If no WHERE clause is specified, all records in the Production.UnitMeasure will be updated

Using DELETE to remove data

DELETE operates on a set
- Set may be filtered with a WHERE clause

Deletion of each row is logged in database's transaction log

DELETE may be rolled back if statement issued within a user-defined transaction or if an error is encountered

```
DELETE FROM Production.UnitMeasure
WHERE UnitMeasureCode = 'Y2';
```

If no WHERE clause is specified, all records in the Production.UnitMeasure will be deleted

Using TRUNCATE TABLE to remove all data

TRUNCATE TABLE clears the entire table
- Storage is physically deallocated, rows not individually removed
- Minimally logged
- Can be rolled back if TRUNCATE issued within a transaction
- TRUNCATE TABLE will fail if the table is referenced by a foreign key constraint in another table

```
TRUNCATE TABLE Production.UnitMeasure
```

Thank you, for your attention!