Database Systems

Etelka Szendrői (PhD)
Associate professor
szendroi@mik.pte.hu

SQL- statements

Definition:		
DDL:	- create an object	CREATE
	- drop an object	DROP
	- modify object	ALTER
Data Manipulation:		
DML:	- insert new records	INSERT
	- delete records	DELETE
	- update records	UPDATE
Retrieve data:		
DQL:	- Query	SELECT
Control		
DCL:	- security	GRANT,..
	- tranzaction management	COMMIT,..

Data Manipulation Language (DML)

- ***Data Manipulation Language*** (DML) is the language element which allows you to use the core statements:
  - SELECT: Retrieves rows from the database and enables the selection of one or many rows or columns from one or many tables in SQL Server.
  - INSERT: Adds one or more new rows to a table or a view in SQL Server.
  - UPDATE: Changes existing data in one or more columns in a table or view.
  - DELETE: Removes rows from a table or view.
  - MERGE: Performs insert, update, or delete operations on a target table based on the results of a join with a source table.

Data Definition Language (DDL)

- ***Data Definition Language (DDL)*** is a subset of the Transact-SQL language.
- It deals with creating database objects like tables, constraints, and stored procedures.
- Some DDL commands include:
  - USE: Changes the database context.
  - CREATE: Creates a SQL Server database object (table, view or stored procedure)
  - ALTER: Changes an existing object
  - DROP: Removes an object from the database

System Tables

- System views belong to the sys schema. Some of these system tables include:
  - sys.Tables
  - sys.Columns
  - sys.Databases
  - sys.Constraints
  - sys.Views
  - sys.Procedures
  - sys.Indexes
  - sys.Triggers
  - sys.Objects

Elements of the SELECT statement

Clause	Expression
SELECT	**<select list>**
FROM	**<table source>**
WHERE	<search condition>
GROUP BY	<group by list>
ORDER BY	<order by list>

Logical query processing

The order in which a query is written is not the order in which it is evaluated by SQL Server.

```
5: SELECT       <select list>
1: FROM      <table source>
2: WHERE     <search condition>
3: GROUP BY <group by list>
4: HAVING <search condition>
6: ORDER BY <order by list>
```

Retrieving columns from a table or view

Use SELECT with column list to display columns

Use FROM to specify a source table or view

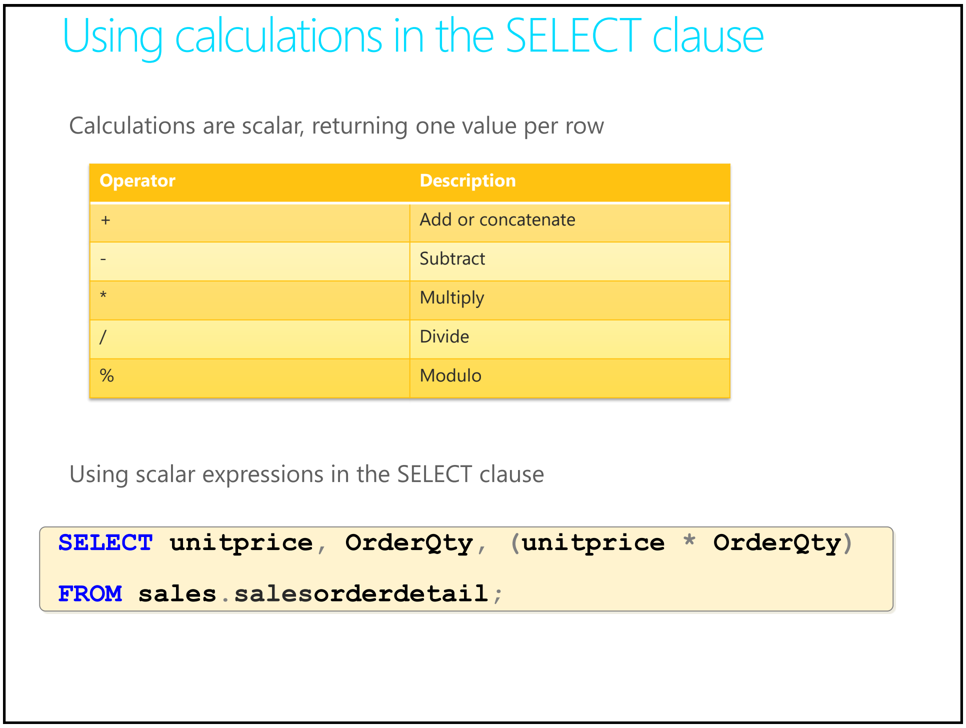
- Specify both schema and table names

Delimit names if necessary

End all statements with a semicolon

Keyword	Expression
SELECT	**<select list>**
FROM	**<table source>**

```
SELECT CustomerID, StoreID
FROM Sales.Customer;
```



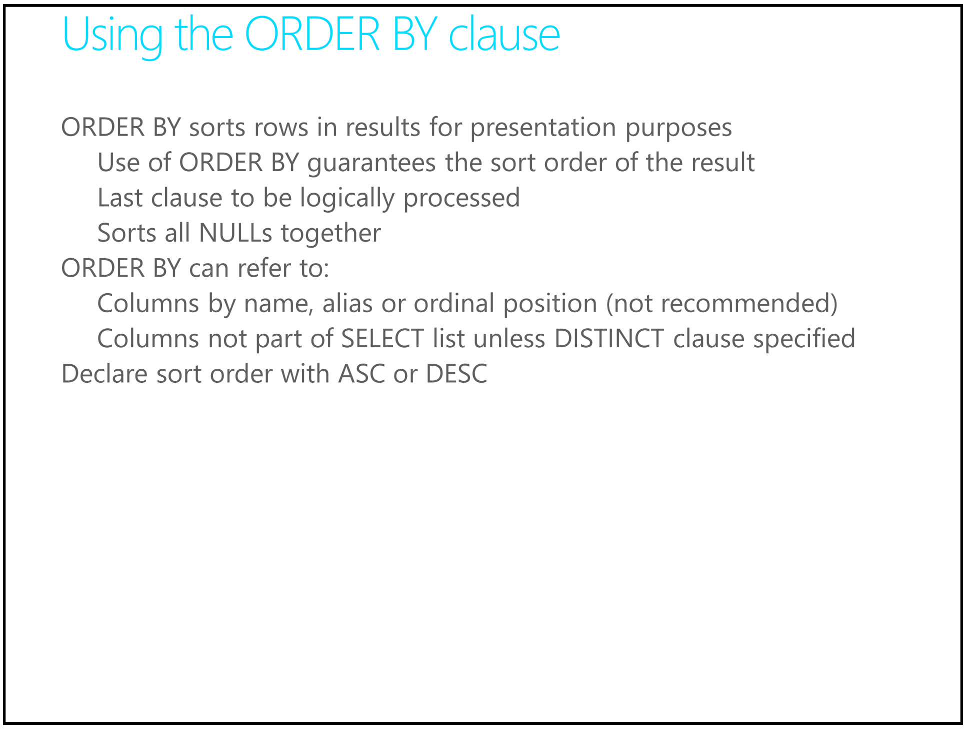
Using calculations in the SELECT clause

Calculations are scalar, returning one value per row

Operator	Description
+	Add or concatenate
-	Subtract
*	Multiply
/	Divide
%	Modulo

Using scalar expressions in the SELECT clause

```
SELECT unitprice, OrderQty, (unitprice * OrderQty)
FROM sales.salesorderdetail;
```



Using the ORDER BY clause

- ORDER BY sorts rows in results for presentation purposes
  - Use of ORDER BY guarantees the sort order of the result
  - Last clause to be logically processed
  - Sorts all NULLs together
- ORDER BY can refer to:
  - Columns by name, alias or ordinal position (not recommended)
  - Columns not part of SELECT list unless DISTINCT clause specified
- Declare sort order with ASC or DESC

ORDER BY clause examples

ORDER BY with column names:

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
ORDER BY OrderDate;
```

ORDER BY with column alias:

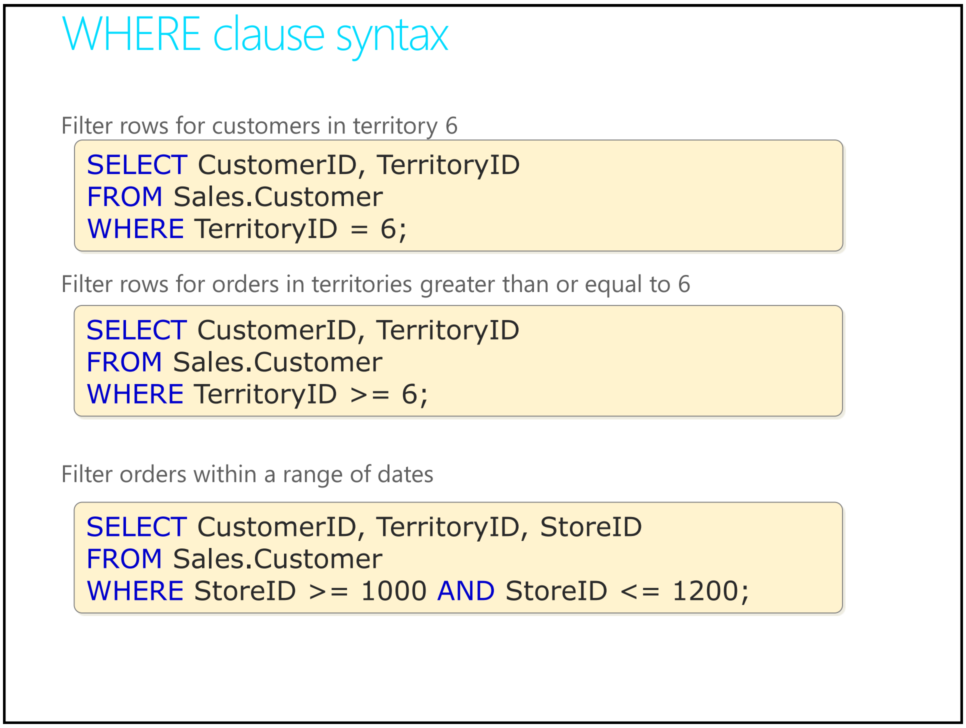
```
SELECT SalesOrderID, CustomerID,
YEAR(OrderDate) AS OrderYear
FROM Sales.SalesOrderHeader
ORDER BY OrderYear;
```

ORDER BY with descending order:

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
ORDER BY OrderDate DESC;
```

Filtering data in the WHERE clause

- WHERE clauses use predicates
  - Must be expressed as logical conditions
  - Only rows for which predicate evaluates to TRUE are accepted
  - Values of FALSE or UNKNOWN are filtered out
- WHERE clause follows FROM, precedes other clauses
  - Can't see aliases declared in SELECT clause
- Can be optimized by SQL Server to use indexes



WHERE clause syntax

Filter rows for customers in territory 6

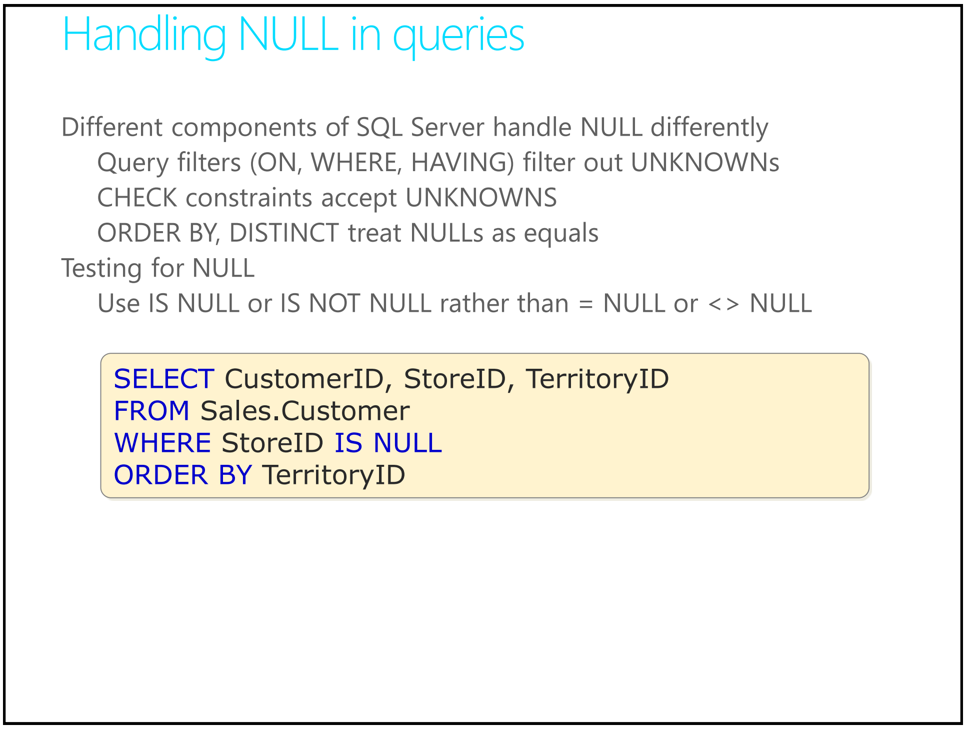
```
SELECT CustomerID, TerritoryID
FROM Sales.Customer
WHERE TerritoryID = 6;
```

Filter rows for orders in territories greater than or equal to 6

```
SELECT CustomerID, TerritoryID
FROM Sales.Customer
WHERE TerritoryID >= 6;
```

Filter orders within a range of dates

```
SELECT CustomerID, TerritoryID, StoreID
FROM Sales.Customer
WHERE StoreID >= 1000 AND StoreID <= 1200;
```



Handling NULL in queries

Different components of SQL Server handle NULL differently

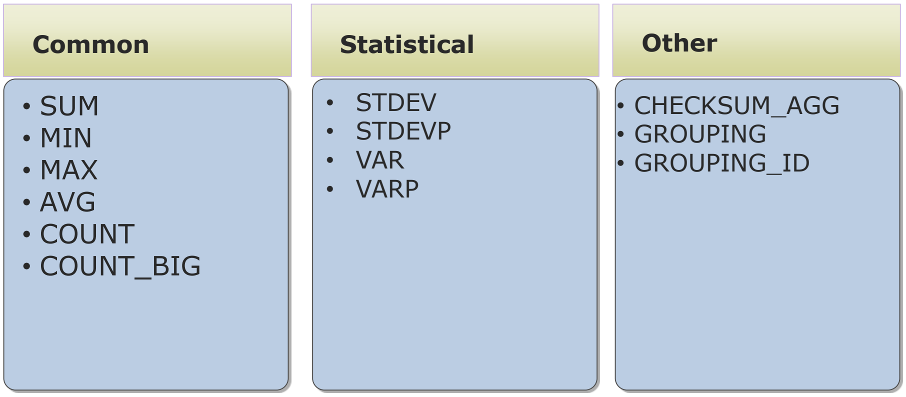
- Query filters (ON, WHERE, HAVING) filter out UNKNOWNs
- CHECK constraints accept UNKNOWNS
- ORDER BY, DISTINCT treat NULLs as equals

Testing for NULL

- Use IS NULL or IS NOT NULL rather than = NULL or <> NULL

```
SELECT CustomerID, StoreID, TerritoryID
FROM Sales.Customer
WHERE StoreID IS NULL
ORDER BY TerritoryID
```

Common built-in aggregate functions



Common

- SUM
- MIN
- MAX
- AVG
- COUNT
- COUNT_BIG

Statistical

- STDEV
- STDEVP
- VAR
- VARP

Other

- CHECKSUM_AGG
- GROUPING
- GROUPING_ID

Working with aggregate functions

Aggregate functions:

- Return a scalar value (with no column name)
- Ignore NULLs except in COUNT(*)
- Can be used in
  - SELECT, HAVING, and ORDER BY clauses
- Frequently used with GROUP BY clause

```
SELECT COUNT (DISTINCT SalesOrderID) AS
UniqueOrders,
AVG(UnitPrice) AS Avg_UnitPrice,
MIN(OrderQty)AS Min_OrderQty,
MAX(LineTotal) AS Max_LineTotal
FROM Sales.SalesOrderDetail;
```

```
UniqueOrders Avg_UnitPrice Min_OrderQty Max_LineTotal
------------ ------------ ------------ -------------
31465          465.0934           1        27893.619000
```

Using DISTINCT with aggregate functions

Use DISTINCT with aggregate functions to summarize only unique values

DISTINCT aggregates eliminate duplicate values, not rows (unlike SELECT DISTINCT)

Compare (with partial results):

```
SELECT SalesPersonID, YEAR(OrderDate) AS OrderYear,
COUNT(CustomerID) AS All_Custs,
COUNT(DISTINCT CustomerID) AS Unique_Custs
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID, YEAR(OrderDate);
```

```
SalesPersonID  OrderYear  All_Custs  Unique_custs
-----------    ----------- ----------- ------------
289             2006        84          48
281             2008        52          27
285             2007         9           8
277             2006       140          57
```

Using the GROUP BY clause

GROUP BY creates groups for output rows, according to unique combination of values specified in the GROUP BY clause

```
SELECT <select_list>
FROM <table_source>
WHERE <search_condition>
GROUP BY <group_by_list>;
```

GROUP BY calculates a summary value for aggregate functions in subsequent phases

```
SELECT SalesPersonID, COUNT(*) AS Cnt
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID;
```

Detail rows are “lost” after GROUP BY clause is processed

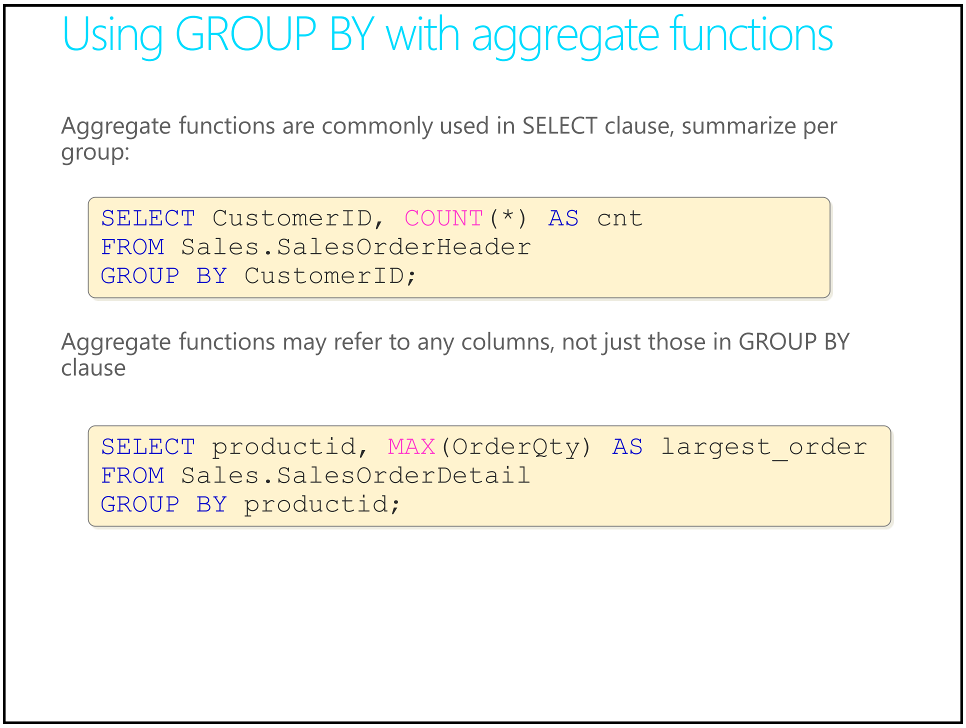
GROUP BY and HAVING

GROUP BY and logical order of operations

HAVING, SELECT, and ORDER BY must return a single value per group

All columns in SELECT, HAVING, and ORDER BY must appear in GROUP BY clause or be inputs to aggregate expressions

Logical Order	Phase	Comments
5	SELECT	
1	FROM	
2	WHERE	
3	GROUP BY	Creates groups
4	HAVING	Operates on groups
6	ORDER BY	



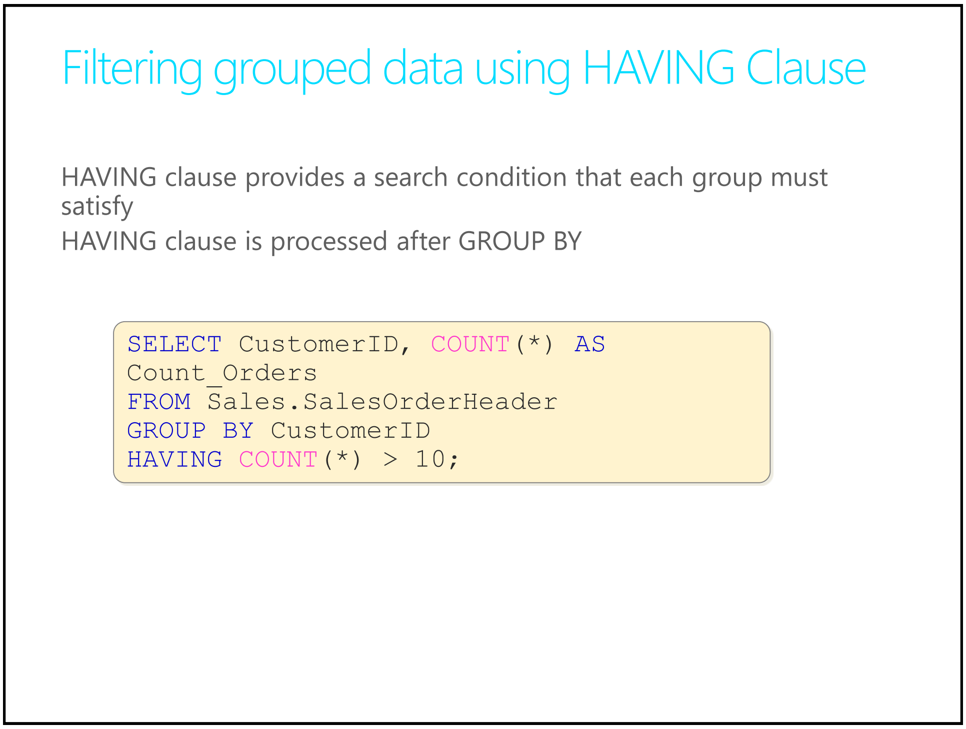
Using GROUP BY with aggregate functions

Aggregate functions are commonly used in SELECT clause, summarize per group:

```
SELECT CustomerID, COUNT(*) AS cnt
FROM Sales.SalesOrderHeader
GROUP BY CustomerID;
```

Aggregate functions may refer to any columns, not just those in GROUP BY clause

```
SELECT productid, MAX(OrderQty) AS largest_order
FROM Sales.SalesOrderDetail
GROUP BY productid;
```



Filtering grouped data using HAVING Clause

HAVING clause provides a search condition that each group must satisfy

HAVING clause is processed after GROUP BY

```
SELECT CustomerID, COUNT(*) AS
Count_Orders
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING COUNT(*) > 10;
```

Compare HAVING to WHERE clauses

- Using a COUNT(*) expression in HAVING clause is useful to solve common business problems:
- Show only customers that have placed more than one order:

```
SELECT Cust.Customerid, COUNT(*) AS cnt
FROM Sales.Customer AS Cust
JOIN Sales.SalesOrderHeader AS Ord ON Cust.CustomerID =
ORD.CustomerID
GROUP BY Cust.CustomerID
HAVING COUNT(*) > 1;
```

- Show only products that appear on 10 or more orders:

```
SELECT Prod.ProductID, COUNT(*) AS cnt
FROM Production.Product AS Prod
JOIN Sales.SalesOrderDetail AS Ord ON Prod.ProductID =
Ord.ProductID
GROUP BY Prod.ProductID
HAVING COUNT(*) >= 10;
```

Subqueries

Working with subqueries

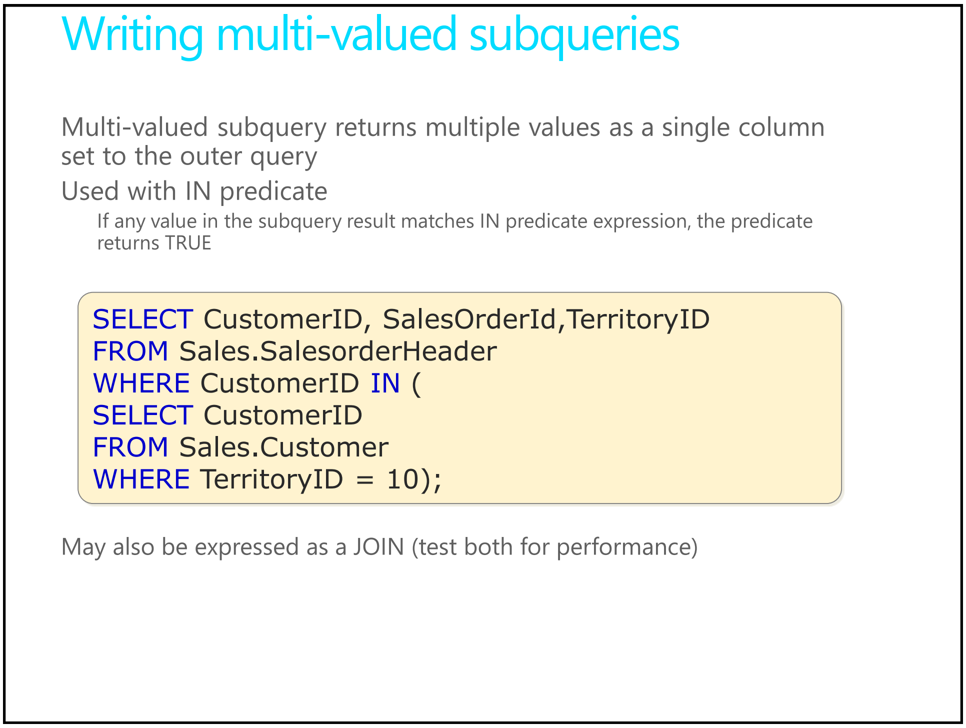
- Subqueries are nested queries or queries within queries
- Results from inner query are passed to outer query
  - Inner query acts like an expression from perspective of outer query
- Subqueries can be self-contained or correlated
  - Self-contained subqueries have no dependency on outer query
  - Correlated 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.

```sql
SELECT SalesOrderID, ProductID, UnitPrice, OrderQty
FROM Sales.SalesOrderDetail
WHERE SalesOrderID =
(SELECT MAX(SalesOrderID) AS LastOrder
FROM Sales.SalesOrderHeader);
```

- 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

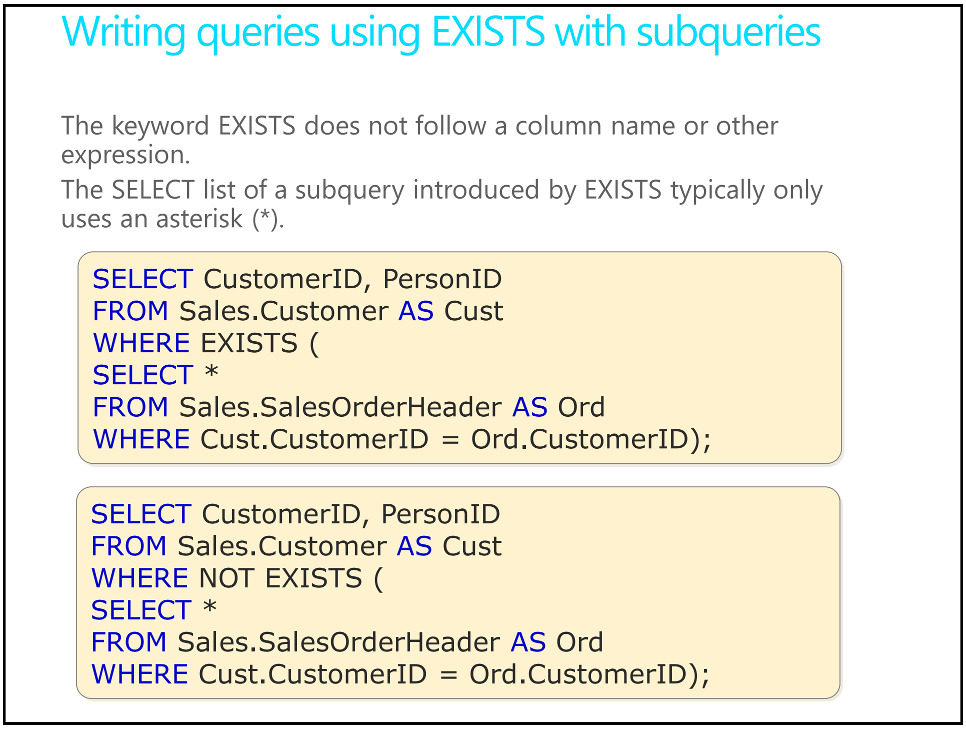
Multi-valued subquery returns multiple values as a single column set to the outer query

Used with IN predicate

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

```
SELECT CustomerID, SalesOrderId,TerritoryID
FROM Sales.SalesorderHeader
WHERE CustomerID IN (
SELECT CustomerID
FROM Sales.Customer
WHERE TerritoryID = 10);
```

May also be expressed as a JOIN (test both for performance)



Writing queries using EXISTS with subqueries

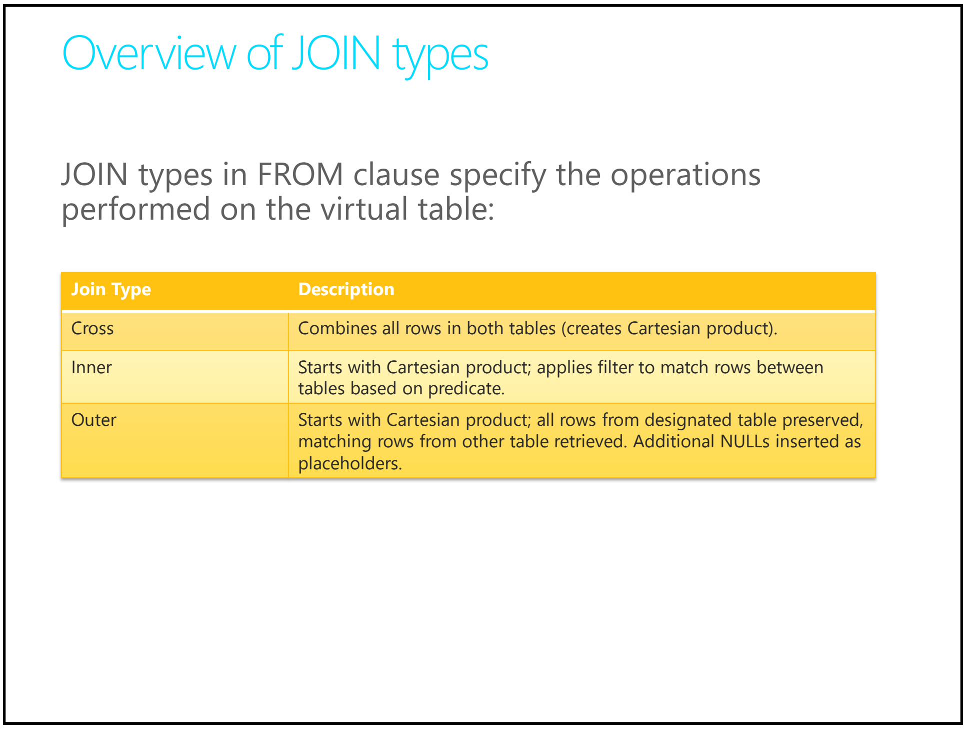
The keyword EXISTS does not follow a column name or other expression.

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

```
SELECT CustomerID, PersonID
FROM Sales.Customer AS Cust
WHERE EXISTS (
SELECT *
FROM Sales.SalesOrderHeader AS Ord
WHERE Cust.CustomerID = Ord.CustomerID);
```

```
SELECT CustomerID, PersonID
FROM Sales.Customer AS Cust
WHERE NOT EXISTS (
SELECT *
FROM Sales.SalesOrderHeader AS Ord
WHERE Cust.CustomerID = Ord.CustomerID);
```

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

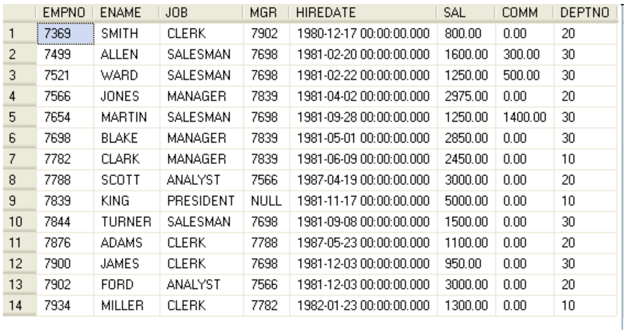
List tables in FROM Clause separated by JOIN operator

Table order does not matter, and aliases are preferred

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

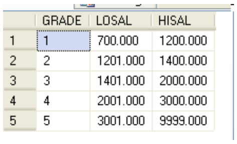
```
SELECT SOH.SalesOrderID,
       SOH.OrderDate,
       SOD.ProductID,
       SOD.UnitPrice,
       SOD.OrderQty
FROM Sales.SalesOrderHeader AS SOH
JOIN Sales.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID;
```

Emploees table (emp)



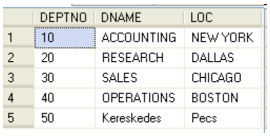
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800.00	0.00	20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600.00	300.00	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250.00	500.00	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975.00	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



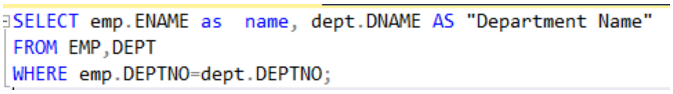
	GRADE	LOSAL	HISAL
1	1	700.000	1200.000
2	2	1201.000	1400.000
3	3	1401.000	2000.000
4	4	2001.000	3000.000
5	5	3001.000	9999.000

Department (dept)

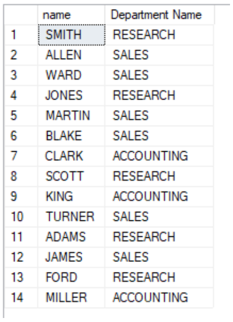


	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	50	Kereskedes	Pecs

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

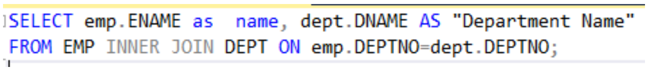


```
SELECT emp.ENAME as  name, dept.DNAME AS "Department Name"
FROM EMP,DEPT
WHERE emp.DEPTNO=dept.DEPTNO;
```



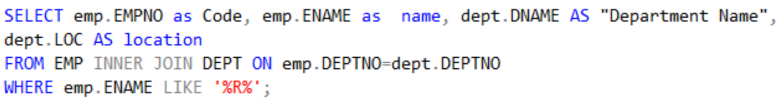
	name	Department Name
1	SMITH	RESEARCH
2	ALLEN	SALES
3	WARD	SALES
4	JONES	RESEARCH
5	MARTIN	SALES
6	BLAKE	SALES
7	CLARK	ACCOUNTING
8	SCOTT	RESEARCH
9	KING	ACCOUNTING
10	TURNER	SALES
11	ADAMS	RESEARCH
12	JAMES	SALES
13	FORD	RESEARCH
14	MILLER	ACCOUNTING

INNER JOIN operation:

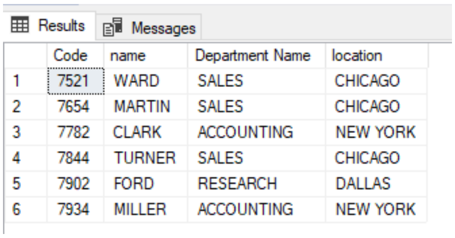


```
SELECT emp.ENAME as  name, dept.DNAME AS "Department Name"
FROM EMP INNER JOIN DEPT ON emp.DEPTNO=dept.DEPTNO;
```

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



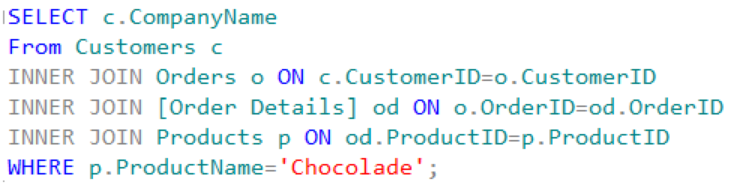
```
SELECT emp.EMPNO as Code, emp.ENAME as  name, dept.DNAME AS "Department Name",
dept.LOC AS location
FROM EMP INNER JOIN DEPT ON emp.DEPTNO=dept.DEPTNO
WHERE emp.ENAME LIKE '%R%';
```



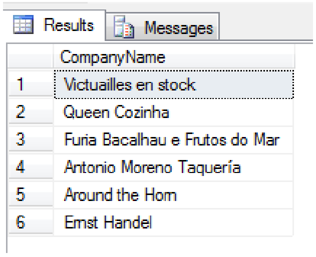
	Code	name	Department Name	location
1	7521	WARD	SALES	CHICAGO
2	7654	MARTIN	SALES	CHICAGO
3	7782	CLARK	ACCOUNTING	NEW YORK
4	7844	TURNER	SALES	CHICAGO
5	7902	FORD	RESEARCH	DALLAS
6	7934	MILLER	ACCOUNTING	NEW YORK

Join more than one table

Display those customers who ordered chocolate. (Northwind)!

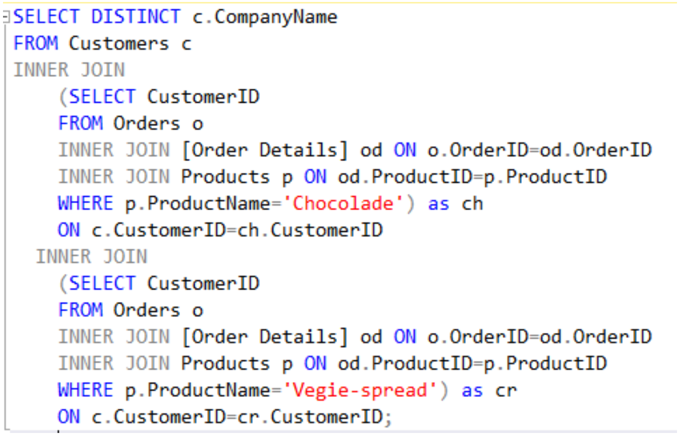


```
SELECT c.CompanyName
From Customers c
INNER JOIN Orders o ON c.CustomerID=o.CustomerID
INNER JOIN [Order Details] od ON o.OrderID=od.OrderID
INNER JOIN Products p ON od.ProductID=p.ProductID
WHERE p.ProductName='Chocolade';
```

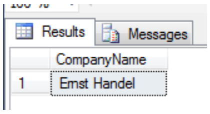


	CompanyName
1	Victuailles en stock
2	Queen Cozinha
3	Furia Bacalhau e Frutos do Mar
4	Antonio Moreno Taquería
5	Around the Horn
6	Ernst Handel

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 ch
    ON c.CustomerID=ch.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 cr
    ON c.CustomerID=cr.CustomerID;
```



```
   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

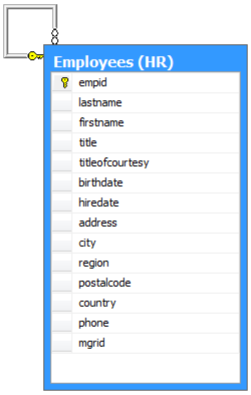
Why use self-joins?

- 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 examples

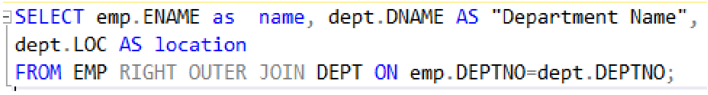
Return all employees with ID of employee’s manager when a manager exists (INNER JOIN):

```
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 ;
```

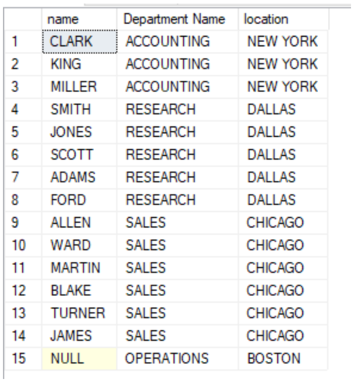
Return all employees with ID of manager (OUTER JOIN). This will return NULL for the CEO:

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

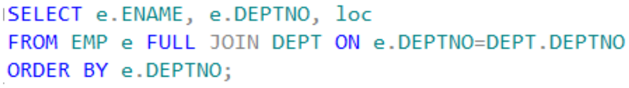


```
SELECT emp.ENAME as  name, dept.DNAME AS "Department Name",
dept.LOC AS location
FROM EMP RIGHT OUTER JOIN DEPT ON emp.DEPTNO=dept.DEPTNO;
```

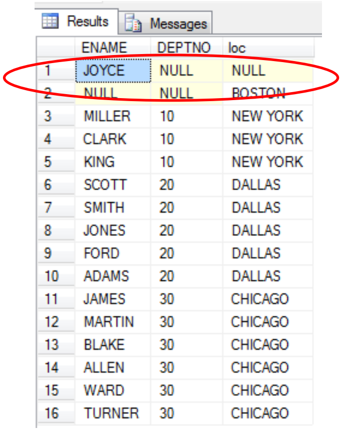


	name	Department Name	location
1	CLARK	ACCOUNTING	NEW YORK
2	KING	ACCOUNTING	NEW YORK
3	MILLER	ACCOUNTING	NEW YORK
4	SMITH	RESEARCH	DALLAS
5	JONES	RESEARCH	DALLAS
6	SCOTT	RESEARCH	DALLAS
7	ADAMS	RESEARCH	DALLAS
8	FORD	RESEARCH	DALLAS
9	ALLEN	SALES	CHICAGO
10	WARD	SALES	CHICAGO
11	MARTIN	SALES	CHICAGO
12	BLAKE	SALES	CHICAGO
13	TURNER	SALES	CHICAGO
14	JAMES	SALES	CHICAGO
15	NULL	OPERATIONS	BOSTON

FULL OUTER JOIN



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



	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