Advanced DML Statements

FSOFT ACADEMY

Lesson Objectives

01 Understand about SQL Joins in SQL Server

02 Understand subqueries in SQL Server

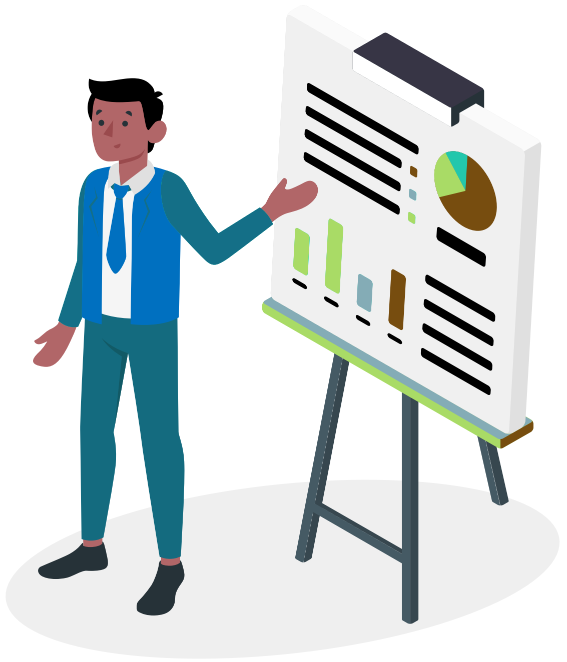
operations on data

03 Understand CTE and ranking functions

04 Apply SQL Join, Subqueries, CTE to real projects

Agenda

1. **SQL JOINS**
2. **Sub-Queries**
3. **CTE and Ranking Functions**

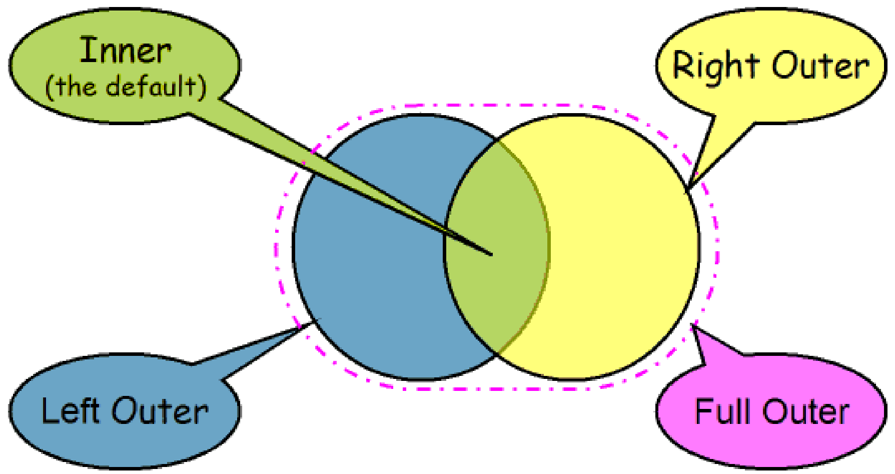


Section 1

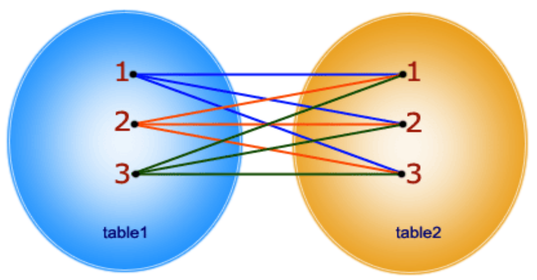
SQL JOINS

WHAT'S SQL JOINS?

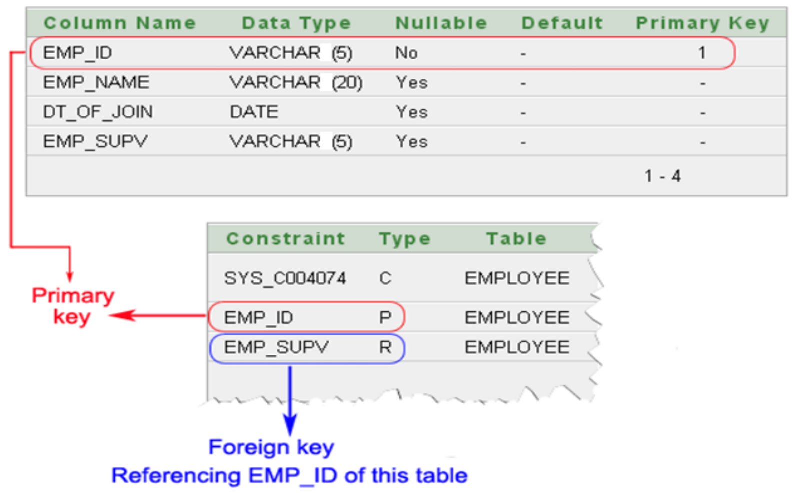
- SQL Joins are used to combine **rows from two or more tables** based on logical relationships between the tables.
- **Types of Join in SQL:**
  - ✓Inner Join
  - ✓Outer Join
  - ✓Cross Join
  - ✓Self Join



SELECT * FROM table1 CROSS JOIN table2;



In CROSS JOIN, each row from 1st table joins with all the rows of another table.
If 1st table contain x rows and y rows in 2nd one the result set will be x * y rows.



Column Name	Data Type	Nullable	Default	Primary Key
EMP_ID	VARCHAR (5)	No	-	1
EMP_NAME	VARCHAR (20)	Yes	-	-
DT_OF_JOIN	DATE	Yes	-	-
EMP_SUPV	VARCHAR (5)	Yes	-	-
				1 - 4

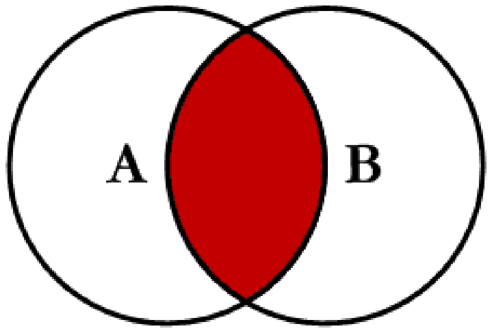
Constraint	Type	Table
SYS_C004074	C	EMPLOYEE
EMP_ID	P	EMPLOYEE
EMP_SUPV	R	EMPLOYEE

Primary
key
Foreign key
Referencing EMP_ID of this table

INNER JOIN

- The INNER JOIN selects **all rows from both tables** as long as there is a match between the columns in both tables.
- Eliminate the rows that do not match with a row from the other table
  - ✓**Syntax**

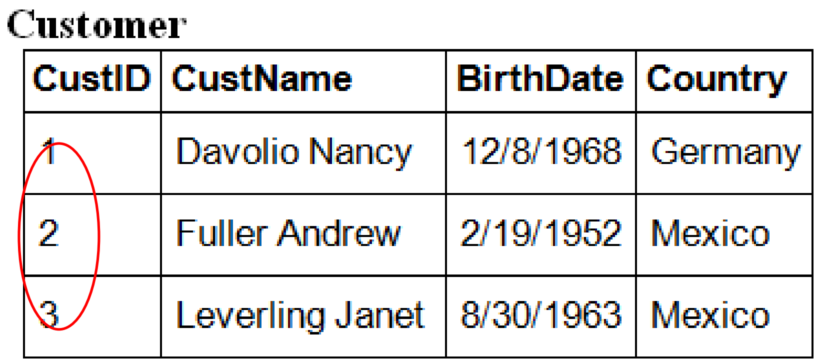
```
SELECT col_names
FROM Table_A  A INNER JOIN Table_B  B
ON A.Col1 = B.Col1
```



INNER JOIN

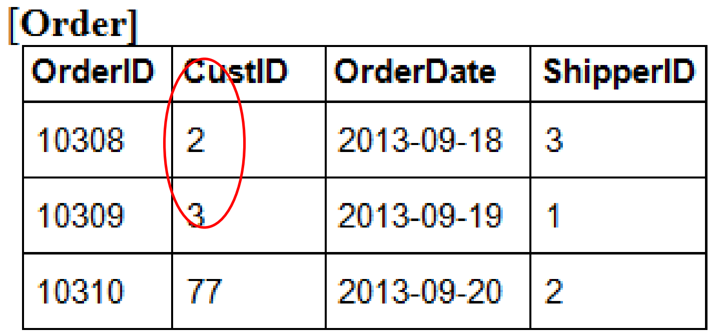
- **Example:**

```
SELECT c.CustName, o.OrderID
FROM Customer c INNER JOIN [Order] o ON c.CustID = o.CustID
ORDER BY c.CustName;
```



Customer

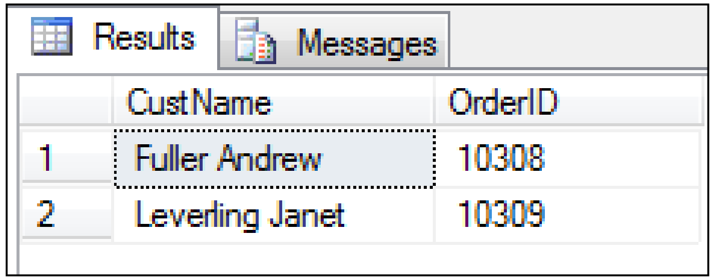
CustID	CustName	BirthDate	Country
1	Davolio Nancy	12/8/1968	Germany
2	Fuller Andrew	2/19/1952	Mexico
3	Leverling Janet	8/30/1963	Mexico



[Order]

OrderID	CustID	OrderDate	ShipperID
10308	2	2013-09-18	3
10309	3	2013-09-19	1
10310	77	2013-09-20	2

- **Result:**



	CustName	OrderID
1	Fuller Andrew	10308
2	Leverling Janet	10309

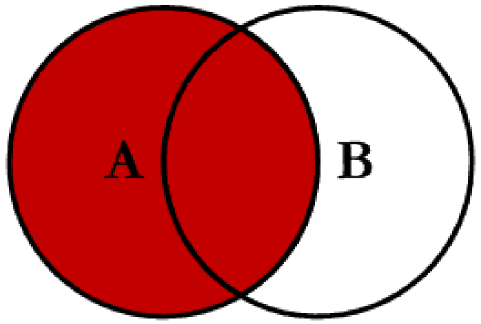
OUTER JOIN

- **Outer Join:** Return all rows from at least one of the tables mentioned in the FROM clause, as long as those rows meet any WHERE or HAVING search conditions:
  - ✓ **LEFT OUTER JOIN (or LEFT JOIN)**
  - ✓ **RIGHT OUTER JOIN (or RIGHT JOIN)**
  - ✓ **FULL OUTER JOIN (or FULL JOIN)**

LEFT OUTER JOIN

- Return all of the records in the left table (table A) regardless if any of those records has a match in the right table (table B)
  - ✓ In the results where there is no matching condition, the row contains NULL values for the right table's columns.
- **Syntax**

```
SELECT col_names
FROM Table_A  A  LEFT JOIN Table_B  B
ON A.Col1 = B.Col1
```



LEFT OUTER JOIN

- **Example:**

```sql
SELECT c.CustName, o.OrderID
FROM Customer c LEFT JOIN [Order] o ON c.CustID = o.CustID
ORDER BY c.CustName;
```

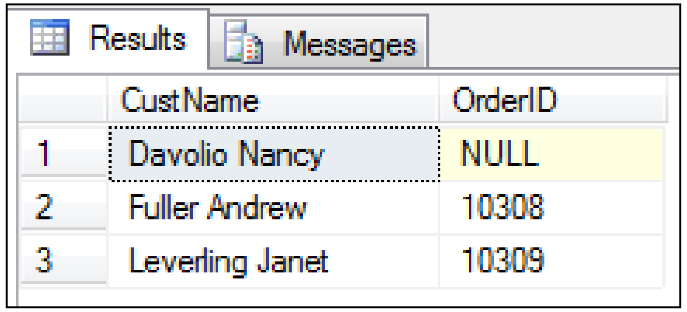
Customer

CustID	CustName	BirthDate	Country
1	Davolio Nancy	12/8/1968	Germany
2	Fuller Andrew	2/19/1952	Mexico
3	Leverling Janet	8/30/1963	Mexico

[Order]

OrderID	CustID	OrderDate	ShipperID
10308	2	2013-09-18	3
10309	3	2013-09-19	1
10310	77	2013-09-20	2

- **Result:**

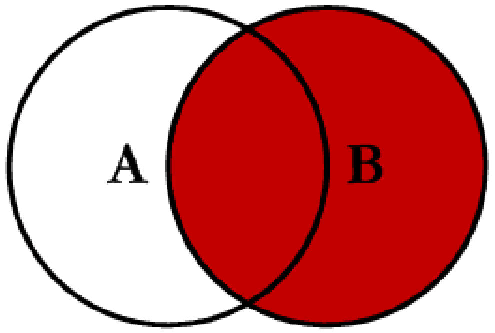


	CustName	OrderID
1	Davolio Nancy	NULL
2	Fuller Andrew	10308
3	Leverling Janet	10309

RIGHT OUTER JOIN

- Return all of the records in the right table (table B) regardless if any of those records have a match in the left table (table A)
  - ✓ In the results where there is no matching condition, the row contains NULL values for the left table's columns.
- **Syntax**

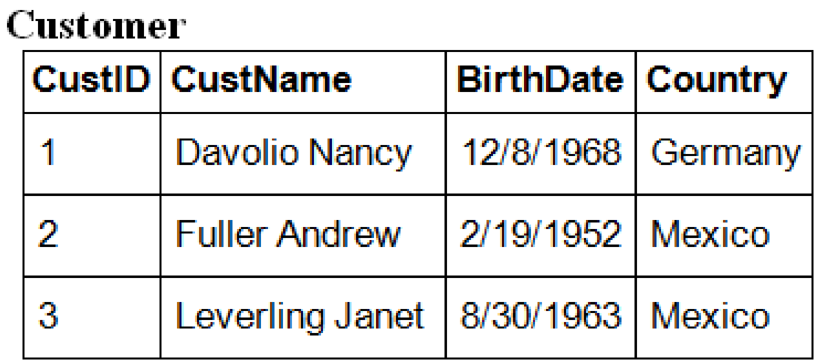
```
SELECT col_names
FROM Table_A  A  RIGHT JOIN Table_B  B
ON A.Col1 = B.Col1
```



RIGHT OUTER JOIN

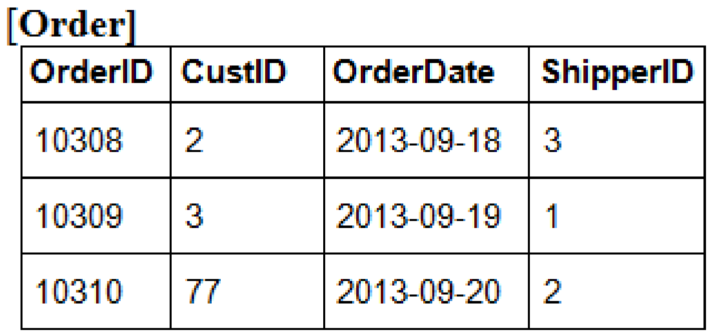
- **Example:**

```sql
SELECT c.CustName, o.OrderID
FROM Customer c RIGHT JOIN [Order] o ON c.CustID = o.CustID
ORDER BY c.CustName;
```



Customer

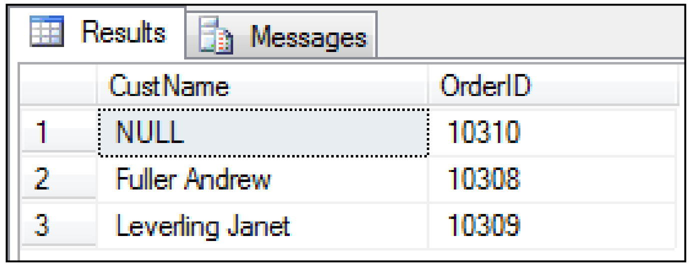
CustID	CustName	BirthDate	Country
1	Davolio Nancy	12/8/1968	Germany
2	Fuller Andrew	2/19/1952	Mexico
3	Leverling Janet	8/30/1963	Mexico



[Order]

OrderID	CustID	OrderDate	ShipperID
10308	2	2013-09-18	3
10309	3	2013-09-19	1
10310	77	2013-09-20	2

- **Result:**



	CustName	OrderID
1	NULL	10310
2	Fuller Andrew	10308
3	Leverling Janet	10309

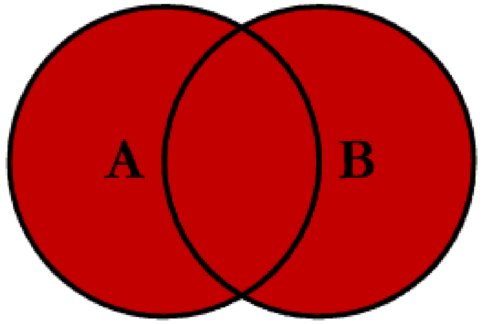
FULL OUTER JOIN

- Return all of the records from both tables, joining records from the left table (table A) that match records from the right table (table B)
- **Syntax**

```
SELECT col_names

FROM Table_A  A  FULL JOIN Table_B  B

ON A.Col1 = B.Col1
```



FULL OUTER JOIN

- **Example:**

```sql
SELECT c.CustName, o.OrderID
FROM Customer c FULL JOIN [Order] o ON c.CustID = o.CustID
ORDER BY c.CustName;
```

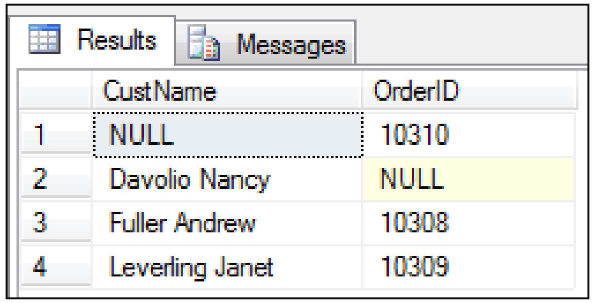
Customer

CustID	CustName	BirthDate	Country
1	Davolio Nancy	12/8/1968	Germany
2	Fuller Andrew	2/19/1952	Mexico
3	Leverling Janet	8/30/1963	Mexico

[Order]

OrderID	CustID	OrderDate	ShipperID
10308	2	2013-09-18	3
10309	3	2013-09-19	1
10310	77	2013-09-20	2

- **Result:**

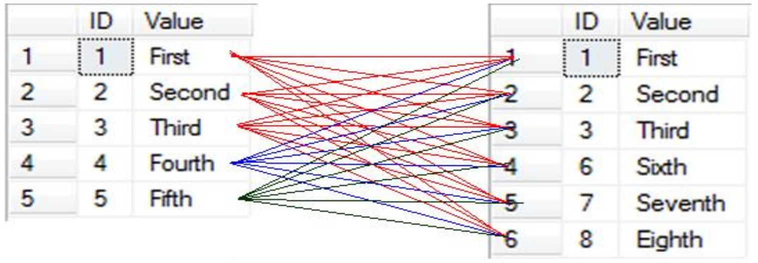


	CustName	OrderID
1	NULL	10310
2	Davolio Nancy	NULL
3	Fuller Andrew	10308
4	Leverling Janet	10309

CROSS JOIN

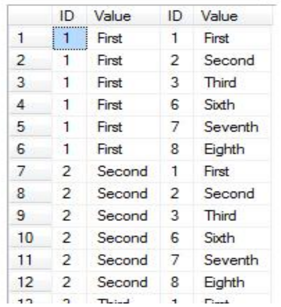
- Return records that are multiplication of record number from both the tables
  - ✓ No need any condition to join
- **Syntax**

```
SELECT col_names
FROM Table_A  A  CROSS JOIN Table_B  B
```



	ID	Value
1	1	First
2	2	Second
3	3	Third
4	4	Fourth
5	5	Fifth

	ID	Value
1	1	First
2	2	Second
3	3	Third
4	6	Sixth
5	7	Seventh
6	8	Eighth



	ID	Value	ID	Value
1	1	First	1	First
2	1	First	2	Second
3	1	First	3	Third
4	1	First	6	Sixth
5	1	First	7	Seventh
6	1	First	8	Eighth
7	2	Second	1	First
8	2	Second	2	Second
9	2	Second	3	Third
10	2	Second	6	Sixth
11	2	Second	7	Seventh
12	2	Second	8	Eighth

CROSS JOIN

- **Example:**

```
SELECT c.CustName, o.OrderID
FROM Customer c CROSS JOIN [Order] o
```

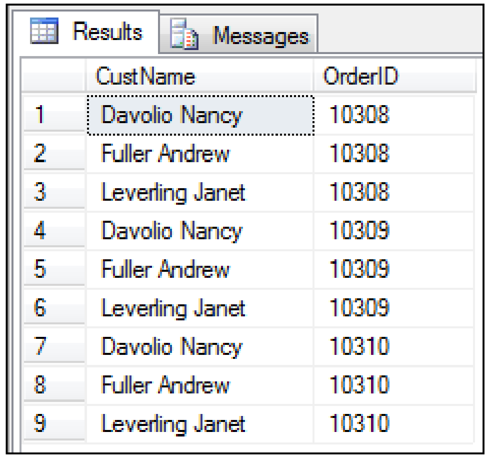
Customer

CustID	CustName	BirthDate	Country
1	Davolio Nancy	12/8/1968	Germany
2	Fuller Andrew	2/19/1952	Mexico
3	Leverling Janet	8/30/1963	Mexico

[Order]

OrderID	CustID	OrderDate	ShipperID
10308	2	2013-09-18	3
10309	3	2013-09-19	1
10310	77	2013-09-20	2

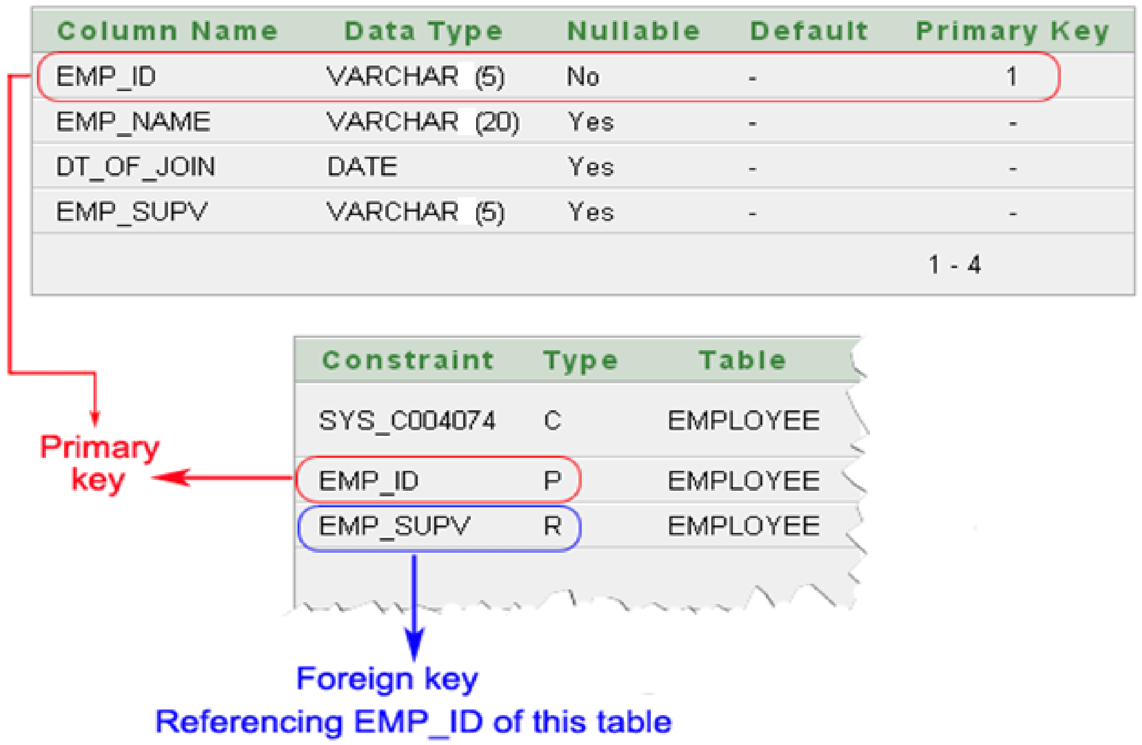
- **Result:**



	CustName	OrderID
1	Davolio Nancy	10308
2	Fuller Andrew	10308
3	Leverling Janet	10308
4	Davolio Nancy	10309
5	Fuller Andrew	10309
6	Leverling Janet	10309
7	Davolio Nancy	10310
8	Fuller Andrew	10310
9	Leverling Janet	10310

Self JOIN

- A SELF JOIN is a join of a table to itself. In SELF JOIN, we can use:
  - ✓INNER JOIN
  - ✓OUTER JOIN
  - ✓CROSS JOIN



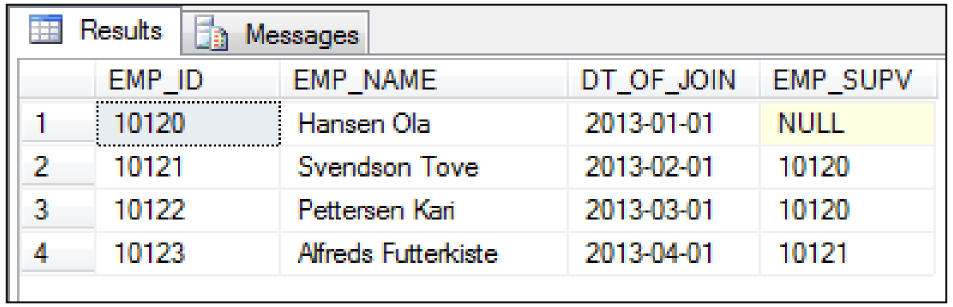
Column Name	Data Type	Nullable	Default	Primary Key
EMP_ID	VARCHAR (5)	No	-	1
EMP_NAME	VARCHAR (20)	Yes	-	-
DT_OF_JOIN	DATE	Yes	-	-
EMP_SUPV	VARCHAR (5)	Yes	-	-
				1 - 4

Constraint	Type	Table
SYS_C004074	C	EMPLOYEE
EMP_ID	P	EMPLOYEE
EMP_SUPV	R	EMPLOYEE

Self JOIN

- **Example:**

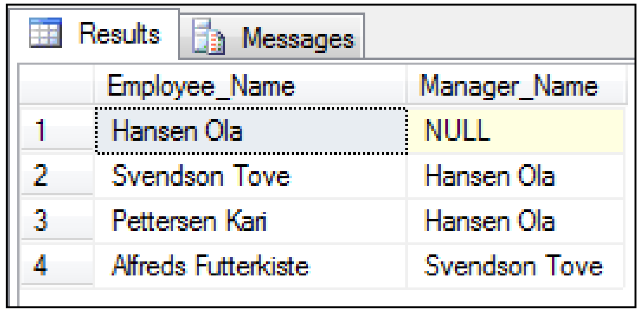
```
SELECT e1.EMP_NAME AS Employee_Name, e2.EMP_NAME AS Manager_Name
FROM Employee e1 LEFT JOIN Employee e2 ON e1.EMP_SUPV = e2.EMP_ID
```



Results | Messages

	EMP_ID	EMP_NAME	DT_OF_JOIN	EMP_SUPV
1	10120	Hansen Ola	2013-01-01	NULL
2	10121	Svendson Tove	2013-02-01	10120
3	10122	Pettersen Kari	2013-03-01	10120
4	10123	Alfreds Futterkiste	2013-04-01	10121

- **Result:**



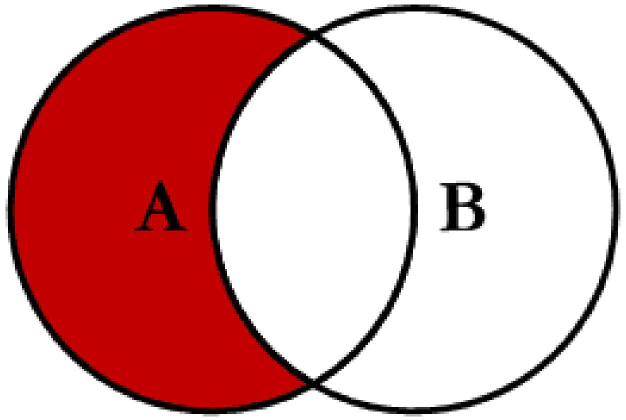
Results | Messages

	Employee_Name	Manager_Name
1	Hansen Ola	NULL
2	Svendson Tove	Hansen Ola
3	Pettersen Kari	Hansen Ola
4	Alfreds Futterkiste	Svendson Tove

LEFT Excluding JOIN

- Return all of the records in the **left table** (table A) that do not match any records in the right table (table B)
- **Syntax**

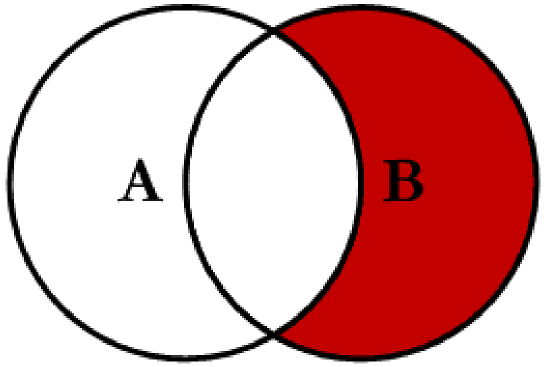
```
SELECT col_names
FROM Table_A  A  LEFT JOIN Table_B  B ON A.Col1 = B.Col1
WHERE B.Col1 IS NULL
```



RIGHT Excluding JOIN

- Returns records in the **right table** (table B) that do not match any records in the left table (table A)
  - ✓ In the results where there is no matching condition, the row contains NULL values for the right table's columns.
- **Syntax**

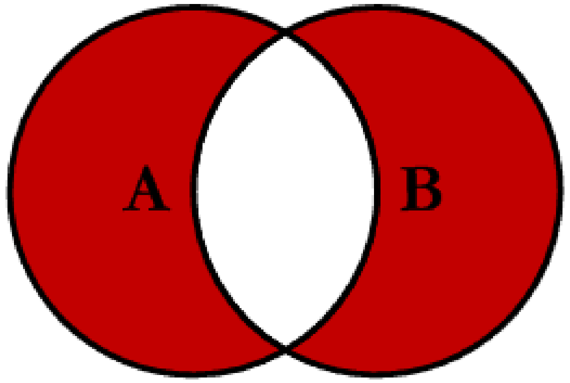
```
SELECT col_names
FROM Table_A  A  RIGHT JOIN Table_B  B ON A.Col1 = B.Col1
WHERE A.Col1 IS NULL
```



OUTER JOIN EXCLUDING JOIN

- Return all of the records in the **left table** (table A) and **all of the records in the right table** (table B) that do not match.
- **Syntax**

```
SELECT col_names
FROM Table_A  A
FULL OUTER JOIN Table_B  B ON A.Col1 = B.Col1
WHERE A.Col1 IS NULL OR B.Col1 IS NULL
```



Joining Three or More Tables

- Since FROM clauses can contain multiple join specifications, this allows many tables to be joined for a single query.
- **Syntax**

```
SELECT col_names
FROM Table_A  A  JOIN Table_B  B
ON A.Col1 = B.Col1 LEFT JOIN Table_C C
ON B.Col2 = C.Col2
....
```

Section 2

SUBQUERY

What is a subquery?

- A **sub-query**, also called an **inner query**, is a SQL query nested inside a larger query.
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.

What is a subquery?

- **Syntax (example: subquery within the Where) :**

```
SELECT      select_list
FROM        table
WHERE       expr operator
                         (SELECT      select_list
                          FROM        table);
```

- **Exam:**

```
SELECT SUM (Sales) AS Sale_Sum FROM Store_Information
WHERE Store_Name IN
              (SELECT Store_Name FROM Geography
              WHERE Region_Name = 'West');
```

Sale_Sum
2050

Table *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	Jan-05-1999
San Diego	250	Jan-07-1999
Los Angeles	300	Jan-08-1999
Boston	700	Jan-08-1999

Table *Geography*

Region_Name	Store_Name
East	Boston
East	New York
West	Los Angeles
West	San Diego

What is a subquery? (3/3)

- **How to work?**:
  - ✓ Inner query is independent of outer query.
  - ✓ Inner query is executed first and the results are stored.
  - ✓ Outer query then runs on the stored results.
- *Note about specific type*: **Correlated** subqueries (be mentioned in the next slides)

Subquery Types

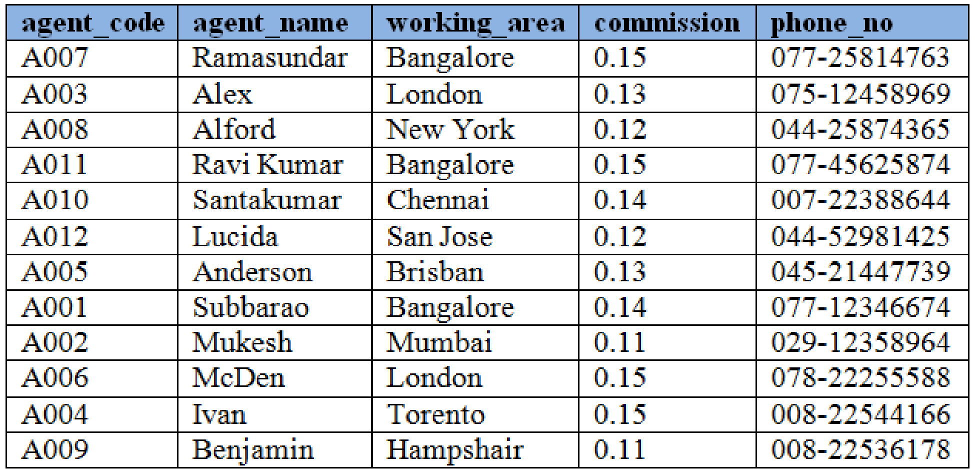
- **Single row** subquery
- **Multiple row** subquery
- **Multiple column** subquery
- **Correlated** subquery
- **Nested** subquery

Single row subquery

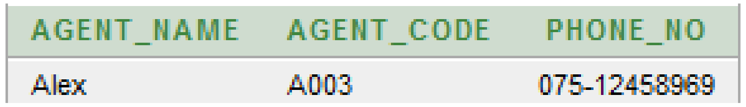
- A single row subquery returns zero or one row to the outer SQL statement. You can place a subquery in a **WHERE** clause, a **HAVING** clause, or a **FROM** clause of a **SELECT** statement.
- **Exam**: Single Row subqueries in WHERE clause

```
SELECT agent_name, agent_code, phone_no
FROM agents
WHERE agent_code = (SELECT agent_code FROM agents WHERE agent_name = 'Alex')
```

agents table



agent_code	agent_name	working_area	commission	phone_no
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.13	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425
A005	Anderson	Brisban	0.13	045-21447739
A001	Subbarao	Bangalore	0.14	077-12346674
A002	Mukesh	Mumbai	0.11	029-12358964
A006	McDen	London	0.15	078-22255588
A004	Ivan	Torento	0.15	008-22544166
A009	Benjamin	Hampshair	0.11	008-22536178



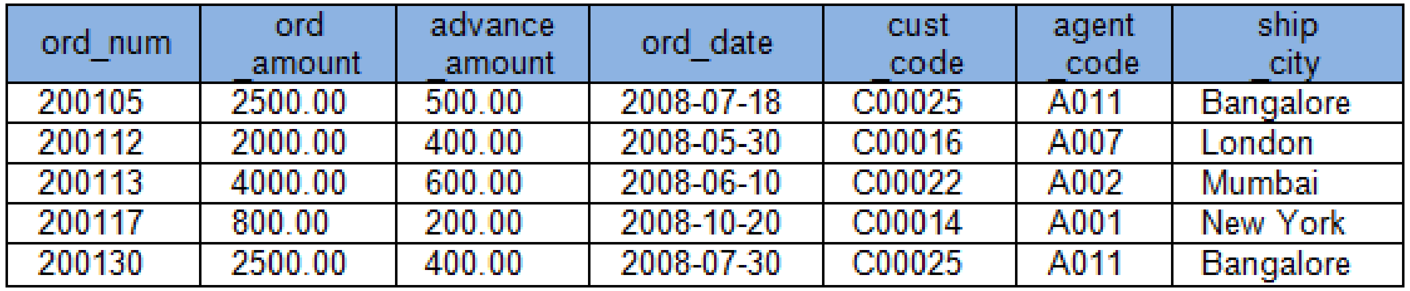
AGENT_NAME	AGENT_CODE	PHONE_NO
Alex	A003	075-12458969

Multiple row subquery

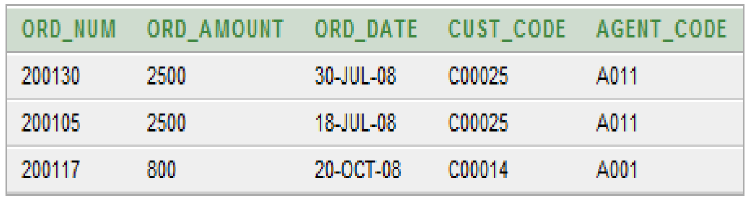
- Multiple row subquery returns one or more rows to the outer SQL statement. You may use the **IN**, **ANY**, or **ALL** operator in outer query to handle a subquery that returns multiple rows.
- **Ex**: Multiple row Subquery in a WHERE clause

```
SELECT ord_num, ord_amount, ord_date, cust_code, agent_code
FROM orders
WHERE agent_code IN (SELECT agent_code FROM agents WHERE working_area='Bangalore')
```

orders table



ord_num	ord_amount	advance_amount	ord_date	cust_code	agent_code	ship_city
200105	2500.00	500.00	2008-07-18	C00025	A011	Bangalore
200112	2000.00	400.00	2008-05-30	C00016	A007	London
200113	4000.00	600.00	2008-06-10	C00022	A002	Mumbai
200117	800.00	200.00	2008-10-20	C00014	A001	New York
200130	2500.00	400.00	2008-07-30	C00025	A011	Bangalore



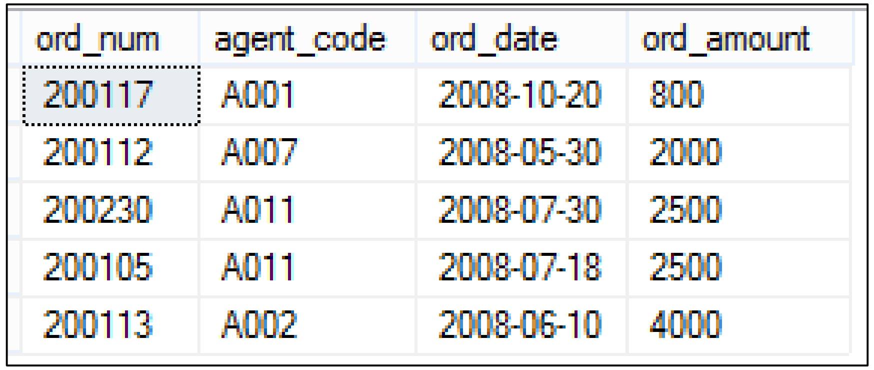
ORD_NUM	ORD_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE
200130	2500	30-JUL-08	C00025	A011
200105	2500	18-JUL-08	C00025	A011
200117	800	20-OCT-08	C00014	A001

Multiple column subquery

- You can write subqueries that return multiple columns.
- **Ex**: Multiple column Subquery in a FROM clause

```sql
SELECT ord_num, agent_code, ord_date, ord_amount
FROM orders o1
WHERE EXISTS(
        SELECT agent_code, ord_amount
        FROM orders o2
        WHERE o1.agent_code = o2.agent_code AND o1.ord_amount = o2.ord_amount)
ORDER BY ord_amount ASC
```

- **Result:**



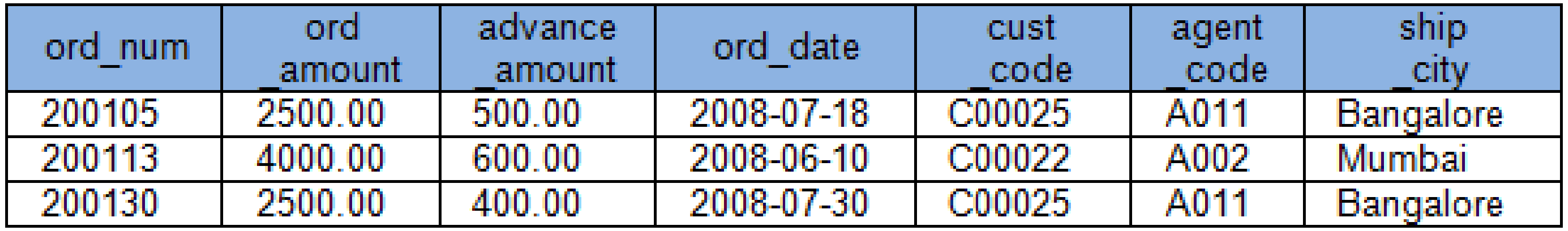
ord_num	agent_code	ord_date	ord_amount
200117	A001	2008-10-20	800
200112	A007	2008-05-30	2000
200230	A011	2008-07-30	2500
200105	A011	2008-07-18	2500
200113	A002	2008-06-10	4000

Correlated subquery

- Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
- **Ex**: Correlated Subquery in a FROM clause

```
SELECT *   FROM orders o
WHERE agent_code IN (   SELECT agent_code FROM agents a
                        WHERE o.ship_city = a.working_area)
```

- **Result:**



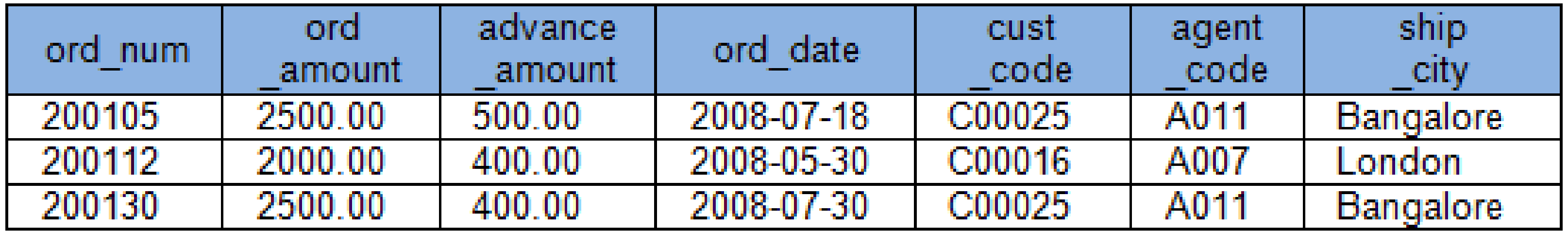
ord_num	ord_amount	advance_amount	ord_date	cust_code	agent_code	ship_city
200105	2500.00	500.00	2008-07-18	C00025	A011	Bangalore
200113	4000.00	600.00	2008-06-10	C00022	A002	Mumbai
200130	2500.00	400.00	2008-07-30	C00025	A011	Bangalore

Nested subquery

- A subquery can be nested inside other subqueries.
- **Ex**: Nested Subquery in a WHERE clause

```
SELECT *
FROM orders
WHERE ship_city IN (SELECT DISTINCT working_area FROM agents WHERE agent_code
                    IN
                    (SELECT agent_code FROM agents WHERE commission >= 0.14))
```

- **Result:**



ord_num	ord _amount	advance _amount	ord_date	cust _code	agent _code	ship _city
200105	2500.00	500.00	2008-07-18	C00025	A011	Bangalore
200112	2000.00	400.00	2008-05-30	C00016	A007	London
200130	2500.00	400.00	2008-07-30	C00025	A011	Bangalore

Common caseS use subquery

- **We focus on some typical usecases for Subquery:**
  - ✓ ***Subqueries with Aliases****:* Many statements in which the subquery and the outer query refer to the same table
  - ✓ ***Subqueries with IN / NOT IN****:* The result of a subquery introduced with IN (or with NOT IN) is a list of zero or more values. After the subquery returns results, the outer query makes use of them
  - ✓ ***Subqueries with EXISTS / NOT EXISTS****:* The subquery functions as an existence test.
  - ✓ ***Subqueries in UPDATE, DELETE, INSERT, SELECT***

Rules that subqueries must follow

- You must enclose a subquery in parenthesis.
- A subquery must include a SELECT clause and a FROM clause.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- A subquery can include optional WHERE, GROUP BY, and HAVING clauses.
- A subquery cannot include COMPUTE or FOR BROWSE clauses.
- You can include an ORDER BY clause only when a TOP clause is included.
- You can nest subqueries up to 32 levels.

Section 3

COMMON TABLE EXPRESSIONS

Common Table Expressions

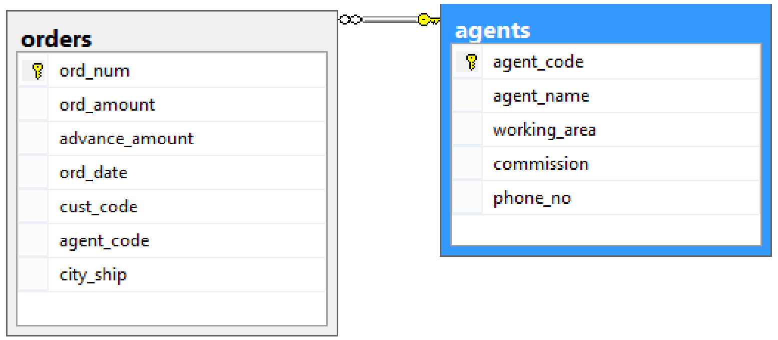
- A CTE can be thought of as a **temporary result set** that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE. It can be used:
  - ✓ This is used to store result of a complex sub query for further use.(As a temporary table)
  - ✓ Create a recursive query.
- **Syntax:**

```
;WITH CTE_Name [ col_names]
AS
(
		CTE_query_definition
)
```

Common Table Expressions

- This is used to store result of a complex sub query for further use.(As a temporary table)
- **Subquery :**

```sql
SELECT a.agent_name, a.working_area, COUNT(o.agent_code) AS AMOUNT_AGENT
FROM dbo.agents a INNER JOIN dbo.orders o ON A.agent_code = O.agent_code
WHERE A.working_area = 'Bungnlme'
GROUP BY a.agent_name, a.working_area
```

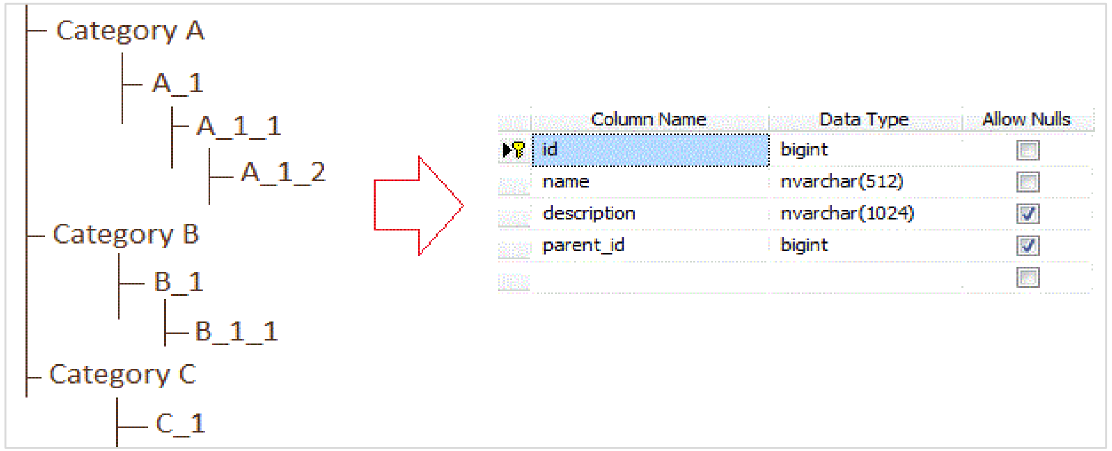


- **Replace by CTE:**

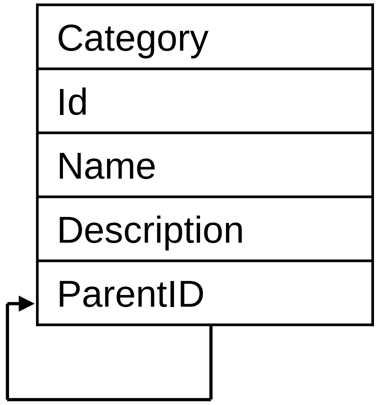
```sql
; WITH cte_Agents(agent_name, working_area, amount_of_agent)
AS
(
SELECT a.agent_name, a.working_area, COUNT(o.agent_code) AS AMOUNT_AGENT
FROM dbo.agents a INNER JOIN dbo.orders o ON A.agent_code = O.agent_code
GROUP BY a.agent_name, a.working_area
)
SELECT * FROM cte_Agents cte
WHERE cte.working_area = 'Bungnlme'
```

Common Table Expressions

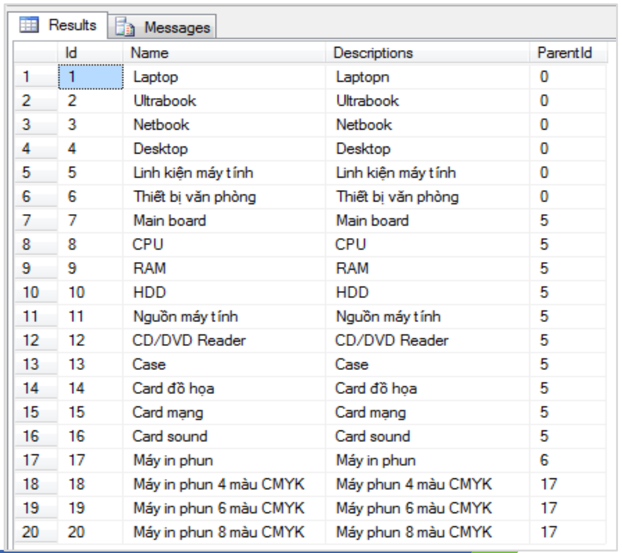
- **Create a recurszive query.**



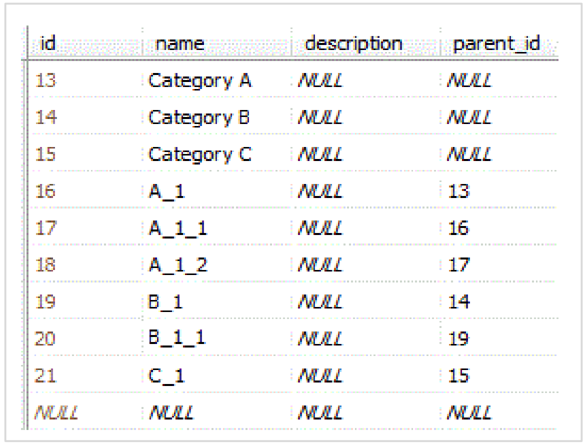
Column Name	Data Type	Allow Nulls
id	bigint	☐
name	nvarchar(512)	☐
description	nvarchar(1024)	☑
parent_id	bigint	☑
		☐



- **Ex:**



	Id	Name	Descriptions	ParentId
1	1	Laptop	Laptopn	0
2	2	Ultrabook	Ultrabook	0
3	3	Netbook	Netbook	0
4	4	Desktop	Desktop	0
5	5	Linh kiện máy tính	Linh kiện máy tính	0
6	6	Thiết bị văn phòng	Thiết bị văn phòng	0
7	7	Main board	Main board	5
8	8	CPU	CPU	5
9	9	RAM	RAM	5
10	10	HDD	HDD	5
11	11	Nguồn máy tính	Nguồn máy tính	5
12	12	CD/DVD Reader	CD/DVD Reader	5
13	13	Case	Case	5
14	14	Card đồ họa	Card đồ họa	5
15	15	Card mạng	Card mạng	5
16	16	Card sound	Card sound	5
17	17	Máy in phun	Máy in phun	6
18	18	Máy in phun 4 màu CMYK	Máy phun 4 màu CMYK	17
19	19	Máy in phun 6 màu CMYK	Máy phun 6 màu CMYK	17
20	20	Máy in phun 8 màu CMYK	Máy phun 8 màu CMYK	17



id	name	description	parent_id
13	Category A	NULL	NULL
14	Category B	NULL	NULL
15	Category C	NULL	NULL
16	A_1	NULL	13
17	A_1_1	NULL	16
18	A_1_2	NULL	17
19	B_1	NULL	14
20	B_1_1	NULL	19
21	C_1	NULL	15
NULL	NULL	NULL	NULL

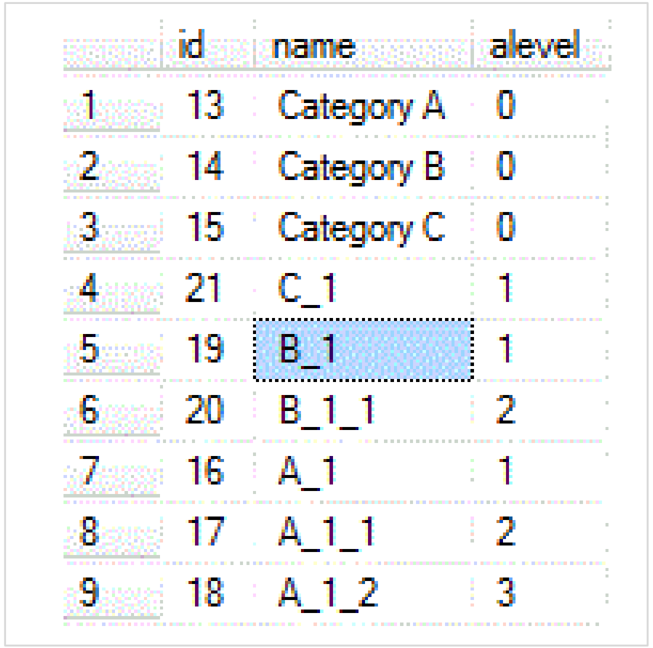
Proplem: Select the Level of each element ??

Common Table Expressions

- **Solution for this example:**

```
;WITH temp(id, name, aLevel)
AS
(
      SELECT id, name, 0 AS aLevel
      FROM Category WHERE parent_id is null
      UNION All
      SELECT b.id, b.name, a.alevel + 1
      FROM temp AS a, Category AS b
      WHERE a.id = b.parent_id
)
SELECT * FROM temp
```

Result:



	id	name	alevel
1	13	Category A	0
2	14	Category B	0
3	15	Category C	0
4	21	C_1	1
5	19	B_1	1
6	20	B_1_1	2
7	16	A_1	1
8	17	A_1_1	2
9	18	A_1_2	3

Common Table Expressions

- **Recursive Queries Using Common Table Expressions**
- **Syntax:**

```
WITH cte_name ( col_names)
AS
(
            -- Anchor member is defined.
            CTE_query_definition
            UNION ALL
            -- Recursive member is defined referencing cte_name.
            CTE_query_definition
)
-- Statement using the CTE
SELECT * FROM cte_name
```

Session 4

RANKING FUNCTIONS

Ranking functions

- **Ranking functions:** Ranking functions provides the ability to rank each row of data.

RANK	NTILE
DENSE_RANK	ROW_NUMBER

- **Four kinds of Ranking functions:**

  ROW_NUMBER

  RANK_DENSE

  NTILE

  RANK

Ranking functions

- Let's take following sample table and data to know about **RANK**, **RANK_DENSE**, **NTILE** and **ROW_NUMBER** with examples:

```
CREATE TABLE ExamResult(FullName varchar(50), Subject varchar(20), Marks int)

INSERT INTO ExamResult VALUES('Adam','Maths',70)
INSERT INTO ExamResult VALUES ('Adam','Science',80)
INSERT INTO ExamResult VALUES ('Adam','Social',60)

INSERT INTO ExamResult VALUES('Rak','Maths',60)
INSERT INTO ExamResult VALUES ('Rak','Science',50)
INSERT INTO ExamResult VALUES ('Rak','Social',70)

INSERT INTO ExamResult VALUES('Sam','Maths',90)
INSERT INTO ExamResult VALUES ('Sam','Science',90)
INSERT INTO ExamResult VALUES ('Sam','Social',80)
```

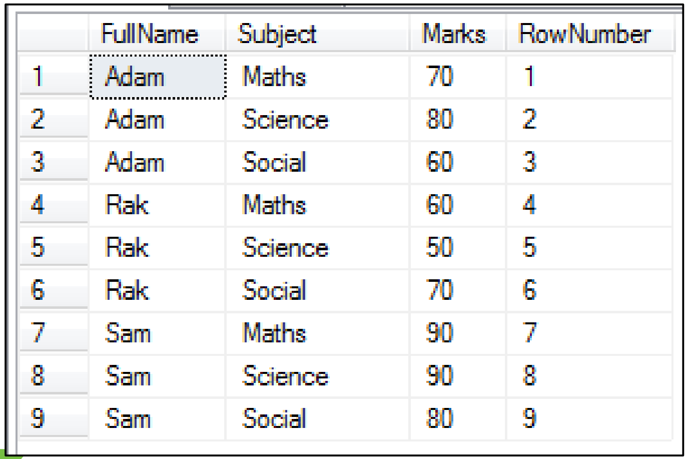
Ranking functions

- **Row_Number:** Returns the sequential number of a row within a partition of a result set
- **Example:**

```
SELECT FullName, Subject, Marks, ROW_NUMBER() OVER(ORDER BY FullName) RowNumber

FROM ExamResult

ORDER BY FullName, Subject
```

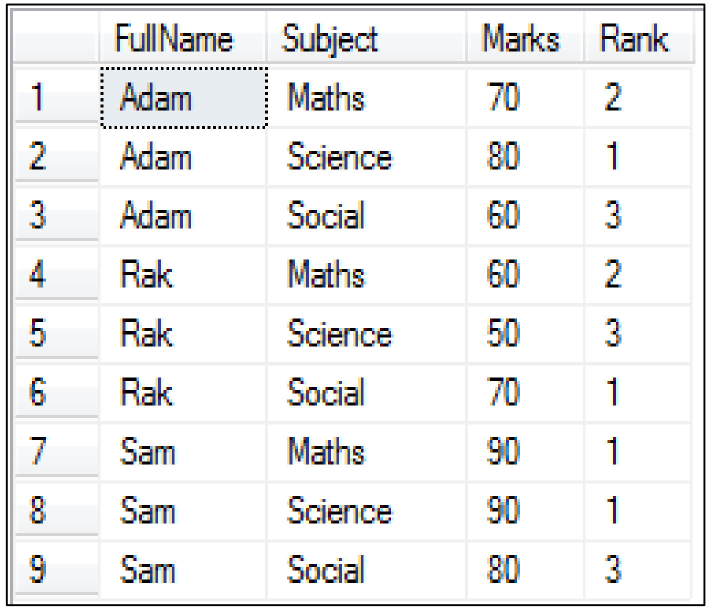


	FullName	Subject	Marks	RowNumber
1	Adam	Maths	70	1
2	Adam	Science	80	2
3	Adam	Social	60	3
4	Rak	Maths	60	4
5	Rak	Science	50	5
6	Rak	Social	70	6
7	Sam	Maths	90	7
8	Sam	Science	90	8
9	Sam	Social	80	9

Ranking functions

- **Rank**: Returns the rank of each row within the partition of a result set
- **Example**:

```
SELECT FullName, Subject, Marks, RANK() OVER(PARTITION BY
                              FullName ORDER BY Marks DESC) Rank
FROM ExamResult
ORDER BY FullName, Subject
```



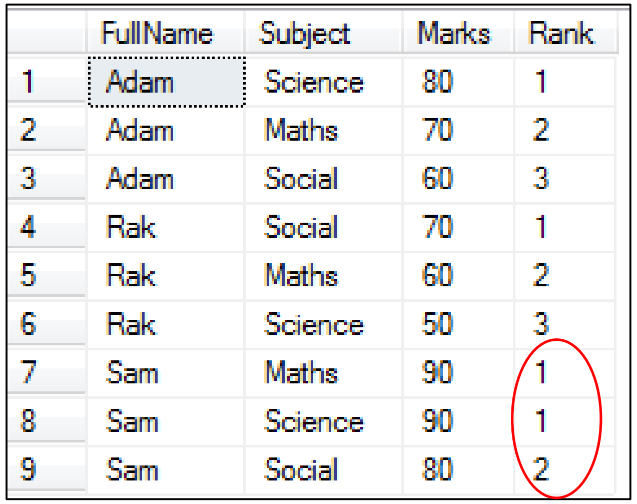
	FullName	Subject	Marks	Rank
1	Adam	Maths	70	2
2	Adam	Science	80	1
3	Adam	Social	60	3
4	Rak	Maths	60	2
5	Rak	Science	50	3
6	Rak	Social	70	1
7	Sam	Maths	90	1
8	Sam	Science	90	1
9	Sam	Social	80	3

Ranking functions

- **Dense_Rank**: Returns the rank of rows within the partition of a result set, without any gaps in the ranking
- **Example**:

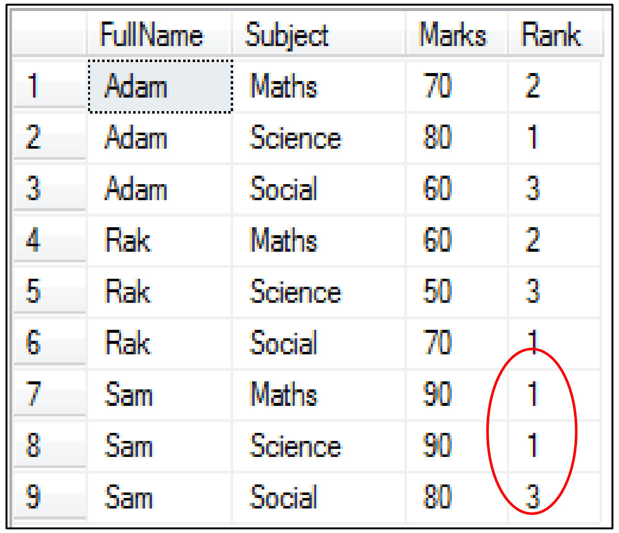
```
SELECT    FullName, Subject, Marks, DENSE_RANK()
                           OVER (PARTITION BY FullName ORDER BY Marks DESC) Rank
FROM ExamResult
ORDER BY FullName
```

Dense_Rank



	FullName	Subject	Marks	Rank
1	Adam	Science	80	1
2	Adam	Maths	70	2
3	Adam	Social	60	3
4	Rak	Social	70	1
5	Rak	Maths	60	2
6	Rak	Science	50	3
7	Sam	Maths	90	1
8	Sam	Science	90	1
9	Sam	Social	80	2

Rank

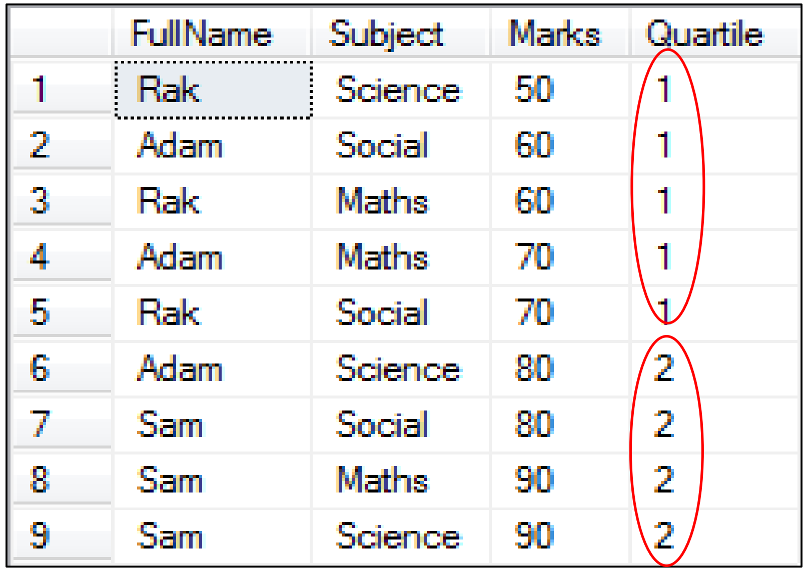


	FullName	Subject	Marks	Rank
1	Adam	Maths	70	2
2	Adam	Science	80	1
3	Adam	Social	60	3
4	Rak	Maths	60	2
5	Rak	Science	50	3
6	Rak	Social	70	1
7	Sam	Maths	90	1
8	Sam	Science	90	1
9	Sam	Social	80	3

Ranking functions

- **Ntitle**: Distributes the rows in an ordered partition into a specified number of groups
- **Example**:

```
SELECT FullName, Subject, Marks,
            NTILE(2) OVER   (ORDER BY Marks DESC)Quartile
FROM ExamResult
```



	FullName	Subject	Marks	Quartile
1	Rak	Science	50	1
2	Adam	Social	60	1
3	Rak	Maths	60	1
4	Adam	Maths	70	1
5	Rak	Social	70	1
6	Adam	Science	80	2
7	Sam	Social	80	2
8	Sam	Maths	90	2
9	Sam	Science	90	2

Summary

- SQL JOINs
  - ✓ Inner Join
  - ✓ Outer Join
  - ✓ Self Join
  - ✓ Cross Join
- Sub-Query
- CTE and RANKING Functions

References

- *https://learn.microsoft.com/en-us/sql/relational-databases/performance/joins?view=sql-server-ver16/*
- *https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/*
- *https://learnsql.com/blog/what-is-common-table-expression/*

THANK YOU!