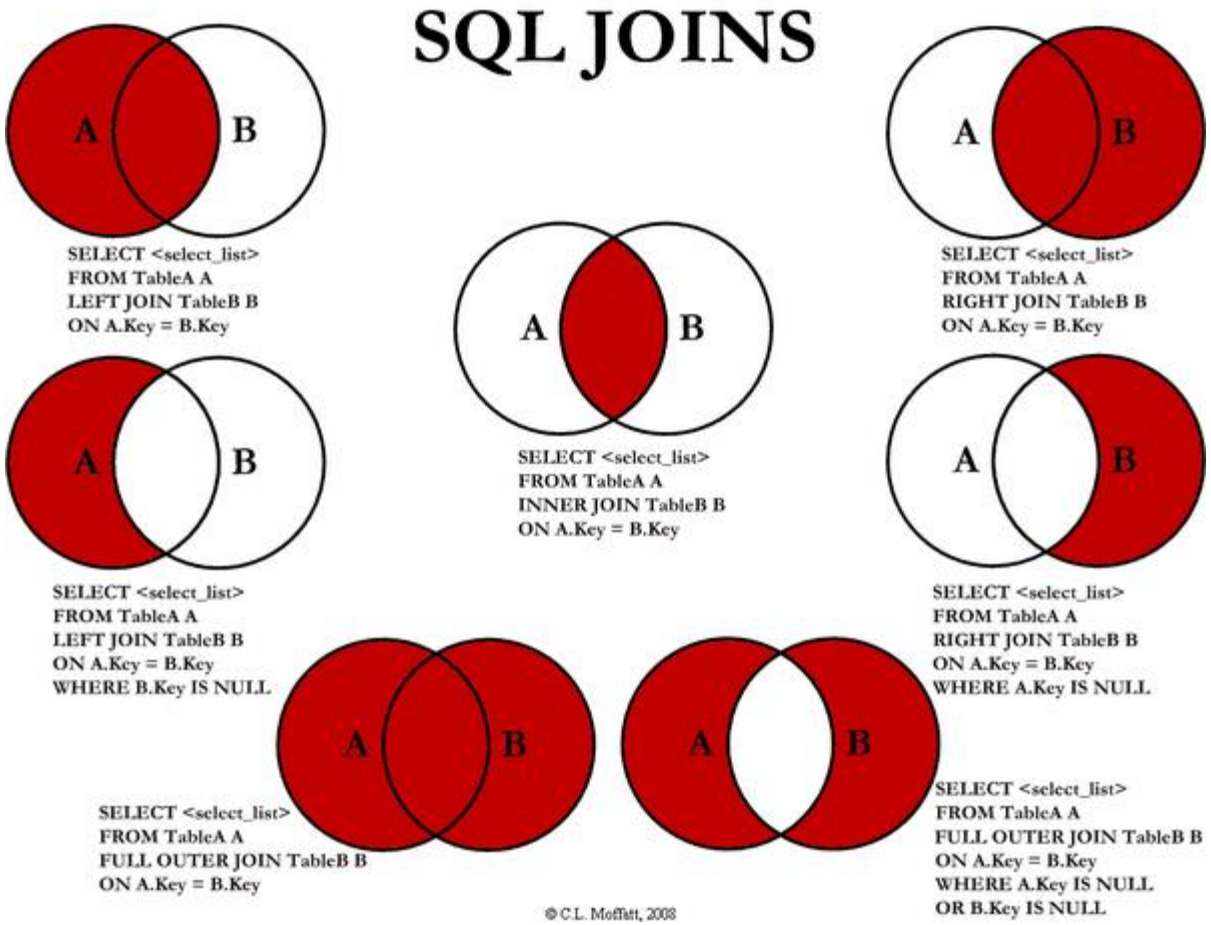
JOINS



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

Union and union all

- UNION combines the results of two or more queries into a single result set that includes all the rows that belong to all queries in the union
- UNION removes duplicate records (where all columns in the results are the same), UNION ALL does not.

Union and union all

```
select * from Student
where St_Age<=23
union
 select * from Student
where St_Age>22
```

```
select * from Student
where St_Age<=23
union all
 select * from Student
where St_Age>22
```

Age 23 will be repeated

Intersect and except

- INTERSECT returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operand
- EXCEPT returns any distinct values from the query to the left of the EXCEPT operand that are not also returned from the right query

Intersect and except

```
 select * from Student
 where St_Age<=23
intersect
  select * from Student
 where St_Age>22
```

Will return age=23

```
 select * from Student
 where St_Age<=23
except
  select * from Student
 where St_Age>22
```

Working with subqueries

- What is subquery?
- Types of subqueries?
- Subqueries vs. joins.
- Exists Condition

What is subquery?

- SQL supports writing queries within queries, or *nesting* queries. The outermost query is a query whose result set is returned to the caller(user) and is known as the *outer query*.
- The inner query is a query whose result is used by the outer query and is known as a *subquery*.

Types of subqueries

- Self-Contained Scalar Subquery
- Self-Contained Multivalued Subquery
- table subqueries
- correlated subqueries

Self-Contained Scalar Subquery

- A scalar subquery is a subquery that returns a single value
- Self-contained subqueries are subqueries that are independent of the outer query that they belong to.

```
select * from Instructor
where Salary=(select max(salary) from Instructor)
```

Self-Contained Multivalued Subquery

- A multivalued subquery is a subquery that returns multiple values as a single column
- There are predicates that operate on a multivalued subquery; those are IN,*ANY*, and *ALL*.

```
select * from Instructor where salary IN
(select distinct top 3  salary from Instructor
order by Salary desc)
```

Will return instructors with top 3 salaries

correlated subqueries

- Correlated subqueries are subqueries that refer to attributes from the table that appears in the outer query.
- This means that the subquery is dependent on the outer query and cannot be invoked independently.

```
select * from Instructor as ins1
where salary=(select MAX(salary)
from Instructor as ins2
where ins2.Dept_Id=ins1.Dept_Id)
```

Instructors take maximum salary in each department

table subqueries

- Derived tables (also known as *table subqueries*) are defined in the *FROM* clause of an outer query. Their scope of existence is the outer query. As soon as the outer query is finished, the derived table is gone.
- You must assign alias for the derived table

To use this derived table in SELECT and WHERE.

table subqueries

```
• select ins1.*
 from Instructor as ins1,
 (select Dept_Id,MAX(salary) as salar from Instructor as ins2 group by
 Dept_Id) as x

 where  ins1.Salary=x.salar and ins1.Dept_Id=x.Dept_Id
```

EXISTS

- T-SQL supports a predicate called *EXISTS* that accepts a subquery as input and returns *TRUE* if the subquery returns any rows and *FALSE* otherwise.

```
SELECT custid, companyname
FROM Sales.Customers AS C
WHERE country = N'Spain'
AND EXISTS
(SELECT * FROM Sales.Orders AS O
WHERE O.custid = C.custid);
```

the following query returns customers from Spain who placed orders.

Sub-queries vs. joins

- Joins are performed faster by SQL Server than subqueries
- Subqueries are useful for answering questions that are too complex to answer with joins(meaningful)
- SQL Server 2012 query optimizer is intelligent enough to convert a subquery into a join if it can be done

Rollup and cube

- ROLLUP generates a result set showing the aggregates for a hierarchy of values in selected columns
- CUBE generates a result set that shows the aggregates for all combination of values in selected columns

Rollup and cube

```
SELECT a, b, c, SUM ( <expression> )

FROM     T

GROUP BY ROLLUP (a,b,c)
```

```
SELECT a, b, c, SUM (<expression>)

FROM     T

GROUP BY CUBE (a,b,c)
```

Other DML statements

- Insert data in a table
- Deleting data from table
- Updating data in tables

Insert

- Inserting simple rows of values

```
INSERT [INTO] schema.table [(columns, ...)]
VALUES (value,...), (value,...), ... ;
```

- Inserting a result set from select

```
INSERT[INTO] schema.Table [(columns, ...)]
SELECTcolumns
FROM data sources
[WHERE conditions];
```

Insert

- Inserting the result set from a stored procedure

```
INSERT  [INTO] schema.Table [(Columns)]
EXEC StoredProcedure Parameters;
```

- Creating a table while inserting data

```
SELECT Columns INTO NewTable
FROM DataSources
[WHERE conditions];
```

Update

```
UPDATE schema.Table
SET column = expression,
column = value...
[FROM data sources]
[WHERE conditions];
```

- The WHERE clause is vital to any UPDATE statement. Without it, the entire table is updated.

Delete

```
DELETE [FROM]  schema.Table
[FROM data sources]
[WHERE condition(s)];
```

```
DELETE FROM dbo.Product
WHERE ProductID = ‘DB8D8D60-76F4-46C3-90E6-A8648F63C0F0’;
```

```
DELETE dbo.Product
FROM dbo.Product
JOIN dbo.ProductCategory
ON Product.ProductCategoryID
= ProductCategory.ProductCategoryID
WHERE ProductCategory.ProductCategoryName = ‘Video’;
```

Delet product with category video

Delete

```
DELETE  schema.Table
```

Delete all table

```
delete top(3)
from New_Table
```

Merge statement

- Using a single statement, we can Add/Update records in our database table, without explicitly checking for the existence of records to perform operations like Insert or Update.
- Joins a data source with a target table or view
- Performs multiple actions based on the results of the join

Merge statement

MERGE [INTO] <target table> USING <source table or table expression> ON <join/merge predicate> (semantics similar to outer join) WHEN MATCHED <statement to run when match found in target> WHEN [TARGET] NOT MATCHED <statement to run when no match found in target>

Merge statement

```
merge into [dbo].[Customer] as c
using [dbo].[CustomerTemp] as ct
on c.nationalid=ct.nationalid
when matched and (c.name!=ct.name or c.phone!=ct.phone or
c.amount!=ct.amount)
then update  set c.name=ct.name ,c.phone=ct.phone,c.amount+=ct.amount
when not matched then insert values
(ct.nationalid,ct.name,ct.phone,ct.amount)
;
```

Ranking functions

- Row_number()
- Rank()
- Dense_rank()
- Ntile()

Row_number()

- The ROW_NUMBER() function generates an auto-incrementing integer according to the sort order of the OVER()clause.

ID	Name	Age	Row_number()
1	mohamed	20	1
2	ahmed	21	2
3	hassan	22	3
4	osama	23	4
5	amr	23	5
6	zkkk	24	6
7	pppp	25	7

Rank() and Dense_rank()

- return values as if the rows were competing according to the windowed sort order. Any ties are grouped together with the same ranked value.

Rank() Example

ID	Name	Age	Age_rank
1	mohamed	20	1
2	ahmed	21	2
3	hassan	22	3
4	osama	23	4
5	amr	23	4
6	zkkk	24	6
7	pppp	25	7

Dense_rank() example

ID	Name	Age	Age_rank
1	mohamed	20	1
2	ahmed	21	2
3	hassan	22	3
4	osama	23	4
5	amr	23	4
6	zkkk	24	5
7	pppp	25	6

Ntile()

- organizes the rows into n number of groups, called tiles, and returns the tile number.

```
select *,Ntile(5)  over (order by st_age) as age_rank
from Student
order by St_Id
```

Ntile()

ID	Name	Age	Age_rank
1	mohamed	20	1
2	ahmed	21	1
3	hassan	22	2
4	osama	23	2
5	amr	23	3
6	zkkk	24	4
7	pppp	25	5

Partitioning within the window

- but it can divide the windowed data into

partitions, which are similar to groups in an aggregate GROUP BY query

- the ranking functions will be able to restart with every partition.

Example

```
select *,row_number() over (partition by dept_id order by st_age) as
age_rank
from Student
```

ID	Name	Age	Dept_id	Age_rank
1	mohamed	20	10	1
2	ahmed	21	10	2
3	hassan	22	11	1
4	osama	23	11	2
5	amr	23	11	3
6	zkkk	24	12	1
7	pppp	25	12	2

View

- Views are sometimes described as virtual tables.
- View is the saved text of a SQL SELECT statement that may be referenced as a data source within a query
- similar to how a subquery can be used as data source

Why we use views?

- Simplify construction of complex queries
- Save complex aggregate queries as views
- Hide DB Objects

Creating views

```
CREATEVIEW schemaname.ViewName [(Column aliases)]
AS
SQL Select Statement;
```

```
CREATE VIEW   dbo.vEmployeeList
AS
SELECT P.BusinessEntityID, P.Title, P.LastName,
P.FirstName, E.JobTitle
FROM Person.Person P
INNER JOIN HumanResources.Employee E
ON P.BusinessEntityID = E.BusinessEntityID
```

Executing views

- A query (SELECT, INSERT, UPDATE, DELETE,orMERGE) can include the view as a data source

```
Select  * from  dbo.vEmployeeList
```

Restrictions on views

- Total number of columns referenced in the view cannot exceed 1024
- Order by Cannot be used in views, inline functions, derived tables
- Select * →Can be used in a view definition if the SCHEMABINDING clause is not specified

Column aliases

- The view’s column list names override any column names or column aliases in the view’s SELECT statement.

```
ALTER VIEW    dbo.vEmployeeList (ID,Last,First,Job)
AS
SELECT P.BusinessEntityID,
P.LastName, P.FirstName, E.JobTitle
FROM Person.Person P
INNER JOIN HumanResources.Employee E
ON P.BusinessEntityID = E.BusinessEntityID
```

Updating through views

- Can insert and update in the view if it is a simple single table view
- Aggregate functions or GROUP BYs in the view will cause the view to be non-updatable.

View with check option

- **with check option** restricts how rows can be modified
  - Inserts attempting to add rows that the view could not see will fail
  - Updates attempting to modify rows so that the view could no longer see them will fail

```
CREATE VIEW   dbo.vEmployeeList
AS
SELECT P.BusinessEntityID, P.Title, P.LastName,
P.FirstName, E.JobTitle
FROM Person.Person P
INNER JOIN HumanResources.Employee E
ON P.BusinessEntityID = E.BusinessEntityID
With check option
```