Kenny R N Wilson

SQL Server

Cheat Sheets

Data Modification

Method	Code	Description
INSERT INTO	`INSERT INTO Person (firstname, secondname) VALUES ('Indiana', 'Jones'), ('Carl', 'Jones')`	Non-standard T-SQL. All rows treated as transaction
SELECT INTO	`SELECT * INTO dbo.Position2 FROM Position`	Non-standard statement that creates a new table if it does not exist.
INSERT SELECT	`INSERT INTO Position2 SELECT * FROM Position`	Uses a select to define the rows to insert into a table.

INSERT EXEC	`CREATE PROC GetPositions AS SELECT * From Position GO`
	
	`INSERT INTO Position2 EXEC GetPositions`
		Insert from a stored procedure.

BULK INSERT		Insert from a file
Create table on fly.	`SELECT * FROM ( VALUES (1, 1), (2, 4) ) AS SQUARES (X, [X^2])`	Non-standard T-SQL

Select

PHASES

```
❺ SELECT empid, YEAR(orderdate) AS orderyear, COUNT(*) AS numorders
❶ FROM Sales.Orders
❷ WHERE custid = 71
❸ GROUP BY empid, YEAR(orderdate)
❹ HAVING COUNT(*) > 1
❻ ORDER BY empid, orderyear;
```

❶ **FROM**	Specify the table we want to query
❷ **WHERE**	Uses a predicate to filter the rows returned by the FROM phrase. Where clauses enable the use of indices to improve performance and reduce the network traffic that would occur if we performed a table scan and filtered on the client.
❸ **GROUPBY**	Produce a group for each unique combination of values specified in this clause.
❹ **HAVING**	Uses a predicate to filter the groups returned. Can utilise aggregate functions in the predicate.
❺ **SELECT**	Specify the columns we want to see in the result
❻ **ORDER BY**	Sort the rows for presentation purposes

GROUP BY

If a query contains a group by phase any subsequent HAVING, SELECT, and ORDERBY clauses work on groups. As such they can only operate on expressions that return a single scalar value per group. Any fields specified in the GROUP BY phase implicitly have this process. Any elements that do not meet this restriction can only be used as inputs to aggregation functions such as COUNT, SUM, AVG, MIN, MAX.

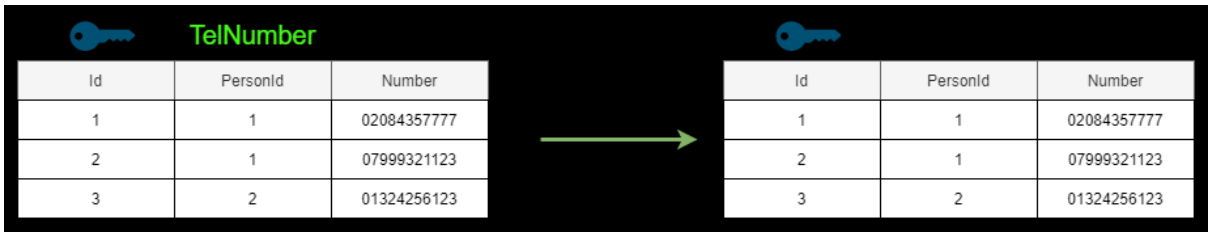
ORDER BY

In SQL, a table has no order. By using order by the result is ordered and hence cannot be considered a table. In SQL the ordered rows are referred to as a cursor. Unlike all other phases the order by phase can utilise column alisases defined in the select phase as the order by is the only phase that follows the select phase. Returns a cursor rather than table.

PHASE ILLUSTRATIONS

SELECT

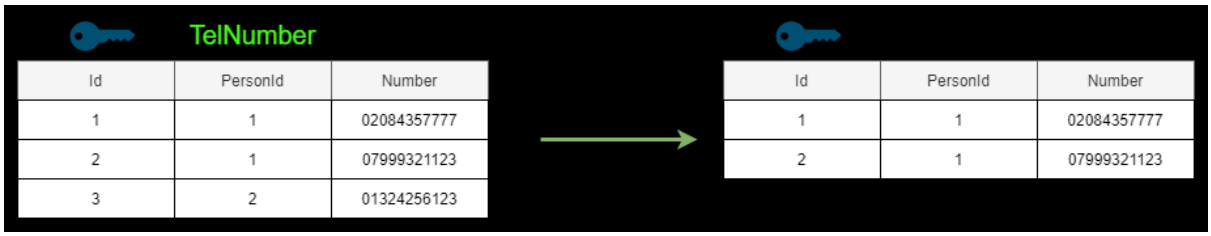
❷ `SELECT *`
❶ `FROM TelNumber`



WHERE

The Where phrase adds a predicate to filter the results.

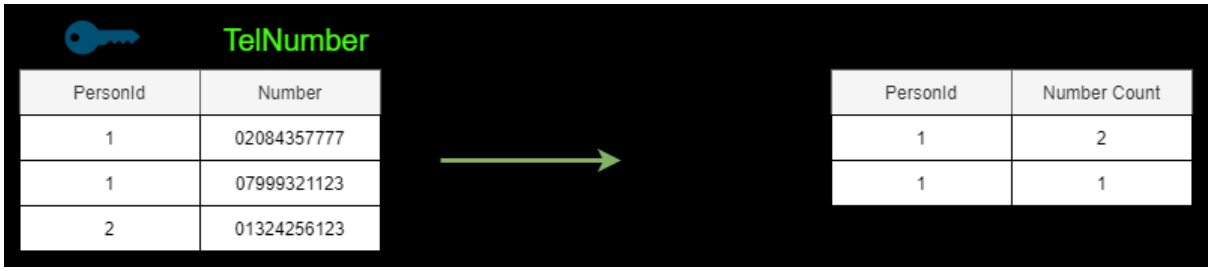
❸ `Select *`
❶ `FROM TelNumber`
❷ `WHERE PersonId = 1`



GROUP BY

Allows grouping. The select can only work on columns appropriate to the grouping.

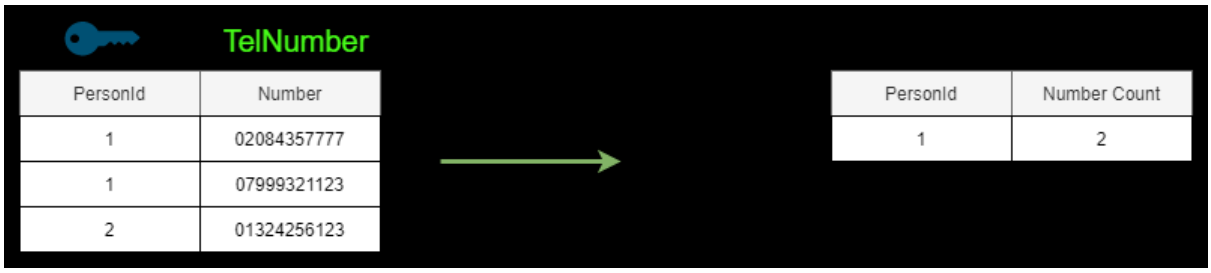
❹ `Select PersonId, COUNT(Number) AS 'Number Count'`
❶ `FROM TelNumber`
❷ `WHERE PersonId IN (1,2)`
❸ `GROUP BY PersonId`



HAVING

The having Phase filters the results of the Group By phase

```
❺ Select PersonId, COUNT(Number) AS 'Number Count'
❶ FROM TelNumber
❷ WHERE PersonId IN (1,2)
❸ GROUP BY PersonId
❹ HAVING count(Number) > 1
```



ORDER BY

Order By returns a cursor rather than a table.

```
❺SELECT PersonId, COUNT(Number) AS 'Number Count'
❶FROM dbo.TelNumber
❷WHERE PersonId in (1,2)
❸GROUP BY PersonId
❹Having Count(Number) >= 1
❻ORDER BY [Number Count] DESC
```

Usage Notes

- It is the only Phase that can access aliases from the SELECT Phase.
- We can order by attributes not in the select phase if we do not use DISTINCT.
- If we use DISTINCT, the ORDER BY elements must be in the SELECT phase.

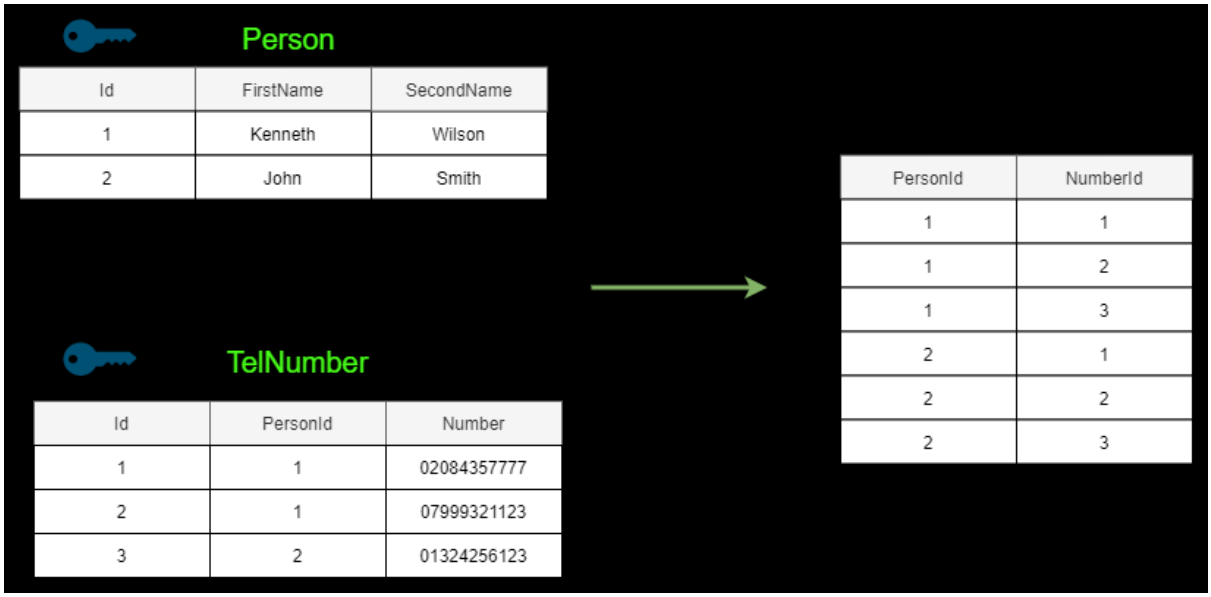
The reason being is that with DISTINCT there are multiple rows for each distinct value and it is not clear which one to use for the ordering.

Joins

Cross Join

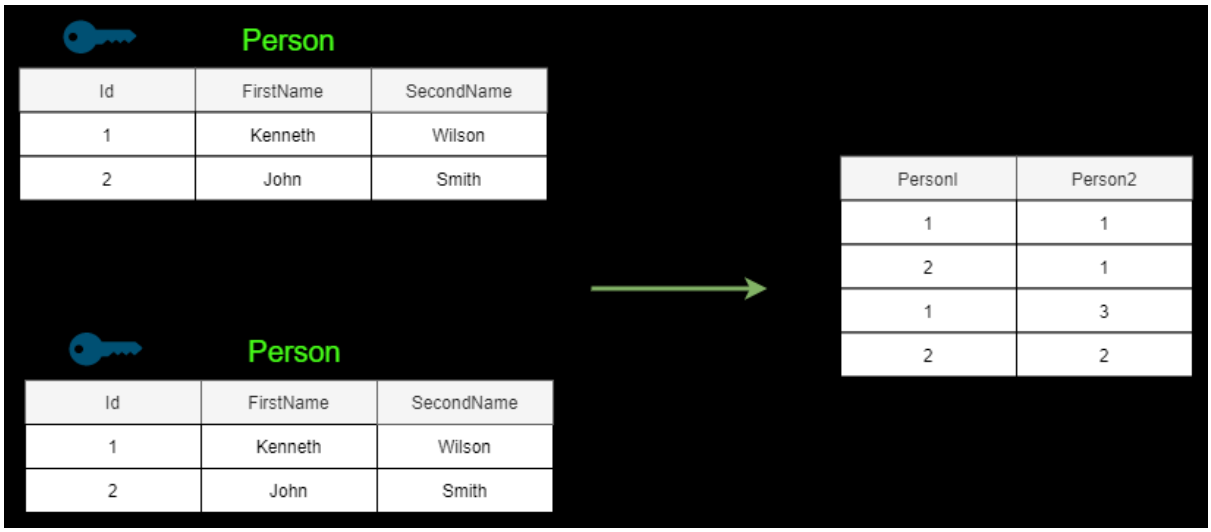
A cross join is a simple cartesian product.

```
SELECT P.id As 'PersonId', T.id AS 'NumberId'
FROM Person P
    CROSS JOIN TelNumber T
```



SELF-CROSS JOIN

```
SELECT  P1.id AS 'Person1', P2.id AS 'Person2'
FROM Person P1
    CROSS JOIN Person P2
```

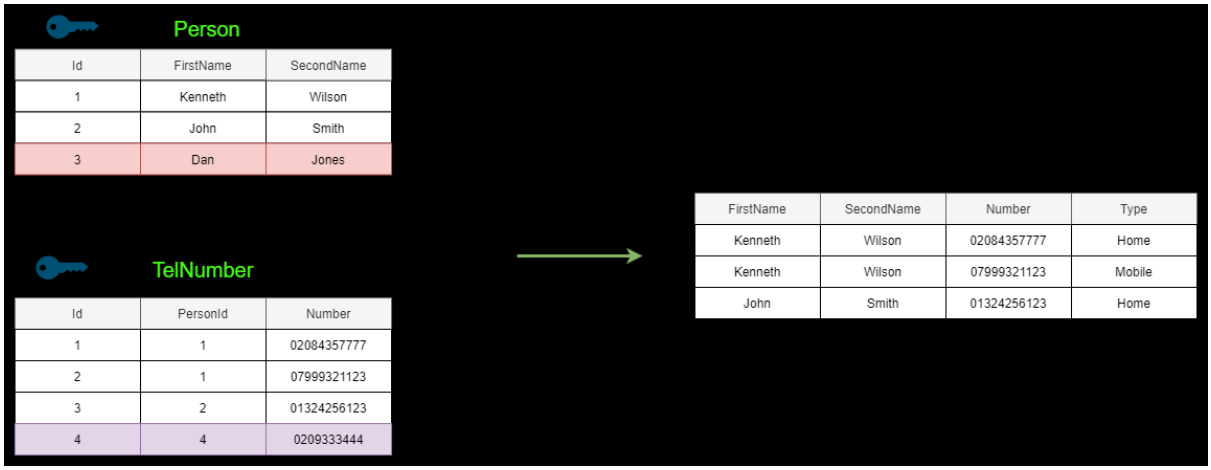


INNER JOIN

An inner join is implemented in two logical phases; a cartesian product followed by a filtration based on a predicate specified in the ON clause.

```
SELECT P.FirstName, P.SecondName, T.Number
FROM Person P
    INNER JOIN TelNumber T
          ON P.Id = T.PersonId
```

Note that the third row in the `Person` table is excluded from the result because it does not have any matching rows in the `TelNumber` table. Similarly the fourth row in the TelNumber table is excluded from the result as it has no match in the Person table. This the defining feature of an inner join.



Composite Join

A composite join is just a join where the AS clause has multiple attributes e.g., something like

```
SELECT
    P.firstName AS 'First Name',
    P.secondName AS 'Second Name',
    T.telNumber AS 'Num'
FROM Person AS P
INNER JOIN TelNumber AS T
    ON P.secondName = T.secondName
    ANF P.firstName = T.firstName
```

NON EQUI-JOIN (JOIN CONDITION HAS ANY OPERATOR OTHER THAN EQUALITY)

```
SELECT
    P1.firstName + ' ' + P1.secondName AS 'Person1',
    P2.firstName + ' ' + P2.secondName AS 'Person2'
FROM Person AS P1
    INNER JOIN Person AS P2
          ON P1.id < P2.id
```

Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

Person

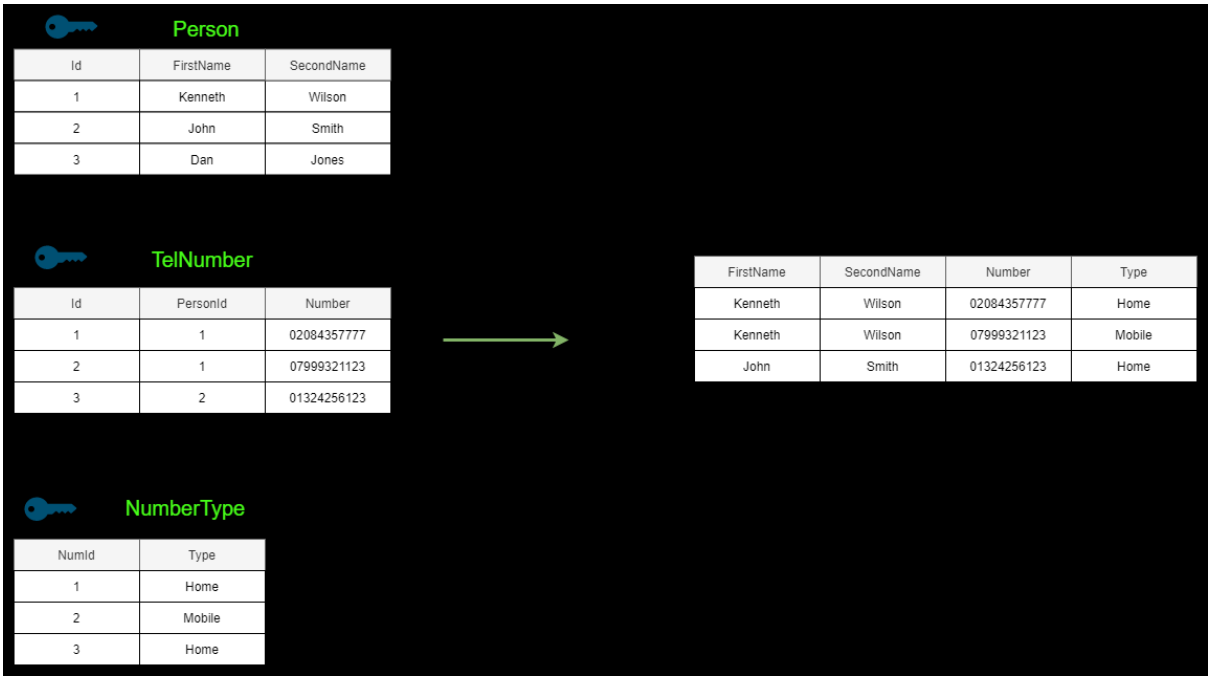
Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

Person1	Person2
Kenneth Wilson	John Smith
Kenneth Wilson	Kelly Clarkson
John Smith	Kelly Clarkson

MULTI-JOIN

Joining more than one table. Logically multi joins proceed from left to right with the result of the first table operator becoming the left input to the second table operator and so on.

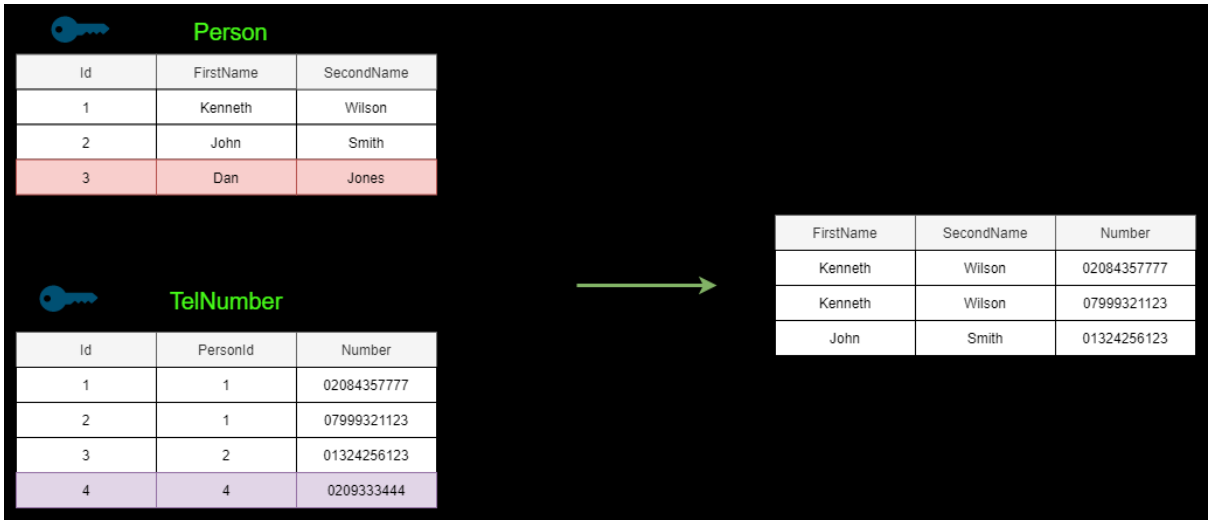
```
SELECT P.FirstName, P.SecondName, T.Number, NT.[Type]
FROM Person P
     INNER JOIN TelNumber T
           ON P.id = T.personId
     INNER JOIN NumberType NT
           ON T.id = NT.NumId
```



Outer Join

An inner join leaves out any rows that do not match.

```
SELECT P.FirstName, P.SecondName, T.Number
FROM Person P
    INNER JOIN TelNumber T
          ON P.Id = T.PersonId
```

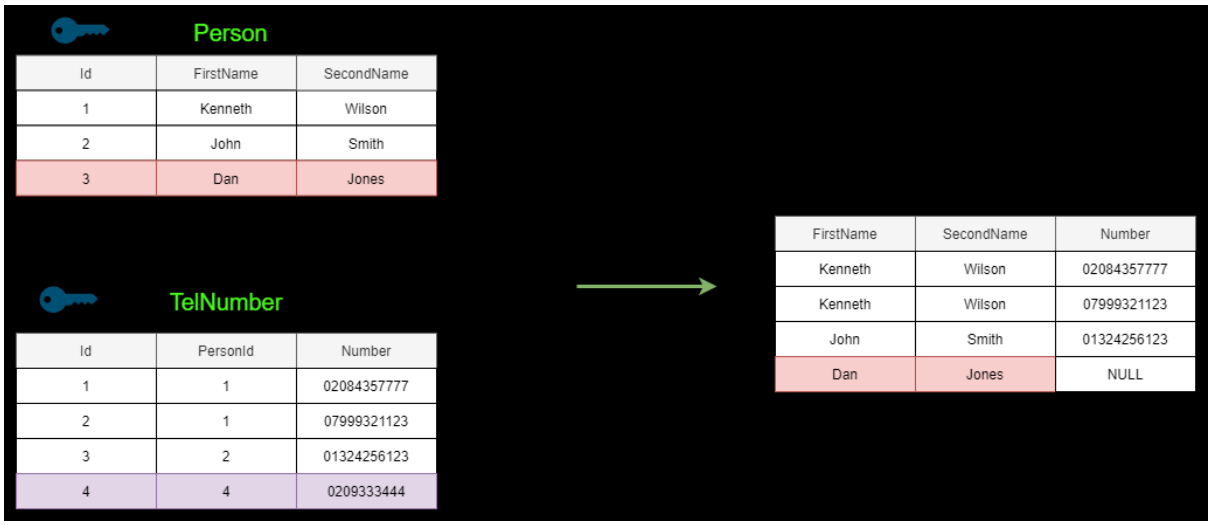


Outer joins enable us to fix this problem.

LEFT OUTER JOIN

If we want to add back in row 3 in the Person table, we can use a LEFT OUTER JOIN

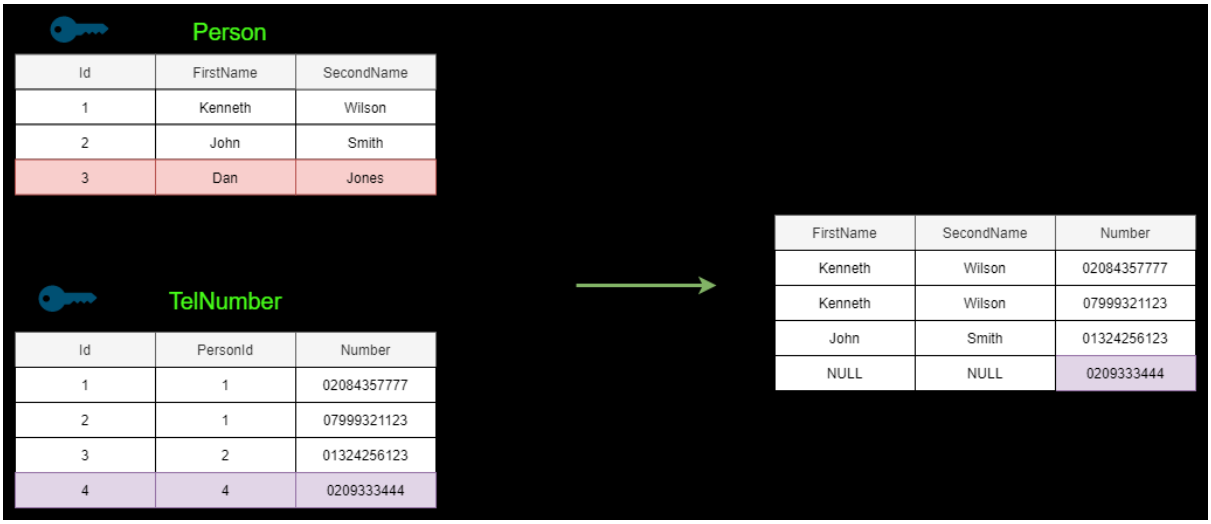
```
SELECT P.FirstName, P.SecondName, T.Number
FROM Person P
  LEFT OUTER JOIN TelNumber T
      ON P.Id = T.PersonId
```



RIGH OUTER JOIN

If we want to add back in row 4 in the TelNumber table, we can use a RIGHT OUTER JOIN.

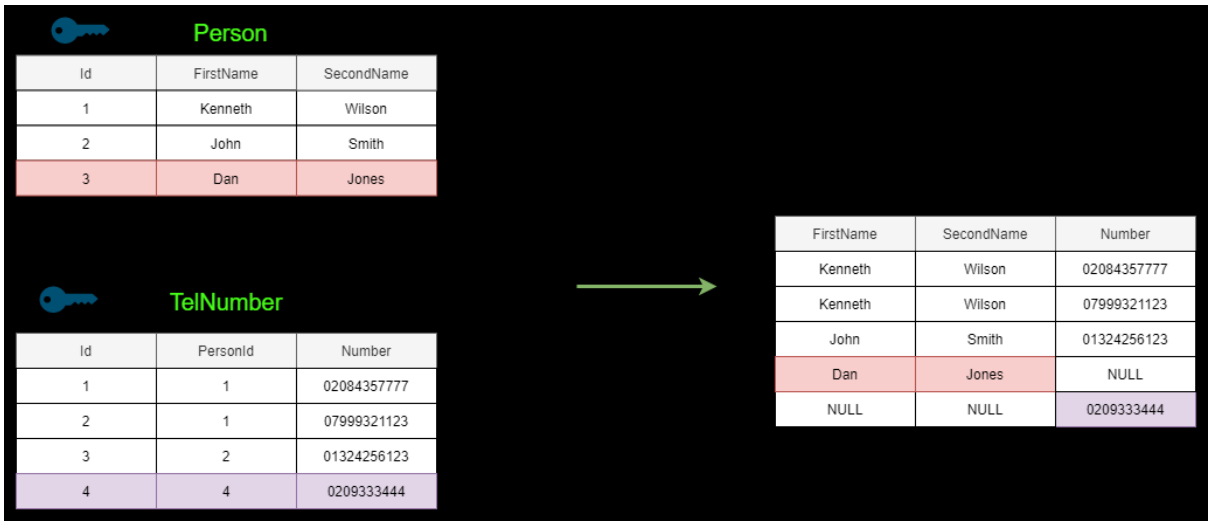
```
SELECT
    P.firstName + ' ' + P.secondName AS 'Person',
    T.telNumber AS 'Number'
FROM Person AS P
    LEFT OUTER JOIN TelNumber AS T
          On P.id = T.personId
```



FULL OUTER JOIN

If we want to add back in row 4 in the TelNumber table row and row 3 in the Person table, we can use a FULL OUTER JOIN

```
SELECT P.FirstName, P.SecondName, T.Number
FROM Person P
    FULL OUTER JOIN TelNumber T
          ON P.Id = T.PersonId
```

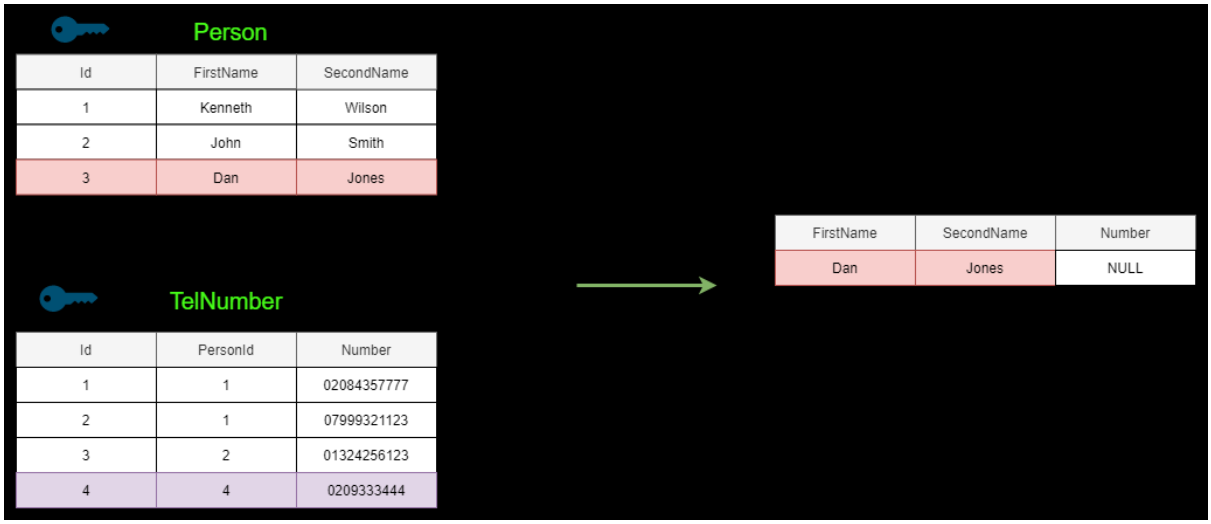


OUTER JOIN ON VERSUS WHERE

The ON clause determines which rows match between the two tables. The WHERE clause filters the matched rows.

SELECT ONLY OUTER ROWS

The following shows how to only return outer rows.




```
SELECT P.FirstName, P.SecondName, T.Number
FROM Person P
  LEFT OUTER JOIN TelNumber T
      ON P.Id = T.PersonId
WHERE T.number IS NULL
```

OUTER JOIN BUGS

In general, we should never refer to columns in the non-preserved side of an outer join from the WHERE clause. This will remove any rows without a match in the non-preserved side effectively changing the join to an INNER JOIN.

Similarly if we have a multiple table join where we first perform an outer join between two tables and then perform an inner join between a third table and a field on the non-preserved side of the outer join then any outer rows will be discarded.

Normalization

Forms

1st Normal Form

For a table to be in first normal form each cell must contain only one value from the domain. The following table violates this restriction

Id	First Name	Second Name	Numbers
1	Kenneth	Wilson	02084357777, 07999321123
2	John	Smith	01324256123, 07999321145

We should note the following table is still in violation of 1st normal form as repeating columns groups are also disallowed.

Id	First Name	Second Name	Number 1	Number 2
1	Kenneth	Wilson	02084357777	07999321123
2	John	Smith	01324256123	07999321145

We can fix this using a schema something like the following. The PersonId field acts as the foreign key that indexes into the person table

Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145

The second restriction on 1st normal form is that every row is unique. We can ensure uniqueness of rows by applying a candidate key to the row.

Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145

2nd Normal Form

Second normal for applies to relations with composite keys. Where there is a composite key, we should not be able to locate the value of any non-key attribute using only part of the composite key. The following table is in violation of 2nd normal form.

Make	Model	Manufacturer Country
Ford	Fiesta	US
Ford	Focus	US

We can fix this as follows

Make	Model
Ford	Fiesta
Ford	Focus

Make	Manufacturer Country
Ford	US

A relation with a single attribute primary key in 1st normal form is automatically in 2nd normal form.

3rd Normal Form

To be in third normal form the relation must first be in second normal form. The second rule is that no non-key attribute can be identified by another non-key attribute. This table is in violation of 3rd normal form

PosId	ProductId	ProductType
1	2	Derivative
2	2	Derivative

We can fix this as follows.

PosId	ProductId
1	2
2	2

ProductId	ProductType
2	Derivative
2	Derivative

We can summarise the second and third forms as meaning than in order to identify the value of any non-key attribute we need to use the full primary key. Furthermore, we cannot identify the value of any non-key field using another non-key field.

Variables

Method	Code	Description
Declare	`DECLARE @x AS INT;`	
Set Value	`SET @x = 10`	
Declare and Initialize	`DECLARE @x AS INT = 10;`	

Set value from scalar sub query	`DECLARE @x AS INT;`
	`SET @x = (SELECT VAL`
	`FROM Position`
	`WHERE PositionId = 1`
	`);`	

Set multiple variables from select	`DECLARE @x AS INT;`
	`DECLARE @y AS INT;`
	
	`SELECT`
	`@x = [Value],`
	`@y = ClientId`
	`FROM Position`
	`WHERE PositionId = 1`
		Non-standard T-SQL extension

Control Flow

IF...ELSE

```
DECLARE @x AS INT = 4;

IF @x = 5
BEGIN
  PRINT 'Value is 5';
  PRINT 'A Second Statement';
END;
ELSE
  IF @x = 4
      BEGIN
            PRINT 'Value is 4';
                  RINT 'A Second Statement';
      END;
  ELSE
      BEGIN
            PRINT 'Value is something else';
            PRINT 'A Second Statement';
      END;
```

WHILE

```
DECLARE @x AS INT = 3;
WHILE @x > 0
BEGIN
  PRINT @x
  SET @x = @x -1
END
```

CURSOR

```
DECLARE @posId As INT;

DECLARE MyCursor CURSOR FAST_FORWARD  FOR
  SELECT PositionId
  FROM Position;

OPEN MyCursor;

FETCH NEXT FROM MyCursor INTO @posId;

WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT @posId
  FETCH NEXT FROM MyCursor INTO @posId
END;

CLOSE MyCursor;
yg
DEALLOCATE MyCursor;
```

Predicates

Predicates can be combined using the AND and OR operators. A predicate is an expression that is either true or false. Predicates can be used to

1. Enforce data integrity.
2. Filter data into subsets.
3. Specify sets by their properties rather than explicit enumeration of elements

Predicate	Example	Description
IN	`SELECT * from Products WHERE prodId IN (1, 2, 3)`	Check if an expression is equal to at least one element in the set.
BETWEEN	`SELECT * from Products WHERE prodId BETWEEN 1 AND 3`	Check if a value is in a specified range
LIKE	`SELECT * from Products WHERE description LIKE 'E%'`	Check if a character string conforms to a patter

Date and Time

Type	Description
DATETIME	Legacy
SMALLDATETIME	Legacy
DATE	3 bytes representing range from 00010101 to 99991231
TIME	3 to 5 bytes giving accuracy to 100 nanoseconds
DATETIME2	6 to 8 bytes giving date and time accuracy to same level as DATE and TIME combined
DATETIMEOFFSET	8 to 10 bytes. Like DATETIME2 but includes offset from UTC

> **LEGACY DATETIME TYPE**
>
> When dealing with the legacy type DATETIME the convention is that we use a time of midnight if we want to only use the date part and a date of January 1st 1900 if we are only interested in the time part.

FUNCTIONS

Function	Returns	Description
GETDATE	DATETIME	Gets current date and time
GETUTCDATE	DATETIME	Get current date and time in UTC
SYSDATETIMEOFFSET	DATETIMEOFFSET	Get current date and time with UTC offset
SWITCHOFFSET		Switches to a different UTC Offset (Timezone)
DATEADD		Add years, months or days to a
DATEDIFF		Give different between dates in some date part (year, month, day etc)
DATEPART		Get year, month, day etc
YEAR,MONTH,DAY		Abbreviations for DATEPART

As none of the above functions return only the date or only the time, we need to do a little extra to get these.

```
SELECT
    CAST(SYSDATETIME() AS DATE) AS [Date],
    CAST(SYSDATETIME() AS TIME) AS [Time]
```

Date	Time
27/04/2020 00:00:00	20:29:54.8753964

ADDING DATES

```
insert Trades values(2,'20200103')
select tradeId AS 'Trade Id', tradeDate AS 'Trade Date' from Trades
```

Trade Id	Trade Date
2	03/01/2020 00:00:00

INEFFICIENT OF MANIPULATION COLUMNS IN FILTERS

We need to careful when applying filters such as in WHERE phases. If we manipulate the filtered column this can prevent the database server using the index in an efficient manner. The following is an inefficient query.

```
SELECT tradeId AS 'Trade Id', tradeDate AS 'Trade Date'
FROM Trades
WHERE YEAR(tradeDate) = 2020
```

We can improve our query using a range filter as follows which enables the efficient use on indices.

```
SELECT tradeId AS 'Trade Id', tradeDate AS 'Trade Date'
FROM Trades
WHERE tradeDate >= '20200101' AND tradeDate < '20210101'
```

Strings

Characters can be either Unicode or regular.

Type	Description
CHAR	Fixed length, 1 byte per character
VARCHAR	
NCHAR	Fixed length Unicode, 2 bytes per character
NVARCHAR	

STRING FUNCTION

Operation	Example	Description
+	`SELECT 'Hello' + 'World'`	String concatenation operator
COALESCE	`DECLARE @str AS VARCHAR(10) = NULL;` `SELECT COALESCE(@STR, 'Empty')`	Return first non-null element in list
SUBSTRING	`SELECT` `SUBSTRING('Hello', 1,3)`	Result is Hel
LEFT	`SELECT LEFT('Hello', 2)`	Shorthand form of SUBSTRING. Result is He
RIGHT	`SELECT Right('Hello', 2)`	Shorthand form of SUBSTRING. Result is lo
LEN	`SELECT LEN('Hello')`	Return number of characters. In this case 5
CHARINDEX	`SELECT CHARINDEX('l','Hello World',5)`	Return the index of first occurrence of substring starting it specified index. In this case result is 10
PATINDEX	`SELECT PATINDEX('%[0-9]%','AA5AA')`	Return first index of pattern. In this case index 3.
REPLACE	`SELECT REPLACE('Boy Meets Boy','Boy', 'Girl')`	'Girl Meets Girl'
REPLICATE	`SELECT REPLICATE('Boy', 3)`	'BoyBoyBoy;
STUFF	`SELECT STUFF('Hello', 1,1,'Boy')`	Remove substring and insert substring
RTRIM/LTRIM		Remove trailing/leading spaces.
FORMAT	`SELECT FORMAT(1234, '0000000');`	Format value based on .NET format string. Result is 0001234
STRING_SPLIT	`SELECT * FROM` `STRING_SPLIT('1,2,3,4',',') AS S`	1 2 3 4

Operators

Equality and Ordinality

Operator	Description
=	Equality
>	
<	
>=	
<=	
<>	

Logical

Operator	Description
AND	and
OR	
NOT	

Arithmetic

Operator	Description
+	
-	
*	
/	
%	Modulo

If two arguments are of the same type the result is of the same type. So, 7 / 2 = 3. We might want to perform a cast in this instance as follows.

```
SELECT CAST(7 as Numeric(12,2))/ CAST(3 as Numeric(12,2)) as num
```

Examples

Basic Select

The following queries work on the below Person table.

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

Name	Code	
Select	`SELECT *` `FROM Person`	id | firstName | secondName 1 | Kenneth | Wilson 2 | John | Smith 3 | Kelly | Clarkson
Select Named Columns	`SELECT P.firstName` `FROM Person P`	firstName Kenneth John Kelly
Select AS	`SELECT P.firstName AS "First"` `FROM Person P`	First Kenneth John Kelly

WHERE (FILTRATION)

The following queries work on the below Person table.

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson
3	John	Wilson

Name	Code	

Equality

```
SELECT firstName
FROM Person P
WHERE firstName = 'John'
```

firstName
John
John

Inequality

```
SELECT *
FROM Person
WHERE firstName <> 'John'
```

id	firstName	secondName
1	Kenneth	Wilson
3	Kelly	Clarkson

Like

```
SELECT *
FROM Person
WHERE firstName LIKE '%enn%'
```

id	firstName	secondName
1	Kenneth	Wilson

IN

```
SELECT *
FROM Person
WHERE firstName
    IN ('Kenneth', 'John')
```

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	John	Wilson

NOT

```
SELECT *
FROM Person
WHERE firstName
  NOT IN ('Kenneth', 'John')
```

id	firstName	secondName
3	Kelly	Clarkson

OR

```
SELECT *
FROM Person
WHERE firstName = 'John'
    OR firstName = 'Kelly'
```

id	firstName	secondName
2	John	Smith
3	Kelly	Clarkson
3	John	Wilson

AND

```
SELECT *
FROM Person
WHERE firstName = 'John'
    AND secondName = 'Smith'
```

id	firstName	secondName
2	John	Smith

The following queries work on the below Products table.

productId	description
1	European Call
2	Variance Swap

Name	Code	

Less than	`SELECT *`
	`FROM Product`
	`WHERE productId < 2`
		productId: 1, description: European Call

Less than or equal to	`SELECT *`
	`FROM Product`
	`WHERE productId <= 2`
		productId: 1, description: European Call
		productId: 2, description: Variance Swap

Greater than	`SELECT *`
	`FROM Product`
	`WHERE productId > 1`
		productId: 2, description: Variance Swap

Transactions

A Transaction is implicitly started using BEGIN TRAN(SACTION)

A Transaction is committed using COMMIT TRAN(SACTION)

A Transaction is rolled back using ROLLBACK TRAN(SACTION

Concurrency is handled differently in an In Memory OLTP Database

Disk based box SQL Server uses locks as default concurrency control

Locks are either exclusive or shared.

A transaction that modifies data obtains exclusive locks on any resources it updates.

No other transaction can obtain an exclusive lock on a resource while another transaction has an exclusive lock on the same resource.

In SQL Server box product the default ISOLATION level is READ COMMITED.

The disk-based box SQL-Server instance uses locking as the default concurrency control. Locking support two modes: exclusive and shared. When modifying data inside a transaction the transaction obtains an exclusive lock on any resources and holds this lock until the transaction completes or is rolled back. While one transaction holds an exclusive lock on a resource no other transaction can obtain an exclusive lock on the same resource until the first transaction completes. Whether another transaction read from the same object while the first transaction holds an exclusive lock depends on isolation level.

In A SQ

Programmable Objects

A batch is one or more SQL server statements sent to the server for execution as a single group.

Language Cheat Sheet

Three Valued Logic

In SQL predicates can evaluate to `TRUE`, `FALSE` or `UNKNOWN`. If one of the arguments in a logical expression is NULL, then the result is `UNKNOWN`. If a logical expression is used in a query filter, then any result of UNKNOWN leads to a rejection (accept true). If a logical expression used in a check constraint returns UNKNOWN, the value is accepted (reject false)

Subqueries

Subqueries can be single value or multi-valued. They can also be self-contained or correlated.

Self-Contained

```
select * from TelNumber T
where T.PersonId IN
    (     Select MIN(P.Id)
          From Person P
    );
```

Correlated

The following uses correlated subqueries to calculate the percentage of portfolio for each position.

```
Select
    P1.PortfolioId,
    P1.PosId,
    P1.Val,
    100 * Val /   (     SELECT SUM(Val)
                        FROM Positions P2
                        WHERE P1.PortfolioId = P2.PortfolioId
                ) AS '%Val'
FROM Positions P1
```

PosId	PortfolioId	Val
1	1	10.0
2	1	20.0
3	1	30.0
4	2	10.0
5	2	20.0

→

PosId	PortfolioId	Val	% Val
1	1	10.0	16.666
2	1	20.0	33.333
3	1	30.0	50.0
4	2	10.0	33.333
5	2	20.0	66.666

Self-Contained

Return everyone who does not have a telephone number.

```
select P.FirstName, P.SecondName from Person P
where P.Id not in
    (
        select distinct T.PersonId
        from TelNumber T
    )
```

I used distinct here but in general we can expect the database engine to make these kinds of optimisations for us.

Exists

We could re-write the previous query as follows.

```
select P.FirstName, P.SecondName from Person P
where NOT EXISTS
    (
        select *
        from TelNumber T
        where T.PersonId = P.Id
    )
```

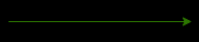
Note that exists has particularly good performance as the database engine can optimise.

Previous value

We can use correlated subqueries to calculate previous values.

```
SELECT PosId, PortfolioId, Val,
(
    SELECT MAX(P2.PosId)
    FROM Positions AS P2
    WHERE P2.PosId < P1.PosId
) As PrevPosId
FROM Positions AS P1
```

PosId	PortfolioId	Val
1	1	10.0
2	1	20.0
3	1	30.0
4	2	10.0
5	2	20.0



PosId	PortfolioId	Val	PrevPosId
1	1	10.0	NULL
2	1	20.0	1
3	1	30.0	2
4	2	10.0	3
5	2	20.0	4

Next value

We can use correlated subqueries to calculate previous values.

```
SELECT PosId, PortfolioId, Val,
(
    SELECT MIN(P2.PosId)
    FROM Positions AS P2
    WHERE P2.PosId > P1.PosId
```

```
) As NextPosId
FROM Positions AS P1
```

PosId	PortfolioId	Val
1	1	10.0
2	1	20.0
3	1	30.0
4	2	10.0
5	2	20.0

PosId	PortfolioId	Val	NextPosId
1	1	10.0	2
2	1	20.0	3
3	1	30.0	4
4	2	10.0	5
5	2	20.0	NULL

RUNNING TOTAL

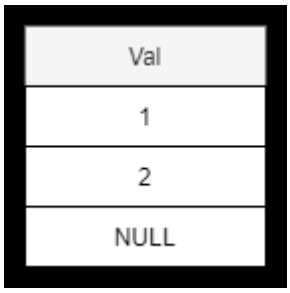
```
SELECT PosId, PortfolioId, Val,
(
    SELECT SUM(P2.VAL)
    FROM Positions AS P2
    WHERE P2.PosId <= P1.PosId
) As RunningTotal
FROM Positions AS P1
```

PosId	PortfolioId	Val
1	1	10.0
2	1	20.0
3	1	30.0
4	2	10.0
5	2	20.0

PosId	PortfolioId	Val	Total
1	1	10.0	10.0
2	1	20.0	30.0
3	1	30.0	60.0
4	2	10.0	70.0
5	2	20.0	90.0

NOT IN BUGS

We need to be careful when using NOT IN with a subquery that might return NULL. Consider the following table.



Val
1
2
NULL

We might expect the following query to return "True" but in fact it returns nothing. This is because the result of the NOT IN is the empty set.

```
SELECT 'True'
WHERE 10  NOT IN
(
    SELECT VAL FROM NUMS
    WHERE VAL IS NOT NULL
)
```

Effectively the NOT IN evaluated to

```
10 NOT IN (1,2,NULL)
```

Which is

```
NOT (10=1 OR 10 = 2 OR 10 =NULL)
```

Which is

```
NOT (FALSE OR FALSE OR UNKNOWN)
```

Which is

```
NOT (UNKNWON)
```

Which is

```
UNKNOWN
```

To fix the bug we need to exclude NULL from the subquery.

Table Expressions

Table expressions are expressions whose values are relational tables. Any manipulations that expect a table can work with table expressions. There are four types

- Derived Tables
- Common Table Expressions (CTEs)
- Views
- Inline Table Value Functions

It is worth noted that table expressions are virtual and are hence used to improve readability of code. They are unnested by the database engine.

Derived Tables

Derived tables or table subqueries are defined in the FROM clause of an outer query.

```
select *
from
    ( SELECT *
      FROM TelNumber
      WHERE PersonId = 1
    ) As PersonOneNumbers
```

Common Table Expressions

```
WITH Person1Numbers AS
(
    SELECT *
    FROM TelNumber T
    WHERE T.PersonId = 1
)
SELECT * FROM Person1Numbers;
```

Aliases can be specified inline.

```
WITH Person1Numbers AS
(
    SELECT T.Id As NumberId,T.Number As Num
    FROM TelNumber T
    WHERE T.PersonId = 1
)
SELECT * FROM Person1Numbers;
```

Or external

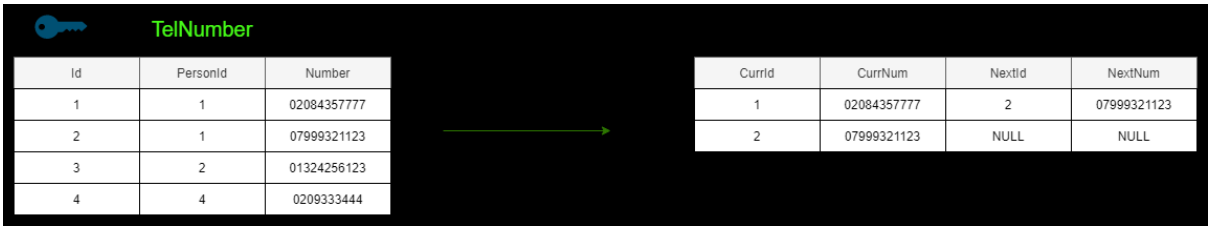
```
WITH Person1Numbers(NumberId, Num) AS
(
    SELECT T.Id,T.Number
    FROM TelNumber T
    WHERE T.PersonId = 1
)
SELECT * FROM Person1Numbers;
```

We can refer to one CTE from another. In such case we use a comma

```
WITH Person1Numbers(NumberId, Num) AS
(
    SELECT T.Id,T.Number
    FROM TelNumber T
    WHERE T.PersonId = 1
),
Person1FirstNumber(NumberId, Num) AS
(
    SELECT P.NumberId, P.Num
    FROM Person1Numbers P
    WHERE P.NumberId = 1
)
SELECT * FROM Person1FirstNumber;
```

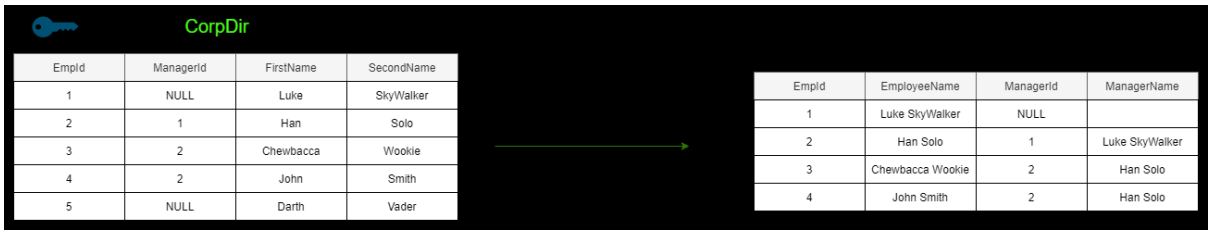
Because a CTE is named and then used we can refer to the same CTE from multiple places in the outer query

```
WITH Person1Numbers(NumberId, Num) AS
(
    SELECT T.Id,T.Number
    FROM TelNumber T
    WHERE T.PersonId = 1
)
SELECT P1.NumberId AS CurrId, P1.Num AS CurrNum,P2.NumberId AS NexId,
P2.Num AS NextNum  FROM Person1Numbers AS P1
    LEFT OUTER JOIN Person1Numbers P2
          ON P1.NumberId = P2.NumberId - 1
```



RECURSION

CTEs also support recursion. This is an advanced technique and one which is quite hard to understand. Consider an example.




```
WITH Hierarchy AS
(
    --The terminating case
    SELECT
                EmpId,
                ManagerId,
                FirstName,
                SecondName,
                CAST('' AS VARCHAR(30)) ManagerName
    FROM
                CorpDir
    WHERE
                EmpId = 1

    UNION ALL

    SELECT
                Subordinates.EmpId,
                Subordinates.ManagerId,
                Subordinates.FirstName,
                Subordinates.SecondName,
                CAST( Superiors.FirstName + ' ' + Superiors.SecondName AS
VARCHAR(30)) AS ManagerName
    FROM
                Hierarchy AS  Superiors
                      INNER JOIN CorpDir AS Subordinates
                            ON Subordinates.ManagerId = Superiors.EmpId
)
select EmpId, FirstName + ' ' + SecondName AS EmployeeName,ManagerId,
ManagerName
FROM Hierarchy
```

Views

Whereas derived tables are scoped to a single expression, views are stored as objects in the database meaning they can be reused. As an object we can manage access to the view in the same was as other database objects.

One point of note is that a View is compiled. So if we use SELECT * in a view definition it will only include columns that exist in the table at the point it is compiled. Any subsequent table alterations will not be included. The `sp_refreshview` procedure can be used to refresh a view in such cases.

Schema Binding

We can bind a view's schema to the schemas of referenced objects. This mean those object schemas cannot be altered or deleted.

Check Option

It is possible to make edits through a view. These edits might not conform to the filter in the view. If we want to prevent this we need to use the `WITH CHECK OPTION` when we create the view.

Inline Table Valued Functions

Inline TVF are like views that accept parameters.

```
CREATE Function dbo.GetDirectReports
    (@mgrId AS INT) RETURNS TABLE
AS
RETURN
    SELECT *
    FROM CorpDir C
    WHERE C.ManagerId = @mgrId
GO
```

We call the Inline TVF as follows.

```
SELECT * FROM GetDirectReports(1)
```

Apply

Apply is a non-standard operator.

CROSS APPLY

The CROSS APPLY operator works on two input tables. Typically, the RHS is a derived table or TVF. A cross apply statement looks very much like a CROSS JOIN, but it is actually different. With CROSS APPLY the left side is evaluated first. Then the rhs is evaluated per row from the left. In this way the rhs can refer to elements of the lhs.

Set Operations

Set operators work on two query result sets. The queries cannot have ORDER BY although this can be added to the set operator result. The LFS and RHS queries must have the same number of columns and data types.

UNION (DISTINCT)

UNION ALL

INTERSECT

INTERSECT (DISTINCT)

EXCEPT

EXCEPT (DISTINCT)

Window Functions

To highlight how window functions work, we will work on the following simple table.

PositionId	ClientId	VALUE
1	1	10.0000
2	1	10.0000
3	1	10.0000
4	2	10.0000
5	2	10.0000

The following query shows a quite simple Window Function against this table.

```
SELECT P.ClientId, P.PositionId, P.VALUE,
  SUM(VALUE) OVER(PARTITION BY P.ClientId
                                ORDER BY P.PositionId
                                ROWS BETWEEN UNBOUNDED PRECEDING
                                AND CURRENT ROW) As Total
FROM Position P
```

The window is specified by the OVER clause which consists of three parts. First the PARTITION BY clause restricts the window to a subset of positions whose values in the partitioning columns match the current row. In our case we restrict the window to the rows that have the same ClientId as the current row (`PARTITION BY P.ClientId`). The ORDER BY clause gives meaning to rank (`ORDER BY P.PositionId`). Finally the window frame clause filters a subset of rows from the preceding expression.

ClientId	PositionId	VALUE	Total
1	1	10.0000	10.0000
1	2	10.0000	20.0000
1	3	10.0000	30.0000
2	4	10.0000	10.0000
2	5	10.0000	20.0000

Pivots

Data Integrity Cheat Sheet

Primary key Constraints

A primary key constraint enforces uniqueness of rows and disallows nulls in constraint attributes. A table can have at most one primary key. Behind the scenes SQLServer will use a unique index to efficiently enforce uniqueness.

Unique Constraints

Although a table can have at most one primary key constraint, it can have multiple unique constraints. Like primary key constraints, unique constraints also enforce unique rows. By using unique constraints SQL Server supports having alternative keys on a table. Unlike primary key constraints, unique constraints allow nulls. Internally SQLServer implements unique constraints using unique indices.

Unique Index

Kenny R N Wilson

Data Integrity Cheat Sheet

The Details

Single Table Queries

Overview

> *An RDBMS is supposed to implement the relational model and provide the means to store, manage, enforce the integrity of, and query data.*
>
> *- T-SQL Fundamentals*

The relational model is based on set theory and predicate logic. SQL is a declarative language in that we describe what we want done and leave the details to the RDBMS. A relation in SQL consists of a heading that specifies the set of attributes (columns) and a body that consists of a set of tuples (rows).

Constraints

RDBMS allows one to model data integrity by specifying constraints. A candidate key is specified on an attribute (column) to enforce uniqueness of tuples (rows). One of the candidate keys is chosen as the primary key and forms the preferred way of uniquely identifying rows. Foreign key is specified on a referencing relation. A foreign key constraint can be used to enforce referential integrity by ensuring only values that exist in the referenced relations are allowed in the referencing relations foreign key.

If we apply key constraints to a table, each element is unique, and it can be considered a set (Otherwise it is a bag or a multiset). Order is unimportant in a set. For this reason, the result of a query has no order unless we explicitly give some sorting criteria.

All SQL commands can be split into three buckets.

	Column Header
Data Definition Language	CREATE , ALTER, DROP
Data Manipulation Language	SELECT, INSERT, UPDATE, DELETE, TRUNCATE, MERGE
Data Control Language	GRANT and REVOKE

Query Language

Kenny R N Wilson

Type

Date and Time

FUNCTIONS

MetaData

Architecture

```
A single SQL server can hold multiple user databases in addition to a set
of system databases (tempdn, model etc)
```

Data Definition

The following creates a table.

```
DROP TABLE IF EXISTS dbo.Products;

CREATE TABLE dbo.Products
(
    prodId INT NOT NULL,
    description VARCHAR(30) NOT NULL
);
```

We can setup declarative data integrity. First, we show how to add a primary key constraint.

PRIMARY KEY CONSTAINT

A primary key has the following properties

- Each table can have only one primary key
- The fields making up the primary key cannot be null
- The server creates an index to efficiently enforce uniqueness and retrieval

```
ALTER TABLE dbo.Products
    ADD CONSTRAINT PK_Products
    PRIMARY KEY(prodId);
```

FOREIGN KEY CONSTRAINT

SQLServer Install

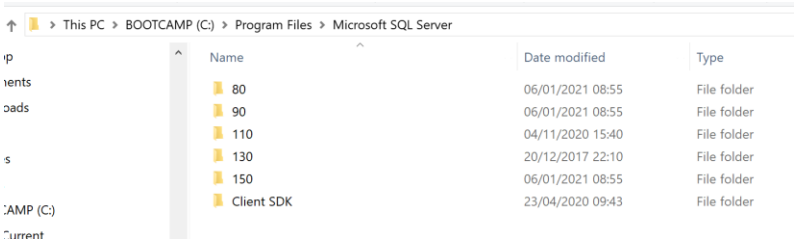
LocalDB

Install

Use the following link and choose the third one from the top

https://www.hanselman.com/blog/download-sql-server-express

The default locations of the installs are `C:\Program Files\Microsoft SQL Server` On my machine I see multiple versions.

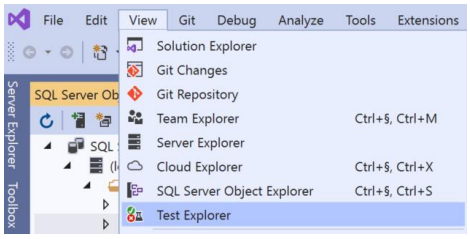


Start/Manage

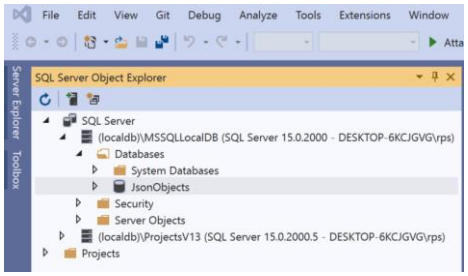
LocalDb supports two kinds on instance: Automatic instances and named instances. The automatic instance for SQLServer is called MSSQLLocalDb.

Visual Studio

The easiest way to manage the local DB instances is to use visual studio.



And when open it looks like this.



Command Prompt

Open a command prompt. We can list all instances of SQLServer using the command.

```
C:\Users\rps>"C:\Program Files\Microsoft SQL
Server\150\Tools\Binn\SqlLocalDB.exe" info

>> MSSQLLocalDB
```

We create a named instance as.

```
C:\Users\rps>"C:\Program Files\Microsoft SQL
Server\150\Tools\Binn\SqlLocalDB.exe" create KennysLocalDb
```

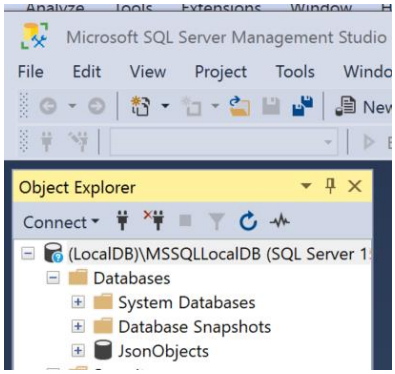
We can view a named install.

```
"C:\Program Files\Microsoft SQL Server\150\Tools\Binn\SqlLocalDB.exe"
info KennysLocalDb

>> Name:                KennysLocalDb
>> Version:             15.0.2000.5
>> Shared name:
>> Owner:               DESKTOP-6KCJGVG\rps
>> Auto-create:         No
>> State:               Stopped
>> Last start time:     19/01/2021 17:47:37
>> Instance pipe name:
```

Microsoft SQL Server Management Studio

If we open management studio it should just automatically connect to the local DB.

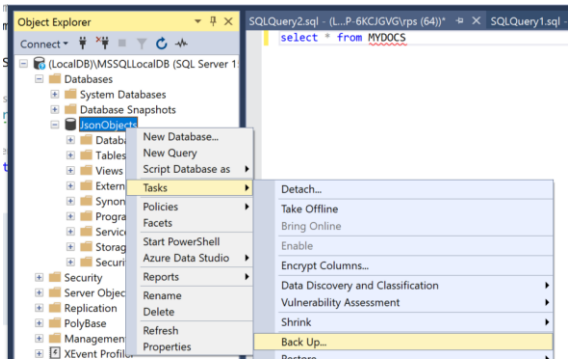


Let us add a JSON table.

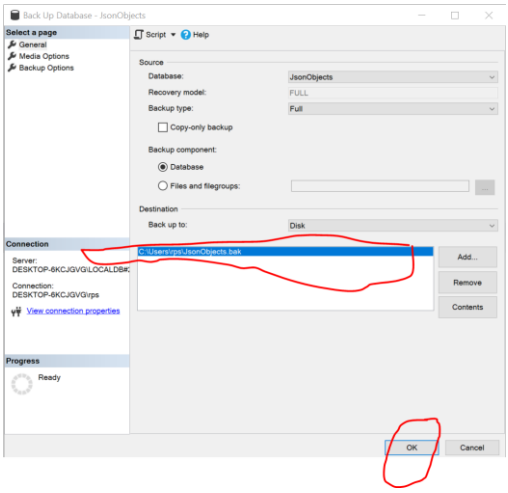
```
create table MYDOCS (
    ID bigint primary key identity,
    DOC nvarchar(max)
);
```

SQLServer Backup/Restore

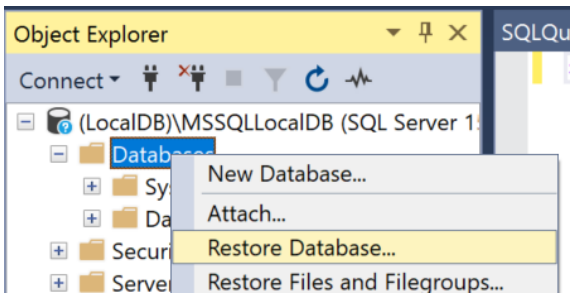
The easiest way to backup and restore our database is to use Microsoft SQL Server Management Studio. Right click the database in the Object Explorer and select as follows.



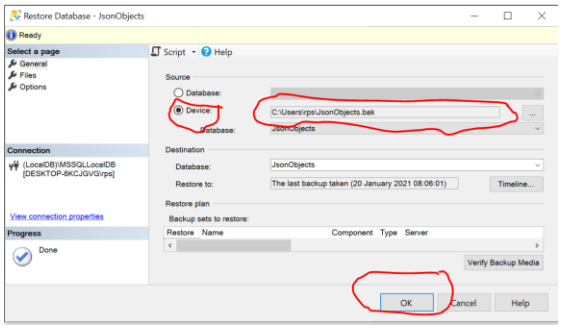
You will then see the following. Just click ok which backs it up to the current user's home directory



We can now safely delete our database safe in the knowledge we can back it up. One deleted restore it as follows. In the Object Explorer right-click on `databases` and select `Restore Database`.



No in the dialogue select Device and enter the location of the file


Restore Database - JsonObjects
Ready
Select a page
General
Files
Options
Script
Help
Source
Database:
Device:
C:\Users\rps\JsonObjects.bak
Database:
JsonObjects
Connection
(LocalDB)\MSSQLLocalDB
[DESKTOP-8KCJGVG\rps]
View connection properties
Progress
Done
Destination
Database:
JsonObjects
Restore to:
The last backup taken (20 January 2021 08:06:01)
Timeline...
Restore plan
Backup sets to restore:
Restore
Name
Component
Type
Server
Verify Backup Media
OK
Cancel
Help

SQLServer Docker

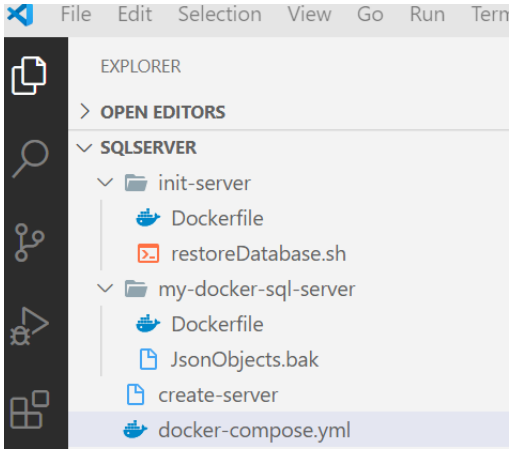
We can run SQLServer 2019 inside docker. The images are here.

Run SQL Server Instance

See

Create and Populate Database.

The following docker logic shows how to get an instance running and copy in a backup file to restore a database and its objects. Note that for the following example we already need backup file taking from somewhere that we can use to do the restore. Our structure looks like this.



Listing 1 my-docker-sql-server/Dockerfile

```
FROM mcr.microsoft.com/mssql/server:2019-latest
USER root
COPY JsonObjects.bak /var/opt/mssql/data/JsonObjects.bak
CMD /opt/mssql/bin/sqlservr
```

Listing 2 init-server/Dockerfile

```
FROM mcr.microsoft.com/mssql-tools
USER root
COPY restoreDatabase.sh /restoreDatabase.sh
CMD "./restoreDatabase.sh"
```

Listing 3 init-server/restoreDatabase.sh

```
echo "sleeping to allow db to start up"
sleep 10
echo 'resoting database JsonObject'
/opt/mssql-tools/bin/sqlcmd -S my-docker-sql-server -U 'SA' -P $SA_PASSWORD -
Q "RESTORE DATABASE [JsonObjects] FROM DISK = N'/var/opt/mssql/data/JsonObjects.bak' WITH R
EPLACE"
```

> **NOTE: FILE ENDING**
>
> For the above shell script make sure the line ending in VS code is set to LF or it will not work.

Listing 4 docker-compose.yml

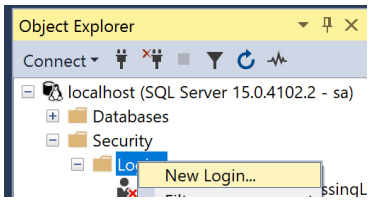
```
version: '3'
services:
  init-server:
    build:
      context: init-server
      dockerfile: Dockerfile
    environment:
      SA_PASSWORD: "Pa!ssWordTwo!"
    links:
      - my-docker-sql-server
  my-docker-sql-server:
    build:
      context: my-docker-sql-server
      dockerfile: Dockerfile
    environment:
      SA_PASSWORD: "Pa!ssWordTwo!"
      ACCEPT_EULA: "Y"
    ports:
      - 1433:1433
```

Logins, Users and Roles

A login is a server level entity and users are database level entities. We can have a login with no user associated. In this case we can log onto the server but not use any of the databases on the server.

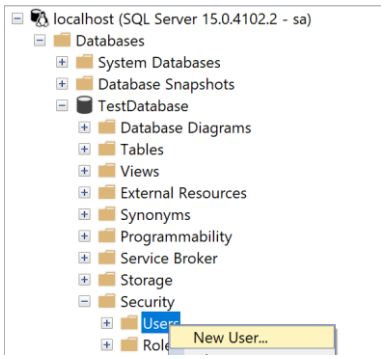
Create Login

On the folder `<ServerName>/Security/Logins` right click and select New Login.



Create User

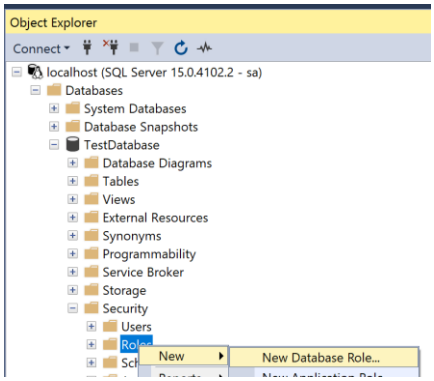
On the folder `Databases/<DatabaseName>/Security/Users` right click and select New User.



On the setup dialogue enter the name and the login we want to associate the user with. You should now be able to log on to the server with the login and then access the database.

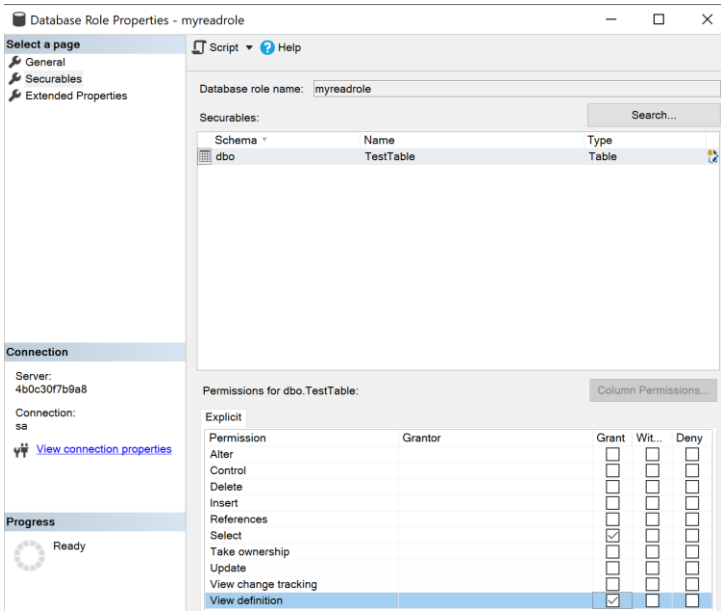
Create Role

To be able to do anything we need to create a role that can read our table. On the folder `Databases/<Database>/Security/Roles` select New Database Role.



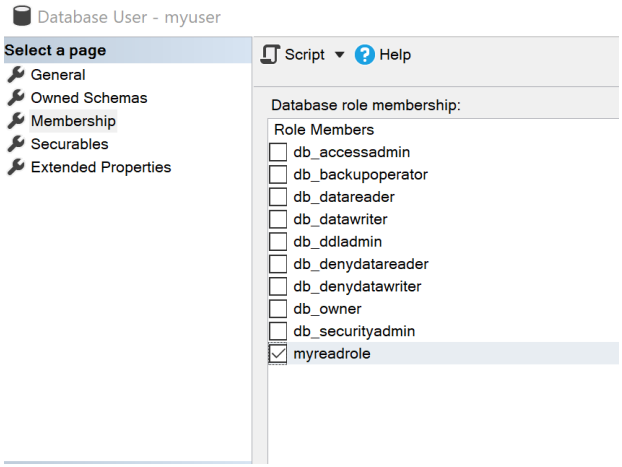
Add permissions to role

Right click on role and select properties and then on the dialogue select Securables.



Associate role with user

Right click on user and select properties then on the Membership page add the new role we created.



We can now read the table.

.NET Core

We can connect to our automatic Local DB from code as follows.

```
SqlConnection connection = new SqlConnection(@"Data
Source=(localdb)\MSSQLLocalDB;Database=JsonObjects;");
connection.Open();
```

We can insert into our table as follows

```
var json = @"{""Hello"":""World2""}";
var sql = $"insert into MYDOCS (DOC) Values('{json}')";
var command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();
```

Questions

SELECT

PHASES

List the phases of a select query.

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

What is the logical order of the phases and what does each do?

1. *FROM select the table we want to query*
2. *WHERE filter the rows returned.*
3. *GROUP BY produce group for each combination.*
4. *HAVING filter the groups.*
5. *SELECT Specify the columns for result*
6. *ORDER BY sort results for presentation*

What is the key benefit of the where clause?

Enable indices to improve performance and reduce network traffic.

Given the following table write a query that first filters out everyone over 40. Then group by age and countryId and show the count for each group. Only include groups with countryId of 1 or 10

firstname	secondname	age	countryId
John	Smith	25	1
Dave	Jones	25	1
Aaro	Litmanen	30	10
Kimi	Raikonen	30	10
Han	Solo	40	2
Luke	Skywalker	35	2
Indiana	Jones	50	3
Carl	Jones	50	3

SELECT age, countryId, COUNT(*)

FROM Person

WHERE age < 50

GROUP BY age, countryId

HAVING countryId IN (1,10)

ORDER BY age, countryId

Basic Select

List the firstname of all rows but rename it “First Name”

SELECT firstname AS 'First Name'

FROM Person

Appendices

Setup Select Database

```
create table dbo.TelNumber
(
    id INT NOT NULL,
    PersonId int NULL,
    Number int NULL
)

INSERT INTO dbo.TelNumber VALUES
(1, 1, 0208435777),
(2, 1, 07999321123),
(3, 2, 01324256123),
(4, 2, 07999321145),
(5, NULL, 02074257777)
```