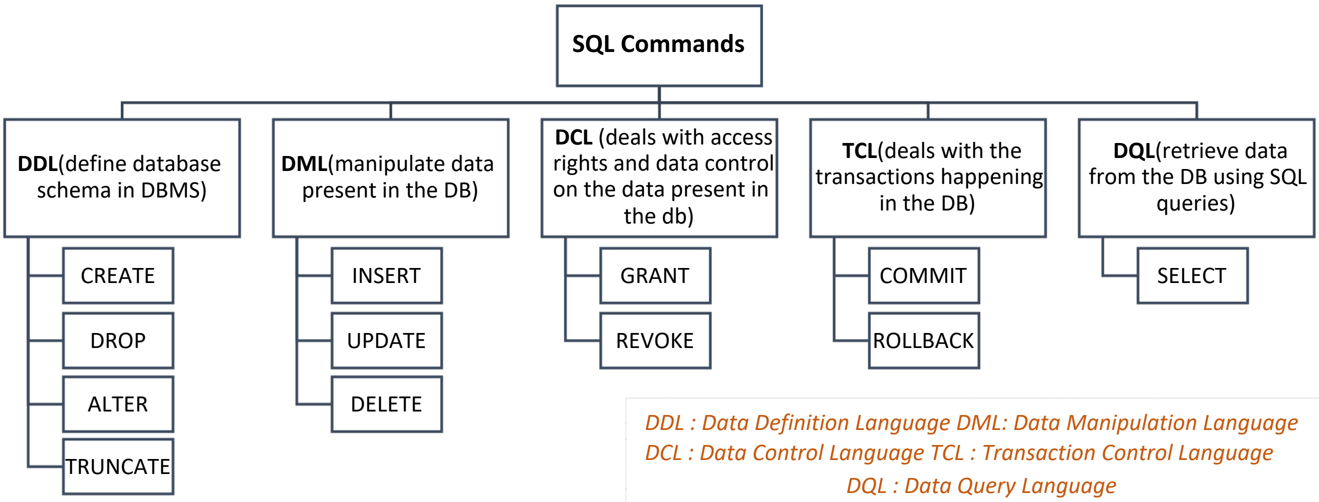
# Structured Query language (SQL)



SQL Commands
DDL(define database schema in DBMS)
DML(manipulate data present in the DB)
DCL (deals with access rights and data control on the data present in the db)
TCL(deals with the transactions happening in the DB)
DQL(retrieve data from the DB using SQL queries)
CREATE
DROP
ALTER
TRUNCATE
INSERT
UPDATE
DELETE
GRANT
REVOKE
COMMIT
ROLLBACK
SELECT

*DDL : Data Definition Language DML: Data Manipulation Language*
*DCL : Data Control Language TCL : Transaction Control Language*
*DQL : Data Query Language*

| | |
|---|---|
| 1. Create database | create database sample2 |
| 2. Use the database | use sample2 |
| 3. Create table | create table customer<br>(<br>customerid int identity(1,1) primary key,<br>customernumber int not null unique check (customernumber>0),<br>lastname varchar(30) not null,<br>firstname varchar(30) not null,<br>areacode int default 71000,<br>address varchar(50),<br>country varchar(50) default 'Malaysia'<br>) |
| 4. Insert values into table | insert into customer values<br>(100,'Fang Ying','Sham','418999','sdadasfdfd',default),<br>(200,'Mei Mei','Tan',default,'adssdsadsd','Thailand'),<br>(300,'Albert','John',default,'dfdsfsdf',default)<br>-- display all records |
| 5. Display record from table | select * from customer<br><br>-- display particular columns<br>select customerid, customernumber, lastname, firstname<br>from customer |
| 6. Add new column to table | alter table customer<br>add phonenumber varchar(20) |
| 7. Add values to newly added column/ Update table | update customer set phonenumber='1234545346' where<br>customerid=1<br>update customer set phonenumber='45554654' where<br>customerid=2 |
| 8. Delete a column | alter table customer<br>drop column phonenumber |
| 9. Delete record from table<br>--if not put 'where', will delete all record | delete<br>from customer<br>where country='Thailand' |
| 10. Delete table | drop table customer |
| 11. Change data type | alter table customer<br>alter column phonenumber varchar(10) |

| | |
|---|---|
| 1. Create database | create database SaleOrder use SaleOrder |
| 2. Use the database | create table dbo.customer ( |
| 3. Create tables | CustomerID int NOT null primary key,<br>CustomerFirstName varchar(50) NOT null,<br>CustomerLastName varchar(50) NOT null,<br>CustomerAddress varchar(50) NOT null,<br>CustomerSuburb varchar(50) null,<br>CustomerCity varchar(50) NOT null,<br>CustomerPostCode char(4) null,<br>CustomerPhoneNumber char(12) null,<br>);<br>create table dbo.inventory (<br>InventoryID tinyint NOT null primary key,<br>InventoryName varchar(50) NOT null,<br>InventoryDescription varchar(255) null,<br>);<br>create table dbo.employee (<br>EmployeeID tinyint NOT null primary key,<br>EmployeeFirstName varchar(50) NOT null,<br>EmployeeLastName varchar(50) NOT null,<br>EmployeeExtension char(4) null,<br>);<br>create table dbo.sale (<br>SaleID tinyint not null primary key,<br>CustomerID int not null references customer(CustomerID),<br>InventoryID tinyint not null references Inventory(InventoryID),<br>EmployeeID tinyint not null references Employee(EmployeeID),<br>SaleDate date not null,<br>SaleQuantity int not null,<br>SaleUnitPrice smallmoney not null<br>); |
| 4. Check what table inside | select * from information_schema.tables |
| 5. View specific row | --top: show only the first two<br>select top 2 * from customer<br>--top 40 percent: also means show the first two<br>select top 40 percent * from customer |
| 6. View specific column | --sort result (by default is ascending)<br>select customerfirstname, customerlastname from customer<br>order by customerlastname desc<br>select customerfirstname, customerlastname from customer<br>order by 4, 2, 3 desc -- Order By Based on column no. without typing column name<br>--distinct: only show unique value<br>select distinct customerlastname from customer<br>order by customerlastname |

| | |
|---|---|
| 7. Save table to another table | --into file_name: save result in another table (BASE TABLE)<br>select distinct customerlastname into temp<br>from customer<br>order by customerlastname<br>select * from temp --see the table (data type will remain) |
| 8. Like (search something) | -- (underscore sign) _ is only specific for **one character** only<br>-- (percent sign) % represents zero, one, or **multiple characters**<br>select * from customer<br>where customerlastname like '_r%' |
| 9. In (search something) | -- search multiple items<br>select * from customer<br>where customerlastname in ('Brown', 'Michael', 'Jim') |
| 10. > (search something) | select * from customer<br>where customerlastname > 'Brown' or customerlastname>'Cross' |
| 11. <> (Not Equal) | select * from customer<br>where customerlastname <> 'Brown' |
| 12. IS NULL | -- check null values<br>select * from customer<br>where customerlastname IS NULL |
| 13. IS NOT NULL | select * from customer<br>where customerlastname IS NOT NULL |
| 14. between | select * from sale<br>where saleunitprice between 5 and 10 --not include 5 & 10 |
| 15. count | -- returns the number of rows in a table<br>-- AS means aliasing, temporary giving name to a column/ table<br>select count(*) as [Number of Records] from customer<br>where customerfirstname like 'B%' |
| 16. sum | select sale.employeeid ,EmployeeFirstName, EmployeeLastName , count(*) as [Number of order] ,<br>sum(salequantity) as [Total Quantity]<br>from sale,employee<br>where sale.employeeid = employee.employeeid<br>group by sale.employeeid ,EmployeeFirstName, EmployeeLastName |
| 17. count month | select month(saledate) as [Month], count ( * ) as [Number of sale],<br>sum(salequantity*saleunitprice) as [Total Amount]<br>from sale<br>group by month(saledate) |
| 18. max | SELECT MAX(Salary)<br>FROM EmployeeSalary |
| 19. min | SELECT MIN(Salary)<br>FROM EmployeeSalary |
| 20. average | SELECT AVG(Salary)<br>FROM EmployeeSalary |

21. having

```
SELECT JobTitle, COUNT(JobTitle)
FROM EmployeeDemographics ED
JOIN EmployeeSalary ES
        ON ED.EmployeeID = ES.EmployeeID
GROUP   BY   JobTitle   HAVING
COUNT(JobTitle) > 1

SELECT JobTitle, AVG(Salary)

FROM EmployeeDemographics ED
JOIN EmployeeSalary ES

        ON ED.EmployeeID = ES.EmployeeID
GROUP BY JobTitle
HAVING AVG(Salary) > 45000
ORDER BY AVG(Salary)
```

22. Change data type

temporary for use

```
-- CAST(expression AS datatype(length))
SELECT CAST('2017-08-25 00:00:00.000' AS date)

-- CONVERT(data_type(length), expression, style)
SELECT CONVERT(date,'2017-08-25 00:00:00.000')
SELECT FirstName, LastName, Age,
```

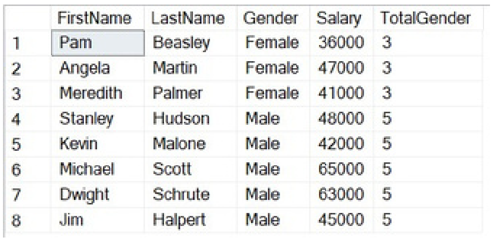
23. CASE Statement

```
CASE

   WHEN Age > 30 THEN 'Old'
   WHEN Age BETWEEN 27 AND 30 THEN 'Young'
   ELSE 'Baby'
END
FROM EmployeeDemographics ED
WHERE Age IS NOT NULL
ORDER BY Age

--

SELECT FirstName, LastName, JobTitle, Salary,
CASE
   WHEN JobTitle = 'Salesman' THEN Salary + (Salary *.10)
   WHEN JobTitle = 'Accountant' THEN Salary + (Salary *.05)
   WHEN JobTitle = 'HR' THEN Salary + (Salary *.000001)
   ELSE Salary + (Salary *.03)
END AS SalaryAfterRaise
FROM EmployeeDemographics ED
JOIN EmployeeSalary ES
ON ED.EmployeeID = ES.EmployeeID
```

24. Partition By

--returns a single value for each row

```
SELECT FirstName, LastName, Gender, Salary,
COUNT(Gender) OVER (PARTITION BY Gender) AS TotalGender
FROM EmployeeDemographics ED
JOIN EmployeeSalary ES
ON ED.EmployeeID = ES.EmployeeID
```



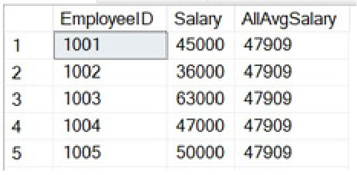
| | FirstName | LastName | Gender | Salary | TotalGender |
|---|---|---|---|---|---|
| 1 | Pam | Beasley | Female | 36000 | 3 |
| 2 | Angela | Martin | Female | 47000 | 3 |
| 3 | Meredith | Palmer | Female | 41000 | 3 |
| 4 | Stanley | Hudson | Male | 48000 | 5 |
| 5 | Kevin | Malone | Male | 42000 | 5 |
| 6 | Michael | Scott | Male | 65000 | 5 |
| 7 | Dwight | Schrute | Male | 63000 | 5 |
| 8 | Jim | Halpert | Male | 45000 | 5 |

|  |  |
|---|---|
| 25. String Functions | -- Remove space<br>Select EmployeeID, TRIM(EmployeeID) AS IDTRIM<br>FROM EmployeeErrors<br>Select EmployeeID, RTRIM(EmployeeID) as IDRTRIM<br>FROM EmployeeErrors<br>Select EmployeeID, LTRIM(EmployeeID) as IDLTRIM<br>FROM EmployeeErrors<br>-- Replace<br>Select LastName, REPLACE(LastName, '- Fired', '') as LastNameFixed<br>FROM EmployeeErrors<br>-- Substring<br>Select Substring(err.FirstName,1,3), Substring(dem.FirstName,1,3), Substring(err.LastName,1,3), Substring(dem.LastName,1,3)<br>FROM EmployeeErrors err<br>JOIN EmployeeDemographics dem<br>on Substring(err.FirstName,1,3) = Substring(dem.FirstName,1,3)<br>and Substring(err.LastName,1,3) = Substring(dem.LastName,1,3)<br>-- UPPER and LOWER CASE<br>Select firstname, LOWER(firstname)<br>from EmployeeErrors<br>Select Firstname, UPPER(FirstName)<br>from EmployeeErrors" |
| 26. Stored Procedure | CREATE PROCEDURE Temp_Employee<br>@JobTitle nvarchar(100)<br>AS<br>DROP TABLE IF EXISTS #temp_employee<br>Create table #temp_employee (<br>JobTitle varchar(100),<br>EmployeesPerJob int ,<br>AvgAge int,<br>AvgSalary int<br>)<br>Insert into #temp_employee<br>SELECT JobTitle, Count(JobTitle), Avg(Age), AVG(salary)<br>FROM EmployeeDemographics emp<br>JOIN EmployeeSalary sal<br>ON emp.EmployeeID = sal.EmployeeID<br>where JobTitle = @JobTitle --- make sure to change this in this script from original above<br>group by JobTitle<br>Select *<br>From #temp_employee<br>GO; |

```
--- only need to run this on next time
EXEC Temp_Employee @JobTitle = 'Salesman'
```

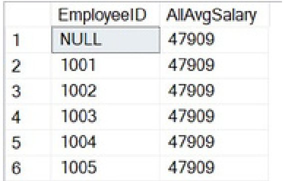
27. Subquery

```
-- Subquery in Select
SELECT EmployeeID, Salary, (SELECT AVG(Salary) FROM
EmployeeSalary) AS AllAvgSalary
FROM EmployeeSalary
-- with Partition By
SELECT EmployeeID, Salary, AVG(Salary) OVER () AS
AllAvgSalary
FROM EmployeeSalary
```



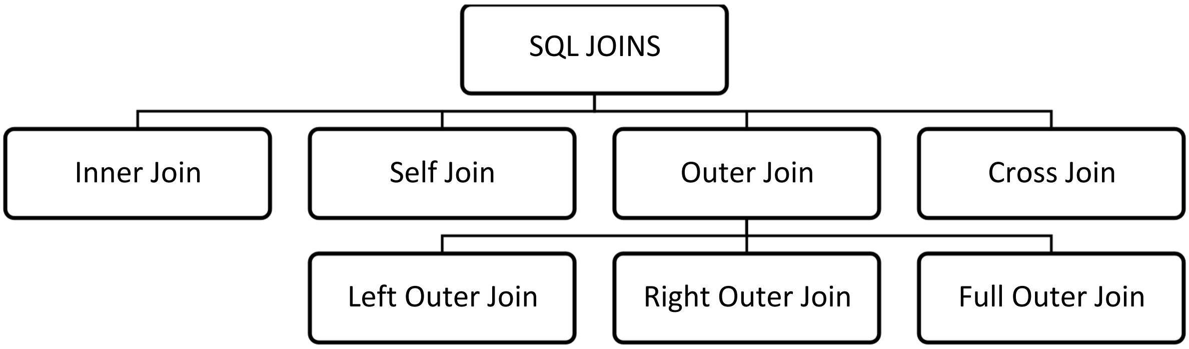
| | EmployeeID | Salary | AllAvgSalary |
|---|---|---|---|
| 1 | 1001 | 45000 | 47909 |
| 2 | 1002 | 36000 | 47909 |
| 3 | 1003 | 63000 | 47909 |
| 4 | 1004 | 47000 | 47909 |
| 5 | 1005 | 50000 | 47909 |

```
-- Subquery in From
SELECT a.EmployeeID, AllAvgSalary
FROM (SELECT EmployeeID, Salary, AVG(Salary) OVER () AS
AllAvgSalary
         FROM EmployeeSalary) a
ORDER BY a.EmployeeID
```



| | EmployeeID | AllAvgSalary |
|---|---|---|
| 1 | NULL | 47909 |
| 2 | 1001 | 47909 |
| 3 | 1002 | 47909 |
| 4 | 1003 | 47909 |
| 5 | 1004 | 47909 |
| 6 | 1005 | 47909 |

```
-- Subquery in Where
SELECT EmployeeID, JobTitle, Salary
FROM EmployeeSalary
WHERE EmployeeID in (SELECT EmployeeID FROM
EmployeeDemographics
                     WHERE Age > 30)

SELECT EmployeeID, JobTitle, Salary
FROM EmployeeSalary
WHERE Salary in (SELECT Max(Salary) FROM EmployeeSalary)
```


SQL JOINS

Inner Join

Self Join

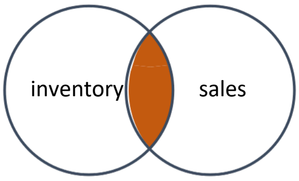
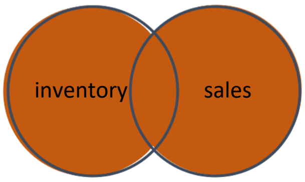
Outer Join

Cross Join

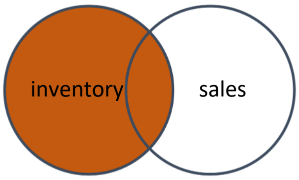
Left Outer Join

Right Outer Join

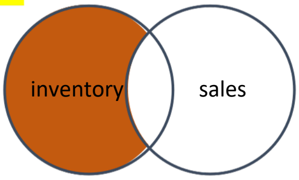
Full Outer Join

| | |
|---|---|
| 1. getting data from multiple tables (explicit join - without using join command) | select * from inventory,sale<br>where sale.inventoryid=inventory.inventoryid |
| | select<br>inventoryname,saledate,saleunitprice,salequantity,salequantity*saleunitprice as [Total amount]<br>from sale,inventory<br>where sale.inventoryid=inventory.inventoryid<br>group by sale.inventoryid,inventoryname,saledate,salequantity,saleunitprice<br>order by inventoryname |
| 2. getting data from multiple tables (implicit join - using join command) | --inner join<br>select * from inventory<br>inner join sale<br>on sale.inventoryid=inventory.inventoryid<br><br>select<br>inventoryname,saledate,saleunitprice,salequantity,saleunitprice*salequantity as [Total Amount]<br>from inventory inner join sale<br>on sale.inventoryid=inventory.inventoryid<br>order by inventoryname<br><br> |
| | --full outer join (shows everything)<br>select sale.inventoryid,inventoryname<br>from inventory<br>full outer join sale on<br>sale.inventoryid=inventory.inventoryid<br>where sale.inventoryid is NULL<br><br> |

```
--left join (might have NULL value, since some inventory might not have sales)
select inventory.inventoryid,inventoryname
from inventory left join sale on
sale.inventoryid=inventory.inventoryid
```

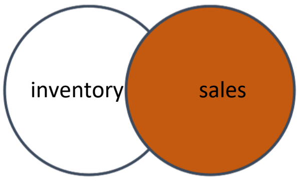


```
--left join
select inventory.inventoryid,inventoryname
from inventory left join sale on
sale.inventoryid=inventory.inventoryid
where sale.inventoryid is NULL
```



```
-- without join: use subquery
select inventoryid,inventoryname from inventory
where inventoryid not in (select inventoryid from sale)
```

```
--right join
select sale.inventoryid,inventoryname
from inventory right join sale on
sale.inventoryid=inventory.inventoryid
```



3. Self Join

--commonly used in processing hierarchy

--inner join

Staff Table

| employeeID | employeefirstname | employeelastname | managerID |
|---|---|---|---|
| **1001** | Tan | Mei Ling | NULL |
| **1002** | Kelvin | Koh | 1001 |
| **1003** | Amin | Wong | 1002 |

```
select E.employeeID, E.employeefirstname+' '+E.employeelastname as [Full
Name], E.managerID, , M.employeefirstname+' '+M.employeelastname as
[Manager Name]
from staff E
inner join staff M
on E.managerID = M.employeeID
```

Output:

| employeeID | Full Name | managerID | managerName |
|---|---|---|---|
| **1002** | Kelvin Koh | 1001 | Tan Mei Ling |
| **1003** | Amin Wong | 1002 | Kelvin Koh |

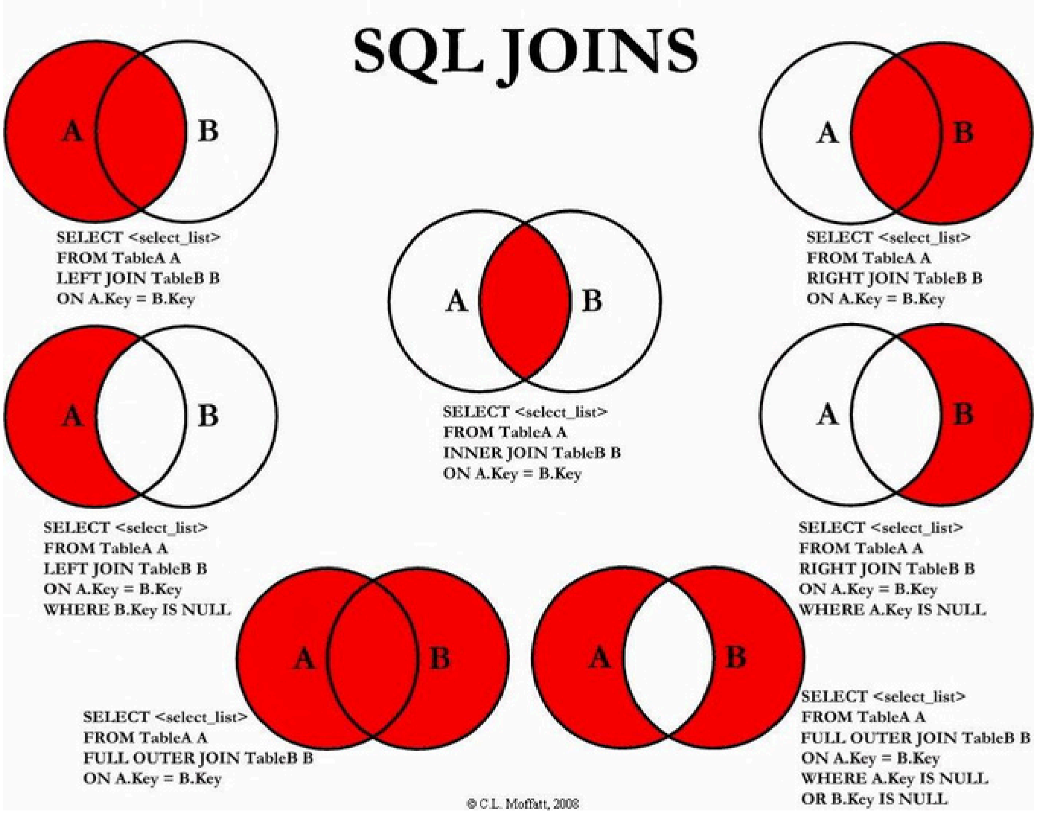
--left outer join (list all the employees)

```
select E.employeeID, E.employeefirstname+' '+E.employeelastname as [F
Name], E.managerID, , M.employeefirstname+' '+M.employeelastname as
[Manager Name]
from staff E
left outer join staff M
on E.managerID = M.employeeID
```

Output:

| employeeID | Full Name | managerID | managerName |
|---|---|---|---|
| **1001** | Tan Mei Ling | | |
| **1002** | Kelvin Koh | 1001 | Tan Mei Ling |
| **1003** | Amin Wong | 1002 | Kelvin Koh |

| | |
|---|---|
| 4. Cross Join<br>--generate all combination of records (all possibility) (Cartesian Product) | select * from inventory1<br>cross join inventory2 |



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

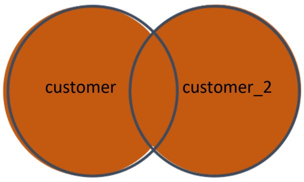
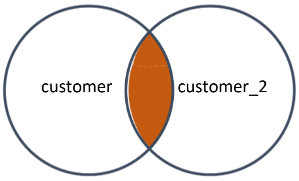
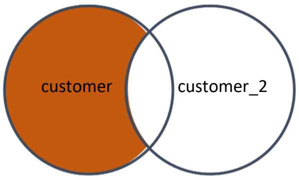
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

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

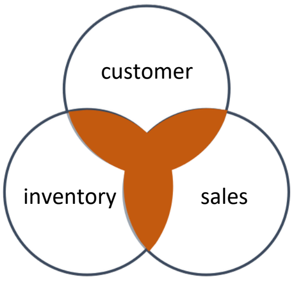
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
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
```

# SQL UNIONS

| | |
|---|---|
| 1. Union<br>--allow you to combine two tables together (but the no. of columns & each column's data types for 2 tables must be match)<br>--don't need common key, only need common attributes<br>--merge, not showing duplicate record<br>2. Union all | select cust_lname,cust_fname from customer<br>union<br>select cust_lname,cust_fname from customer_2 |
| --merge, but show you everything, even the duplicate record | select cust_lname,cust_fname from customer<br>union all<br>select cust_lname,cust_fname from customer_2<br> |
| 3. Intersect<br>--keep only the rows in common to both query<br>--not showing duplicate record | select cust_lname,cust_fname from customer<br>intersect<br>select cust_lname,cust_fname from customer_2<br> |
| | select c.cust_lname,c.cust_fname from customer c,customer_2 c2<br>where c.cust_lname=c2.cust_lname and c.cust_fname=c2.cust_fname |
| 4. Except<br>--generate only the records that are unique to<br>the CUSTOMER table | select cust_lname,cust_fname from customer<br>except<br>select cust_lname,cust_fname from customer_2<br> |
| | --use subquery<br>select cust_lname,cust_fname from customer<br>where(cust_lname) not in<br>(select cust_lname from customer_2) and<br>(cust_fname) not in<br>(select cust_fname from customer_2) |

# Table & View

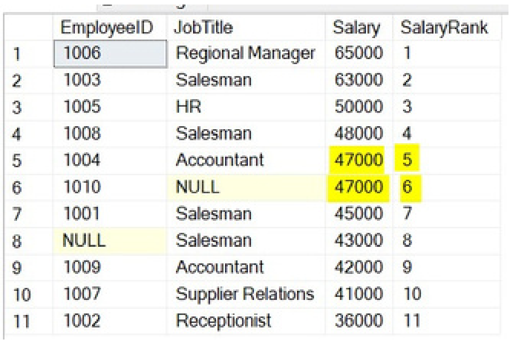
| | |
|---|---|
| 1. view table<br>(view will be updated when update base)<br>--view is a result set of SQL statements, exists only for a single query | create view CustomerView as<br>select customerfirstname+' '+customerlastname as [Customer Name] , customerphonenumber, inventoryname,saledate,salequantity,saleunitprice,salequantity*saleunitprice as [Total Amount]<br>from customer inner join sale on customer.customerid=sale.customerid inner join inventory<br>on sale.inventoryid=inventory.inventoryid<br><br> |
| 2. Temp table<br>(temp will NOT be updated when update base)<br>--a single hashtag (#) sign must be added in front of their names<br>--used to store data temporarily, physically created in the Tempdb database<br>--can perform CRUD, join, and some other operations like the persistent database tables | DROP TABLE IF EXISTS #temp_Employee<br>Create table #temp_Employee (<br>JobTitle varchar(100),<br>EmployeesPerJob int,<br>AvgAge int,<br>AvgSalary int<br>)<br>Insert INTO #temp_Employee<br>SELECT JobTitle, Count(JobTitle), Avg(Age), AVG(salary)<br>FROM EmployeeDemographics emp<br>JOIN EmployeeSalary sal<br><br>        ON emp.EmployeeID = sal.EmployeeID<br>group by JobTitle<br>SELECT * FROM #temp_Employee |
| 3. CTE (Common Table Expression)<br>--create temporary result set which is used to manipulate the complex sub-queries data<br>--created in memory rather than Tempdb database, so cannot create any index on CTE. | WITH CTE_Employee AS<br>(<br>SELECT FirstName, LastName, Gender, Salary,<br>COUNT(Gender) OVER (PARTITION BY Gender) AS TotalGender<br>FROM EmployeeDemographics ED<br>JOIN EmployeeSalary ES<br>       ON ED.EmployeeID = ES.EmployeeID<br>WHERE Salary > '45000'<br>)<br><br>SELECT FirstName, LastName, Gender, TotalGender<br>FROM CTE_Employee<br>WHERE TotalGender = (SELECT MIN(TotalGender) FROM CTE_Employee) |
| 4. Duplicate Table | select customerfirstname+' '+customerlastname as [Customer Name] , customerphonenumber, inventoryname,saledate,salequantity,saleunitprice,salequantity*saleunitprice as [Total Amount] into customerRec<br>from customer inner join sale on customer.customerid=sale.customerid inner join inventory<br>on sale.inventoryid=inventory.inventoryid<br>order by customerfirstname +' '+ customerlastname,inventoryname |

# SQL RANKS

1. ROW_NUMBER()

--get a unique sequential number for each row
--get different ranks for the row having similar values

```
SELECT *,
	ROW_NUMBER() OVER(ORDER BY Salary DESC) SalaryRank
FROM EmployeeSalary
```



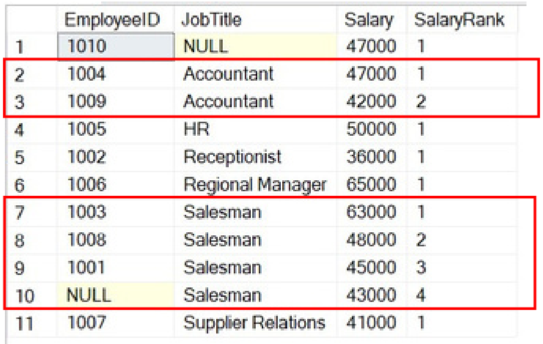
| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1006 | Regional Manager | 65000 | 1 |
| 2 | 1003 | Salesman | 63000 | 2 |
| 3 | 1005 | HR | 50000 | 3 |
| 4 | 1008 | Salesman | 48000 | 4 |
| 5 | 1004 | Accountant | 47000 | 5 |
| 6 | 1010 | NULL | 47000 | 6 |
| 7 | 1001 | Salesman | 45000 | 7 |
| 8 | NULL | Salesman | 43000 | 8 |
| 9 | 1009 | Accountant | 42000 | 9 |
| 10 | 1007 | Supplier Relations | 41000 | 10 |
| 11 | 1002 | Receptionist | 36000 | 11 |

2. RANK()

--specify rank for each row in the result set
--use PARTITION BY to performs calculation on each group
--each subset get rank as per Salary in descending order

**USING PARTITION BY**

```
SELECT *,
	RANK() OVER(PARTITION BY JobTitle ORDER BY Salary DESC) SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank
```

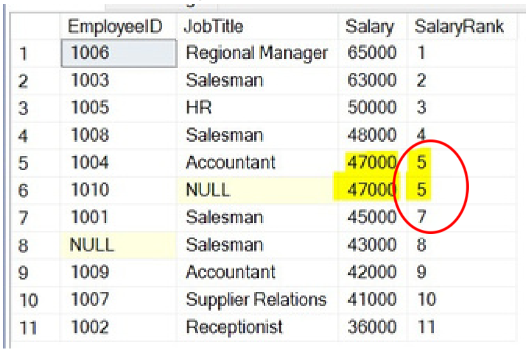


| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1010 | NULL | 47000 | 1 |
| 2 | 1004 | Accountant | 47000 | 1 |
| 3 | 1009 | Accountant | 42000 | 2 |
| 4 | 1005 | HR | 50000 | 1 |
| 5 | 1002 | Receptionist | 36000 | 1 |
| 6 | 1006 | Regional Manager | 65000 | 1 |
| 7 | 1003 | Salesman | 63000 | 1 |
| 8 | 1008 | Salesman | 48000 | 2 |
| 9 | 1001 | Salesman | 45000 | 3 |
| 10 | NULL | Salesman | 43000 | 4 |
| 11 | 1007 | Supplier Relations | 41000 | 1 |

**NOT USING PARTITION BY**

-- get SAME ranks for the row having similar values

```
SELECT *,
	RANK() OVER(ORDER BY Salary DESC) SalaryRank
FROM EmployeeSalary
ORDER BY SalaryRank
```

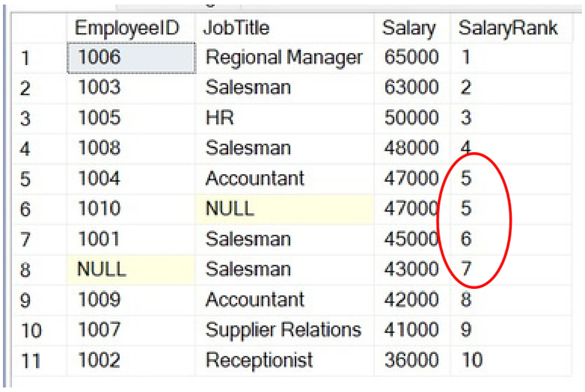


| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1006 | Regional Manager | 65000 | 1 |
| 2 | 1003 | Salesman | 63000 | 2 |
| 3 | 1005 | HR | 50000 | 3 |
| 4 | 1008 | Salesman | 48000 | 4 |
| 5 | 1004 | Accountant | 47000 | 5 |
| 6 | 1010 | NULL | 47000 | 5 |
| 7 | 1001 | Salesman | 45000 | 7 |
| 8 | NULL | Salesman | 43000 | 8 |
| 9 | 1009 | Accountant | 42000 | 9 |
| 10 | 1007 | Supplier Relations | 41000 | 10 |
| 11 | 1002 | Receptionist | 36000 | 11 |

3. DENSE_RANK()

-- if have duplicate values, SQL assigns different ranks to those rows.
-- will get the same rank for duplicate or similar values

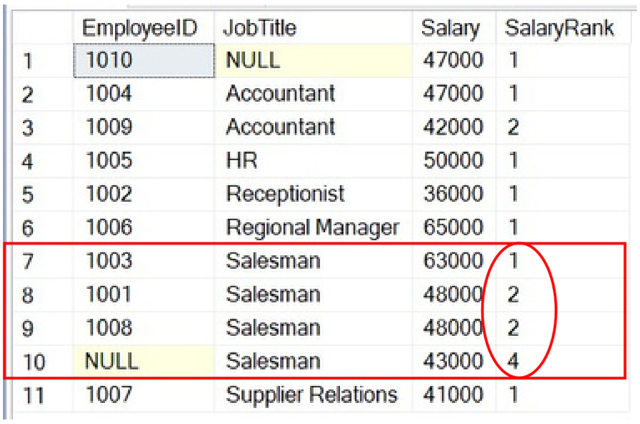
```
SELECT *,
       DENSE_RANK() OVER(ORDER BY Salary DESC) SalaryRank
FROM EmployeeSalary
ORDER BY SalaryRank
```



| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1006 | Regional Manager | 65000 | 1 |
| 2 | 1003 | Salesman | 63000 | 2 |
| 3 | 1005 | HR | 50000 | 3 |
| 4 | 1008 | Salesman | 48000 | 4 |
| 5 | 1004 | Accountant | 47000 | 5 |
| 6 | 1010 | NULL | 47000 | 5 |
| 7 | 1001 | Salesman | 45000 | 6 |
| 8 | NULL | Salesman | 43000 | 7 |
| 9 | 1009 | Accountant | 42000 | 8 |
| 10 | 1007 | Supplier Relations | 41000 | 9 |
| 11 | 1002 | Receptionist | 36000 | 10 |

**RANK()**

```
SELECT *,
       RANK() OVER(PARTITION BY JobTitle ORDER
BY Salary DESC) SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank
```

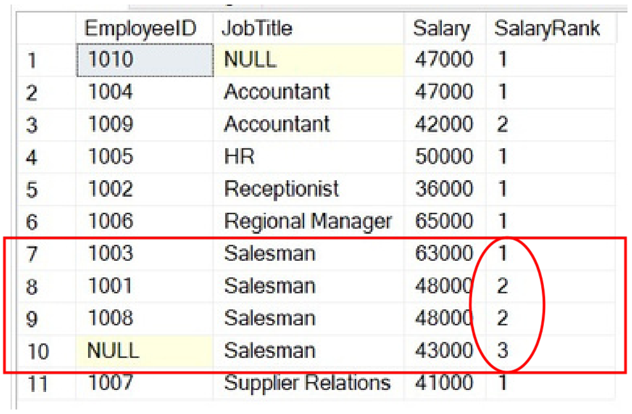


| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1010 | NULL | 47000 | 1 |
| 2 | 1004 | Accountant | 47000 | 1 |
| 3 | 1009 | Accountant | 42000 | 2 |
| 4 | 1005 | HR | 50000 | 1 |
| 5 | 1002 | Receptionist | 36000 | 1 |
| 6 | 1006 | Regional Manager | 65000 | 1 |
| 7 | 1003 | Salesman | 63000 | 1 |
| 8 | 1001 | Salesman | 48000 | 2 |
| 9 | 1008 | Salesman | 48000 | 2 |
| 10 | NULL | Salesman | 43000 | 4 |
| 11 | 1007 | Supplier Relations | 41000 | 1 |

```
-- skip a rank if have similar values
```

**DENSE_RANK()**

```
SELECT *,
       DENSE_RANK() OVER(PARTITION BY JobTitle
ORDER BY Salary DESC) SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank
```



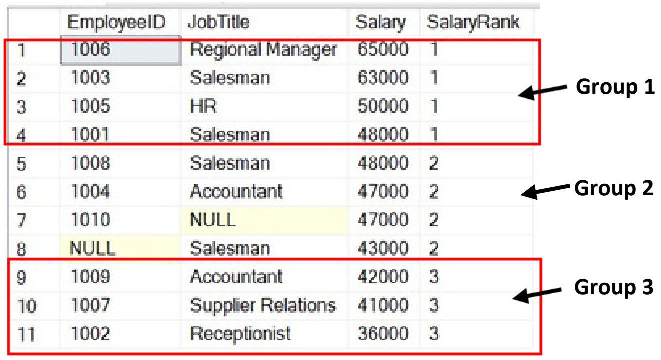
| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1010 | NULL | 47000 | 1 |
| 2 | 1004 | Accountant | 47000 | 1 |
| 3 | 1009 | Accountant | 42000 | 2 |
| 4 | 1005 | HR | 50000 | 1 |
| 5 | 1002 | Receptionist | 36000 | 1 |
| 6 | 1006 | Regional Manager | 65000 | 1 |
| 7 | 1003 | Salesman | 63000 | 1 |
| 8 | 1001 | Salesman | 48000 | 2 |
| 9 | 1008 | Salesman | 48000 | 2 |
| 10 | NULL | Salesman | 43000 | 3 |
| 11 | 1007 | Supplier Relations | 41000 | 1 |

```
-- maintains the rank and does not give any gap
for the values
```

4. NTILE()

-- can specify required how many group of result, and it will rank accordingly

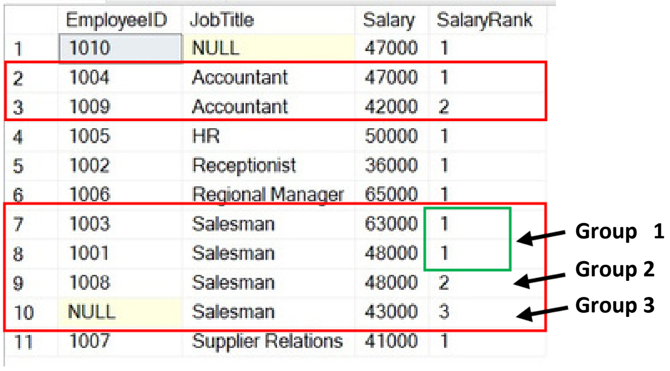
```
SELECT *,

       NTILE(3) OVER(ORDER BY Salary DESC) SalaryRank
FROM EmployeeSalary
ORDER BY SalaryRank;
```



| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1006 | Regional Manager | 65000 | 1 |
| 2 | 1003 | Salesman | 63000 | 1 |
| 3 | 1005 | HR | 50000 | 1 |
| 4 | 1001 | Salesman | 48000 | 1 |
| 5 | 1008 | Salesman | 48000 | 2 |
| 6 | 1004 | Accountant | 47000 | 2 |
| 7 | 1010 | NULL | 47000 | 2 |
| 8 | NULL | Salesman | 43000 | 2 |
| 9 | 1009 | Accountant | 42000 | 3 |
| 10 | 1007 | Supplier Relations | 41000 | 3 |
| 11 | 1002 | Receptionist | 36000 | 3 |

**USING PARTITION BY**

```
SELECT *,
       NTILE(3) OVER(PARTITION BY JobTitle ORDER BY Salary DESC)
SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank;
```



| | EmployeeID | JobTitle | Salary | SalaryRank |
|---|---|---|---|---|
| 1 | 1010 | NULL | 47000 | 1 |
| 2 | 1004 | Accountant | 47000 | 1 |
| 3 | 1009 | Accountant | 42000 | 2 |
| 4 | 1005 | HR | 50000 | 1 |
| 5 | 1002 | Receptionist | 36000 | 1 |
| 6 | 1006 | Regional Manager | 65000 | 1 |
| 7 | 1003 | Salesman | 63000 | 1 |
| 8 | 1001 | Salesman | 48000 | 1 |
| 9 | 1008 | Salesman | 48000 | 2 |
| 10 | NULL | Salesman | 43000 | 3 |
| 11 | 1007 | Supplier Relations | 41000 | 1 |

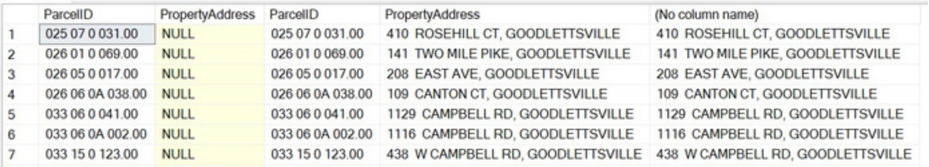
1. Write the query to show the invoice number, the customer number, the customer name, the invoice date, and the invoice amount for all customers with a customer balance of $1,000 or more.

```
select
invoice_num,c.cust_num,c.cust_lname,c.cust_fname,inv_date,inv_amount
from customer c, invoice
where c.cust_num=invoice.cust_num and cust_balance>=1000
```

```
select invoice_num,c.cust_num,cust_lname+' '+cust_fname as
[Name],inv_date,inv_amount
from customer c join invoice i
on c.cust_num=i.cust_num
where cust_balance>=1000
```

2. ISNULL(expression, value)
--expression: to test whether is NULL, value: to return if expression is NULL

```
--ParcelID is same, but UniqueID is different; can assume that if the ParcelID is
same, the Property Address will be same
Select  a.ParcelID,  a.PropertyAddress,  b.ParcelID,
b.PropertyAddress,
ISNULL(a.PropertyAddress,b.PropertyAddress)    From
NashvilleHousing a JOIN NashvilleHousing b

      on a.ParcelID = b.ParcelID
      AND a.[UniqueID] <> b.[UniqueID]
Where a.PropertyAddress is null
```



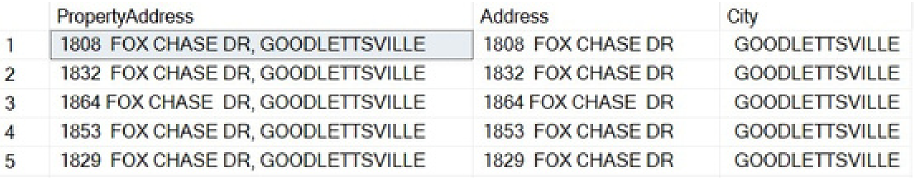
| | ParcelID | PropertyAddress | ParcelID | PropertyAddress | (No column name) |
|---|---|---|---|---|---|
| 1 | 025 07 0 031.00 | NULL | 025 07 0 031.00 | 410 ROSEHILL CT, GOODLETTSVILLE | 410 ROSEHILL CT, GOODLETTSVILLE |
| 2 | 026 01 0 069.00 | NULL | 026 01 0 069.00 | 141 TWO MILE PIKE, GOODLETTSVILLE | 141 TWO MILE PIKE, GOODLETTSVILLE |
| 3 | 026 05 0 017.00 | NULL | 026 05 0 017.00 | 208 EAST AVE, GOODLETTSVILLE | 208 EAST AVE, GOODLETTSVILLE |
| 4 | 026 06 0A 038.00 | NULL | 026 06 0A 038.00 | 109 CANTON CT, GOODLETTSVILLE | 109 CANTON CT, GOODLETTSVILLE |
| 5 | 033 06 0 041.00 | NULL | 033 06 0 041.00 | 1129 CAMPBELL RD, GOODLETTSVILLE | 1129 CAMPBELL RD, GOODLETTSVILLE |
| 6 | 033 06 0A 002.00 | NULL | 033 06 0A 002.00 | 1116 CAMPBELL RD, GOODLETTSVILLE | 1116 CAMPBELL RD, GOODLETTSVILLE |
| 7 | 033 15 0 123.00 | NULL | 033 15 0 123.00 | 438 W CAMPBELL RD, GOODLETTSVILLE | 438 W CAMPBELL RD, GOODLETTSVILLE |

```
-- Update record
Update     a     SET     PropertyAddress     =
ISNULL(a.PropertyAddress,b.PropertyAddress)
From       NashvilleHousing       a       JOIN
NashvilleHousing b

      on a.ParcelID = b.ParcelID
      AND a.[UniqueID] <> b.[UniqueID]
Where a.PropertyAddress is null
```

3. Split by delimiter

- SUBSTRING(string, start, length)
- CHARINDEX(substring, string, start)
- LEN(string)

```
SELECT PropertyAddress,
SUBSTRING(PropertyAddress, 1, CHARINDEX(',',
PropertyAddress) -1 ) as Address
, SUBSTRING(PropertyAddress, CHARINDEX(',',
PropertyAddress) + 1 , LEN(PropertyAddress)) as City
From NashvilleHousing
```



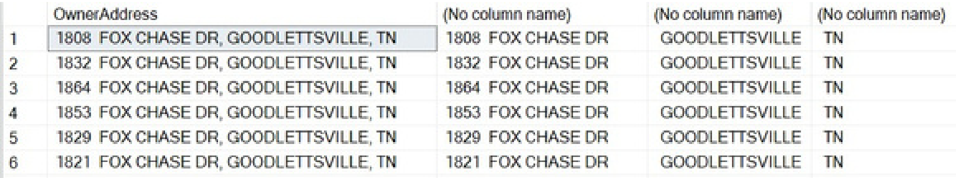
| | PropertyAddress | Address | City |
|---|---|---|---|
| 1 | 1808 FOX CHASE DR, GOODLETTSVILLE | 1808 FOX CHASE DR | GOODLETTSVILLE |
| 2 | 1832 FOX CHASE DR, GOODLETTSVILLE | 1832 FOX CHASE DR | GOODLETTSVILLE |
| 3 | 1864 FOX CHASE DR, GOODLETTSVILLE | 1864 FOX CHASE DR | GOODLETTSVILLE |
| 4 | 1853 FOX CHASE DR, GOODLETTSVILLE | 1853 FOX CHASE DR | GOODLETTSVILLE |
| 5 | 1829 FOX CHASE DR, GOODLETTSVILLE | 1829 FOX CHASE DR | GOODLETTSVILLE |

```
ALTER TABLE NashvilleHousing
Add PropertySplitAddress Nvarchar(255);
ALTER TABLE NashvilleHousing
Add PropertySplitCity Nvarchar(255);
```

```sql
Update NashvilleHousing
SET PropertySplitAddress = SUBSTRING(PropertyAddress, 1,
CHARINDEX(',', PropertyAddress) -1 )

Update NashvilleHousing

SET PropertySplitCity = SUBSTRING(PropertyAddress,
CHARINDEX(',', PropertyAddress) + 1 , LEN(PropertyAddress))
```

- PARSENAME('object_name' , object_piece)
  --numbering works from right to left
- REPLACE(string, old_string, new_string)

```sql
Select OwnerAddress,

PARSENAME(REPLACE(OwnerAddress, ',', '.') , 3)
,PARSENAME(REPLACE(OwnerAddress, ',', '.') , 2)
,PARSENAME(REPLACE(OwnerAddress, ',', '.') , 1)
From NashvilleHousing
```



| | OwnerAddress | (No column name) | (No column name) | (No column name) |
|---|---|---|---|---|
| 1 | 1808 FOX CHASE DR, GOODLETTSVILLE, TN | 1808 FOX CHASE DR | GOODLETTSVILLE | TN |
| 2 | 1832 FOX CHASE DR, GOODLETTSVILLE, TN | 1832 FOX CHASE DR | GOODLETTSVILLE | TN |
| 3 | 1864 FOX CHASE DR, GOODLETTSVILLE, TN | 1864 FOX CHASE DR | GOODLETTSVILLE | TN |
| 4 | 1853 FOX CHASE DR, GOODLETTSVILLE, TN | 1853 FOX CHASE DR | GOODLETTSVILLE | TN |
| 5 | 1829 FOX CHASE DR, GOODLETTSVILLE, TN | 1829 FOX CHASE DR | GOODLETTSVILLE | TN |
| 6 | 1821 FOX CHASE DR, GOODLETTSVILLE, TN | 1821 FOX CHASE DR | GOODLETTSVILLE | TN |

```sql
ALTER TABLE NashvilleHousing
Add OwnerSplitAddress Nvarchar(255);
ALTER TABLE NashvilleHousing
Add OwnerSplitCity Nvarchar(255);
ALTER TABLE NashvilleHousing
Add OwnerSplitState Nvarchar(255);
Update NashvilleHousing

SET OwnerSplitAddress = PARSENAME(REPLACE(OwnerAddress,
',', '.') , 3)

Update NashvilleHousing

SET OwnerSplitCity = PARSENAME(REPLACE(OwnerAddress, ',',
'.') , 2)

Update NashvilleHousing

SET OwnerSplitState = PARSENAME(REPLACE(OwnerAddress, ',',
'.') , 1)

WITH RowNumCTE AS(
```

5. Remove duplicate records

```sql
Select *,

	ROW_NUMBER() OVER (
	PARTITION BY ParcelID,
				 PropertyAddress,
				 SalePrice,
				 SaleDate,
				 LegalReference
				 ORDER BY UniqueID) as row_num

From NashvilleHousing
order by ParcelID
)
--DELETE
Select * From RowNumCTE
Where row_num > 1
Order by PropertyAddress
```