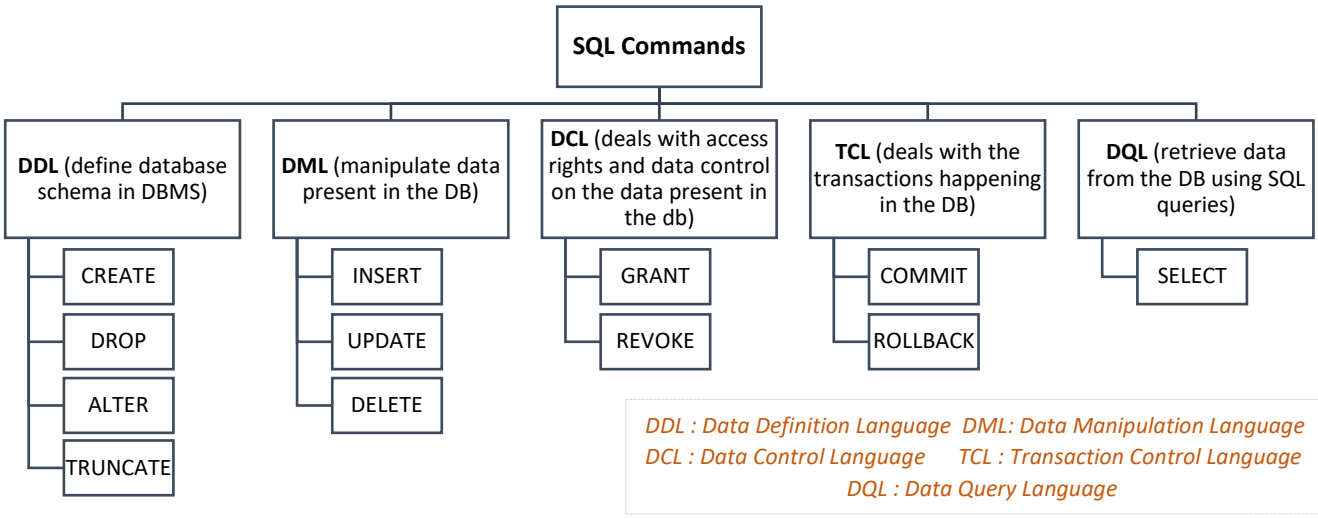
Structured Query language (SQL)



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`
	`(`
	`customerid int identity(1,1) primary key,`
	`customernumber int not null unique check (customernumber>0),`
	`lastname varchar(30) not null,`
	`firstname varchar(30) not null,`
	`areacode int default 71000,`
	`address varchar(50),`
	`country varchar(50) default 'Malaysia'`
	`)`

4. Insert values into table	`insert into customer values`
	`(100,'Fang Ying','Sham','418999','sdadasfdfd',default),`
	`(200,'Mei Mei','Tan',default,'adssdsadsd','Thailand'),`
	`(300,'Albert','John',default,'dfdsfsdf',default)`

5. Display record from table	`-- display all records`
	`select * from customer`
	
	`-- display particular columns`
	`select customerid, customernumber, lastname, firstname`
	`from customer`

6. Add new column to table	`alter table customer` `add phonenumber varchar(20)`
7. Add values to newly added column/ Update table	`update customer set phonenumber='1234545346' where customerid=1` `update customer set phonenumber='45554654' where customerid=2`
8. Delete a column	`alter table customer` `drop column phonenumber`

9. Delete record from table
`--if not put 'where', will delete all record`
	`delete`
	`from customer`
	`where country='Thailand'`

10. Delete table	`drop table customer`
11. Change data type	`alter table customer` `alter column phonenumber varchar(10)`

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

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

23. CASE Statement

```
SELECT FirstName, LastName, Age,
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

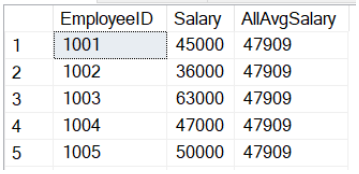
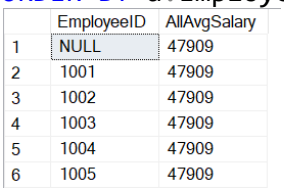
```
SELECT FirstName, LastName, Gender, Salary,
COUNT(Gender) OVER (PARTITION BY Gender) AS TotalGender
FROM EmployeeDemographics ED
JOIN EmployeeSalary ES
ON ED.EmployeeID = ES.EmployeeID
```

	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	
26. Stored Procedure	

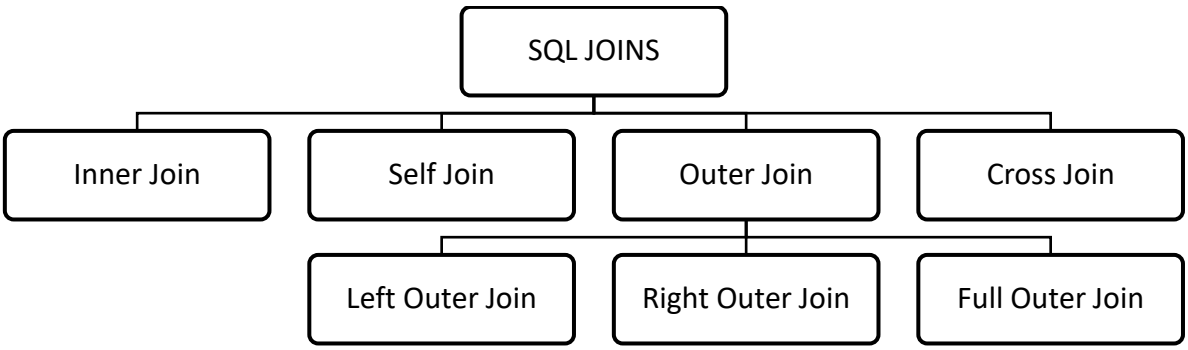
```sql
-- Remove space
Select EmployeeID, TRIM(EmployeeID) AS IDTRIM
FROM EmployeeErrors

Select EmployeeID, RTRIM(EmployeeID) as IDRTRIM
FROM EmployeeErrors

Select EmployeeID, LTRIM(EmployeeID) as IDLTRIM
FROM EmployeeErrors

-- Replace
Select LastName, REPLACE(LastName, '- Fired', '') as
LastNameFixed
FROM EmployeeErrors

-- Substring
Select Substring(err.FirstName,1,3),
Substring(dem.FirstName,1,3), Substring(err.LastName,1,3),
Substring(dem.LastName,1,3)
FROM EmployeeErrors err
JOIN EmployeeDemographics dem
      on Substring(err.FirstName,1,3) =
Substring(dem.FirstName,1,3)
      and Substring(err.LastName,1,3) =
Substring(dem.LastName,1,3)

-- UPPER and LOWER CASE
Select firstname, LOWER(firstname)
from EmployeeErrors

Select Firstname, UPPER(FirstName)
from EmployeeErrors"
```

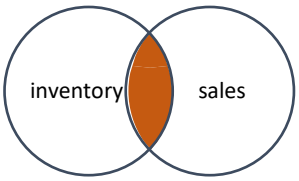
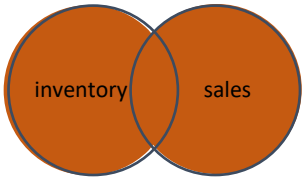
```sql
CREATE PROCEDURE Temp_Employee
@JobTitle nvarchar(100)
AS
DROP TABLE IF EXISTS #temp_employee
Create table #temp_employee (
JobTitle varchar(100),
EmployeesPerJob int ,
AvgAge int,
AvgSalary int
)

Insert into #temp_employee
SELECT JobTitle, Count(JobTitle), Avg(Age), AVG(salary)
FROM EmployeeDemographics emp
JOIN EmployeeSalary sal
        ON emp.EmployeeID = sal.EmployeeID
where JobTitle = @JobTitle --- make sure to change this in
this script from original above
group by JobTitle

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

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

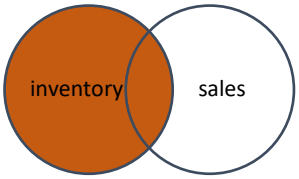
	EmployeeID	AllAvgSalary
1	NULL	47909
2	1001	47909
3	1002	47909
4	1003	47909
5	1004	47909
6	1005	47909



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

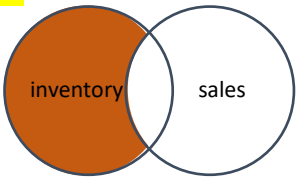
--left join (might have NULL value, since some inventory might not have sales)

```
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
```

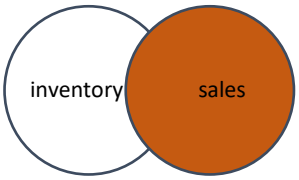


-- without join: use subquery

```
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

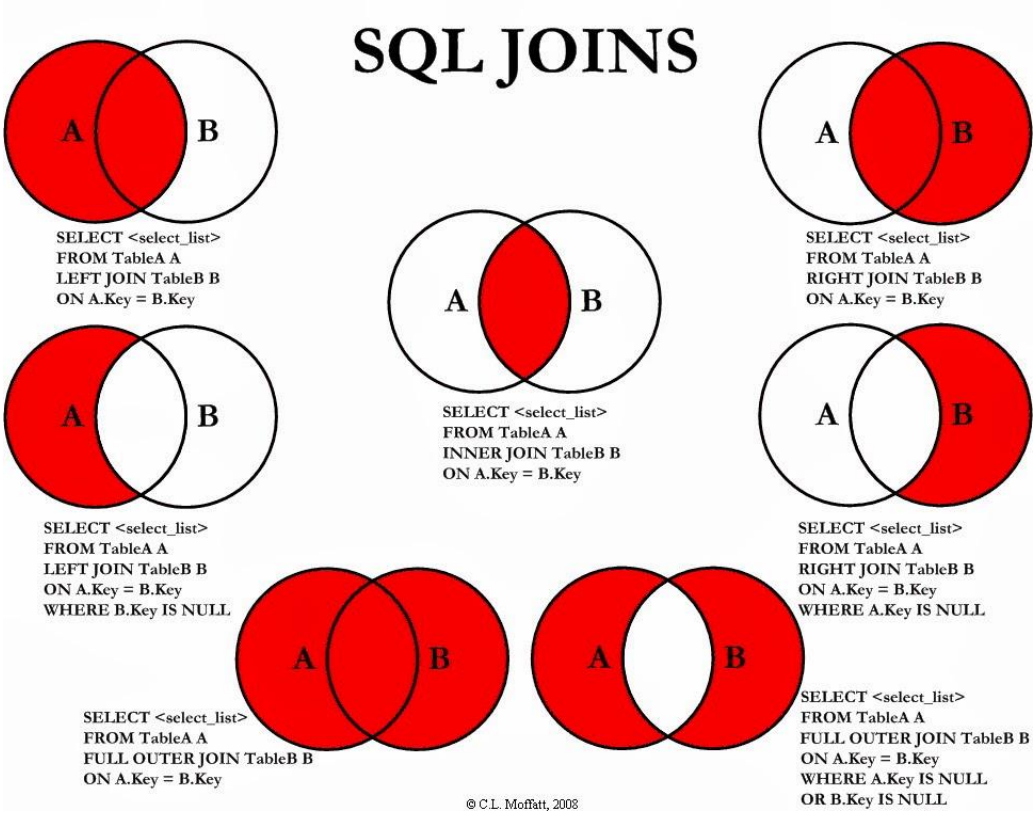
employeeID	employeefirstname	employeelastname	managerID
1001	Tan	Mei Ling	NULL
1002	Kelvin	Koh	1001
1003	Amin	Wong	1002

```
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
```

<table>
<tr>
<td></td>
<td>
Output:
<table>
<tr><th>employeeID</th><th>Full Name</th><th>managerID</th><th>managerName</th></tr>
<tr><td>1002</td><td>Kelvin Koh</td><td>1001</td><td>Tan Mei Ling</td></tr>
<tr><td>1003</td><td>Amin Wong</td><td>1002</td><td>Kelvin Koh</td></tr>
</table>

--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:
<table>
<tr><th>employeeID</th><th>Full Name</th><th>managerID</th><th>managerName</th></tr>
<tr><td>1001</td><td>Tan Mei Ling</td><td></td><td></td></tr>
<tr><td>1002</td><td>Kelvin Koh</td><td>1001</td><td>Tan Mei Ling</td></tr>
<tr><td>1003</td><td>Amin Wong</td><td>1002</td><td>Kelvin Koh</td></tr>
</table>
</td>
</tr>
<tr>
<td>4. Cross Join
--generate all combination of records (all possibility) (Cartesian Product)</td>
<td>select * from inventory1
cross join inventory2</td>
</tr>
</table>



SQL UNIONS

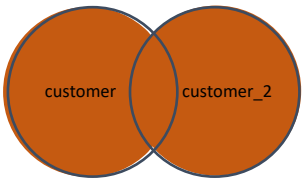
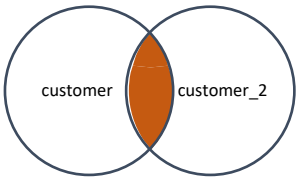
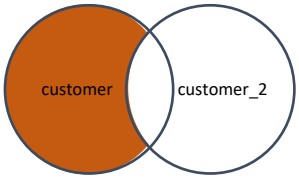
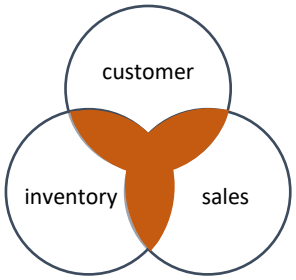
<table>
<tr>
<td>1. Union
--allow you to combine two tables together (but the no. of columns & each column's data types for 2 tables must be match)
--don't need common key, only need common attributes
--merge, not showing duplicate record</td>
<td>select cust_lname,cust_fname from customer
union
select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

Table & View

1. view table
(view will be updated when update base)
--view is a result set of SQL statements, exists only for a single query

```
create view CustomerView as
select customerfirstname+' '+customerlastname as [Customer Name] ,
customerphonenumber,
inventoryname,saledate,salequantity,saleunitprice,salequantity*saleunitprice
as [Total Amount]
from customer inner join sale on customer.customerid=sale.customerid inner
join inventory
on sale.inventoryid=inventory.inventoryid
```



2. Temp table
(temp will NOT be updated when update base)
--a single hashtag (#) sign must be added in front of their names
--used to store data temporarily, physically created in the Tempdb database
--can perform CRUD, join, and some other operations like the persistent database tables

```
DROP TABLE IF EXISTS #temp_Employee

Create table #temp_Employee (
JobTitle varchar(100),
EmployeesPerJob int,
AvgAge int,
AvgSalary int
)

Insert INTO #temp_Employee
SELECT JobTitle, Count(JobTitle), Avg(Age), AVG(salary)
FROM EmployeeDemographics emp
JOIN EmployeeSalary sal
        ON emp.EmployeeID = sal.EmployeeID
group by JobTitle

SELECT * FROM #temp_Employee
```

3. CTE (Common Table Expression)
--create temporary result set which is used to manipulate the complex sub-queries data
--created in memory rather than Tempdb database, so cannot create any index on CTE.

```
WITH CTE_Employee AS
(
SELECT FirstName, LastName, Gender, Salary,
COUNT(Gender) OVER (PARTITION BY Gender) AS TotalGender
FROM EmployeeDemographics ED
JOIN EmployeeSalary ES
       ON ED.EmployeeID = ES.EmployeeID
WHERE Salary > '45000'
)

SELECT FirstName, LastName, Gender, TotalGender
FROM CTE_Employee
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
from customer inner join sale on customer.customerid=sale.customerid inner
join inventory
on sale.inventoryid=inventory.inventoryid
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

Group 1
Group 2
Group 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

Group 1
Group 2
Group 3

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);
```

```
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)

```
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

```
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)
```

5. Remove duplicate records

```
WITH RowNumCTE AS(
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
```