SQL Server

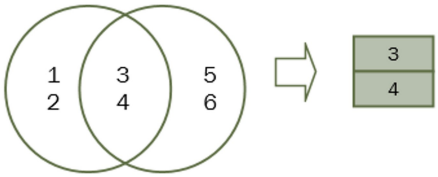
23 July 2021 11:38 AM

1. Structured query language for all RDBMS databases such as
2. MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

Table - data in RDBMS is stored in form of tables
Field - column heads
Record - row
Column - vertical entry
Null - no value ,left blank

As - for giving different name to database

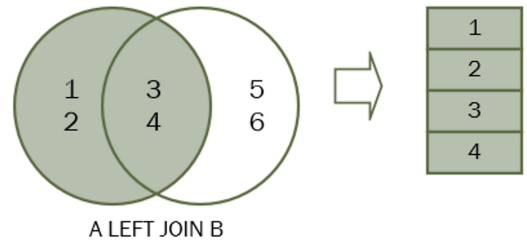
Inner Joins




```
SELECT
    first_name,
    last_name,
    employees.department_id,
    departments.department_id,
    department_name
FROM
    employees
        INNER JOIN
    departments ON departments.department_id = employees.department_id
WHERE
    employees.department_id IN (1 , 2, 3);f
```

```
SELECT
        first_name,
        last_name,
        job_title,
        department_name
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
INNER JOIN jobs j ON j.job_id = e.job_id
WHERE
        e.department_id IN (1, 2, 3);
```

Left Join



A LEFT JOIN B

Full outer join

A

n
1
2
3

B

c
x
y
z

```
SELECT *
FROM A
CROSS JOIN B
```

A x B

n	c
1	x
1	y
1	z
2	x
2	y
2	z
3	x
3	y
3	z

Type	Great fit for	Watch out if
Replicated	✓ Small dimension tables in a star schema with less than 2 GB of storage after compression (~5x compression)	❑ Many DML operations are on table (such as insert, merge, delete, update) ❑ You change Data Warehouse Units (DWU) so frequently ❑ You only use 2-3 columns but your table has many columns ❑ You have index in replicated table
Round Robin (default)	✓ Temporary/staging table ✓ No obvious joining key or good candidate column	❑ Performance is slow due to data movement
Hash	✓ Fact tables ✓ Large dimension tables greater than 2 GB	❑ The distribution key cannot be updated

Subtitles/closed captions unavailable

Hash - use hash function , large fact table, table with frequent insert , update and delete
Round Robin - temporary , staging table - when you have no idea of hash key,
Replicated - for small dim tables,

SQL server Tutorial

Data types

NULL - absence of data, represents nothing ctrl+ 0 to insert null
Identity Column - numeric & auto increment - it should not be nullable - not reliable
Unique Key - cannot have duplicate , can have one null -
Primary Key - used to locate a record , want to locate record , no null , cannot be changed
Composite key - both are primary and their composite is unique

Foreign Key - to maintain referential integrity , Reference of one table from another -
we can put constraint on table using foreign key

Normalization - > we try to split the table depend on logic
Data should not be bad, redundant, and duplicate
One column should store only one value

Non Atomic - Pirates , Clash

Design mistake	Problem			
No primary key	Duplicate			
Atomic value	Duplicate, update ,delete			
Repeating groups	Redundant	1nf (key)	Only atomic values	

Partially depend on primary key	Redundant, Integrity	2nf (Full key)	No partial dependency, primary key	
Transitive dependency	Redundant, Integrity	3nf (only on the key)	No transient dependency	

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

1NF - single valued attribute , no duplicacy

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

2NF - FNF and no partial dependency (There should be single column primary key)

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

3NF - No transitive dependency (non-key column Full Name may change Salutation.)

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

DENORMALIZATION

	OLTP	OLAP
Design	Normalized. (1st normal form, second normal form and third normal form).	Denormalized (Dimension and Fact design).
Source	Daily transactions.	OLTP.
Motive	Faster insert, updates, deletes and improve data quality by reducing redundancy.	Faster analysis and search by combining tables.
SQL complexity	Simple and Medium.	Highly complex due to analysis and forecasting.

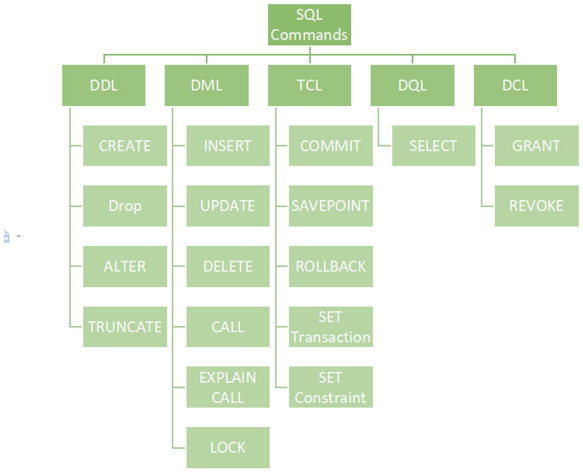
STAR and SNOW FLAKE DESIGN

Dimensions - Dimensions of data ,stored in dimension table
Measures - values, stored in fact table

Star design - pure denormalized
Snow flake - normalized

	Snowflake Schema	Star Schema
Normalization	Can have normalized dimension tables.	Pure denormalized dimension tables.
Maintenance	Less redundancy so less maintenance.	More redundancy due to denormalized format so more maintenance.
Query	Complex Queries due to normalized dimension tables.	Simple queries due to pure denormalized design.
Joins	More joins due to normalization.	Less joins.
Usage guidelines	If you are concerned about integrity and duplication.	More than data integrity speed and performance is concern here.

SQL command



DML

1. Order by c1,c2 - sort on c1+c2
2. Distinct c1,c2 - distinct on c1+c2
3. Like - % - any number of letter , _ - one lettter , ^ not
   select * from demo where Name like '[K]%' - values starting with K
   select * from demo where Name like '[^K]%' - values not starting with K
4.
5. Case -
   select *,
   case
   when CustomerId >= 100 and CustomerId< 105 then 'Medium'
   when CustomerId >= 105 and CustomerId< 110 then 'High'
   else 'NA'
   end as TRE
   from demo
6. Union - gives unique rows

select id, Name from demo
union
select CustomerId, CustomerName from Customer

7. Union all - includes duplicate rows also

Inner Join -

```
select  demo.id, demo.Name, demo.City, demo.CustomerId, Address.Address
from demo Inner join  Address on demo.CustomerId = Address.CustomerId
```

Left Join -

```
select  demo.id, demo.Name, demo.City, demo.CustomerId, Address.Address
from demo left join  Address on demo.CustomerId = Address.CustomerId
```

Right Join -

```
select  demo.id, demo.Name, demo.City, demo.CustomerId, Address.Address
from demo right join  Address on demo.CustomerId = Address.CustomerId
```

Full outer Join

```
select  demo.id, demo.Name, demo.City, demo.CustomerId, Address.Address
from demo full outer join  Address on demo.CustomerId = Address.CustomerId
```

Cross Join - all rows of both table permutation and combination
select * from demo cross join Address

Having - apply filter on group - >

Self Join - Join on same table

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

ISNULL
select isnull(CustomerPhone,-1) from demo

SUBQUERY

```
select * from demo
where id in (select CustomerCode from demo)
```

CORELATED QUERY - get first, second, third highest

```
select * from demo as d
where 1 = (select count(1) from demo d2 where d2.CustomerId > d.CustomerId )
```

BETWEEN () - both inclusive

Max, Min , Avg - aggregated process

SEELECT * INTO -
create new table with existing data without create syntax
select * into Customer from demo

```
Insert into Address(CustomerId, Address) values ( 105, 'Pune')
```

BULK insert

```
Insert into Address select * from customer
```

```
update Address set Address = 'Luknow' where CustomerId =101
```

```
Delete from Address
```

DDL

CREATE database Customer

- Creates 2 files mdf - Row data , ldf - log file

Create table Customer

```
create table Employee2
(
	id int unique,
	name nvarchar(50) unique not null
)
```

Alter table

```
Alter table Address
ADD Email varchar(255)
DROP COLUMN column_name
RENAME COLUMN old_name to new_name
ALTER COLUMN column_name datatype;
```

```
TRUNCATE TABLE Student_details;
```

Add Constraint

```
alter table Employee
add constraint constraint_salary2
 check (salary > 10000)
```

Default

```
add constraint def_const default 0 for dependents
```

Transaction

- set of tasks , all will get executed or all will be rolled back

```
begin tran
insert into Employee values(12,'fread',45000,0)
insert into Employee values(11,'fread',45000,0)
if(@@ERROR > 0)
begin
	rollback tran
end
begin
	commit tran
end
```

@@error - global variable
@@trancount - gives the current transaction count

Concurrency

A. By default the transaction of SQL server are locked at **row level. Only the Particular row is locked. The user cannot see the volatility of transaction.**

```
begin transaction
	update Employee set name='Vikas' where id=7
	waitfor delay '00:00:10'
	update Employee set name='Ramesh' where id=7
rollback transaction
```

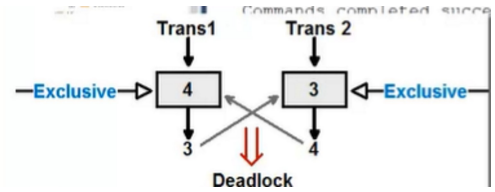
When you try to query data in between transaction
, query will go in wait state and you wont see any output
Until the transaction is complete

B. If you don't want to go in blocking mode, you want to see volatility of data,
Then you can use nolock.

```
select * from Employee with (nolock)
```

Blocking and deadlock

A deadlock occurs when two or more processes or transactions block each other from continuing because each has locked a database resource that the other transaction needs. SQL Server handles deadlocks by terminating and rolling back transactions that were started after the first transaction.



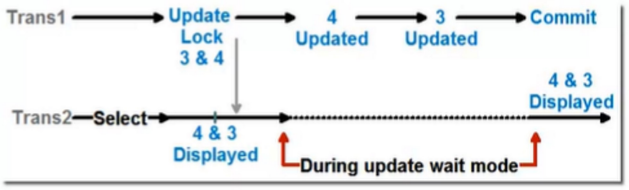
```
--transaction 1
begin transaction
update Employee set name='kada' where id=5
waitfor delay '00:00:20'
update Employee set name='Vada' where id=6
commit transaction


--transaction 2
begin transaction
update Employee set name='kada' where id=6
waitfor delay '00:00:20'
update Employee set name='Vada' where id=5
commit transaction
```

How to avoid deadlock -
Keep the time small
Maintain same chronological order

Updlock -
It will lock all the locks in deadlock mode

```
begin transaction
select * from Employee with (updlock) where id=5 or id =6
update Employee set name='kada' where id=5
waitfor delay '00:00:20'
update Employee set name='Vada' where id=6
commit transaction
```



Shared lock

it is applied only during select query
Update tblCustomer set name ='sometax' where id=4

When we execute any update query in sql server, internally it applies all 3 locks

Select (shared)
Calculation (update lock)
Actual update (exclusive lock)

Lock can be applied on
Rows - Pages - Table - DB

spid	dbid	ObjId	IndId	Type	Resource	Mode	Status
51	5	0	0	DB		S	GRANT
52	5	0	0	DB		S	GRANT
52	1	1467152272	0	TAB		IS	GRANT
52	32767	-571204656	0	TAB		Sch-S	GRANT
54	5	0	0	DB		S	GRANT
54	5	1029578706	0	TAB		IX	GRANT
54	5	1029578706	1	PAG	0.313888889	IX	GRANT
54	5	1029578706	1	KEY	(a0c936a3c965)	X	GRANT

I - Intent
IS - Intent shared lock
IX - Intent Exclusive lock

ISOLATIONAL LEVEL

Its very important when we have multiple transactional in sql server

Problems with nolock
1. Repeatable read -

We see new data in different reads within same transaction

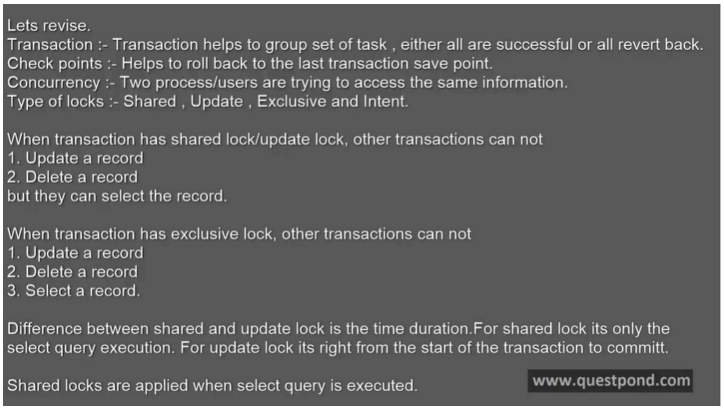
2. Dirty read
   We see data which never got committed.

3. Phantom read (Ghost)
   We don't see data at first time , but the next time we see that record

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

ACID

Atomicity - All transaction should commit or roll back
Begin tran , commit tran
Consistency - transaction should create a valid state of new data
Save tran
Isolation - transaction must be isolated from other transaction
Isolation level
Durability - commited data must be stored in correct state , back up
Back up, mirroring



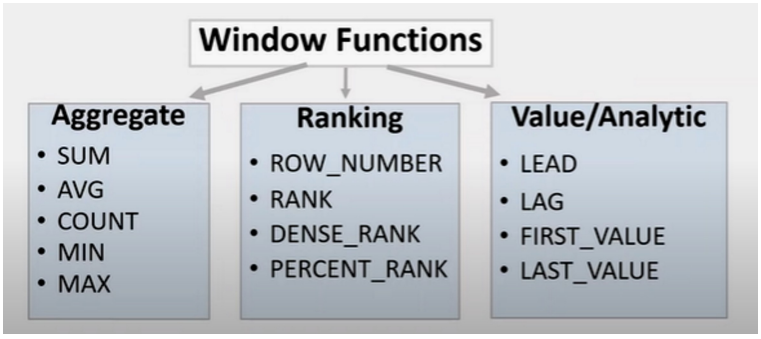
---

1. Isnull - can take 2 column
   select isnull(Name, 'Vada') as t from demo
2. Coalesce - returns the first non null column from the list of column
   Select coalesce(FirstName, lastName, PetName) as Name from tblPerson

---

Windows Function

A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result



3.
4. row_number() - generates a unique numbering for each row in a column

5. Partition - if we want different row numbering for different groups , then we use partition
6. Rank() - generates same number for same records and do numbering ,but skips the continuous numbers
7. Dense_rank()- generates same number for same records and do numbering ,but skips the continuous numbers

```
select ROW_NUMBER() over (order by customer) as row_num,
ROW_NUMBER() over (partition by Vendor order by Vendor) as row_num_partition,
rank() over (order by customer) as rank_customer,
dense_rank() over (order by customer) as dense_rank_customer,
* from productDetails
```

row_num	row_num_partition	rank_customer	dense_rank_customer	customer	Product	Amount	Vendor
1	1	1	1	Abhay	Bag	200	Atlas
6	2	6	4	Vasu	Pant	300	Atlas
2	1	2	2	Amar	Shoes	100	Data
4	1	4	3	Ramesh	Shirt	150	Polo
5	2	4	3	Ramesh	Shirt	250	Polo
3	1	2	2	Amar	Shoes	100	Reebook

```
select * from Employee
-- Get total aggregates
select id , name,
sum(id) over () as 'total',
avg(id) over () as 'avg' ,
count(id) over () as 'count',
min(id) over () as 'min'
from Employee
```

	id	name	total	avg	count	min
1	1	Ajay	66	6	11	1
2	2	Vikas	66	6	11	1
3	3	Kukas	66	6	11	1
4	4	Freak	66	6	11	1
5	5	Ful	66	6	11	1
6	6	Ful	66	6	11	1
7	7	fread	66	6	11	1
8	8	fread	66	6	11	1
9	9	fread	66	6	11	1
10	10	fread	66	6	11	1
11	11	fread	66	6	11	1

```
-- Get partition by aggregates
select id , name,
sum(id) over (partition by name) as 'total',
avg(id) over (partition by name) as 'avg' ,
count(id) over (partition by name) as 'count',
min(id) over (partition by name) as 'min'
from Employee
```

	id	name	total	avg	count	min
1	1	Ajay	1	1	1	1
2	7	fread	45	9	5	7
3	8	fread	45	9	5	7
4	9	fread	45	9	5	7
5	10	fread	45	9	5	7
6	11	fread	45	9	5	7
7	4	Freak	4	4	1	4
8	5	Ful	11	5	2	5
9	6	Ful	11	5	2	5
10	3	Kukas	3	3	1	3
11	2	Vikas	2	2	1	2

```
-- Get Running aggregates
select id , name,
sum(id) over (order by id) as 'total',
avg(id) over (order by id) as 'avg' ,
count(id) over (order by id) as 'count',
min(id) over (order by id) as 'min'
from Employee
```

	id	name	total	avg	count	min
1	1	Ajay	1	1	1	1
2	2	Vikas	3	1	2	1
3	3	Kukas	6	2	3	1
4	4	Freak	10	2	4	1
5	5	Ful	15	3	5	1
6	6	Ful	21	3	6	1
7	7	fread	28	4	7	1
8	8	fread	36	4	8	1
9	9	fread	45	5	9	1
10	10	fread	55	5	10	1
11	11	fread	66	6	11	1

```
-- Get partitioned running aggregates
select id , name,
sum(id) over (partition by name order by id) as 'total',
avg(id) over (partition by name order by id) as 'avg' ,
count(id) over (partition by name order by id) as 'count',
min(id) over (partition by name order by id) as 'min'
from Employee
```

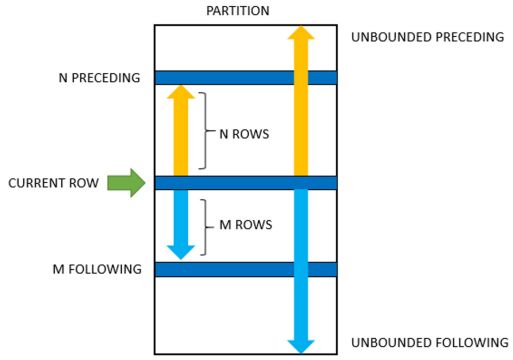
	id	name	total	avg	count	min
1	1	Ajay	1	1	1	1
2	7	fread	7	7	1	7
3	8	fread	15	7	2	7
4	9	fread	24	8	3	7
5	10	fread	34	8	4	7
6	11	fread	45	9	5	7
7	4	Freak	4	4	1	4
8	5	Ful	5	5	1	5
9	6	Ful	11	5	2	5
10	3	Kukas	3	3	1	3
11	2	Vikas	2	2	1	2

```
-- Get Ranking Functions
select id , name,
row_number() over (order by id) as 'row_number',
rank() over (order by id) as 'rank' ,
dense_rank() over (order by id) as 'dense_rank',
PERCENT_RANK() over (order by id) as 'PERCENT_RANK'
from Employee
```

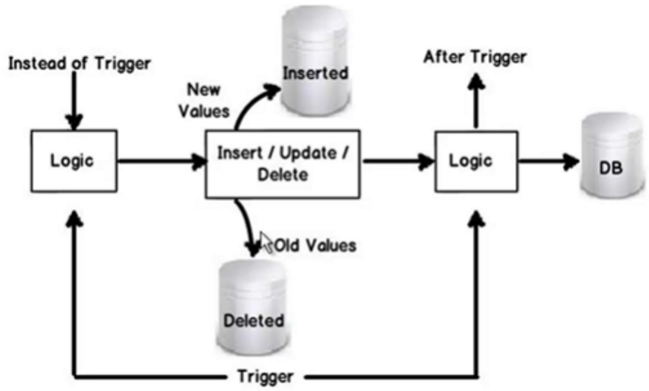
	id	name	row_number	rank	dense_rank	PERCENT_RAN
1	1	Ajay	1	1	1	0
2	2	Vikas	2	2	2	0.1
3	3	Kukas	3	3	3	0.2
4	4	Freak	4	4	4	0.3
5	5	Ful	5	5	5	0.4
6	6	Ful	6	6	6	0.5
7	7	fread	7	7	7	0.6
8	8	fread	8	8	8	0.7
9	9	fread	9	9	9	0.8
10	10	fread	10	10	10	0.9
11	11	fread	11	11	11	1

```
-- Get Analytic Functions
select id , name,
first_value(id) over (order by id) as 'first_value',
last_value(id) over (order by id ROWS BETWEEN UNBOUNDED PRECEDING  AND UNBOUNDED FOLLOWING ) as 'last_value' ,
lead(id) over (order by id) as 'lead',
lag(id) over (order by id) as 'lag'
from Employee
```

	id	name	first_value	last_value	lead	lag
1	1	Ajay	1	11	2	NULL
2	2	Vikas	1	11	3	1
3	3	Kukas	1	11	4	2
4	4	Freak	1	11	5	3
5	5	Ful	1	11	6	4
6	6	Ful	1	11	7	5
7	7	fread	1	11	8	6
8	8	fread	1	11	9	7
9	9	fread	1	11	10	8
10	10	fread	1	11	11	9
11	11	fread	1	11	NULL	10



Triggers , inserted, deleted tables



Trigger -

```
CREATE TRIGGER trigger
  ON  triggerCustomer
  AFTER  INSERT,DELETE,UPDATE
AS
BEGIN
        Insert into triggerAudit values (GETDATE())
END
GO
```

Inserted & Deleted table

1) Temporary table created by SQL server
2) Table structure is same as original table

Query for getting trigger in SSMS

```
SELECT     o.[name],
         c.[text]
FROM       sys.objects AS o
INNER JOIN  sys.syscomments AS c
ON      o.object_id = c.id
WHERE   o.[type] = 'TR'
```

Declare variables

Inserted and deleted table

```
ALTER TRIGGER [dbo].[TriggerUpdate]
  ON [dbo].[triggerCustomer]
  AFTER INSERT,DELETE,UPDATE
AS
BEGIN

        declare @OldValue varchar(50)
        declare @NewValue varchar(50)

        --fetch old value
        select @OldValue =customerName from deleted
        --fetch new value
```

```
        select @NewValue=customerName from inserted

    -- Insert statements for trigger here
        Insert into triggerAudit(LastUpdatedDate,OldName, NewName) values (GETDATE(),@OldValue , @NewValue )
END
```

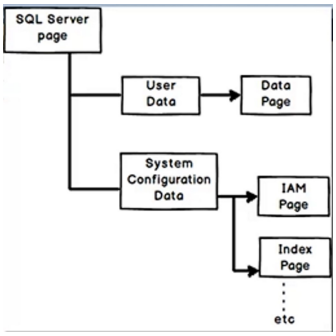
INSTEAD OF AND AFTER TRIGGER

After trigger - happens after the operation on main table
Instead of - happens before the operation on main table and it happens instead of Insert, update and delete

---

SQL Server 8 KB page

When you write or read, SQL server does it into the 8 kb page.



Data page - User data
IAM page - Index allocation map (it stores info about all pages)

To get the page list
DBCC IND('InvoiceDb',Employee, -1)

PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType	IndexLevel	NextPageFID	NextPagePID	PrevPageFID	PrevPagePID
1	385	NULL	NULL	3.38E+08	0	1	7.21E+16	In-row data	10	NULL	0	0	0	0
1	400	1	385	3.38E+08	0	1	7.21E+16	In-row data	1	0	0	0	0	0

Page 10 - IAM page
Page 1 - data page

To get page details

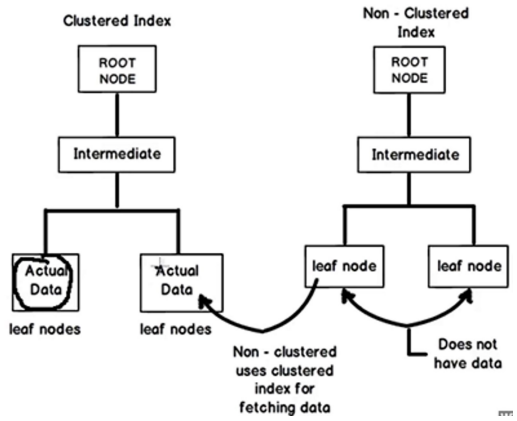
DBCC TRACEON(3604)
DBCC PAGE('InvoiceDb',1,400,1)

INDEX and Performances

Index - makes search faster by making binary tree structure. But it slows down performance of insert , update and delete

Page split and Index

Clustured and Non clustured index -



	char	nchar	varchar	nvarchar
Character Data Type	Non-Unicode fixed-length	Unicode fixed-length can store both non-Unicode and Unicode characters (i.e. Japanese, Korean etc.)	Non-Unicode variable length	Unicode variable length can store both non-Unicode and Unicode characters (i.e. Japanese, Korean etc.)
Maximum Length	up to 8,000 characters	up to 4,000 characters	up to 8,000 characters	up to 4,000 characters
Character Size	takes up 1 byte per character	takes up 2 bytes per Unicode/Non-Unicode character	takes up 1 byte per character	takes up 2 bytes per Unicode/Non-Unicode character
Storage Size	n bytes	2 times n bytes	Actual Length (in bytes)	2 times Actual Length (in bytes)
Usage	use when data length is constant or fixed length columns	use only if you need Unicode support such as the Japanese Kanji or Korean Hangul characters due to storage overhead	used when data length is variable or variable length columns and if actual data is always way less than capacity	use only if you need Unicode support such as the Japanese Kanji or Korean Hangul characters due to storage overhead
			query that uses a varchar parameter does an index seek due to column collation sets	query that uses a nvarchar parameter does an index scan due to column collation sets

Stored Procedure -

Using the cache
Centralized query

It has 3 parameters

```
CREATE procedure selectCustomer
 @CustomerName nvarchar(100),  --input parameter
 @CurrentDate Datetime output --output parameter

AS
BEGIN
 set @CurrentDate = GETDATE()
 select * from demo where name = @CustomerName
 return @@rowcount    --return parameter
END
```

___

```
declare @DatetimeCurrent datetime
declare @myrowcount int
execute @myrowcount = selectCustomer 'Vaibhav', @DatetimeCurrent output
select  @DatetimeCurrent
print @myrowcount
```

___

CTE

CTE is a temporary result set which can be used in
subsequent execution scope only once.
In CTE we can use recursive query - self referencing - employee manager hierarchy

	id	name	managerId	sales
1	1	Jitendra	0	500
2	2	Saurabh	1	100
3	3	Rahul	1	300
4	4	Shoumik	1	300
5	5	Mounika	4	220
6	6	Kalyani	3	150

For single recursion

```
select z.manager, count(1), sum(z.sales)
from
(select A.id managerId, A.name  manager, B.id, B.name , B.sales
from SaleMap A , SaleMap B  where A.id= B.managerId )Z group by z.manager
```

	manager	(No column name)	(No column name)
1	Jitendra	3	700
2	Mudit	1	500
3	Rahul	1	150
4	Shoumik	1	220

For multiple recursion

```
WITH UserCTE
    AS (SELECT id,
            name,
            managerId,
                            sales,
            0 AS EmpLevel
        FROM   SaleMap
        WHERE  managerId = 0
        UNION ALL
        SELECT usr.id,
            usr.name,
            usr.managerId,
                            usr.sales,
            mgr.[EmpLevel] + 1
        FROM   SaleMap AS usr
            INNER JOIN UserCTE AS mgr
                ON usr.managerId = mgr.id
        WHERE  usr.managerId IS NOT NULL)
SELECT *
FROM   UserCTE AS u
ORDER  BY EmpLevel;
```

Can we perform insert , update and delete in cte - yes and it will also affect the physical table

Temporary table

Temporary table are physical table which gets created for one session.
Once session is closed table is deleted.

CTE	Temporary table
Created in RAM	Created in hard disk
Lifetime - only for one subsequent execution	Lifetime - till the connection is closed
Used for recursive call	Store temp data

```
create table #temporary
(
name nvarchar(50) not null
)
```

SUBQUERY and Corelated Query

```
Select * from employee where id = (select id from salary)

select * from demo as d
where 1 = (select count(1) from demo d2 where d2.CustomerId > d.CustomerId )
```

JOIN vs Subquery

Subquery - series of filtering criterion
Join - select common part

Each query will be distributed into 60 distributions
Distribution -

Hash -using hash function - very large tables
Round_Robin - circular distribution - staging tables
Replicated - data in all nodes - small tables

Index option
Row store - transactional indexes
1. Heap - table without clustered index, no order, faster index,

Column store - analytics

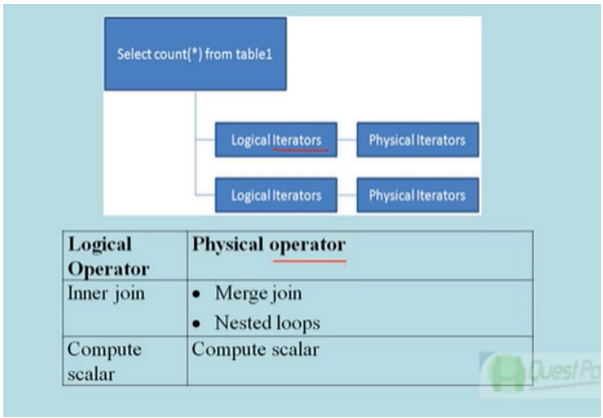
Heap - base row store
Clustured Index - Base row store maintained as B-Tree
Clustured columnstore index - base column stored

Partitions ,indexes , distributions,
Partition are created on date column - queries will generally take recent data - mini dataset

What is polybase , copy, bulk insert in adf ,

SQL Performance tuning

Iterator/ operator - logic for execution

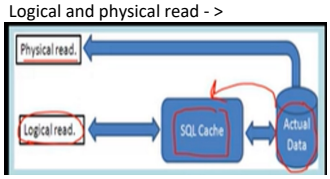


Logical Operator	Physical operator
Inner join	• Merge join • Nested loops
Compute scalar	Compute scalar

Query execution

Check syntax - > Query processor tree -> Data statistics -> Estimated plan -> execute -> Actual plan

Logical and physical read - >



Logical read - read from memory
Physical read - read from actual data

How to find logical and physical read -
set statistics io on
SELECT * from demo

1. Unique key improves table scan performance. Always create primary key on table.

2. Choose table scan for small records and seek scan for large records.
   Table scan - row wise scan
   Seek scan - B Tree scan - only with indexes

3. Use covering index to reduce row id lookup (RID)
   Both clustured and non clustured index has B Tree structure

   Clustured indexes - > leaf node point to actual data - >
   Clustered indexes physically order the data on the disk. - > primary key

   Non clustured index - > leaf node point to RID
   Heap table - table without indexes
   Use composite key

4. Keep index size as small as possible

   Keep pages less - >

_____

Data in SQL server is stored as tree structure based on clustered Index(key).

Table scan - if you want to get data by column other than clustered index, you need to perform whole table scan.

Index - it contains data pointer in sorted order basis of specific column..

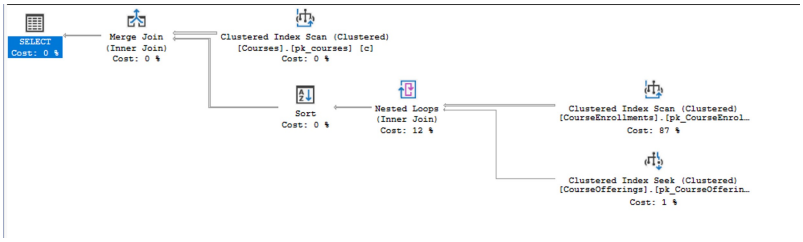
While searching on index- we need to perform 2 operations , index search and seed search.

Estimated execution plan - occurs right to left and top to bottom

Query - >

```
select c.DepartmentCode, c.CourseNumber, c.CourseTitle,c.Credits,ce.grade
from CourseEnrollments ce
inner join CourseOfferings co on co.CourseOfferingId = ce.CourseOfferingId
inner join Courses c on co.DepartmentCode = c.DepartmentCode and co.CourseNumber = c.CourseNumber
where ce.StudentId = 29717
```

1. Estimated execution plan



It doesn't actually execute statement.
Read from right to left , top to bottom.

2. Get Statistics

To get detailed IO and CPU statistics about a query, need to actually execute query

set statistics io on
set statistics time on

Result
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'CourseOfferings'. Scan count 0, logical reads 91, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'CourseEnrollments'. Scan count 1, logical reads 12023, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Courses'. Scan count 1, logical reads 7, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

SQL Server Execution Times:
CPU time = 78 ms, elapsed time = 92 ms.

Logical read - reading each 8 kb page in sql server - minimize it

3. Adding an Index

   Create Index on studentId in courseEnrollments, it reduces performance drastically.

   Point of comparison
- Data access operation
- Statement cost - Subtree cost
- Logical read
- CPU time
- Elapsed time

4. Rewrite Sql statement
   Use exists, not exists to get out of matching loop.

5. Common execution plan operations

   Clustered Index scan -> Read all rows as clustered index
   Table scan -> Read all rows as heap (Normal reading)
   Clustered index Seek - traverse tree structure of table as clustered index to find matching row
   Index scan - Read all key values to find matching data
   Index seek - Traverse tree structure of index as clustered index to find matching row

   Reduce scan and increase seek

   Nested loop join -
   Merge Join -
   Hash Join -

6. Build effective indexes -

Command -

```
CREATE INDEX IX_Students_State
on Students (State);
```

Clustered index - primary key
Non Clustered index - other than primary key

Add Indexes for where clause
Primary key are indexed by default in SQL server
Index foreign key
For combination of column as index, include the most frequently used column first
Make selective index - try to get unique combination of columns with least records, preferably 1.

Hint - force SQL server to use some feature, don't use this as optimizer is already smart

LIKE - for prefix %, SQL server will not be able to use indexes as indexes are based on sorting logic. Always specify some minimum characters in LIKE search

FUNCTION operating on column value - not able to use the index, create a computed column for function

INCLUDE column - > it will stored the column values in index, creating covering index
Covering index - to reduce key lookup operation, it will get all data from index seek only

```
CREATE INDEX IX_Students_Email
on Students (Email)
INCLUDE (FIrstName, LastName) ;

select Email, FirstName, LAstName from Students
where Email = 'PaulLWilson@superrito.com'
```

7. Over Indexing
   Why not create index on every column ?
   Index has maintenance cost , for DML (insert, update, delete) it will have to update the index.
   Index are investments. Invest wisely.

   Dynamic management views

8. Finding Performance bottleneck

   SQL server is always collecting data.
   DMV - dynamic management views

   Essentials-of-Sql-Server-Performance-for-Every-Developer/Exercise Files/Module 4/Part 4/DMV.txt at master · iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer · GitHub

1. Getting Information about the sessions.

```
SELECT
   database_id,   -- SQL Server 2012 and after only
   session_id,
   status,
   login_time,
   cpu_time,
   memory_usage,
   reads,
   writes,
   logical_reads,
   host_name,
   program_name,
   host_process_id,
   client_interface_name,
   login_name as database_login_name,
   last_request_start_time
FROM sys.dm_exec_sessions
WHERE is_user_process = 1
ORDER BY cpu_time DESC;
```

2. What SQL statements are currently executing

```
-- Finding statements running in the database right now (including if a statement is blocked by another)
-- ------------------------------------------------------------------------------------------------
SELECT
      [DatabaseName] = db_name(rq.database_id),
      s.session_id,
      rq.status,
      [SqlStatement] = SUBSTRING (qt.text,rq.statement_start_offset/2,
         (CASE WHEN rq.statement_end_offset = -1 THEN LEN(CONVERT(NVARCHAR(MAX),
         qt.text)) * 2 ELSE rq.statement_end_offset END - rq.statement_start_offset)/2),
      [ClientHost] = s.host_name,
      [ClientProgram] = s.program_name,
      [ClientProcessId] = s.host_process_id,
      [SqlLoginUser] = s.login_name,
      [DurationInSeconds] = datediff(s,rq.start_time,getdate()),
      rq.start_time,
      rq.cpu_time,
      rq.logical_reads,
      rq.writes,
      [ParentStatement] = qt.text,
      p.query_plan,
      rq.wait_type,
      [BlockingSessionId] = bs.session_id,
      [BlockingHostname] = bs.host_name,
      [BlockingProgram] = bs.program_name,
      [BlockingClientProcessId] = bs.host_process_id,
      [BlockingSql] = SUBSTRING (bt.text, brq.statement_start_offset/2,
         (CASE WHEN brq.statement_end_offset = -1 THEN LEN(CONVERT(NVARCHAR(MAX),
         bt.text)) * 2 ELSE brq.statement_end_offset END - brq.statement_start_offset)/2)
   FROM sys.dm_exec_sessions s
   INNER JOIN sys.dm_exec_requests rq
      ON s.session_id = rq.session_id
   CROSS APPLY sys.dm_exec_sql_text(rq.sql_handle) as qt
   OUTER APPLY sys.dm_exec_query_plan(rq.plan_handle) p
   LEFT OUTER JOIN sys.dm_exec_sessions bs
      ON rq.blocking_session_id = bs.session_id
   LEFT OUTER JOIN sys.dm_exec_requests brq
      ON rq.blocking_session_id = brq.session_id
   OUTER APPLY sys.dm_exec_sql_text(brq.sql_handle) as bt
   WHERE s.is_user_process =1
      AND s.session_id <> @@spid
 AND rq.database_id = DB_ID()  -- Comment out to look at all databases
   ORDER BY rq.start_time ASC;
```

3. Find the slowest , Most expensive query

```
-- Finding the most expensive statements in your database
-- ------------------------------------------------------------------------------------------------
SELECT TOP 20
      DatabaseName = DB_NAME(CONVERT(int, epa.value)),
      [Execution count] = qs.execution_count,
      [CpuPerExecution] = total_worker_time / qs.execution_count ,
      [TotalCPU] = total_worker_time,
      [IOPerExecution] = (total_logical_reads + total_logical_writes) / qs.execution_count ,
      [TotalIO] = (total_logical_reads + total_logical_writes) ,
      [AverageElapsedTime] = total_elapsed_time / qs.execution_count,
      [AverageTimeBlocked] = (total_elapsed_time - total_worker_time) / qs.execution_count,
    [AverageRowsReturned] = total_rows / qs.execution_count,
    [Query Text] = SUBSTRING(qt.text,qs.statement_start_offset/2 +1,
         (CASE WHEN qs.statement_end_offset = -1
            THEN LEN(CONVERT(nvarchar(max), qt.text)) * 2
            ELSE qs.statement_end_offset end - qs.statement_start_offset)
         /2),
      [Parent Query] = qt.text,
      [Execution Plan] = p.query_plan,
    [Creation Time] = qs.creation_time,
    [Last Execution Time] = qs.last_execution_time
   FROM sys.dm_exec_query_stats qs
   CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as qt
   OUTER APPLY sys.dm_exec_query_plan(qs.plan_handle) p
   OUTER APPLY sys.dm_exec_plan_attributes(plan_handle) AS epa
   WHERE epa.attribute = 'dbid'
      AND epa.value = db_id()
   ORDER BY [AverageElapsedTime] DESC; --Other column aliases can be used-- Finding the most expensive statements in your database
-- ------------------------------------------------------------------------------------------------
SELECT TOP 20
      DatabaseName = DB_NAME(CONVERT(int, epa.value)),
      [Execution count] = qs.execution_count,
      [CpuPerExecution] = total_worker_time / qs.execution_count ,
      [TotalCPU] = total_worker_time,
```

```
    [IOPerExecution] = (total_logical_reads + total_logical_writes) / qs.execution_count ,
    [TotalIO] = (total_logical_reads + total_logical_writes) ,
    [AverageElapsedTime] = total_elapsed_time / qs.execution_count,
    [AverageTimeBlocked] = (total_elapsed_time - total_worker_time) / qs.execution_count,
  [AverageRowsReturned] = total_rows / qs.execution_count,
  [Query Text] = SUBSTRING(qt.text,qs.statement_start_offset/2 +1,
       (CASE WHEN qs.statement_end_offset = -1
          THEN LEN(CONVERT(nvarchar(max), qt.text)) * 2
          ELSE qs.statement_end_offset end - qs.statement_start_offset)
       /2),
    [Parent Query] = qt.text,
    [Execution Plan] = p.query_plan,
  [Creation Time] = qs.creation_time,
  [Last Execution Time] = qs.last_execution_time
  FROM sys.dm_exec_query_stats qs
  CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as qt
  OUTER APPLY sys.dm_exec_query_plan(qs.plan_handle) p
  OUTER APPLY sys.dm_exec_plan_attributes(plan_handle) AS epa
  WHERE epa.attribute = 'dbid'
    AND epa.value = db_id()
  ORDER BY [AverageElapsedTime] DESC; --Other column aliases can be used
```

4. Getting SQL Server Recommendation

```
-- Looking for Missing Indexes
-- ------------------------------------------------------------------------------------------
SELECT
   TableName = d.statement,
   d.equality_columns,
   d.inequality_columns,
   d.included_columns,
   s.user_scans,
   s.user_seeks,
   s.avg_total_user_cost,
   s.avg_user_impact,
   AverageCostSavings = ROUND(s.avg_total_user_cost * (s.avg_user_impact/100.0), 3),
   TotalCostSavings = ROUND(s.avg_total_user_cost * (s.avg_user_impact/100.0) * (s.user_seeks + s.user_scans),3)
FROM sys.dm_db_missing_index_groups g
INNER JOIN sys.dm_db_missing_index_group_stats s
   ON s.group_handle = g.index_group_handle
INNER JOIN sys.dm_db_missing_index_details d
   ON d.index_handle = g.index_handle
WHERE d.database_id = db_id()
ORDER BY TableName, TotalCostSavings DESC;
```

SQL best practices

1. Create Execution plan
2. Monitor resource usage
3. Use SQL DMV
4. Select column instead of select *
5. Avoid select Distinct
6. Avoid cartesian join
7. Avoid prefix like , use suffix like always
8. Use limit for sampling query result
9. Function in sql

Execution plan

1. Estimated execution plan - before execution
2. Actual execution plan - after execution
3. Live query statistics - on execution

You can download the execution plan in xml, graphical, text

_____

SQL server profiler - > try to get tunning recommendation

SQL server

Non clustered index takes help of clustered index to get data

Data page and index page

DBCC - database consistency checker

Type 1 - data page

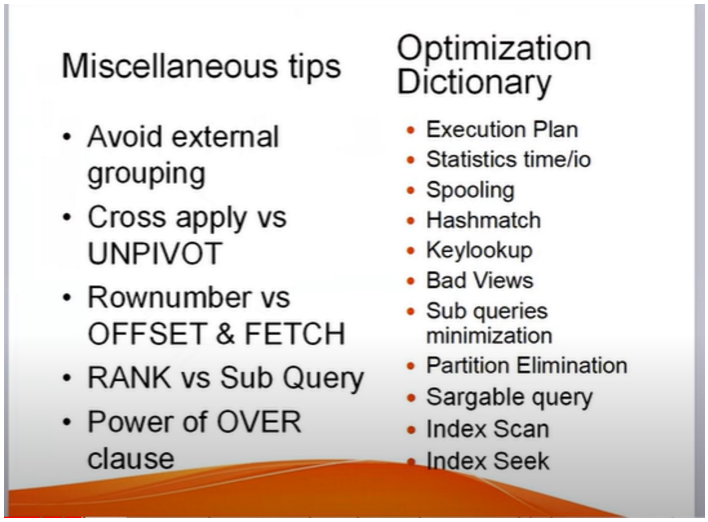
Type 2 - index page

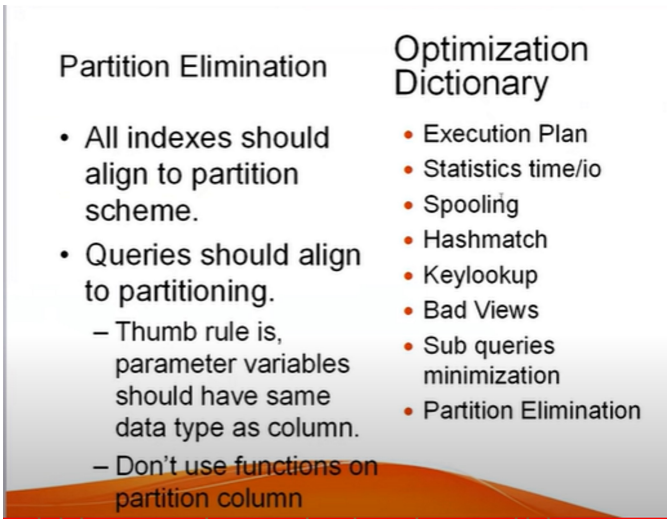
8kb page

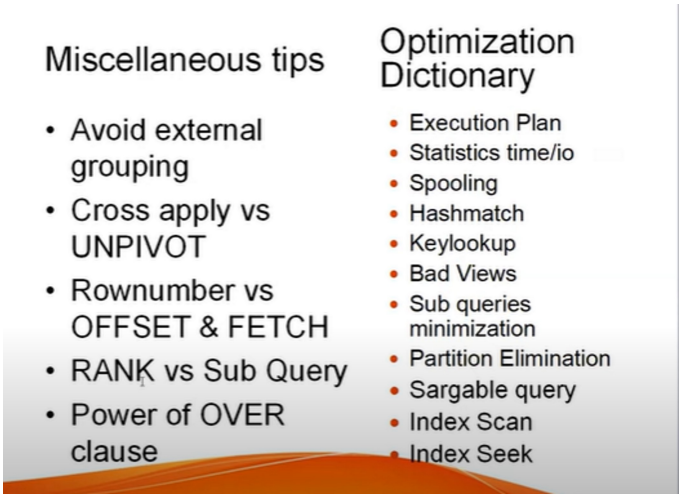
SET STATISTICS io, time on

Logical read - important -

Spooling -
lazy spooling - SQL Server will store the data ,
occurs due to duplicate aggregation , Recursive cte have lazy spooling

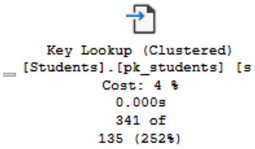






HashMatch -
Unsorted data
Group by - it will do order by first
Solution - use index, dont use function in where clause where indexed column is present

Key Lookup - due to missing column in index
Add the column to index - use include




```
CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]
ON [dbo].[Students] ([DateOfBirth])
INCLUDE ([FirstName])
```

Data type should be same when we are comparing in sql.

Avoid function in where clause as much as possible

Avoid external group

Partition Elimination - ?
CREATE PARTITION FUNCTION

No of executions

No of rows read

Thick lines - more rows
Narrow lines - low rows

Start with thick lines
Lookup
Spool
Sort
Hash
Nested loops

Index scan -

Lookup
RID lookup (Heap) -
Key lookup (Clustered) -

Spools -
cache in query processor, implemented as hidden table in tempdb,
Table spool (Lazy)
Index spool (Eager)
Table spool (Eager)

Lack of adequate index or unique information

Sort -
Sorting is required for - merger join, order by, merge join, stream agg, window function,
Sort is expensive , use effectively

Hash
Build a hash table
Hash Match (join)
Hash Match (aggregation)

Nested loops -

Scan -
Table scan -
Index scan-
Clustered index scan -

1. Recursive cte
2. Self join
3. Join and then select in apache spark
4. CROSS APPLY , writing SP, FUNCTIONS, Table valued function,

Table valued function

```
Create function fn_getEmployeeByCity(@City varchar(50))
returns table
as
return
(
select * from Employees where  City = @City
)
```

APPLY - Join for table valued function.

“The APPLY operator is similar to the JOIN operator, but the difference is that the right-hand side operator of APPLY can reference columns from the left-hand side”.

CROSS APPLY
Ex. Find the last 3 orders from all customers that live in the UK.

```
select e1.EmployeeID,e2.FirstName from Employees e1
cross apply  fn_getEmployeeByCity(e1.city) e2
where  e1.City = e2.City
```

OUTER APPLY

```
CREATE TABLE Table1 WITH (DISTRIBUTION = HASH(c1), CLUSTERED COLUMNSTORE INDEX ORDER(c1) )
AS SELECT * FROM ExampleTable
OPTION (MAXDOP 1);
```

Check the index of table -

```
select * from sys.indexes
where object_id in (select object_id from sys.objects where name = 'logs_rapidlr')
```

Check Statistics

```
SELECT *
FROM sys.stats
WHERE object_id = OBJECT_ID('fact_loan');
```

```
DBCC SHOW_STATISTICS ( 'dbo.dim_loan'  ,IDX_DIM_LOAN_APPL_ID )
```

Check size of data

```
exec sp_spaceused 'dbo.Applicants'
```

Check SP syntax

```
----
SELECT *
FROM sys.sql_modules
WHERE object_id = (OBJECT_ID(N'dbo.uspLogError'));

------
CREATE PROC [dbo].[CD] @schema_name [varchar](500),
@tablename [varchar](500)
AS BEGIN
DECLARE @OBJECT_ID INT
SELECT
  @OBJECT_ID = OBJECT_iD(@schema_name + '.' + @tablename)
select
  *
from
  sys.objects
WHERE
  OBJECT_iD = @OBJECT_ID
select
  type_desc,
  name CCI_Name
```

```
from
  sys.indexes
where
  OBJECT_ID = @OBJECT_ID
  and type_desc in ('CLUSTERED COLUMNSTORE', 'HEAP');
with cdp as (
  select
    *
  FROM
    sys.pdw_column_distribution_properties
  WHERE
    [object_id] = OBJECT_iD(@schema_name + '.' + @tablename)
)
select
  c.name distribution_column_name
FROM
  cdp
  inner JOIN sys.columns c ON cdp.[object_id] = c.[object_id]
  AND cdp.[column_id] = c.[column_id]
where
  distribution_ordinal = 1
```
