1,N''+null=null

2,在业务不确定会产生几条数据的时候,禁止用TOP 1 取数据,将位置的错误暴露出来,以便于及时修复

--3,两个or条件,查出来的数据有同一条,不会重复显示,下例子   
drop table tmp1  
CREATE TABLE TMP1 (cid int ,type1 int ,type2 int, name nvarchar(50))  
insert into TMP1  
select 1,1,1,'chen1'  
union all  
select 1,1,1,'chen2'  
union all  
select 1,0,2,'chen3'  
union all  
select 1,1,2,'chen4'  
union all  
select 1,0,1,'chen5'  
   select \* from TMP1 where type1=1 or type2=2

4,自动备份自动删除(建立作业)

--每小时备份一次

declare  @sql  varchar(100)

set @sql='D:\BACKUP\clms\_backup\_small\'+RTRIM(REPLACE(REPLACE(CONVERT(varchar,getdate(),120),' ','\_'),':',''))+'.bak'

backup database clms to disk=@sql

GO

--每天执行一次(删除3天后的数据)

DECLARE @OLDDATE DATETIME

SELECT @OLDDATE=GETDATE()-3

EXECUTE master.dbo.xp\_delete\_file 0,N'D:\BACKUP\clms\_backup\_small',N'bak',@olddate,1

5,查询新插入的ID,用select scope\_Identity(),比用@@identity 更准确

6,分页查询

用这个row\_number() over(order by Temp.UserID)…where UserInfo.ID>=11 and UserInfo.ID<=15 效率比较快

7, 找到所有重复项

create table table1(name1 varchar(500),id int)

insert into table1 values('a',1)

insert into table1 values('b',2)

insert into table1 values('b',2)

insert into table1 values('b',2)

select \* from table1

select \* from

(

select row\_number() over(partition by name1,id order by name1) IDs,\* from table1

) Temp

where Temp.ID>1

8,写存储过程时,不需要返回受影响行数的信息时,写SET NOCOUNT ON关闭.它还可以防止执行存储过程时,锁住该表

9,当索引结构发生变化时，在EXECUTE语句中(第一次)使用WITH RECOMPILE子句，以便存储过程可以利用最新创建的索引;

10,统计表的行数(性能更好):SELECT rows FROM sysindexes WHERE id =OBJECT\_ID('dbo.aspnet\_Users') AND indid <2

11,空的pk值串联会影响查询速度

12, select datediff(DD,'2016-1-1','2016-1-5') 时间间隔

13, 索引都是为了提高查询速度的，索引一般添加到不是频繁改动的字段上。聚集索引影响排序，非聚集索引不影响排序。

14,跨服务器查询

Select \* from openrowset( 'SQLOLEDB', '10.100.254.101'; 'sa'; 'bjcit2015.qwe',BJC\_FinanceTest.dbo.ISBG\_DeliveryCityEnum)

15,sql闲聊

一开始是数据量慢慢大了查询特别慢，于是在**不经常修改又经常使用的列**建立了**索引**，等差不多表里有100w左右的数据了，开始有点吃不消了，于是就有了**分表技术**。分表技术很多，hashcode取余，路由表等等。。。刚开始就是伪分表，也就是传说中的**水平分表**，还是在一个数据库里面，**主要目的就是为了解决ID溢出或者单个表数据太多而导致查询太慢**。

后来还是有点吃不消，总不能**因为某个表而影响整体性能**吧，于是就把这个特别影响数据库整体性能的表拎出来，放到另外的数据库里面，这个就是**分库技术**，把一些影响整体性能的表单独放到其他数据库里面叫做**垂直分库**，因为不在同一个数据库了，也就可以不放在一个盘里面了，大大**化解了IO的压力**。后来衍生出了**垂直分表**的概念（把某些分表放在其他库里面，这时候路由表的表名就得写全了）。

后来系统继续用继续用，发现...又不给力了，这时候是**服务器瓶颈**了（网络，IO，连接数，CPU，内存等硬件瓶颈），这时候所谓的本机垂直分库就意义不大了，就有了**分布式**的概念，分布式分布式，也就是单机变成多机器嘛，这时候sql上遇到各种问题，为了解决这些问题引入了**同义词**和**链接服务器**的概念（19题考察的就是这个），这下以为没啥事情了，发现...什么情况，增删改各种慢？查询倒是还行。

细细研究发现，我去，是**索引的问题**（16，20题考察的内容）。然后借鉴MySQL的相关概念，他们天天说什么**读写分离**，那么我们是不是也可以走一个呢？于是就搞了多个库，2个读1个写。这时候想到一个问题！**数据同步**怎么办？**数据**怎么保证**一致性**？！！！

于是就有了**发布订阅**（这个里面又有两种，一种是主数据库一改变就推送给从数据库，一种是从数据库定期向主数据库发起同步请求【效率低】）这种**读写分离**，主数据库进行增删改，2个从数据库只用来查，只给新手读库的权限，再也不用担心他们修改不加where了~

后来就是业务问题了，我点，我点，我再点~我去，报了一个莫名的错误怎么办？

靠，是谁删了这条数据！怎么知道？

靠，磁盘快满了，怎么没人说？！！

不用担心==》引入数据库**异常预警**的功能（**XEVENT**+数据库发邮件）【这个是站在前人肩上的成果】

现在：集群怎么搞？故障转移怎么走起？逆天正在研究中........

如果经过上面优化而且数据库数据不算大（百G左右吧），那么可以得出个结论==》代码太烂，重构去，二期走起~