**MCP**

**主讲老师：张胜利**

1. **索引**

* **索引分类：**

1. **聚集索引（每张表只能建立一个）**
2. **非聚集索引（每张表可以建立多个）**

**注意：**

**主键是一种约束，唯一索引是一种索引，两者在本质上是不同的。**

**主键创建后一定包含一个唯一性索引，唯一性索引并不一定就是主键。 唯一性索引列允许空值，而主键列不允许为空值。一个表最多只能创建一个主键，但可以创建多个唯一索引。唯一索引的本质是不允许索引键中存在相同的值。**

* **基本语法：**

**create [unique][clustered][nonclustered] index 索引名称on 表名称(列名称 [asc|desc],....)**

**Eg：**

**alter table Customer add constraint customer\_key primary key clustered (SourceID,CustomeriD)**

* **索引的一些其它用法**

1. **Columnstore索引**

**内存中列存储索引通过使用基于列的数据存储和基于列的查询处理来存储和管理数据。  
列存储索引适合于主要执行大容量加载和只读查询的数据仓库工作负荷。 与传统面向行的存储方式相比，使用列存储索引存档可最多提高 10 倍查询性能 ，与使用非压缩数据大小相比，可提供多达 7 倍数据压缩率 。**

**无法聚集，只有非聚集列存储索引才可用；不能修改，只能删除后添加。**

1. **用DROP\_EXISTING改变既有索引**

**可以使用CREATE INDEX...DROP\_EXISTING来改变既有索引的列定义。这个选项还拥有在单条命令中删除和重建索引的优点（取代同时使用DROP INDEX和CREATE INDEX）。而且，在聚集索引上使用DROP\_EXISTING也不会引起既有的非聚集索引自动重建，除非索引列定义已发生改变。**

**Eg:**

* **CREATE NONCLUSTERED INDEX NCI\_TerminationReason\_DepartmentID ON**

**HumanResources.TerminationReason (DepartmentID ASC)**

**WITH (DROP\_EXISTING = ON)**

* **CREATE NONCLUSTERED INDEX NCI\_TerminationReason\_DepartmentID ON**

**HumanResources.TerminationReason (ViolationSeverityLevel, DepartmentID DESC)**

**WITH (DROP\_EXISTING = ON)**

**在第一个示例中，CREATE INDEX没有改变既有索引定义的任何内容，只是通过使用DROP\_EXISTING子句重建了索引。重建索引可以帮助对数据的碎片进行整理.**

**在第二个语句中，将一个新列添加到既有的索引中，并且把它放置在挨着DepartmentID的前面。使用新索引建列重建索引，将它变为组合索引。**

* **REORGANIZE**

**消除碎片可能是索引维护最常规的任务，微软官方给出的建议是当碎片等级为 5% - 30% 之间时采用 REORGANIZE 来“重整”索引，如果达到 30% 以上则使用 REBUILD 来“重建”索引。**

**（二）视图**

**在 SQL 中，视图是基于 SQL 语句的结果集的可视化的表。**

**视图包含行和列，就像一个真实的表。视图中的字段就是来自一个或多个数据库中的真实的表中的字段。我们可以向视图添加 SQL 函数、WHERE 以及 JOIN 语句，我们也可以提交数据，就像这些来自于某个单一的表。**

1. **标准视图**

**标准视图组合了一个或多个表中的数据，您可以获得使用视图的大多数好处，包括将重点放在特定数据上及简化数据操作。**

1. **索引视图**

**索引视图是被具体化了的视图，即它已经过计算并存储。可以为视图创建索引，即对视图创建一个唯一的聚集索引。索引视图可以显著提高某些类型查询的性能。索引视图尤其适于聚合许多行的查询。但它们不太适于经常更新的基本数据集。**

1. **分区视图**

**分区视图在一台或多台服务器间水平连接一组成员表中的分区数据。这样，数据看上去如同来自于一个表。联接同一个 SQL Server 实例中的成员表的视图是一个本地分区视图。**

* **视图创建**

**语法：CREATE VIEW view\_name**

**AS SELECT column\_name(s) FROM table\_name WHERE condition**

**Eg:**

**CREATE VIEW HumanResources.EmployeeHireDate AS**

**SELECT p.FirstName, p.LastName, e.HireDate FROM HumanResources.Employee AS e JOIN Person.Person AS p ON e.BusinessEntityID = p.BusinessEntityID**

* **索引视图**

1. **使用 WITH SCHEMABINDING 选项创建视图: 主要是防止视图所引用的表在视图未被调整的情况下发生改变的选项,也就是说，一旦视图被指定了WITH SCHEMABINDING 选项，那么，在修改用于生成当前视图的表或视图 时，一旦对当前视图产生影响（导致视图失效），则不允许修改**
2. **不允许在SELECT子句中使用(\*)**
3. **视图创建的第一个索引必须是唯一聚集索引。 创建唯一聚集索引后，可以创建更多非聚集索引。**
4. **视图不能调用存储过程。**

**CREATE VIEW 视图名 WITH SCHEMABINDING AS sql语句**

* **通过视图修改数据**

**无论在什么时候修改试图的数据，实际上都是在修改对应的数据表中的数据。如果视图引用多个基表，则不能删除行。 只能更新属于单个基表的列。如果视图引用多个基表，则不能插入行。如果视图引用多个基表，修改的数据列都来自一个基表.如要更新多个基表的数据，则需要为视图创建INSTEAD OF触发器，视图也只能创建INSTEAD OF触发器。**

* **with check option**

**强制视图上执行的所有数据修改语句都必须符合由 select\_statement 设置的准则。**

**Eg:**

**create view stucs as select \* from table where 字段1 ='a' with check option 以后对该视图插入、修改、删除操作时，会自动加上字段1='a' 的条件。  
（三）触发器**

1. **按照触发时机，可以分为INSTEAD OF 触发器 和AFTER触发器。**

**触发器与存储过程的唯一区别是触发器不能执行EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发执行。**

1. **INSTEAD OF触发器**

**又称为替代触发器。当引起触发器执行的修改语句停止执行时，该类触发器代替触发操作执行。该类触发器既可在表上定义，也可在视图上定义。对于每个触发操作（INSERT、UPDATE和DELETE），只能定义一个INSTEAD OF触发器。**

1. **AFTER触发器**

**又称为后触发器。该类触发器是在引起触发器执行的修改语句成功完成之后执行。如果修改语句因错误(如违反约束或语法错误)而执行失败,触发器将不会中以执行。此类触发器只能定义在表上,不能创建在视图上。可以创建多个AFTER触发器。**

1. **语法：**

**create/alter trigger trigger\_name on {table\_name | view\_name}**

**{for | After | Instead of } [ insert, update,delete ] as sql\_statement**

**Eg:**

**alter trigger addOrder on Orders for insert as insert into Order select inserted.Id, inserted.goodName,inserted.Number from inserted。**

**理解触发器里面的两个临时的表：Deleted , Inserted 。注意Deleted 与Inserted分别表示触发事件的表“旧的一条记录”和“新的一条记录”。**

**注意：**

|  |  |  |
| --- | --- | --- |
|  | **虚拟表Inserted** | **虚拟表Deleted** |
| **在表新增时** | **存放新增的记录** | **不存储记录** |
| **修改时** | **存放用来更新的新记录** | **存放更新前的记录** |
| **删除时** | **不存储记录** | **存放被删除的记录** |

* **判断字段的更新函数：if update(判断更新字段)**

**Eg:IF UPDATE(HomeNumber) OR UPDATE(MobileNumber)**