浙江大学管理学院
SCHOOL OF MANAGEMENT
ZHEJIANG UNIVERSITY

分布式文件系统及数据库技术
第九讲

主讲人：曹仔科 彭希羡
浙江大学管理学院
数据科学与工程管理系

AACSB ACCREDITED EFMD EQUIS ACCREDITED ASSOCIATION AMBA ACCREDITED CAMEA 中国高质量MBA教育认证 IQA CEEMAN Accredited

1、登录用户设置

2、索引

3、触发器

需求分析阶段

权限总结表：

权限	StudentRole	TeacherRole	AdminRole
Students表 SELECT	✅	✅	✅
Courses表 SELECT	✅	✅	✅
Takes表 SELECT	✅	✅	✅
Courses表 INSERT/UPDATE/DELETE	❌	✅	✅
Takes表 INSERT/UPDATE/DELETE	❌	✅	✅

登录用户与角色设定

```
--pengxx1创建登录名和用户
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'pengxx1’)
        CREATE LOGIN pengxx1 WITH PASSWORD = 'pengxx1’;
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'pengxx1’)
        CREATE USER pengxx1 FOR LOGIN pengxx1;
GO
```

登录用户与角色设定

浙江大学管理学院 SCHOOL OF MANAGEMENT ZHEJIANG UNIVERSITY

```
-- 1. 创建数据库角色（如果尚未创建）
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'TeacherRole')
        CREATE ROLE TeacherRole;
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'AdminRole')
        CREATE ROLE AdminRole;
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'StudentRole')
        CREATE ROLE StudentRole;
GO
```

登录用户与角色设定

浙江大学 管理学院
SCHOOL OF MANAGEMENT
ZHEJIANG UNIVERSITY

```
-- 2. 设置角色权限 -- 为学生角色设置只读权限
GRANT SELECT ON Students TO StudentRole;
GRANT SELECT ON Courses TO StudentRole;
GRANT SELECT ON Takes TO StudentRole;
GO

-- 3. 为老师角色设置权限
GRANT SELECT ON Students TO TeacherRole;
GRANT SELECT ON Courses TO TeacherRole;
GRANT SELECT ON Takes TO TeacherRole;
GRANT INSERT, UPDATE, DELETE ON Courses TO TeacherRole;
GRANT INSERT, UPDATE, DELETE ON Takes TO TeacherRole;
GO

-- 4. 将用户添加到相应角色
ALTER ROLE TeacherRole ADD MEMBER pengxx1;
```

设计阶段

（3）物理设计：

- **目标：**为特定的 **数据库管理系统**（如MySQL，Oracle，SQL Server）设计底层的存储结构和访问机制，以优化性能。
- 决定文件的存储位置、分区策略。
- 设计**索引**（哪些列需要创建索引，是什么类型的索引）以加速查询。
- 确定磁盘存储结构和访问方法。

索引（Index）

- 数据库中一个表的数据量越来越大后，查询性能会急剧下降。创建索引是保持良好的性能的关键手段。
- 索引能够轻易地将查询性能提高几个数量级。
- 索引是一种特殊的文件，存储着对目标数据表中所有记录的引用指针。

索引

- 数据库中的索引类似于一本书的目录，可以帮助你在一本厚厚的书中通过目录快速找到你想要的信息，而不需要浏览完全书。

- 例如需要遍历2万条数据，在没有索引的情况下，数据库会遍历全部2万条数据后选择符合条件的，而有了相应的索引后，数据库会在索引中更快速地查找符合条件的选项。

- **数据库索引就是为了提高表的搜索效率而对某些字段的值建立的目录。**

索引的作用

- 建立索引的主要目的是为了极大提高数据库的查询性能。其核心作用包括：
  - **大幅加速查询速度**：通过索引快速定位数据，极大减少了需要扫描的数据量，尤其是在处理大数据表或多表关联时，性能提升尤为显著。
  - **降低系统开销**：减少了磁盘I/O操作和排序过程中的CPU消耗。
  - **确保数据完整性**：唯一性索引能保证表中各行数据的唯一性。
  - **优化连接、分组和排序**：显著加速表与表之间的连接操作，以及使用 GROUP BY和 ORDER BY子句时的处理速度。

建立索引的成本

- 索引需要占用额外的磁盘空间
- 在插入和修改数据时要花费更多的时间（因为索引也要随之进行维护更新）

索引的类型（按约束性划分）

- **普通索引**：根据特定规则为表中某列或某些列的值创建索引，对列的值无唯一性要求，仅用于加速查询
  - **单列索引** vs **组合索引**（多列的值组合在一起生成索引）

- **唯一性索引**：要求被索引的列（或列组合）的值必须是唯一的。
  - **主键索引**：唯一索引的特例。它要求列值唯一且非空（NOT NULL）。每个表最多只能有一个主键索引，用于唯一标识每一行数据。定义主键时自动创建。

索引的类型（按物理存储结构划分）

- **聚集索引**：
  - **定义**：索引项的顺序决定了表中数据行的物理存储顺序。一张表只能有一个聚集索引，因为数据本身只能按一种方式排序。
  - **类比**：就像字典本身按拼音顺序排列，内容（数据行）的顺序就是索引的顺序。
- **非聚集索引**：
  - **定义**：索引的顺序与数据行的物理存储顺序完全无关。索引结构独立于数据，索引数据结构中包含的是指向数据行位置的指针（行定位器）。
  - **类比**：就像字典后面的偏旁部首检字表，检字表（索引）的顺序与正文（数据）的顺序无关，通过检字表找到字所在的页码。

索引的设计原则

- 索引的本质是**以空间换时间**。增加索引会提升查询速度，但也会降低数据更新速度并占用额外空间。
- 避免过度索引：并非索引越多越好。每个多余的索引都是性能负担。

- 应该创建索引的情况：
  - **主键与外键**：主键自动创建唯一索引。外键字段必须建立索引，以优化表连接（JOIN）操作和引用完整性检查。
  - **高频查询条件**：频繁出现在 WHERE子句中的字段。
  - **连接与排序字段**：用于表连接（JOIN）、排序（ORDER BY）和分组（GROUP BY）的字段。
  - **数据量较大的表**：当表记录较多时（例如超过数百行），索引能避免全表扫描，显著提升查询性能。

索引的设计原则

- 不应或谨慎创建索引的情况：
  - **低唯一性字段**：重复值过多的字段（如“性别”、“状态”等枚举值），索引效率极低，优化器通常会忽略。
  - **频繁更新的字段**：索引会增加INSERT、UPDATE、DELETE操作的开销，因为需要同时维护索引结构。
  - **很少被查询的字段**：为不参与查询的字段创建索引只会增加额外存储和维护成本。

聚集索引vs非聚集索引应用场景

- 由于聚集索引决定了数据行的物理存储顺序，表的主键一般就是聚集索引，而且一个表只允许一个聚集索引。
  - **用于范围查询**：经常使用 BETWEEN、>、<等操作符的列。因为数据按顺序物理存储，连续的数据在磁盘上也大概率是连续的，可以快速读取。
  - **用于排序**：查询结果经常需要按某列排序。因为数据已经排好序了，数据库无需额外的排序操作。

- 非聚集索引是独立的结构，存储的是索引键值和指向数据行的指针。
  - 用于经常出现在 WHERE、JOIN ... ON、ORDER BY、GROUP BY子句中的列。
  - 创建组合索引以实现覆盖查询（查询的所有列都包含在非聚集索引的键中）。

创建索引

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX
index_name
ON table_name (column1 [ASC|DESC], column2 [ASC|DESC], ...)
[INCLUDE (include_column1, include_column2, ...)]
[WHERE condition]
[WITH (index_option = value, ...)];
```

- INCLUDE：创建覆盖索引，包含某些列，提高查询速度
- WHERE：限定只对某些行创建索引
- WITH：设定某些特定的选项，比如填充因子

创建索引

```
CREATE TABLE authors(
    auth_id int IDENTITY(1,1) NOT NULL,
    auth_name   varchar(20) NOT NULL,
    auth_gender tinyint NOT NULL,
    auth_phone   varchar(15) NULL,
    auth_note   varchar(100) NULL
)

CREATE UNIQUE CLUSTERED INDEX Idx_phone
ON authors(auth_phone DESC)
WITH
FILLFACTOR=30;
```

在authors表中auth_phone列上创建一个名称为Idx_phone的唯一聚集索引，降序排列，填充因子为30%

创建索引

```
CREATE UNIQUE NONCLUSTERED INDEX Idx_nameAndgender
ON authors(auth_name, auth_gender)
WITH
FILLFACTOR=10;
```

- 在authors表中auth_name和auth_gender列上创建一个名称为Idx_nameAndgender的唯一非聚集组合索引，升序排列，填充因子为10%

填充因子

- 创建索引时，可以指定一个填充因子，以便在索引的每个叶级页上留出额外的间隙和保留一定百分比的空间，供将来表的数据存储容量进行扩充和减少页拆分的可能性。

- 100%：表示页将填满，所留出的存储空间量最小。只有当不会对数据进行更改时（例如，在只读表中）才会使用此设置。

- 值越小则数据页上的空闲空间越大，这样可以减少在索引增长过程中对数据页进行拆分的需要，但需要更多的存储空间。当表中数据会发生更改时，这种设置更为适当。

触发器

- 触发器（trigger）是一种特殊类型的存储过程（procedure）
- 不同之处：触发器通过事件进行触发而被执行；而存储过程通过过程名进行调用
- 当往某一个表中插入、修改或者删除记录时，SQL可以自动执行触发器所定义的SQL语句， 从而确保对数据的处理必须符合规则

触发器

- 触发器和引起触发器执行的SQL语句被当作一次**事务**处理
- 事务如果未获成功，SQL会自动返回该事务执行前的状态（**回滚**）
- 实现复杂的数据完整性约束的高效手段

触发器的优势

- 触发器可以实现由主键和外键所不能保证的复杂的参照完整性和数据一致性

- 相比于其他约束手段（如CHECK， ADD CONSTRAINT）：

  - 触发器是自动的，由事件触发的

  - 触发器可以通过数据库中的相关表进行层叠更改

  - 触发器可以实现更复杂的约束条件（如触发器可以引用其他表中的列）

触发器

- **作用总结：**
  - 强制数据库间的引用完整性
  - 级联修改数据库中所有相关的表，自动触发其他与之相关的操作
  - 跟踪变化，撤销或回顾违法操作，防止非法修改数据
  - 返回自定义的错误信息
  - 触发器可以调用更多的存储过程

触发器分类

- 数据操作语言（DML）触发器
  - 当数据库服务器中发生数据操作(INSERT, UPDATE, DELETE)事件时被激活调用
  - 这类触发器又可以分为: **AFTER触发器**和**INSTEAD OF触发器**

- 数据定义语言（DDL）触发器
  - 当数据库服务器中发生数据定义(CREATE, ALTER, DROP)事件时被激活调用

DML触发器

- INSERT触发器：由INSERT语句触发
- UPDATE触发器：由UPDATE语句触发
- DELETE触发器：由DELETE语句触发

- 针对每个DML触发器定义两个特殊的表：DELETED表和INSERTED表
  - DELETED表存放执行DELETE或者UPDATE语句而要删除的所有行
  - INSERTED表存放执行INSERT或者UPDATE语句而要插入的所有行

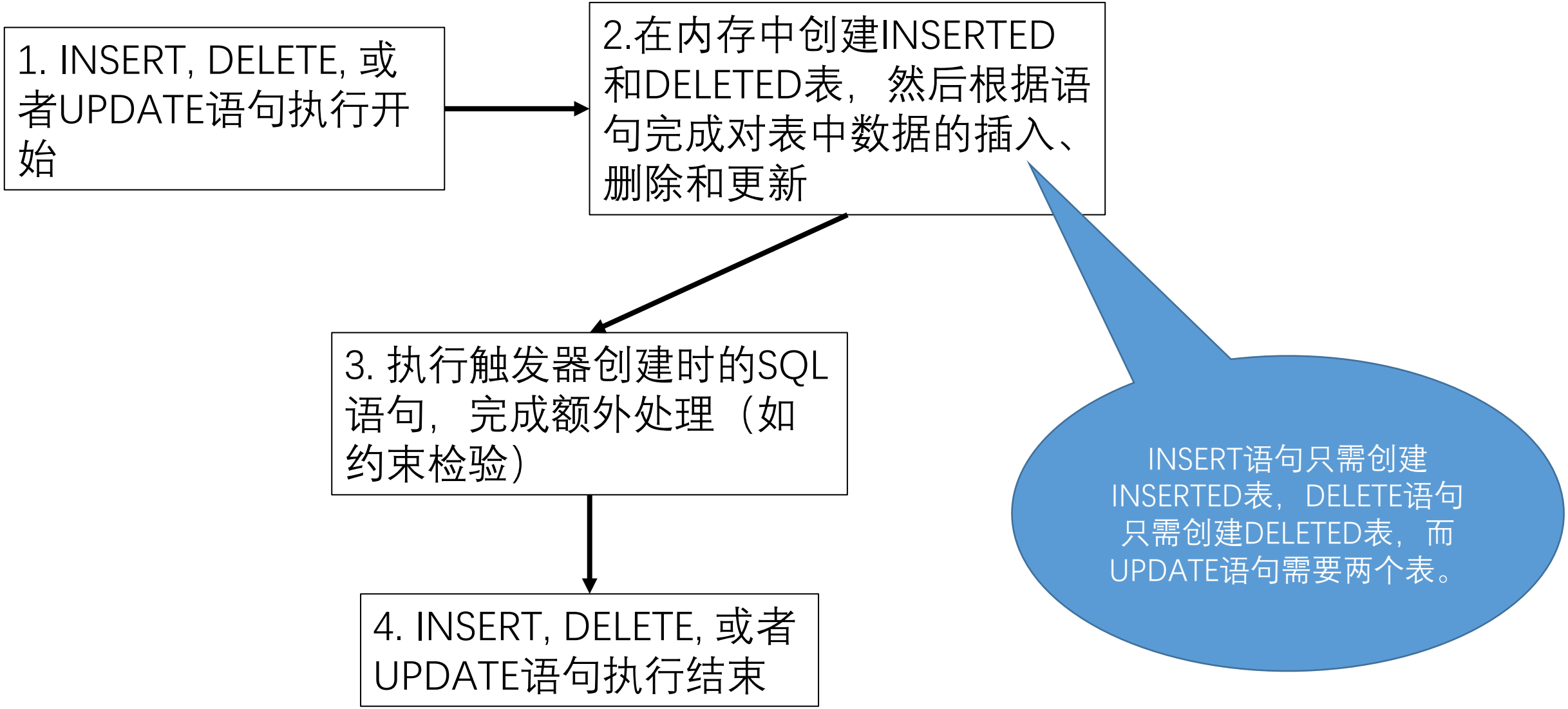
- 这两个逻辑表在内存中存放，用户不可修改
  - 触发器执行完后，与该触发器相关的两个表也会在内存中删除

触发器创建语法

```
CREATE TRIGGER trigger_name
ON {table | view}
AFTER {DELETE, UPDATE, INSERT}
AS
SQL_statement
```

- 触发器名字必须是唯一的
- 触发器可以在表或者视图上执行

DML触发器执行过程



触发器示例

假设已创建下表 stu_info:

	列名	数据类型	允许 Null 值
🔑	s_id	int	☐
	s_name	nvarchar(50)	☐
	s_score	int	☐
	s_sex	nchar(10)	☐
			☐

INSERT触发器：示例

```
USE my_first_db
GO

CREATE TRIGGER insert_student
ON stu_info
AFTER INSERT
AS
BEGIN
    IF OBJECT_ID(N'stu_Sum', N'U') IS NULL          --执行第一次INSERT操作时，创建存储学生总人数的stu_Sum表
        CREATE TABLE stu_Sum(number INT)

    IF NOT EXISTS (SELECT * FROM stu_Sum)           --给表stu_Sum赋一个初始值0
        INSERT INTO stu_Sum VALUES(0);

    DECLARE @stuNumber INT;                         --每次INSERT操作之后将总人数更新
    SELECT @stuNumber = COUNT(*) FROM stu_info;
    UPDATE stu_Sum SET number=@stuNumber;

END
GO
```

INSERT触发器：示例

```
/*测试INSERT触发器*/
SELECT COUNT(*) stu_info表中总人数 FROM stu_info;

INSERT INTO stu_info VALUES(20, '白雪', 87, '女');

SELECT COUNT(*) stu_info表中总人数 FROM stu_info;

SELECT number FROM stu_Sum;

INSERT INTO stu_info VALUES(36, '王五', 85, '男');

SELECT COUNT(*) stu_info表中总人数 FROM stu_info;

SELECT number FROM stu_Sum;
```

	stu_info表中总人数
1	3

	number
1	3

INSERT触发器：示例（禁止插入数据）

```
/*禁止向表stu_Sum直接插入数据*/
CREATE TRIGGER Insert_forbidden
ON stu_Sum
AFTER INSERT
AS
BEGIN
    ROLLBACK TRANSACTION
    PRINT('不允许直接向该表插入记录，操作被禁止')
END

/*测试*/
INSERT INTO stu_Sum VALUES(5);
```

```
不允许直接向该表插入记录，操作被禁止
消息 3609，级别 16，状态 1，第 52 行
事务在触发器中结束。批处理已中止。
```

DELETE触发器：示例

```
/*定义DELETE触发器*/
CREATE TRIGGER delete_student
ON stu_info
AFTER DELETE
AS
BEGIN
    SELECT s_id AS 已删除学生编号, s_name, s_score, s_sex
    FROM DELETED
END
GO
```

DELETE触发器：示例

```
/*测试DELETE触发器*/
DELETE FROM stu_info WHERE s_id=36;
```

	已删除学生编号	s_name	s_score	s_sex
1	36	王五	85	男

这里返回的结果记录是从DELETED表中查询得到的

UPDATE触发器

- UPDATE触发器主要用来约束对现有数据的修改

- UPDATE触发器可以执行两种操作：
  - 更新前的记录存储到DELETED表
  - 更新后的记录存储到INSERTED表

UPDATE触发器：示例

```
/*定义UPDATE触发器*/
CREATE TRIGGER update_student
ON stu_info
AFTER UPDATE
AS
BEGIN
DECLARE @stuCount INT;
SELECT @stuCount = COUNT(*) FROM stu_info;
UPDATE stu_Sum SET number=@stuCount;

SELECT s_id AS 更新前学生编号, s_name AS 更新前学生姓名 FROM DELETED
SELECT s_id AS 更新后学生编号, s_name AS 更新后学生姓名 FROM INSERTED
END
GO
```

UPDATE触发器：示例

```
/*测试UPDATE触发器*/
UPDATE stu_info SET s_name='白雪公主' WHERE s_id=20;
GO
```

	更新前学生编号	更新前学生姓名
1	20	白雪

	更新后学生编号	更新后学生姓名
1	20	白雪公主

这里返回的结果记录是分别从INSERTED和DELETED表中查询得到的

UPDATE触发器：示例2

```
CREATE TRIGGER update_student_2
ON stu_info
AFTER UPDATE
AS
BEGIN
DECLARE @stuNewScore INT;
SELECT @stuNewScore = s_score FROM INSERTED;
IF @stuNewScore>100
    BEGIN
        PRINT('分数超过100，不合理！')
        ROLLBACK TRANSACTION
    END
END
GO
```

UPDATE触发器：示例2

```
/*测试UPDATE触发器*/
UPDATE stu_info SET s_score=120 WHERE s_id=20;
GO

UPDATE stu_info SET s_score=100 WHERE s_id=20;
GO
```

```
(1 行受影响)

(1 行受影响)

(1 行受影响)
分数超过100，不合理!
消息 3609，级别 16，状态 1，第 105 行
事务在触发器中结束。批处理已中止。
```

	s_id	s_name	s_score	s_sex
1	20	白雪公主	100	女

DML替代触发器：创建语法

```
CREATE TRIGGER trigger_name
ON {table | view}
INSTEAD OF {DELETE, UPDATE, INSERT}
AS
SQL_statement
```

- 与AFTER触发器语法上的区别：关键词AFTER变成INSTEAD OF

替代触发器

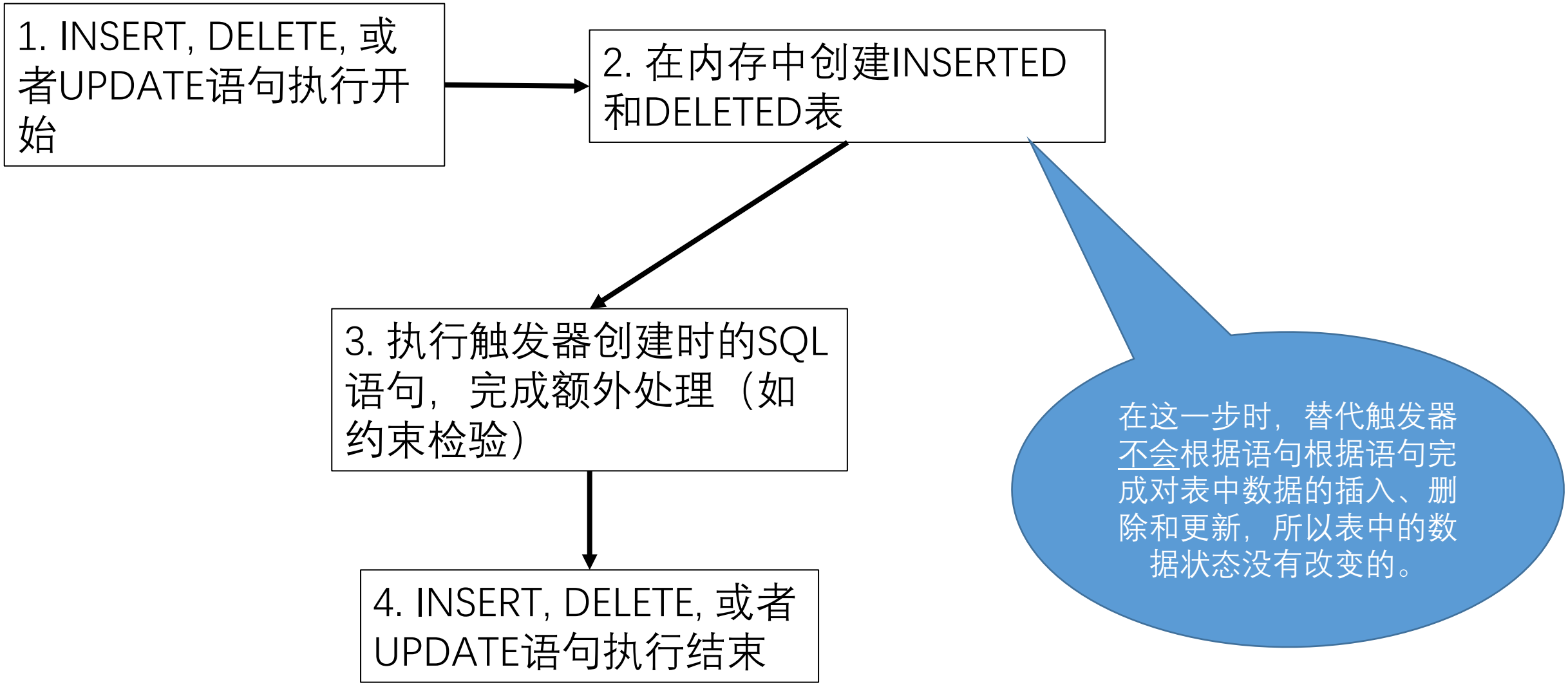
与之前触发器的区别：

- 当一个AFTER触发器被激发后，先建立临时的INSERTED和DELETED表，然后执行对表中数据的操作，最后再执行触发器中的SQL代码

- 当一个替代(INSTEAD OF)触发器被激发后，先建立临时的INSERTED和DELETED表，然后直接执行触发器中的SQL代码，而不会执行对表中数据的操作

- 即区别在于激发触发器的数据操作语句（INSERT/UPDATE/DELETE）是否会被执行

DML替代触发器执行过程



INSERT替代触发器：示例

```
/*定义INSERT替代触发器*/
ALTER TRIGGER InsteadOfInsertStudent
ON stu_info
INSTEAD OF INSERT
AS
BEGIN
DECLARE @stuScore INT;
SELECT @stuScore = s_score FROM INSERTED;
If @stuScore > 100
    SELECT '输入成绩有误' AS 失败原因
ELSE
    INSERT INTO stu_info SELECT * FROM INSERTED;
END
GO
```

INSERT替代触发器：示例

```
/*测试INSERT替代触发器*/
INSERT INTO stu_info VALUES(22, '周鸿', 120, '男');
SELECT * FROM stu_info;
GO
```

	失败原因
1	输入成绩有误

	s_id	s_name	s_score	s_sex
1	20	白雪公主	100	女

嵌套触发器

- 如果一个触发器在执行操作时调用了另外一个触发器，而这个触发器可能又接着调用了下一个触发器，这就是嵌套触发器
- 触发器嵌套默认开启，但可以禁用
- SQL Server 2014中触发器最多嵌套32层；如果嵌套的次数超过限制，那么该触发器将被终止，并回滚整个事务

嵌套触发器

- SQL Server 2014中禁用和启用嵌套触发器：
  - 禁用：EXEC sp_configure ‘nested triggers’, 0
  - 启用：EXEC sp_configure ‘nested triggers’, 1

递归触发器

- 触发器的递归是指一个触发器从其内部再一次激活该触发器
- SQL Server 2014中递归触发器默认是禁用的
- 递归触发器最多只能递归16层；超过限制后会回滚整个事务

DDL触发器

- 顾名思义，该类触发器当用户对数据库对象创建、修改和删除时触发

- 语法：

```
CREATE TRIGGER trigger_name
ON {ALL SERVER | DATABASE}
FOR {event_type}
AS
SQL_statement
```

DDL触发器

- 语法：

```
CREATE TRIGGER trigger_name
ON {ALL SERVER | DATABASE}
FOR {event_type}
AS
SQL_statement
```

  - ALL SERVER表示该DDL触发器作用于整个服务器
  - DATABASE表示该DDL触发器只作用于当前数据库
  - event_type: 如CREATE_DATABASE, ALTER_DATABASE, CREATE_TABLE, ALTER_TABLE, DROP_TABLE; CREATE_VIEW, ALTER_VIEW, DROP_VIEW

DDL触发器：示例

```
/*定义DDL触发器*/
USE my_first_db;
GO

CREATE TRIGGER DenyDelete
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
BEGIN
PRINT '禁止进行删除和修改表操作！'
ROLLBACK TRANSACTION
END
GO
```

```
/*测试DDL触发器*/
DROP TABLE stu_info;
GO
```

```
禁止进行删除和修改表操作！
消息 3609，级别 16，状态 2，第 151 行
事务在触发器中结束。批处理已中止。
```

DDL触发器：示例

```
/*定义DDL触发器*/
USE my_first_db;
GO

CREATE TRIGGER DenyCreate_AllServer
ON ALL SERVER
FOR CREATE_DATABASE, ALTER_DATABASE
AS
BEGIN
PRINT '用户无权限创建或修改服务器上的数据库！'
ROLLBACK TRANSACTION
END
GO
```

```
/*测试DDL触发器*/
CREATE DATABASE test;
GO
```

用户无权限创建或修改服务器上的数据库！
消息 3609，级别 16，状态 2，第 170 行
事务在触发器中结束。批处理已中止。

查看触发器

```
/*查看触发器*/
sp_helptext InsteadOfInsertStudent
GO
```

	Text
1	CREATE TRIGGER InsteadOfInsertStudent
2	ON stu_info
3	INSTEAD OF INSERT
4	AS
5	BEGIN
6	DECLARE @stuScore INT;
7	SELECT @stuScore = s_score FROM INSERTED;
8	If @stuScore > 100
9	SELECT '输入成绩有误' AS 失败原因
10	END

修改触发器

```
ALTER TRIGGER trigger_name
ON {table | view}
{AFTER | INSTEAD OF} {DELETE, UPDATE, INSERT}
AS
SQL_statement
```

- 除了关键字由**CREATE**换成**ALTER**之外，修改触发器的语句和创建触发器的语句格式完全一样
- 旧的定义会被新的定义覆盖

删除触发器

- DROP TRIGGER trigger_name [, … n]
  ON {对象名}

如

- DROP TRIGGER insert_student ON my_first_db
- DROP TRIGGER DenyCreate_AllServer ON ALL SERVER

启用和禁用触发器

- DISABLE TRIGGER trigger_name
  ON {对象名}

- ENABLE TRIGGER trigger_name
  ON {对象名}

浙江大学管理学院
SCHOOL OF MANAGEMENT
ZHEJIANG UNIVERSITY

谢谢！

下次见！

AACSB ACCREDITED
EFMD EQUIS ACCREDITED
ASSOCIATION AMBA ACCREDITED
CAMEA 中国高质量MBA教育认证
IQA CEEMAN Accredited