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
第六讲

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

视图

- 目前为止，我们都假定给定的关系都是实际存储在数据库中的（即**基本关系**）
- SQL允许通过查询来定义“**虚关系**”，它在概念上包含查询的结果
  - 虚关系不预先计算和存储，在使用时才通过执行查询计算出来。
- 视图就像创建了一个智能文件夹：
  - 只包含你关心的文件（特定列）
  - 自动整理相关文件（连接多个表）
  - 标记重要信息（计算列）
  - 设置访问权限（安全控制）
- 下次只需要打开这个"智能文件夹"（视图），就能立即看到整理好的信息，无需每次重新翻找。

创建视图

定义视图：

```
CREATE VIEW [schema_name.]view_name
[WITH <view_attribute> [,...n]]
AS
    select_statement
[WITH CHECK OPTION]
```

创建一个视图显示学生姓名和所选课程

--创建一个视图显示学生姓名和所选课程，然后就可以对该视图进行查询

```
CREATE VIEW vw_student_course_names
AS
SELECT s.name AS student_name, c.course_name, t.grade
FROM Students s
INNER JOIN Takes t ON s.student_id = t.student_id
INNER JOIN Courses c ON t.course_id = c.course_id;

SELECT * FROM vw_student_course_names
WHERE grade < 80;
```

视图使用

- 一个视图被定义后，系统只存储定义本身，并不存储执行结果；一旦视图关系出现在查询中，它会被已存储的表达式替代，然后重新查询结果。

常见实际应用场景：

- 报表系统：创建预聚合视图加速报表生成
- API数据层：为应用程序提供定制数据视图
- 权限管理：不同角色访问不同视图
- 数据迁移：创建兼容旧系统的视图
- 简化开发：封装复杂业务逻辑

视图管理

- 修改视图：**ALTER VIEW**
  - 用新的视图定义覆盖旧的

```
ALTER VIEW view_name
AS
    select_statement
```

删除视图

DROP VIEW viewName **[RESTRICT | CASCADE]**

- **CASCADE**：所有依赖于此视图的对象都会被删除
- **RESTRICT**：如果存在任何对象依赖于此视图，则拒绝执行

1、授权

2、存储过程

3、函数

4、游标

授权

- 我们可能会给某个用户在数据库的某些部分授予某种形式的权限(privilege)

- 对**数据**的授权包括：
  - 授权读取数据(SELECT)
  - 授权插入新数据(INSERT)
  - 授权更新数据(UPDATE)
  - 授权删除数据(DELETE)

授权

- 除了数据上的授权，还可以授权用户数据库模式上的权限，如允许用户创建、修改或删除关系
  - ALTER、EXECUTE、CONTROL等
- 最大的授权形式是授予所有权限（ALL）
- 当用户提交查询或更新等操作请求时，SQL先检查用户权限；未经授权，则拒绝执行

SQL授权：GRANT语句

GRANT {权限列表 | ALL PRIVILEGES}
ON 关系名或视图名
TO {用户/角色列表 | PUBLIC}
[WITH GRANT OPTION]

- 权限列表包含一个或多个权限（多个权限用逗号隔开）； ALL PRIVILEGES代表授予所有权限
- PUBLIC授权所有当前和未来的合法用户
- WITH GRANT OPTION允许被授权的用户将此权限**传递**给其他用户

SQL授权：GRANT语句

- 假设我们有一个数据库 SalesDB，其中包含：
  - 架构：Sales
  - 表： Sales.Customers, Sales.Orders
  - 数据库用户：UserA, UserB

- 允许 UserA读取 Customers表的所有数据：
  - **GRANT SELECT ON Sales.Customers TO UserA;**

SQL授权：GRANT语句

- 允许UserB向Orders表插入新记录并更新现有记录：
  - **GRANT INSERT, UPDATE ON Sales.Orders TO UserB;**

- 允许 UserA查看Customers表的CustomerName和Email列，但不能查看其他列（如银行卡号、地址等）：
  - **GRANT SELECT ON Sales.Customers(CustomerName, Email) TO UserA;**

SQL授权：GRANT语句

- 还可以进行架构级权限的授予

GRANT permission [,...n] ON SCHEMA :: schema_name
TO database_principal [,...n]
[WITH GRANT OPTION]
[AS granting_principal]

- 例如，允许UserA对该数据库下面的架构Sales中的所有的表进行SELECT操作：
- **GRANT SELECT ON SCHEMA::Sales TO UserA;**

角色ROLE

- 如果一个数据库有有成千上万的用户，如果需要给每个用户分配他们工作所需的精确权限，工作量巨大且繁琐，容易出错，难以维护。

- 数据库角色就是为了解决这些问题而引入的核心权限管理机制。它的核心思想是：
  - 权限的容器： 角色本身是一个命名的实体，它本身不拥有权限。它的作用是将一组权限 (Permissions) 打包在一起。
  - 权限的分配单元： 你不再直接给单个用户授权，而是将权限授予角色。
  - 用户的成员身份： 然后将用户（Users）或其他角色添加为这个角色的成员 (Members)。
  - 权限继承： 成为角色的成员后，用户就自动继承了该角色所拥有的所有权限。

创建角色ROLE

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

CREATE ROLE [role_name]

[AUTHORIZATION owner_name]

- [role_name]: 必需。 要创建的新角色的名称。
- [AUTHORIZATION owner_name]: 可选。指定新角色的所有者。所有者可以是数据库用户或另一个数据库角色。如果省略，执行CREATE ROLE语句的用户将成为该角色的所有者。所有者可以管理角色的成员身份和权限。

创建角色ROLE

- 创建一个名为SalesStaff的角色：

```
CREATE ROLE SalesStaff;
```

- 使用 GRANT语句将权限授予角色（就像之前授予用户一样）：

```
GRANT SELECT ON Sales.Customers TO SalesStaff;
GRANT SELECT, INSERT ON Sales.Orders TO SalesStaff;
```

创建角色ROLE

- 将用户添加到角色（使角色生效)： 使用 ALTER ROLE ... ADD MEMBER ...语句 （一次只能添加一个用户）。

  ALTER ROLE SalesStaff ADD MEMBER UserA;

  ALTER ROLE SalesStaff ADD MEMBER UserB;

- 将用户从角色中移除:

  ALTER ROLE SalesStaff DROP MEMBER UserA;

- 删除角色： (必须先移除所有成员)

  DROP ROLE SalesStaff;

SQL回收授权：REVOKE语句

REVOKE [GRANT OPTION FOR] {权限列表 | ALL PRIVILEGES}
ON 关系名或视图名
FROM {用户/角色列表 | PUBLIC}
[RESTRICT | CASCADE]

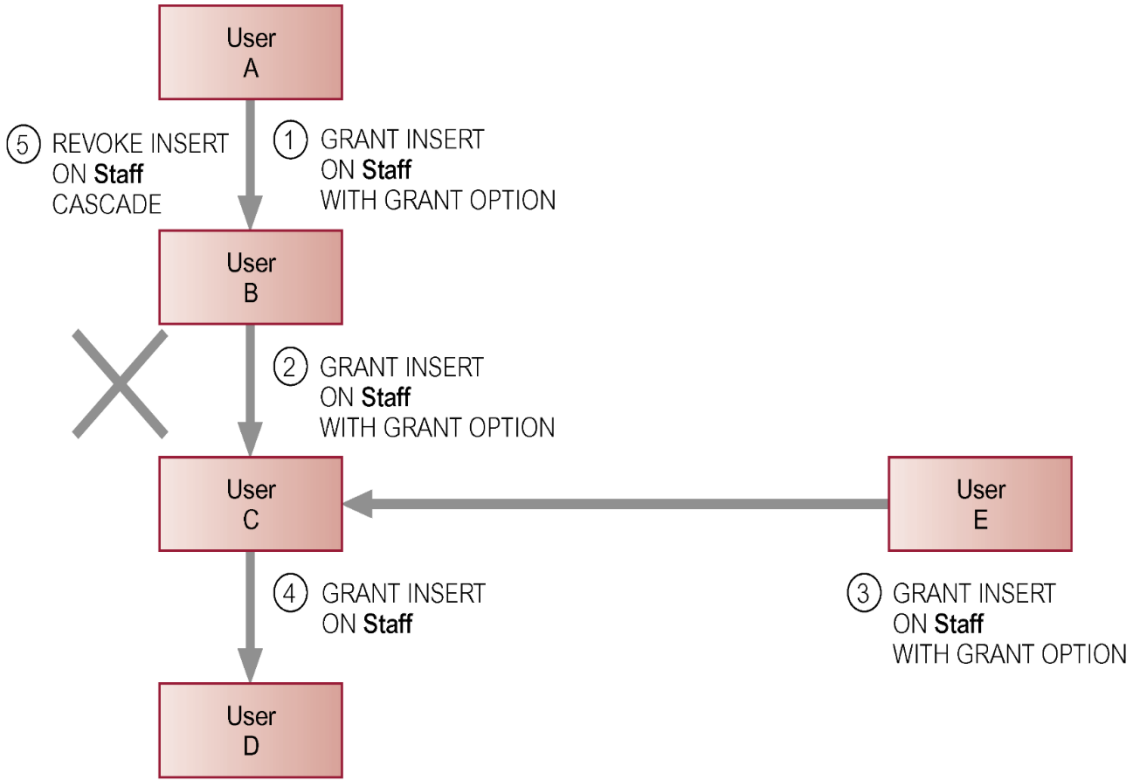
- **GRANT OPTION FOR**：可选项，如果指明则只撤销GRANT语句中通过WITH GRANT OPTION传递的那些权限，即撤销一个用户传递权限的权限（已经传递出的权限因此撤销），但其自己的权限不被撤销

- 如果权限的授予时有WITH GRANT OPTION，回收该权限时必须使用CASCADE选项，否则拒绝执行。

SQL回收授权：REVOKE

- **当一个用户被多个用户授权时，回收授权是不会跨用户影响的**

- 右图中的C、D用户仍然有插入新数据权限，B用户则没有了



浙江大学 管理学院
SCHOOL OF MANAGEMENT
ZHEJIANG UNIVERSITY

SQL：过程、函数

SQL

- SQL是一种**面向转换**的语言
  - 它只关心将输入关系转换成所需要的输出关系
  - 非过程化：用户只需告诉需要，不需关系**如何**得到所需结果的过程
  - 由标准的英语单词组成（如SELECT, INSERT, CREATE等）

- SQL分为两大部分
  - 数据定义语言（DDL）：用于定义数据库结构和数据的访问控制
  - 数据操作语言（DML）：用于查询和更改数据

SQL的过程化扩展

- SQL本身是一个**非过程化**语言

- 对SQL进行**过程化扩展**是处理复杂数据库系统需要的， 如：
  - Oracle的PL/SQL
  - Microsoft SQL Server的TransactSQL
  - PostgreSQL的PL/pgSQL

- 每个SQL“方言”语法会有差别，但基本原理一致
  - 我们着重讲解Microsoft SQL Server的TransactSQL中SQL过程化扩展

存储过程Procedure

- 在很多情况下，一些代码会被开发者重复编写多次
- 如果每次都编写相同功能的代码，不但繁琐、容易出错，而且执行效率也低
- 存储过程(procedure)就是为了实现特定任务，将一些需要多次调用的固定操作语句编写成程序段，这些程序段存储在服务器上，以便其他应用来调用

SQL Server：创建和执行基本存储过程

```
-- 创建存储过程：获取所有学生及其选课信息
CREATE PROCEDURE GetAllStudentCourseInfo
AS
BEGIN
SELECT s.student_id AS 学号, s.name AS 姓名, s.age AS 年龄, s.gender AS 性别,
s.enrollment_date AS 入学日期, c.course_name AS 课程名称, t.grade AS 成绩
FROM Students s LEFT JOIN Takes t ON s.student_id = t.student_id
LEFT JOIN Courses c ON t.course_id = c.course_id
ORDER BY s.student_id, c.course_name;
END
GO
-- 执行存储过程 EXEC GetAllStudentCourseInfo;
```

SQL Server：创建带有输入参数的存储过程

```
-- 创建存储过程：根据课程名称获取学生成绩
CREATE PROCEDURE GetStudentsByCourse @CourseName NVARCHAR(100) -- 输入参数
AS
BEGIN
SELECT s.student_id AS 学号, s.name AS 姓名, t.grade AS 成绩
FROM Takes t JOIN Students s ON t.student_id = s.student_id JOIN Courses c ON t.course_id =
c.course_id
WHERE c.course_name = @CourseName
ORDER BY t.grade DESC;
END
GO
-- 执行存储过程（按参数名传递） EXEC GetStudentsByCourse @CourseName = '高等数学'; -- 执行存
储过程（按位置传递） EXEC GetStudentsByCourse '高等数学';
```

SQL Server：创建带有输出参数的存储过程

```
-- 创建存储过程：统计学生选课情况
CREATE PROCEDURE GetStudentCourseStats
@StudentName NVARCHAR(50), -- 输入参数
@CourseCount INT OUTPUT, -- 输出参数1
@AvgGrade FLOAT OUTPUT, -- 输出参数2
@HighestGrade INT OUTPUT -- 输出参数3
AS BEGIN
-- 获取学生ID
DECLARE @StudentID INT;
SELECT @StudentID = student_id FROM Students WHERE name = @StudentName
-- 计算统计数据
SELECT @CourseCount = COUNT(*), @AvgGrade = AVG(CAST(grade AS FLOAT)), @HighestGrade = MAX(grade)
FROM Takes WHERE student_id = @StudentID;
END
GO
```

SQL Server：创建带有输出参数的存储过程

```
-- 执行存储过程 OUT作为关键词一定要写
DECLARE @Count INT, @Avg FLOAT, @Max INT;
EXEC GetStudentCourseStats
@StudentName = '李四',
@CourseCount = @Count OUTPUT,
@AvgGrade = @Avg OUTPUT,
@HighestGrade = @Max OUTPUT;

-- 显示结果
PRINT '学生: 李四';
PRINT '选课数量: ' + CAST(@Count AS VARCHAR);
PRINT '平均成绩: ' + CAST(@Avg AS VARCHAR(10));
PRINT '最高成绩: ' + CAST(@Max AS VARCHAR);
```

注意：
@CourseCount 是存储过程的参数名

@Count 是调用方的变量名

OUTPUT 表示数据流向是从存储过程→调用方变量

SQL Server：用户自定义函数

- 用户自定义函数可以像系统函数一样在查询或过程中调用，也可以像过程一样使用EXECUTE命令来执行
- **标量**函数：返回一个确定类型的标量值
- **表值**函数：返回数据类型为table的函数；相当于一个参数化的视图

SQL Server：创建标量函数

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

```
-- 创建函数：计算学生平均成绩
CREATE FUNCTION dbo.GetStudentAverageGrade (@StudentID INT)
RETURNS DECIMAL(5,2)
AS
BEGIN
        DECLARE @AvgGrade DECIMAL(5,2);
        SELECT @AvgGrade = AVG(CAST(grade AS DECIMAL(5,2))) FROM Takes
        WHERE student_id = @StudentID;
        RETURN ISNULL(@AvgGrade, 0); -- 如果学生没有成绩，返回0
END
GO
```

SQL Server：使用标量函数


```
-- 在SELECT语句中使用
 SELECT
name AS 姓名,
dbo.GetStudentAverageGrade(student_id) AS 平均成绩
FROM Students;

-- 在WHERE子句中使用
SELECT name
FROM Students
WHERE dbo.GetStudentAverageGrade(student_id) > 85;
```

函数必须使用两部分名称（架构.函数名）调用

SQL Server：创建表值函数


```
--创建函数：获取学生选课详情
CREATE FUNCTION dbo.GetStudentCourses (@StudentID INT)
RETURNS TABLE
AS
RETURN
( SELECT c.course_name AS 课程名称, t.grade AS 成绩, c.teacher AS 授课教师,
c.credit AS 学分
FROM Takes t
JOIN Courses c ON t.course_id = c.course_id
WHERE t.student_id = @StudentID )
GO
```

SQL Server：创建表值函数


```
-- 像表一样使用函数
SELECT * FROM dbo.GetStudentCourses(1); -- 获取学生ID为1的选课信息
-- 与其他表连接
SELECT s.name AS 姓名, sc.课程名称, sc.成绩
FROM Students s
CROSS APPLY dbo.GetStudentCourses(s.student_id) sc;
```

CROSS APPLY是 SQL Server 中一个强大的运算符，用于将表值函数或子查询应用于另一个表的每一行。它类似于 INNER JOIN，但更灵活，特别适合处理需要为每一行调用函数的场景。

存储过程 vs. 函数

- 当你需要执行一个动作，完成一个任务，或者不需要返回值，或者需要返回多个值/结果集时，使用存储过程。
  - 例如：数据迁移、复杂的业务逻辑处理、生成报告（返回多个结果集）。
- 当你需要封装一个计算逻辑，并且希望它的结果能作为一个表达式的一部分在SQL语句中使用时，使用函数。
  - 例如：计算折扣、格式化字符串、作为一个可重用的数据源（表值函数）。

- 存储过程是“做事情”的，函数是“算东西”的。

使用控制语句

- 条件 IF 语句
- 条件 CASE 语句
- 循环WHILE语句

条件 IF 语句

IF Boolean_expression
{ sql_statement | statement_block }
[**ELSE** { sql_statement | statement_block }]

IF语句示例

```
DECLARE @maxWeight FLOAT, @productKey INTEGER
SET @maxWeight = 100.00
SET @productKey = 424
IF @maxWeight <= (SELECT Weight from DimProduct WHERE ProductKey = @productKey)
    SELECT @productKey AS ProductKey, EnglishDescription, Weight, 'This product is too heavy to ship
        AS ShippingStatus
    FROM DimProduct WHERE ProductKey = @productKey
ELSE
    SELECT @productKey AS ProductKey, EnglishDescription, Weight, 'This product is available for shi
        AS ShippingStatus
    FROM DimProduct WHERE ProductKey = @productKey
```

参考： IF...ELSE (Transact-SQL) - SQL Server | Microsoft Docs

条件 CASE 语句

```
CASE input_expression
     WHEN when_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

CASE语句例子

```
USE AdventureWorks2022;
GO

SELECT ProductNumber,
    Name,
    "Price Range" = CASE
        WHEN ListPrice = 0 THEN 'Mfg item - not for resale'
        WHEN ListPrice < 50 THEN 'Under $50'
        WHEN ListPrice >= 50 AND ListPrice < 250 THEN 'Under $250'
        WHEN ListPrice >= 250 AND ListPrice < 1000 THEN 'Under $1000'
        ELSE 'Over $1000'
        END
FROM Production.Product
ORDER BY ProductNumber;
GO
```

参考：CASE (Transact-SQL) - SQL Server | Microsoft Docs

CASE语句例子

```
-- 将百分制成绩转换为等级制
SELECT s.name AS 学生姓名, c.course_name AS 课程名称, t.grade AS 原始成绩,
CASE
        WHEN t.grade >= 90 THEN 'A’
        WHEN t.grade >= 80 THEN 'B’
        WHEN t.grade >= 70 THEN 'C’
        WHEN t.grade >= 60 THEN 'D’
        ELSE 'F’
END AS 成绩等级
FROM Takes t JOIN Students s ON t.student_id = s.student_id
JOIN Courses c ON t.course_id = c.course_id;
```

CASE语句例子

```
-- 根据课程类型设置学分
UPDATE Courses
SET credit =
        CASE
        WHEN course_name LIKE '%数学%' THEN 4
        WHEN course_name LIKE '%科学%' THEN 3
        WHEN course_name IN ('英语', '语文') THEN 2
        ELSE 3
        END;
```

CASE语句例子

```
-- 按课程类型自定义排序
SELECT *
FROM Courses
ORDER BY
        CASE
                WHEN course_name LIKE '%数学%' THEN 1
                WHEN course_name LIKE '%科学%' THEN 2
                WHEN course_name = '英语' THEN 3
                ELSE 4
        END;
```

循环WHILE语句

WHILE Boolean_expression

{ sql_statement | statement_block | **BREAK** | **CONTINUE** }

循环WHILE语句-例子

```
USE AdventureWorks2022;
GO
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
BEGIN
   UPDATE Production.Product
      SET ListPrice = ListPrice * 2
   SELECT MAX(ListPrice) FROM Production.Product
   IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
      BREAK
   ELSE
      CONTINUE
END
PRINT 'Too much for the market to bear';
```

参考：WHILE (Transact-SQL) - SQL Server | Microsoft Docs

SQL：游标

游标

- 游标在SQL Server中是一种**逐行处理结果集**的机制，虽然通常不推荐作为首选解决方案（因为性能较低），但在某些特定场景下仍然非常有用。

- 以下是游标的主要使用场景：
  - 当业务逻辑需要基于前一行结果进行复杂计算或决策时
  - 需要调用存储过程或函数处理每一行
  - 当需要复杂的数据转换时
  - 当需要基于不同条件执行不同SQL语句时

游标

- 游标类似于C语言中的指针，它可以指向结果集中的任意位置
- 游标可以提供在结果集中向前或向后浏览数据的功能，实现灵活操作
- SQL标准语言都是面向集合的，并没有一种描述表中单一记录的表达形式，游标即为弥补此缺点而设计的

使用游标

- 对游标的操作主要包括以下内容：
  - 声明游标 `DECLARE student_cursor CURSOR FOR SELECT student_id, name FROM Students;`
  - 打开游标 `OPEN student_cursor;`
  - 读取游标中的数据 `FETCH NEXT FROM student_cursor INTO @StudentID, @StudentName;`
  - 关闭游标 `CLOSE student_cursor;`
  - 释放游标 `DEALLOCATE student_cursor;`

SQL Server：游标的简单使用


```
-- 使用游标计算并显示每个学生的平均成绩
DECLARE @StudentID INT, @StudentName NVARCHAR(50);
DECLARE @AvgGrade DECIMAL(5,2);

-- 声明游标：获取所有学生
DECLARE student_cursor CURSOR FOR
SELECT student_id, name FROM Students;

-- 打开游标OPEN student_cursor;

-- 获取第一行数据
FETCH NEXT FROM student_cursor INTO @StudentID, @StudentName;
```

SQL Server：游标的简单使用


```
-- 循环处理每一行
PRINT '学生平均成绩报告';
PRINT '-----------------';
WHILE @@FETCH_STATUS = 0
BEGIN -- 计算该学生的平均成绩
SELECT @AvgGrade = AVG(CAST(grade AS DECIMAL(5,2)))
FROM Takes WHERE student_id = @StudentID;
--如果学生没有成绩，显示0
SET @AvgGrade = ISNULL(@AvgGrade, 0);
-- 显示结果
PRINT @StudentName + ' 的平均成绩: ' + CAST(@AvgGrade AS VARCHAR(10));
-- 获取下一行
FETCH NEXT FROM student_cursor INTO @StudentID, @StudentName; END
```

SQL Server：游标的简单使用


```
-- 循环处理每一行
PRINT '学生平均成绩报告';
PRINT '------------------';
WHILE @@FETCH_STATUS = 0 --一个系统全局变量，用于返回最后一条FETCH语句的状态（0 True -1 False）
BEGIN -- 计算该学生的平均成绩
SELECT @AvgGrade = AVG(CAST(grade AS DECIMAL(5,2)))
FROM Takes WHERE student_id = @StudentID;
        --如果学生没有成绩，显示0
        SET @AvgGrade = ISNULL(@AvgGrade, 0);
        -- 显示结果
        PRINT @StudentName + ' 的平均成绩: ' + CAST(@AvgGrade AS VARCHAR(10));
        -- 获取下一行
        FETCH NEXT FROM student_cursor INTO @StudentID, @StudentName;
END
```

SQL Server：游标的简单使用

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

```
-- 关闭并释放游标
CLOSE student_cursor;
DEALLOCATE student_cursor;

PRINT '-----------------’;
PRINT '报告结束';
```

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