

分布式文件系统及数据库技术

第五讲

主讲人：曹仔科 彭希羨
浙江大学管理学院
数据科学与管理工程学系

1、数据类型

2、数据定义

SQL 标识符

- 用于标识数据库中的对象，例如表名字，视图名和列
 - 标识符不能长于128个字符
 - 标识符必须要用字母开头
 - 标识符中不能有空格
 - ISO定义的字符集：大写字母A到Z、小写字母a到z，数字0到9和下划线字符(_)
- 每个SQL数据库供应商可能会有自己定义的标准（称其为 ‘SQL 方言’ ）

ISO SQL 标量(常量)数据类型

DATA TYPE	DECLARATIONS				
boolean	BOOLEAN				
character	CHAR	VARCHAR			
bit [†]	BIT	BIT VARYING			
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT	BIGINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION		
datetime	DATE	TIME	TIMESTAMP		
interval	INTERVAL				
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT		

[†]BIT and BIT VARYING have been removed from the SQL:2003 standard.

- 每个SQL方言也可能会与ISO标准有细微差别

SQL Server 数据类型

SQL Server 中的数据类型归纳为下列类别：

精确数字

Unicode 字符串

近似数字

二进制字符串

日期和时间

其他数据类型

字符串

<https://docs.microsoft.com/zh-cn/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-2016>

SQL Server 常用数据类型 - 整数类型

int	存储范围是-2,147,483,648到2,147,483,647之间的整数，主键列常设置此类型。 (每个数值占用 4字节)
smallint	存储范围是-32,768 到 32,767 之间的整数，用来存储限定在特定数值范围内的数据。 (每个数值占用 2 字节)
tinyint	存储范围是0到255 之间的整数，用来存储有限数目的数值。 (每个数值占用 1 字节)
bigint	存储范围是-9,223,372,036,854,775,808到 9,223,372,036,854,775,807之间的整数。 (每个数值占用 8 字节)
bit	值只能是0或1，当输入0以外的其他值时，系统均把它们当1看待。常用来表示真假、男女等二值选择。

SQL Server 常用数据类型 - 数值类型

decimal(p, s)	<p>p 为精度（有效位），表示可储存数值的最大位数，小数点左右两侧都包括在内，默认最大位为38 位；s为小数位数，标识小数点后 面所能储存的最大位数，默认最小位为0位。如：123.45,则 p=5, s=2（内存大小取决于精度p）</p> <p>在 Transact-SQL 语句中，小数数值的常量自动转换为 decimal 数据类型，在转换时，使用最小的精度和小数位数。例如，常量 12.345 被转换为 numeric 值，其精度为 5，小数位为 3。</p>
numeric(p, s)	numeric 和 decimal 是功能相同的，同是用来保存精度可变的浮点型数据。
float	<p>浮点型，它是一种近似数值类型，float(n)可储存1-53的可变精度浮点数值。（内存大小取决于精度n）</p> <p>在 Transact-SQL 语句中，在计算小数的除法时，就近进行数据类型的升级，转换为float(24)或float(53) 数据类型。</p>
money	货币型，能存储从-9220 亿到 9220 亿之间的数据，精确到小数点后四位。（每个数值占用 8 字节）

SQL Server 常用数据类型 - 日期时间

date	仅存储日期（年、月、日）。范围：0001-01-01 到 9999-12-31。存储大小：3 字节。
time(n)	仅存储时间（小时、分钟、秒、小数秒）。n指定小数秒的精度（0-7）。存储大小：3-5 字节。
datetime	储存有效日期范围是1753/1/1~9999/12/31，可精准到3.33毫秒。（每个数值占用 8 字节）
smalldatetime	储存有效日期范围是1900/1/1~2079/6/6，精确到分钟。（每个数值占用 4 字节）
datetime2(n)	datetime的扩展，具有更大的日期范围和更高的秒精度。n指定小数秒的精度（0-7）。范围：0001-01-01 到 9999-12-31。 存储大小：6-8 字节。（微软推荐使用）

日期时间的存储方式

- **核心思想：将日期和时间信息转换成数值进行存储。**
- **date(仅日期)：** 存储一个表示从基准日期（1900-01-01）开始的天数的整数。
- **time(n)(仅时间)：** 存储一个表示从午夜（00:00:00.00000000）开始经过的时间的数值（单位是纳秒）。
- **datetime2(n)(推荐的日期时间类型)：** 使用一个组合的数值存储日期和时间。基于一个公共的基准日期：0001-01-01 00:00:00.00000000。存储的值表示从基准日期时间开始经过的 100 纳秒间隔数 (也称为 tick)。日期和时间信息被统一转换为这个总的 tick 数。精度 n 决定了实际存储时对 tick 数的舍入或截断程度。

SQL Server 常用数据类型 - 字符串类型

char(m)	固定长度字符串，长度为 m。
nchar(m)	固定长度Unicode字符串，长度为 m。
varchar(m)	可变长度字符串，最大长度为m，且必须是一个介于 1 和 8,000 之间的数值。VARCHAR需要使用1或2个额外字节记录字符串的长度。
nvarchar(m)	可变长度Unicode字符串，最大长度为m，且必须是一个介于 1 和 4,000 之间的数值。
text	可变长度字符串，最大长度为 $2^{31} - 1$ 个字节。
ntext	可变长度Unicode字符串，最大长度为 $2^{30} - 1$ 个字符。

CHAR vs. VARCHAR

- 选 CHAR: 当你百分之百确定每条数据的长度都完全一样时（例如：国家代码 ISO 3166-1 alpha-2 总是 2 个字母，如 'US', 'CN', 'JP'）。这时 CHAR(2)是最高效的，没有浪费，读取也最快。
- 选 VARCHAR: 这是更常见的选择，因为现实世界中的数据长度往往变化很大（人名、地址、产品描述、评论等）。使用 VARCHAR可以显著节省存储空间，尤其是在数据平均长度远小于最大长度时。虽然理论上 CHAR的读取可能稍快，但在现代硬件和数据库优化下，这种差异通常很小，空间节省带来的好处（更少 I/O，更多数据可缓存）往往更重要。

ASCII vs. Unicode

- 计算机内部处理的所有信息最终都是 0 和 1（比特）。
- 我们需要一种方法，把字母 'A'、数字 '7'、符号 '!'、汉字 '中'、表情 '😊' 等等转换成计算机能存储和处理的数字（字节序列）。
- 字符集 (Character Set): 定义了哪些字符可以被表示（例如，“我包含了英文字母、数字和常见符号”）。
- 编码 (Encoding): 定义了字符集中的每个字符具体对应哪个（或哪些）数字/字节（例如，“'A'对应数字 65”）。

ASCII：最初的“英语世界”标准



浙江大学 管理学院
SCHOOL OF MANAGEMENT
ZHEJIANG UNIVERSITY

- 诞生背景 (1960s): 早期计算机主要在英语国家发展，需要一种标准来表示英文文本。
- 字符集很小：只定义了 128 个字符 (0到 127)。包含内容：英文字母（大写 A-Z和小写 a-z）、数字 (0-9)、基本标点符号 (.,!@#\$% 等)、控制字符（不可打印，用于控制设备，如换行 LF、回车 CR、响铃 BEL）。
- 编码方式极其简单：每个字符用 1 个字节（8 位）表示。但实际上只用了低 7 位（0-127），最高位通常是 0或用于其他目的（如早期扩展）。
- 例子： 'A' -> 65(01000001), 'a' -> 97(01100001), '0' -> 48(00110000), '!' -> 33(00100001), 换行符-> 10(00001010)

ASCII：最初的“英语世界”标准




优点：

- 简单高效：一个字符一个字节，存储和处理都非常快。
- 标准化：解决了早期混乱的局面，成为事实上的基础标准。

致命缺点：

- 只能表示英文！无法表示其他语言（法语的重音字母 é、德语的 ß、西班牙语的 ñ）、非拉丁字母（中文、日文、韩文、阿拉伯文、俄文、希腊文）、数学符号、特殊符号、表情符号等。
- 128 个字符远远不够：连基本的西欧语言都无法完整支持。

Unicode: 拥抱全球的“统一大字典”

- 诞生背景 (1990s): 互联网兴起, 计算机全球化。需要一种能表示世界上所有语言和符号的统一标准。
- 字符集极其庞大: 定义了超过 14 万个字符 (还在不断增加), 覆盖:
- 所有主要现代语言 (英语、中文、日语、韩语、法语、德语、俄语、阿拉伯语、印地语...)
- 许多历史文字和符号 (古埃及象形文字、楔形文字...)
- 大量的符号 (数学符号、货币符号、箭头、图形符号、表情符号
   ...)

Unicode: 拥抱全球的 “统一大字典”

- 码点 (Code Point): 这是 Unicode 的核心概念。每个字符被分配一个唯一的、抽象的数字编号。通常用 U+后面跟十六进制数字表示, 例如:
 - 'A' → U+0041 (十进制: 65)
 - '中' → U+4E2D (十进制: 20013)
 - '😊' → U+1F60A (十进制: 128522)

Unicode: 拥抱全球的“统一大字典”

- Unicode 本身不是编码！它只定义了字符和码点的映射关系。如何将码点转换成计算机存储的字节序列，需要编码方案。

最常见的方案是：

- UTF-8: 可变长度编码（1 到 4 个字节）。关键优势：完全兼容 ASCII！ASCII 字符（U+0000到 U+007F）在 UTF-8 中仍然用 1 个字节表示，且编码值完全相同。其他字符用 2、3 或 4 个字节表示。这使它成为互联网（HTML、电子邮件等）和现代系统的首选编码。

SQL Server 二进制字符串类型

binary(n)	固定长度的二进制数据。n定义长度（1-8000 字节）。存储大小：n字节。
varbinary(n)	可变长度的二进制数据。n定义最大长度（1-8000 字节）。存储大小：实际数据长度 + 2 字节开销。
varbinary(max)	可变长度的二进制数据。最大长度约为 $2^{31}-1$ 字节 (2GB)。存储大小：实际数据长度 + 2 字节开销（存储规则类似 varchar(max)）。

二进制字符串

- 二进制字符串类型本质上用于存储原始的、未经解释的字节序列 (Sequence of Bytes)。这与存储文本的字符类型 (如 char, varchar, nchar, nvarchar) 有根本区别。
- 字符类型：存储的是文本信息 (字母、数字、符号)。数据库知道这些字节代表什么字符 (根据指定的编码, 如 ASCII 或 Unicode), 并能进行排序、比较等文本操作。
- 二进制类型：存储的是纯粹的 0 和 1 组成的字节流。数据库不尝试去解释这些字节代表什么含义 (是图片? 是文档? 是加密数据? 它不知道, 也不关心)。它的任务就是原封不动地存储和检索这些原始字节。

用二进制字符串存储的数据

- 图像文件：.jpg, .png, .gif, .bmp等图片的二进制数据。
- 文档文件：.pdf, .docx, .xlsx, .pptx等文档的原始字节。
- 音频/视频文件：.mp3, .wav, .mp4, .avi等媒体文件的二进制内容。
- 可执行文件/程序：.exe, .dll等。
- 压缩文件：.zip, .rar等。
- 序列化对象：当应用程序需要将内存中的复杂对象（如 .NET 或 Java 对象）保存到数据库时，会先将对象序列化（转换成字节流），然后存储在 varbinary 字段中。读取时再反序列化回对象。
- 加密数据：加密算法（如哈希、对称加密、非对称加密）的输出通常是二进制数据块。这些密文或哈希值可以存储在二进制字段中。
- 特定格式的数据：一些专有格式或协议的数据包，其内部结构是二进制的。
- 指纹/生物特征数据：如指纹模板、面部识别数据等，通常以二进制格式存储。

二进制字符串

- 想象数据库有一列用来存储“描述”：
- 用 varchar：你存的是文字描述，比如“一只棕色的小狗在草地上玩耍”。数据库能理解这些文字，可以搜索包含“小狗”的记录。
- 用 varbinary：你存的是小狗照片的 .jpg 文件本身的二进制数据。数据库看到的就是一堆 01001010...（毫无意义的数字）。它无法“看”照片，也不知道照片里有什么。**只有你的图片查看器应用程序才能读取这些字节并将其显示成照片。**

数据完整性控制机制

- 在数据定义阶段（即创建或修改表结构时）SQL有数据完整性控制机制。这些机制是数据库设计的核心，用于确保存储在数据库中的数据准确、一致且符合业务规则。
- **核心目标：防止无效或不一致的数据进入数据库表**
- SQL Server 主要通过以下几种机制在 CREATE TABLE或 ALTER TABLE语句中定义数据完整性：

完整性控制机制

一组施加的约束，以避免数据库出现不一致的现象

- 必须有值的数据（即不允许为空，NULL）
- 域约束
- 实体完整性
- 缺省值设置
- 引用完整性

完整性控制机制

--创建学生信息表

```
CREATE TABLE Students (  
    student_id INT IDENTITY(1,1),  
    name VARCHAR(100),  
    age INT,  
    gender VARCHAR(10),  
    email VARCHAR(255),  
    enrollment_date DATE,  
    major VARCHAR(50),  
    gpa DECIMAL(3,2),  
)
```


完整性控制机制——必须有值的数据

- (关键字: NOT NULL)
- 例如: 定义表Students中的列major为可变长字符串, 长度最大为50, 且不能为空;

major VARCHAR(10) NOT NULL

name VARCHAR(100) NOT NULL

完整性控制机制—域约束

方法一： CREATE Type （用户定义类型+规则）

-- 创建用户定义类型

```
CREATE TYPE GPA_TYPE FROM DECIMAL(3,2);
```

-- 创建规则

```
CREATE RULE GPA_RULE AS @value BETWEEN 0.00 AND 4.00;
```

-- 绑定规则到类型

```
EXEC sp_bindrule 'GPA_RULE', 'GPA_TYPE' ;
```

- MYSQL 不支持 **CREATE TYPE**

完整性控制机制—域约束

方法二： CHECK 约束 (推荐方式)

限制列中可接受的值，使其符合特定的逻辑条件。 定义一个返回 TRUE或 FALSE 的逻辑表达式。只有使表达式为 TRUE的值才能插入或更新到列中。

可以使用各种运算符 (=, <>, <, >, <=, >=, BETWEEN, IN, LIKE, IS NULL, IS NOT NULL) 和逻辑运算符 (AND, OR, NOT) 构建复杂的业务规则

```
CHECK (age BETWEEN 16 AND 80),  
enrollment_date DATE CHECK (enrollment_date >= '2000-01-  
01'),
```

完整性控制机制—域约束

方法二： CHECK 约束 (推荐方式)

(a) : 列级 CHECK 约束, 直接放在列名后

```
CREATE TABLE Students (  
    student_id INT IDENTITY(1,1) PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    age INT CHECK (age BETWEEN 16 AND 80),  
    gender VARCHAR(10) CHECK (gender IN ('Male', 'Female', 'Other')),  
    email VARCHAR(255) CHECK (email LIKE '%@%.%'),  
    enrollment_date DATE CHECK (enrollment_date >= '2000-01-01'),  
    gpa DECIMAL(3,2) CHECK (gpa BETWEEN 0.00 AND 4.00)  
);
```

方法二： CHECK 约束 (推荐方式)

(b)：表级 CHECK 约束，放在所有列后

CREATE TABLE Students (

student_id INT IDENTITY(1,1) PRIMARY KEY,

name VARCHAR(100) NOT NULL,

age INT,

gender VARCHAR(10),

email VARCHAR(255),

enrollment_date DATE,

major VARCHAR(50),

gpa DECIMAL(3,2),

-- 域约束作为表级 CHECK 约束

CONSTRAINT CHK_Student_Age CHECK (age BETWEEN 16 AND 80),

CONSTRAINT CHK_Student_Gender CHECK (gender IN ('Male', 'Female', 'Other')),

CONSTRAINT CHK_Student_Email CHECK (email LIKE '%@%.%' AND LEN(email) >= 5),

CONSTRAINT CHK_Student_GPA CHECK (gpa BETWEEN 0.00 AND 4.00),

CONSTRAINT CHK_Student_EnrollmentDate CHECK (enrollment_date >= '2000-01-01')

);

注：后续可以修改CONSTRAINT

方法二： CHECK 约束（推荐方式） 后续可以修改CONSTRAINT

- 可以在**CREATE TABLE**和**ALTER TABLE**时使用**CONSTRAINT**来定义所有的约束。
- 使用**CONSTRAINT**的一个优势是可以更灵活地删除或更改约束来适应业务规则的变化。
- 向现有表中**添加约束**：**ALTER TABLE ... ADD CONSTRAINT ...**
- **删除约束**：**ALTER TABLE ... DROP CONSTRAINT ...**

完整性控制机制——实体完整性

- **实体完整性**

一个表中每一行的主码值必须是非空且唯一(unique)的

在**CREATE TABLE** 中定义主码:

PRIMARY KEY(student_id)

PRIMARY KEY(course_id)

PRIMARY KEY(student_id, course_id)

注：单个属性也可以直接将PRIMARY KEY放在列名后

完整性控制机制—实体完整性

每个表只能有一个**PRIMARY KEY**子句

但可以用关键字**UNIQUE**保证其他列的唯一性，例如Students表中：
UNIQUE(email)

注意：**UNIQUE**子句中出现的每个列必须被声明为**NOT NULL**

- 一个表只能有一个主键，但可以有多多个唯一约束。
- 主键通常用于标识行，唯一约束用于确保业务关键数据的唯一性（如身份证号、邮箱，即使它们不是主键）。

完整性控制机制

- 引用完整性

外部关键词必须是母表（即被引用的表）中已存在的有效元组

在**CREATE** 和 **ALTER TABLE** 中定义外码:

FOREIGN KEY(branchNo) REFERENCES Branch

完整性控制机制

- **默认值定义 (DEFAULT) :**

- 当向表中插入新行时，如果用户没有为某列指定值，则自动为该列提供一个预定义的值。确保数据完整性，特别是对于不允许 NULL 的列。
- 为列指定一个常量值、表达式（如 GETDATE()）或系统函数。
- 在age列定义中指定 DEFAULT为18。
- age DEFAULT 18

完整性控制机制—引用完整性



```
CREATE TABLE Takes (  
    student_id INT,  
    course_id INT,  
    grade INT,  
    CONSTRAINT FK_takes_student  
        FOREIGN KEY (student_id) REFERENCES Students(student_id),  
    CONSTRAINT FK_takes_course  
        FOREIGN KEY (course_id) REFERENCES Courses(course_id)  
);
```

当对子表进行INSERT和UPDATE操作会创建与母表中候选码不匹配的值时，SQL会拒绝该操作（即为保证引用完整性）

若对母表进行UPDATE和DELETE操作会更新或删除与子表有匹配的候选码值时，SQL将根据FOREIGN KEY子句中的ON UPDATE 或ON DELETE子句来决定如何操作

完整性控制机制

- 当用户企图删除母表中某行时，子表中有一个或多个匹配行，SQL子句**ON DELETE**有四种选择：
 - **CASCADE**：自动删除子表中匹配的行。由于删除的行可能有候选码为另一个表的外码，这个规则会以级联方式相继触发
 - **SET NULL**：设置子表中的外码值为NULL；只有当外码定义时没有指定为NOT NULL时才有效
 - **SET DEFAULT**：设置子表中的外码值为默认值；只有当外码定义时指定了默认值才有效
 - **NO ACTION**：拒绝对母表进行删除操作（这也是默认设置）

完整性控制机制

- 当用户企图更新母表中某行时，子表中有一个或多个匹配行，SQL子句**ON UPDATE**同样有四种选择：
 - **CASCADE**：自动更新子表中匹配的行。如果子表中更新的列又是另一个表的外码，就会依次更新
 - **SET NULL**：设置子表中的外码值为NULL；只有当外码定义时没有指定为NOT NULL时才有效
 - **SET DEFAULT**：设置子表中的外码值为默认值；只有当外码定义时指定了默认值才有效
 - **NO ACTION**：拒绝对母表进行更新操作（这也是默认设置）

完整性控制机制—引用完整性

- **FOREIGN KEY (student_id) REFERENCES Students
ON DELETE SET NULL**

当 Students 表中的某个学生被删除时, Takes 表中所有引用该学生的记录的 student_id 会被自动设置为 NULL, 选课记录不会删除, 但会变成"无主"记录

- **FOREIGN KEY (course_id) REFERENCES Courses
ON UPDATE CASCADE**

当 Courses 表中的某个课程的 course_id 被更新时, Takes 表中所有引用该课程的记录的 course_id 会自动同步更新

数据定义

- DDL允许创建和删除模式、域、表、视图和索引等数据库对象
- SQL的主要数据定义语句:

CREATE SCHEMA
CREATE/ALTER TYPE
CREATE/ALTER TABLE
CREATE VIEW

DROP SCHEMA
DROP TYPE
DROP TABLE
DROP VIEW

- 许多 DBMSs 还提供:

CREATE INDEX

DROP INDEX

删除/添加约束

- **DROP/ADD CONSTRAINT**可以移除指定约束或者添加新的约束。
- 当试图删除之前创建的学生表中的某个课程记录时，外键约束会拒绝操作。
- 可以通过DROP/ADD CONSTRAINT来更新外键约束使得这样的操作可行。

删除/添加外键约束

```
DELETE FROM Courses
```

```
WHERE course_id = 7;
```

--直接删除会出现错误，由于外键约束的默认操作是NO ACTION

```
ALTER TABLE Takes
```

```
DROP CONSTRAINT FK_takes_course;
```

```
ALTER TABLE Takes
```

```
ADD CONSTRAINT FK_takes_course_new
```

```
FOREIGN KEY (course_id) REFERENCES Courses(course_id) ON DELETE CASCADE;
```

```
DELETE FROM Courses
```

```
WHERE course_id = 7;
```

--设置了级联删除后，课程记录可以被删除，而且选课表中有关该课程的选课记录也会被删除

数据库模式

- **数据库模式 (Database Schema) 是数据库的结构化描述, 包含:**
 - 1. 所有数据库对象的定义: 表(Table)的结构、视图(View)的逻辑关系、索引(Index)的组织方式
 - 2. 数据之间的关系: 表与表之间的关联 (外键)、数据约束规则 (如: 年龄不能为负数)
 - 3. 完整性约束: 主键约束 (身份证号不能重复)、检查约束 (成绩必须在0-100之间)

CREATE SCHEMA

- 在 SQL Server 中，CREATE SCHEMA语句用于创建一个数据库架构 (Schema) 。
- 架构是数据库内部的一个逻辑容器，用于组织和管理数据库对象（如表、视图、存储过程、函数等），并提供权限控制的边界。
- 将相关的数据库对象（如表、视图、函数）分组到一个命名的架构下，可以提高数据库结构的清晰度和可维护性。
 - 可以将 Sales相关的表 (Orders, Customers, Products) 放入 sales架构。
 - 将 HR相关的表 (Employees, Departments, Salaries) 放入 hr架构。

CREATE SCHEMA

- **CREATE SCHEMA schema_name [AUTHORIZATION owner_name]**
 - 创建架构Sales，并授权给某个用户，则对应的SQL语句为：
CREATE SCHEMA Sales AUTHORIZATION Xixian_Peng
 - **AUTHORIZATION owner_name**: 可选。指定拥有此架构的数据库级主体（用户、角色）。如果省略，执行 CREATE SCHEMA语句的用户将成为所有者。
- **DROP SCHEMA Name [RESTRICT | CASCADE]**
 - RESTRICT, 此为默认值，必须是空，否则删除失败
 - CASCADE: 按照定义的顺序级联地删除与该对象相关的所有对象

CREATE SCHEMA

-- 创建基本架构

```
CREATE SCHEMA Sales
```

```
GO
```

```
CREATE SCHEMA HR;
```

```
GO
```

-- 创建企业资源规划系统的多个架构

```
CREATE SCHEMA HR
```

```
AUTHORIZATION dbo
```

```
CREATE TABLE Employees (
```

```
    employee_id INT IDENTITY PRIMARY KEY,
```

```
    first_name VARCHAR(50) NOT NULL,
```

```
    last_name VARCHAR(50) NOT NULL,
```

```
    hire_date DATE,
```

```
    salary DECIMAL(10,2)
```

```
)
```

```
CREATE TABLE Departments (
```

```
    department_id INT IDENTITY PRIMARY KEY,
```

```
    department_name VARCHAR(100) NOT NULL
```

```
);
```

```
GO
```

```
CREATE SCHEMA Sales
```

```
AUTHORIZATION dbo
```

```
CREATE TABLE Customers (
```

```
    customer_id INT IDENTITY PRIMARY KEY,
```

```
    company_name VARCHAR(100) NOT NULL,
```

```
    contact_name VARCHAR(100)
```

```
)
```

```
CREATE TABLE Orders (
```

```
    order_id INT IDENTITY PRIMARY KEY,
```

```
    customer_id INT,
```

```
    order_date DATE,
```

```
    total_amount DECIMAL(10,2)
```

```
);
```

```
GO
```

创建表

```
CREATE TABLE TableName  
({colName dataType [NOT NULL] [UNIQUE]  
[DEFAULT defaultOption]  
[CHECK searchCondition] [...]}  
[PRIMARY KEY (listOfColumns),  
{[UNIQUE (listOfColumns),] [...,]}  
{[FOREIGN KEY (listOfFKColumns)  
REFERENCES ParentTableName [(listOfCKColumns)],  
[ON UPDATE referentialAction]  
[ON DELETE referentialAction ]] [...]}  
{[CHECK (searchCondition)] [...] })
```

修改表定义

- 在表中添加一个新列
- 在表中删除一个列
- 添加一项新的表的约束
- 删除一项表约束
- 设置列默认值
- 删除列默认值

修改表定义

ALTER TABLE table_name
[**ADD [COLUMN]** column_name dataType [**NOT NULL**] [**UNIQUE**]
[**DEFAULT defaultOption**] [**CHECK (searchCondition)**]

[**DROP [COLUMN]** column_name [**RESTRICT|CASCADE**]]

[**ADD [CONSTRAINT [ConstraintName]]** TableConstraintDefinition]

[**DROP CONSTRAINT** ConstraintName [**RESTRICT|CASCADE**]]

[**ALTER [COLUMN]** **SET DEFAULT** defaultOption]

[**ALTER [COLUMN]** **DROP DEFAULT**]

修改表定义

- 对于Students表中的age属性进行修改

-- 首先, 删除已存在的默认约束 (如果知道约束名)

```
ALTER TABLE Students DROP CONSTRAINT DF_Students_Age;
```

-- 然后, 为Age字段设置新的默认约束

```
ALTER TABLE Students ADD CONSTRAINT DF_Students_Age  
DEFAULT 18 FOR Age;
```

-- 添加一个检查约束, 确保年龄在16到80之间

```
ALTER TABLE Students ADD CONSTRAINT CK_Students_Age  
CHECK (Age BETWEEN 16 AND 80);
```

修改表定义

- 利用ALTER添加外键约束

```
ALTER TABLE Takes  
ADD CONSTRAINT FK_Takes_Course  
FOREIGN KEY (course_id) REFERENCES Courses(course_id)  
ON DELETE NO ACTION    -- 阻止删除有选课记录的课程  
ON UPDATE CASCADE;    -- 课程ID更新时同步更新选课记录
```

删除表

DROP TABLE TableName [RESTRICT | CASCADE]

例如：DROP TABLE Takes;

- 注意：这个命令不仅删除指名的表，还删除了所有的行。
- RESTRICT：如果存在任何其他对象依赖于将要删除的表，则拒绝进行Drop操作
- CASCADE:存在依赖的情况下，也允许进行DROP操作，只是同时自动删除所有依赖的对象，包括依赖于这些对象的对象（慎用！）。

谢谢！ 下次见！