**《数据库系统原理》**

**课程实验指导书**

**北京邮电大学**

**计算机学院**

**2021年3月**

**目录**

[第一章 概述 6](#_Toc66958133)

[第二章 数据库实验平台的安装、数据库创建与维护实验 9](#_Toc66958134)

[实验目的 9](#_Toc66958135)

[实验环境 9](#_Toc66958136)

[实验内容与步骤 9](#_Toc66958137)

[2.1 Microsoft SQL Server数据库安装 9](#_Toc66958138)

[2.2 熟悉Microsoft SQL Server数据库基本用户界面： 11](#_Toc66958139)

[2.3 使用Microsoft SQL Server创建数据库: 12](#_Toc66958140)

[2.4 Microsoft SQL Server数据库维护 12](#_Toc66958141)

[实验要求 14](#_Toc66958142)

[实验总结 14](#_Toc66958143)

[第三章 E-R建模与关系表转换实验 21](#_Toc66958144)

[实验目的 21](#_Toc66958145)

[实验原理 21](#_Toc66958146)

[实验环境 21](#_Toc66958147)

[实验内容与步骤 21](#_Toc66958148)

[实验要求 22](#_Toc66958149)

[实验总结 22](#_Toc66958150)

[第四章 数据库表/视图的创建与维护实验 23](#_Toc66958151)

[实验目的 23](#_Toc66958152)

[实验环境 23](#_Toc66958153)

[实验内容与步骤 23](#_Toc66958154)

[实验要求 25](#_Toc66958155)

[实验总结 25](#_Toc66958156)

[第五章 数据查询与修改实验 26](#_Toc66958157)

[实验目的 26](#_Toc66958158)

[实验环境 26](#_Toc66958159)

[实验内容 26](#_Toc66958160)

[实验步骤 26](#_Toc66958161)

[5.1 Basic structure of SQL Queries 26](#_Toc66958162)

[5.2 Additional Basic Operations 26](#_Toc66958163)

[5.3 Set Operation 27](#_Toc66958164)

[5.4 Null Values 27](#_Toc66958165)

[5.5 Aggregate Functions 27](#_Toc66958166)

[5.6 Nested (嵌套) Subqueries 27](#_Toc66958167)

[5.7 综合---查询语句 28](#_Toc66958168)

[5.8 Modification of the Database 28](#_Toc66958169)

[实验要求 29](#_Toc66958170)

[实验总结 29](#_Toc66958171)

[第六章 数据库完整性与安全性实验 30](#_Toc66958172)

[实验目的 30](#_Toc66958173)

[实验环境 30](#_Toc66958174)

[实验内容 30](#_Toc66958175)

[6.1 完整性实验与要求 30](#_Toc66958176)

[6.2 安全性实验内容与要求 33](#_Toc66958177)

[实验总结 40](#_Toc66958178)

[第七章 数据库访问接口实验 41](#_Toc66958179)

[实验目的 41](#_Toc66958180)

[实验原理 41](#_Toc66958181)

[实验环境 42](#_Toc66958182)

[实验内容 42](#_Toc66958183)

[实验要求 42](#_Toc66958184)

[实验步骤 44](#_Toc66958185)

[7.1 ODBC访问 44](#_Toc66958186)

[7.2 JDBC接口实验 52](#_Toc66958187)

[实验总结 57](#_Toc66958188)

[第八章 数据库物理设计实验 58](#_Toc66958189)

[实验目的 58](#_Toc66958190)

[实验环境 58](#_Toc66958191)

[实验要求 58](#_Toc66958192)

[实验内容与步骤 59](#_Toc66958193)

[8.1 数据库文件及文件组的设计 59](#_Toc66958194)

[8.2 数据库索引的设计 63](#_Toc66958195)

[8.3 定长变长数据类型对数据访问影响分析 76](#_Toc66958196)

[实验总结 82](#_Toc66958197)

[第九章 数据查询分析优化实验 83](#_Toc66958198)

[实验目的 83](#_Toc66958199)

[实验环境 83](#_Toc66958200)

[实验内容 84](#_Toc66958201)

[9.1 SQLServer执行计划的查看与分析 84](#_Toc66958202)

[9.2 观察视图查询、with临时视图查询的执行计划 86](#_Toc66958203)

[9.3 优化SQL语句 88](#_Toc66958204)

[实验总结 103](#_Toc66958205)

[第十章 数据库备份与恢复实验 104](#_Toc66958206)

[实验目的 104](#_Toc66958207)

[实验环境 104](#_Toc66958208)

[实验内容与步骤 104](#_Toc66958209)

[10.1 校验数据库 104](#_Toc66958210)

[10.2 备份数据库 105](#_Toc66958211)

[10.3 恢复数据库 105](#_Toc66958212)

[10.4 备份恢复方法比较 114](#_Toc66958213)

[10.5 创建维护计划 114](#_Toc66958214)

[实验总结 115](#_Toc66958215)

[第十一章 事务及其并发控制实验 116](#_Toc66958216)

[实验目的 116](#_Toc66958217)

[实验环境 117](#_Toc66958218)

[实验背景 117](#_Toc66958219)

[实验内容与要求 123](#_Toc66958220)

[11.1 单事务/串行事务提交与回滚 123](#_Toc66958221)

[11.2 事务并发控制机制 142](#_Toc66958222)

[11.3 事务日志观察 152](#_Toc66958223)

[实验总结 158](#_Toc66958224)

[附录一 实验平台及T-SQL介绍 159](#_Toc66958225)

[一、数据库系统实验平台简介（略） 159](#_Toc66958226)

[二、T-SQL简介 159](#_Toc66958227)

[三、T-SQL数据类型 164](#_Toc66958228)

[四、变量 170](#_Toc66958229)

[五、注释运算符与通配符 172](#_Toc66958230)

[六、流程控制命令 174](#_Toc66958231)

[七、常用命令简介 179](#_Toc66958232)

[八、常用函数简介 185](#_Toc66958233)

[附录二 ODBC与JDBC介绍 212](#_Toc66958234)

[一、应用程序使用ODBC接口的详细步骤 212](#_Toc66958235)

[二、ODBC API函数介绍 213](#_Toc66958236)

[三、JDBC类、JDBC API函数介绍 216](#_Toc66958237)

[附录三 备份和恢复 218](#_Toc66958238)

[一、数据备份 218](#_Toc66958239)

[二、数据恢复 220](#_Toc66958240)

[三、检查点（check-point） 220](#_Toc66958241)

[附录四 多连接多事务访问数据库示例 221](#_Toc66958242)

# 概述

《数据库系统原理》课程学习需要理论和实践紧密结合，课程实验是其中重要的教学环节。本指导书介绍了数据库基本课程实验相关内容，包括数据库实验所需基础知识、实验环境、实验内容和实验要求等。

《数据库系统原理》课程学习的目标是：

1. 了解和掌握数据库的理论知识。学习理解数据库系统的基本概念，掌握数据库系统的基本理论，包括数据库的特点、结构、关系数据模型、数据存储、数据查询及优化、关系数据库设计、事务管理等内容。
2. 了解和熟悉实际数据库的操作和使用。在学生掌握基本的数据库概念和理论基础上，通过基本课程实验培养学生的实践能力，使学生能掌握基本的数据库操作和数据访问方法。
3. 全面理解数据库系统设计与开发过程，通过课程综合设计完成一个简单但完整的数据库应用系统的设计与实现。

本基本课程实验指导书包括以下几方面内容。

1. **课程实验目的**：

学生可以通过实验加深对数据库相关基本理论和概念的认识和理解，并且通过对Microsoft SQL Server数据库的操作和使用，了解具体商用数据库管理系统，熟悉数据库操作，提高数据库实践能力。

1. **课程实验环境：**

实验选择商用Microsoft SQL Server数据库作为基本实验环境，操作系统平台为Windows2000/XP/NT/Win7。利用商用数据库系统提供的管理工具，如企业管理器Enterprise Manager和查询分析器Query Analysis等，以及Transact-SQL语言，学生可以完成相关实验。

1. **实验方式和要求：**

学生分组完成各个基本课程实验。每组学生3-4人。

学生需要理解每次实验内容，掌握实验所需基本知识，在此基础上制定实验步骤，完成实验内容；

基本课程实验包括多组实验。各组实验虽然分次完成，但内容上相互衔接，前一次实验数据可能为后面实验所用，因此需要保留每一次实验结果。

实验完成后，学生需要完成实验报告，报告内容包括实验目的、内容、步骤、结果和实验总结；实验指导教师负责每次实验的上机验收。

1. **实验内容设置：** 在下述几组实验中选取若干组

实验一 数据库安装、数据库创建与维护

实验二 E-R建模与关系表转换【略】

实验三 数据库表/视图的创建与维护

实验四 数据查询与修改

实验五 数据库完整性与安全性

实验六 数据库访问接口

实验七 数据库物理设计

实验八 数据查询分析优化

实验九 数据库备份与恢复

实验十 事务及其并发控制

1. **实验报告要求**：

实验报告需要完成以下内容：

1. 实验目的：

描述本次实验的目的。

1. 实验原理：

根据自己的理解，描述本次实验的原理。

1. 实验平台及环境：

介绍自己实验所基于的软硬件环境及所用到的工具。

1. 实验内容：

介绍本次实验的具体内容。

1. 实验步骤：

根据实验内容和实验过程情况介绍实验步骤。

1. 实验小结：

分析实验结果，总结本次实验遇到的问题和心得。

# 数据库实验平台的安装、数据库创建与维护实验

### 实验目的

1．通过对Microsoft SQL Server数据库的安装和简单使用：

1. 了解安装Microsoft SQL Server数据库的软硬件环境和安装方法；
2. 熟悉Microsoft SQL Server数据库相关使用；
3. 熟悉Microsoft SQL Server数据库的构成和相关工具；
4. 通过Microsoft SQL Server数据库的使用来理解数据库系统的基本概念。

2．通过创建数据库、并进行相应的维护，了解并掌握Microsoft SQL Server数据库数据库的创建和维护的不同方法和途径，并通过这一具体的数据库理解实际数据库所包含的各要素。

### 实验环境

采用Microsfot SQL Server数据库管理系统作为实验平台，可以选用Microsoft SQL Server 2008及以后的某一版本，可以在“北邮人BT上下载”。

### 实验内容与步骤

#### 2.1 Microsoft SQL Server数据库安装

1．下载安装SQL Server。

下载相应版本。选择实验室Windows XP或者Win7系统或者虚拟机（vmware）上自行安装的操作系统或者是自己的机器之一安装该版本的Microsoft SQL Server。可选版本有Microsoft SQL Server 2008/2012/2014或者2016的Express，Develop，Standard或者Enterprise等各版本。

注意：安装前最好下载微软对于安装说明文件，并仔细阅读，看某版本的数据库软件是否兼容我们使用的操作系统，是否需要打补丁，以及需要什么硬件环境等。SQL Server数据库安装作为一个操作过程，是比较简单的。但因为计算机软硬件是一个复杂系统，有各种约束条件（比如操作系统是否兼容，软件是否有冲突，系统是否已经损坏，是否有病毒等等），安装时对付它们才是最复杂和困难的。

安装时通常可同时安装SQL Server Management Studio Express，以后建立数据库等很方便。

以下以在windows Server 2012上安装SQL Server 2012标准版为例讲解安装数据库的步骤：

1. 下载相关Sql Server软件。可以从数据库实验服务器上下载，也可以从北邮微软正版软件网站下载。
2. 将ISO文件装载为虚拟驱动器：在资源管理器中找到ISO文件，然后鼠标右击该ISO文件，选择安装成虚拟驱动器，然后系统自然会将ISO文件装载成一个虚拟驱动器，内容如图2.1：

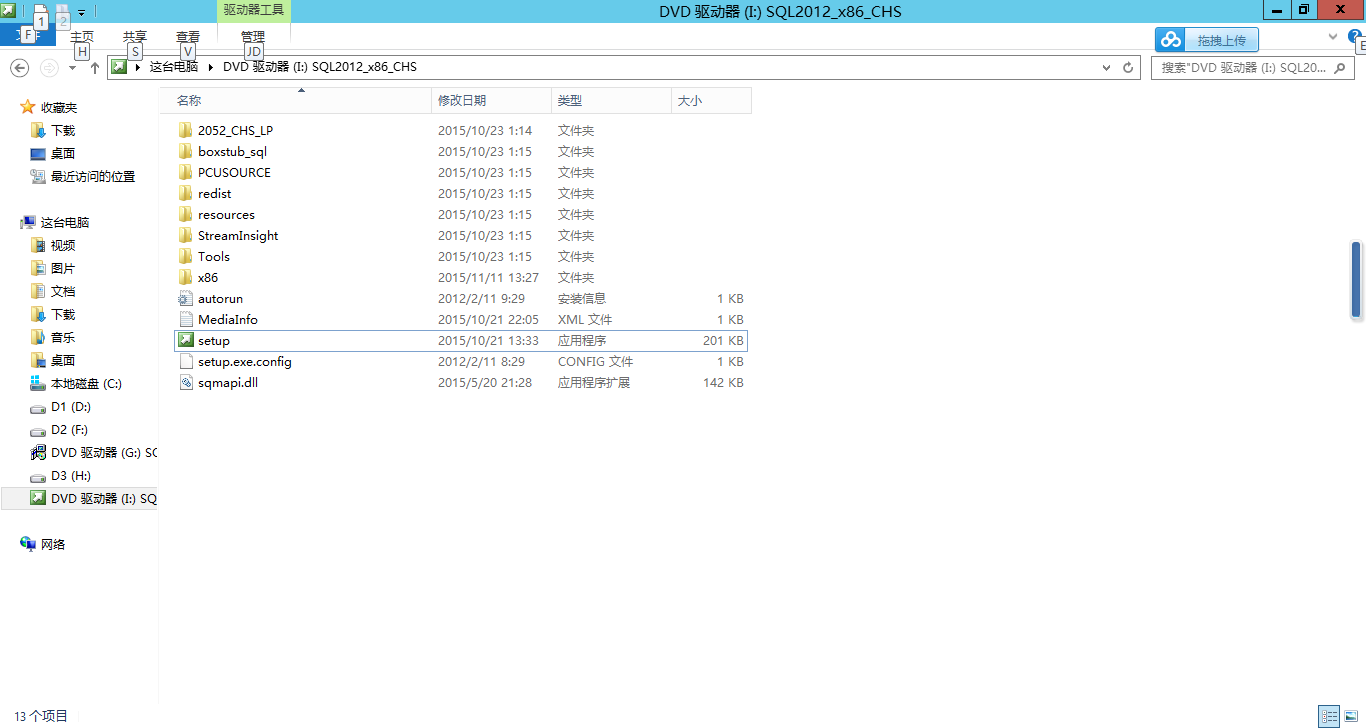


图2.1 虚拟驱动器

1. 点击setup，开始安装，出现如下图2.2界面。值得注意的是软件和硬件需求等信息，应该先仔细查看自己的系统是否满足安装要求。
2. 选择安装，如图2.3，然后再选择全新SQL Server独立安装或向现有安装添加功能，有一个安装程序支持规则界面，没有问题直接通过，否则按照说明进行修正。然后可到达如图2.4的界面。
3. 直接选择下一步，接受许可条款后，再选择下一步，如图2.5。可选择SQL Server功能安装，进行自定义功能安装。
4. 然后如图2.6，选择安装所有功能，免得实验中缺少什么，需要重装。
5. 如图2.7，安装SQL Server下的实例，一个实例下可以创建多个数据库。
6. 几个页面以后，图2.8中身份认证方式要改成混合身份认证。设置sa账户密码（请牢记该密码）并且添加管理员用户（可以选择当前用户）。在下一个添加管理员用户界面中也可以一样添加。
7. 进入图2.9，选择安装和配置。
8. 最后是安装进度和完成。如图2.10。



图2.2 SQL Server安装中心

2．练习启动和停止数据库服务。

打开控制面板|管理工具|服务，找出与所使用数据库相关的服务，尝试停止和启动它们。如图2.11.

最后保持这些服务为启动状态。如果这些服务没有启动，非常可能导致数据库系统无法工作，从而后续实验无法进行。

#### 2.2 熟悉Microsoft SQL Server数据库基本用户界面：

1. 熟悉数据库管理系统的基本用户界面，启动SQL Server Management Studio。登录界面可采用windows身份认证，如图2.12。然后有如图2.13所示。
2. 点击左边窗口的各种数据库、视图、表等的对象，查看状态等。
3. 打开文件|新建|数据库|使用当前链接的查询或者数据引擎查询，可以进入命令行界面。请尝试输入SQL语句以及其它命令。

#### 2.3 使用Microsoft SQL Server创建数据库:

1. Microsfot SQL Server数据库创建

了解数据库系统的命令行操作界面，创建数据库，并且指定数据库存储位置和文件大小：

1. 编写包含CREATE DATABASE命令的脚本文件，数据库SQL Server Management Studio下打开该文件并执行之。该命令的语法是：

CREATE DATABASE database\_name CONTAINMENT = { NONE | PARTIAL } ]

[ ON [ PRIMARY ] <filespec> [ ,...n ] [ , <filegroup> [ ,...n ] ]

[ LOG ON <filespec> [ ,...n ] ] ] [ COLLATE collation\_name ] [ WITH <option> [,...n ] ] [;] <option> ::= { FILESTREAM ( <filestream\_option> [,...n ] ) | DEFAULT\_FULLTEXT\_LANGUAGE = { lcid | language\_name | language\_alias }

| DEFAULT\_LANGUAGE = { lcid | language\_name | language\_alias }

1. SQL Server Management Studio下使用向导创建数据库。并将创建数据库脚本保存到文件中。打开文件查看相应命令。理解用户界面上相关参数，指定数据库存储位置和存储文件大小。后续实验也同样的学习脚本和理解参数的要求。
2. 找出系统数据库，查看其下的表及其它数据库对象。

#### 2.4 Microsoft SQL Server数据库维护

1. 对数据库属性和参数进行查询、相应的修改和维护，内容包括：

使用ALTER DATABASE下面的操作：

* 1. 调整数据库的大小，完成数据库大小的增加、减小；
  2. 增加文件组；
  3. 修改日志文件的最大值；
  4. 查看数据库的属性值（用图形界面）。

ALTER DATABASE语法是：

ALTER DATABASE database\_name

{

<add\_or\_modify\_files>

| <add\_or\_modify\_filegroups>

}

[;]

<add\_or\_modify\_files>::=

{

ADD FILE <filespec> [ ,...n ]

[ TO FILEGROUP { filegroup\_name } ]

| ADD LOG FILE <filespec> [ ,...n ]

| REMOVE FILE logical\_file\_name

| MODIFY FILE <filespec>

}

<filespec>::=

(

NAME = logical\_file\_name

[ , NEWNAME = new\_logical\_name ]

[ , FILENAME = {'os\_file\_name' | 'filestream\_path' | 'memory\_optimized\_data\_path' } ]

[ , SIZE = size [ KB | MB | GB | TB ] ]

[ , MAXSIZE = { max\_size [ KB | MB | GB | TB ] | UNLIMITED } ]

[ , FILEGROWTH = growth\_increment [ KB | MB | GB | TB| % ] ]

[ , OFFLINE ]

)

<add\_or\_modify\_filegroups>::=

{

| ADD FILEGROUP filegroup\_name

[ CONTAINS FILESTREAM | CONTAINS MEMORY\_OPTIMIZED\_DATA ]

| REMOVE FILEGROUP filegroup\_name

| MODIFY FILEGROUP filegroup\_name

{ <filegroup\_updatability\_option>

| DEFAULT

| NAME = new\_filegroup\_name

| { AUTOGROW\_SINGLE\_FILE | AUTOGROW\_ALL\_FILES }

}

}

<filegroup\_updatability\_option>::=

{

{ READONLY | READWRITE }

| { READ\_ONLY | READ\_WRITE }

}

1. 数据库的删除
   1. 使用脚本文件删除数据库。

命令 drop database

* 1. 使用SQL Server Management Studio删除数据库

直接右击该数据库，选择删除命令。

### 实验要求

1. Microsoft SQL Server数据库的安装实验要求学生在微机上安装Microsoft SQL Server数据库数据库系统，为后续各个实验搭建实验环境。
2. 数据库创建与维护实验以LTE网络配置数据库为背景，要求面向具体应用领域，利用Microsoft SQL Server数据库相关机制，创建并维护数据库系统，为后续各个实验提供前期准备。
3. 要求学生根据以上要求确定实验步骤，独立完成以上实验内容。并在安装和数据库运行后熟悉Microsoft SQL Server数据库的各种运行管理。
4. 对实验结果进行总结和分析；给出完成以上实验内容的Transact\_SQL语句，并对相应的SQL语句进行分析；说明建立的数据库的相关内容，和维护后的相关属性的改变。
5. 实验完成后完成实验报告。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。



图2.3 SQL Server安装选项界面

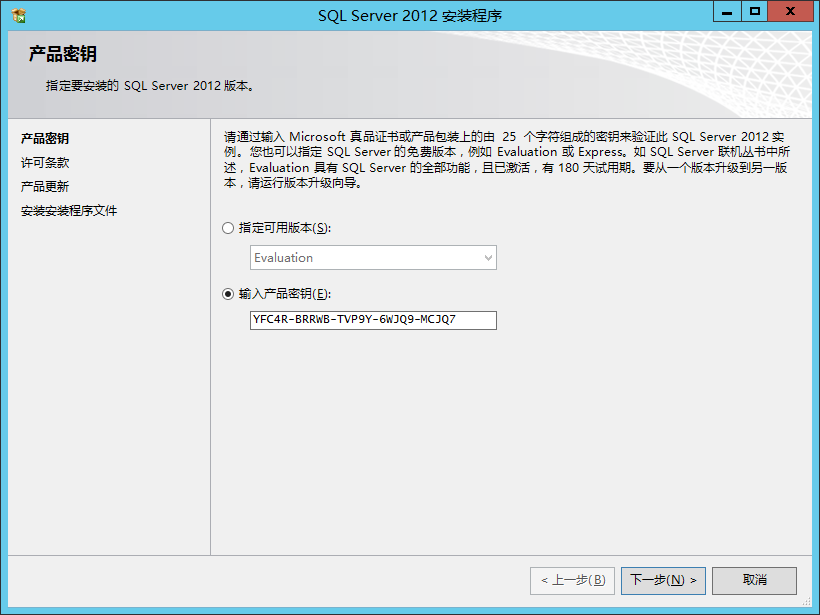


图2.4 安装界面

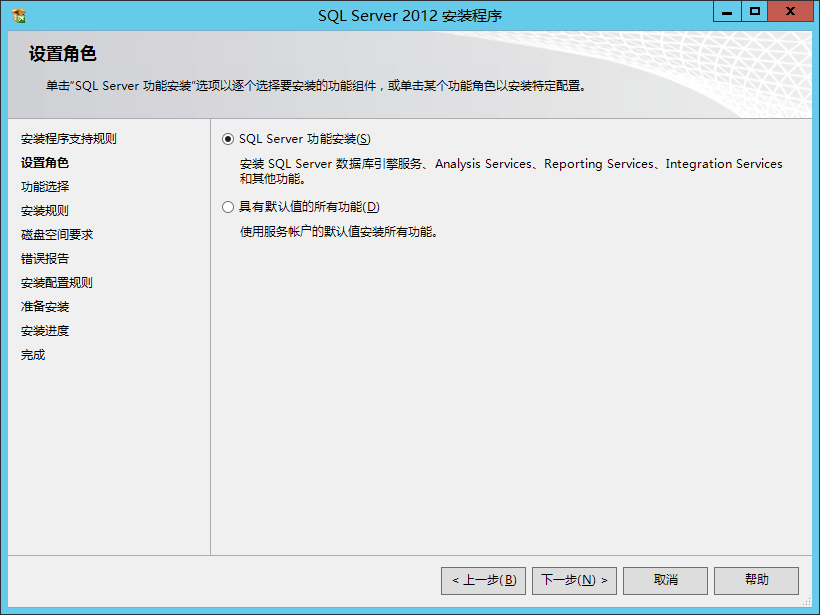


图2.4 SQL Server功能安装

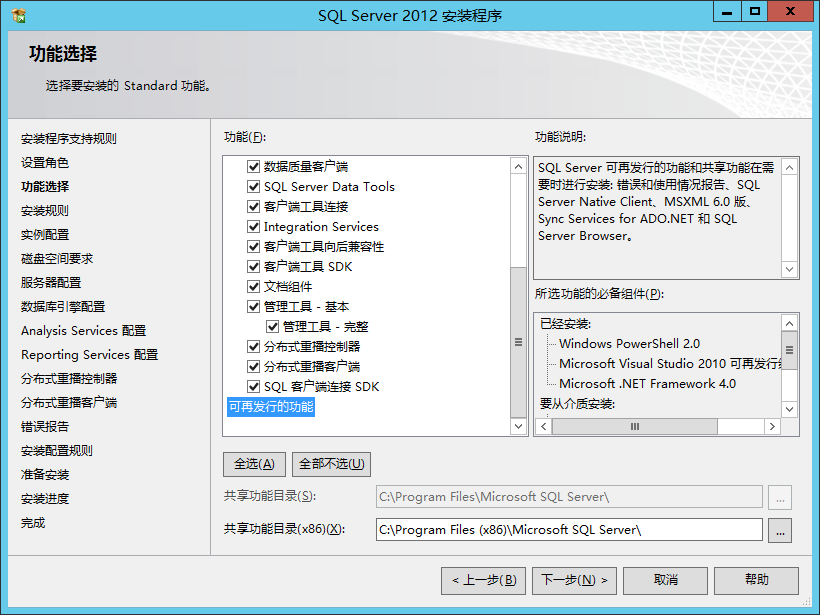


图2.6 选择所有功能和共享目录

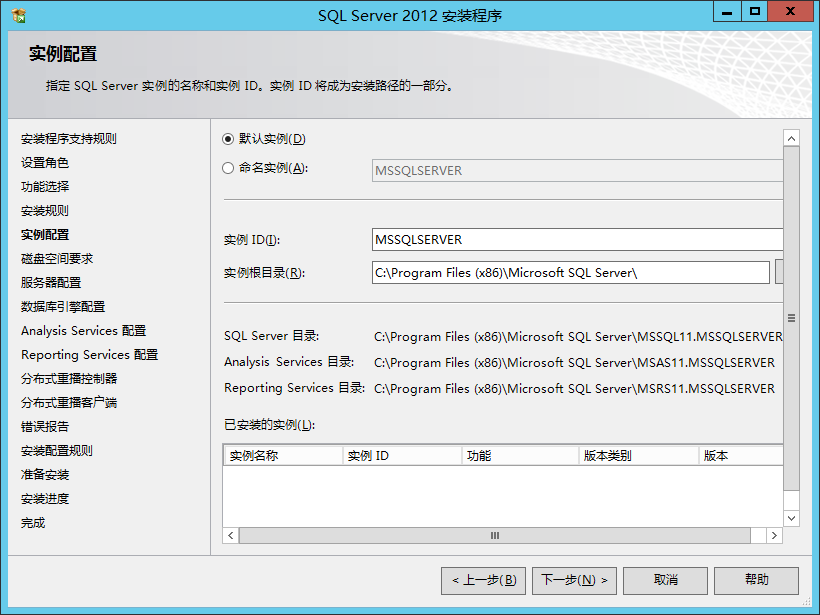


图2.7 安装SQL Server实例

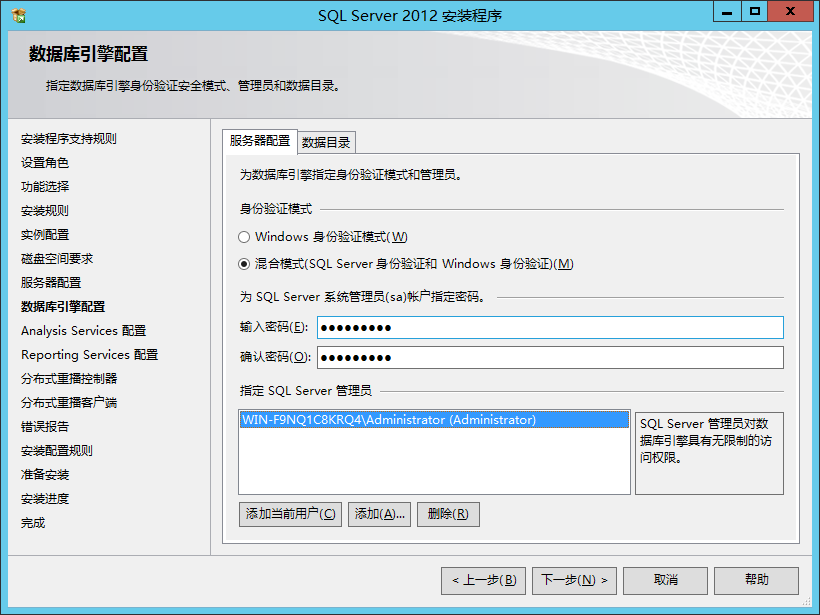


图2.8 SQL Server身份认证，选择混合模式

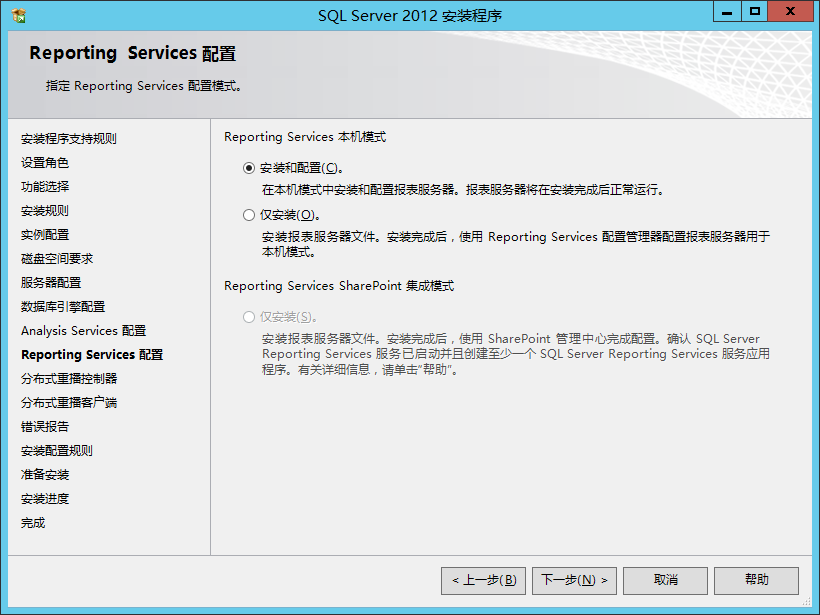


图2.9 安装和配置SQL Server

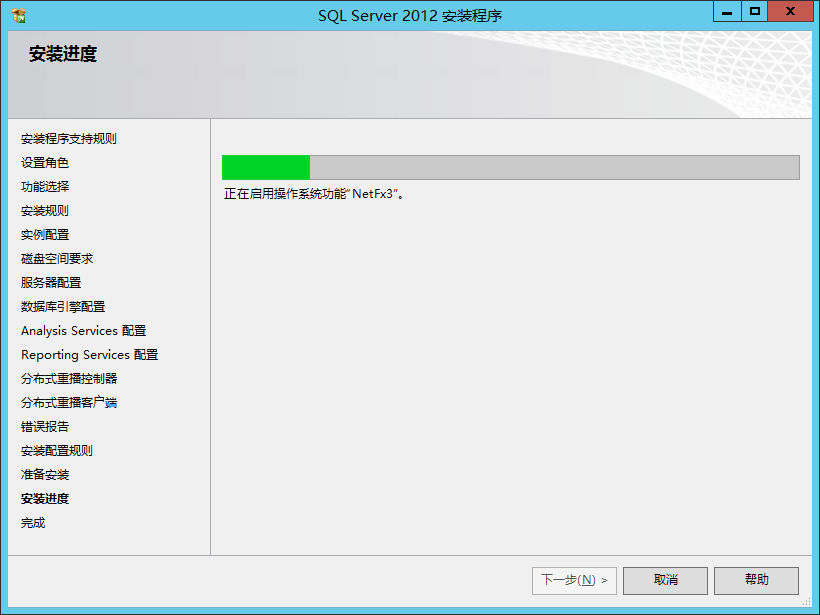


图2.10 安装进度

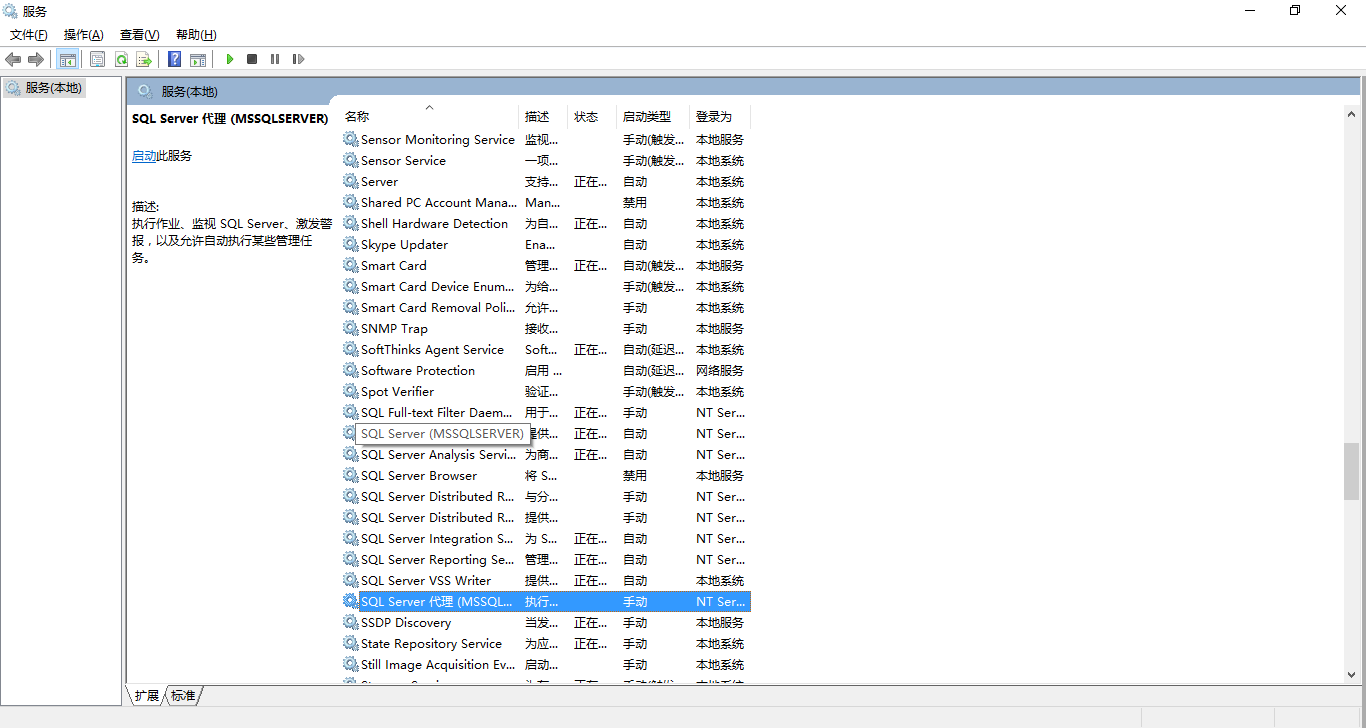
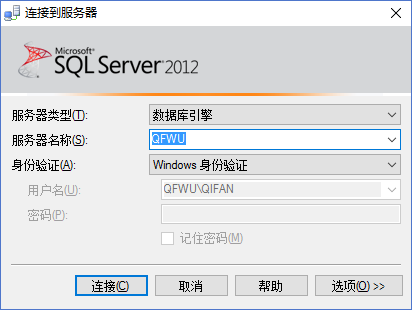


图2.11 SQL SERVER的相关服务



SQL Server 2012 Management登录

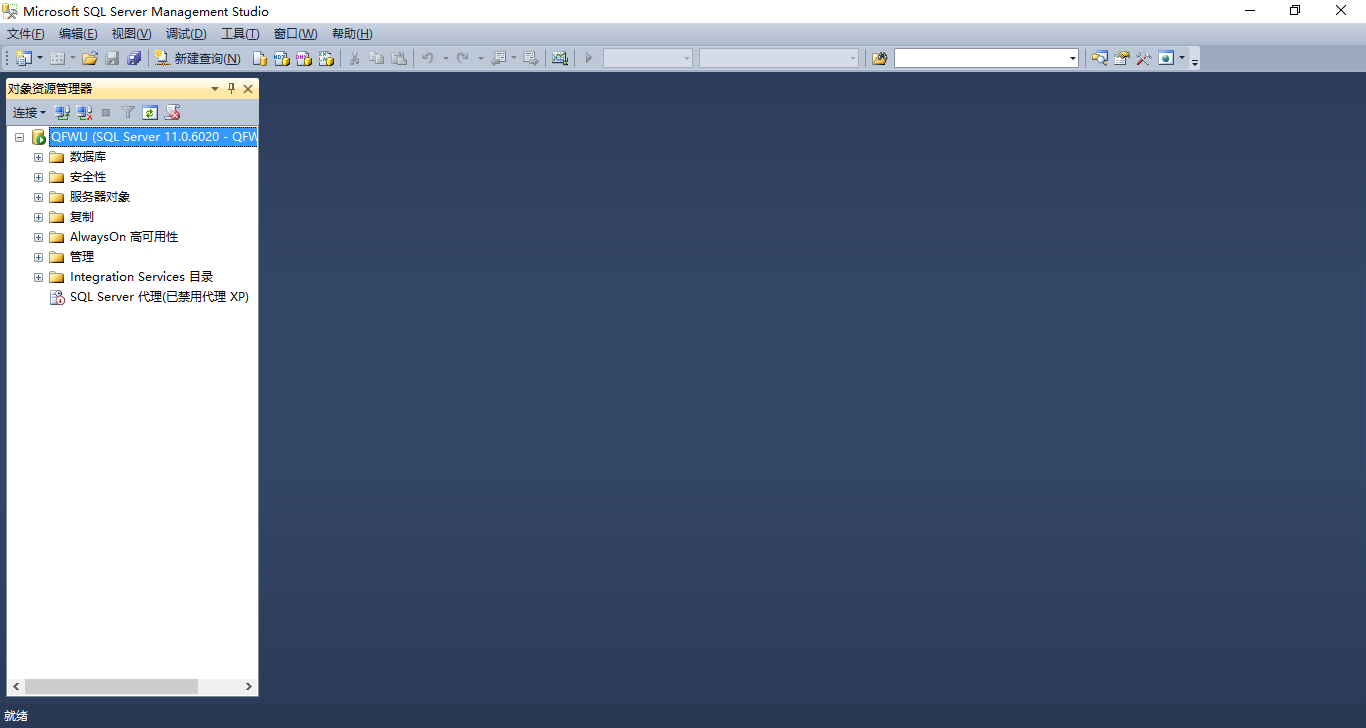


图2.13 SQL Server Management Studio

# E-R建模与关系表转换实验

### 实验目的

1. 理解和掌b握E-R图的基本概念。
2. 培养根据实际应用领域数据对象描述，抽取数据对象特征、关联关系等信息，设计数据库概念结构的能力。
3. 选做：学习Power Designer或其它工具，进行数据模型转换和关系表的自动创建，培养软件辅助设计工具的使用能力。

### 实验原理

根据需求自动生成程序是现代软件开发的一个趋势，可以提高编程效率和降低软件错误。在数据库系统设计开发中，可以利用多种数据复制工具和直接将E-R图转化成数据库表结构的工具。Power Designer是一种数据库概念设计工具，它支持将数据库概念结构转化为物理结构，然后再转化为SQL脚本，从而在数据库中直接生成表结构。此外还可以使用Microsoft Visio等工具做E-R图。

### 实验环境

采用Microsoft SQL Server数据库管理系统作为实验平台。

数据库系统概念设计工具建议采用Sybase Power Designer设计工具。

### 实验内容与步骤

1. 根据数据需求描述抽象出E-R图

阅读《TD-LTE网络配置数据库》课程实验背景资料及数据建模-17-v3.doc，根据LTE的基本概念，分析其中的数据需求，将其描述抽象成实体和联系，并确定实体和联系的属性，特别要注意标明其主键和外键等约束关系，最终形成E-R图。

1. 将E-R图输入相关设计工具（ERWin、Power Design、SQL Server自带设计工具或者Visio等）形成概念模型。
2. 使用工具将E-R图转换为数据库物理结构。
3. 使用将物理模型转化为生成数据库中的表和视图的脚本，注意要选择数据库为SQL Server。
4. 执行SQL脚本，生成表和视图。

成功后，查看生成的表和视图的情况。

### 实验要求

要求在实验报告中详细记录整个过程并详细描述遇到的主要问题。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 数据库表/视图的创建与维护实验

### 实验目的

1. 掌握将E-R图转换为数据库逻辑模式（关系表）的方法。
2. 通过进行数据库表的建立操作，熟悉并掌握在Microsoft SQL Server数据库中建立表的方法，理解关系数据库表的结构，巩固SQL标准中关于数据库表的建立语句。
3. 通过对Microsoft SQL Server数据库中建立、维护视图的实验，熟悉Microsoft SQL Server数据库中建立和维护视图的方法，理解和掌握视图的概念。
4. 掌握从Excel表向Microsoft SQL Server关系数据库导入数据的方法，利用实际数据建立TD-LTE网络配置数据库。

### 实验环境

采用Microsoft SQL Server数据库管理系统作为实验平台。

### 实验内容与步骤

1. 阅读《TD-LTE网络配置数据库》课程实验背景资料及数据建模-17-v3.doc。
2. 将其中数据需求转化为E-R图。
3. 将E-R图转化为逻辑模式（确定表结构及其属性，特别要注意标明其主键、候选键、外键等约束关系）。
4. 进一步明确数据类型等，将逻辑模式转化为物理模式。

方法有两种：

* 1. 写好SQL脚本文件（包括多个**create table、create view**等命令），一次性生成这些表和视图，在Microsoft SQL Server中打开SQL脚本文件并执行。
  2. 一条一条SQL语句执行。

因为这是非常基本的操作，所以一定要了解SQL语句，而不能全部使用图形界面功能来完成。

4.5 视图定义：

1. 创建单表上的视图：
2. 在表“小区/基站工参表”上创建“频点为38400的小区的基本信息”的视图tbcell38400,

属性包括（小区ID，基站ID，小区配置的频点编号，物理小区标识，小区所属基站

的经度，小区所属基站的纬度，基站类型）。

b. 在表“MRO测量报告数据表”上创建“服务小区参考信号接收功率大于40”的视

图MROInfo，属性包括（服务小区/主小区ID，干扰小区ID，服务小区参考信号接

收功率RSRP）。

1. 创建多表上的视图：

根据优化小区表、基于MR测量报告的干扰分析表、小区切换统计性能表，创建“小

区类型为优化区的小区的邻小区ID，C2I干扰的均值、切换目标小区ID、切换成功率”

视图OptCellInfo，属性包括（小区ID，小区类型，C2I干扰邻小区ID，C2I干扰的均

值、切换目标小区ID，切换成功率）。

* 1. 几个重要的涉及表、视图的SQL语句
  2. 查看在表或视图中的所有列、在表中找到的该表上的所有索引。
  3. **alter table**此语句用于修改表定义、禁用相关视图或使表能够参与复制服务器复制。
  4. **alter view**此语句用于用修改的版本替换视图定义。
  5. **drop table**此语句用于删除表对象。
  6. **drop view**此语句用于用于删除视图。
  7. 数据导入：

1. 将Excel表向Microsoft SQL Server关系数据库导入的方法

1> 右击所选数据库，选择“任务”--->导入数据，打开sql server导入导出向导；

2> 根据向导选择数据源为EXCEL文件；

4> 修改目标数据源的数据库为要导入的数据库；

5> 修改表名为要导入的表；

6> 编辑映射，修改数据类型（这一步可忽略，可在导入数据之后修改）；

7> 点击下一步直至完成；

8> 导入成功。

1. 将csv文件向Microsoft SQL Server关系数据库导入的方法

1> 右击所选数据库，选择“任务”--->导入数据，打开sql server导入导出向导；

2> 根据向导选择数据源为平面文件源；

3> 选好要导入的csv文件后，勾选“在第一个数据行中显示列名称”；

4> 修改目标数据源的数据库为要导入的数据库；

5> 修改表名为要导入的表；

6> 编辑映射，修改数据类型；

7> 在“查看数据类型映射”页面中，将“出错时”那一列由“使用全局”改为“忽略”；

8> 点击下一步直至完成；

9> 导入成功。

数据导入成功后，可以在Microsoft SQL Server Management Studio（或者Microsoft SQL Server Management Studio）中查看表中的数据。

### 实验要求

本实验内容比较繁多，要求同学一定要进行完全的实验，并做出详尽的记录。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 数据查询与修改实验

### 实验目的

对实验三中建立的GSM数据库关系表和视图进行各种类型的查询操作和修改操作，加深对SQL语言中DML的了解，掌握相关查询语句和数据修改语句的使用方法。

### 实验环境

采用Microsoft SQL Server数据库管理系统作为实验平台。

### 实验内容

1. 简单的查询操作，包括单表的查询、选择条件、结果排序等的练习；
2. 复杂的查询操作，包括等值连接、自然连接等；
3. 统计查询操作，包括带有分组、集函数的查询操作；
4. ~~嵌套查询操作，包括带有~~**~~in、exists、not exists、~~**~~集合操作的嵌套查询；~~
5. 练习对关系表的其他操作如插入、删除、更新；
6. 练习视图查询、视图修改等视图操作。

### 实验步骤

#### 5.1 Basic structure of SQL Queries

5.1.1 Query on a A Single Relation---The select, where Clause

根据路测ATU数据表，使用distinct语句列出服务小区频点为38400的所有去重后的服务小区ID。

5.1.2 Query on multiple relations——The from Clause

根据路测ATU C2I干扰矩阵表和路测ATU切换统计矩阵表，查询主小区ID为“238397-1”的小区的同站干扰小区ID和切换目标小区ID。

5.1.3 natural join

使用nature join语句重写1.2中的查询。

#### 5.2 Additional Basic Operations

5.2.1 The Rename Operation

根据基于MR测量报告的干扰分析表，使用as语句查询所有比主小区ID为“124673-0”，邻小区ID为“259772-0”的小区间C2I干扰均值高的主小区ID、邻小区ID。

5.2.2 String Operations & 5.2.3 Attribute Specification

根据小区PCI优化调整结果表，使用like语句查询小区名中包含“B马”的相关信息。

5.2.4 Ordering the Display of Tuples

根据路测ATU切换统计矩阵表，查询各小区的最大切换次数及相应的切换目标小区ID，并按降序排列。

5.2.5 Where Clause Predicates

根据小区/基站工参表和基于MR测量报告的干扰分析表，使用between语句查询经度位于112到113之间、纬度位于33.7到33.9之间的小区的C2I干扰的均值最大的邻小区ID。

#### 5.3 Set Operation

5.3.1 根据小区/基站工参表，使用union语句中查询所属城市为yiyang、频点为38544，或所属

城市为sanxia、频点为38400的小区。

5.3.2根据小区一阶邻区关系表和二阶（同频）邻区关系表，使用interset语句查询一阶邻区和二

阶邻区相同的小区。

5.3.3根据一阶邻区关系表和二阶（同频）邻区关系表，使用except语句查询二阶邻区不是一阶邻区的小区。

5.3.4 根据路测ATU C2I干扰矩阵表，使用except语句查询主小区和邻小区间干扰强度最大的小区。

#### 5.4 Null Values

根据路测ATU数据表，查询第1邻小区/干扰小区物理小区标识不为空的服务小区ID、服务小区PCI。

#### 5.5 Aggregate Functions

5.5.1 根据优化小区/保护带小区表和小区一阶邻区关系表，查询一阶邻区数大于10的优化小

区，并将查询结果降序排列。

5.5.2 根据小区/基站工参表和路测ATU数据表，查询所属基站为“253903”的小区的最大信噪比SINR，最小信噪比SINR，平均信噪比SINR。

#### 5.6 Nested (嵌套) Subqueries

5.6.1 Set Membership

根据优化小区/保护带小区表和小区PCI优化调整结果表，查询小区类型为“优化区”的小区经调整后的PCI。

5.6.2 Set Comparison – “some” Clause

1. 根据路测ATU数据表和小区/基站工参表，使用some语句查询“服务小区参考信号接收功率RSRP”大于部分（至少一个）所属基站ID为5660的小区的“服务小区参考信号接收功率RSRP”的服务小区。

1. 根据路测ATU切换统计矩阵表和MRO测量报告数据表，使用all语句查询切换次数最多 的小区的干扰小区ID，干扰小区PCI。

5.6.3 Test for Empty Relations---Use of “not exists” Clause

根据路测ATU数据表和优化小区表，使用not exists语句查询小区类型不为保护带小区的第1邻小区/干扰小区的标识、第1邻小区/干扰小区频点、第1邻小区/干扰小区物理小区标识、第1邻小区/干扰小区参考信号接收强度。

5.6.4 Test for Absence of Duplicate Tuples

根据基于MR测量报告的干扰分析表和路测ATU切换统计矩阵表，查询主小区ID在路测ATU切换统计矩阵表中只出现过一次的加权C2I干扰。

5.6.5 Subqueries in the From Clause

根据路测ATU数据表，查询服务小区参考信号接收功率RSRP的均值大于-70的小区。

5.6.6 With Clause

根据路测ATU切换统计矩阵表和MRO测量报告数据表，使用with语句找出所有具有最低切换次数的小区的MRO测量信息。

5.6.7 Scalar Subquery——Subqueries in the Select Clause

根据小区/基站工参表和一阶邻区关系表，列出频点为38400的所有小区的一阶邻区数目。

#### 5.7 综合---查询语句

根据小区/基站工参表和小区切换统计性能表，查询具有最多二阶邻区数的小区的最大切换成功次数、相应的切换目标小区ID、尝试切换次数。

#### 5.8 Modification of the Database

5.8.1 Deletion

根据路测ATU切换统计矩阵表和小区切换统计性能表，删除切换次数均值小于3的小区切换性能统计数据。

5.8.2 Insertion

向小区/基站工参表中插入一条新信息。

5.8.3 Updates

将优化小区/保护带小区表中，小区ID为“246506-3”的小区的小区类型改为“优化区”。

用小区PCI优化调整结果表中“优化调整后的本小区PCI值”，替换小区/基站工参表中小区的“物理小区标识”。

针对路测ATU C2I干扰矩阵表表，使用case语句作出如下修改：如果主小区与干扰小区为同站小区且干扰强度排序不小于1，则干扰强度排序减1；如果主小区与干扰小区不为同站，干扰强度排序加1。

### 实验要求

1. 用Transact\_SQL语句完成以上操作。
2. 要求学生独立完成以上内容。
3. 实验完成后完成要求的实验报告内容。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 第六章 数据库完整性与安全性实验

### 实验目的

1. 通过对完整性规则的定义实现，熟悉了解SQL Server数据库中实体完整性、参照完整性、断言等完整性保证的规则和实现方法，加深对数据完整性的理解。
2. 通过对安全性相关内容的定义，熟悉了解SQL Server数据库中安全性的内容和实现方法，加深对数据库安全性的理解。

### 实验环境

采用SQL Server数据库管理系统作为实验平台。SQL Server要求选用要求使用2005、2008或者2012版本，可以采用SQL Server Express、SQL Server Develop或SQL Server Enterprise等版本。

### 实验内容

#### 6.1 完整性实验与要求

* 1. 添加语义约束

在前面完成的实验2、3中已建立了本实验所需的11张表。本实验将针对这11张表，采用alter table语句，添加主键、候选键、外键、check约束、缺省/默认值约束，并观察当用户对数据库进行增、删、改操作时，DBMS如何维护完整性约束。

建表阶段，在create table语句中添加各类完整性约束的内容请自行练习。

（1）新建主键，找到未定义主键的表，利用下面的语句建立主键:

alter table 表名

add constraint PK\_字段名--"PK"为主键的缩写，字段名为要在其上创建主键的字段名,'PK\_字段名'就为约束名

primary key (字段名) --字段名同上

（2）候选键，找到有非主键的其它候选键的表，利用下面的语句建立候选键:

alter table 表名

add constraint UQ\_字段名

unique (字段名)

（3）外键约束，找到拥有主外键关系的表，利用下面的语句新建外键：

alter table 表名

add constraint FK\_字段名--"FK"为外键的缩写

foreign key (字段名) references 关联的表名(关联的字段名) --注意'关联的表名'和'关联的字段名'。

（4）check约束，利用下面的语句给相关表建立约束：

altertable表名

add constraint CK\_字段名check(表达式)

（5）缺省/默认值约束

alter table 表名 add [contraint](https://www.baidu.com/s?wd=CONSTRAINT&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YzPjbvuycsuHf3nW9BnyD10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f3PWc3rjcd) DF\_字段名 [default](https://www.baidu.com/s?wd=DEFAULT&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YzPjbvuycsuHf3nW9BnyD10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f3PWc3rjcd) 默认值for列名

注意：

根据表中数据的情况，在进行本实验的时候，可能会有冲突并且造成数据变化，为了保证表中原始数据的正确性，可以先对原表内容进行备份。或者干脆新建一个数据库进行相关操作。对于中间出现的冲突等情况应自行选择适当方法解决。

6.1.2 主键/候选键约束验证

6.1.2.1 选取定义了主键的关系表，如tbCell、tbAdjCell、tbOptCell等，向该表插入在主属性上取值为空的元组，观察DBMS反应；

选取表中某些或某个元组，修改这些元组在主属性上的取值，或向表中插入新元组，使这些元组与表中已有其它元组的主属性取值相同，或者将选定的元组在主属性上的取值修改为null，观察系统反应。

6.1.2.2 选取定义了候选键的关系表，向该表插入在候选键属性上取值为空的元组，观察DBMS的反应；

选取表中某些或某个元组，修改这些元组在候选键属性上的取值，或插入新元组，使这些元组与表中已有其它元组的候选键属性取值相同，或者将选定的元组在候选键属性上的取值修改为null，观察系统反应；

（1）将SSECTOR\_ID，NSECTOR\_ID设置为tbATUHandOver的候选键，并修改某一个元组的该属性值为空。

改值:将tbATUHandOver 表中SSECTOR\_ID 为 '15113-129'的记录中的SSECTOR\_ID修改为NULL。

**比较在主键、候选键属性上插入null值或重复值时，DBMS的不同反应和处理方式。**

* + 1. 外键约束验证

6.1.3.1 选取相互间定义了外键关联的一组表，分别在参照关系、被参照关系上，对表的主属性/外键属性作插入、删除、更新等操作，观察当其中1个表（如参照关系表、被参照关系表）在外键属性或主属性上的取值发生变化时，数据库管理系统DBMS对这些操作的反应，以及另外一个表（如被参照关系表、参照关系表）在主属性或外键属性上的取值的变化，并记录实验结果。

上述插入、删除、更新操作操作分为违反约束和不违反约束两种情况。

下面以删除为例：delete from tbOptCell



参照关系-被参照关系对可以从以下关系队中选取：

tbCell, tbOptCell;

tbCell, tbAdjCell;

tbOptCell, tbPCIAssignment;

tbCell, tbHandover;

tbCell, tbC2I;

注：实验时，从中选取一对即可。

* + - 1. 观察级联操作对外键约束的影响

1）对相互间定义了外键关联关系的一组表，分别使用

**foreign key－references**

**on delete cascade**

**on update cascade**

语句创建级联外键约束(先删除之前创建的外键，使用Alter table 中的Drop constraint参数)。

然后重新创建级联外键（使用alter table 中的add constraint …

foreignkey … references … on delete/update/insert …参数）

2) 分别在参照关系、被参照关系上，对表的主属性/外键属性作插入、删除、更新等操作，观察当1个表（如参照关系表、被参照关系表）在外键或主属性属性上的取值发生变化时，数据库管理系统对这些操作的反应，以及另外一个表（如被参照关系表、参照关系表）在主属性或外键属性上的取值的变化，并记录实验结果。

上述插入、删除、更新操作操作分为违反约束和不违反约束两种情况。

6.1.4 缺省/默认值约束实验

以tbAdjCell为对象，针对S\_EARFCN、N\_EARFCN，建立默认/缺省值约束——将这2个频点的默认值设置为38400，然后向表中插入数据。

1. 在现有表中添加缺省/默认值约束
2. 插入一个新元组（缺少频点）
3. 检查新元组的频点是否为默认值

#### 6.2 安全性实验内容与要求

6.2.1 SQL Server 数据库实验：

SQL Server提供了多种安全机制。主要的有如下3条：

1．最小权限原则：

SQL Server通过给不同用户赋予不同权限的方式来保证安全。本原则要求只给某用户完成工作所必须的权限，以尽量减小用户有意或无意的错误对数据库造成的损害。

2．CIA原则：

C：机密性，未授权者或使用非法手段无法访问数据。

I：完整性，未授权者或使用非法手段无法修改数据。

A：可用性，已授权者可随时使用数据。

3．深度防护：

实际上任何一个程序都运行在操作系统上，一台计算机上还有其它应用程序。所以SQL Server的安全不能仅仅考虑本身的安全，而必须结合操作系统安全及其它相关因素（比如杀毒软件和防火墙）来考虑。这就是深度防护。SQL Server仅仅运行在Windows操作系统上，允许直接使用Windows用户作为SQL Server用户，由于是同一公司的产品，其和操作系统的联系和配合特别紧密，为其它数据库所远远不及的。

在SQL Server服务器及其它Windows服务器上，应该进行多方面的安全配置：包括操作系统补丁管理、管理员用户和组管理、其它本地用户组管理和设置本地安全策略。为了提供网络上的客户端访问，还需要进行端点等相关设置。

对于整个SQL Server的安全机制，我们先进行名词解释：

**服务器登录名：指有权限登录到某服务器的用户；**

**服务器角色：指一组固定的服务器用户，默认有9组；**

* **登录名一定属于某些角色，默认为public**
* **服务器角色不容许更改**
* **登录后也不一定有权限操作数据库**

**数据库用户：指有权限能操作数据库的用户；**

**数据库角色：指一组固定的有某些权限的数据库角色；**

**数据库架构：指数据库对象的容器；**

* **数据库用户对应于服务器登录名以便登录者可以操作数据库**
* **数据库角色可以添加，可以定制不同权限**
* **数据库架构，类似于数据库对象的命名空间，用户通过架构访问数据库对象**

**服务器角色**

sysadmin

--在 SQL Server 中进行任何活动。该角色的权限跨越所有其它固定服务器角色。

serveradmin

--配置服务器范围的设置。

setupadmin

--添加和删除链接服务器，并执行某些系统存储过程（如 sp\_serveroption）。

securityadmin

--管理服务器登录。

processadmin

--管理在 SQL Server 实例中运行的进程。

dbcreator

--创建和改变数据库。

diskadmin

--管理磁盘文件。

bulkadmin

--执行 BULK INSERT 语句。

**数据库角色**

public  
--public 角色是一个特殊的数据库角色，每个数据库用户都属于它。public 角色：   
--捕获数据库中用户的所有默认权限。  
--无法将用户、组或角色指派给它，因为默认情况下它们即属于该角色。  
--含在每个数据库中，包括 master、msdb、tempdb、model 和所有用户数据库。  
--无法除去。

db\_owner   
--进行所有数据库角色的活动，以及数据库中的其它维护和配置活动。  
--该角色的权限跨越所有其它固定数据库角色。

db\_accessadmin   
--在数据库中添加或删除 Windows NT 4.0 或 Windows 2000 组和用户以及 SQL Server 用户。

db\_datareader   
--查看来自数据库中所有用户表的全部数据。

db\_datawriter   
--添加、更改或删除来自数据库中所有用户表的数据

db\_ddladmin   
--添加、修改或除去数据库中的对象(运行所有 DDL)

db\_securityadmin   
--管理 SQL Server 2000 数据库角色的角色和成员，并管理数据库中的语句和对象权限

db\_backupoperator   
--有备份数据库的权限

db\_denydatareader   
--拒绝选择数据库数据的权限

db\_denydatawriter  
--拒绝更改数据库数据的权限

然后来理解这些概念之间的关系：

sqlserver里面的**[数据库](http://lib.csdn.net/base/14" \t "_blank" \o "MySQL知识库)**级别设置：

服务器级 -> 数据库级 -> [**架构**](http://lib.csdn.net/base/16)级 - > 数据对象级，比如说：Server.DataBase1.dbo.Table1;这里的意思就是Table1这个表属于dbo这个架构，dbo这个架构属于DataBase1这个数据库，DataBase1这个数据库属于Server这个服务器。里面的架构其实就是一个容器，好像就是面向对象里面的命名空间，一个用户可以拥有多个架构，但是不能对没有拥有的架构进行操作。

角色，角色意味着一种身份，一个角色通常拥有一系列权限。一个数据库角色，可能包含对不同架构里面数据对象的一系列权限，就是有可能涉及到多个架构。

**服务器登录名**，指有权限登录到某服务器的用户，可以在有权限的情况下创建新的登录名，超级管理员的登录名是sa。

**服务器角色**，服务器的身份，拥有一组服务器权限，可以将一些登录名授予该角色；另外登录名一定属于某个或某些角色，默认为public。 我们登录后也不一定有权限操作数据库。

**数据库用户**，指有权限能操作数据库的用户；

* **数据库角色**，数据库上的某种身份，有一组数据库权限；数据库角色可以添加，可以定制不同权限。

**数据库架构**，指数据库对象（表）的容器；类似于数据库对象的命名空间，用户通过架构访问数据库对象。

我们用服务器登录名登录后，由于其与一个或多个数据库用户有对应关系，则登录者可以以该对应数据库用户的权限操作相关的一个或者多个数据库。登录名与用户在服务器级是一对多的，而在数据库里是一对一的。比如说Server这个服务器上有4个数据库，DB1，DB2，DB3，DB4，每个数据库都有一个用户USER1，USER2，USER3，USER（不同数据库上可以有同名用户），再创建一个登录名my的时候可以通过用户映射的操作，为这个登录名在每一个具体的数据库中指定用户，比如可以如下指定my在DB1中的用户是USER1，在DB2上就是User2,而在DB3上又是USER1.则my在Db1上就是User1，而不能变化为User2，除非重新指定它对DB1数据库的用户映射为user2。

总结为：

**1.一个数据库用户可以对应多个架构（架构是表容器）。架构里面包含的是数据库表。**

**2.一个数据库角色有可能涉及多个架构。数据库角色对应的是一组权限。**

**3.一个用户对应一个数据库角色。**

**4.登录名与数据库用户在服务器级别是一对多的；在数据库级别是一对一的。**

**5.服务器登录最后会映射为某个数据库用户。**

**6.一个服务器角色会被授予某些服务器登录。**

现在有3种登录名：windows域登录名（就是域用户），windows本地登录名（本地用户）和SQL Server登录名。用户登录以后，SQL Server将其映射为自己的user，进行相应安全管理。

从而提供了两种身份认证模式：Windows认证模式和SQL Server认证模式。

1．Windows认证模式

SQL Server数据库系统通常运行在NT服务器平台或基于NT 构架的Windows server上，而NT作为网络操作系统，本身就具备管理登录、验证用户合法性的能力，所以Windows认证模式正是利用这一用户安全性和账号管理的机制，允许SQL Server也可以使用NT的用户名和口令。在该模式下，用户只要通过Windows的认证就可连接到SQL Server，而SQL Server本身也不需要管理一套登录数据。

2．SQL Server认证模式

在SQL Server认证模式下，用户在连接到SQL Server时必须提供建立在SQL Server上的用户名（登录名）和登录密码，这些登录信息存储在系统表syslogins中，与NT的登录账号无关。SQL Server自己执行认证处理，如果输入的登录信息与系统表syslogins中的某条记录相匹配则表明登录成功。注：该方式常用于系统开发中，因为客户机常常与服务器不是同一台计算机，甚至也不在同一个windows域中，所以使用该方式进行登录比较方便。

实验内容：

1. 以DBA身份（可以是SQL Server上的sa或者windows上的系统管理组的某个成员）登陆系统，在图形界面下创建新登录，在安全性选择登录，然后新建登录用户，可以使用Windows认证模式和SQL Server认证模式。选择Windows认证登录则必须添加已经存在的Windows本机或者域上的用户或者用户组。通常选择搜索，然后选择高级和立即查找，找到实际存在的Windows用户或者组。最后可以选择默认数据库为当前使用的数据库，而不是默认的系统数据库。
2. 使用T-SQL命令create login login-name …和create login login-name from windows重复第1步的内容。

-- Syntax for SQL Server

CREATE LOGIN login\_name { WITH <option\_list1> | FROM <sources> }

<option\_list1> ::=

PASSWORD = { 'password' | hashed\_password HASHED } [ MUST\_CHANGE ]

[ , <option\_list2> [ ,... ] ]

<option\_list2x> ::=

SID = sid

| DEFAULT\_DATABASE = database

| DEFAULT\_LANGUAGE = language

| CHECK\_EXPIRATION = { ON | OFF}

| CHECK\_POLICY = { ON | OFF}

| CREDENTIAL = credential\_name

<sources> ::=

WINDOWS [ WITH <windows\_options>[ ,... ] ]

| CERTIFICATE certname

| ASYMMETRIC KEY asym\_key\_name

<windows\_options> ::=

DEFAULT\_DATABASE = database

| DEFAULT\_LANGUAGE = language

可查阅<https://msdn.microsoft.com/zh-cn/library/ms189751.aspx>。下同，不再一一引述。MSDN上关于T-SQL的地址在：<https://msdn.microsoft.com/zh-cn/library/bb510741.aspx>。

1. 理解服务器角色的概念。在服务器角色选项卡中赋予新建用户角色。完成后查找T-SQL中对应命令，在图形界面上撤消该权限后使用命令完成授予权限和收回权限的任务。
   1. 首先查看现在存在的服务器角色，任选服务器角色下的一个实际角色，查看其属性。
   2. 选择登录打开，将刚刚新建的Login添加上去并且删除掉。
   3. 增删新建服务器角色的权限。
2. 对特定服务器对象设置权限，双击登录名以后选择安全对象，赋予该登录名对服务器内的某个对象的权限。执行相关SQL操作，检查该用户的权限。用户在权限范围内、超出权限范围访问相应数据对象，查看访问结果和DBMS的反映。
3. 理解数据库角色的概念。
   1. 新建一个数据库角色。在移动通信数据库中，选择安全性，角色，数据库角色，新建。
   2. 授予访问移动通信数据库的读、写权限，并对其中的Cell表数据进行修改。
   3. 新建一个数据库用户。
   4. 将某登录映射到移动通信数据库某用户。
4. 根据实际操作，指出服务器角色和数据库角色有多少种，分别有什么权限？
5. 创建数据库时，还会自动创建SYS、GUEST 和dbo组。通过帮助文档，了解这些组在数据库中起到的作用。
6. 使用GRANT语句设置对某表的SELECT和INSERT权限，并进行相关数据访问或者修改操作。验证权限已经设置成功。执行超出权限范围访问相应数据对象的操作，查看访问结果和DBMS的反映。
7. 分别在图形化界面、命令行中，撤销用户权限，命令行中使用REVOKE语句。验证原来能正常进行的操作现在是否仍能正常进行。
8. 分别在图形化界面、命令行中，删除某数据库用户。查看有什么结果。
9. 分别采用Windows认证方式和SQL Server认证方式用不同的登录进行登录连接，重做上述内容，比较两种登录有不同吗？

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 第七章 数据库访问接口实验

### 实验目的

1. 通过编写数据库应用程序，培养数据库应用程序开发能力。
2. 熟悉数据库应用程序设计的多种接口的配置，培养相关的软件配置能力。

### 实验原理

数据库应用程序设计是数据库应用的一个重要方面。SQL语言除了以用户交互的方式使用外，还可以被数据库应用程序直接访问。

通常DBMS支持的能直接访问数据库的是SQL语言，而SQL语言不像高级语言（例如C、C++等）具备很好的数据处理能力。通常的情况是需要将两种语言结合起来，利用SQL访问数据，而将数据传递给某种高级语言程序，处理后又利用SQL写回数据库。

这种嵌在高级语言程序中的SQL语句称为嵌入式SQL（或者称为ESQL），和以前使用的交互式SQL（或者称为ISQL）不一样，它们随着程序执行被调用，辅助程序完成数据库数据读写的功能，而高级语言程序则负责对数据库中的数据的分析处理转换等操作。

ESQL又分成两种，如果在程序执行前SQL的结构就已经确定，最多是可以在执行时传递一些数值参数，那么这种ESQL语句称为静态ESQL。如果程序执行时才知道执行的SQL语句时，称为动态ESQL。

SQL SERVER中执行静态ESQL（比如SQLJ）时，首先利用预编译将宿主程序语言中的SQL语句分离出来，代之以过程或函数调用，然后对剩余程序正常编译和连接库函数等。分离出来的SQL语句则在数据库端进行处理，进行语法检查、安全性检查和优化执行策略，绑定在数据库上形成包（packet）供应用程序调用。

而通过ODBC、JDBC、OLEDB等访问数据库都是动态ESQL。SQL SERVER执行它们时因为无法事先确实知道是什么样的SQL语句，从而无法进行静态绑定，而这种绑定过程只能在程序执行过程中生成了确定的要执行的SQL语句时才能进行，称为动态绑定。

应用程序的接口则有JDBC、ADO、OLEDB、ODBC等多种。微软在SQL SERVER 2005开始还增加了几项功能：一是service broker，围绕发送和接收消息而设计，处理通信中的消息收发工作，使开发人员集中精力于解决实际问题。二是CLR（Common Language Runtime）组件，就是Microsoft.NET Framework中的公共语言运行库组件，使开发人员可用任何CLR语言来编写过程、函数、触发器等，甚至自定义数据类型。三是对T-SQL做出了许多扩展。

### 实验环境

采用SQL SERVER数据库管理系统作为实验平台。其中，SQL SERVER可以采用Express、Develop或者Enterprise版本，选择SQL SERVER 2008、2012、2014或2016。

### 实验内容

1. 了解通用数据库应用编程接口（例如JDBC、ODBC等）的配置方法。
2. 利用C语言(或其它支持某种数据库应用程序接口的高级程序设计语言)编程实现简单的数据库应用程序，掌握基于ODBC、JDBC接口的数据库访问的基本原理和方法。
3. 掌握获取、修改数据库连接时长的方法。
4. 通过ODBC、JDBC接口，在数据库应用程序中，执行查找、增加、删除、更新等操作。

### 实验要求

1. **基于JDBC接口或基于ODBC接口的数据库访问实验，二选一，完成一个即可；**
2. **实验时选取TD-LTE数据库作为数据源，可参照后文的例子（以university为数据源）；**
3. **需要完成的操作**
4. **连接时长的获取和修改**

TD-LTE数据库中tbAdjCell、tbMRODat、tbATUData等表中的数据很大，对这些表进行查询（尤其是多表查询）、更新等操作时会花费较多的时间。

为了避免连接中途断掉，可用相应ODBC/JDBC的API函数查询和修改连接时长（即连接数据库成功后的语句执行时间）。

实验时，要求针对tbC2I、tbAdjCell、tbMRODat、tbATUData等表，设计执行时间较长的访问操作，观察不同连接时间设置对访问执行过程的影响。具体**要求如下**：

**1）**通过API函数获取数据库默认连接时间；

2)在默认连接时间下，完成“4.**（建议的）对TD-LTE配置数据库的访问操作**”中给出的执行时间较长的查询（三选一），观察是否会超时；

3) 增大、降低默认连接时间，观察是否超时。

不要求返回完整的数据，只需观察查询是否成功即可。

1. **查询**

选取一张表或几张表执行查询操作，并打印出数据（几行数据即可）。

1. **插入**

选取一张表执行插入操作，插入成功后打印出新插入的数据。

1. **更新**

选取一张表执行更新操作，更新成功后打印出更新后的数据。

1. **删除**

选取一张表执行删除操作，并检查操作是否成功。

1. **（建议的）对TD-LTE配置数据库的访问操作**

以下3个操作针对tbAdjCell、tbMRODat、tbATUData等表，执行时间（可能）较长，可用于实验内容“连接时长获取和修改”。也可用于后续的增删改查实验内容(2), (3), (4), (5)。

* 查询1：（对tbAdjCell的连接访问）**根据tbAdjCell表给出的小区1阶邻区关系，计算小区间的二阶邻区表tbSecAdjCell**。

原理：如果<a,b>∈tbAdjCell, <a,c>∈tbAdjCell，即b为a的1阶邻区，c为a的1阶邻区，则b与c间存在2阶邻区关系。

方法：

Select T1.N\_Sector\_ID, T2.N\_Sector\_ID into tbNewSecAdjCell //保原表tbSecAdjCell内容

From tbAdjCell as T1, tbAdjCell as T2

Where T1.S\_Sector\_ID = T2.S\_Sector\_ID

* 查询2：对tbMROData的update操作

针对tbMROdata表，如果邻小区的频点为38400（即LteNcEarfcn=38400），并且邻小区的PCI范围在0到300之间（即LteNcPci between 0 and 300），则将主小区的参考信号接收信号强度LteScRSRP的值增加1.

* 查询3：对tbATUData的delete操作

从tbATUData表中，删除同时满足下述条件的元组：

time=16:23,

Latitude between 33.75314 and 33.75316,Logitude between 112.82840 and 112.82842,

PCI=166, EARFCN=38400

### 实验步骤

1. 实验准备：

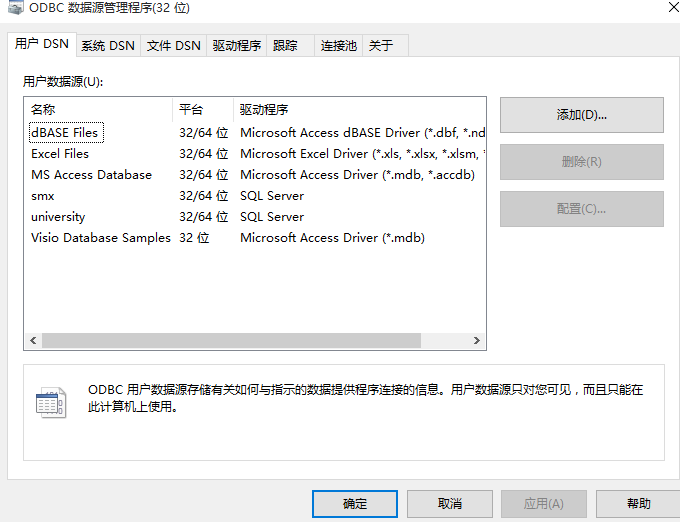
以教科书第四章关于SQL语言相关内容为基础，课后查阅、自学ODBC/JDBC等接口有关内容，包括体系结构、工作原理、数据访问过程、主要API接口的语法和使用方法等。

1. 在Windows控制面板中通过管理工具下的ODBC数据源工具在客户端新建连接到SQL SERVER数据库服务器的ODBC数据源，测试连接通过后保存，注意名字要和应用程序中引用的数据源一样。**另外要注意创建ODBC数据源时使用的驱动程序必须是支持SQL SERVER的，而不可选择其它驱动程序。**
2. 以实验二建立的数据库为基础，编写 C语言(或其它支持ODBC/JDBC等接口的高级程序设计语言) 数据库应用程序，利用SQLExecDirect语句，实现数据库应用程序对数据库中表（有数据）进行数据查询、删除、插入、更新等操作。**注意：SQL SERVER应设置为应用程序可以访问。**

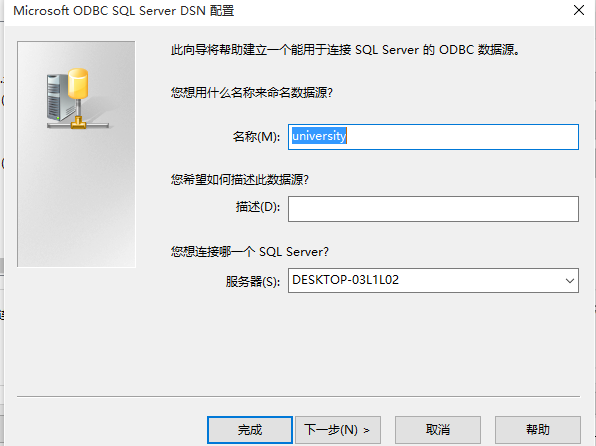
#### 7.1 ODBC访问

7.1.1环境配置

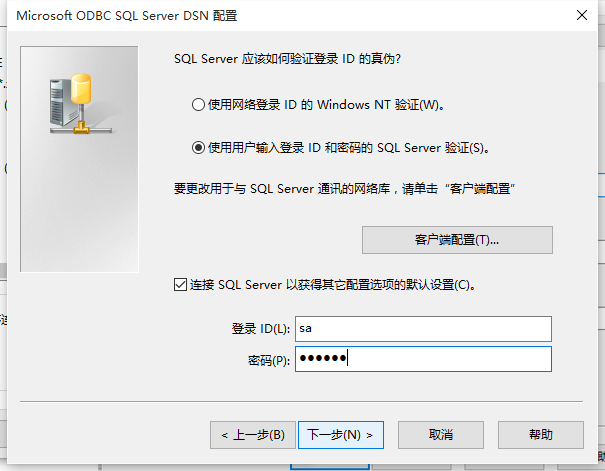
7.1.1.1 打开ODBC数据源后点击“添加”，添加一个名为“university”的数据源；



7.1.1.2 设置数据源名称，服务器选择本机名称；



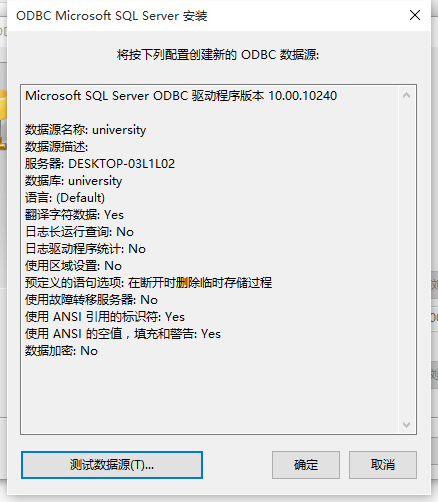
7.1.1.3 设置数据库的用户名和密码（在此之前确保SQLServer的登录方式为SQLServer身份验证登录）；



7.1.1.4 更改默认数据库为要进行操作的数据库（在此之前SQLServer里已建立好university数据库，并将相应的表导入其中）；



7.1.1.5 测试数据源，如果测试成功，则表示数据源设置成功。



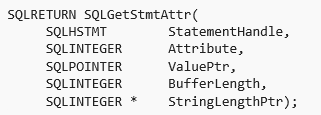
7.1.2 连接时长的获取和修改

可用相应的函数获取和修改连接时长（连接数据库成功后的语句执行时间），例如1.3部分所示代码的默认连接时长为表示无限制，修改连接时间为；

微软为ODBC提供了完整的技术文档，在实验时可按需查阅：

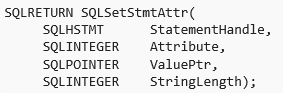
<https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/sqlgetstmtattr-function>

* **获取语句属性函数SQLGetStmtAttr**



参数分别表示（语句句柄，返回的属性名，指向存储返回属性值的内存，ValuePtr的长度，ValuePtr指向的数据的长度）。

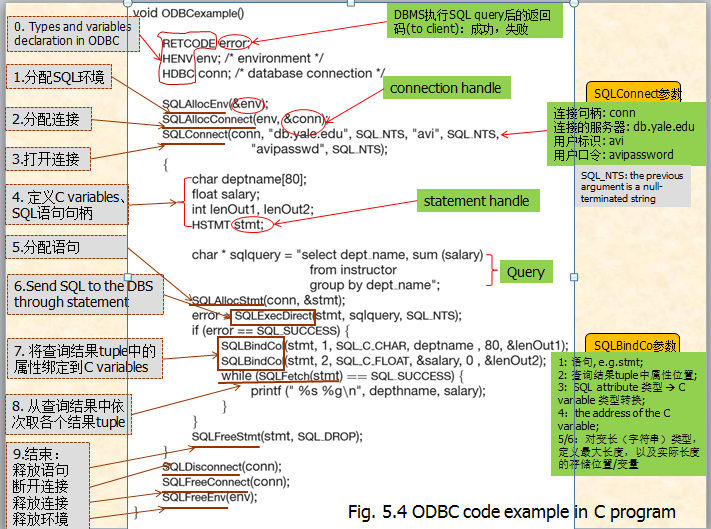
* **设置语句属性函数 SQLSetStmtAttr**



参数分别表示（语句句柄，属性名称，取值与属性相关，取值与Attribute和ValuePtr有关）

7.1.3 代码实现：查询不同部门的总工资并按部门名字进行排序

7.1.3.1 可参考例子：教科书代码实现



* **注意事项**

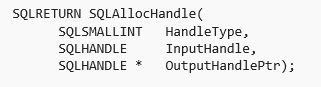
教科书所给例子的函数属于ODBC早期版本，从Windows7开始，内置的ODBC版本都为3.X，根据实验所选的环境和编译器，例子所示程序有可能运行出结果，也有可能报错。

下图为不同版本的函数对比



7.1.3.2 部分函数说明

* **分配函数SQLAllocHandle**



**参数说明:**

HandleType：句柄类型，取值为SQL\_HANDLE\_ENV（环境）、SQL\_HANDLE\_DBC（连接）、SQL\_HANDLE\_STMT（语句）等；

InputHandle：要分配新句柄的上下文中的输入句柄。如果HandleType是SQL\_HANDLE\_ENV，其值为SQL\_NULL\_HANDLE。另外，SQL\_HANDLE\_DBC对应环境句柄，SQL\_HANDLE\_STMT对应连接句柄；

OutputHandlePtr：指向存储当前分配句柄的变量的指针。

* **释放函数SQLFreehandle**

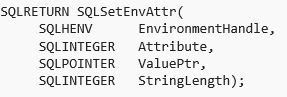


**参数说明:**

HandleType：同上；

Handle：需要释放的句柄。

**设置环境属性函数SQLSetEnvAttr**



参数分别表示（环境句柄，需设置的环境属性名称，前一个属性的取值，ValuePtr指向的数据的长度<如果是字符串则为字符串的长度，整数则忽略.>）。

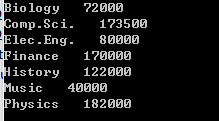
7.1.3.3 书上例子的一个完整实现





程序说明

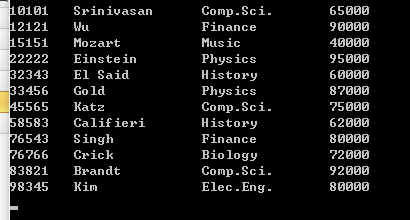
1. 实验环境：OS：Windows10，编译器：vs2015；
2. 在每一步都检查返回值是好习惯，便于程序失败时定位错误点，上述例子在运行时总是连接出错（后来证明是与所选的编译器有关，在创建新项目时需选择“空项目”），目前的检查比较粗略，可用SQLGetDiagRec获得更准确的信息。
3. 运行结果



7.1.4其他操作

（1） 查询表中所有数据





（2） 添加一条‘65432’、‘Bob’、‘Psychology’、‘75000’的数据到数据库

char \*sqlquery = "insert into instructor values('65432','Bob','Psychology',75000)";



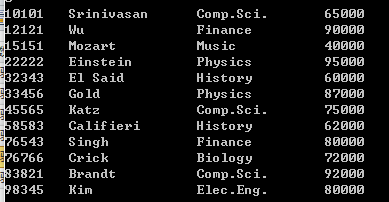
（3）更新，将Bob的salary改为80000

char \*sqlquery = "update instructor set salary='80000' where name='Bob'";



（4）删除，删除Bob的信息

char \*sqlquery = "delete from instructor where name='Bob'";



#### 7.2 JDBC接口实验

7.2.1实验准备

1. 下载jdbc驱动，将下载好的sqljdbc4.jar放到C:\Program Files。
2. **设置 Classpath，**将 classpath 设置为包含 sqljdbc.jar 文件。如果 classpath 缺少 sqljdbc.jar 项，应用程序将引发“找不到类”的常见异常。
3. **设置SQL Server服务器。一般默认情况不需要配置，如需要配置按下列步骤配置:**

(i)“开始”→“程序”→“Microsoft SQL Server 2005”→“配置工具”→“SQL Server 配置管理器”→“SQL Server 2005 网络配置”→“MSSQLSERVER 的协议”

(ii)如果“TCP/IP”没有启用，右键单击选择“启动”。

(iii)双击“TCP/IP”进入属性设置，在“IP 地址”里，可以配置“IPAll”中的“TCP 端口”，默认为1433。

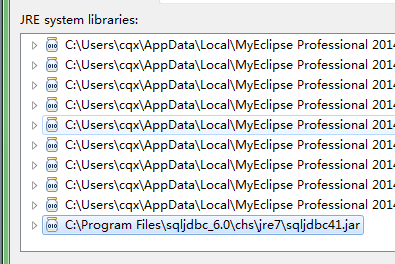
(iv) 重新启动SQL Server或者重启计算机。

7.2.2创建数据库（以书上instructor表为例）

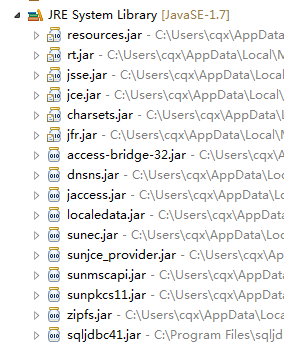
7.2.3在eclipse或者myeclipse中测试

首先创建新的java项目。

选择“window”→“preferences”→“Java”→“installed JREs”，选择已安装的 JRE，点击“Edit”→“Add External JARs”，选择sqljdbc4.jar，要先打开其所在文件夹。



在项目的“JRE 系统库”中可以看见sqljdbc4.jar，如图。如果没有可以右键单击项目→“构建路径”→“配置构建路径...”→“Java 构建路径”→“库”→“添加外部 JAR...”，同样可选择sqljdbc4.jar



7.2.4 设置超时时间

在JDBC中Statement类可以通过getQueryTimeout()获取驱动程序等待Statement 对象执行的秒数。setQueryTimeout()来限制statement的执行时长。即java.sql.Statement.setQueryTimeout(int timeout)，在创建statement实例后直接输入限制的时长即可。

可用相应的函数获取和修改连接时长（连接数据库成功后的语句执行时间），C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\X_CP_BI5)W{HH89AZ0}2}SE.png

默认连接时长为C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\R`I9{%JZ_41GDAHRPFOUSH5.png表示无限制。

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\YTH8_2WYGGLZSA)C%@RI}1E.png

修改连接时间为。

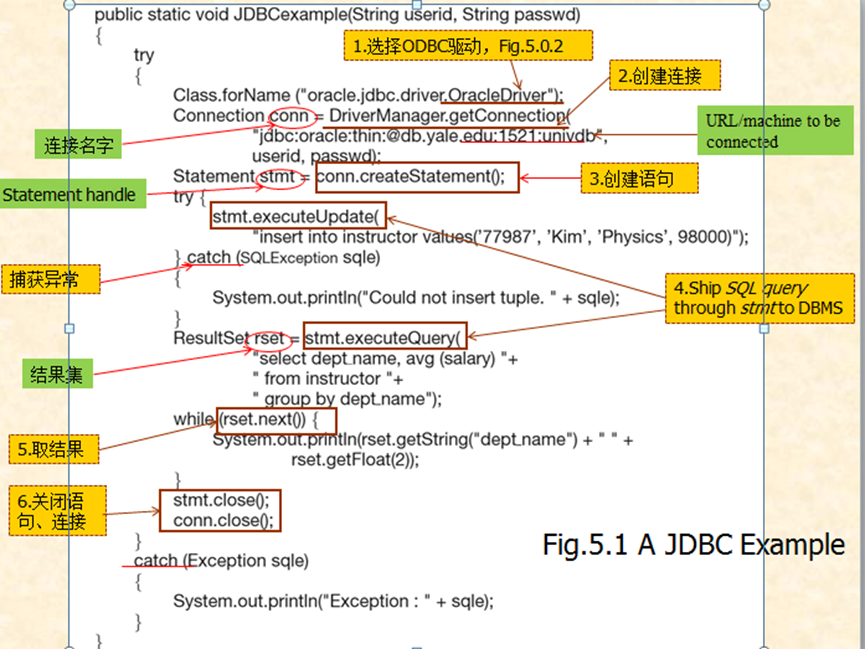
7.2.5编写java代码

注意sql server的版本不同，连接语句也不同，以sqlserver2008为例，加载驱动和URL路径的语句分别为

String driverName = "com.microsoft.sqlserver.jdbc.SQLServerDriver";

String dbURL = "jdbc:sqlserver://localhost:1433; DatabaseName=inst\_dept";

代码逻辑与所用部分函数说明如下图：



全部代码为：

**import java.sql.\*;**

**public class API {**

**public static void main(String[] args) {**

**// TODO Auto-generated method stub**

**String driverName = "com.microsoft.sqlserver.jdbc.SQLServerDriver";**

**String dbURL = "jdbc:sqlserver://localhost:1433; DatabaseName=inst\_dept";**

**String userid = "sa"; //默认用户名**

**String passwd = "123"; //密码**

**try**

**{**

**Class.forName(driverName);**

**Connection conn=DriverManager.getConnection(dbURL,userid,passwd);**

**Statement stmt=conn.createStatement();**

**try**

**{**

**stmt.executeUpdate("insert into instructor values('65432','Bob', 'Psychology'，'75000')");**

**}catch(SQLException sqle)**

**{**

**System.out.println("Could not insert tuple."+sqle);**

**}**

**ResultSet rset=stmt.executeQuery("select dept\_name,avg(salary) from instructor group by dept\_name ");**

**while(rset.next())**

**{**

**System.out.println(rset.getString("dept\_name")+" "+rset.getFloat(2));**

**}**

**stmt.close();**

**conn.close();**

**}**

**catch(Exception sqle)**

**{**

**System.out.println("Exception:"+sqle);**

**}**

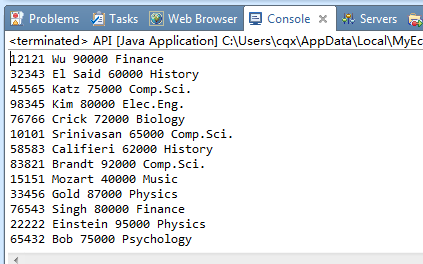
**}**

**}**

7.2.6对数据库进行操作(通过jdbc接口操作)

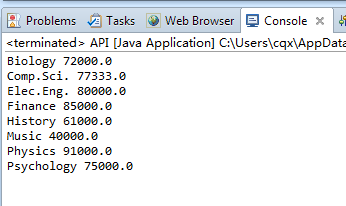
（1）添加一条‘65432’、‘Bob’、‘Psychology’、‘75000’的数据到数据库。并打印整张表。

insert into instructor values('65432','Bob', 'Psychology'，'75000'

****

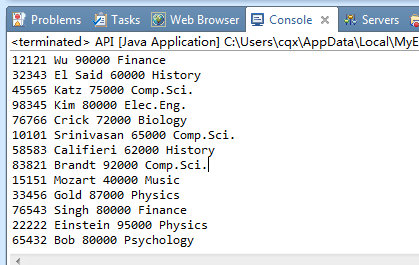
1. 查询，查每栋公寓的人的平均收入，并按公寓名排序。

select dept\_name,avg(salary) from instructor group by dept\_name

****

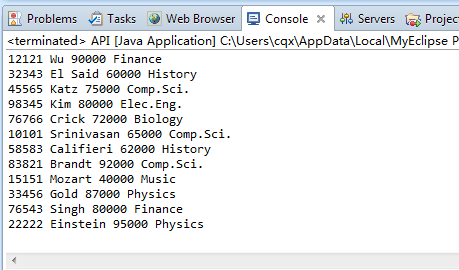
1. 更新，将Bob的salary改为80000，并打印整张表。

update instructor set salary=’80000’ where name=’Bob’



1. 删除，删除Bob的信息，并打印整张表。

delete from instructor where name=’Bob’

****

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 第八章 数据库物理设计实验

### 实验目的

1. 理解SQL SERVER数据库的物理设计的几个主要部分。
2. 掌握SQL SERVER数据库的文件（包括主文件、辅助文件和日志文件）和文件组（主文件组、其他文件组）的设计。
3. 掌握SQL SERVER数据库索引的设计，包括聚集索引、非聚集索引和复合索引等，分析索引对select/update/insert/delete的影响。
4. 掌握利用SQL Server Management Studio分析观察SQL语句的查询执行计划，对比查询需求相同、实现方式不一样的不同SQL语句在查询结果、查询执行计划、执行时间上的区别。
5. 了解掌握定长、变长属性对关系表存储空间和访问速度的影响。

### 实验环境

采用SQL Server数据库管理系统作为实验平台。SQL Server要求选用2008或者2012版本，可以采用SQL Server Express、SQL Server Develop或SQL Server Enterprise等版本。

### 实验要求

了解数据库文件及文件组设计，理解不同类型文件和文件组的区别和工作机制，掌握查看数据库文件存储格式和创建文件及文件组的方法。

索引是物理设计的重要内容，有助于提高与索引相关部分查询的执行效率，但会增加空间开销和数据增删改的成本，而对不相关查询的效率则无影响。所以在实际数据库物理设计中需要综合平衡考虑后决定要建立什么样的索引。

设计2条SQL语句，这2条语句查询需求相同，但1条使用索引，另1条不使用索引。将这2条SQL语句同时提交给DBMS，比较它们在执行效果、执行速度、查询执行计划方面的区别。

设计2条SQL语句，这2条语句insert/update/delete需求相同，但1条使用索引，另1条不使用索引。将这2条SQL语句同时提交给DBMS，比较它们在执行效果、执行速度、查询执行计划方面的区别。

设计2条SQL语句，这2条语句查询需求相同，但1条使用聚集索引，另1条使用非聚集索引。将这2条SQL语句同时提交给DBMS，比较它们在执行效果、执行速度、查询执行计划方面的区别。

### 实验内容与步骤

针对TD-LTE配置数据库，参考下述实验示例，按照要求，完成下述实验内容。

#### 8.1 数据库文件及文件组的设计

背景知识

每个SQL Server 数据库至少具有两个操作系统文件：一个数据文件和一个日志文件。数据文件包含数据和对象，例如表、索引、存储过程和视图。日志文件包含恢复数据库中的所有事务所需的信息。为了便于分配和管理，可以将数据文件集合起来，放到文件组中。

SQL Server 数据库具有三种类型的文件，如下表所示：

|  |  |
| --- | --- |
| 文件类型 | 说明 |
| 主文件 | 主要数据文件包含数据库的启动信息，并指向数据库中的其他文件。用户数据和对象可存储在此文件中，也可以存储在次要数据文件中。每个数据库有一个主要数据文件。主要数据文件的建议文件扩展名是 .mdf。 |
| 辅助/副本文件 | 次要数据文件是可选的，由用户定义并存储用户数据。通过将每个文件放在不同的磁盘驱动器上，次要文件可用于将数据分散到多个磁盘上。另外，如果数据库超过了单个 Windows 文件的最大大小，可以使用次要数据文件，这样数据库就能继续增长。  次要数据文件的建议文件扩展名是 .ndf。 |
| 事务日志文件 | 事务日志文件保存用于恢复数据库的日志信息。每个数据库必须至少有一个日志文件。事务日志的建议文件扩展名是 .ldf。 |

8.1.1 查看数据库的文件存储格式

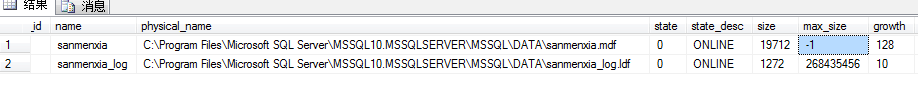
实验要求和内容

针对在实验1、实验2/3中创建的TD-LTE配置数据库，采用select语句，从数据库元数据视图sys.database\_files中观察数据库文件的组成，包括文件类型、文件名、文件大小、文件存储位置等。

示例

对已有的数据库，可用相应SQL语句查询数据库现有的文件及其属性，这些信息可从数据库元数据视图sys.database\_files中看到。





8.1.2 数据库文件设计

实验内容和要求

1. 使用CREATE DATABASE语句创建TD-LTE配置数据库，指定数据库相关属性，如逻辑文件名（NAME）、物理文件名（FILENAME）、初始文件大小（SIZE）、文件最大尺寸（MAXSIZE）、文件成长增量（FILEGROWTH）；
2. 通过ALTER DATABASE语句，对已经创建的数据库文件的属性进行修改；
3. 通过数据库元数据视图sys.database\_files中观察创建、修改后的数据库的文件属性。

示例

1. 在创建数据库阶段，可以利用CREATE DATABASE语句创建数据库，并在该语句中定义数据库文件相关属性对应的参数。如果语句未指定参数，则系统创建数据库时采用默认参数。



注意文件路径选择自己电脑下相应的文件路径，创建完成后可以用第一部分的语句查看文件的属性。默认情况下，数据和事务日志被放在同一个驱动器上的同一个路径下。这是为处理单磁盘系统而采用的方法。但是，在生产环境中，这可能不是最佳的方法。建议将数据和日志文件放在不同的磁盘上。

1. 如果表已经创建好，可以用ALTER DATABASE语句对文件的属性进行修改。



8.1.3 数据库文件组的设计

实验内容和要求

利用CREATE DATABASE语句，创建数据库文件、文件组，并将不同文件组分不到不同盘片上。

背景知识

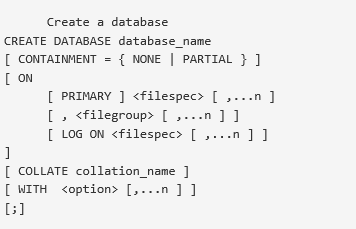
每个数据库有一个主要文件组。此文件组包含主要数据文件和未放入其他文件组的所有次要文件。可以创建用户定义的文件组，用于将数据文件集合起来，以便于管理、数据分配和放置。

例如，可以分别在三个磁盘驱动器上创建三个文件Data1.ndf、Data2.ndf和Data3.ndf，然后将它们分配给文件组fgroup1，接着可以明确地在文件组fgroup1上创建一个表。对表中数据的查询将分散到三个磁盘上，从而提高了性能。通过使用在RAID（独立磁盘冗余阵列）条带集上创建的单个文件也能获得同样的性能提高。但是，文件和文件组能够轻松地在新磁盘上添加新文件。

下表列出了存储在文件组中的所有数据文件。

|  |  |
| --- | --- |
| 文件组 | 说明 |
| 主 | 包含主要文件的文件组。所有系统表都被分配到主要文件组中。 |
| 用户定义 | 用户首次创建数据库或以后修改数据库时明确创建的任何文件组。 |

1. 在创建数据库时创建文件组语法



说明：如果在数据库中创建对象时没有指定对象所属的文件组，对象将被分配给默认文件组。不管何时，只能将一个文件组指定为默认文件组。

默认文件组中的文件必须足够大，能够容纳未分配给其他文件组的所有新对象。PRIMARY文件组是默认文件组，除非使用ALTER DATABASE语句进行了更改。但系统对象和表仍然分配给PRIMARY文件组，而不是新的默认文件组。

示例

文件和文件组示例（在设计自己的文件组时，注意文件路径。）

（1）创建文件和文件组

（2）修改默认文件组



（3）在用户定义的文件组上新建表：



更正：主文件组中的主文件必须大于5MB。

#### 8.2 数据库索引的设计

实际数据库应用系统中，索引是影响系统性能的一个非常重要的因素，因此必须合理设置数据库系统内各表上的索引。

索引影响到select、update、insert、delete等数据库访问的执行效率。

对select-from-where操作，如果在where查询条件涉及到的查询、连接属性上建立了聚集或非聚集索引，则有利于提高访问速度。

对update-set-where操作，如果在where查询条件属性上建立了索引，有利于在数据库文件中快速定位update需要访问的元组。但如果set操作修改了索引涉及到的属性的值，则会引发DBMS对索引的重组。索引重组带来额外的系统开销，降低了update的执行速度。

例如，对关系表Student(ID, name, dept, age)，在属性ID上建立可聚集或非聚集索引，该索引未必能提高下述update的执行速度。

update tbStudent

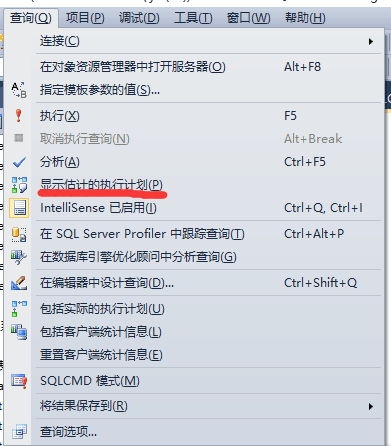
set ID=ID +1

where ID>20

对delete-from-where操作，如果在where查询条件属性上建立了索引，有利于在数据库文件中快速定位由where条件定义的需要删除的元组。但删除这些元组后，将引起DBMS将对索引进行调整重组，引起额外的系统开销，可能会降低delete的执行速度。

对insert操作，在关系表中插入新元组后，DBMS将根据新插入的元组在索引属性上的值，对表中的索引进行调整重组，必然引起额外系统开销，降低insert的执行速度。

**注意事项**：查看SQL语句查询执行计划的方法(在做比较时,将两条语句放在一条查询里一起执行,可看到执行计划开销的百分比的比较)。



8.2.1 数据库索引的设计及对select操作的影响分析

在做这部分实验时，要求使用tbATUData表,按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条select语句（比较有无索引，比较强制使用索引、强制不使用索引），判断这2条语句的执行结果是否一样，以及对比执行效果和执行速度，比较索引创建前后的索引和数据的存储空间

**第1步** 创建tbATUData的备份表tbATUDataNew；

**第2步** 查看表属性里面的索引和数据存储空间。

**第3步** 在tbATUDatanew创建索引。

索引属性建议：可以在Longitude和Latitude上设计索引（见表2.1）。

**第4步** 编写select语句，使用索引访问tbATUDatanew；编写select语句，不使用索引方式，访问tbATUData，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

**第5步** 编写select语句，强制使用索引访问tbATUDatanew；编写select语句，强制不使用索引方式，访问tbATUData，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

**第6步** 查看表属性里面的索引和数据存储空间。

示例

实验步骤和内容可以参照给出的示例，这些示例中的访问对象为tbMROData。

下面以tbMROData为例，首先使用select into语句备份一个tbMRODataNew表，在tbMRODataNew上面创建索引，和在tbMROData上进行操作，并比较执行计划和语句执行的时间。

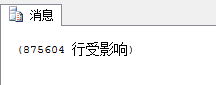
表2.1

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段中文名称 | 数据类型 | 数据取值范围  说明/完整性约束 |
| Seq |  | bigint |  |
| FileName | 路测轨迹文件名 | nvarchar(255) | 主属性 |
| Time | 测试时间 | varchar(100) |  |
| Longitude | 测试点经度 | float |  |
| Latitude | 测试点纬度 | float |  |
| CellID | 服务小区ID | nvarchar(50) |  |
| TAC | 服务小区跟踪区编码 | Int |  |
| ECI |  | varchar(50) |  |
| EARFCN | 服务小区频点 | int |  |
| PCI | 服务小区PCI | smallint |  |
| RSRP | 服务小区参考信号接收功率RSRP | float |  |
| RS\_SINR | 服务小区信噪比SINR | float |  |
| SCell\_Dist\_km |  | float |  |
| SCell\_Azimuth\_Diff |  | float |  |
| APP\_Speed\_Mbps | 路测点APP下载速率 | float |  |
| NCell\_ID\_1 | 第1邻小区/干扰小区的标识 | nvarchar(50) |  |
| NCell\_EARFCN\_1 | 第1邻小区/干扰小区频点 | int |  |
| NCell\_PCI\_1 | 第1邻小区/干扰小区物理小区标识 | smallint |  |
| NCell\_RSRP\_1 | 第1邻小区/干扰小区参考信号接收强度 | float |  |
| 略 |  |  |  |

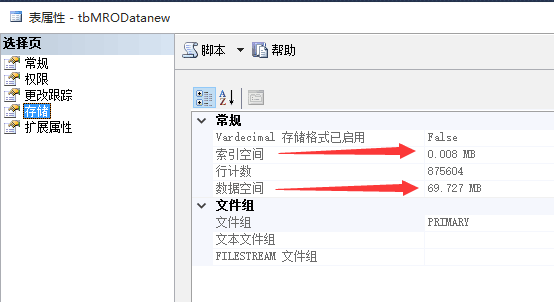
1.创建备份表tbMRODatanew

select\*intotbMRODatanew

fromtbMROData



2.查看表属性里面的索引和数据存储空间。

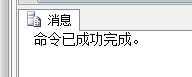


3.创建索引

在备份表上创建索引

createindextimeide

ontbMRODatanew(TimeStamp)



4.比较有无索引的时间开销

无索引

select \* from tbMRODataNew

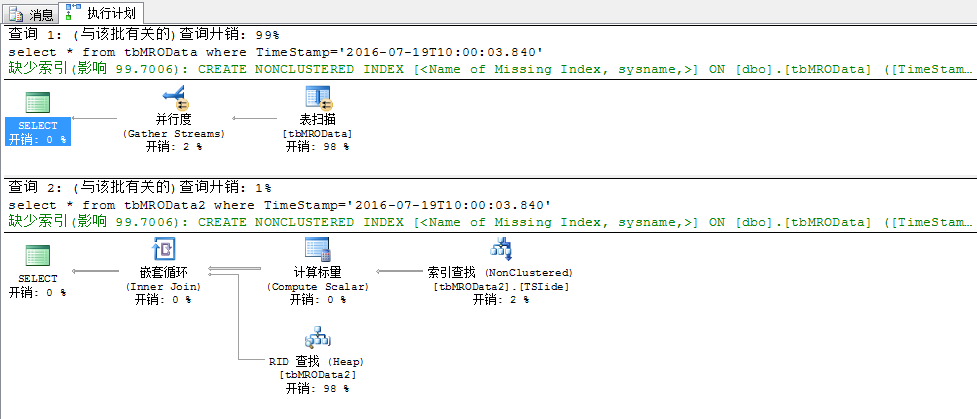
where TimeStamp='2020-07-19T10:00:03.840'

有索引

select \* from tbMRODatanew

where TimeStamp = '2020-07-19T10:00:03.840'

无索引和有索引的执行计划比较



5.强制使用索引和强制不使用索引的比较

强制使用

select \* from tbMRODataNew with(index = timeide)

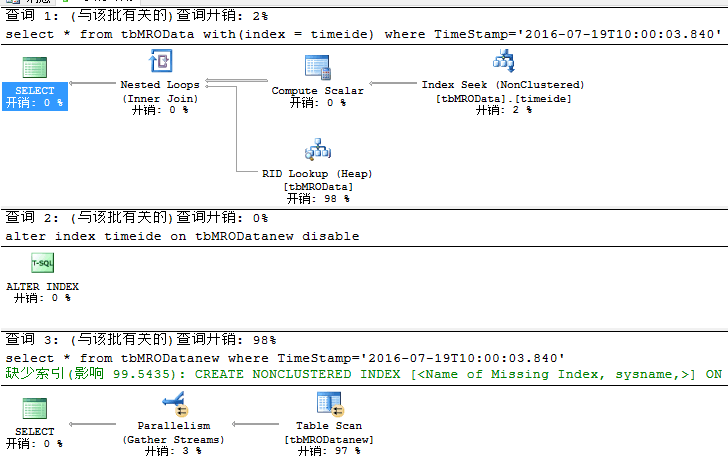
where TimeStamp='2020-07-19T10:00:03.840'

强制不使用

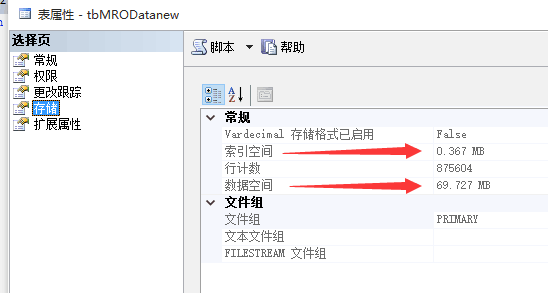
alter index timeide on tbMRODataNew disable

select \* from tbMRODataNew

where TimeStamp='2020-07-19T10:00:03.840'



6.查看索引空间和数据空间



8.2.2 数据库索引的设计及对insert/update/delete操作的影响分析

在做这部分实验时，要求使用tbAdjCell表,按照下述步骤完成实验内容。对访问相同的两张表且需求完全一样的2条insert/update/delete语句（比较有无索引的影响），判断这2条语句的执行结果是否一样，以及对比执行效果和执行速度，比较创建索引前后的索引和数据存储空间。

**第1步** 创建tbAdjCell的备份表tbAdjCellNew；

**第2步**  在tbAdjCellNew创建索引。

索引属性建议：可以在S\_SECTOR\_ID或N\_SECTOR\_ID上设计索引（见表2.2）。

**第3步** 在tbATUDatanew编写insert语句，添加一条元组数据（包含索引）；在tbATUData编写同样的一条insert语句，将2条语句同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

**第4步** 在tbATUDatanew编写update语句，修改一条元组数据（包含索引）；在tbATUData编写同样的一条update语句，将2条语句同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

**第5步** 在tbATUDatanew编写delete语句，删除第三步添加的数据；在tbATUData编写同样的一条delete语句，将2条语句同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

表2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段中文名称 | 数据类型 | 数据取值范围  说明/完整性约束 |
| S\_SECTOR\_ID | 主小区/服务小区ID | nvarchar(50) | 主属性 |
| N\_SECTOR\_ID | 邻小区ID | nvarchar(50) | 主属性 |
| S\_EARFCN | 主小区频点 | int | 取值{  {37900,38098,38400,38950,39148,…} |
| N\_EARFCN | 邻小区频点 | int | 取值{37900,38098,38400,38950,39148,…} |

示例

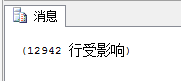
实验步骤和内容可以参照给出的示例，这些示例中的访问对象为tbCell。

下面以tbCell为例，首先使用select into语句备份一个tbCellnew表，在tbCellnew上面创建索引，和在tbCell上进行操作，并比较执行计划和语句执行的时间。

1.备份新表

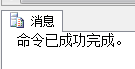
select\*into tbCellnew

fromtbCell



2.创建索引

createindexIDontbCellnew(SECTOR\_ID)



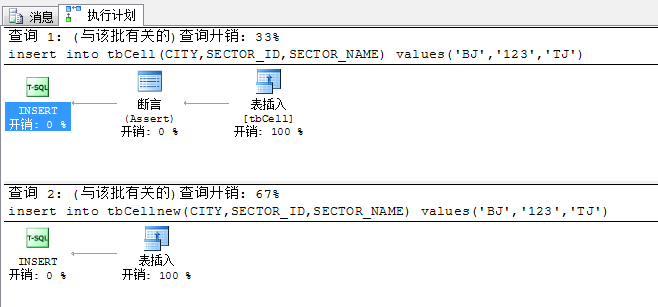
3.Insert

无索引插入

insertintotbCell(CITY,SECTOR\_ID,SECTOR\_NAME)values('BJ','123','TJ')

有索引插入

insertintotbCellnew(CITY,SECTOR\_ID,SECTOR\_NAME)values('BJ','123','TJ')



4.Update

修改索引：

无索引修改

updatetbCellsetSECTOR\_ID='11111-0'

whereENODEBID='124674'

有索引修改

updatetbCellnewsetSECTOR\_ID='11111-0'

whereENODEBID='124674'



不修改索引：

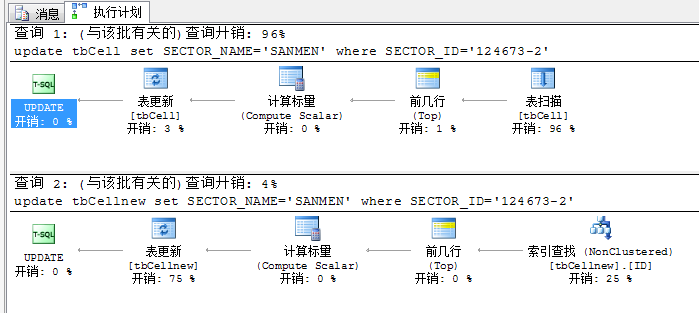
无索引修改

updatetbCellsetSECTOR\_NAME='SANMEN'

whereSECTOR\_ID='124673-2'

有索引修改

updatetbCellnewsetSECTOR\_NAME='SANMEN'

whereSECTOR\_ID='124673-2'

5.Delete

无索引查找删除

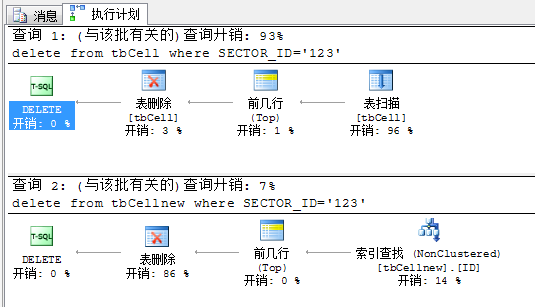
deletefromtbCell

whereSECTOR\_ID='123'

按索引查找删除

deletefromtbCellnew

whereSECTOR\_ID='123'



8.2.3 聚集索引和非聚集索引的比较

在做这部分实验时，要求使用tbATUData表,按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条select语句（比较有无聚集索引的区别，以及使用聚集索引和非聚集索引的区别），判断这2条语句的执行结果是否一样，对比执行效果和执行速度。并查看强制不使用聚集索引的影响。

实验步骤

**第1步** 创建tbATUData的备份表tbATUDataNew；

**第2步** 在tbATUDatanew创建聚集索引。

编写select语句，使用聚集索引访问tbATUDatanew；编写select语句，不使用聚集索引方式，访问tbATUData，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

索引属性建议：可以在FileName或CellID 上设计聚集索引，在Longitude和Latitude上设计非聚集索引（见表2.1）。

**第3步** 在tbATUData创建非聚集索引。

编写select语句，使用聚集索引访问tbATUDatanew；编写select语句，使用非聚集索引方式，访问tbATUData，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

**第4步** 在tbATUDatanew强制不使用聚集索引，写一条查询语句观察执行结果。

示例

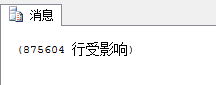
实验步骤和内容可以参照给出的示例，这些示例中的访问对象为tbMROData。

下面以tbMROData为例，首先使用select into语句备份一个tbMRODataNew表，在tbMRODataNew上面创建聚集索引和非聚集索引，和在tbMROData上进行操作，并比较执行计划和语句执行的时间。

1.创建备份表tbMRODatanew

select\*intotbMRODatanew

fromtbMROData



2.创建聚集索引并比较有无索引的时间开销

在备份表上创建聚集索引

createclustered

indexSIideontbMRODatanew(ServingSector,InterferingSector)

无索引

select\*fromtbMROData

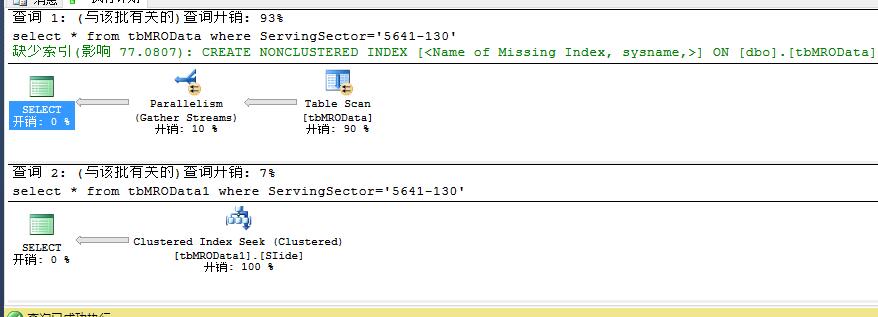
whereServingSector='5641-130'

有索引

select\*fromtbMRODatanew

whereServingSector='5641-130'

有无索引的执行计划比较



3.聚集索引和非聚集索引的比较

在tbMROData创建非聚集索引

createindexSIideontbMROData(ServingSector,InterferingSector)

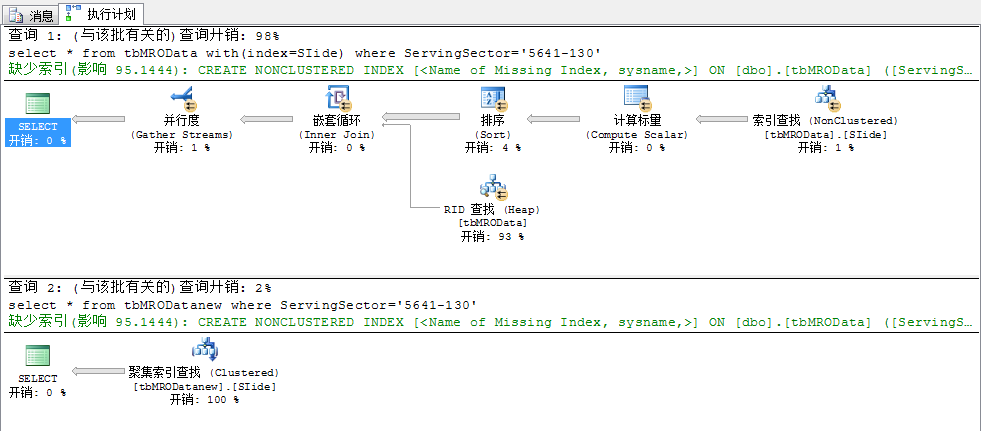
查询比较

select\*fromtbMRODatawith(index=SIide)

whereServingSector='5641-130'

select\*fromtbMRODatanew

whereServingSector='5641-130'



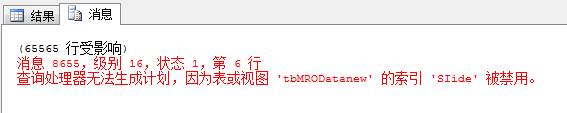
4. 强制不使用聚集索引

alterindexSIideontbMRODatanew

disable

select\*fromtbMRODatanew

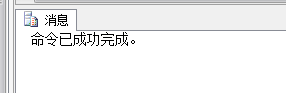
whereServingSector='5641-130'



结果显示强制不使用聚集索引会使得表不能访问，这时可以重建聚集索引

alterindexSIideontbMRODatanew

rebuild



#### 8.3 定长变长数据类型对数据访问影响分析

背景知识

数据库中与字符串相关的数据类型有char、nchar、varchar、nvarchar等，其中前两个是定长数据类型，后两个是变长数据类型。所谓定长就是长度固定的，当输入的数据长度没有达到指定的长度时将自动以英文空格在其后面填充，使长度达到相应的长度，有var前缀的，表示是实际存储空间是变长的，变长字符数据则不会以空格填充。此外，在数据库中，英文字符只需要一个字节存储就足够了，但汉字和其他众多非英文字符，则需要两个字节存储。如果英文与汉字同时存在，由于占用空间数不同，容易造成混乱，导致读取出来的字符串是乱码。Unicode字符集就是为了解决字符集这种不兼容的问题而产生的，它所有的字符都用两个字节表示，即英文字符也是用两个字节表示。而前缀n就表示Unicode字符，比如nchar,nvarchar，这两种类型使用了Unicode字符集。

char存储定长属性数据很方便，在char字段上的索引效率很高；varchar存储变长属性数据，存储效率没有char高，但是更灵活。从空间上考虑，用varchar合适；从效率上考虑，用char合适，关键是根据实际情况找到权衡点。再从属性字符的角度考虑，如果含有中文字符，用nchar/nvarchar，如果纯英文和数字，用char\varchar。

**实验目的和要求**

将前面实验中用到的tbATUData或tbAdjCell中的变长字符串类型属性(如小区标识)改成定长字符串类型，组成新表。在新旧两表都不使用索引的情况下，观察两表的大小差异，对两张表提交查询需求相同的select语句，观察对比这两条语句在查询执行计划、查询执行时间方面的异同，并解释执行时间出现差异的原因。

**第1步** 将tbATUDatanew或tbATUData中的变长字符串属性修改为定长字符串类型，分为两种情况，第一种情况将字符串定长长度改为50，第二种情况将字符串定长长度改为表中列名长度；

**第2步** 分别查看两种情况下tbATUDatanew和tbATUData的文件大小；

**第3步** 分别在两种情况下，对两张表提交查询需求相同的select语句，观察对比这两条语句在查询执行计划、查询执行时间方面的异同。

**第4步** 分别在两种情况下，对两张表提交查询需求相同的不带where条件的select语句，观察对比这两条语句在查询执行计划、查询执行时间方面的异同。

示例

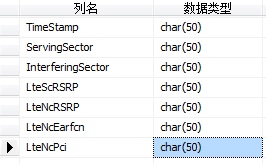
实验步骤和内容可以参照给出的示例，示例中的访问对象为tbMROData。

首先修改tbMRODatanew中的变长字符串类型为定长，然后分别查看tbMROData和tbMRODatanew的文件大小，最后在两表上执行相同的查询语句并比较。

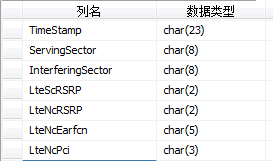
8.3.1修改tbMRODatanew字符串类型

直接点开表里面的列属性界面进行修改。

8.3.1.1 将字符串定长长度改为50



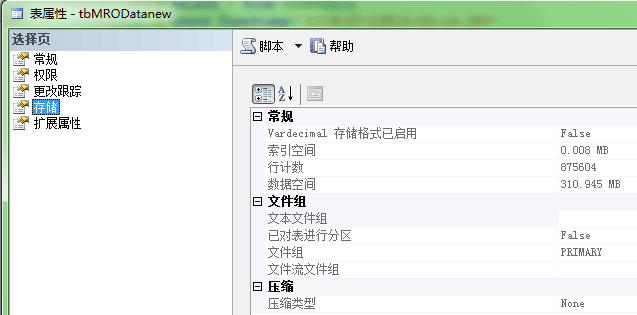
8.3.1.2 将字符串定长长度改为表中对应的列名长度



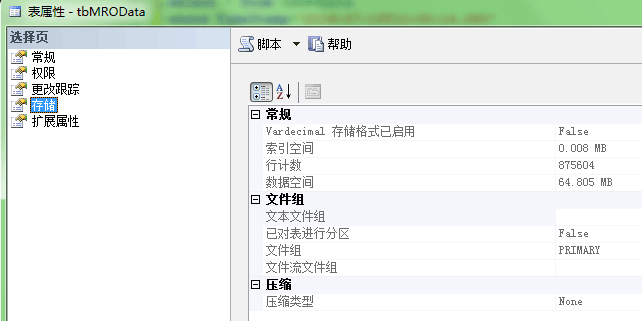
8.3.2 查看两张表文件的大小

8.3.2.1 字符串长度为50的情况下两张表文件的大小

tbMRODatanew：

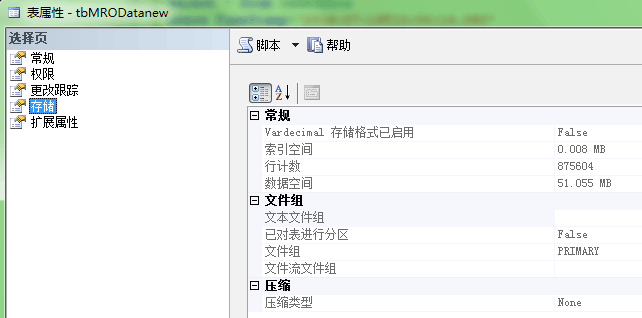


tbMROData：

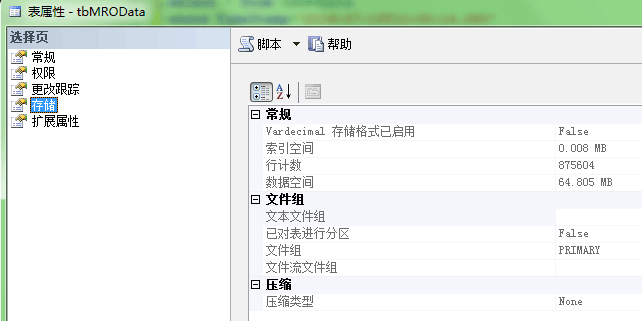


8.3.2.2 字符串长度为表格中对应列名长度的情况下两张表文件的大小

tbMRODatanew:



tbMROData:



8.3.3 执行相同的查询语句并比较执行计划

8.3.3.1 字符串长度为50的情况下执行相同的查询语句并比较执行计划

语句：

select\*fromtbMROData

whereTimeStamp='2020-07-19T10:00:08.960'

select\*fromtbMRODatanew

whereTimeStamp='2020-07-19T10:00:08.960'

查询执行计划：



8.3.3.2 字符串长度为表格中对应列名长度的情况下执行相同的查询语句并比较执行计划

语句：

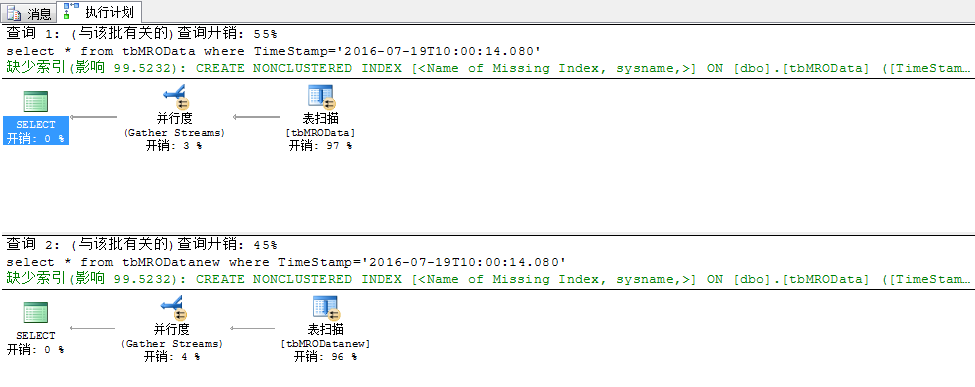
select \* from tbMROData

where TimeStamp='2020-07-19T10:00:14.080'

select \* from tbMRODatanew

where TimeStamp='2020-07-19T10:00:14.080'

查询执行计划：



8.3.4 执行相同的查询语句（不带where条件）并比较执行计划

8.3.4.1 字符串长度为50的情况下执行相同的查询语句并比较执行计划

语句：

select \* from tbMROData

select \* from tbMRODatanew

查询执行计划：



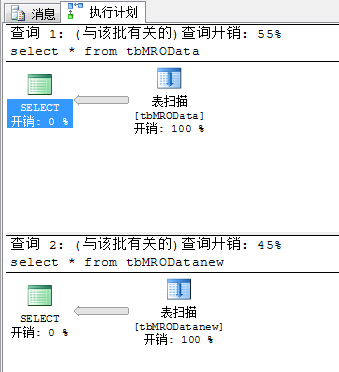
8.3.4.2 字符串长度为表格中对应列名长度的情况下执行相同的查询语句并比较执行计划

语句：

select \* from tbMROData

select \* from tbMRODatanew

查询执行计划：



8.3.5 结果分析

理论上，表中属性全部采用定长数据类型吋，表中每行存贮为一个定长记录; 当表中有属性采用变长数据类型时，表中每行采用变长记录存贮方式。数据库文件采用定长记录时，单个记录的访问效率较高，但可能导致整个数据库文件占用更多存贮空间，对整表访问的速度反而有可能降低。上述例子中，在定长和变长数据类型的参数n相同时，定长表所占的空间更大，查询的开销也更大。将定长表中的数据类型的n值改为对应属性的长度后，不仅存储空间变小了，查询也变快了。这说明在设置定长数据类型时，需要准确的设置长度，从而避免空间和效率上的浪费，查询语句中带不带where条件没有显著的影响。所以在建表时要均衡多方面的因素来选择数据类型，从而使数据库性能更佳。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 第九章 数据查询分析优化实验

### 实验目的

* 1. 在SQL Server Management Studio平台下，通过观察Select/Insert/delete/update等SQL语句的查询执行计划，分析查询执行计划中连接、选择、投影等关系代数操作的实现方式及其执行成本。熟悉了解SQL SERVER数据库中查询优化的使用，理解数据库查询优化的基本概念。
  2. 掌握利用SQL Server Management Studio提供的机制，分析对比形式不同、执行结果等价的不同SQL语句的查询执行计划的执行成本和执行时间差异。
  3. 熟悉了解视图和with临时视图的创建，观察视图查询、with临时视图查询的执行计划。
  4. 参照文档“数据库物理设计及查询优化”中SQL语句查询优化相关内容，在多种情况下，对比实现方式不同但查询结果相同的等价SQL语句在执行计划和成本方面的差异，加深对查询优化的理解，进行书写优化SQL语句的初步训练，提高编写高效SQL语句的能力。涉及以下几方面：

1）复合索引左前缀；

2）多表连接属性上建立索引；

3）索引对小表查询的作用；

4）查询条件中函数对索引的影响；

5）多表嵌入式SQL查询；

6）where查询条件中复合查询条件OR对索引的影响；

7）聚集运算中索引设计；

8）Select子句中有无distinct的区别；

9）union、union all的区别；

10）from子句中多余关系的影响。

### 实验环境

SQL查询语句提交到DBMS后，DBMS需要对查询语句进行优化，以提高查询语句执行效率，降低查询执行成本。SQL SERVER在执行查询语句时分为两个步骤：第一步是对语句进行查询处理和优化，生成查询执行计划，第二步执行该计划。查询处理和优化分为语法分析、中间代码生成（表示为关系代数表达式）和优化三个阶段，完成查询处理和优化后系统将把执行计划保存在缓存中，以后执行该查询时可直接调用，而省略重新处理优化过程。然后执行引擎将计划复制为可执行形式并执行之。

本实验采用SQL SERVER数据库管理系统作为实验平台，可以采用SQL SERVER 2008或2012，并使用其各种版本。

### 实验内容

#### 9.1 SQLServer执行计划的查看与分析

##### 实验要求

在SQL Server Management Studio平台下，按照下述实验步骤，编写相应的SQL语句，观察Select/Insert/delete/update等SQL语句的查询执行计划，分析查询执行计划中连接、选择、投影等关系代数操作的实现方式，观察分析查询执行计划中关系代数操作的执行成本。

##### 实验步骤

1. 编写SQL语句完成下面两条查询；
2. 查询tbATUData中PCI小于100且在tbATUC2I中干扰强度小于3的主小区的切换目标小区。

（2）查询tbHandOver中小区类型为“优化区”且设备厂家为华为的切换源小区和切换成功率。

1. 查看语句的执行计划；
2. 分析执行计划中各关系代数的实现方式与执行成本。

##### 示例

1. 查询要求

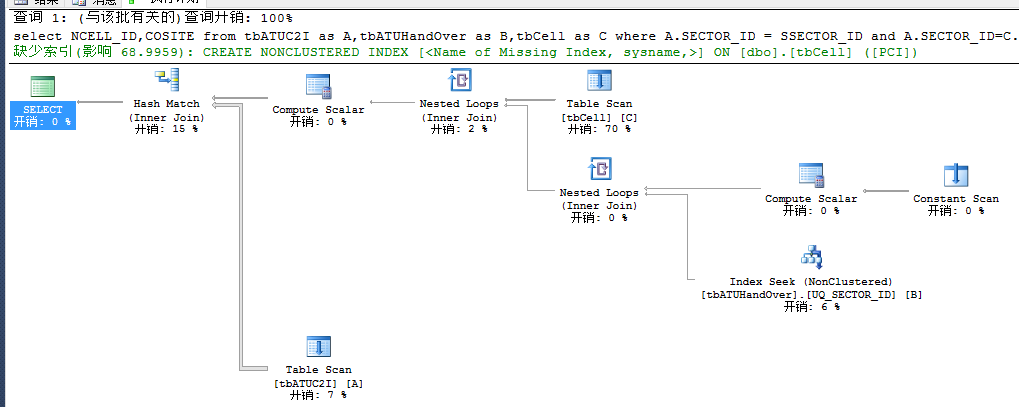
查询tbATUC2I中与tbATUHandOver中切换源小区相同且PCI为323的服务小区的干扰小区ID并查看它们是否为同站小区；

select NCELL\_ID,COSITE

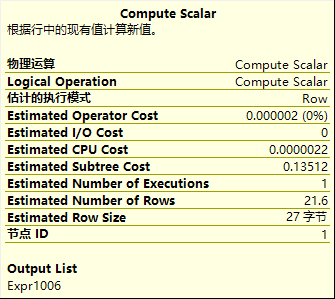
from tbATUC2I asA,tbATUHandOverasB,tbCellas C

where A.SECTOR\_ID = SSECTOR\_ID and A.SECTOR\_ID=C.SECTOR\_ID and PCI ='323'

1. 查看执行计划



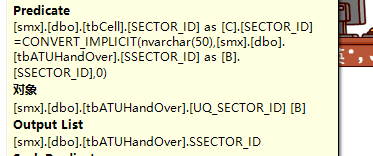
把光标移到特定的部分可查看其详细信息，比如上图的Compute Scalar（计算标量）



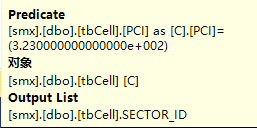
1. 分析

执行计划应该从右往左看，箭头的粗细表示每一步产生的数据量的大小，每一步操作下面有其对应的开销，理论上，通过优化开销最大的部分可以优化整个查询。上述例子主要通过以下几个步骤完成查询：

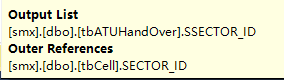
（1）在tbATUHandOver进行了非聚集索引的扫描（开销：6%），从而筛选出符合条件的SSECTOR\_ID（与tbCell中的SECTOR\_ID相等）；



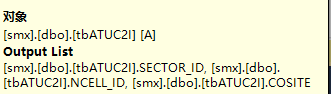
（2）对tbCell进行全表扫描（开销70%），选出PCI=323的行；



（3）通过嵌套循环（内部连接，开销2%）参照tbCell的SECTOR\_ID输出tbATUHandOver中的SSECTOR\_ID；



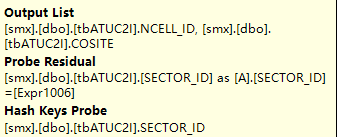
（4）对tbC2I进行全表扫描（开销7%），输出其SECTOR\_ID、NCELL\_ID和COSITE；



（5）通过哈希匹配（内部连接，开销15%），输出最终结果。



（顶部和底部参考完整执行计划中的各个部分的位置。此例中是通过筛选第（4）步中的SECTOR\_ID从而输出最终结果。）



说明：此查询中扫描tbCell全表开销最大，一个可选的优化方案是在该表上建立索引。

#### 9.2 观察视图查询、with临时视图查询的执行计划

##### 实验要求

从实验数据库中选取一张表在上面建立视图和with临时视图，执行一个查询，观察其执行计划。

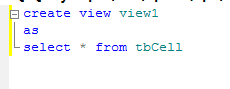
##### 实验步骤

1. 选取数据表的一张表（示例中的表除外），分别在上面建立视图和with临时视图；
2. 分别在两个视图上执行同一个查询语句，将它们同时提交给DBMS；
3. 查看执行计划并分析。

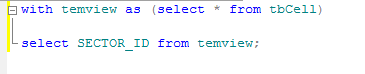
##### 示例

1. 分别在tbCell上建立视图和临时视图

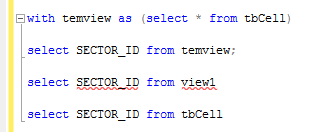
建立视图：



建立临时视图：



1. 查询



1. 查看执行计划



说明：在视图、临时视图和原表上的查询没有太大的差异，但这并不是说视图对性能无影响。

#### 9.3 优化SQL语句

结合文档“数据库物理设计及查询优化”中相关内容，通过对以下各种不同情况下查询语句的执行情况的对比分析，巩固加深查询优化的理解，并进行书写优化SQL语句的初步训练，提高编写高效SQL语句进行数据查询的能力。

##### 9.3.1 复合索引左前缀

###### 实验要求

在做这部分实验时，要求使用tbATUData表,按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条select语句实现两个要求（在tbATUData上创建包含至少三个索引的组合索引）：

1. 比较有最左前缀索引和无索引的2条select语句的执行结果和执行计划；
2. 比较组合索引中其他索引（不包含最左前缀索引）和无索引的2条select语句的执行计划。

分别判断2条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

###### 实验步骤

1. 创建tbATUData的备份表tbATUDataNew；
2. 在tbATUDatanew创建组合索引（至少包含三个索引）。

索引属性建议：可以在Longitude、Latitude、CellID上设计组合索引。

1. 比较有最左前缀索引和无索引的2条select语句的执行结果和执行计划

编写select语句，使用最左前缀索引访问tbATUDatanew；编写select语句，不使用索引方式，访问tbATUData，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

1. 在比较组合索引中其他索引（不包含最左前缀索引）和无索引的2条select语句的执行计划

编写select语句，使用不含最左前缀的索引访问tbATUDatanew；编写select语句，不使用索引方式，访问tbATUData，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

###### 示例

所用表为tbMROData。

1.创建组合索引

createindextsi

ontbMRODatanew(TimeStamp,ServingSector,InterferingSector)

2.无索引和有最左前缀索引

无索引

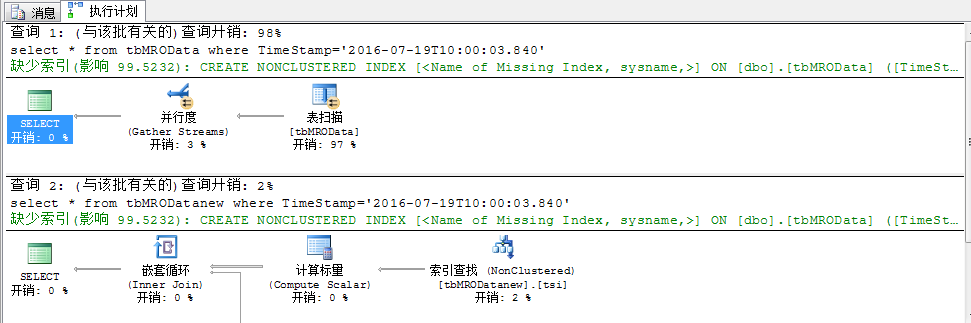
select\*fromtbMROData

whereTimeStamp='2020-07-19T10:00:03.840'

有最左前缀索引

select\*fromtbMRODatanew

whereTimeStamp='2020-07-19T10:00:03.840'



3.无索引和用组合索引中的第二个索引

无索引

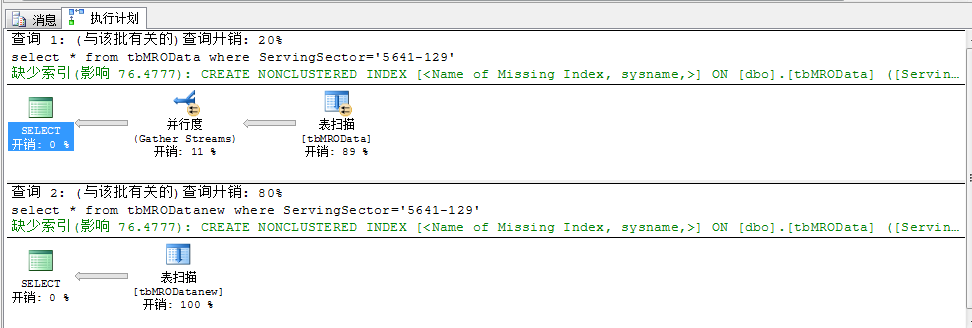
select\*fromtbMROData

whereServingSector='5641-129'

用组合索引中的第二个索引

select\*fromtbMRODatanew

whereServingSector='5641-129'



说明：在使用复合索引时要注意最左前缀的原则，以防索引失效。

##### 9.3.2 多表连接操作，在连接属性上建立索引

###### 实验要求

在做这部分实验时，要求使用tbATUData和tbAdjCell表,按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条select语句实现两个要求之一（在连接属性上建立索引）：

（1）在tbATUDatanew表的CellID上设计索引，列举在tbAdjCell中找到此小区的主小区频点。

（2）在tbATUDatanew表的NCell\_ID\_1上设计索引，列举在tbAdjCell中找到此邻小区对应的主小区。

每个要求中，分别判断2条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

###### 实验步骤

1．创建tbATUData的备份表tbATUDatanew；

2．在tbATUDatanew创建索引；

3．比较有索引和无索引的2条两表select语句的执行结果和执行计划。

编写select语句，使用索引访问tbATUDatanew，tbAdjCell；编写select语句，不使用索引方式，访问tbATUData，tbAdjCell，将2条select语句，同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

###### 示例

所用表为tbMROData**。**

1.创建索引

createindexInd

ontbMRODatanew(ServingSector)

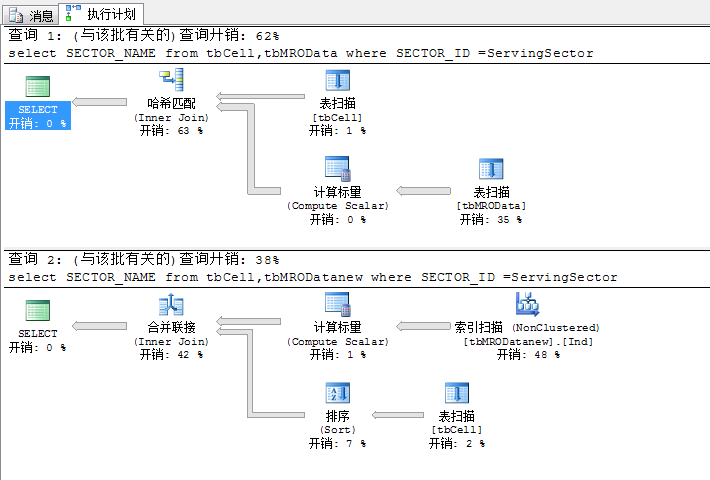
2.比较有索引和无索引的2条两表select语句

selectSECTOR\_NAMEfromtbCell,tbMROData

whereSECTOR\_ID=ServingSector

selectSECTOR\_NAMEfromtbCell,tbMRODatanew

whereSECTOR\_ID=ServingSector



说明：建立适当的索引可以节省开销。

##### 9.3.3 索引对小表查询的作用

###### 实验要求

在做此部分实验时要求使用元组数目少、表占用空间少的tbOptCell表，按照下述步骤完成实验内容。在SECTOR\_ID上创建索引，执行一个查询，查看执行计划观察是否用到索引，如果没有则强制使用并与不强制使用做比较。

###### 实验步骤

1. 创建tbOptCell的备份表tbOptCellnew；
2. 在tbOptCellnew上创建索引；
3. 在tbOptCellnew上新建查询，观察是否用到索引；
4. 编写select语句，访问tbOptCell，强制使用索引访问tbOptCellnew，将2条select语句同时提交给DBMS执行后，从2个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

###### 示例

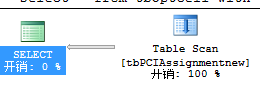
所用表为tbPCIAssignment，该表只有200+行数据，所占空间在一个page内。

1. 创建tbPCIAssignment的备份表，并在其上创建索引；

createindex id1 ontbPCIAssignmentnew(ASSIGN\_ID,SECTOR\_ID)

1. 查询并观察是否有用到索引；

select\*fromtbPCIAssignmentnew

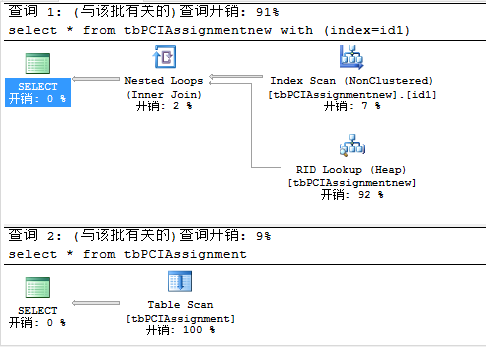


从执行计划可以看出没有用到索引。

1. 强制使用索引与无索引比较；

select\*fromtbPCIAssignmentnewwith (index=id1)

select\*fromtbPCIAssignment



说明：强制使用索引开销会增大很多，对于较小的表不建立索引反而查询会更快。

##### 9.3.4 查询条件中函数对索引的影响

###### 实验要求

针对同一张表，编写2条查询需求完全一样的等价的select语句，一条语句在where子句查询条件中使用了函数，另外一条语句则没有使用函数。

观察这2条等价的SQL语句在查询执行计划、索引使用和实现效率方面的差异，掌握如何避免函数导致查询条件索引失效的方法，编写优化的SQL语句。

实现以下两个要求之一：

（1）在路测ATU数据表tbATUData中查询与经度为111.82841且小区标识为253903-0的小区A周边一定范围内的全部小区B，周边一定范围定义为：

小区B所在基站的经纬度与小区A所在基站的纬度差距满足：

ABS(小区A所在基站纬度-小区A所在基站纬度)<=0.001

（2）在路测ATU数据表tbATUData中查询与纬度为34.75102且小区标识为253903-2的小区A周边一定范围内的全部小区B，周边一定范围定义为：

小区B所在基站的经纬度与小区A所在基站的经度差距满足：

ABS(小区A所在基站经度-小区A所在基站经度)<=0.001,

###### 实验步骤

1. 创建备份表并在备份表上创建索引；
2. 编写select语句，完成实验要求中的查询，对备份表同时提交2条等价的查询语句；
3. 查看执行计划，比较两个语句的执行结果和执行速度，观察函数对索引的影响。

###### 示例

以tbATUData为例。

1. 在备份表tbATUDatanew上创建索引

create index newindex

on tbATUDatanew(Latitude)

create index alindex

on tbATUDatanew(LONGITUDE)

1. 在备份表上提交2条等价的查询语句（第一条的where条件中含有ABS函数,第二条是不含ABS函数的第一条查询语句的等价实现形式）

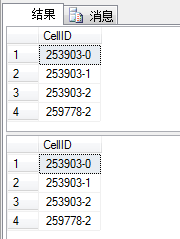
select distinct B.CellID FROM tbATUDatanew as A,tbATUDatanew as B

where A.CellID='253903-0' and A.Latitude='33.75315' and ABS(A.Longitude-B.Longitude)<=0.001

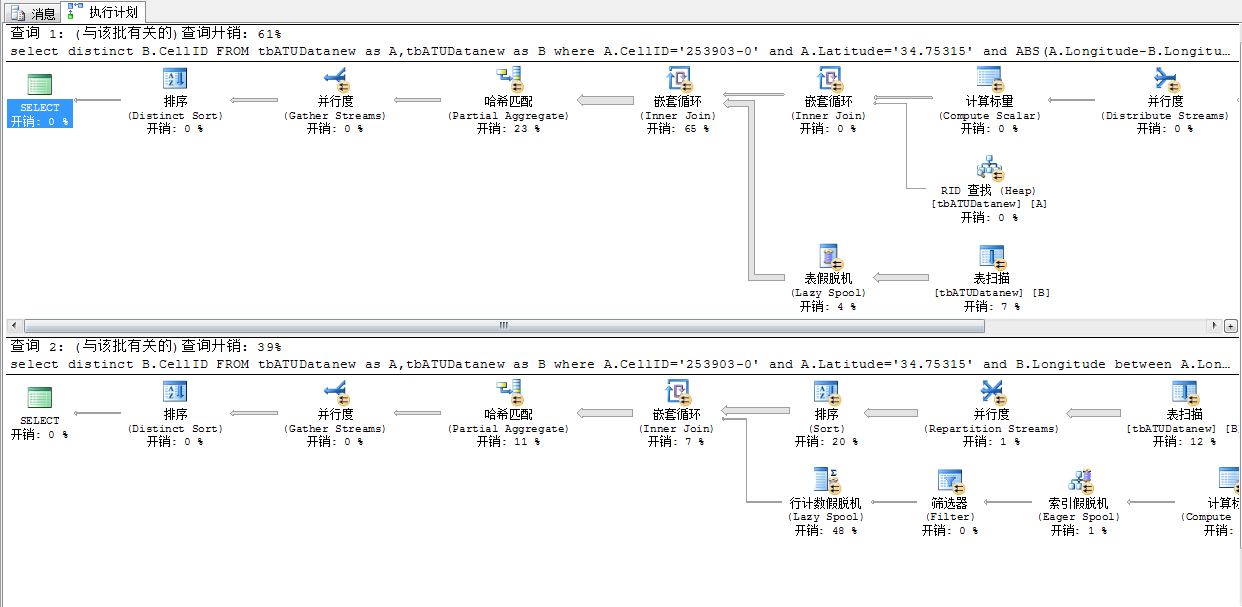
select distinct B.CellID FROM tbATUDatanew as A,tbATUDatanew as B

where A.CellID='253903-0' and A.Latitude='33.75315' and B.Longitude between A.Longitude-0.001 and A.Longitude+0.001

1. 查看执行结果



1. 查看执行计划



说明：当索引上存在函数计算时，索引就不起作用了，可转换为等价的实现方式。

##### 9.3.5 多表嵌入式SQL查询

###### 实验要求

对访问相同的两张表且查询需求完全一样的2条select语句，一条使用嵌套查询，一条使用连接查询，实现两个要求之一：

（1）根据优化小区/保护带小区表和小区PCI优化调整结果表，查询小区类型为“优化区”的小区经调整后的PCI。

（2）根据路测ATU数据表和一阶邻区关系表，查询列举服务小区PCI为167的主小区频点。

在每个要求中，分别判断2条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

###### 示例

1. 嵌套查询

select\*fromtbMROData,tbCell

whereTimeStamp='2020-07-19T10:00:03.840'andServingSector='5641-129'andInterferingSector=SECTOR\_ID

andInterferingSectorin(selectSECTOR\_ID

fromtbCell

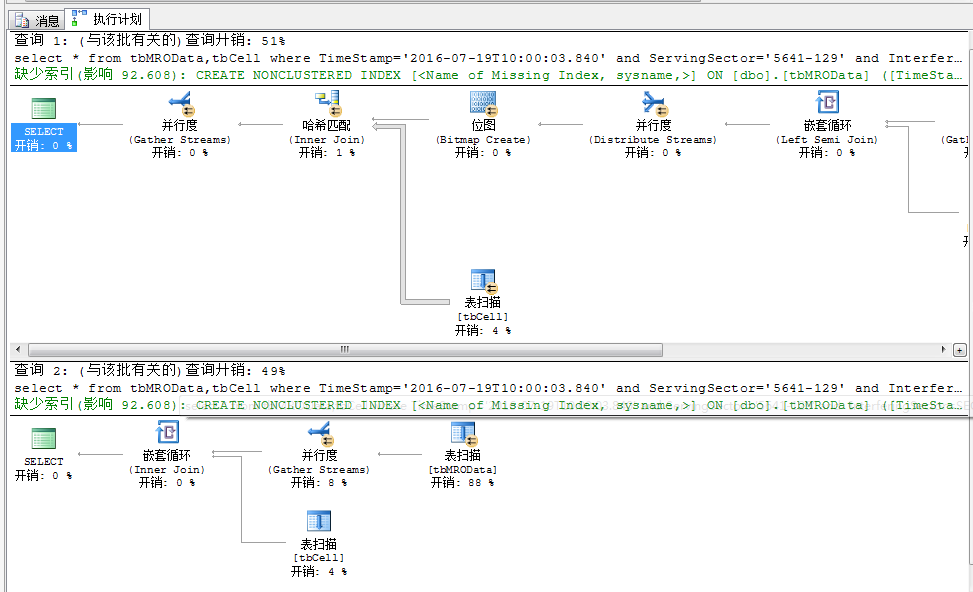
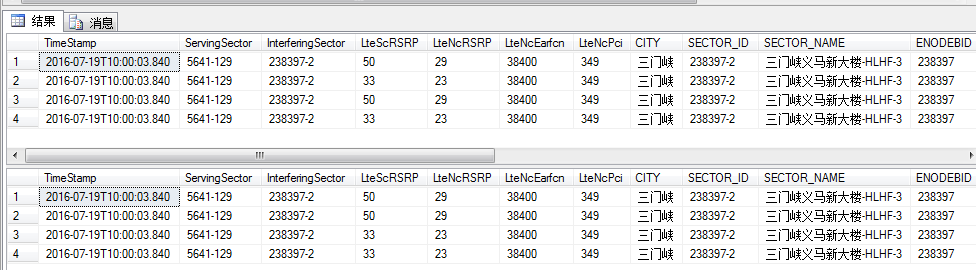
whereSECTOR\_ID='238397-2')

1. 连接查询

select\*fromtbMROData,tbCell

whereTimeStamp='2020-07-19T10:00:03.840'andServingSector='5641-129'andInterferingSector=SECTOR\_IDandtbCell.SECTOR\_ID='238397-2'

1. 查看结果和执行计划



说明：一般在使用多表查询的时候应该避免使用嵌套查询，但在实际应用中有时会出现两种情况性能相同甚至是嵌套查询性能更优的情况。

##### 9.3.6 where查询条件中复合查询条件OR对索引的影响

###### 实验要求

在做这部分实验时，要求使用tbATUData表,按照下述步骤完成实验内容。在tbATUDatanew上创建一个索引，查询条件中为“A OR B”，A为创建的索引条件，B为非索引条件，提交select语句并查看执行计划中是否还能用到索引。再将含有“A OR B”查询条件的语句改为等价的两条select语句的union，对比这两种实现方式的执行效率。

###### 实验步骤

1. 创建tbATUData的备份表tbATUDatanew；
2. 在tbATUDatanew上创建索引；
3. 编写select语句，where查询条件为“A OR B”，A为定义在索引属性上的查询条件，B为定义在非索引属性上的查询条件；将语句提交给DBMS执行后，查看执行计划是否使用了索引；
4. 将含有“A OR B”查询条件的select语句：

Select …

From …

Where … and (A OR B)

转换为等价的2条select语句的union：

Select …

From …

Where … and A

Union

Select …

From …

Where … and B

分析对比这两种查询实现方式的执行效率。

###### 示例

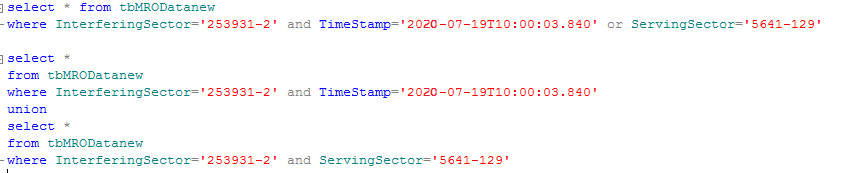
1.创建索引

在tbMRODatanew的TimeStamp上创建索引

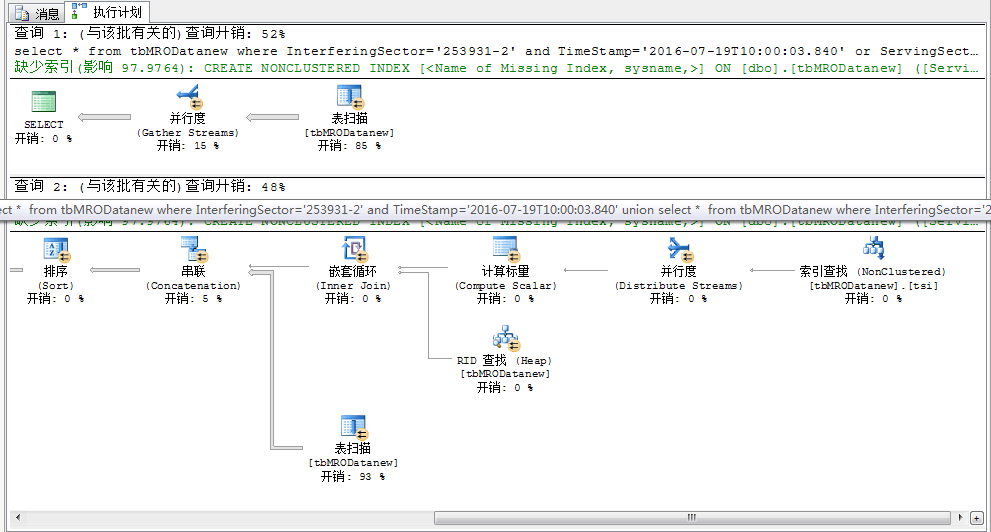
createindextimindex

ontbMRODatanew(TimeStamp)

2.查询语句（同时执行where中含有OR条件的语句和将OR改成union的语句）



3.查看执行计划



说明：含有union的语句执行效率更低，从执行计划来看，第一个语句因为where条件中引入了OR而使得索引不再起作用，但是后一个查询步骤繁多，虽然使用索引，但是却还是进行了全表扫描，效率反而更低。

##### 9.3.7 聚集运算中的索引设计

###### 实验要求

选定作为聚集运算查询对象的关系表，对group by操作的分组属性，建立聚集/非聚集索引，对聚集运算（如count、sum、avg）的属性，建立非聚集索引。分析比较等价的有索引聚集运算、无索引聚集运算查询在查询执行计划、执行速度方面的区别。完成以下两个要求之一：

（1）在路测ATU数据表tbATUData，计算同一纬度的小区的平均经度值是多少并按纬度分组（注意索引的建立）；

（2）在路测ATU数据表tbATUData，计算同一经度的小区的平均纬度值是多少并按经度分组（注意索引的建立）。

###### 实验步骤

1. 创建备份表；
2. 根据查询要求在备份表上创建索引；
3. 根据查询要求，在原表和备份表上编写相同的select语句并将它们同时提交给DBMS；
4. 查询执行计划，分析比较等价的有索引聚集运算、无索引聚集运算查询在查询执行计划、执行速度方面的区别。

###### 示例

以tbATUData为例。

1. 分别在seq和Longtitude创建索引；、

create index log on tbATUData(Longitude)

create index seq on tbATUData(seq)

1. 查询同一seq的最大经度并按seq分组；

select seq,max(Longitude) as max\_Longitude

from tbATUData

group by seq

select seq ,max(Longitude) as max\_Longitude

from tbATUData with (index=seq,index=log)

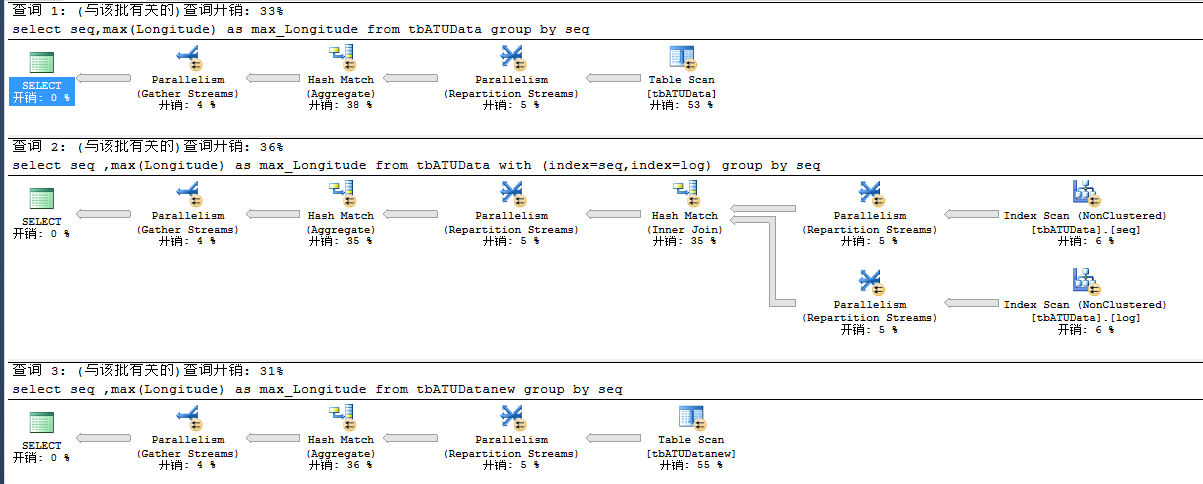
group by seq

select seq ,max(Longitude) as max\_Longitude

from tbATUDatanew

group by seq

1. 查询执行计划。



说明：上述例子，有索引的表在查询时没有使用索引，但仍然比无索引的表查询慢，在有索引的表上强制使用索引后，查询速度为三者中最慢。

##### 9.3.8 Select子句中有无distinct的区别

###### 实验要求

在做这部分实验时，按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条select语句，一条使用distinct，一条不使用distinct，实现两个要求之一：

（1）根据优化小区/保护带小区表，查询频点编号为38400且小区类型为保护带的小区ID。

（2）根据路测ATUC2I干扰矩阵表，查询干扰小区ID为253936-1的主小区ID。

在每个要求中，分别判断2条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

###### 示例

以tbPCIAssignment为例。

1.无distinct

selectPCI

fromtbPCIAssignment

whereSECTOR\_ID='124818-0'andPSS=0

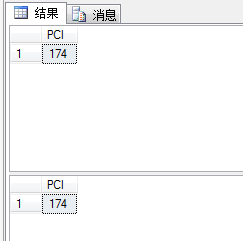
2.有distinct

selectdistinctPCI

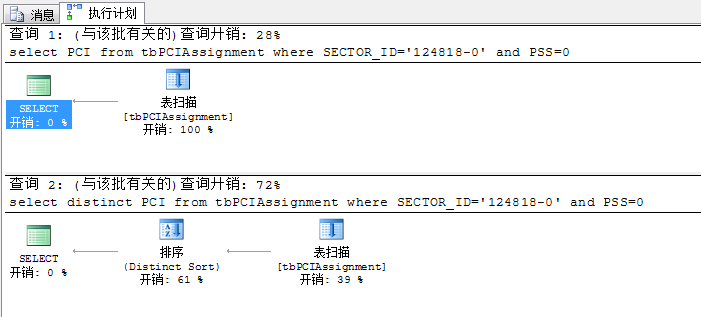
fromtbPCIAssignment

whereSECTOR\_ID='124818-0'andPSS=0

3.查看执行结果



4.查看执行计划



说明：使用distinct有时反而会增大开销。

##### 9.3.9 union、union all的区别

###### 实验要求

在做这部分实验时，按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条select语句，一条使用union，一条使用union，实现两个要求之一：

（1）找出所有二阶（同频）邻区关系表，一阶邻区关系表邻小区ID。

（2）找出路测ATUC2I干扰矩阵表中的主小区ID和路测ATU切换统计矩阵的切换源小区ID并总和到一起。

在每个要求中，分别判断2条语句的执行结果是否一样，对比执行效果和执行速度，解释执行时间出现差异的原因。

###### 示例

以tbCell和tbOptCell为例。

1.使用union

selectSECTOR\_IDfromtbCell

union

selectSECTOR\_IDfromtbOptCell

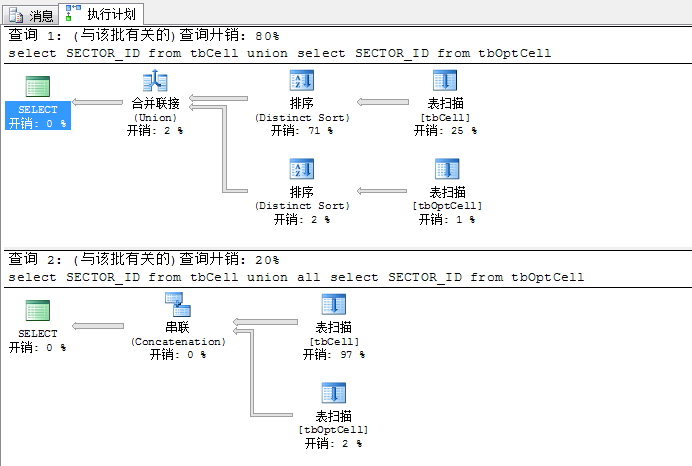
2.使用union all

selectSECTOR\_IDfromtbCell

unionall

selectSECTOR\_IDfromtbOptCell

3.查看执行计划



说明：上述例子union all的性能要优于union。

##### 9.3.10 from中存在多余的关系表，即查询非最简化

###### 实验要求

在做这部分实验时，按照下述步骤完成实验内容。对访问相同的两张表且查询需求完全一样的2条等价select语句，一条在from后面增加一个多余的关系表（但在where子句中增加连接条件），另一条from后面最简化。

可自行选择查询语句，要求至少两表连接查询。分别判断2条语句的执行效果和执行速度，解释执行时间出现差异的原因。

###### 示例

1.没有多余关系表

select distinct tbOptCell.EARFCN

from tbPCIAssignment,tbOptCell

where tbPCIAssignment.SECTOR\_ID=tbOptCell.SECTOR\_ID

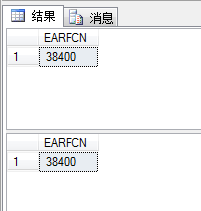
2.增加了一个多余的关系表tbCell和对应的连接条件

select distinct tbOptCell.EARFCN

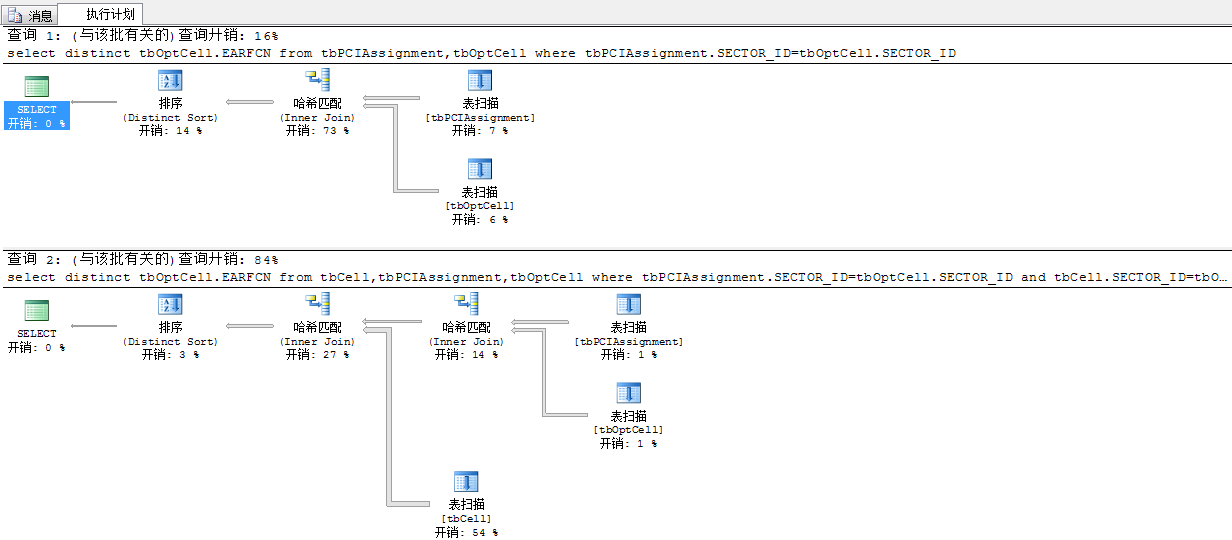
from tbCell,tbPCIAssignment,tbOptCell

where tbPCIAssignment.SECTOR\_ID=tbOptCell.SECTOR\_ID and tbCell.SECTOR\_ID=tbOptCell.SECTOR\_ID

3.查看执行结果



4.查看执行计划



说明：增加多余的表会大大的增加开销。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 第十章 数据库备份与恢复实验

### 实验目的

1. 理解SQL SERVER数据库的数据备份和恢复机制。
2. 掌握SQL SERVER数据库的数据备份和恢复的基本概念，例如备份方式（增量备份和完全备份）、备份介质（文件或者设备）等等。
3. 掌握备份和恢复的实际操作，能够备份和将备份恢复，特别是能够恢复到一个新的数据库中。
4. 理解备份/维护策略或备份/维护计划的概念。

### 实验环境

采用SQL SERVER数据库管理系统作为实验平台。其中，SQL SERVER数据库可以采用SQL SERVER 2008、2012、2014或2016。

备份（Backup）和恢复（Restore）为最强有力的恢复手段，它不仅仅应用于保护数据库，而是应用于保护一切信息：我们对于操作系统或者其它各种信息（诸如多媒体文件）等都可以进行备份，待事故时进行恢复。由于硬件故障、用户错误、应用程序故障、软件故障、某用户拥有过多权限、局部灾难等可能导致数据库故障，所以DBA（Database Administrator数据库管理员）必须在平时做好备份工作，而且因为故障发生时刻是未知的，还必须做多个备份，从而可以恢复到故障前尽可能短时间的数据库状态以减少损失。DBA必须做好备份/恢复计划，并实际验证之，以确保故障时能够恢复数据库。

### 实验内容与步骤

#### 10.1 校验数据库

在执行数据库备份之前，首先必须保证数据库的一致性。这里最重要的工具是：DBCC。可以用两种方法运行DBCC：通过命令行窗口或查询分析器（Query Analyzer）窗口。如果你认为必要，你还可以确定其操作的时间。DBCC命令包括以下扩展：

CheckDB：检测整个数据库的一致性，是检查数据库破坏的基本方法。   
CheckTable：检测特定表的问题。   
CheckAlloc：检测数据库的单个页面，包括表和目录。   
Reindex：重建某个特定表的目录。   
CacheStats：说明当前存储在内存缓存中的对象。   
DropCleanBuffers：释放当前存储在缓冲区中的所有数据，这样你就可以继续进行检测，而不必使用前面的结果。   
Errorlog：删除（缩短）当前日志。

1. 检查整个数据库的有效性
   1. 校验整个数据库的有效性；
2. 校验单个表
   1. 校验单个表的有效性

经过检查未发现错误的数据库是我们开始备份操作的基础。

#### 10.2 备份数据库

**选定一个数据库：**

1. 执行完全备份，在图形界面上是在数据库名上鼠标右击选择任务，然后选择备份，命令行是backup database。
2. 执行部分备份，同上。
3. 在下列基于SQL SERVER 2008的备份类型中任意执行3种，并解释其含义
   1. 文件/文件组备份
   2. 差异备份
   3. 文件差异备份
   4. 仅复制备份
   5. 纯事务日志备份
4. 执行压缩备份

#### 10.3 恢复数据库

1. 恢复完整备份到原来的数据库。在图形界面上是在数据库名上鼠标右击选择任务，然后选择还原，命令行是restore database。下同。
2. 恢复部分备份到原来的数据库。
3. 恢复压缩备份为原来的数据库。
4. 恢复完整备份为一个新的数据库，要求新数据库内容和老数据库一样，但数据库名字不同，下同。另外使用图形界面则需要先新建一个新的空数据库，然后还原。而且要在文件选项中修改数据库文件和日志文件名称为新的数据库的相关文件。在选项中选择覆盖现有数据库。
5. 恢复部分备份为一个新的数据库。
6. 恢复压缩备份为一个新的数据库。

backup database和restore database命令语法：

Backing Up a Whole Database

BACKUP DATABASE { database\_name | @database\_name\_var }

TO <backup\_device> [ ,...n ]

[ <MIRROR TO clause> ] [ next-mirror-to ]

[ WITH { DIFFERENTIAL | <general\_WITH\_options> [ ,...n ] } ]

[;]

Backing Up Specific Files or Filegroups

BACKUP DATABASE { database\_name | @database\_name\_var }

<file\_or\_filegroup> [ ,...n ]

TO <backup\_device> [ ,...n ]

[ <MIRROR TO clause> ] [ next-mirror-to ]

[ WITH { DIFFERENTIAL | <general\_WITH\_options> [ ,...n ] } ]

[;]

Creating a Partial Backup

BACKUP DATABASE { database\_name | @database\_name\_var }

READ\_WRITE\_FILEGROUPS [ , <read\_only\_filegroup> [ ,...n ] ]

TO <backup\_device> [ ,...n ]

[ <MIRROR TO clause> ] [ next-mirror-to ]

[ WITH { DIFFERENTIAL | <general\_WITH\_options> [ ,...n ] } ]

[;]

Backing Up the Transaction Log (full and bulk-logged recovery models)

BACKUP LOG { database\_name | @database\_name\_var }

TO <backup\_device> [ ,...n ]

[ <MIRROR TO clause> ] [ next-mirror-to ]

[ WITH { <general\_WITH\_options> | <log-specific\_optionspec> } [ ,...n ] ]

[;]

<backup\_device>::=

{

{ logical\_device\_name | @logical\_device\_name\_var }

| { DISK | TAPE | URL} =

{ 'physical\_device\_name' | @physical\_device\_name\_var }

}

<MIRROR TO clause>::=

MIRROR TO <backup\_device> [ ,...n ]

<file\_or\_filegroup>::=

{

FILE = { logical\_file\_name | @logical\_file\_name\_var }

| FILEGROUP = { logical\_filegroup\_name | @logical\_filegroup\_name\_var }

}

<read\_only\_filegroup>::=

FILEGROUP = { logical\_filegroup\_name | @logical\_filegroup\_name\_var }

<general\_WITH\_options> [ ,...n ]::=

--Backup Set Options

COPY\_ONLY

| { COMPRESSION | NO\_COMPRESSION }

| DESCRIPTION = { 'text' | @text\_variable }

| NAME = { backup\_set\_name | @backup\_set\_name\_var }

| CREDENTIAL

| ENCRYPTION

| FILE\_SNAPSHOT

| { EXPIREDATE = { 'date' | @date\_var }

| RETAINDAYS = { days | @days\_var } }

--Media Set Options

{ NOINIT | INIT }

| { NOSKIP | SKIP }

| { NOFORMAT | FORMAT }

| MEDIADESCRIPTION = { 'text' | @text\_variable }

| MEDIANAME = { media\_name | @media\_name\_variable }

| BLOCKSIZE = { blocksize | @blocksize\_variable }

--Data Transfer Options

BUFFERCOUNT = { buffercount | @buffercount\_variable }

| MAXTRANSFERSIZE = { maxtransfersize | @maxtransfersize\_variable }

--Error Management Options

{ NO\_CHECKSUM | CHECKSUM }

| { STOP\_ON\_ERROR | CONTINUE\_AFTER\_ERROR }

--Compatibility Options

RESTART

--Monitoring Options

STATS [ = percentage ]

--Tape Options

{ REWIND | NOREWIND }

| { UNLOAD | NOUNLOAD }

--Log-specific Options

{ NORECOVERY | STANDBY = undo\_file\_name }

| NO\_TRUNCATE

--Encryption Options

ENCRYPTION (ALGORITHM = { AES\_128 | AES\_192 | AES\_256 | TRIPLE\_DES\_3KEY } , encryptor\_options ) <encryptor\_options> ::=

SERVER CERTIFICATE = Encryptor\_Name | SERVER ASYMMETRIC KEY = Encryptor\_Name

--To Restore an Entire Database from a Full database backup (a Complete Restore):

RESTORE DATABASE { database\_name | @database\_name\_var }

[ FROM <backup\_device> [ ,...n ] ]

[ WITH

{

[ RECOVERY | NORECOVERY | STANDBY =

{standby\_file\_name | @standby\_file\_name\_var }

]

| , <general\_WITH\_options> [ ,...n ]

| , <replication\_WITH\_option>

| , <change\_data\_capture\_WITH\_option>

| , <FILESTREAM\_WITH\_option>

| , <service\_broker\_WITH options>

| , <point\_in\_time\_WITH\_options—RESTORE\_DATABASE>

} [ ,...n ]

]

[;]

--To perform the first step of the initial restore sequence

-- of a piecemeal restore:

RESTORE DATABASE { database\_name | @database\_name\_var }

<files\_or\_filegroups> [ ,...n ]

[ FROM <backup\_device> [ ,...n ] ]

WITH

PARTIAL, NORECOVERY

[ , <general\_WITH\_options> [ ,...n ]

| , <point\_in\_time\_WITH\_options—RESTORE\_DATABASE>

] [ ,...n ]

[;]

--To Restore Specific Files or Filegroups:

RESTORE DATABASE { database\_name | @database\_name\_var }

<file\_or\_filegroup> [ ,...n ]

[ FROM <backup\_device> [ ,...n ] ]

WITH

{

[ RECOVERY | NORECOVERY ]

[ , <general\_WITH\_options> [ ,...n ] ]

} [ ,...n ]

[;]

--To Restore Specific Pages:

RESTORE DATABASE { database\_name | @database\_name\_var }

PAGE = 'file:page [ ,...n ]'

[ , <file\_or\_filegroups> ] [ ,...n ]

[ FROM <backup\_device> [ ,...n ] ]

WITH

NORECOVERY

[ , <general\_WITH\_options> [ ,...n ] ]

[;]

--To Restore a Transaction Log:

RESTORE LOG { database\_name | @database\_name\_var }

[ <file\_or\_filegroup\_or\_pages> [ ,...n ] ]

[ FROM <backup\_device> [ ,...n ] ]

[ WITH

{

[ RECOVERY | NORECOVERY | STANDBY =

{standby\_file\_name | @standby\_file\_name\_var }

]

| , <general\_WITH\_options> [ ,...n ]

| , <replication\_WITH\_option>

| , <point\_in\_time\_WITH\_options—RESTORE\_LOG>

} [ ,...n ]

]

[;]

--To Revert a Database to a Database Snapshot:

RESTORE DATABASE { database\_name | @database\_name\_var }

FROM DATABASE\_SNAPSHOT = database\_snapshot\_name

<backup\_device>::=

{

{ logical\_backup\_device\_name |

@logical\_backup\_device\_name\_var }

| { DISK | TAPE | URL } = { 'physical\_backup\_device\_name' |

@physical\_backup\_device\_name\_var }

}

Note: URL is the format used to specify the location and the file name for the Windows Azure Blob. Although Windows Azure storage is a service, the implementation is similar to disk and tape to allow for a consistent and seemless restore experince for all the three devices.

<files\_or\_filegroups>::=

{

FILE = { logical\_file\_name\_in\_backup | @logical\_file\_name\_in\_backup\_var }

| FILEGROUP = { logical\_filegroup\_name | @logical\_filegroup\_name\_var }

| READ\_WRITE\_FILEGROUPS

}

<general\_WITH\_options> [ ,...n ]::=

--Restore Operation Options

MOVE 'logical\_file\_name\_in\_backup' TO 'operating\_system\_file\_name'

[ ,...n ]

| REPLACE

| RESTART

| RESTRICTED\_USER | CREDENTIAL

--Backup Set Options

| FILE = { backup\_set\_file\_number | @backup\_set\_file\_number }

| PASSWORD = { password | @password\_variable }

--Media Set Options

| MEDIANAME = { media\_name | @media\_name\_variable }

| MEDIAPASSWORD = { mediapassword | @mediapassword\_variable }

| BLOCKSIZE = { blocksize | @blocksize\_variable }

--Data Transfer Options

| BUFFERCOUNT = { buffercount | @buffercount\_variable }

| MAXTRANSFERSIZE = { maxtransfersize | @maxtransfersize\_variable }

--Error Management Options

| { CHECKSUM | NO\_CHECKSUM }

| { STOP\_ON\_ERROR | CONTINUE\_AFTER\_ERROR }

--Monitoring Options

| STATS [ = percentage ]

--Tape Options

| { REWIND | NOREWIND }

| { UNLOAD | NOUNLOAD }

<replication\_WITH\_option>::=

| KEEP\_REPLICATION

<change\_data\_capture\_WITH\_option>::=

| KEEP\_CDC

<FILESTREAM\_WITH\_option>::=

| FILESTREAM ( DIRECTORY\_NAME = directory\_name )

<service\_broker\_WITH\_options>::=

| ENABLE\_BROKER

| ERROR\_BROKER\_CONVERSATIONS

| NEW\_BROKER

<point\_in\_time\_WITH\_options—RESTORE\_DATABASE>::=

| {

STOPAT = { 'datetime'| @datetime\_var }

| STOPATMARK = 'lsn:lsn\_number'

[ AFTER 'datetime']

| STOPBEFOREMARK = 'lsn:lsn\_number'

[ AFTER 'datetime']

}

<point\_in\_time\_WITH\_options—RESTORE\_LOG>::=

| {

STOPAT = { 'datetime'| @datetime\_var }

| STOPATMARK = { 'mark\_name' | 'lsn:lsn\_number' }

[ AFTER 'datetime']

| STOPBEFOREMARK = { 'mark\_name' | 'lsn:lsn\_number' }

[ AFTER 'datetime']

}

#### 10.4 备份恢复方法比较

1. 新建一个空数据库，试用导入和导出向导将数据从旧数据库复制到新数据库中。复制数据时请考虑如何复制主键、外键、索引等。
2. 新建一个空数据库，将“移动通信网络配置数据库”备份直接恢复到该数据库中。

比较两者的差异。具体包括恢复的完整性（除了数据以外的其它东西，比如主键、外键、索引等是否同步复制）和执行效率。

#### 10.5 创建维护计划

1．运行sp\_configure打开服务器配置选项。首先要先把show advanced options设置为1，然后将Agent XPs设置为1，注意一定要用RECONFIG使修改后的配置生效，其中RECONFIGURE WITH OVERRIDE可代替RECONFIG并且更加安全。

sp\_configure 'show advanced options', 1;     
GO     
RECONFIGURE WITH OVERRIDE;      
GO     
sp\_configure 'Agent XPs', 1;     
GO     
RECONFIGURE WITH OVERRIDE       
GO   
2．在SQL SERVER Managemnet Studio中打开管理（一个文件夹），选择维护计划下的新建维护计划，再选择维护计划向导进行相关备份操作的设置。设置每天凌晨1点自动执行备份，当然实际上为了安全应该在DBCC完成后再进行，现在也可以设置成备份是在某时刻运行，但同时一定要在DBCC结束后才行，这样如果DBCC运行时间加长，则备份时间就算到了，也会自动推迟到DBCC结束后。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 第十一章 事务及其并发控制实验

### 实验目的

通过单事务、串行事务、并发事务实验，了解SQL SERVER数据库系统中 1）事务组成方式和执行模式；2）对单事务和串行事务的原子性保障机制；3）基于锁和隔离级别的事务并发控制和对并发事务的一致性、独立性保障机制。

具体目标为：

1. 通过update操作对现有关系表进行修改，观察违反check约束的update执行结果，理解SQL SERVER对单个事务提供的原子性保障机制；

2. 在SQL SERVER三种事务执行模式下，完成串行事务提交与回滚实验，对数据库表进行查询（select）、更新（update）、插入(insert)、删除(delete)，了解SQL SERVER事务组成方式和commit/rollback对查询结果的影响，了解数据库内容/状态发生变化时SQL SERVER提供的事务原子性的保障机制；

3. 在显式执行模式下，通过串行执行多个事务，对现有数据库表增加、删除属性列，观察commit/rollback对执行结果的影响，了解数据库模式结构发生变化时SQL SERVER提供的事务原子性保障机制；

4. 观察单事务或串行事务中保存点（Save Point）回滚rollback对数据库访问结果的影响，了解SQL SERVER保存点机制；

5. 通过set XACT\_ABORT ON/OFF语句，控制多条SQL语句组成的单一事务的整体回滚、局部回滚，理解实际数据库系统中原子性保障机制与教科书中事务原子性概念的差异。

6. 观察SQL SERVER的加锁机制，包括对数据库、关系表、数据行等不同粒度的数据对象所施加的各种不同类型的锁；

7. 了解SQL SERVER提供的各种事务隔离级别。观察分析在“读提交”隔离级别下，并发事务执行导致的数据不一致性，如丢失修改（写-写错误）、读脏数据（写-读错误）、不可重复读（读-写错误）、幻象；

8. 分析对比多种隔离级别下，如“读提交”与“读未提交”、“可串行化”，并发事务执行导致的数据不一致性，了解SQL SERVER提供的事务一致性和独立性保障机制。

9.观察单事务、串行事务、并发事务执行update/insert/delete等操作时，SQL Server记录的日志类型和日志内容，了解SQL Server日志机制；观察分析当事务以commit结束、rollback结束，和系统发生crash时，DBMS采取的故障恢复动作日志内容。

### 实验环境

采用SQL SERVER数据库管理系统作为实验平台。其中，SQL SERVER 可以采用2008及2012、2014、2016的企业版本等高级版本。

### 实验背景

##### 一、SQL SERVER三种事务执行模式

用户对数据库的访问体现为数据库系统内部事务的执行。事务是DBMS管理用户访问的基本单位，事务执行应满足ACID原则。SQL SERVER中，事务分为以下三种类型，也称为事务的三种执行模式：

1. 显示事务执行模式

以BEGIN TRANSACTION开始，COMMIT TRANSACTION、ROLLBACK TRANSACTION结束，中间是一系列属于该事务数据操作，如SQL语句。除非显式执行COMMIT TRANSACTION或者ROLLBACK TRANSACTION，该事务不会完成。

注意：SQL SERVER中事务不会自己检查与应用领域相关的业务逻辑错误，所以需要事务设计实现者根据业务逻辑要求在事务中进行错误处理，进行相应的回滚ROLLBACK处理。显示事务具有如下形式：

**BEGIN TRAN**

BEGIN TRY

一系列SQL语句

**COMMIT TRAN**

END TRY

BEGIN CATCH

RAISERROR（‘Transaction Aborted’,16,1）

**ROLLBACK TRAN**

END CATCH

**ENDS TRANSACTION**

1. 隐式事务执行模式

使用SET IMPLICIT\_TRANSACTIONS ON语句来启动隐式事务执行模式，使用SET IMPLICIT\_TRANSACTIONS OFF语句来关闭隐式事务执行模式，使用COMMIT TRANSACTION或ROLLBACK TRANSACTION语句来结束隐式事务。

隐式事务具有如下形式：

SET IMPLICIT\_TRANSACTIONS ON

一系列SQL语句

COMMIT TRANSACTION (或COMMIT TRANSACTION)

SET IMPLICIT\_TRANSACTIONS OFF

1. 自动提交模式/自动提交事务

如果数据库连接没有设置为前两种事务模式，则DBMS对每一条SQL语句自动提交，即在该模式下每一条SQL语句是1个独立的事务。

##### 二、保存点SavePoint

SQL Server利用SAVEPOINT 语句定义事务执行过程中的中间点/状态，使用保存点将各组相关语句分开可以在事务内标识重要的状态。

使用 ROLLBACK TO SAVEPOINT 语句可以撤消该点后的所有更改，即将数据库状态退回到保存点以前的状态。

具体例子参见第15章讲义PPT中的“Appendix B SavePoint”。

##### 三、事务的局部回滚/整体回滚控制

根据教科书中事务原子性定义，事务中的数据库访问操作必须作为1个整体，要么都做，要么都不做；当事务执行过程中发生数据库访问操作错误导致回滚rollback时，必须回滚事务中已经完成的全部操作，使数据库恢复到事务开始之前的状态。

但在实际数据库系统中，从提高事务执行效率角度，允许事务执行发生故障时，只回滚导致错误的操作，即只进行局部回滚，无需回滚已经执行的全部操作。概念上，这种机制并不满足事务原子性要求。

SQL Server通过以下语句控制事务的局部回滚、全局回滚：

SET XACT\_ABORT ON/OFF

指定当Transact-SQL语句产生运行时错误时，是否自动回滚当前事务的全部操作。

语法：

SET XACT\_ABORT {ON|OFF}

注释：

当SET XACT\_ABORT为ON时，如果Transact-SQL语句产生运行时错误，整个事务将终止并回滚，数据库返回事务执行前的状态；为OFF时，只回滚事务中产生错误的Transact-SQL语句，而事务将继续执行其它SQL语句。编译错误（如语法错误）不受SET XACT\_ABORT的影响。

##### 四、SQL SERVER的锁与隔离级别

###### 1. 锁的粒度与模式

SQL Server 数据库引擎（即DBMS）具有多粒度加锁机制加锁，允许一个事务加锁不同类型的数据对象资源。为了尽量减少加锁加锁的开销，数据库引擎自动将资源加锁在适合任务的级别。加锁的数据对象的粒度小，例如加锁对象为“行”级，可以提高并发度，但系统管理锁的开销较高；加锁的数据对象的粒度大，例如加锁对象为表，会降低事务执行的并发度，因为加锁整个表限制了其他事务对表中任意部分的访问。但其开销较低，因为需要维护的锁较少。

下表列出了数据库引擎可以加锁的资源。

|  |  |
| --- | --- |
| **资源** | **说明** |
| RID | 用于加锁堆中的单个行的行标识符。 |
| KEY | 索引中用于保护可序列化事务中的键范围的行锁，e.g. 加锁SECTOR\_ID='124672-0的tbCell元组 |
| PAGE | 数据库中大小为 8 KB 页，例如数据页或索引页。 |
| EXTENT | 一组连续的大小为8KB的页，例如数据页或索引页。 |
| HoBT | 堆或B树。用于保护没有聚集索引的表中的B树（索引）或堆数据页的锁。 |
| TABLE | 包括所有数据和索引的整个表。 |
| FILE | 数据库文件。 |
| APPLICATION | 应用程序专用的资源。 |
| METADATA | 元数据锁。 |
| ALLOCATION\_UNIT | 分配单元。 |
| DATABASE | 整个数据库。 |

SQL Server数据库引擎使用不同的锁模式加锁数据对象资源，这些锁模式确定了并发事务访问资源的方式。

下表显示了数据库引擎使用的资源锁模式。

|  |  |
| --- | --- |
| **锁模式** | **说明** |
| 共享（S） | 用于不更改或不更新数据的读取操作，如SELECT语句。 |
| 更新（U） | 用于可更新的资源中。防止当多个会话在读取、加锁以及随后可能进行的资源更新时发生常见形式的死锁。  当一个进程获得U锁访问一个数据对象时，不允许其它进程申请定义在该数据对象上的X锁或U锁 |
| 排他/互斥（X） | 用于数据修改操作，例如 INSERT、UPDATE 或 DELETE。 确保不会同时对同一资源进行多重更新。  当一个进程获得X锁访问一个数据对象时，不允许其它进程修改（update、insert）或读取被加锁的该数据对象，除非该进程的隔离级别为Read uncommitted |
| 意向 | 用于建立锁的层次结构。意向锁包含四种类型：意向共享(IS)、意向更新锁(IU)、意向排他(IX)和意向排他共享(SIX) |
| 模式Schema | 在执行依赖于表模式的操作时使用。模式锁包含两种类型：模式修改(Sch-M) 和模式稳定性(Sch-S)。 |
| 大容量更新（BU） | 在向表进行大容量数据复制且指定了TABLOCK提示时使用。 |
| 键范围 | 当使用可序列化事务隔离级别时保护查询读取的行的范围。确保再次运行查询时其他事务无法插入符合可序列化事务的查询的行。  例如，加锁latitude在[34.75102, 36.75102]间的全部tbCell元组 |

四种意向锁的解释如下：

|  |  |  |
| --- | --- | --- |
| IS | 意向共享锁 | 表示资源一部分被施加了共享锁，只能用于表级、页级资源。  e.g. 对 table S 施加IS锁，对表中的#S between [100,200]的行施加了S锁 |
| IU | 意向更新锁 | 表示资源一部分被施加了更新锁，只能用于表级、页级资源。 |
| IX | 意向互斥锁 | 表示资源一部分被施加了互斥锁，只能用于表级、页级资源。 |
| SIX | 共享与意向互斥锁 | 表示资源被施加了共享锁，并且资源一部分（页、行）被施加了互斥锁，一般用于DB、表级资源。 |

###### 2.事务的隔离级别与数据一致性

多个事务并发执行时，如果调度顺序不当，会导致数据不一致，如

1. 丢失修改，即写-写错误；
2. 读脏数据/写-读错误；
3. 不可重复读/读-写错误；
4. 幻象

具体解释参见第15章PPT讲义“Appendix A SQL Server中事务隔离级别与数据一致性”。

事务执行时可以指定一个隔离级别，该隔离级别定义了一个事务必须与由其他事务进行的资源或数据更改相隔离的程度。按照所允许或可接受的事务并发执行带来的副作用即造成的数据不一致性)的程度，例如，脏读或幻读取，定义多种隔离级别：未提交读/脏读(Read uncommitted)，已提交读(Read committed)，可重复读(Repeatable Read)，快照(Snapshot)，可串行化。

选择事务隔离级别不影响为保护数据修改而获取的锁。事务总是在其修改的任何数据上获取互斥锁并在事务完成之前持有该锁，不管为该事务设置了什么样的隔离级别。对于读取操作，事务隔离级别主要定义保护级别，以防受到其他事务所做更改的影响。

较低的隔离级别可以增强许多用户同时访问数据的能力，但也增加了用户可能遇到的并发副作用（例如脏读或丢失更新）的数量。相反，较高的隔离级别减少了用户可能遇到的并发副作用的类型，但需要更多的系统资源，并增加了一个事务阻塞其他事务的可能性。应平衡应用程序的数据完整性要求与每个隔离级别的开销，在此基础上选择相应的隔离级别。最高隔离级别（可串行化）保证事务在每次重复读取操作时都能准确检索到相同的数据，但需要通过执行某种级别的加锁来完成此操作，而加锁可能会影响多用户系统中的其他用户。最低隔离级别（未提交读）可以检索其他事务已经修改、但未提交的数据。在未提交读中，所有并发副作用都可能发生，但因为没有读取加锁或版本控制，所以开销最少。

下表显示了不同隔离级别允许的并发副作用。

|  |  |  |  |
| --- | --- | --- | --- |
| 隔离级别 | 脏读 | 不可重复读 | 幻读 |
| 未提交读/脏读 | 是 | 是 | 是 |
| 已提交读 | 否 | 是 | 是 |
| 可重复读 | 否 | 否 | 是 |
| 快照 | 否 | 否 | 否 |
| 可串行化 | 否 | 否 | 否 |

事务必须至少在可重复读的隔离级别运行，才能在两个事务都检索同一行，然后根据原先检索的值更新行时，防止出现丢失更新的情况。如果两个事务使用一个UPDATE语句更新行，而且更新并不以先前检索的值为基础，则在默认的已提交读的隔离级别上不会发生丢失更新的情况。

若要为事务设置隔离级别，可以使用[SQLServer Connection](http://technet.microsoft.com/zh-cn/library/ms378730.aspx)类的 s[etTransactionIsolation](http://technet.microsoft.com/zh-cn/library/ms378477.aspx) 方法。此方法接受int值作为其参数，此参数基于如下所示的连接常量之一：

con.set Transaction Isolation(Connection.TRANSACTION\_READ\_COMMITTED);

若要使用SQL Server新的快照隔离级别，可以使用以下所示的[SQLServer Connection](http://technet.microsoft.com/zh-cn/library/ms378730.aspx)常量之一：

1. con.setTransactionIsolation(SQLServerConnection.TRANSACTION\_SNAPSHOT);
2. con.setTransactionIsolation(Connection.TRANSACTION\_READ\_COMMITTED + 4094);

在ADO.NET的设置则是：

用System.Data.SqlClient托管命名空间的ADO.NET应用程序，可以调用SqlConnection.BeginTransaction方法，并将IsolationLevel选项设置为 Unspecified、Chaos、ReadUncommitted、ReadCommitted、RepeatableRead、Serializable或Snapshot。

代码如下：

System.Data.SqlClient.SqlConnection con = new SqlConnection();

con.BeginTransaction(IsolationLevel.ReadUncommitted);

### 实验内容与要求

#### 11.1 单事务/串行事务提交与回滚

##### 11.1.1 违反check约束的update操作

###### 实验要求：

根据现网实际情况，小区/基站工参表tbCell中的小区天线高度不能小于0。 在tbCell的备份表tbCellnew上，用Alter table add check添加约束，并在该备份表上完成以下实验内容：

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于20的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

Step2. 更新小区/基站工参表将step1中的HEIGHT设置为当前值减去15（注意此时有可能违反check约束）

Step3. 查询小区/基站工参表的小区天线高度（HEIGHT）小于20的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

针对以上操作分别进行如下的操作：

（1）将以上操作组织成普通的SQL语句，顺序执行。

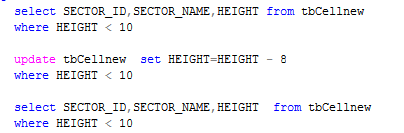
（2）将以上操作组织成事务执行（**以begin tran开始，以commit tran结束**）。

查看数据库，观察两次的执行结果有何异同。

###### 示例

**以下示例是按照“小区天线高度（HEIGHT）小于10”、“HEIGHT设置为当前值减去8”，完成的实验内容，供参考。**

11.1.1.1 顺序执行

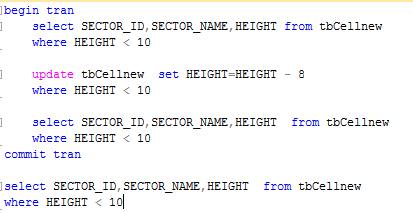


执行结果：

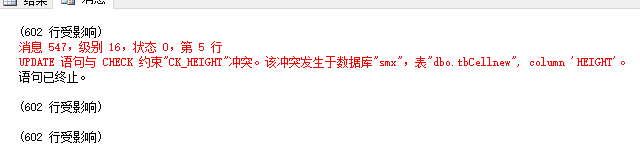


说明：update操作违反了check约束，更新失败。

11.1.1.2 组织成事务执行



执行结果：



说明：update操作违反了check约束，更新失败。

##### 11.1.2 数据表更新

###### 实验要求：

**分别以三种事务执行方式，在tbCell的备份表tbCellnew上执行以下操作，并观察、分析、解释执行结果**

Step1. 查看小区ID在'122880-0'和'122882-2'之间的小区配置的频点编号；

Step2. 将小区ID在'122880-0'和'122882-2''的小区配置的频点编号更新为37900;

Step3. 再次查看小区ID在'122880-0'和'122882-2'之间的小区配置的频点编号。

将step1、step2和step3的数据库访问组织成1个单一事务T1，再将step3作为1个独立事务，提交DBMS，串行执行这2个事务，观察T1中的rollback、commit对事务执行结果的影响。

由step1、step2和step3组成的事务T1采用以下2种结束方式：

（1）以commit结束；

（2）以rollback结束。

**说明：可以采用LTE数据库中其它的数据库表作为实验对象。**

###### 示例：

**以下示例是按照小区ID范围在'124672-0'和'124675-2'之间、更新后的频点为38950的要求，完成的实验内容，供参考。**

11.1.2.1 显示执行模式：

完整过程：

begin tran

begin try

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

update tbCellnew set EARFCN=38950

where SECTOR\_ID between '124672-0' and '124675-2'

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

end try

begin catch

select Error\_number() as ErrorNumber, --错误代码

Error\_severity() as ErrorSeverity, --错误严重级别，级别小于try catch 捕获不到

Error\_state() as ErrorState , --错误状态码

Error\_Procedure() as ErrorProcedure , --出现错误的存储过程或触发器的名称。

Error\_line() as ErrorLine, --发生错误的行号

Error\_message() as ErrorMessage --错误的具体信息

if(@@trancount>0) --全局变量@@trancount，事务开启此值+1，他用来判断是有开启事务

rollback tran ---由于出错，这里回滚到开始，第一条语句也没有插入成功。

end catch

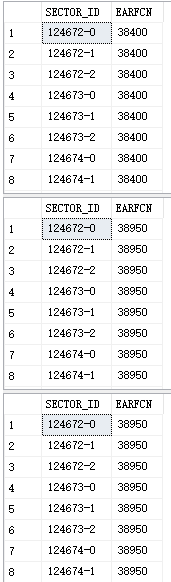
if(@@trancount>0)

commit tran

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

查看执行结果：



只执行rollback过程：

begin tran

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

update tbCellnew set EARFCN=38950

where SECTOR\_ID between '124672-0' and '124675-2'

select SECTOR\_ID,EARFCN from tbCellnew

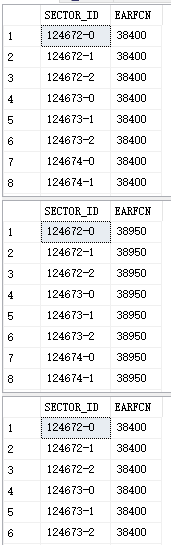
where SECTOR\_ID between '124672-0' and '124675-2'

rollback tran

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

查看执行结果：



说明：只用rollback结束事务，在事务内部结果改变，但是事务结束后结果没有改变。

而执行commit后结果发生改变并提交至数据库，不能再进行回滚。

11.1.2.2 隐式事务：

Commit提交事务：

Set IMPLICIT\_TRANSACTIONS ON

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

update tbCellnew set EARFCN=38950

where SECTOR\_ID between '124672-0' and '124675-2'

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

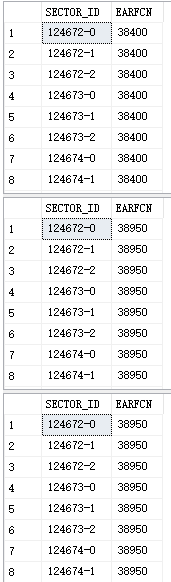
COMMIT TRANSACTION

Set IMPLICIT\_TRANSACTIONS OFF

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

查看执行结果



Rollback回滚事务：

Set IMPLICIT\_TRANSACTIONS ON

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

update tbCellnew set EARFCN=38950

where SECTOR\_ID between '124672-0' and '124675-2'

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

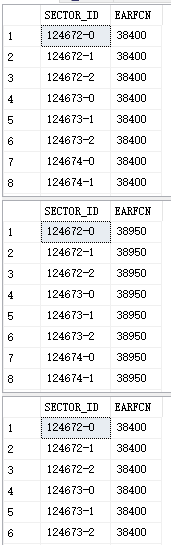
ROLLBACK TRANSACTION

Set IMPLICIT\_TRANSACTIONS OFF

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

查看执行结果：



说明：结果与显式提交事务一致。

11.1.2.3 自动提交模式：

完整过程：

select SECTOR\_ID,EARFCN from tbCellnew

where SECTOR\_ID between '124672-0' and '124675-2'

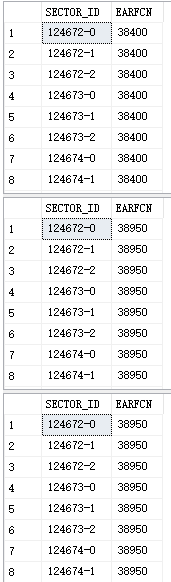
update tbCellnew set EARFCN=38950

where SECTOR\_ID between '124672-0' and '124675-2'

select SECTOR\_ID,EARFCN from tbCellnew where

SECTOR\_ID between '124672-0' and '124675-2'

查看执行结果：



说明：自动提交模式下，每个SQL语句相当于1个事务，多条顺序提交的SQL语句对应于串行执行多个事务，未出错不会发生回滚。

##### 11.1.3 数据库模式修改

###### 实验要求：

Step1. 修改TD-LTE数据库中的tbOptCell表的备份表tbOptCellnew的模式，删除列CELL\_TYPE（使用**alter table drop**）。

Step2. 修改tbOptCell表的备份表tbOptCellnew的模式，增加列CELL\_TYPE（使用**alter table add**）。

将step1、step2的数据库访问组织成1个单一事务（以显式事务的方式），采用以下2种结束方式：

（1）以commit结束；

（2）以rollback结束。

查看数据库（用select语句查看被删除/增加的列），观察数据库模式修改语句（**alter table**），是否会受到**rollback**，**commit**语句的影响。

也可以自行创建表、删除表，重复以上两步，查看数据库，观察数据库模式定义语句（**create table**）模式修改语句（**drop table**）是否会受到**rollback**，**commit**语句的影响。

###### 示例

以下示例以备份表tbPCIAssignmentnew为对象，在表中增加新属性列OPT\_DATETIME，然后再删除，供参考。

11.1.3.1 删除列OPT\_DATETIME

11.1.3.1.1 只有rollback的过程：

begin tran

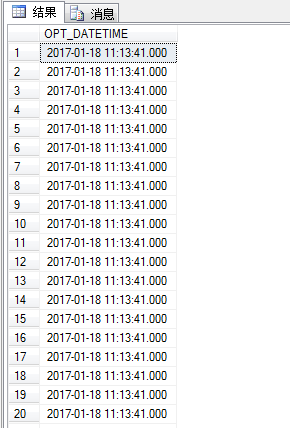
alter table tbPCIAssignmentnew

drop column OPT\_DATETIME

rollback tran

select OPT\_DATETIME from tbPCIAssignmentnew

查看执行结果：



说明：只执行rollback语句时，事务结束后没有真正提交结果，没有真正删除OPT\_DATETIME。

11.1.3.1.2 执行commit的过程：

begin tran

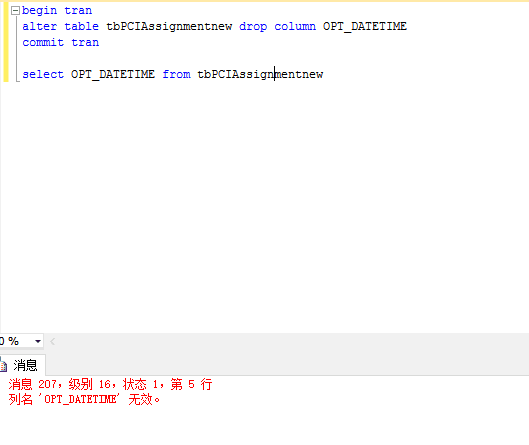
alter table tbPCIAssignmentnew

drop column OPT\_DATETIME

commit tran

select OPT\_DATETIME from tbPCIAssignmentnew

查看执行结果：



说明：执行commit语句后，成功删除OPT\_DATETIME，事务结束后结果已提交，已不能再进行回滚。

11.1.3.2 增加列OPT\_DATETIME

11.1.3.2.1 只有rollback的过程

begin tran

alter table tbPCIAssignmentnew

add OPT\_DATETIME DATETIME NULL

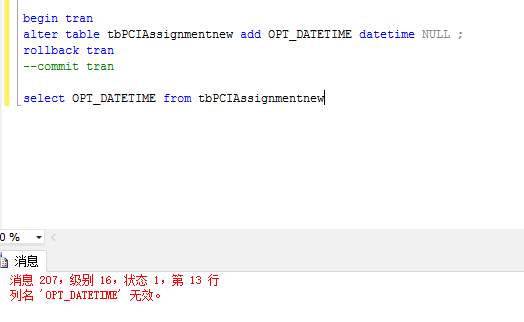
rollback tran

执行结果：



检查是否增加成功：

select OPT\_DATETIME from tbPCIAssignmentnew



说明：执行rollback语句后，事务结束后查找OPT\_DATETIME显示无效，没有真正添加进去。

11.1.3.2.2 执行commit的过程

begin tran

alter table tbPCIAssignmentnew

add OPT\_DATETIME DATETIME NULL;

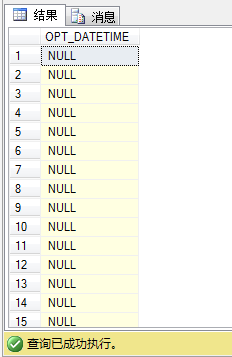
commit tran

执行结果：



检查是否增加成功：

select OPT\_DATETIME from tbPCIAssignmentnew



说明：执行commit语句后，成功添加OPT\_DATETIME，事务结束后结果已提交，已不能再进行回滚。

##### 11.1.4 多条insert/delete操作执行比较

###### 实验要求

以备份表tbCellnew为访问对象，

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

Step2. 在小区/基站工参表中，添加一条SECTOR\_ID为“211100-2”、HEIGHT为6的信息；

Step3. 删除step2所添加的信息；

Step4. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

针对以上操作分别进行如下的操作：

（1）将以上操作组织成普通的SQL语句，顺序执行。

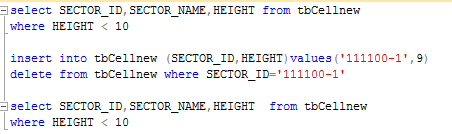
（2）将以上操作组织成事务执行（**以begin tran开始，以commit tran结束**）。

查看数据库，观察两次的执行结果有何异同。

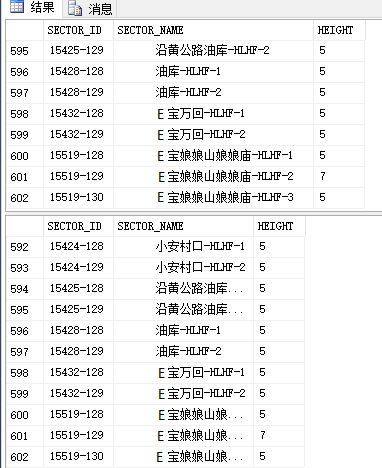
###### 示例

**以下实验查询条件为“小区天线高度（HEIGHT）小于10”，插入的SECTOR\_ID为“111100-1”，供参考。**

11.1.4.1 顺序执行



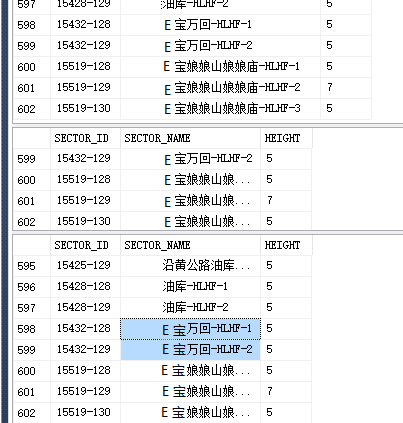
执行结果：



说明：先增加一行再删除同一行最后的结果不变，两次查询的数目相同，只是结果排列的顺序不同。

11.1.4.2 组织成事务执行

执行结果：



说明：在事务提交后再添加一条select语句查看，先增加一行再删除同一行，表中的数据没有发生改变。

##### 11.1.5 保存点Savepoint设置与回滚实验

本实验要求在事务内部不同执行位置设置，例如添加之后、添加之前、删除之后等，使用 SAVE TRANSACTION savepoint\_name 语句创建保存点,使用ROLLBACK TRANSATCTION savepoint\_name语句将事务回滚，观察每次操作的结果。保存点提供了回滚部分事务的机制，而不是回滚到事务的开始。

###### 实验要求

以小区PCI优化调整结果表tbPCIAssignment为访问对象，在创建的事务中insert插入语句后设置保存点，然后删除添加的信息，并回滚至保存点并提交事务；事务完成后再查询相应的行，观察执行结果是否插入成功，具体如下：

Step1. 查询小区PCI优化调整结果表的小区PCI为106的小区的SECTOR\_ID；

Step2. 在小区PCI优化调整结果表中，添加一条SECTOR\_ID为“220102-5”、PCI为106的信息；

Step3. 设置保存点；

Step4. 删除step2所添加的信息；

Step5. 回滚至保存点；

Step6. 事务提交结束；

Step7. 查询小区PCI优化调整结果表的小区PCI为106的小区的SECTOR\_ID；

事务结构如下：

begin tran

select语句（检查表中原始数据）

insert语句（向表中添加一行新的数据）

**save transaction ppp（设置保存点）**

delete语句（删除insert语句添加的行）

rollback transaction ppp（回滚至保存点）

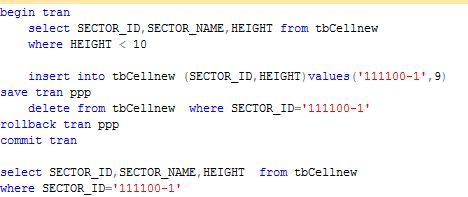
commit tran（提交事务）

select语句（检查插入数据是否成功）

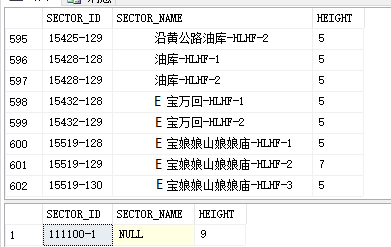
###### 示例

例如，以下事务以tbCellnew为访问对象，在插入操作之后设置了保存点：

事务语句：



执行结果：



**说明：提交前delete语句被回滚，因此最后的结果中没有执行delete语句，SECTOR\_ID为111100-1的元组被插入数据库。**

##### 11.1.6 事务局部回滚/整体回滚

###### 实验要求：

根据现网实际情况，小区PCI优化调整结果表tbPCIAssignment中的小区PCI在0到503之间。 在tbPCIAssignment的备份表tbPCIAssignmentnew上，用Alter table add check添加约束，并在该备份表上完成以下实验内容：

Step1. 在tbPCIAssignmentnew表上添加约束：加入约束check(PCI between 0 and 503)。

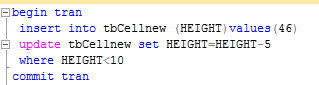
Step2. 以备份表tbPCIAssignmentnew表为访问对象，依次添加PCI为‘300’和‘600’的两条数据，将2条对备份表tbPCIAssignmentnew表进行顺序访问的insert语句组织成1个显示执行模式下的事务；

Step3. 利用SET XACT\_ABORT ON/OFF语句，控制事务执行过程中的进行整体回滚、局部回滚。观察并对比当事务执行违反约束时，事务结束后tbPCIAssignmentnew的内容，分析在整体回滚、局部回滚两种情况下，SQL Server提供的事务原子性保障机制。

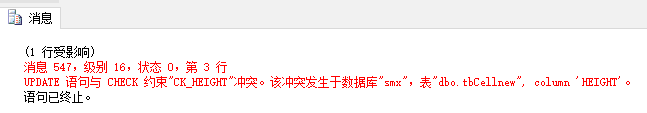
###### 示例

以下实验内容以tbCellnew为访问对象，首先添加小区天线高度（HEIGHT）为‘46’的数据，然后更新所有的小区天线高度（HEIGHT）为当前高度减去5，将2条对tbCellnew进行顺序访问的insert语句、update语句组织成1个显示执行模式下的事务。以下分别为不设置XACT\_ABORT开关和设置XACT\_ABORT开关的执行过程。

11.1.6.1 局部回滚



**查看执行结果：**



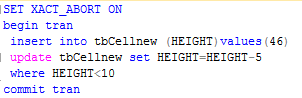
检查是否插入：

C:\Users\cqx\Documents\Tencent Files\631485932\Image\C2C\PPZ048VBJZ2)F93WE(6[VL5.png

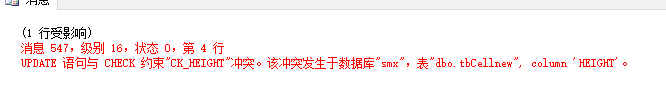


**说明：**不设置开关，然后先插入再更新，导致Height<0，违反约束，但是插入执行成功。

11.1.6.2 整体回滚

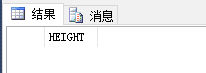


**查看执行结果：**



检查是否插入：

C:\Users\cqx\Documents\Tencent Files\631485932\Image\C2C\U_%2M3~5NEAKT4@}09EKDEM.png



**说明：**本来表中是没有HEIGHT为46的这一行的，不设置XACT\_ABORT，事务出错只回滚了更新，所以插入成功。删除插入的行，设置XACT\_ABORT为ON，事务出错全部回滚，所以插入也不成功。

#### 11.2 事务并发控制机制

本组实验涉及到事务的隔离级别、锁信息。相关知识介绍查看实验背景部分。

多个事务并发执行时，根据选择的事务隔离级别，DBMS自动使用不同粒度和不同类型的锁来保证隔离级别所要求的一致性等级，可以使用模式锁、行锁、表锁。，事务本身不需要显式请求对数据对象使用某个特定的加锁，而是由DBMS通过选择最符合要求的隔离级别来控制所维护的一致性级别。

了解各种锁类型有助于选择隔离级别和理解各个级别对性能的影响。

##### 11.2.1 查看锁信息

本实验要求观察SQL Server中有关当前活动的事务的锁管理器资源的信息。向锁管理器发出的已授予锁或正等待授予锁的每个当前活动请求分别对应一行。结果集中的列分为两组：资源组和请求组。资源组说明正在进行锁请求的资源，请求组说明锁请求。

由于结果集中的列太多，以下列举几个重要的属性与其相对应的值：

|  |  |  |
| --- | --- | --- |
| **列名** | **数据类型** | **说明** |
| resource\_type | **nvarchar(60)** | 表示资源类型。该值可以是下列值之一：DATABASE、FILE、OBJECT、PAGE、KEY、EXTENT、RID、APPLICATION、METADATA、HOBT 或 ALLOCATION\_UNIT。 |
| **resource\_database\_id** | **int** | 此资源位于其范围之内的数据库的ID。由锁管理器处理的所有资源均按该数据库ID划分范围 |
| resource\_description | nvarchar(256) | 资源的说明，其中只包含从其他资源列中无法获取的信息。 |
| request\_mode | nvarchar(60) | 请求的模式。对于已授予的请求，为已授予模式；对于等待请求，为正在请求的模式。 |
| request\_type | nvarchar(60) | 请求类型。该值为 LOCK。 |
| request\_status | nvarchar(60) | 该请求的当前状态。可能值为GRANTED、CONVERT或WAIT。 |
| request\_session\_id | int | 当前拥有该请求的会话 ID。对于分布式事务和绑定事务，拥有请求的会话 ID 可能不同。该值为 -2 时，指示该请求属于孤立的分布式事务。该值为 -3 时，指示请求属于延迟的恢复事务，例如因其回滚未能成功完成而延迟恢复该回滚的事务。 |

###### 实验要求

在SQL Server的默认隔离级别read committed下，从元数据表sys.dm\_tran\_locks中，查看由select/update/insert/delete等SQL访问语句组成的事务的锁信息。

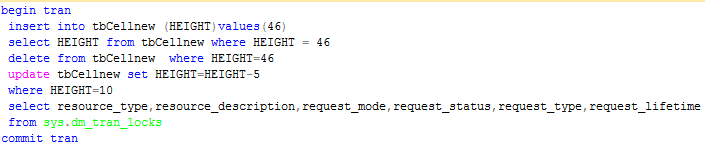
可自行选择TD-LTE数据库中的一张表（tbCell/tbCellnew除外）作为事务的访问对象，要求事务中至少包含两种不同类型SQL访问语句。

可用前面实验中编写的事务语句。并根据相应的参数作简要分析。

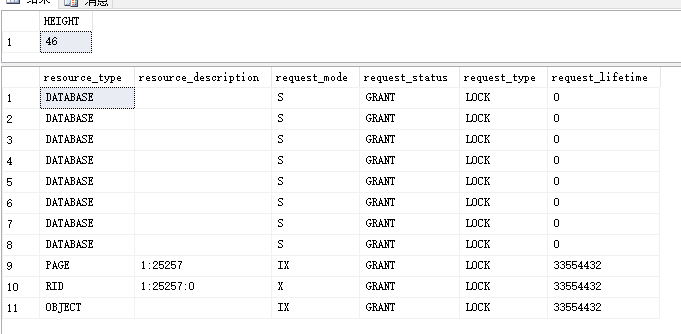
**注意**：查看锁信息语句的作用域为当前连接，为了更好的查看当前事务的锁信息，可将相应的语句写在commit语句之前。

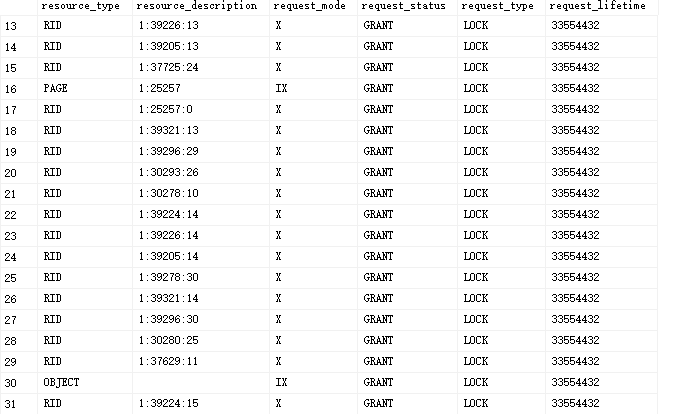
###### 示例

在tbCell的备份表tbCellnew表上进行如下操作：



执行结果（主要看锁信息相关）：





说明：

（1）从上图可看出不同粒度数据对象上的加锁锁模式：

DATABASE施加了S（共享）锁，页PAGE和表OBJECT施加了意向互斥锁，表中的行RID施加了互斥锁X。

（2）在resource\_description部分，不是所有的锁粒度都支持，其中PAGE的取值表示：<file\_id>:<page\_in\_file>，RID的取值表示：<file\_id>:<page\_in\_file>:<row\_on\_page>，对应与此例中16行的锁是加在file\_id为1的page\_id为25257的意向互斥锁。

（3）查看数据库内部的存储信息，可用DBCC IND和DBCC PAGE，具体可参考：

<http://www.cnblogs.com/woodytu/p/4484328.html>

##### 11.2.2 单事务隔离级别及加锁信息和执行结果观察

###### 实验要求

比较不同隔离级别下，事务加锁的粒度、锁类型。

Step1. 以LTE数据库中某个关系表为对象，设计1个由多条SQL访问语句（增、删、改、查）组成的显示执行事务；

Step2. 使用DBCC USEROPTIONS查看当前的隔离级别和锁信息；然后设置不同的隔离级别，查看对比在不同的隔离级别的并发副作用；

Step3. 分析对比不同隔离级别下，事务的执行结果差异。

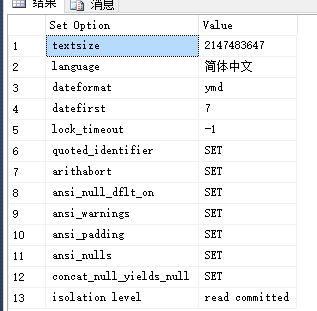
###### 示例

11.2.2.1 查看系统默认的当前隔离级别

在当前会话执行下面语句：

DBCC USEROPTIONS

查看执行结果：



说明：最后一行表示隔离级别，为“读提交”。

11.2.2.2 设置不同的隔离级别并查看并发副作用

11.2.2.2.1 未提交的相关性（脏读）

对2个并发事务，当第2个事务选择访问第1个事务正在更新的行时，会发生未提交的相关性问题。第2个事务正在读取的数据还没有确认提交，并且之后可能由正在更新此行的第1个事务所更改。

实验要求

1. 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
2. 在会话一中将隔离级别设置为未提交读read uncommitted，以tbCellnew为访问对象，查询SECTOR\_ID为‘124686-2’的小区的SECTOR\_NAME和PCI；
3. 在会话二中开始事务，更新SECTOR\_ID为‘124686-2’的小区的PCI为当前PCI值加20，保持未提交状态执行语句；
4. 在会话一中将隔离级别设置为未提交读，以tbCellnew为访问对象，查询SECTOR\_ID为‘124686-2’的小区的SECTOR\_NAME和PCI；
5. 消除并发副作用。

示例

步骤1：会话1：

执行：

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\IS[GB}QU]BH60ES2OWXRL}X.png

结果：

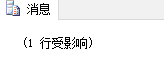


步骤2：会话2：

执行：（注意此时事务未提交）

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\6@UW{DD2~IQ~YH8~IFT0`KL.png

结果：



步骤3：再次回到会话1：

执行：

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\8{2261[GJV89YZ6OH~$XGGG.png

结果：

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\9K0ELDC~8FZ9_B{UA7$$DFF.png

出现脏读

步骤4：消除副作用

不能简单的把会话1的隔离级别设置为高于“未提交读”的隔离级别，这样会出现死锁，因为会话2的事务未提交，会话1要等提交完之后再读数据，就会一直等待。

为消除脏读，可以直接在会话2用commit或rollback结束事务即可。

11.2.2.2.2 不一致的分析（不可重复读）

当第2个事务多次访问表中同一行数据而且每次读取不同内容时，会发生不一致的分析问题。不一致的分析与未提交的相关性类似，因为其它事务也是正在更改第二个事务正在读取的数据。然而，在不一致的分析中，第二个事务读取的数据是由已进行了更改的事务提交的。而且，不一致的分析涉及多次（两次或更多）读取同一行，而且每次信息都由其它事务更改；因而该行被非重复读取。

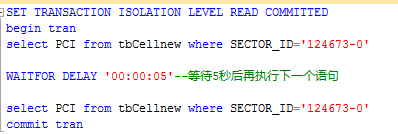
实验要求

1. 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
2. 在会话一中将隔离级别设置为提交读或者未提交读，以tbCellnew为访问对象，查询SECTOR\_ID为‘124694-0’的小区的SECTOR\_NAME和PCI；设置推迟等待5秒语句，然后再写一条查询语句，同样是查询SECTOR\_ID为‘124694-0’的小区的SECTOR\_NAME和PCI，执行语句后的五秒内开始第三步；
3. 第二步语句执行的五秒内在会话二中执行语句，开始事务，更新SECTOR\_ID为‘124694-0’的小区的PCI为400，并提交事务；
4. 在会话一中查看执行结果；
5. 消除并发副作用。

示例

步骤1：会话1:

执行：（将隔离级别设置为READ UNCOMMITTED和READ COMMITTED都可以）

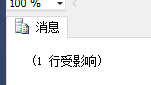


步骤2：步骤1执行后马上切换到会话2执行以下语句：

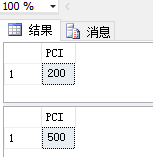


执行结果：

会话2：



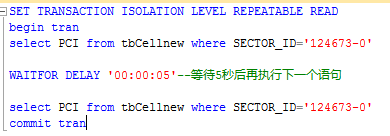
会话1：



两次结果不一致，出现“不可重复读”。

步骤3：消除副作用

将会话1的隔离级别设置为REPEATABLE READ或SERIALIZABLE



执行完会话1马上执行会话2



会话1结果：



此时前后结果一致，为更新前的数据，因为查询语句在更新语句之前完成。

11.2.2.2.3 幻读

当事务1对表中某行执行插入或删除操作时，如果该行属于某个事务2正在读取的行的范围内，会发生幻读问题。事务2第一次读的行范围显示出其中一行已不复存在于第二次读或后续读中，因为该行已被其它事务1删除。同样，由于其它事务1的插入操作，事务2的第二次或后续读显示有一行已不存在于原始读中。

实验要求

（1） 开启两个连接，模拟两个并行的事务。简称会话一和会话二；

（2） 在会话一中将隔离级别设置为READ UNCOMMITTED、READ COMMITTED或REPEATABLE READ，以tbCellnew为访问对象，查询LONGITUDE小于110.37644的小区的LONGITUDE并去重；设置推迟等待5秒语句，然后再写一条查询语句，同样是查询LONGITUDE小于110.37644的小区的LONGITUDE并去重；

（3） 第二步语句执行的五秒内在会话二中执行语句，开始事务，添加LONGITUDE为110.366666的信息，并提交事务；

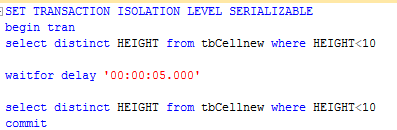
（4） 查看会话一执行结果；

（5） 消除并发副作用。

示例

步骤1：在会话1中执行：

（将隔离级别设置为READ UNCOMMITTED、READ COMMITTED和REPEATABLE READ都可以）

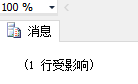


步骤2：步骤1执行后马上切换到会话2执行以下语句：



执行结果：

会话2：



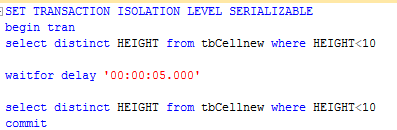
会话1：



同一个事务中两次查询返回的结果记录数不同，出现了“幻读”。

步骤3：消除副作用

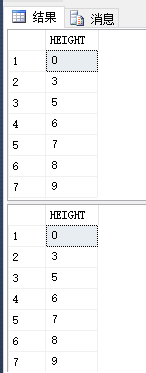
将会话1的隔离级别设置为SERIALIZABLE



执行完会话1之后马上执行会话2：



会话1结果：



同一事务中前后两次查询结果记录数相同，为插入之前的数据。

#### 11.3 事务日志观察

使用下列SQL语句查看所在数据库的事务日志记录

SELECT \* FROM [sys].[fn\_dblog](NULL,NULL)

在事务日志记录中，Operation列记录了对应的LSN所做的操作，其中LOP\_BEGIN\_XACT表示一个事务的开始，Transaction Name显示了创建的事务的名称，而Trasaction ID则记录了所对应的事务ID。下面列出Operation几种比较常见而重要的值：

* LOP\_BEGIN\_XACT 事务的开始
* LOP\_LOCK\_XACT 获取锁
* LOP\_MODIFY\_ROW 修改行(具体修改的对象可以查看AllocUnitName)
* LOP\_COMMIT\_XACT 提交事务
* LOP\_DELETE\_ROWS 删除数据
* LOP\_INSERT\_ROWS 插入数据
* LOP\_ABORT\_XACT 事务回滚
* Checkpoint Begin/CheckPoint End事务日志开始、结束

##### 11.3.1 单条SQL语句（select/update/insert/delete）在自动提交模式下的日志观察

###### 实验要求

以备份表tbCellnew为访问对象，

Step1. 查询小区/基站工参表的SECTOR\_ID为‘124674-1’的小区的基站类型；

Step2. 修改小区/基站工参表的SECTOR\_ID为’ 124674-1’的小区的基站类型为‘室分’；

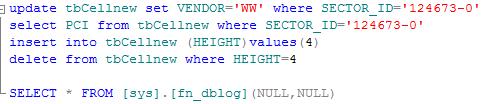
Step3. 在小区/基站工参表中，添加一条SECTOR\_ID为“211100-2”、HEIGHT为6的信息；

Step4. 删除step3所添加的信息；

Step5. 查看事务日志，观察operation中具体的操作内容、事务日志记录中update语句修改前和修改后的数据值，并找出一对checkpoint begin和checkpoint end的值；观察各个属性，理解每个属性的含义；

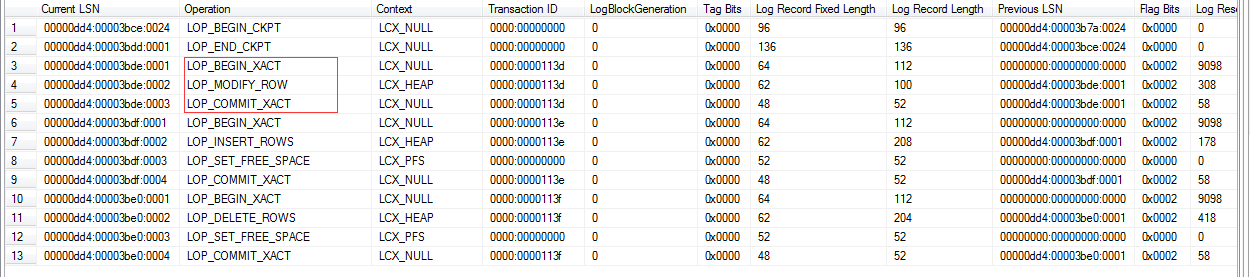
###### 示例

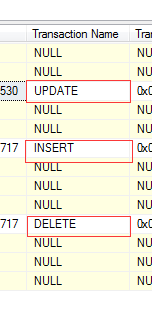
执行过程：



执行结果：

(1) LOP\_BEGIN\_XACT表示一个事务的开始，对应于<T,start>,LOP\_COMMIT\_XACT表示事务的提交，对应于<T,commit>；以update为例，从第3行开始，第5行结束。



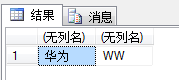
(2) 与Operaiton对应的Transanction Name，比如LOP\_MODIFTY\_ROW对应于UPDATE； 

（3）修改前数据是[RowLog Contents 0]字段内容，如果是UPDATE操作，修改后数据存放在[RowLog Contents 1]字段内单条语句下，如下图。UPDATE操作的事务日志是一种<T,V,x1,x2>类型的。

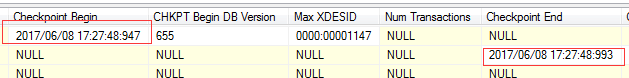


读取两个字段代表的数据：





（4）事务日志中可能有多对checkpoint begin和checkpoint end，在checkpoint end对应的时间将缓存的数据写到磁盘上，对应于<Checkpoint，L>。



##### 11.3.2 显示执行模式下，多条语句组成的单个事务的日志的观察

###### 11.3.2.1 以commit结束的事务日志实验

实验要求

以备份表tbCellnew为访问对象，

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID；

Step2. 修改小区/基站工参表的SECTOR\_ID为’124678-2’的小区的高度为’10’；

Step3. 在小区/基站工参表中，添加一条SECTOR\_ID为“211100-2”、HEIGHT为6的信息；

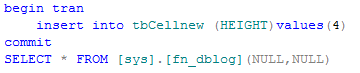
Step4. 删除step3所添加的信息；

Step5. 将以上操作组织成事务执行（**以begin tran开始，以commit tran结束**）

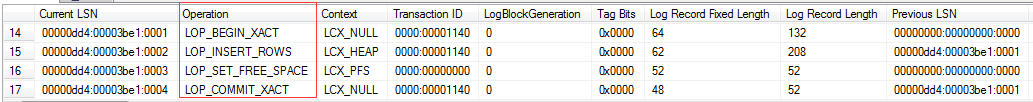
Step6. 查看事务日志，观察operation中具体的操作内容，并观察各个属性，理解每个属性的含义；

示例

执行过程：



执行结果：



说明：此事务从第14行开始，17行结束。

###### 11.3.2.2 以rollback结束的事务日志实验

实验要求

以备份表tbCellnew为访问对象，

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID；

Step2. 修改小区/基站工参表的SECTOR\_ID为’124678-2’的小区的高度为’10’；

Step3. 在小区/基站工参表中，添加一条SECTOR\_ID为“211100-2”、HEIGHT为6的信息；

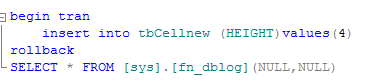
Step4. 删除step3所添加的信息；

Step5. 将以上操作组织成事务执行（**以begin tran开始，以rollback tran结束**）

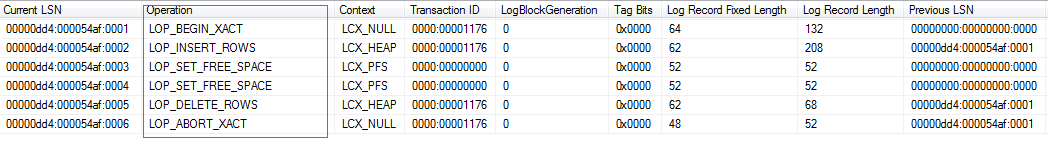
Step6. 查看事务日志，观察operation中具体的操作内容，并观察各个属性，理解每个属性的含义；

示例

执行过程：



执行结果：



说明：因为以rollback结束事务，此事务以LOP\_ABORT\_XACT结束，对应于<T,abort>，

###### 11.3.2.3 事务发生中断的事务日志实验

实验要求

以备份表tbCellnew为访问对象，

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID；

Step2. 修改小区/基站工参表的SECTOR\_ID为’124678-2’的小区的高度为’10’；

Step3. 添加一个延时2~3分钟；

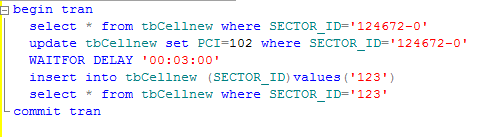
Step4. 在小区/基站工参表中，添加一条SECTOR\_ID为“211100-2”、HEIGHT为6的信息删除step3所添加的信息

Step5. 将以上操作组织成事务执行（**以begin tran开始，以commit tran结束**），要求在延时的时间范围内发生一个外部冲突（实验时可启动任务管理器结束SQLServer进程）；

Step6. 重新连接数据库，并查看事务日志。

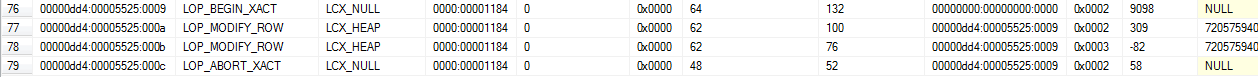
示例

执行过程：



在延时期间发生外部中断；

重新连接SQLServer，查看事务日志：



检查更新前后的PCI值：

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\T{@Z}J$1]4M(6E258V3C4}Y.png

C:\Users\lyz\Documents\Tencent Files\985106592\Image\C2C\9{Q7EA7X%%RR55_6]A~$B8Q.png

说明：在事务的执行过程中，如果发生外部中断，会产生<T,abort>型的事务日志，事务会被回滚，事务内的各项更新不成功。

##### 11.3.3 显示执行模式下，多条或单条语句组成的多个并发事务的日子观察

###### 实验要求

以备份表tbCellnew为访问对象，

step1. 开启两个连接，模拟两个并行的事务。简称会话一和会话二；

step2. 在会话一中将隔离级别设置为提交读或者未提交读，以tbCellnew为访问对象，查询SECTOR\_ID为‘124694-0’的小区的SECTOR\_NAME和PCI；设置推迟等待5秒语句，然后再写一条查询语句，同样是查询SECTOR\_ID为‘124694-0’的小区的SECTOR\_NAME和PCI，执行语句后的五秒内开始第三步；

step3. 第二步语句执行的五秒内在会话二中执行语句，开始事务，更新SECTOR\_ID为‘124694-0’的小区的PCI为400，并提交事务；

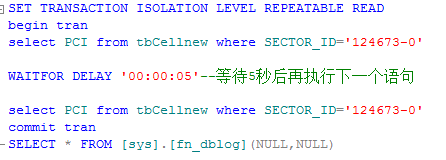
step4. 在会话一中查看执行结果；

step5. 查看事务日志，观察operation中具体的操作内容，观察并观察各个属性，理解每个属性的含义；

###### 示例

执行过程：

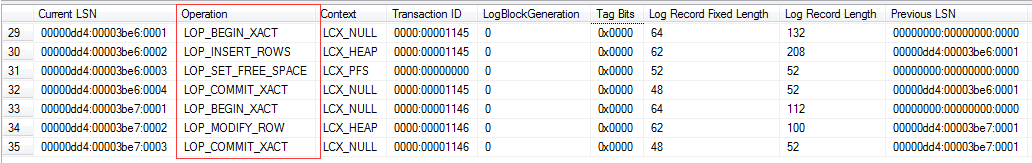
会话一：



会话二：



执行结果：



说明：在并发事务里，第29行到32行是会话一的事务，第33行到35行是会话二的事务。

### 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# 附录一 实验平台及T-SQL介绍

## 一、数据库系统实验平台简介（略）

## 二、T-SQL简介

SQL语言是1974年Boyce和Chamberlin提出的，1975年IBM公司研制的关系数据库管理系统System R实现了这种语言。因为其功能强大、使用方便，获得用户及业界青睐，经过多年发展，SQL已成为关系数据库的标准语言。

SQL 语言是一种介于关系代数与关系演算之间的语言，其功能包括查询、操纵、定义和控制4 个方面，是一个通用的、功能极强的关系数据库语言。

Transact-SQL（简称为T-SQL）是ANSI-SQL的加强版语言，它包括基本的SQL命令，还有许多扩充，比如拥有类似于Pascal或者C等第三代语言的程序结构。**T-SQL语言是Microsoft SQL SERVER、DB2或Sybase数据库使用的基本语言。**

虽然T-SQL拥有部分高级语言的处理功能，但这些功能主要是用于处理数据库中的数据的，因此其它功能仍然需要其它应用程序开发工具来协助完成。因此我们将DB2或Sybase数据库称为数据库引擎。

SQL语言的组成为：

数据定义语言（DDL，Data Definition Language）；

数据操纵语言（DML，Data Manipulation Language）；

数据控制语言（DCL，Data Control Language）；

其它语言要素（Additional Language Elements）。

SQL 语句数目、种类较多、其主体大约由40 条语句组成，如下表所示：

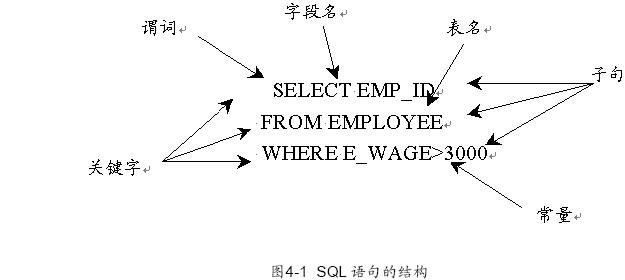




SQL 语句的结构：

 所有的SQL 语句均有自己的格式，如上表所示每条SQL 语句均由一个谓词（Verb）

开始，该谓词描述这条语句要产生的动作，如下表中的SELECT 关键字。谓词后紧接着一个或多个子句（Clause）子句中给出了被谓词作用的数据或提供谓词动作的详细信息。每一条子句由一个关键字开始，如下表中的WHERE。



常用的SQL 语句  
     在使用数据库时用得最多的是数据操纵语言（Data Manipulation Language）。 DML包含了最常用的核心SQL 语句，即SELECT 、INSERT 、UPDATE 、DELETE。 下面对以后章节中经常用到的SELECT 语句作简单介绍。

简单的SELECT 语句的语法如下：  
      **SELECT [ALL|DISTINCT] <目标表达式>[，<目标表达式>]...**  
      **FROM <表或视图名>[, <表或视图名>]...**  
      **[WHERE <条件表达式>]**  
      **[GROUP BY <列名1> [HAVING <条件表达式>]]**  
      **[ORDER BY <列名2> [ASC | DESC] ]**  
    整个SELECT 语句的含义是:根据WHERE 子句的条件表达式,从FROM 子句指定的基本表或视图中找出满足条件的元素组,再按SELECT 子句中的目标列表达式选出元素组中的属性值形成结果表。如果有GROUP 子句，则将结果按<列名1>的值进行分组，该属性列值相等的元素组为一个组，每个组产生结果表中的一条记录。如果GROUP 子句带有HAVING 短语，则只有满足指定条件的组才予以输出。如果有ORDER 子句，则结果表还要按<列2>的值升序或降序排序。下面对SELECT 语句的常用形式举例说明。

**例1：查询所有员工的员工号和工资**  
use pangu  
select emp\_id，e\_wage  
from employee  
  
运行结果如下：  
emp\_id e\_wage  
-------- ---------------------  
10010001 8000.0000  
......（因数据太多故省略之）

**例2：查询表的全部数据**  
use pangu  
select \*  
from employee  
  
运行结果如下  
emp\_id e\_name birthday job\_level dept\_id hire\_date e\_wage  
-------- -------------------- --------------------------- --------- ------- ---------------------------  
10010001 张三 1968-02-14 00:00:00.000 1 1001 1996-08-02 00:00:00.000 8000.0000  
......  
  
    **例3：查询工作级别为"2"的员工姓名，查询结果按部门分组**  
use pangu  
select e\_name， dept\_id  
from employee  
where job\_level = '2'  
group by dept\_id e\_name

运行结果如下：  
e\_name dept\_id  
-------------------- -------  
李四 1001  
......

Transact-SQL语言的分类：

Transact-SQL 语言中标准的SQL 语句畅通无阻.Transact-SQL 也有类似于SQL 语言的分类，不过做了许多扩充。Transact-SQL 语言的分类如下：

·变量说明： 用来说明变量的命令.

·数据定义语言(DDL,Data Definition Language)： 用来建立数据库、数据库对象和定义其列，大部分是以CREATE 开头的命令，如：CREATE TABLE 、CREATE VIEW 、DROP TABLE 等。

·数据操纵语言（DML，Data Manipulation Language）： 用来操纵数据库中的数据的命令，如：SELECT、 INSERT、 UPDATE 、DELETE 、CURSOR等。

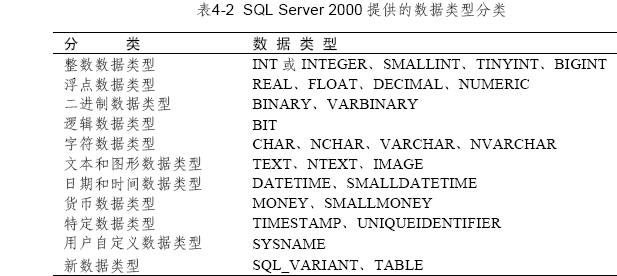
·数据控制语言（DCL， Data Control Language）： 用来控制数据库组件的存取许可、存取权限等的命令。如GRANT、 REVOKE 等。

流程控制语言（Flow Control Language）：用于设计应用程序的语句如IF WHILE CASE 等。

内嵌函数：一种用户定义函数。

其它命令：嵌于命令中使用的标准函数。

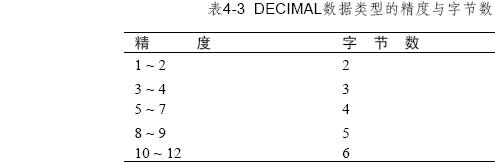
## 三、T-SQL数据类型

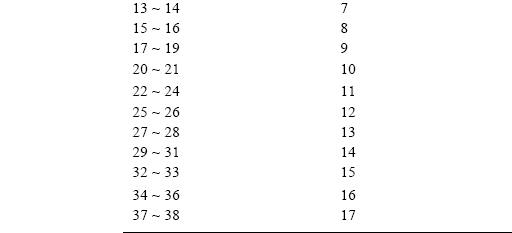
在计算机中数据的特征包括：类型和长度。所谓数据类型就是以数据的表现方式和存储方式来划分的数据的种类。  
    在DB2或Sybase数据库 中每个变量、参数、表达式等都有数据类型。系统提供的数据类型分为几大类，如下表所示。

其中，BIGINT、 SQL\_VARIANT 和TABLE 是DB2或Sybase数据库 2000 中新增加的3 种数据类型。下面分类讲述各种数据类型。

一）整数数据类型   
    整数数据类型是最常用的数据类型之一。  
     1、INT （INTEGER）  
INT （或INTEGER）数据类型存储从-2的31次方 （-2 ，147 ，483 ，648） 到2的31次方-1 （2 ，147 ，483，647） 之间的所有正负整数。每个INT 类型的数据按4 个字节存储，其中1 位表示整数值的正负号，其它31 位表示整数值的长度和大小。   
     2、SMALLINT  
SMALLINT 数据类型存储从-2的15次方（ -32， 768） 到2的15次方-1（ 32 ，767 ）之间的所有正负整数。每个SMALLINT 类型的数据占用2 个字节的存储空间，其中1 位表示整数值的正负号，其它15 位表示整数值的长度和大小。  
     3、TINYINT  
TINYINT数据类型存储从0 到255 之间的所有正整数。每个TINYINT类型的数据占用1 个字节的存储空间。  
     4、BIGINT  
BIGINT 数据类型存储从-2^63 （-9 ，223， 372， 036， 854， 775， 807） 到2^63-1（ 9， 223， 372， 036 ，854 ，775， 807） 之间的所有正负整数。每个BIGINT 类型的数据占用8个字节的存储空间。

二）浮点数据类型  
    浮点数据类型用于存储十进制小数。浮点数值的数据在DB2或Sybase数据库 中采用上舍入（Round up 或称为只入不舍）方式进行存储。所谓上舍入是指，当（且仅当）要舍入的数是一个非零数时，对其保留数字部分的最低有效位上的数值加1 ，并进行必要的进位。若一个数是上舍入数，其绝对值不会减少。如：对3.14159265358979 分别进行2 位和12位舍入，结果为3.15 和3.141592653590。  
     1、REAL 数据类型  
REAL数据类型可精确到第7 位小数，其范围为从-3.40E -38 到3.40E +38。 每个REAL类型的数据占用4 个字节的存储空间。  
     2、FLOAT  
FLOAT数据类型可精确到第15 位小数，其范围为从-1.79E -308 到1.79E +308。 每个FLOAT 类型的数据占用8 个字节的存储空间。FLOAT数据类型可写为FLOAT[ n ]的形式。n 指定FLOAT 数据的精度。n 为1到15 之间的整数值。当n 取1 到7 时，实际上是定义了一个REAL 类型的数据，系统用4 个字节存储它；当n 取8 到15 时，系统认为其是FLOAT 类型，用8 个字节存储它。  
     3、DECIMAL  
DECIMAL数据类型可以提供小数所需要的实际存储空间，但也有一定的限制，您可以用2 到17 个字节来存储从-10的38次方-1 到10的38次方-1 之间的数值。可将其写为DECIMAL[ p [s] ]的形式，p 和s 确定了精确的比例和数位。其中p 表示可供存储的值的总位数（不包括小数点），缺省值为18； s 表示小数点后的位数，缺省值为0。 例如：decimal （15 5），表示共有15 位数，其中整数10 位，小数5。 位表4-3 列出了各精确度所需的字节数之间的关系。





　 4、NUMERIC

NUMERIC数据类型与DECIMAL数据类型完全相同。

**注意：**DB2或Sybase数据库 为了和前端的开发工具配合，其所支持的数据精度默认最大为28位。但可以通过使用命令来执行sqlserver.exe程序以启动DB2或Sybase数据库，可改变默认精度。命令语法如下：SQLSERVR[/D master\_device\_path][/P precisim\_leve1]  
    **例4: 用最大数据精度38 启动DB2或Sybase数据库**  
sqlservr /d c:\ Mssql2000\data\master.dat /p38  
/\*在使用了/P 参数后,如果其后没有指定具体的精度数值,则默认为38 位. \*/

三） 二进制数据类型  
     1、BINARY

BINARY 数据类型用于存储二进制数据。其定义形式为BINARY（ n）， n 表示数据的长度，取值为1 到8000 。在使用时必须指定BINARY 类型数据的大小，至少应为1 个字节。BINARY 类型数据占用n+4 个字节的存储空间。在输入数据时必须在数据前加上字符“0X” 作为二进制标识，如：要输入“abc ”则应输入“0xabc ”。若输入的数据过长将会截掉其超出部分。若输入的数据位数为奇数，则会在起始符号“0X ”后添加一个0，如上述的“0xabc ”会被系统自动变为“0x0abc”。   
     2、VARBINARY

VARBINARY数据类型的定义形式为VARBINARY（n）。 它与BINARY 类型相似，n 的取值也为1 到8000， 若输入的数据过长，将会截掉其超出部分。不同的是VARBINARY数据类型具有变动长度的特性，因为VARBINARY数据类型的存储长度为实际数值长度+4个字节。当BINARY数据类型允许NULL 值时，将被视为VARBINARY数据类型。  
一般情况下，由于BINARY 数据类型长度固定，因此它比VARBINARY 类型的处理速度快。

四）逻辑数据类型   
    BIT： BIT数据类型占用1 个字节的存储空间，其值为0 或1 。如果输入0 或1 以外的值，将被视为1。 BIT 类型不能定义为NULL 值（所谓NULL 值是指空值或无意义的值）。

五） 字符数据类型  
    字符数据类型是使用最多的数据类型。它可以用来存储各种字母、数字符号、特殊符号。一般情况下，使用字符类型数据时须在其前后加上单引号’或双引号” 。   
     1 CHAR  
CHAR 数据类型的定义形式为CHAR[ （n） ]。 以CHAR 类型存储的每个字符和符号占一个字节的存储空间。n 表示所有字符所占的存储空间，n 的取值为1 到8000， 即可容纳8000 个ANSI 字符。若不指定n 值，则系统默认值为1。 若输入数据的字符数小于n，则系统自动在其后添加空格来填满设定好的空间。若输入的数据过长，将会截掉其超出部分。  
     2、NCHAR  
NCHAR数据类型的定义形式为NCHAR[ （n） ]。 它与CHAR 类型相似。不同的是NCHAR数据类型n 的取值为1 到4000。 因为NCHAR 类型采用UNICODE 标准字符集（CharacterSet）。 UNICODE 标准规定每个字符占用两个字节的存储空间，所以它比非UNICODE 标准的数据类型多占用一倍的存储空间。使用UNICODE 标准的好处是因其使用两个字节做存储单位，其一个存储单位的容纳量就大大增加了，可以将全世界的语言文字都囊括在内，在一个数据列中就可以同时出现中文、英文、法文、德文等，而不会出现编码冲突。  
     3、VARCHAR  
VARCHAR数据类型的定义形式为VARCHAR [ （n） ]。 它与CHAR 类型相似，n 的取值也为1 到8000， 若输入的数据过长，将会截掉其超出部分。不同的是，VARCHAR数据类型具有变动长度的特性，因为VARCHAR数据类型的存储长度为实际数值长度，若输入数据的字符数小于n ，则系统不会在其后添加空格来填满设定好的空间。  
一般情况下，由于CHAR 数据类型长度固定，因此它比VARCHAR 类型的处理速度快。  
     4、NVARCHAR  
NVARCHAR数据类型的定义形式为NVARCHAR[ （n） ]。 它与VARCHAR 类型相似。不同的是，NVARCHAR数据类型采用UNICODE 标准字符集（Character Set）， n 的取值为1 到4000。

六） 文本和图形数据类型  
    这类数据类型用于存储大量的字符或二进制数据。  
     1、TEXT  
TEXT数据类型用于存储大量文本数据，其容量理论上为1 到2的31次方-1 （2， 147， 483， 647）个字节，在实际应用时需要视硬盘的存储空间而定。  
DB2或Sybase数据库 2000 以前的版本中，数据库中一个TEXT 对象存储的实际上是一个指针，它指向一个个以8KB （8192 个字节）为单位的数据页（Data Page）。 这些数据页是动态增加并被逻辑链接起来的。在DB2或Sybase数据库 2000 中，则将TEXT 和IMAGE 类型的数据直接存放到表的数据行中，而不是存放到不同的数据页中。 这就减少了用于存储TEXT 和IMA- GE 类型的空间，并相应减少了磁盘处理这类数据的I/O 数量。  
    2 NTEXT  
NTEXT数据类型与TEXT.类型相似不同的,是NTEXT 类型采用UNICODE 标准字符集(Character Set), 因此其理论容量为230-1(1, 073, 741, 823)个字节。  
    3 IMAGE  
IMAGE数据类型用于存储大量的二进制数据Binary Data。 其理论容量为2的31次方-1(2,147,483,647)个字节。其存储数据的模式与TEXT 数据类型相同。通常用来存储图形等OLE Object Linking and Embedding，对象连接和嵌入）对象。在输入数据时同BINARY数据类型一样，必须在数据前加上字符“0X”作为二进制标识

七） 日期和时间数据类型  
    1 DATETIME  
DATETIME 数据类型用于存储日期和时间的结合体。它可以存储从公元1753 年1 月1 日零时起到公元9999 年12 月31 日23 时59 分59 秒之间的所有日期和时间，其精确度可达三百分之一秒，即3.33 毫秒。DATETIME 数据类型所占用的存储空间为8 个字节。其中前4 个字节用于存储1900 年1 月1 日以前或以后的天数，数值分正负，正数表示在此日期之后的日期，负数表示在此日期之前的日期。后4 个字节用于存储从此日零时起所指定的时间经过的毫秒数。如果在输入数据时省略了时间部分，则系统将12:00:00:000AM作为时间缺省值：如果省略了日期部分，则系统将1900 年1 月1 日作为日期缺省值。  
    2 SMALLDATETIME  
SMALLDATETIME 数据类型与DATETIME 数据类型相似，但其日期时间范围较小，为从1900 年1 月1 日到2079 年6 月6：日精度较低，只能精确到分钟，其分钟个位上为根据秒数四舍五入的值,即以30 秒为界四舍五入。如：DATETIME 时间为14:38:30.283  
时SMALLDATETIME 认为是14:39:00 SMALLDATETIME 数据类型使用4 个字节存储数据。其中前2 个字节存储从基础日期1900 年1 月1 日以来的天数，后两个字节存储此日零时起所指定的时间经过的分钟数。  
    下面介绍日期和时间的输入格式  
    日期输入格式  
    日期的输入格式很多大致可分为三类：

* 英文+数字格式  
  此类格式中月份可用英文全名或缩写，且不区分大小写；年和月日之间可不用逗号；  
  年份可为4 位或2 位；当其为两位时，若值小于50 则视为20xx 年，若大于或等于50 则  
  视为19xx 年；若日部分省略，则视为当月的1号。以下格式均为正确的日期格式：  
  June 21 2000 Oct 1 1999 January 2000 2000 February  
  2000 May 1 2000 1 Sep 99 June July 00
* 数字+分隔符格式  
  允许将斜杠（/）、连接符（-）和小数点（.）作为用数字表示的年、月、日之间的分  
  隔符。如：  
  YMD：2000/6/22 2000-6-22 2000.6.22  
  MDY：3/5/2000 3-5-2000 3.5.2000  
  DMY：31/12/1999 31-12-1999 31.12.2000
* 纯数字格式  
  纯数字格式是以连续的4 位6、位或8 位数字来表示日期。如果输入的是6 位或8 位  
  数字，系统将按年、月、日来识别，即YMD 格式，并且月和日都是用两位数字来表示：  
  如果输入的数字是4 位数，系统认为这4 位数代表年份,其月份和日缺省为此年度的1 月  
  1 日。如:  
  20000601---2000 年6 月1 日 991212---1999 年12 月12 日 1998---1998 年
* 时间输入格式  
  在输入时间时必须按“小时、分钟、秒、毫秒”的顺序来输入。在其间用冒号“：”隔开。但可将毫秒部分用小数点“.” 分，隔其后第一位数字代表十分之一秒，第二位数字代表百分之一秒，第三位数字代表千分之一秒。当使用12 小时制时用AM。am 和PM（pm）分别指定时间是午前或午后，若不指定，系统默认为AM。AM 与PM 均不区分大小写。如：  
      3:5:7.2pm---下午3 时5 分7 秒200 毫秒  
      10:23:5.123Am---上午10 时23 分5 秒123 毫秒  
      可以使用SET DATEFORMAT 命令来设定系统默认的日期-时间格式。

八）货币数据类型

货币数据类型用于存储货币值。在使用货币数据类型时，应在数据前加上货币符号，系统才能辨识其为哪国的货币，如果不加货币符号，则默认为“￥”。  
    1 MONEY  
MONEY 数据类型的数据是一个有4 位小数的DECIMAL 值，其取值从-2的63次方（-922，337，203，685，477.5808到2的63次方-1（+922，337，203，685，477.5807），数据精度为万分之一货币单位。MONEY 数据类型使用8个字节存储。  
  
    2 SMALLMONEY  
SMALLMONEY数据类型类似于MONEY 类型，但其存储的货币值范围比MONEY数据类型小,其取值从-214,748.3648到+214,748.3647,存储空间为4 个字节。

九）特定数据类型  
DB2或Sybase数据库 中包含了一些用于数据存储的特殊数据类型。

1 TIMESTAMP  
TIMESTAMP数据类型提供数据库范围内的惟一值此类型相当于BINARY8或VARBINARY（8），但当它所定义的列在更新或插入数据行时，此列的值会被自动更新，一个计数值将自动地添加到此TIMESTAMP数据列中。每个数据库表中只能有一个TIMESTAMP数据列。如果建立一个名为“TIMESTAMP”的列，则该列的类型将被自动设为TIMESTAMP数据类型。

2 UNIQUEIDENTIFIER  
UNIQUEIDENTIFIER 数据类型存储一个16 位的二进制数字。此数字称为（GUIDGlobally Unique Identifier ，即全球惟一鉴别号）。此数字由SQLServer 的NEWID函数产生的全球惟一的编码，在全球各地的计算机经由此函数产生的数字不会相同。

十） 用户自定义数据类型  
SYSNAME SYSNAME 数据类型是系统提供给用户的，便于用户自定义数据类型。它被定义为NVARCHAR（128），即它可存储128个UNICODE字符或256个一般字符。其具体使用方法请参见第7章“管理数据库表”中的“自定义数据类型”章节。

十一）新数据类型  
DB2或Sybase数据库 2000 中增加了3 种数据类型：BIGINT、SQL\_VARIANT和TABLE。其中BIGINT数据类型已在整数类型中介绍，下面介绍其余两种：  
    1 SQL\_VARIANT  
SQL\_VARIANT数据类型可以存储除文本、图形数据（TEXT、NTEXT、IMAGE）和TIMESTAMP类型数据外的其它任何合法的DB2或Sybase数据库数据。此数据类型大大方便了DB2或Sybase数据库的开发工作。  
    2 TABLE  
TABLE 数据类型用于存储对表或视图处理后的结果集。这一新类型使得变量可以存储一个表，从而使函数或过程返回查询结果更加方便快、捷其、使用请参见第13章“游标、视图和自定义函数”。

## 四、变量

Transact-SQL中可以使用两种变量，一种是局部变量（Local Variable）另外一种是全局变量（Global Variable）。

局部变量

局部变量是用户可自定义的变量，它的作用范围仅在程序内部。在程序中通常用来储存从表中查询到的数据，或当作程序执行过程中暂存变量使用。局部变量必须以“@”开头，而且必须先用DECLARE命令说明后才可使用。其说明形式如下：

DECLARE @变量名 变量类型 [@变量名 变量类型…]

其中变量类型可以是DB2或Sybase数据库 2000支持的所有数据类型，也可以是用户自定义的数据类型。

在Transact-SQL中不能像在一般的程序语言中一样使用“变量=变量值”来给变量赋值。必须使用SELECT或SET命令来设定变量的值，其语法如下：

SELECT@局部变量=变量值

SET @局部变量=变量值

例5：声明一个长度为10 个字符的变量“id”并赋值

declare@id char（10）  
select@id=‘10010001’

**注意：**可以在Select命令查询数据时，在Select命令中直接将列值赋给变量。

例6:查询编号为“10010001”的员工和工资，将其分别赋予变量name和wage。

use pangu  
declare@name char（30）@wage money  
select@name=e\_name，@wage=e\_wage  
from employee  
where emp\_id='10010001'  
select@name as e\_name，@wage as e\_wage  
运行结果如下：  
e\_name e\_wage  
------------------------------ -----------  
张三 8000.0000

**注意：**数据库语言和编程语言有一些关键字，关键字是在某一一样下能够促使某一操作发生的字符组合，为避免冲突和产生错误，在命令表、列、变量以及其它对象时应避免使用关键字。

二）全局变量

全局变量是DB2或Sybase数据库系统内部使用的变量，其作用范围并不局限于某一程序，而是任何程序均可随时调用全局变量通常存储一些DB2或Sybase数据库的配置设定值和效能统计数据。用户可在程序中用全局变量来测试系统的设定值或Transact-SQL命令执行后的状态值。

**注意：**全局变量不是由用户的程序定义的，它们是在服务器级定应义的。只能使用预先说明及定义的变局变量。引用全局变量时，必须以“@@”开头。局部变量的名称不能与全局变量的名称相同，否则会在应用中出错。

## 五、注释运算符与通配符

一） 注释符（Annotation）  
在Transact-SQL 中可使用两类注释符。  
ANSI 标准的注释符“--” 用于单行注释；  
与C语言相同的程序注释符号，即“/\*\*/”。“/\*”用于注释文字的开头，“\*/”用于注释文字的结尾，可在程序中标识多行文字为注释。

4.5.2 运算符（Operator）  
    1 算术运算符  
包括：+（加）、―（减）、（乘）、（除）、%（取余）  
    2 比较运算符  
包括：>（大于）<（小于）、= （等于）、>=（大于等于）、<=（小于等于）、<>  
（不等于）、!=（不等于）、!>（不大于）!<（ 不小于）  
其中!=、!>、!<不是ANSI标准的运算符。  
    3 逻辑运算符  
包括：AND（与）、OR（或）、NOT（非）  
    4 位运算符  
包括：按位与、|（按位或）、~（按位非）^（按位异或）  
    5 连接运算符  
连接运算符“+”用于连接两个或两个以上的字符或二进制串、列名或者串和列的混合体，将一个串加入到另一个串的末尾。其语法如下：  
<expression1>+<expression2>  
例7  
use pangu  
declare@startdate datetime  
set@startdate=‘1/1/2000’  
select‘Start Date：’+convert（varchar（12）@startdate）  
--convert（）函数用于转换数据类型，见4.8.4  
运行结果如下：  
Start Date：Jan 1 2000  
例8：  
use pangu  
select‘月薪最高的员工是：‘+ e\_name+’月薪为：‘+convert（varchar（10）e\_wage  
from employee  
where e\_wage=  
（select max （e\_wage）  
from employee）  
运行结果如下：  
月薪最高的员工是：张三月薪为8000.00  
（1 row （s） affected）  
在Transact-SQL中运算符的处理顺序如下所示，如果相同层次的运算出现在一起时则处理顺序位从左到右。

* 括号 （）
* 位运算符 ~
* 算术运算符 \*、/、%
* 算术运算符 +、-
* 位运算符 ^
* 位运算符 &
* 位运算符 |
* 逻辑运算符 NOT\
* 逻辑运算符 AND
* 逻辑运算符 OR

三） 通配符Wildcard  
在DB2或Sybase数据库中可以使用如下表所示的通配符。

表Transact-SQL的通配符

## 六、流程控制命令

Transact-SQL 语言使用的流程控制命令与常见的程序设计语言类似主要有以下几种控制命令。  
  
一） IF…ELSE  
其语法如下：  
IF <条件表达式>  
<命令行或程序块>  
[ELSE [条件表达式]  
<命令行或程序块>]  
其中<条件表达式>可以是各种表达式的组合，但表达式的值必须是逻辑值“真”或“假”。ELSE子句是可选的，最简单的IF语句没有ELSE子句部分。IF…ELSE用来判断当某一条件成立时执行某段程序，条件不成立时执行另一段程序。如果不使用程序块，IF或ELSE只能执行一条命令。IF…ELSE可以进行嵌套。  
  
例9  
declare@x int,@y int,@z int  
select @x=1,@y=2, @z=3  
if@x>@y  
  print'x>y' --打印字符串'x>y'  
else if@y>@z  
print'y>z'  
    else print'z>y'  
运行结果如下  
z>y   
**注意：**在Transact-SQL中最多可嵌套32级。

二） BEGIN…END  
其语法如下：  
BEGIN  
<命令行或程序块>  
END  
BEGIN…END用来设定一个程序块，将在BEGIN…END内的所有程序视为一个单元执行BEGIN…END经常在条件语句，如IF…ELSE中使用。在BEGIN…END中可嵌套另外的BEGIN…END来定义另一程序块。

三） CASE  
CASE 命令有两种语句格式:  
CASE <运算式>  
WHEN <运算式>THEN<运算式>  
…   
WHEN<运算式>THEN<运算式>  
[ELSE<运算式>]  
END

CASE  
WHEN <条件表达式> THEN <运算式>  
WHEN <条件表达式> THEN <运算式>  
[ELSE <运算式>]  
END  
CASE命令可以嵌套到SQL命令中。  
例10：调整员工工资，工作级别为“1”的上调8%，工作级别为“2”的上调7%，工作级别为“3”的上调6%，其它上调5%。  
use pangu  
update employee  
set e\_wage =  
case  
when job\_level = ’1’ then e\_wage\*1.08  
when job\_level = ’2’ then e\_wage\*1.07  
when job\_level = ’3’ then e\_wage\*1.06  
else e\_wage\*1.05  
end  
**注意:**执行CASE子句时，只运行第一个匹配的子名。

四） WHILE…CONTINUE…BREAK  
其语法如下:  
WHILE <条件表达式>  
BEGIN  
<命令行或程序块>  
[BREAK]  
[CONTINUE]  
[命令行或程序块]  
END  
WHILE 命令在设定的条件成立时会重复执行命令行或程序块。CONTINUE命令可以让程序跳过CONTINUE 命令之后的语句，回到WHILE 循环的第一行命令。BREAK 命令则让程序完全跳出循环，结束WHILE 命令的执行。WHILE 语句也可以嵌套。

例11:  
declare @x int, @y int, @c int  
select @x = 1, @y=1  
while @x < 3  
begin  
print @x --打印变量x 的值  
while @y < 3  
begin  
select @c = 100\*@ x+ @y  
print @c --打印变量c 的值  
select @y = @y + 1  
end  
select @x = @x + 1  
select @y = 1  
end  
运行结果如下  
1  
101  
102  
2  
201  
202  
  
五） WAITFOR  
其语法如下:  
WAITFOR {DELAY <‘时间’> | TIME <‘时间’>  
| ERROREXIT | PROCESSEXIT | MIRROREXIT}  
WAITFOR 命令用来暂时停止程序执行，直到所设定的等待时间已过或所设定的时间已到才继续往下执行。其中‘时间’必须为DATETIME 类型的数据，如：‘11:15:27’，  
但不能包括日期各关键字含义如下：

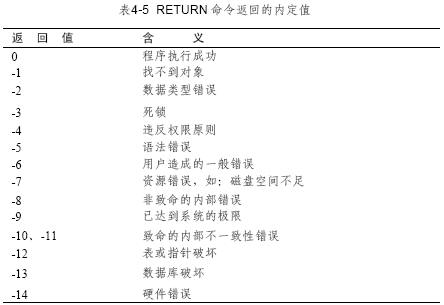
* DELAY 用来设定等待的时间最多可达24 小时；
* TIME 用来设定等待结束的时间点；
* ERROREXIT 直到处理非正常中断；
* PROCESSEXIT 直到处理正常或非正常中断；
* MIRROREXIT 直到镜像设备失败。

例12 等待1 小时2 分零3 秒后才执行SELECT 语句  
waitfor delay ’01:02:03’  
select \* from employee

例13：等到晚上11 点零8 分后才执行SELECT 语句  
waitfor time ’23:08:00’  
select \* from employee

六） GOTO  
语法如下：  
GOTO 标识符  
GOTO 命令用来改变程序执行的流程，使程序跳到标有标识符的指定的程序行再继续往下执行。作为跳转目标的标识符可为数字与字符的组合，但必须以“：”结尾，如‘12：’或‘a\_1：’。在GOTO 命令行，标识符后不必跟“：”。  
例14 分行打印字符‘1’、‘2’、‘3’、‘4’、‘5’  
declare @x int  
select @x = 1  
label\_1  
print @x  
select @x = @x + 1  
while @x < 6  
goto label\_1

七） RETURN  
语法如下  
RETURN [整数值]  
RETURN 命令用于结束当前程序的执行，返回到上一个调用它的程序或其它程序。在括号内可指定一个返回值。  
例15  
declare @x int @y int  
select @x = 1 @y = 2  
if x>y  
return 1  
else  
return 2  
如果没有指定返回值，DB2或Sybase数据库 系统会根据程序执行的结果返回一个内定值，如  
下表所示。



如果运行过程产生了多个错误，DB2或Sybase数据库系统将返回绝对值最大的数值；如果此时用户此时定义了返回值，则以返回用户定义的值。RETURN语句不能返回NULL值。

## 七、常用命令简介

一）BACKUP  
BACKUP 命令用于将数据库内容或其事务处理日志备份到存储介质上（软盘、硬盘、磁带）。等DB2或Sybase数据库 7.0 以前的版本用的是DUMP 命令来执行此功能，从DB2或Sybase数据库 2000起，不再使用DUMP 命令。关于BACKUP 命令的详情请参见“数据备份与恢复”章节。  
    二）CHECKPOINT  
语法如下：  
CHECKPOINT  
CHECKPOINT 命令用于将当前工作的数据库中被更改过的数据页data page 或日志页（log page）从数据缓冲器（data buffer cache）中强制写入硬盘。  
    三）DBCC  
DBCC Database Base Consistency Checker，数据库一致性检查程序）命令用于验证数据库完整性、查找错误、分析系统使用情况等。  
DBCC 命令后必须加上子命令，系统才知道要做什么。如DBCC CHECKALLOC 命令检查目前数据库内所有数据页的分配和使用情况。  
关于DBCC 命令的详情，请参见附录。

    四）DECLARE  
语法如下:  
DECLARE {{ @local\_variable data\_type }  
| { @cursor\_variable\_name CURSOR }  
| { table\_type\_definition }  
} [，...n]]  
DECLARE 命令用于声明一个或多个局部变量、游标变量或表变量。在用DECLARE命令声明之后，所有的变量都被赋予初值NULL。需要用SELECT 或SET命令来给变量赋值。变量类型可为系统定义的或用户定义的类型，但不能为TEXT、NTEXT、IMAGE类型。CURSOR 指名变量是局部的游标变量。  
例16  
declare @x float @y datetime  
select @x = pi @y = getdate  
print @x  
print @y  
运行结果如下:  
3.14159  
Jun 17 2000 4:32PM  
**注意：**如果变量为字符型，那么在data\_type表达中应指明  
其最大长度，否则系统认为其长度为1。  
例17  
declare @x char @y char 10  
select @x = '123' @y = 'data\_type'  
print @x  
print @y  
运行结果如下  
1  
data\_type   
  
    五）EXECUTE  
EXECUTE 命令用来执行存储过程。   
    六） KILL  
KILL 命令用于终止某一过程的执行。  
    七） PRINT  
语法如下：  
PRINT 'any ASCII text' | @local\_variable | @@FUNCTION | string\_expression  
PRINT 命令向客户端返回一个用户自定义的信息，即显示一个字符串（最长为255个字符）、局部变量或全局变量。如果变量值不是字符串的话，必须先用数据类型转换函数CONVERT（），将其转换为字符串。其中，string\_expression 是可返回一个字符串的表达式。表达式的长度可以超过8000 个字符，但超过8000 的字符将不会显示。  
例18：  
declare @x char 10 @y char 10  
select @x = 'SQL' @y = 'Server'  
print '科技'  
print @x + @y

运行结果如下：  
科技  
DB2或Sybase数据库  
  
   4.7.8 RAISERROR  
RAISERROR 命令用于在DB2或Sybase数据库 系统返回错误信息时，同时返回用户指定的信息。其具体用法请参见“存储过程”章节。  
    4.7.9 READTEXT  
READTEXT 命令语法如下  
READTEXT {table.column text\_pointer offset size} [HOLDLOCK]  
READTEXT 命令用于从数据类型为TEXT、NTEXT 或IMAGE 的列中读取数据。命令从偏移位置offset+1 个字符起读取size 个字符，如果size 为0， 则会读取4KB 的数据。其中text\_pointer 是指向存储文本的第一个数据库页的指针，它可以用TEXTPTR（）函数来获取。关于TEXTPTR （）函数的使用请参见“4.8.7 TEXT 和IMAGE 函数”。HOLDLOCK 选项用于锁定所读取的数据直到传输结束，这段时间内，其它用户只能读取数据，不能更改数据。  
例19：  
use pangu  
declare @t\_pointer varbinary （16）  
select @t\_pointer = textptr （d\_intro）  
from department  
where d\_name = '技术部'  
readtext department.d\_intro @t\_pointer 0 42  
运行结果如下：  
d\_intro  
--------------------------------------------------------------------  
进行技术研究与开发，提供最新的技术动态信息  
（1 row （s） affected）  
  
**注意：**如果数据列为汉字，则offset值应取o或其它偶数，如果用奇数，则会出现乱码  
  
   十） RESTORE  
RESTORE 命令用来将数据库或其事务处理日志备份文件由存储介质回存到DB2或Sybase数据库系统中。DB2或Sybase数据库 7.0 以前的版本用的是LOAD命令来执行此功能，从DB2或Sybase数据库 2000起，不再使用LOAD 命令。关于RESTORE 命令的详情，请参见“数据备份与恢复”章节  
    十一） SELECT  
SELECT 命令可用于给变量赋值其语法如下：  
SELECT {@local\_variable = expression } [ ...n]  
SELECT 命令可以一次给多个变量赋值。当表达式expression 为列名时，SELECT 命令可利用其查询功能一次返回多个值，变量中保存的是其返回的最后一个值。如果SELECT命令没有返回值，则变量值仍为其原来的值。当表达式expression 是一个子查询时，如果子查询没有返回值则变量被设为NULL。  
例20：  
use pangu  
declare @x char （30）  
select @x = 'CPU'  
select @x = d\_name  
from department  
where dept\_id = 9999   
select @x  
运行结果如下:  
-----------------------  
CPU  
例21:  
use pangu  
declare @x char 30  
select @x = 'Main Board'  
select @x = （select d\_name  
from department  
where dept\_id=9999）  
select @x  
运行结果如下  
-----------------------  
NULL

十二）SET  
SET 命令有两种用法：  
1 用于给局部变量赋值  
其语法如下：  
SET { {@local\_variable = expression} | { @cursor\_variable =  
{ @cursor\_variable cursor\_name  
| { CURSOR  
[FORWARD\_ONLY | SCROLL]  
[STATIC | KEYSET | DYNAMIC | FAST\_FORWARD]  
[READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC]  
[TYPE\_WARNING]  
FOR select\_statement  
[FOR {READ ONLY  
| UPDATE [OF column\_name [ ...n]] } ] } } } }  
其中的CURSOR 等参数将在“游标”一章中讲述。  
在用DECLARE 命令声明之后，所有的变量都被赋予初值NULL。 需要用SET 命令来给变量赋值。但与SELECT 命令不同的是，SET 命令一次只能给一个变量赋值。不过由于SET 命令功能更强，且更严密，因此，DB2或Sybase数据库 推荐使用SET 命令来给变量赋值。

例22  
declare @x char （30）  
set @x = 'This is a SET command.'  
select @x  
运行结果如下  
------------------------------  
This is a SET command.  
例4-23  
use pangu  
declare @department\_num int @x char （10）  
set @department\_num = select count （\*）  
from department）  
set @x = '部门数目'  
print @x  
select @department\_num  
运行结果如下  
部门数目  
-----------  
7

2 用于设定用户执行SQL 命令，时DB2或Sybase数据库 的处理选项设定  
有以下几种设定方式：

* SET 选项ON
* SET 选项OFF
* SET 选项值

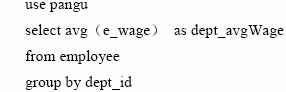
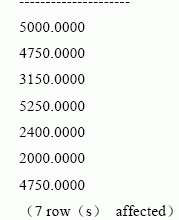
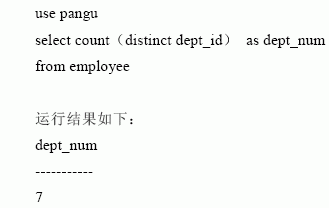
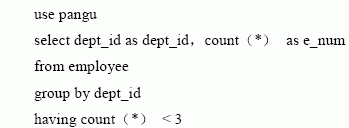
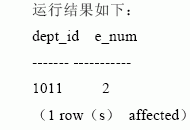
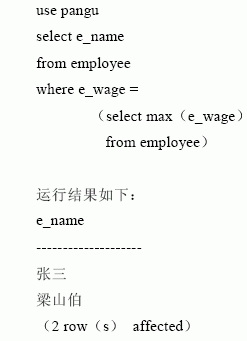
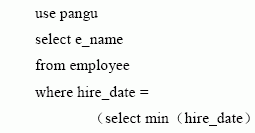
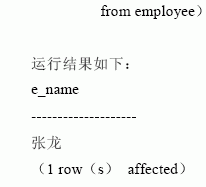
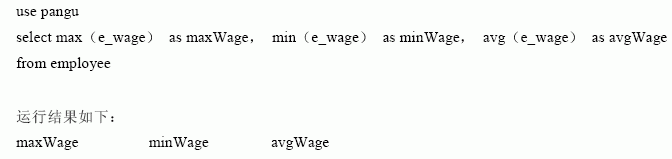
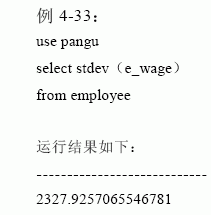
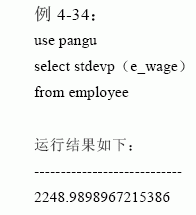
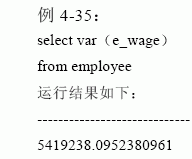
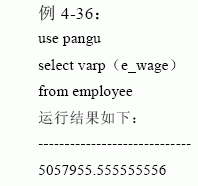
语法如下  
SHUTDOWN [WITH NOWAIT]  
SHUTDOWN 命令用于停止DB2或Sybase数据库 的执行,当使用NOWAIT 参数时，SHUTDOWN命令立即停止DB2或Sybase数据库，在终止所有的用户过程并对每一现行的事务发生一个回滚后，退出DB2或Sybase数据库。当没有用NOWAIT 参数时，SHUTDOWN 命令将按以下步骤执行：  
（1） 终止任何用户登录DB2或Sybase数据库。  
（2） 等待尚未完成的Transact-SQL 命令或存储过程执行完毕。  
（3） 在每个数据库中执行CHECKPOINT 命令  
（4） 停止DB2或Sybase数据库 的执行  
例24  
shutdown with nowait  
运行结果如下：  
------------------------------  
Changed language setting to us\_english.  
Server shut down by request.  
DB2或Sybase数据库 is terminating this process.  
停止DB2或Sybase数据库 的执行的方法还有好几种，我们将在“企业管理器EnterpriseManager” 章节中介绍。

    十四） WRITETEXT  
语法如下：  
WRITETEXT {table.column text\_pointer} [WITH LOG] {data}  
WRITETEXT 命令用于向数据类型为TEXT、 NTEXT 或IMAGE 的列中读取数据。其中text\_pointer 是指向存储文本的第一个数据库页的指针,它可以用TEXTPTR（）来获取。WOTH LOG选项用于记录所写入的数据。data 可为文字或变量，其最大长度为120KB。注意：WRITETEXT命令不能作用于视图。

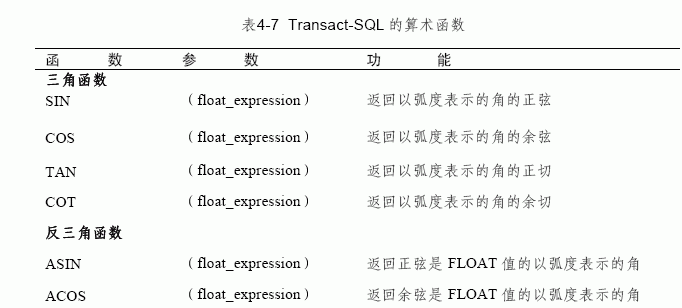
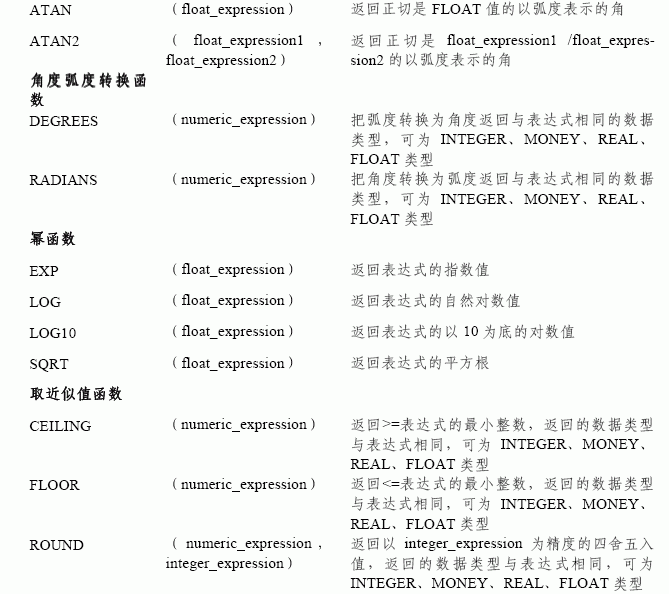
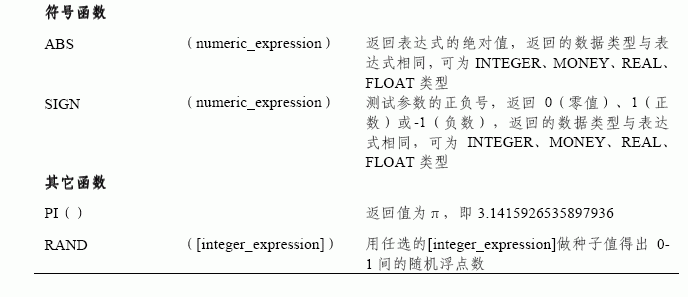
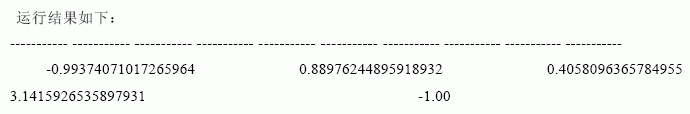
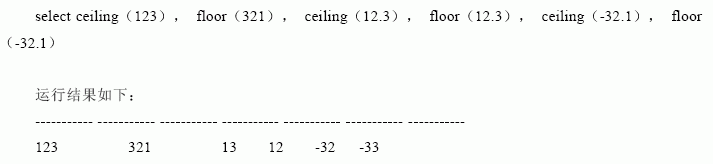
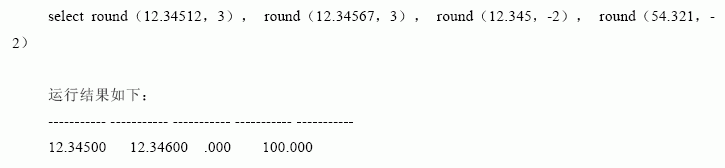
推荐使用UPDATETEXT 命令来修改TEXT、NTEXT 和IMAGE 类型的数据。因为WRITETEXT 命令将数据列的数据完全替换，而UPDATETEXT 命令可以只更改数据列的一部分。关于UPDATETEXT 命令的使用将在“数据库更新”章节中讲述。  
例25  
declare @t\_pointer binary （16）  
use pangu  
select @t\_pointer = textptr d\_intro  
from department  
where d\_name = '技术部'  
writetext department.d\_intro @t\_pointer '进行技术研究与开发提供最新的技术动态信息'  
  
    十五） USE  
语法如下  
USE {databasename}  
USE 命令用于改变当前使用的数据库为指定的数据库。用户必须是目标数据库的用户成员或目标数据库建有GUEST 用户账号时，使用USE 命令才能成功切换到目标数据库。

## 八、常用函数简介

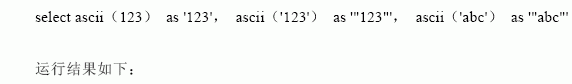
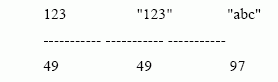
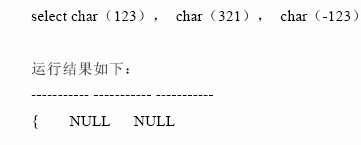
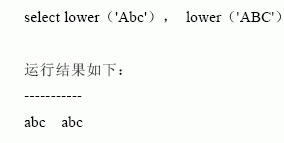
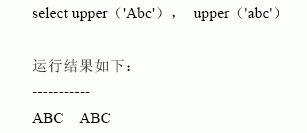
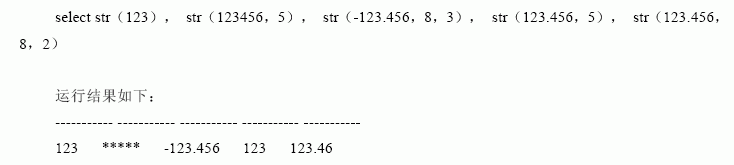
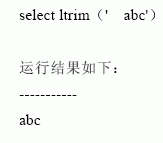
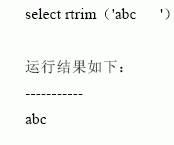
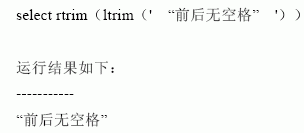
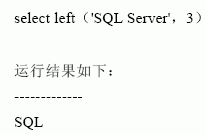
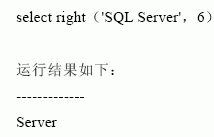
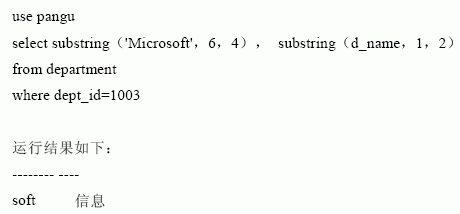
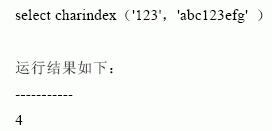
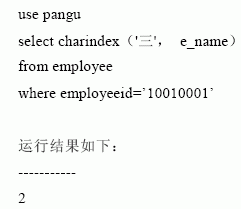
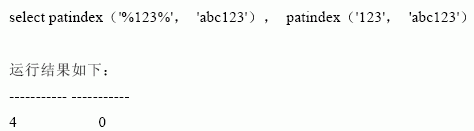
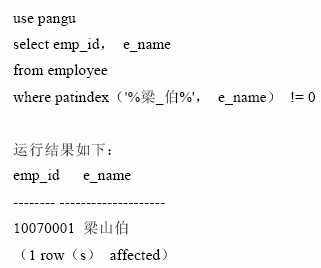
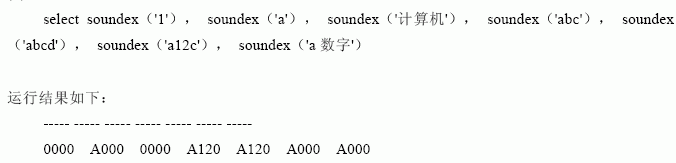
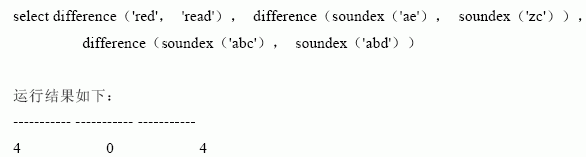
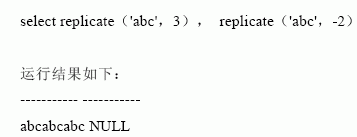
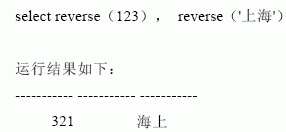
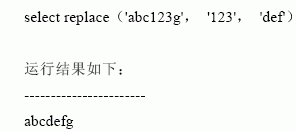
一）统计函数：

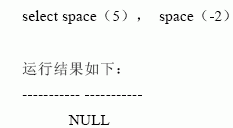
统计函数是在数据库操作中时常使用的函数,又称为基本函数或集函数,常用的统计函数如下表所示.  
  
  
  
这些函数通常用在SELECT 子句中，作为结果数据集的字段返回的结果。在SELECT语句的SELECT 子句中使用函数的语法如下：   
    **SELECT 函数名（列名1 或\*），...函数名（列名n）**  
    **FROM 表名**  
**提醒：**函数的对象或自变量必须包括在圆括号内。如果函数需要一个以上的自变量，可用逗号隔开各个自变量。  
  
1、AVG（）  
    AVG（） 函数返回有关列值的算术平均值。此函数只适用数值型的列。其语法如下：  
数据库  
    例26： 求各部门的平均工资  
  
  
  
    运行结果如下：  
数据库  
  
  
2、COUNT（）  
    COUNT（） 函数返回与选择表达式匹配的列中不为NULL 值的数据个数。COUNT（）函数的语法如下：  
数据库  
    例27： 计算企业的部门数目  
  
  
**提醒：**如果用COUNT（）函数引用一个列名，则返回列值的个数。   
  
    COUNT（） 函数在计算中重复计算了相同的值。如果使用了关键字DISTINCT， 则COUNT（） 函数就返回行惟一值的个数。  
    如果在SELECT 子句的列名位置上使用符号（\*）， 即使用COUNT （\*）， 则指定了与SELECT 语句的判别式匹配的所有行，COUNT（）函数将计算字段的行数，包括为NULL值的行。  
  
    例28： 列出员工少于3 人的部门编号  
  
  
  
3、MAX（）  
    MAX（） 函数返回某一列的最大值。此函数适用于数值型、字符型、和日期型的列。对于列值为NULL 的列，MAX（） 函数不将其列为对比的对象。其语法如下：  
数据库  
    例29： 求工资最高的员工姓名  
  
  
4、MIN（）  
    MIN（） 函数返回某一列的最小值。此函数适用于数值型、字符型、和日期型的列。对于列值为NULL 的列，MIN（） 函数不将其列为对比的对象。其语法如下：  
数据库  
    例30： 求最资深的员工姓名  
  
  
  
5、SUM（）  
    SUM（） 函数用来返回诸如列值这样的实体的总和。此函数只适用于数值型的列，不包括NULL 值。其语法如下：  
数据库  
    例31： 求各部门的员工工资总额  
  
    可以在一个语句中使用多个函数。  
    例32： 求员工工资的最大值、最小值、平均值  
  
数据库  
  
除了上述常用的函数外，统计函数还有以下几种：  
  
6、STDEV（）  
    STDEV（） 函数语法如下：  
数据库  
    STDEV（） 函数返回表达式中所有数据的标准差（Statistical Standard Deviation）。表达式通常为表的某一数据类型为NUMERIC 的列（或近似NUMERIC 类型的列，如MONEY 类型，但BIT 类型除外）。表达式中的NULL 值将被忽略。其返回值为FLOAT类型。  
  
  
7、STDEVP（）  
    STDEVP（） 函数语法如下：  
数据库  
   STDEVP（） 函数返回总体标准差（Statistical Standard Deviation for The Population）。表达式及返回值类型同STDEV（） 函数。  
  
  
8、VAR（）  
    VAR（） 函数语法如下：  
数据库  
    VAR（） 函数返回表达式中所有值的统计变异数（Statistical Variance）。 表达式及返回值类型同STDEV（） 函数。  
  
  
9、VARP（）  
    VARP（） 函数语法如下：  
数据库  
    VARP（） 函数返回总体变异数（Statistical Variance for The Population）。 表达式及返回值类型同STDEV（） 函数。  


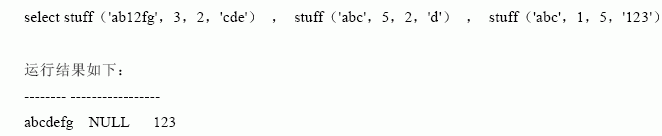
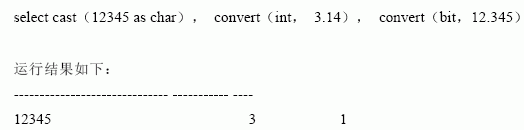
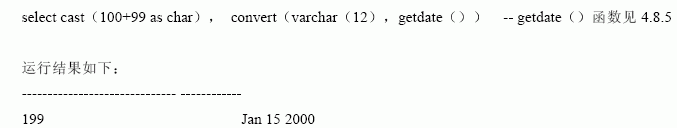
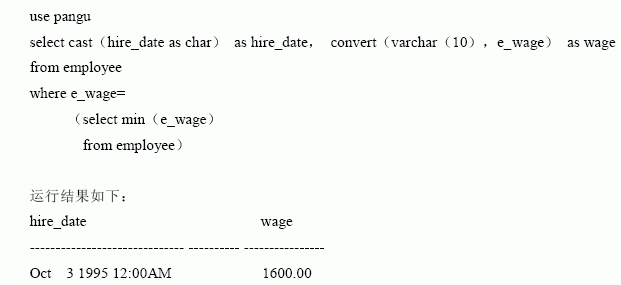
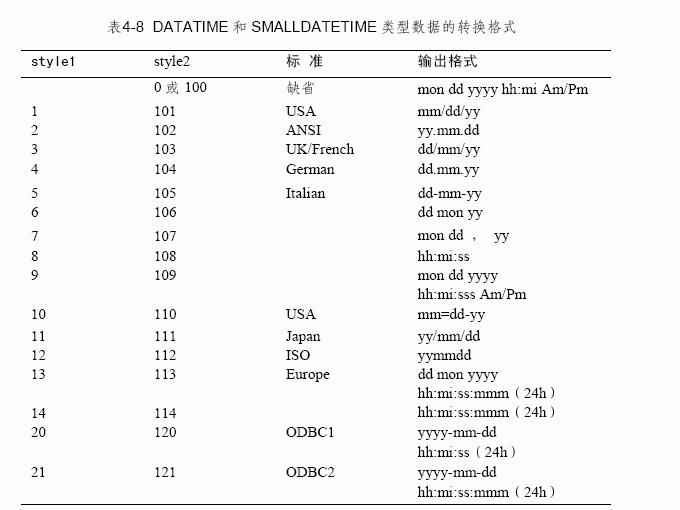
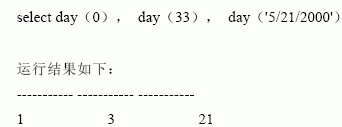
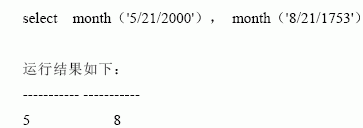
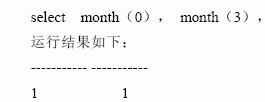
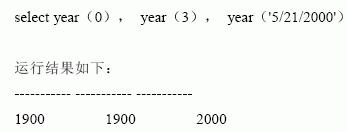
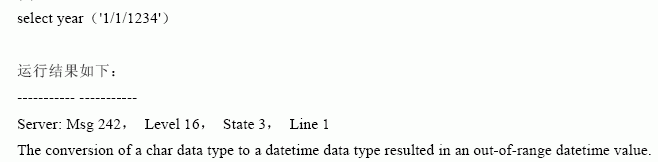
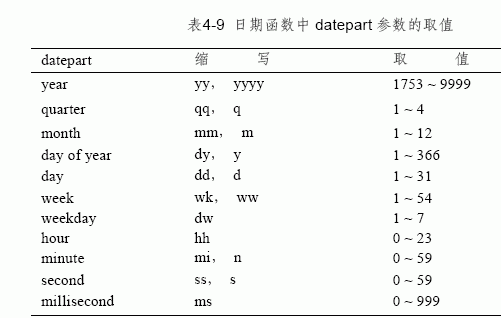
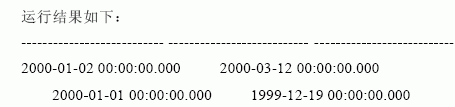
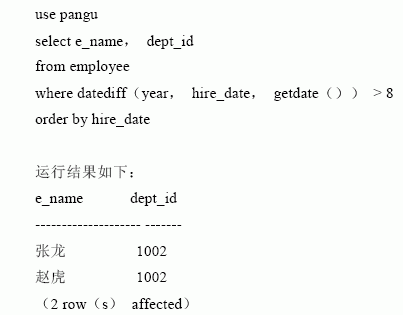
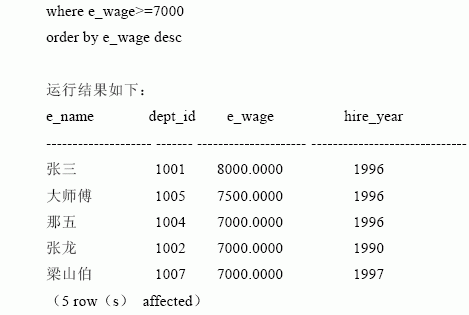
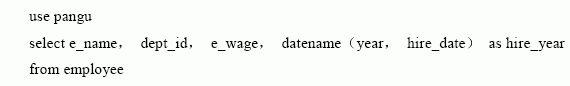
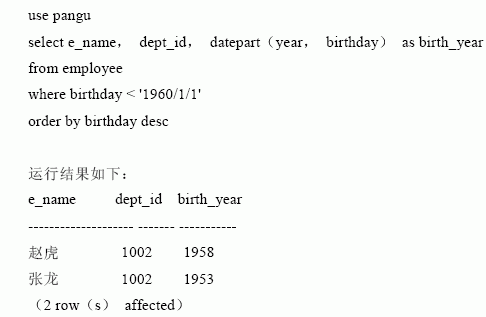
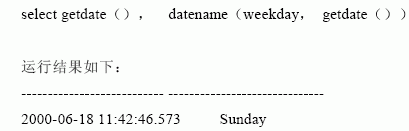
二）算术函数：

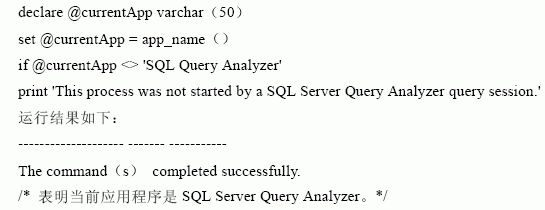
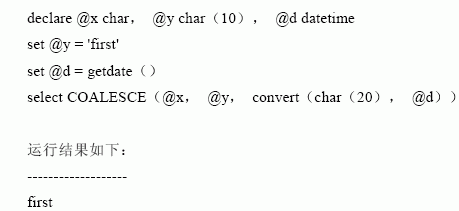
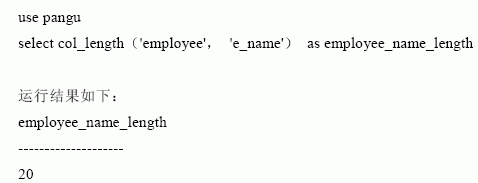
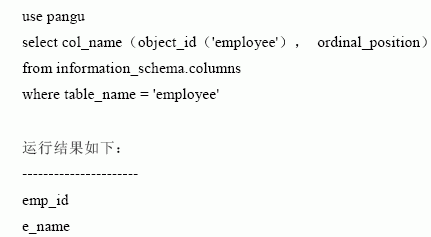
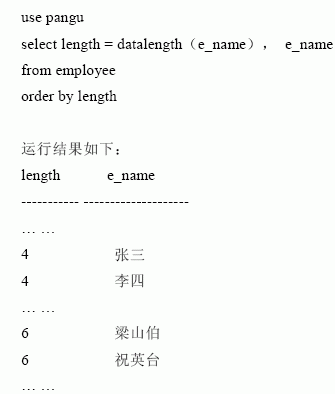
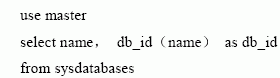
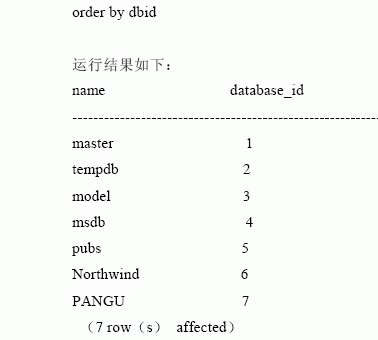
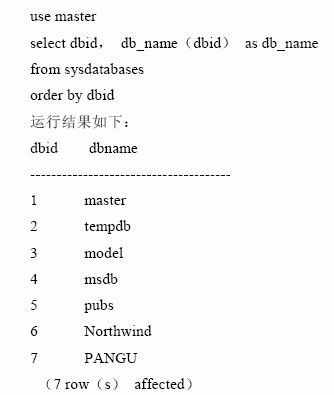
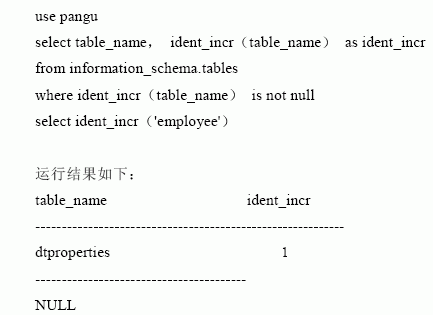
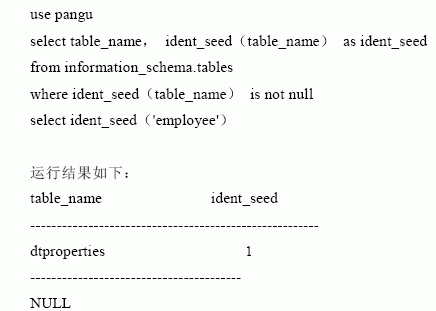
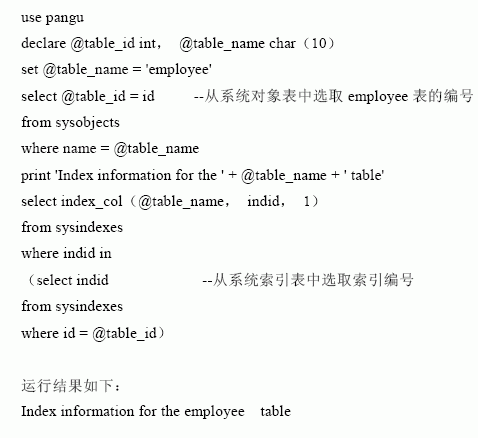
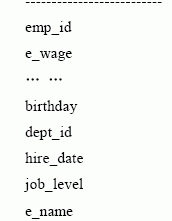
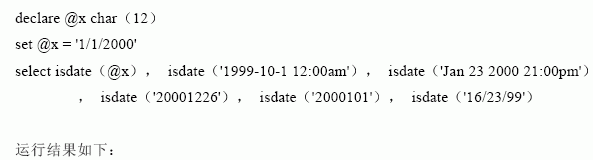
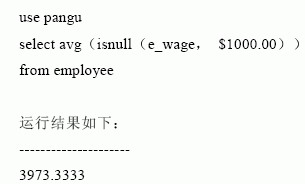
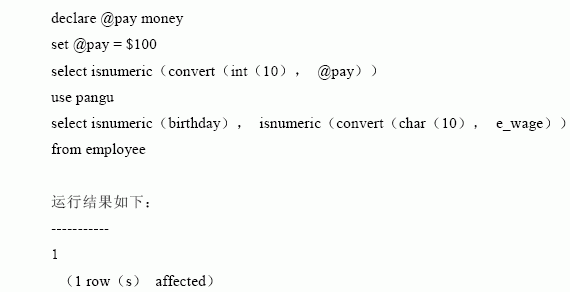
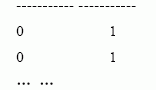
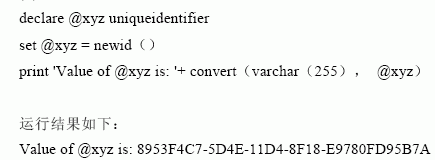
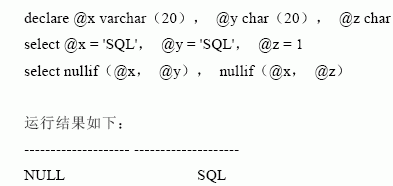
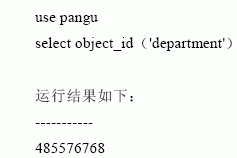
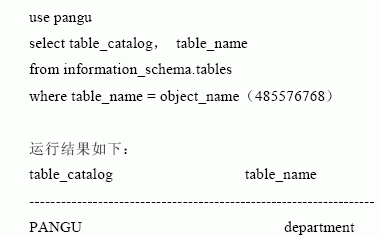
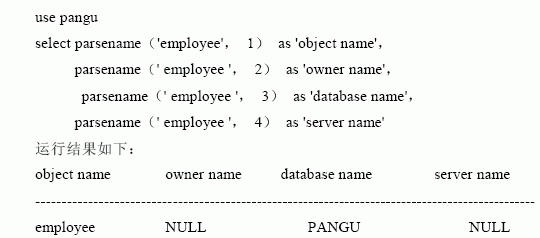
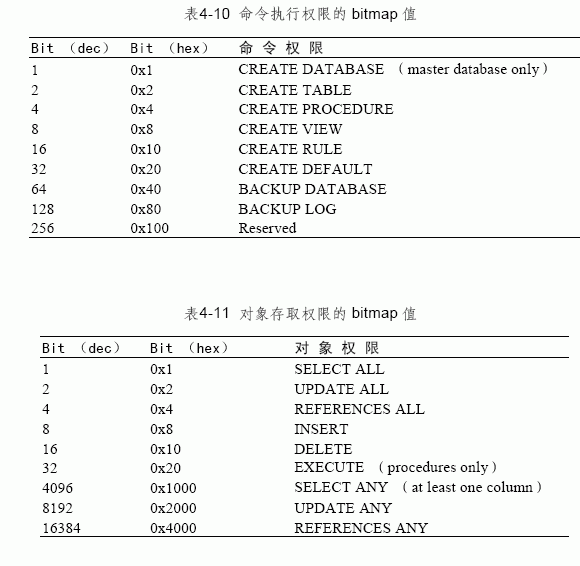
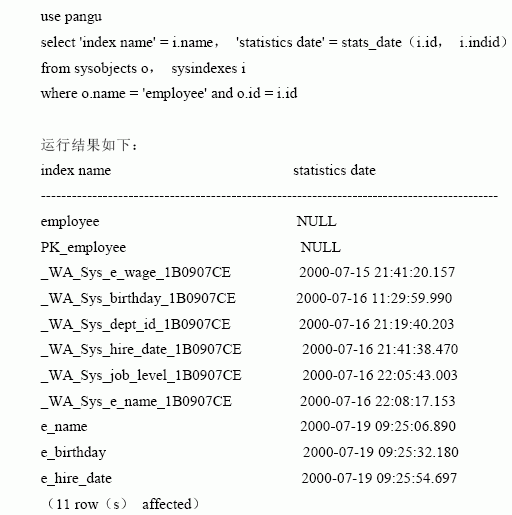
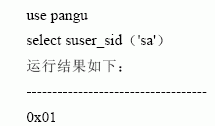
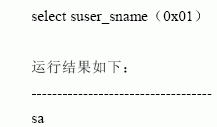
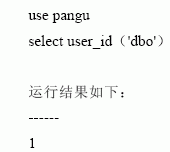
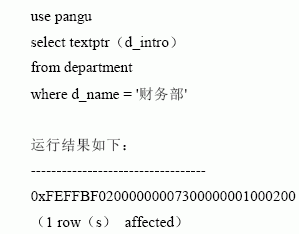
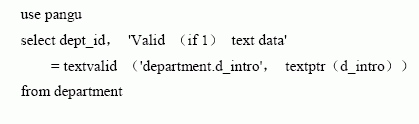
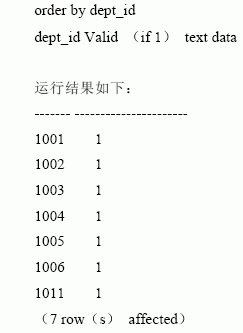
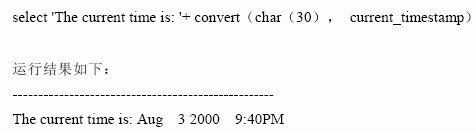
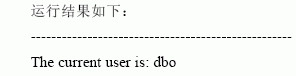
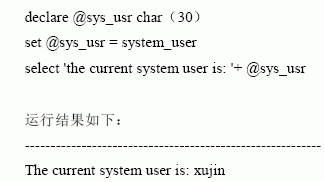
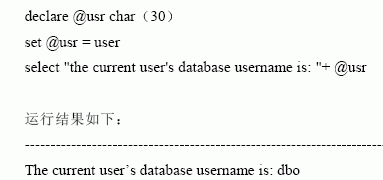
算术函数可对数据类型为整型INTEGER、 浮点型FLOAT、 实型REAL、 货币型MONEY和SMALLMONEY 的列进行操作。它的返回值是6 位小数，如果使用出错，则返回NULL值并显示警告信息。可以在SELECT 语句的SELECT 和WHERE 子句以及表达式中使用算术函数。Transact-SQL 中的算术函数见下表。  
  
  
  
  
    **例37：** 在同一表达式中使用SIN（）、 ATAN（）、 RAND（）、 PI（）、 SIGN（）函数   
算术函数  
  
    **例38：** 用CEILING（） 和FLOOR（） 函数返回大于或等于指定值的最小整数值和小于或等于指定值的最大整数值。  
  
  
  
    **例39：**ROUND（） 函数的使用  
  


三）字符串函数

字符串函数对二进制数据、字符串和表达式执行不同的运算。此类函数作用于CHAR、VARCHAR、 BINARY、 和VARBINARY 数据类型以及可以隐式转换为CHAR 或VARCHAR的数据类型。可以在SELECT 语句的SELECT 和WHERE 子句以及表达式中使用字符串函数。常用的字符串函数有：  
  
1、字符转换函数  
有以下几种字符转换函数：  
·ASCII（）  
ASCII（） 函数返回字符表达式最左端字符的ASCII 码值。ASCII（） 函数语法如下：  
字符串函数（一）  
  
  **例40：**  
  
  
  
**提醒：**在ASCII（）函数中，纯数字的字符串可不用‘’括起来，但含其它字符的字符串必须用‘’括起来使用，否则会出错。  
  
·CHAR（）  
CHAR（） 函数用于将ASCII 码转换为字符。其语法如下：  
字符串函数（一）  
如果没有输入0 ~ 255 之间的ASCII 码值，CHAR（） 函数会返回一个NULL 值。  
  
  **例41：**  
  
  
·LOWER（）  
LOWER（） 函数把字符串全部转换为小写，其语法如下：  
   字符串函数（一）  
  
  **例42：**  
  
  
·UPPER（）  
UPPER（） 函数把字符串全部转换为大写，其语法如下：  
 字符串函数（一）  
  **例43：**  
  
  
·STR（）  
STR（） 函数把数值型数据转换为字符型数据，其语法如下：  
STR （<float \_expression>[，length[， <decimal>]]）  
  自变量length 和decimal 必须是非负值，length 指定返回的字符串的长度，decimal 指定返回的小数位数。如果没有指定长度，缺省的length 值为10， decimal 缺省值为0。 小数位数大于decimal 值时，STR（） 函数将其下一位四舍五入。指定长度应大于或等于数字的符号位数+小数点前的位数+小数点位数+小数点后的位数。如果<float \_expression>小数点前的位数超过了指定的长度，则返回指定长度的“\*”。  
  **例44：**  
  
2、去空格函数  
去空格函数有如下两种：  
·LTRIM（）  
LTRIM（） 函数把字符串头部的空格去掉，其语法如下：  
  LTRIM （<character \_expression>）  
  **例45：**  
  
  
·RTRIM（）  
RTRIM（） 函数把字符串尾部的空格去掉，其语法如下：  
  RTRIM （<character \_expression>）  
  
  **例4-46：**  
  
  
**提示：**在许多情况下，往往需要得到头部和尾部都没有空格字符的字符串，这时可将上两个函数嵌套使用。  
  
  **例4-47：**  
  
  
3、取子串函数  
取子串函数如下所示：  
·LEFT（）  
LEFT（） 函数返回部分字符串，其语法如下：  
  LEFT （<character\_expression>， <integer\_expression>）  
  LEFT（） 函数返回的子串是从字符串最左边起到第integer\_expression 个字符的部分。若integer\_expression 为负值，则返回NULL 值。  
  
  **例4-48：**  
  
  
·RIGHT（）  
RIGHT（） 函数返回部分字符串，其语法如下：  
  RIGHT （<character\_expression>， <integer\_expression>）  
  RIGHT（） 函数返回的子串是从字符串右边第integer\_expression 个字符起到最后一个字符的部分。若integer\_expression 为负值，则返回NULL 值。  
  
  **例49：**  
  
  
·SUBSTRING（）  
SUBSTRING（） 函数返回部分字符串，其语法如下：  
  SUBSTRING （<expression>， <starting\_ position>， length）  
  SUBSTRING（） 函数返回的子串是从字符串左边第starting\_ position 个字符起length个字符的部分。其中表达式可以是字符串或二进制串或含字段名的表达式。SUBSTRING（）函数不能用于TEXT 和IMAGE 数据类型。  
  
  **例50：**  
  
  
4、字符串比较函数  
字符串比较函数如下所示：  
·CHARINDEX（）  
CHARINDEX（） 函数返回字符串中某个指定的子串出现的开始位置，其语法如下：  
  CHARINDEX （<’substring\_expression’>， <expression>）  
  其中substring \_expression 是所要查找的字符表达式，expression 可为字符串也可为列名表达式。如果没有发现子串，则返回0 值。此函数不能用于TEXT 和IMAGE 数据类型。  
  
  **例51：**  
  
  
  **例52：**  
  
  
·PATINDEX（）  
PATINDEX（） 函数返回字符串中某个指定的子串出现的开始位置，其语法如下：  
  PATINDEX （<’%substring \_expression%’>， <column\_ name>）  
  其中子串表达式前后必须有百分号“%”否则返回值为0。  
  
  **例53：**  
  
  
  与CHARINDEX（） 函数不同的是，PATINDEX（） 函数的子串中可以使用通配符，且此函数可用于CHAR、 VARCHAR 和TEXT 数据类型。  
  
  **例54：**  
  
  
·SOUNDEX（）  
SOUNDEX（） 函数返回一个四位字符码，其语法如下：  
  SOUNDEX （<character \_expression>）  
  SOUNDEX（） 函数将character \_expression 转换为4 个字符的声音码。其中第一个码为原字符串的第一个字符，第二到第四个字符则为数字，是该字符串的声音字母所对应的数字，但是忽略了除首字母外的串中的所有元音字母 。SOUNDEX（） 函数可用来查找声音相似的字符串。但SOUNDEX 函数对数字和汉字均只返回0 值。  
  
  **例55：**  
  
  
·DIFFERENCE（）  
DIFFERENCE（） 函数返回由SOUNDEX（） 函数返回的两个字符表达式的值的差异。其语法如下：  
  DIFFERENCE （<character\_expression1>， <character\_expression2>）  
  值的差异是用0、 1、 2、 3、 4 来表示的，含义如下：  
  0--两个SOUNDEX（） 函数返回值的第一个字符不同  
  1--两个SOUNDEX（） 函数返回值的第一个字符相同  
  2--两个SOUNDEX（） 函数返回值的第一、二个字符相同  
  3--两个SOUNDEX（） 函数返回值的第一、二、三个字符相同  
  4--两个SOUNDEX（） 函数返回值完全相同  
  
  **例56：**  
  
  
5、字符串操作函数  
字符串操作函数如下所示：  
·QUOTENAME（）  
QUOTENAME（） 函数返回被特定字符括起来的字符串，其语法如下：  
QUOTENAME （<’character\_expression’>[， quote\_ character]）  
其中quote\_ character 标明括字符串所用的字符，如“‘”、“（”、“[” 等，缺省值为“[]”  
  
  **例57：**  
  
  
·REPLICATE（）  
REPLICATE（） 函数返回一个重复character\_expression 指定次数的字符串，其语法如下：  
REPLICATE （character\_expression integer\_expression）  
如果integer\_expression 值为负值，则REPLICATE（） 函数返回NULL 串。  
  
  **例58：**  
  
  
·REVERSE（）  
REVERSE（） 函数将指定的字符串的字符排列顺序颠倒，其语法如下：  
REVERSE （<character\_expression>）  
其中character\_expression 可以是字符串、常数或一个列的值。  
  
  **例59：**  
  
  
·REPLACE（）  
REPLACE（） 函数返回被替换了指定子串的字符串，其语法如下：  
REPLACE （<string\_expression1>， <string\_expression2>， <string\_expression3>）  
REPLACE（） 函数用string\_expression3 替换在string\_expression1 中的子串string\_expression2。  
  
  **例60：**  
  
  
·SPACE（）  
SPACE（） 函数返回一个有指定长度的空白字符串，其语法如下：  
SPACE （<integer\_expression>）  
如果integer\_expression 值为负值，则SPACE（） 函数返回NULL 串。

**例61：**  
  
  
·STUFF（）  
STUF（）函数用另一子串替换字符串指定位置、长度的子串，其语法如下：  
STUFF （<character\_expression1>， <start\_ position>， <length>，<character\_expression2>）  
如果起始位置为负或长度值为负，或者起始位置大于character\_expression1 的长度，则STUFF（） 函数返回NULL 值。如果length 长度大于character\_expression1 的长度，则character\_expression1 只保留首字符。

**例62：**  
  
四）数据类型转换函数  
在一般情况下，IBM DB2或Sybase ASE和SAS 会自动完成数据类型的转换，例如，可以直接将字符数据类型或表达式与DATATIME 数据类型或表达式比较当表达式中用了INTEGER、 SMALLINT或TINYINT 时，IBM DB2或Sybase ASE和SAS 也可将INTEGER 数据类型或表达式转换为SMALLINT数据类型或表达式，这称为隐式转换。如果不能确定IBM DB2或Sybase ASE和SAS 是否能完成隐式转换或者使用了不能隐式转换的其它数据类型，就需要使用数据类型转换函数做显式转换了。此类函数有两个：   
·CAST（）  
CAST（） 函数语法如下：  
CAST （<expression> AS <data\_ type>[ length ]）  
  
·CONVERT（）  
CONVERT（） 函数语法如下：  
CONVERT （<data\_ type>[ length ]， <expression> [， style]）   
  
**提醒：**1、data\_type为IBM DB2或Sybase ASE和SAS系统定义的数据类型，用户自定义的数据类型不能在此使用。  
2、length用于指定数据的长度，缺省值为30。  
3、把CHAR或VARCHAR类型转换为诸如INT或SAMLLINT这样的INTEGER类型、结果必须是带正号（+）或负号（-）的数值。  
4、TEXT类型到CHAR或VARCHAR类型转换最多为8000个字符，即CHAR或VARCHAR数据类型是最大长度。  
5、IMAGE类型存储的数据转换到BINARY或VARBINARY类型，最多为8000个字符。  
6、把整数值转换为MONEY或SMALLMONEY类型，按定义的国家的货币单位来处理，如人民币、美元、英镑等。  
7、BIT类型的转换把非零值转换为1，并仍以BIT类型存储。  
8、试图转换到不同长度的数据类型，会截短转换值并在转换值后显示“+”，以标识发生了这种截断。  
  
  **例63：**  
  
  
  **例64：**  
  
  
  **例65：**  
  
字符串函数（一）  
  
用CONVERT（） 函数的style 选项能以不同的格式显示日期和时间。style 是将DATATIME 和SMALLDATETIME 数据转换为字符串时所选用的由IBM DB2或Sybase ASE和SAS 系统提供的转换样式编号，不同的样式编号有不同的输出格式。如表4-8 所示。  
  
  
  **例66：**  
  
  
五）日期函数  
日期函数用来操作DATETIME 和SMALLDATETIME 类型的数据，执行算术运算。与其它函数一样，可以在SELECT 语句的SELECT 和WHERE 子句以及表达式中使用日期函数。其使用方法如下：  
 日期函数参数，其中参数个数应不同的函数而不同。  
  
·DAY（）  
DAY（） 函数语法如下：  
DAY （<date\_expression>）  
DAY（） 函数返回date\_expression 中的日期值。  
  
  **例67：**  
  
  
在上例中，DAY（） 函数使用整数值作参数，系统从1900 年1 月1 日起往后数指定整数值+1， 天再返回其日期。  
  
·MONTH（）  
MONTH（） 函数语法如下：  
MONTH (<date\_expression>)  
MONTH（） 函数返回date\_expression 中的月份值。  
  
  **例68：**  
  
  
与DAY（） 函数不同的是，MONTH（） 函数的参数为整数时，一律返回整数值1，即IBM DB2或Sybase ASE和SAS 认为其是1900 年1 月。  
  
  **例69：**  
  
  
·YEAR（）  
YEAR（） 函数语法如下：  
YEAR （<date\_expression>）  
YEAR（） 函数返回date\_expression 中的年份值。  
  
  **例70：**  
  
  
**提醒：**在使用日期函数时，其日期值应在1753年到9999年之间，这是IBM DB2或Sybase ASE和SAS系统所能识别的日期范围，否则会出现错误。  
  
  **例71：**  
  
  
·DATEADD（）  
DATEADD（） 函数语法如下：  
DATEADD （<datepart>， <number>， <date>）  
DATEADD（） 函数返回指定日期date 加上指定的额外日期间隔number 产生的新日期。参数“datepart” 在日期函数中经常被使用，它用来指定构成日期类型数据的各组件，如年、季、月、日、星期等。其取值如表4-9 所示：  
  
  
  **例72：**  
字符串函数（一）  
  
  
·DATEDIFF（）  
DATEDIFF（） 函数语法如下：  
DATEDIFF（） （<datepart>， <date1>， <date2>）  
DATEDIFF（） 函数返回两个指定日期在datepart 方面的不同之处，即date2 超过date1的差距值，其结果值是一个带有正负号的整数值。针对不同的datepart， DATEDIFF（）函数所允许的最大差距值不一样，如：datepart 为second 时，DATEDIFF（） 函数所允许的最大差距值为68： 年datepart 为millisecond 时，DATEDIFF（） 函数所允许的最大差距值为24 天20 小时30 分23 秒647 毫秒。  
  
  **例73：** 查询在本单位工作了8 年以上的员工的姓名和所在的部门，结果按在本单位工作的时间长短排序。  
  
  
·DATENAME（）  
DATENAME（） 函数语法如下：  
DATENAME （<datepart>， <date）>  
DATENAME（） 函数以字符串的形式返回日期的指定部分此部分。由datepart 来指定。  
  
  **例74：** 查询工资大于等于7000 的员工的姓名、部门编号、工资和进入单位的年份，结果按工资高低降序排列。  
  
  
·DATEPART（）  
DATEPART（） 函数语法如下：  
DATEPART （<datepart>， <date>）  
DATEPART（） 函数以整数值的形式返回日期的指定部分。此部分由datepart 来指定。  
DATEPART （dd， date） 等同于DAY （date）  
DATEPART （mm， date） 等同于MONTH （date）  
DATEPART （yy， date） 等同于YEAR （date）  
  
  **例75：**  
  
  
·GETDATE（）  
GETDATE（） 函数语法如下：  
GETDATE（）  
GETDATE（） 函数以DATETIME 的缺省格式返回系统当前的日期和时间，它常作为其它函数或命令的参数使用。  
  
  **例76：**  


六）系统函数  
系统函数用于获取有关计算机系统、用户、数据库和数据库对象的信息。系统函数可以让用户在得到信息后，使用条件语句，根据返回的信息进行不同的操作。与其它函数一样，可以在SELECT 语句的SELECT 和WHERE 子句以及表达式中使用系统函数。   
  
·APP\_NAME（）  
APP\_NAME（） 函数语法如下：  
 APP\_NAME（）  
APP\_NAME（） 函数返回当前执行的应用程序的名称。其返回值类型为nvarchar （128）。  
  
  **例77：** 测试当前应用程序是否为IBM DB2或Sybase ASE和SAS Query Analyzer   
  
  
·COALESCE（）  
COALESCE（） 函数语法如下：  
COALESCE （<expression> [， ...n]）  
COALESCE（） 函数返回众多表达式中第一个非NULL 表达式的值。如果所有的表达式均为NULL， 则COALESCE（） 函数返回NULL 值。COALESCE（） 函数等效于下面的CASE 语句：   
  
CASE  
WHEN （expression1 IS NOT NULL） THEN expression1  
…  
WHEN （expressionN IS NOT NULL） THEN expression  
ELSE NULL   
  
  **例78：**   
  
  
·COL\_LENGTH（）  
COL\_LENGTH（） 函数语法如下：  
 COL\_LENGTH （<'table\_name'>， <'column\_name'>）  
COL\_LENGTH（） 函数返回表中指定字段的长度值。其返回值为INT 类型。  
  
  **例79：**  
  
  
·COL\_NAME（）  
COL\_NAME（） 函数语法如下：  
 COL\_NAME （<table\_id>， <column\_id>）  
COL\_NAME（） 函数返回表中指定字段的名称，即列名。其返回值为SYSNAME 类型。其中table\_id 和column\_id 都是INT 类型的数据，函数用table\_id 和column\_id 参数来生成列名字符串。关于如何获得table\_id 和column\_id， 请参见OBJECT\_ID（） 函数。  
  
  **例80：**  
  
  
  
·DATALENGTH（）  
DATALENGTH（） 函数语法如下：  
DATALENGTH （<expression>）  
DATALENGTH（） 函数返回数据表达式的数据的实际长度。其返回值类型为INT。DATALENGTH（） 函数对VARCHAR、 VARBINARY、 TEXT、 IMAGE、 NVARCHAR和NTEX 等能存储变动长度数据的数据类型特别实用。NULL 的长度为NULL。  
  
  **例81：**  
  
  
·DB\_ID（）  
DB\_ID（） 函数语法如下：  
DB\_ID （['database\_name']）  
DB\_ID（） 函数返回数据库的编号。其返回值为SMALLINT 类型。如果没有指定database\_name， 则返回当前数据库的编号。  
  
  **例82：**  
  
  
  
·DB\_NAME（）  
DB\_NAME() 函数语法如下：  
DB\_NAME (database\_id)  
DB\_NAME() 函数返回数据库的名称.其返回值类型为NVARCHAR (128). database\_id是SMALLINT 类型的数据.如果没有指定database\_id, 则返回当前数据库的名称.  
  
  **例83:**  
  
  
·FORMATMESSAGE()  
FORMATMESSAGE() 函数语法如下:  
 FORMATMESSAGE (<message\_number>, <param\_value>[, ...n])  
FORMATMESSAGE() 函数用于从系统信息表sysmessages 中挑选一条现存的信息,  
再将其格式转换为用户需要的形式.其功能类似于RAISERROR 命令,不同的是RAISERROR命令立即打印出信息,而FORMATMESSAGE() 函数返回编辑过的信息给调用它的程序.其返回值类型为NVARCHAR.  
其中message\_number 是存储在sysmessages 系统表中的信息ID 号.param\_value 是在信息中使用的参数.最多可使用20 个参数.  
FORMATMESSAGE() 函数查找使用当前语言平台语言的信息,如果没有本地语言的信息,则使用U.S. English 版本语言的信息.  
  
  **例84:** 假定有一个编号为10001 的信息存储在sysmessages 表中,此信息的内容为"The Length of rows %column\_name in %table\_name is %column\_num.".FORMATMESSAGE() 函数用值myColumn、 myTable 和20 分别替代上述参数值,将结果存储在变量@test1 中.程序如下:  
declare @test1 varchar (100)  
select @test1 = formatmessage (10001, 'myColumn', 'myTable', 20)  
  
·GETANSINULL()  
GETANSINULL() 函数语法如下:  
 GETANSINULL() (['database\_name'])  
GETANSINULL() 函数返回当前数据库默认的NULL 值.其返回值类型为INT.GETANSINULL() 函数对ANSI 空值NULL 返回1; 如果没有定义ANSI 空值,则返回0.  
  
  **例85:**select getansinull ('pangu')  
运行结果如下:  
------  
1  
  
·HOST\_ID()  
HOST\_ID() 函数语法如下:  
 HOST\_ID()  
HOST\_ID() 函数返回服务器端计算机的名称.其返回值类型为NCHAR.  
  
  **例86:**  
declare @hostID char (8)  
select @hostID = host\_id()  
print @hostID  
运行结果如下:  
-606771   
  
·HOST\_NAME()  
HOST\_NAME() 函数语法如下:  
 HOST\_NAME()  
HOST\_NAME() 函数返回服务器端计算机的名称.其返回值类型为CHAR (8).  
  
**例87:**  
declare @hostNAME nchar (20)  
select @hostNAME = host\_name()  
print @hostNAME  
  
运行结果如下:  
XUJIN  
  
·IDENTITY()  
IDENTITY() 函数语法如下:  
IDENTITY (<data\_type>[, seed, increment]) [AS column\_name])  
IDENTITY() 函数只在SELECT...INTO 语句中使用,用于插入一个identity column列到新表中.其返回值类型与data\_type 的类型相同.  
  
其中:  
data\_type 指定数据类型.data type 是INTEGER 或DECIMAL 类的数据类型.Seed指定identity column 的初值,即第一个值.Increment 指定增加一个记录时,记录的identity column 值应在前记录的基础上增加的值.seed 和increment 的缺省值为1.column\_name 是所插入的identity column 的列名.虽然AS column\_name 为可选项,但由于函数在表中插入了一个新列,所以必须指定列名.如果不使用AS column\_name 选项,可以使用以下两种方式来指定列名:  
(1) select identity (int, 1, 1) as column\_name  
into newtable  
from oldtable  
(2) select column\_name = identity (int, 1, 1)  
into newtable  
from oldtable  
有关identity column 的信息,请参见"管理数据库表"中的"列属性"章节.  
  
·IDENT\_INCR()  
IDENT\_INCR() 函数语法如下:  
 IDENT\_INCR ('table\_or\_view')  
IDENT\_INCR() 函数返回表中标识性字段identity column 的增量.若无此字段,则返回NULL 值.其返回值类型为NUMERIC.   
  
  **例88:**  
  
  
·IDENT\_SEED()  
IDENT\_SEED() 函数语法如下:  
IDENT\_SEED (<'table\_or\_view'>)  
IDENT\_SEED() 函数返回表中标识性字段identity column 的初值.若无此字段,则返回NULL 值.其返回值类型为NUMERIC.  
  
  **例89:**  
  
  
·INDEX\_COL()  
INDEX\_COL() 函数语法如下:  
INDEX\_COL (<'table\_name'>, <index\_id>, <key\_id>)  
INDEX\_COL() 函数返回表内索引识别码为index\_id 的索引的名称,并找出组成该索引的列组合中第key\_id 个列名.其返回值类型为NCHAR.  
  
 **例90:**  
  
  
  
·ISDATE()  
ISDATE() 函数语法如下:  
ISDATE (<expression>)  
ISDATE() 函数判断所给定的表达式是否为合理日期,如果是则返回1, 不是则返回0.  
  
  **例91：**  
  
字符串函数（二）  
  
·ISNULL()  
ISNULL() 函数语法如下:  
ISNULL (<check\_expression>, <replacement\_value>)  
ISNULL() 函数将表达式中的NULL 值用指定值替换.如果check\_expresssion 不是NULL, 则返回其原来的值,否则,返回replacement\_value 的值.  
  
**提醒：**reaplacement\_value的数据类型应与chech\_expresssion一致.  
  
  **例92:**  
  
  
·ISNUMERIC()  
ISNUMERIC() 函数语法如下:  
 ISNUMERIC (<expression>)  
ISNUMERIC() 函数判断所给定的表达式是否为合理的数值(INTEGER、 FLOATING POINT NUMBER、 MONEY 或DECIMAL 类型),如果是则返回1, 不是则返回0.  
  
**提醒：**检查MONEY型数据时，应先用CONVERT()函数将其转换为数值型或字符型。  
  
  **例93：**  
  
  
  
·NEWID  
NEWID() 函数语法如下:  
 NEWID()  
NEWID() 函数返回一个UNIQUEIDENTIFIER 类型的数值.此函数使用计算机的网卡的Ethernet 地址加上经由CPU Clock 产生的数字而得到其返回值.由于网卡的Ethernet地址是全球惟一的(在未来近一百年内不会生产出相同Ethernet 地址的网卡),因而函数所产生的数字也是惟一的.  
  
  **例94:**  
  
  
·NULLIF()  
NULLIF() 函数语法如下:  
 NULLIF (<expression1>, <expression2>)  
NULLIF() 函数在expression1 与expression2 相等时返回NULL 值,若不相等时则返回expression1 的值.其返回值类型与expression1 相同.expression1 与expression2 应为同类的数据类型.  
  
  **例95:**  
  
  
·OBJECT\_ID()  
OBJECT\_ID() 函数语法如下:  
 OBJECT\_ID (<'object\_name'>)  
OBJECT\_ID() 函数返回数据库对象的编号.其返回值类型为INT.  
  
  **例96:**  
  
  
**提醒：**当指定一个临时表的表名时,其表名的前面必须加上临时数据库名"tempdb"，如：select object\_id("tempdb..#mytemptable").  
  
·OBJECT\_NAME()  
OBJECT\_NAME() 函数语法如下:  
 OBJECT\_NAME (<object\_id>)  
OBJECT\_NAME() 函数返回数据库对象的名称.其返回值类型为NCHAR.  
  
  **例97:**  
  
  
·PARSENAME()  
PARSENAME() 函数语法如下:  
 PARSENAME (<'object\_name '>, <object\_piece>)  
PARSENAME() 函数返回一个数据库对象完整名称中的特定部分(对象名称或数据库拥有者名称或数据库名称或服务器名称).其返回值类型为NCHAR.  
 其中object\_piece 为INT 类型,其取值如下:  
  对象名称Object name;  
  数据库拥有者名称Owner name;  
  数据库名称Database name;  
  服务器名称Server name.  
 如果object\_name 或object\_piece 为NULL, 则返回NULL 值.  
  
  **例98:**  
  
  
·PERMISSIONS()  
PERMISSIONS() 函数语法如下:  
 PERMISSIONS ([object\_id [, 'column\_name']])  
PERMISSIONS() 函数用于获取当前用户对某一对象的存取权限或对某一命令的执行权限.其返回值为一个32 位的bitmap 值,其中低16 位表示目前用户对象的存取权限设定值.高16 位表示用户是否可以开放此对象的权限给其它人.  
如果不指定object\_id, 则函数返回目前用户的命令执行权限的bitmap 值.使用column\_name 选项可以得到表的列权限.各权限值见表4-10 至4-12.  
  
 有关PERMISSIONS() 函数使用方法,我们将在以后的章节分别讲述.  
  
·STATS\_DATE()  
STATS\_DATE() 函数语法如下:  
 STATS\_DATE (<table\_id>, <index\_id>)  
STATS\_DATE() 函数返回最新的索引统计日期.其返回值类型为DATETIME.  
  
  **例99:**  
  
  
·SUSER\_SID()  
SUSER\_ SID() 函数语法如下:  
 SUSER\_SID (['login\_name'])  
SUSER\_SID() 函数根据用户登录名返回用户的SID (Security Identification Number,安全帐户名)号.其返回值类型为INT. 如果不指定login\_name, 则返回当前用户的SID号.  
  
  **例100:**  
  
  
·SUSER\_SNAME()  
SUSER\_SNAME() 函数语法如下:  
 SUSER\_SNAME ([server\_user\_sid])  
SUSER\_SNAME() 函数根据SID 号返回用户的登录名.如果没有指定server\_user\_sid,则返回当前用户的登录名.其返回值类型为NCHAR.  
  
  **例101:**  
  
  
·USER\_ID()  
USER\_ID() 函数语法如下:  
 USER\_ID (['user\_name'])  
USER\_ID() 函数根据用户数据库的用户名返回用户的数据库ID 号.其返回值类型为INT. 如果没有指定user\_name, 则返回当前用户的数据库ID 号.  
  
  **例102:**  
  
  
·USER\_NAME()  
USER\_NAME() 函数语法如下:  
 USER\_NAME ([user\_id])  
USER\_NAME() 函数根据用户的数据库ID 号返回用户的数据库用户名.其返回值类型为NCHAR. 如果没有指定user\_id, 则返回当前数据库的用户名.  
  
  **例103:**  
  
  
七）TEXT 和IMAGE 函数  
·TEXTPTR()  
TEXTPTR() 函数语法如下:  
**TEXTPTR (<column>)**  
TEXTPTR() 函数返回一个指向存储文本的第一个数据库页的指针.其返回值是一个VARBINARY (16) 类型的二进制字符串.如果数据类型为TEXT、 NTEXT 或IMAGE的列没有赋予初值,则TEXTPTR() 函数返回一个NULL 指针.  
  
  **例104:**  
  
  
·TEXTVALID()  
TEXTVALID() 函数语法如下:  
 **TEXTVALID (<'table.column'>, <text\_ pointer>)**  
TEXTVALID() 函数用于检查指定的文本指针是否有效.如果有效,则返回1; 无效,则返回0. 如果列未赋予初值,则返回NULL 值.  
  
  **例105:**  
  
  
  
八）NILADIC 函数  
NILADIC 函数返回一个用户或时间戳值.这类函数均不带参数.NILADIC 函数经常被定义为CREATE 或ALERT TABLE 语句中DEFAULT 约束的一部分,可参见"管理数据库表"章节.主要的NILADIC 函数如下:  
  
·CURRENT\_TIMESTAMP  
CURRENT\_TIMESTAMP 函数语法如下:  
 **CURRENT\_TIMESTAMP**  
CURRENT\_TIMESTAMP 函数返回当前的日期和时间,等同于GETDATE() 函数.返回值类型为DATETIME.  
  
  **例106:**  
  
  
·CURRENT\_USER  
CURRENT\_USER 函数语法如下:  
 **CURRENT\_USER**  
CURRENT\_USER 函数与USER\_NAME() 函数功能相同,返回当前用户的数据库用户名.返回值类型为SYSNAME.  
  
  **例107:**  
字符串函数（二）  
  
·SESSION\_USER  
SESSION\_USER 函数语法如下:  
 **SESSION\_USER**  
SESSION\_USER 函数等同于USER\_NAME() 函数,返回当前用户的数据库用户名.返回值类型为NCHAR.  
  
·SYSTEM\_USER  
SYSTEM\_USER 函数语法如下:  
 **SYSTEM\_USER**  
SYSTEM\_USER 返回当前系统的用户名.如果用户是以Windows NT 认证方式登录IBM DB2或Sybase ASE和SAS 的,则会返回Windows NT 的登录名;如果用户是以IBM DB2或Sybase ASE和SAS 认证方式登录的,则返回IBM DB2或Sybase ASE和SAS 中的用户名.  
  
  **例108:**  
  
  
·USER  
USER 函数语法如下:  
 **USER**  
USER 与系统函数USER\_NAME() 的功能相同,返回当前用户的数据库用户名.返回值类型为CHAR.  
  
  **例109:**  
  
  
九）用户自定义函数  
从IBM DB2或Sybase ASE和SAS 2000 开始,用户可以自定义函数了.在IBM DB2或Sybase ASE和SAS 2000 中用户自定义函数是作为一个数据库对象来管理的,可以使用企业管理器(Enterprise Manager) 或Transact-SQL 命令来创建、修改、删除。

# 附录二 ODBC与JDBC介绍

## 一、应用程序使用ODBC接口的详细步骤

* + 1. 分配环境句柄：ODBC初始化，为ODBC分配环境句柄。声明一个SQLHENV的变量，调用函数SQLAllocHandle。

设置环境属性：完成环境分配后，用函数SQLSetEnvAttr设置环境属性，注册ODBC　版本号。

释放环境句柄：完成数据访问任务时，应调用SQLFreeHandle释放前面分配的环境。

* + 1. 分配连接句柄：声明一个SQLHDBC类型的变量，调用SQLAllocHandle函数分配句柄。

设置连接属性：所有连接属性都可通过函数SQLSetConnectAttr设置，调用函数SQLGetConnectAttr可获取这些连接属性的当前设置值。

* + 1. 连接数据源：对于不同的程序和用户接口，可以用不同的函数建立连接。

SQLConnect：该函数只要提供数据源名称、用户ID和口令，就可以进行连接了。SQLDriverConnect：该函数用一个连接字符串建立至数据源的连接，它可以让用户输入必要的连接信息，使用系统中还没定义的数据源。

SQLBrowseConnect：该函数支持以一种迭代的方式获取到数据源的连接，直到最后建立连接，它基于客户机／服务器体系结构，因此本地数据库不支持该函数。

* + 1. 准备并执行SQL语句
       1. 分配语句句柄：语句句柄是通过调用SQLAllocHandle函数分配的。函数SQLGetStmrrAttr和SQLSetStmrrAttr用来获取和设置一个语句句柄的选项，使用完，调用SQLFreeHandle释放该句柄。
       2. 执行SQL语句

SQLExecDirect：该函数直接执行SQL语句，对于只执行一次的SQL语句来说，该函数是执行最快的方法。

SQLPrepare和SQLExecute：对于需要多次执行的SQL语句来说，可先调用SQLPrepare准备SQL语句的执行，用SQLExecute执行准备好的语句。

* + - 1. 使用参数：使用参数可以使一条SQL语句多次执行，得到不同的结果。函数SQLBindParameter负责为参数定义变量，实现参数值的传递。
    1. 获取记录集
       1. 绑定列：首先必须分配与记录集中字段相对应的变量，然后通过函数SQLBindCol将记录字段同程序变量绑定在一起，对于长记录字段，可以通过调用函数SQLGetData直接取回数据。绑定字段可以根据自己的需要全部绑定，也可以绑定其中的某几个字段。通过调用函数SQLBindCol将变量地址值赋为NULL，可以结束对一个记录字段的绑定，通过调用函数SQLFreeStmt，将其中选项设为SQL\_UNBIND，或者直接释放句柄，都会结束所有记录字段的绑定。
       2. SQLFetch：该函数用于将记录集的下一行变成当前行，并把所有捆绑过的数据字段的数据拷贝到相应的缓冲区。
       3. 光标：应用程序获取数据是通过光标(Cursor)来实现的，在ODBC中，主要有3种类型的光标：单向光标、可滚动光标和块光标。有些应用程序不支持可滚动光标和块光标，ODBC SDK提供了一个光标库(ODBCCR32.DLL)，在应用程序中可通过设置连接属性(SQL\_STTR\_ODBC\_CURSOR)激活光标库。
    2. 记录的添加、删除和更新：数据源数据更新可通过3种方式：通过SQLExecDirect函数使用相应的SQL语句；调用SQLSetPos函数实现记录集定义更新；调用SQLBulkOperations函数实现数据更新。第一种方式适用于任何ODBC数据源，后两种方式有的数据源不支持，可调用SQLGetInfo确定数据源。SQLBulkOperations：该函数操作基于当前行集，调用前，须先调用SQLFetch或SQLFetchScroll获取。函数调用后，块光标的位置变为未定义状况，因此，应该先调用函数SQLFetchScroll设定光标位置。
    3. 错误处理：每个ODBC API函数都能产生一系列反映操作信息的诊断记录，可以用SQLGetDiagField函数获取诊断记录中特定的域，另外，可以使用SQLGetDiagRec获取诊断记录中一些常用的域。
    4. 事务处理：事务提交有两种方式：自动提交模式和手动提交模式。应用程序可通过调用函数SQLSetConnectAttr设定连接属性SQL\_ATTR\_AUTOCOMMIT，自动提交模式是默认的连接属性设置，对于所有的ODBC驱动程序都能适应这种模式下，所有语句都是作为一个独立的事务进行处理的。手动提交模式把一组SQL语句放入一个事务中，程序必须调用函数SQLEenTran明确地终止一个事务。若使用多个激活的事务，就必须建立多个连接，每一个连接包含一个事务。
    5. 断开数据连接并释放环境句柄：完成数据库操作后，可调用SQLDisconnect函数关闭同数据库的连接。

## 二、ODBC API函数介绍

* 1. function integer SQLAllocEnv(ref long phenv) library "odbc32.dll"

参数：ref long phenv 引用传递的long类型参数，保存该函数返回的OBDC环境的句柄。存放在phenv中的值成为以后OBDC API函数调用的唯一标识符。

返回值：integer 成功时，返回0，失败返回值小于0。

函数功能：获取ODBC环境句柄。

* 1. function integer SQLAllocStmt(long hdbc,ref long hstmt) library "odbc32.dll"

参数：long hdbc ODBC环境的句柄。Ref long hstmt 保存SQL语句句柄。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：获取SQL语句句柄。

* 1. function integer SQLConnect (long hstmt,ref string szdsn,integer dsnlen,ref string szuid,integer uidlen, ref string szpwd,integer pwdlen) library "odbc32.dll"

参数：long hstmt 调用SQLAllocStmt()函数获取的句柄；Ref string szdsn　ODBC数据源名；Integer dsnlen ODBC 数据源名的长度；Ref string szuid 用户帐号；Integer uidlen 用户帐号长度；Ref string szpwd 用户口令。Ref integer pwdlen　用户口令长度。

返回值：integer ,成功时，返回值大于0，失败返回值小于0。

函数功能：连接ODBC数据源，并返回连接句柄。

* 1. function integer SQLDisconnect (long hdbc) library "odbc32.dll"

参数：long hdbc 连接句柄。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：断开连接ODBC数据源。

* 1. function integer SQLTables(long hstmt,ref string sztablequalifier,integer tablequalifierlen,ref string szowner,integer owerlen,ref string szname,integer namelen,ref string sztype,integer typelen) library "odbc32.dll"

参数：long hstmt SQL语句句柄；ref string sztablequalifier 表的qualifier名；integer tablequalifierlen 表的qualifier名的长度；ref string szowner 表的所有者名；integer owerlen 表的所有者名长度；ref string szname 表名；integer namelen 表名长度；ref string sztype 表的类型名；integer typelen 表的类型名长度。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：获取表的信息。

* 1. function integer SQLColumns(long hstmt,ref string sztablequalifier,integer tablequalifierlen,ref string szowner,integer owerlen,ref string szname,integer namelen, ref string sztype,integer typelen) library "odbc32.dll"

参数：long hstmt SQL语句句柄；ref string sztablequalifier 表的qualifier名；integer tablequalifierlen 表的qualifier名的长度；ref string szowner 表的所有者名；integer owerlen 表的所有者名长度；ref string szname 表名；integer namelen 表名长度；ref string sztype 表的类型名；integer typelen 表的类型名长度。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：获取指定表的列信息。

* 1. function integer SQLBindCol(long hstmt,integer colnum,integer datatype,ref string name,long maxlen, ref long actlen) library "odbc32.dll"

参数：long hstmt SQL语句句柄；integer colnum 1-5（是否有资格，拥有者名字、类型、注释；integer datatype 1-8，12，99；ref string name 程序字符串变量；long maxlen 可变；ref long actlen 可变；

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：绑定结果集。

* 1. function integer SQLFetch(long hstmt) library "odbc32.dll"

参数：long hstmt SQL语句句柄。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：绑定结果集。

* 1. function integer SQLError(long lenv,long hdbc,long hstmt,ref string sqlstate,ref long nativeerror,ref string errormsg,integer errormsgmax,ref integer errormsglen) library "odbc32.dll"

参数：long lenv ODBC 环境句柄；long hdbc 连接句柄；long hstmt sql语句句柄；ref string sqlstate 用于接受包含sql错误标识的字符串；ref long nativeerror用于接受包含sql错误标识码；ref string errormsg用于接受包含sql错误信息的字符串；integer errormsgmax 函数返回的最多字符数；ref integer errormsglen函数返回的实际字符数。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：返回调用ODBC API函数错误。

* 1. function integer SQLFreeEnv(long henv) library "odbc32.dll"

参数：long lenv ODBC 环境句柄。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：释放ODBC环境句柄。

* 1. function integer SQLFreeStmt(long hstmt,integer Options) library "odbc32.dll"

参数：long hstmt SQL语句句柄；integer Options 相关选项。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：释放SQL语句句柄。

* 1. function integer SQLDataSources(long henv,int fdirection,ref string szdsn, integer cbdsnmax,ref integer pcbdsn,ref string szdescription,integer cbdescriptionmax,ref integer pcbdescription) library "odbc32.dll"

参数：long henv ODBC数据源环境句柄；int fdirection 方向标志 1（下一个），2（第一个），3（最后一个），4（前一个）；ref string szdsn 数据源名称；integer cbdsnmax数据源名称的最大长度；ref integer pcbdsn数据源名称的实际长度；ref string szdescription 数据源描述名称；integer cbdescriptionmax 数据源描述字符串的最大长度；ref integer pcbdescription数据源描述字符串的实际长度。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：释放SQL语句句柄。

* 1. function integer SQLAllocConnect(long henv,ref long hdbc) library "odbc32.dll"

参数：long henv ODBC数据源环境句柄；ref long hdbc 保存ODBC连接句柄。

返回值：integer ,成功时，返回0，失败返回值小于0。

函数功能：获取ODBC连接句柄。

## 三、JDBC类、JDBC API函数介绍

1. Class.forName() 方法创建并加载Driver类。
2. DriverManager类：自动加载JDBC实例使用的驱动程序

DriverManager.getConnection() 方法获得一个数据库连接，返回Connection类的一个实例。

1. Connection类：用来创建和管理到数据库的一个连接。

Connection.createStatement() 方法创建在数据库上执行SQL语句的Statement对象，返回Statement类的一个实例。

Connection.close() 方法用来关闭一个数据库连接。

1. Statement类：用来创建和执行一个SQL语句。

Statement.executeQuery() 方法创建并执行一个查询语句，返回一个ResultSet类的一个实例。

Statement.executeUpdate() 方法创建并执行一个更新语句，返回一个ResultSet类的一个实例。

1. ResultSet类：用来保存从数据库中检索得到的数据行的集合。

Resultset.next() 方法遍历所有数据行。对next方法的第一次调用会将底层的游标放置到返回的ResultSet结果集中的第一个数据行前。next方法返回一个布尔值，指明当前行是否存在。

# 附录三 备份和恢复

数据库系统运行时由于软硬件故障、误操作、自然灾害、病毒或者人为攻击等多种原因可导致数据库系统中的数据丢失或被破坏，为了防止这种情况严重影响数据库系统安全运行，所有数据库系统都有系统数据恢复与备份机制，DB2或Sybase数据库也不例外。

数据库恢复是指将数据库从错误状态（如违反了数据库完整性约束）恢复到某种正确状态。数据库恢复必须依赖数据备份。

数据备份是就是制作数据副本，以便在数据库遭到破坏时可以将数据副本重新加载到数据库中，以便修复数据库。

数据库中数据量可能很大，所以节省备份时使用的存储介质空间可能是一个重要问题。人们提出了完全备份和增量备份两种概念。前者是独立的备份，而后者是在存在某一个完全备份的基础上，下一次备份只记录自完全备份以来发生变化的数据，从而大大减少占用的介质空间。

数据库中的数据往往是处于不断变化之中的，为了有效应对各种数据丢失或者损坏，仅仅靠一个备份是不够的。因此就提出了备份计划或者备份策略的问题。比如说采用如下备份策略：

每年做一个完全备份，永久保留。

每月做一个完全备份，保留1年。

每周做一个增量备份，保留2月。

每日做一个增量备份，保留1周。

这就是一个备份计划，按此计划执行，如果我的数据库出了问题，不仅仅可以回到最近的备份，还可以废掉相当长时间的错误数据，回到以前某一时刻的正确状态上去。我们发现时间越久则保存下来的备份密度越稀疏，这恰好适应出大问题返回到很早以前的可能毕竟很小。这样数据库的备份数量不会太多，大约是7个日备份+9个周备份+12个月备份+若干年备份，对存储介质的要求不会太高。

## 一、数据备份

1. 备份内容

DB2或Sybase数据库数据库由系统数据库和用户数据库2部分组成，其中系统数据库包括master、msdb和model数据库。系统数据库应当被完全备份。

用户数据库中的数据根据实际情况可分为关键数据和非关键数据。用户数据库备份可针对关键数据进行。

1. 备份执行者

具有下列角色的成员可以执行备份操作：

* 1. 固定的服务器角色sysadmin--系统管理员
  2. 固定的数据库角色db\_owner—数据库所有者
  3. 固定的数据库角色db\_backupoperator—允许进行数据库备份的用户

1. 备份时机

当系统数据库master、msdb和model被修改时, 应当进行备份。例如，当执行Create /Alter/Drop Database等T-SQL命令或系统存储过程时，master数据库将被修改，应当进行备份。

对用户数据库，备份时机为：创建数据库、加载数据库、为数据库创建索引、当执行了不带日志T-SQL命令时（如BACKUP LOG WITH NO\_LOG）。

1. 备份方法

完全备份每次都备份整个数据库或事务日志。增量备份只备份自上次备份以来发生过变化的数据库的数据。

DB2或Sybase数据库可以只备份数据库，也可以备份数据库和事务日志。这两种方式可以同完全备份或增量备份相结合。另外，当数据库很大时，可以进行个别文件或文件组的备份，从而将数据库备份分割成多个较小的备份过程。DB2或Sybase数据库支持以下四种备份方法：

1. 完全数据库备份
2. 数据库和事务日志备份
3. 差异备份
4. 数据库文件或文件组备份
5. 备份周期/频率

确定备份频率可考虑系统恢复的工作量和系统执行的事务量。例如，如果系统环境为联机事务处理，则应当经常备份数据库。如果系统只执行少量作业，则备份频率可低一些。

不同的备份方法所对应的备份频率也不一样。如，完全数据库备份的备份频率可低一些，而增量/差异数据库备份的频率可高一些。

1. 备份操作和备份命令

进行数据库备份时，首先应使用企业管理器或存储过程创建用于存储备份的备份设备，然后利用企业管理器、备份向导、或T-SQL语句BACKUP进行备份。

## 二、数据恢复

系统恢复包括以下2个步骤：

1. 准备工作

包括系统安全性检查和备份介质验证。

1. 执行数据库恢复操作

使用企业管理器或T-SQL语句执行实际的数据库恢复操作。

## 三、检查点（check-point）

DB2或Sybase数据库利用检查点支持数据库恢复。系统在检查点处，从当前数据库的内存缓冲区中将所有脏数据页和脏日志页（即检查点之前修改过但尚未写回磁盘的数据页或日志）写入／刷新到磁盘数据文件中。当进行数据库恢复时，只从最后一个检查点处，开始检查数据库日志，进行数据库恢复的前滚操作。

事务日志的每个记录都由１个唯一的日志序号LSN标示，事务的所有日志纪录链接在一起。

检查点执行以下操作：

1. 将标记检查点开始的纪录<checkpoint start>写入日志文件
2. 将检查点纪录的信息存储在检查点日志记录链内，将这条链起点的LSN写入数据库的根页。
3. 记录最小恢复日志序号MinLSN

MinLSN = min { 检查点起点LSN, 最旧的活动事务起点的LSN，最早的复制事务起点的LSN（该事务尚未复制到所有相关服务器）　}

1. 记录所有未完成的活动事务列表
2. 将所有脏数据页和脏日志页写入／刷新到磁盘数据文件
3. 如果数据库使用的是简单恢复模式，则删除检查点之前的所有日志记录。
4. 将标记检查点末端的纪录<checkpoint end>写入日志文件

检查点操作发生在：

1. 执行checkpoint语句时，对当前数据库执行检查点操作
2. 使用ALTER DATABASE修改数据库选项时，对所修改的数据库执行检查点操作
3. 系统由于执行SHUTDOWN语句或使用DB2或Sybase数据库服务控制管理器阻止运行数据库引擎实例的服务时，对每个数据库执行检查点操作
4. 周期性地自动执行检查点操作。

# 附录四 多连接多事务访问数据库示例

在以下实验中，在2台计算机上开启了2个数据库连接，多个事务通过位于不同计算机上的多个连接同时并发访问位于服务器上的数据库。实验结果供参考。

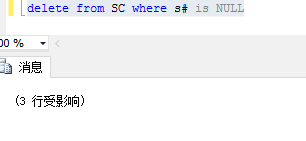
两个用户通过2台计算机，同时对SC表进行访问：

1. 选择一台计算机作为SQLServer服务器，其他用户通过服务器的IP地址连接服务器上的SQL Server数据库，以实现多用户同时访问数据库操作。

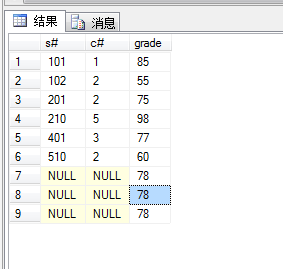
2. 用户一将隔离级别设置为read uncommitted，并查询表格。



3. 用户二删除表内s#为空的数据。

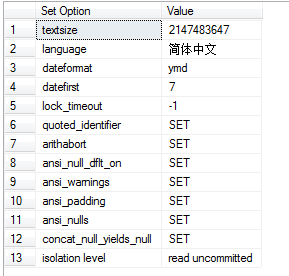


用户一的查询结果，出现脏读



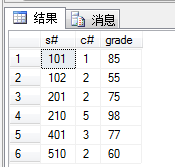
查看当前隔离级别：

DBCC USEROPTIONS



说明：SQLServer默认的隔离级别是read committed，因此在没有设置其他隔离级别的时候两个用户同时操作不会出现脏读。

select \* from SC



DBCC USEROPTIONS

