目录

[第一章 数据库入门 2](#_Toc491280824)

[1.1 使用数据库的原因 2](#_Toc491280825)

[1.2 数据库系统的概念 2](#_Toc491280826)

[1.3 Access工作台：第一部分——熟悉MS Access 3](#_Toc491280827)

[第二章 关系模型 3](#_Toc491280828)

[2.1 关系 3](#_Toc491280829)

[2.2 键的类型 4](#_Toc491280830)

[2.3 NULL值的问题 4](#_Toc491280831)

[2.4 函数依赖与规范化 4](#_Toc491280832)

[2.5 Access工作台：第二部分——在Access中操作多个表 5](#_Toc491280833)

[第三章 结构化查询语言 5](#_Toc491280834)

[3.1 示例数据库 5](#_Toc491280835)

[3.2 用于数据定义的SQL语句 5](#_Toc491280836)

[3.3 插入关系数据的SQL语句 8](#_Toc491280837)

[3.4 SQL关系查询语句 8](#_Toc491280838)

[3.5 修改和删除关系数据的SQL语句 12](#_Toc491280839)

[3.6 修改和删除表和约束的SQL语句 12](#_Toc491280840)

[3.7 SQL视图 13](#_Toc491280841)

[3.8 Access工作台：第三部分——在Access中使用查询 13](#_Toc491280842)

[3.9 SQL视图的创建 13](#_Toc491280843)

[3.10 SQL视图的使用 13](#_Toc491280844)

[3.11 Access工作台：在Access中使用视图 14](#_Toc491280845)

[第四章 数据建模与实体-关系模型 14](#_Toc491280846)

[4.1 需求分析 14](#_Toc491280847)

[4.2 实体-关系数据模型 14](#_Toc491280848)

[4.3 实体-关系图 15](#_Toc491280849)

[4.4. 开发E-R图示例 16](#_Toc491280850)

[4.5 Access工作台：第四部分——使用Access开发原型 17](#_Toc491280851)

[第五章 数据库设计 17](#_Toc491280852)

[5.1 把数据模型转换为数据库的设计方案 17](#_Toc491280853)

[5.2 使用关系模型表示实体 17](#_Toc491280854)

[5.3 关系的表示 18](#_Toc491280855)

[5.4 Heather Sweeney Designs公司的数据库设计 19](#_Toc491280856)

[5.5 Access工作台：第五部分——Access中的关系 19](#_Toc491280857)

[第六章 数据库管理 19](#_Toc491280858)

[6.1 Heather Sweeney Designs数据库 19](#_Toc491280859)

[6.2 控制，安全和可靠的必要性 20](#_Toc491280860)

[6.3 并发控制 20](#_Toc491280861)

[6.4 游标类型 22](#_Toc491280862)

[6.5 数据库安全 22](#_Toc491280863)

[6.6 数据库备份与恢复 24](#_Toc491280864)

[6.7 DBA的其他职责 24](#_Toc491280865)

[6.8 分布式数据库的处理 24](#_Toc491280866)

[6.9 对象-关系数据库 25](#_Toc491280867)

[6.10 Access工作台：第六部分——Access中的数据库管理 25](#_Toc491280868)

[第七章 数据库处理应用程序 25](#_Toc491280869)

[7.1 数据库处理的环境 25](#_Toc491280870)

[7.2 Web数据库处理 25](#_Toc491280871)

[7.3 数据库处理和XML 27](#_Toc491280872)

[7.4 Access工作台：第七部分——使用Microsoft Access进行Web数据库处理 27](#_Toc491280873)

[第八章 商业智能系统的数据库处理 27](#_Toc491280874)

[8.1 商业智能系统 27](#_Toc491280875)

[8.2 可操作系统与BI系统的关系 27](#_Toc491280876)

[8.3 报表系统和数据挖掘应用程序 27](#_Toc491280877)

[8.4 数据仓库和数据集市 28](#_Toc491280878)

[8.5 报表系统 28](#_Toc491280879)

[8.6 数据挖掘 29](#_Toc491280880)

[8.7 Access工作台：第八部分——使用Access的商业智能系统 29](#_Toc491280881)

**第I部分 数据库基础**

# 第一章 数据库入门

## 1.1 使用数据库的原因

1.1.1 关于列表的问题

•电子表格或列表只能用来表示和某一事物直接相关的信息，无法便捷处理多重事物关系的情况。p4/21

1.1.2 使用关系数据库表

•表和电子表格（或称工作表）直观结构类似。但是表有列名，工作表只有字母坐标，而表没有自带的行号。

1.1.3 关系表的处理

## 1.2 数据库系统的概念

•数据库系统包括四个组成部分：用户，数据库应用程序，数据库管理系统（database management system, DBMS）和数据库。

•DBMS用以接收SQL请求，并将这些请求转化为数据库中的操作。DBMS是一个庞大且复杂的程序，由专门的软件公司开发。

•数据库应用程序是作为用户和DBMS之间媒介的一个或多个计算机程序。是向DBMS传送SQL语句的程序，也是向用户显示返回的数据的载体。说白了它就是使用了数据库的应用程序，开发目的与功能完全随意。

1.2.1 数据库

•数据库是关联记录的自描述集合。现在几乎所有的数据库都是关系型数据。

•自描述（self-describing）：数据库本身含有对数据库结构的描述。

•元数据（metadata）：关于数据库结构的数据。表名，列名和列所属的表，表和列的属性等都是元数据。

•数据库的内容：用户数据，元数据，索引和其他改进数据库性能的结构，应用元数据。

1.2.2 DBMS与数据库应用程序

•DBMS的目的是创建，处理和管理数据库。

•常用的DBMS：Oracle公司：Oracle，MySQL；IBM公司：DB2；Microsoft公司：SQL Server，Microsoft Access。

•DBMS的功能：

①创建数据库，创建表，创建支持结构（例如索引，便于查找）

②读取数据库数据，增删改查，维护数据库结构

③执行规则（也称为参照完整性约束，referential integrity constraints）

④并发控制（保证用户之间工作互不干扰），提供安全性（访问级别），备份与恢复

•应用程序的功能：创建并处理表单，处理用户查询，创建并处理报表，执行应用程序逻辑，控制应用程序。

•表单（form）是应用程序隐藏并整合底层表结构之后，显示对用户有效的格式的信息。比如输入表单，即一个给用户使用的为数据库增加信息的表格，表格的格式显然是根据用户需要而设计的，与表的结构无关。而表（table）是数据库中底层实现的数据结构，前面提及的类似excel工作表的结构。DBMS是一个程序或软件，在这里可以看到表的结构，而应用程序中可以看到封装并整合的表单。

•报表（report）和表单类似，是一个格式化后的查询结果。格式由用户需要而定。

1.2.3 个人数据库系统和企业级数据库系统

•个人数据库系统：简言之是使用范围很小，用户群体不大的数据库系统。这样的数据库系统可以将DBMS和应用程序合二为一，不必区分。比如MS Access就是针对个人数据库系统的。

•企业级数据库系统：规模巨大，用户众多，应用程序众多，全天候工作，备份困难，数据库载体（服务器）众多且不临近。

•数据库服务器：一些存放数据库内容的磁盘或专用计算机。因规模较大，必须独立集中存放和管理，所以对其的连接一般都是远程网络访问的。

## 1.3 Access工作台：第一部分——熟悉MS Access

# 第二章 关系模型

## 2.1 关系

•在DBMS中，确切的说，数据以关系（relation）的形式存储，关系是一种特殊类型的表。具体而言是一个由行和列组成的二维表，且有以下特征：

①表的每行存储了某个实体或其部分数据，顺序任意

②表的每列包含了用于表示实体某个属性的数据

③表中每个单元格都只包含一个值，不能含有多个条目

④任意一列所有条目类型必须一致

⑤每列都必须有唯一名称，但顺序任意

⑥任意两行不能有完全相同的数据值（但实际中查重非常耗时，因此这个条件经常被忽略）

2.1.1 关系表与非关系表示例

2.1.2 显示关系表结构要注意的问题

2.1.3 术语说明

|  |  |  |  |
| --- | --- | --- | --- |
| 标准术语 | 表 | 行 | 列 |
| 其他称呼组合（传统数据处理） | 文件 | 记录 | 字段 |
| 其他称呼组合（数据库理论） | 关系表 | 元组（tuple） | 属性（attribute） |

•上表中同一列术语等价。

## 2.2 键的类型

•键（key）是表中用来标识行的一列或多列。键可以唯一也可以不唯一。表不同于excel工作表，虽然列名唯一，但没有固有的行号。键就是被用来作为“行号”的列，可以是单列键，也可以是多列组成的复合键。

2.2.1 复合键

•复合键也可以唯一可不唯一。

2.2.2 候选键与主键

•候选键：可唯一标识表中每一行的键。可以是单列键或复合键。可以有多个候选键。

•主键（primary key）：被选用为唯一标识表中每一行的候选键。主键只有一个。主键习惯用下划线表示书写。

2.2.3 代理键

•代理键：具有DBMS分配的唯一标识符的列，该标识符已经作为主键添加到表中。每次创建新行时由DBMS分配代理键的唯一值，且分配好的值永远不改变。即由DBMS自行生成和管理的主键，一般不具有实际意义（生成报表时可省略），只是一个区分代号。

2.2.4 外键与参照完整性

•外键（foreign key）：将一个表的主键放入另一个表来表示关联，在第二个表中，该属性称为外键。而第二个表中外键的属性名可以和第一个表中的主键不同，但是数据的值和类型必须一致（才能关联）。外键习惯用斜体字书写。

•参照完整性约束（referential integrity constraint）：外键的取值必须在其对应主键的表中有对应项。

## 2.3 NULL值的问题

•NULL值即属性值为空的情况，在DBMS中可以设置是否允许其存在，否则必须为每个单元格赋值。NULL值的存在会产生很多不确定问题以及很严重的问题。

## 2.4 函数依赖与规范化

2.4.1 函数依赖（functionally dependent）

•函数依赖：一个值通过函数公式关系依赖于另外一个值，如：当y=f(x)，则x→y；或当z=f(x,y)，则(x,y)→z。则称x决定y，或z函数依赖于x和y。箭头左边的变量称为决定因子（determinant）。箭头右边也不一定只有一个值，如z→(x,y)，即z决定x和y。

•显然函数依赖关系是用于关系型数据库设计的基本参照。

2.4.2 再论主键与候选键

•主键新定义：表中一个或多个可以通过函数决定其他所有属性的属性。候选键也有这样的类似定义。之所以选择某个候选键成为主键，取决于常识性的最有意义的或最重要的一组依赖关系。

2.4.3 规范化（normalization）

•规范化：将一个具有多个主题的表分割为一组表，使每个表只有一个主题。

•更多理论书籍：

①David M. Kroenke and David J. Auer, Database Processing: Fundamentals, Design and Implementation, 第11版。

②C. J. Date, An Introduction to Database Systems, 第八版

2.4.4 表的设计原则

•设计结构良好的表的规则，也是规范化的核心：

①每个决定因子都必须是候选键，即每个独立的函数依赖关系都应该构建一个单独的表

②非结构良好的表应分解为两个或更多结构良好的表

•满足以上两个设计规则，可以避免大多数与非规范化表有关的问题，虽然仍然有小概率问题存在。

•范式（normal form）：结构良好的表总是满足Boyce-Codd范式（BCNF），只具备关系表特征的称为满足第一范式（first normal form, 1NF）。

**范式**p235/252

范式的定义有很多，发展历史比较复杂。现当下的结论是，使用中期发现的BCNF范式可以解决几乎所有问题。而对于特例——多值依赖，可以采用扩展定义的BCNF范式解决。BCNF范式的含义是保证函数依赖的每个决定因子都是候选键，对于多值依赖只需将多值映射的两者或多者组成一个复合键来考虑即可（可以参考N:M关系的处理）。

2.4.5 规范化过程

•规范化过程：

①标识表的所有候选键

②标识表中的所有函数依赖

③检查函数依赖的决定因子。如果某决定因子不是候选键，则表的结构就不好。则：

a.把函数依赖的列放在它自己的新表中

b.把函数依赖的决定因子作为新表主键

c.将决定因子的副本作为原表中的外键

d.在新表和原表之间创建参照完整性约束

④根据需要，多次重复步骤③，直至每个表的决定因子都是候选键。

2.4.6 规范化的示例

•多值依赖（multi-value dependency）是一种比较复杂和高级的关系，更多内容借助高级书籍

## 2.5 Access工作台：第二部分——在Access中操作多个表

# 第三章 结构化查询语言

•SQL（structured query language）并不是完全的编程语言，是一种数据子语言（sublanguage），仅包括定义和处理数据库的结构。要成为完全的编程语言，必须嵌入脚本语言（如VBScript）或编程语言（Java等），也可以使用支持DBMS的命令提示交互式地提交SQL语句。

•SQL是面向文本的，其出现远远早于图形用户界面，现在的DBMS提供的图形工具只能完成部分SQL可以完成的任务，而且必须在程序代码中使用SQL动态生成SQL语句。

•SQL命令分为几大类，最重要的两类：用于定义数据库结构的数据定义语言（data definition languange，DDL），用于查询修改数据库数据的数据操作语言（data manipulation language，DML）。

## 3.1 示例数据库

•SQL的标准：SQL-92（或ANSI-92 SQL）；SQL:1999（或SQL3），SQL:2003，SQL:2006，SQL:2008。较早的版本比如ANSI-89 SQL不建议使用，现在常见的SQL特性从92版开始。

•每个DBMS实现SQL的方式稍有不同，同样目的的SQL语句跨DMBS平台时需要调整，可以参考DBMS的帮助文档。

## 3.2 用于数据定义的SQL语句

**命名规则**

•Table和Column的名字必须以字母开头，最多30个字符长，可以包含数字字母和underscore。

•表名是全部大写字母，单词之间用下划线；列名是首字母大写。如：  
RELATION\_NAME (Column01, Column02, …, LastColumn)

•当表列的名称有歧义时，即有多个表列名称一致但是同时出现于同一语句中，此时可以使用TABLENAME.ColumnName的形式进行限制。

•所有的SQL语句中的字符类型（Char等）和时间类型要用单引号，整数和Numeric等数字不需要。特别的，SQL只认可无方向引号。某些软件中，对于日期必须放在#...#号之间。

**常用的数据类型**

（根据不同DBMS有区别，表中例子为SQL Server）

|  |  |
| --- | --- |
| Char(size) | 固定长度的字符串，输入长度不足时用空格填满，size不能超过255 |
| VarChar(size) | 可变长度字符串，占据空间始终和输入长度一致，size表示最大长度 |
| Int | 整数 |
| Numeric(size,d) | 数值，size表示数值最大位数digit，d表示小数点后最大位数，size不包括小数点所占的一位 |
| DateTime | 日期和时间 |

•注意：

①VarChar类型需要额外处理，DBMS需要额外几个字节来存储这些值的长度。因此适当时候仍需优先选用Char类型。

②SQL中数据类型包括后面的括号

③更加完整常用数据类型表 p98/115

**Creating Tables**

•一般格式：create table

CREATE TABLE TABLENAME

(ColumnName1 datatype1 [constraint],

ColumnName2 datatype2 [constraint],

optional table constrains

etc.);

•列约束的可选值与关键字：

(1)UNIQUE：在这个列中，不能出现两个相同的值

(2)NOT NULL：此列中不能为空

(3)NULL：允许空值，创建行时可以不赋值

(4)DEFAULT：用于设定初始值（DEFAULT是否可用和用法取决于DBMS）

(5)PRIMARY KEY：主键，可以在列约束添加，但建议在尾部表约束添加

(6)CHECK：见ALTER TABLE部分，建议在表约束添加

•表约束举例：具体约束名称的习惯见后面各节的例子

(1) CONSTRAINT ConstraintName UNIQUE(ColumnName)

(2)

(3)

(4) CONSTRAINT ConstraintName CHECK(conditions)

(5) CONSTRAINT ConstraintName PRIMARY KEY(ColumnName)

(6) CONSTRAINT ConstraintName FOREIGN KEY(ColumnName)

REFERENCES TABLENAME(ColumnName)

ON UPDATE CASCADE

ON DELETE NO ACTION

举例1：SQL Server

CREATE TABLE EMPLOYEE(

EmployeeNumber Int PRIMARY KEY,

FirstName Char(25) NOT NULL,

LastName Char(25) NOT NULL,

Age Char(3) NULL,

Department Char(35) NOT NULL DEFAULT ′Human Resources′,

Email VarChar(100) NOT NULL UNIQUE, //使用多个列约束

…);

3.2.1 使用表约束定义主键

•定义方法：和往常一样定义表的列，但必须为即将设为主键的列提供列约束NOT NULL，在列定义的最后使用表约束CONSTRAINT来创建主键。

举例2：SQL Server

CREATE TABLE EMPLOYEE(

EmployeeNumber Int NOT NULL IDENTITY (1,1),

FirstName Char(25) NOT NULL,

LastName Char(25) NOT NULL,

Age Char(3) NULL,

Department Char(35) NOT NULL DEFAULT ′Human Resources′,

Email VarChar(100) NOT NULL UNIQUE,

CONSTRAINT EMPLOYEE\_PK PRIMARY KEY(EmployeeNumber)

);

//IDENTITY用于SQL Server中定义代理键，第一个数字表示初值，第二个数字表示增量

•约束名称EMPLOYEE\_PK是任意的但必须唯一，但是有约定习惯TABLENAME\_PK

•优点：只能使用这种方法定义复合键主键，并且添加约束名称方便管理数据库，最后可以方便的在一些DBMS中定义代理键

3.2.2 使用表约束定义外键

•可以使用表约束来定义外键及其参照完整性约束，用来实现一些业务规则

举例3：SQL Server

CREATE TABLE EMPLOYEE(

EmployeeNumber Int NOT NULL IDENTITY (1,1),

FirstName Char(25) NOT NULL,

LastName Char(25) NOT NULL,

Age Char(3) NULL,

Department Char(35) NOT NULL DEFAULT ′Human Resources′,

Email VarChar(100) NOT NULL UNIQUE,

CONSTRAINT EMPLOYEE\_PK PRIMARY KEY(EmployeeNumber)，

CONSTRAINT EMP\_DEPART\_FK FOREIGN KEY(Department)

REFERENCES DEPARTMENT(DepartmentName)

ON UPDATE CASCADE

ON DELETE NO ACTION

);

•外键定义属性Department为外键，关联主键为DEPARTMENT表中的属性DepartmentName，ON UPDATE表示在数据更新时进行的操作，CASCADE表示级联处理，即如果主键变化外键随着变化。ON DELETE NO ACTION是默认设置，可不写，表示禁止删除和外键关联的主键（取相同的值的主键）。

3.2.3 向DBMS提交SQL语句

## 3.3 插入关系数据的SQL语句

**Insert into语句**

•一般格式：指定列

INSERT INTO TABLENAME

(firstcolumn, …, lastcolumn)

VALUES (firstvalue, …, lastvalue)

•一般格式：全部列

INSERT INTO TABLENAME

VALUES (…)

举例1：

INSERT INTO employee

(first, last, age, address, city, state)

VALUES (′Luke′, ′Duke′, 45, ′2130 Boars Nest′, ′Hazard Co′, ′Georgia′);

•对于代理键无需提供数值，对于未指定的列则赋空值（但必须是允许NULL值的列）

•指定列赋值时，列名顺序须与值的顺序一致但是可以与原表中不一致

## 3.4 SQL关系查询语句

3.4.1 SQL SELECT/FROM/WHERE架构

3.4.2 从单个表中读取指定列

3.4.3 从单个表中读取指定行

3.4.4 从单个表中读取指定行和指定列

**SELECT语句**

•一般格式：

SELECT [ALL | DISTINCT] Column1, Column2, etc //选取所有列时，可以用SELECT \*

FROM TABLENAME1, TABLENAME2

[WHERE condition]

[GROUP BY columnlist]

[HAVING conditions]

[ORDER BY columnlist [ASC | DESC]];

•ALL/DISTINCT用来指示是否忽略相同或重复的记录，即从中只选取一个，ALL是默认的

•SELECT语句返回的结果始终是一个表，即便有时只有一个值。SELECT关键字后的列名顺序决定了结果表中列的顺序。

•WHERE语句(where clause)后接一个条件表达式，多个条件表达式之间可用AND和OR连接，其中表达式中可选的条件关系：=, >, <, >=, <=, <>或!=(不等于), LIKE。LIKE的使用方法和关系符是一样的，其值中可以使用“%”代替任意长度的任意字符。使用LIKE进行条件判断时，对于字符是case sensitive的，而其他的比较运算符使用默认的WHERE语句时是case insensitive的，但可以使用WHERE BINARY打头的方式使之变为case sensitive。

•WHERE和GROUP BY子句一起使用时，WHERE语句须置前，产生的效果也居前。否则会有歧义。

•GROUP BY clause顾名思义用来根据选取的column将条目分组，即被指定的column中具有相同值的条目分为一组，一般都和aggregate function配合使用，即可指定每个函数应用的范围

•HAVING clause要用必须和GROUP BY一起使用，是一个限定条件，和WHERE的功能类似，因为置于GROUP BY之后，可以理解为对分组之后的结果进行限定，不满足的结果无法被SELECT

•ORDER BY clause即用来将结果进行排序的指令。可以指定多个column作为排序依据，每个column都可以指定升序或降序，以逗号相隔。结果是先按照居前的指定列进行排序，在前列取值相同的多个行中，使用居后的指定列进行排序。ASC升序是默认设定。

举例1：（选取3列，其中first列以Er为首的条目）

SELECT First, Last, City

FROM EMPINFO

WHERE First LIKE ′Er%′; //每个列的char型值要用单引号包围，number型值不需要

举例2：(选取全列，其中first列为Eric的条目)

SELECT \* FROM EMPINFO

WHERE First = ′Eric′;

3.4.5 WHERE子句中的范围，通配符和空值

3.4.6 对查询结果排序

**条件语句详解**

WHERE等条件语句后面可带的内容与格式类型：

①条件表达式

•含有条件关系符=, >, <, >=, <=, <>或!=(不等于), LIKE的式子

举例：WHERE First LIKE ′%s′

②多个条件表达式的连接

•使用AND或OR

举例：WHERE First = ′s′ AND Last = ′s′

③IN结构

•可以改写成OR连接的多条件式，IN结构更加简便，当取括号中枚举值中的任一值时，条件成立；也有NOT IN结构，用法与功能可自推

•IN后面的括号内（不包括括号）可以看成是一个单列的表，具体理解参考子查询内容

举例：WHERE LastName IN (′Hernandez′, ′Jones′, ′Ruiz′)

④BETWEEN结构

•可以改写为AND多条件式，表示取值在范围内（包括边界）的时候，条件成立；NOT BETWEEN的方法可自推

举例：WHERE Age BETWEEN 30 AND 40

⑤LIKE与通配符

•SQL中有两种通配符，下划线“\_”与百分号“%”。前者代表单个未指定字符，后者代表任意数量的未指定字符。可以使用n个连续下划线代表n个未指定字符。

•LIKE可以和NOT一起使用，NOT LIKE表示反面。

•Access中不使用SQL-92标准通配符，而是MS-DOS的通配符。即”?”代表单个字符，”\*”代表多个字符。

举例：WHERE Department NOT LIKE ′%ing′;

⑥IS NULL

•用于在WHERE子句中查找NULL值，反之可以用IS NOT NULL

举例：WHERE Phone IS NULL;

3.4.7 SQL内置函数和计算

3.4.8 内置函数和分组

**内置函数Aggregate Functions与算数计算**

•SELECT选取的内容可以不只是column，也可以是aggregate functions的返回结果或标准的算数计算结果

•Aggregate Functions：

|  |  |
| --- | --- |
| MIN(…) | return the smallest value in any given column |
| MAX(…) | return the largest value in any given column |
| SUM(…) | return the sum of the numeric values for any given column |
| AVG(…) | return average values of any given column |
| COUNT(…) | return the number of rows in any given column |
| COUNT(\*) | return the number of rows in the table |

•内置函数返回的结果往往没有列名，可以使用AS关键字赋予。AS关键字也可以为已有列名的列在SELECT语句中赋予一个别名来显示（是仅仅在当前SELECT语句返回的表中有效还是彻底更改了列名暂不知）。若一个SELECT语句包含多个内置函数（以逗号相隔），在每个内置函数返回列之后都可以用一个AS为其命名。

•一般内置函数和算数计算不能用于WHERE子句，如果作为条件，可使用于HAVING子句；如果想将内置函数或算数计算的结果（实际上最终也显示为列，并且可赋予名称）用于WHERE子句，可以使用视图功能。即无法在同一个指令中将内置函数或算数计算结果用于WHERE子句。

•一般普通列名不能和内置函数一起使用，除非是GROUP BY子句指定的列。可以理解为，SELECT无法一次返回一个每个列的行数不相等的“锯齿形”表。

•COUNT函数可用于任何数据类型，但其他四个函数只能针对整型，数值型和其他数字类型进行操作。

•COUNT函数中在列名前加DISTINCT，如COUNT(DISTINCT Department)，返回不重复的行数统计结果。

举例3：（选取两列，根据Dept列分组并将Salary列取平均，显示平均值大于2万的条目）

SELECT Dept,avg(Salary)

FROM EMPLOYEE

GROUP BY Dept

HAVING avg(Salary)>20000;

举例4：（选取所有行数的一个1\*1列，列名为NumberOfProjects）

SELECT COUNT(\*) AS NumberOfProjects

FROM PROJECT;

举例5：（选取一个由MaxHours列运算获得的列，并命名）

SELECT (18.50\*MaxHours) AS MaxProjectCost

FROM PROJECT;

•以下是SQL中常用的算数计算函数，但是不一定每个DBMS都支持

|  |  |
| --- | --- |
| +, -, \*, /, % | 加减乘除，取余（Modulo） |
| ABS(x) | returns the absolute value of x |
| SIGN(x) | returns the sign of x as -1, 0 or 1 |
| MOD(x, y) | modulo, same as x%y |
| FLOOR(x) | returns the largest integer value <= x |
| CEILING(x) or CEIL(x) | returns the smallest integer value >= x |
| POWER(x, y) | returns the value of x raised to the power of y |
| ROUND(x) | returns the value of x rounded to the nearest whole integer |
| ROUND(x, d) | 返回近似小数值，小数点后位数为d |
| SQRT(x) | 返回平方根 |

3.4.9 使用子查询处理多个表

**子查询和语句嵌套**

•SELECT语句返回的一定是一个表，虽然其行列数可能为1，而IN结构后接的内容可以是一个单列，因此可以使用SELECT语句嵌套其中

•子查询使用在SELECT中，因此子查询嵌套可以进行多层

举例6：选取工作时间大于50的员工姓名，姓名存于EMPLOYEE中，而工时存于ASSIGNMENT

SELECT FirstNAme,LastName

FROM EMPLOYEE

WHERE EmployeeNumber IN(

SELECT DISTINCT EmployeeNumber

FROM ASSIGNMENT

WHERE HoursWorked > 50);

3.4.10 使用连接查询多个表

3.4.11 SQL JOIN…ON语法

3.4.12 外连接

**Table Join连接**

•当从两个或多个表中SELECT数据时，即可视为连接。连接的条件由WHERE子句表明，一般都是一对外键和主键的相等关系。当有两个以上的表时，在WHERE子句中用AND连接多个连接条件。

•因为连接之后的表存在一对多的映射关系，所以也可以使用GROUP BY和内置函数。实际上，表连接可以不当成特殊用法，其用法和单个表的SELECT操作在相同原则下一致。

•返回结果也可以使用ORDER BY

举例7：第一种写法，非标准写法

SELECT FirstName, LastName, SUM(HoursWorked) AS TotalHoursWorked

FROM EMPLOYEE AS E, ASSIGNMENT AS A

WHERE E.EmployeeNumber = A.EmployeeNumber

GROUP BY LastName, FirstName;

//AS关键字也可以用来为表赋予别名，以方便书写；这种情况下，也可以省略AS，直接FROM EMPLOYEE E来为表赋予别名。

举例8：第二种写法，ANSI SQL标准

SELECT ProjectName, FirstName, LastName, HoursWorked

FROM (EMPLOYEE AS E JOIN ASSIGNMENT AS A

ON E.EmployeeNumber = A.EmployeeNumber)

JOIN PROJECT AS P

ON A.ProjectID = P.ProjectID

ORDER BY P.ProjectID, A.EmployeeNumber;

//连接三个表的写法如上

•以上提供的两种写法都是内连接或同等连接（equijoin），即必须满足连接条件才会被SELECT选中。有的DBMS对这种连接要求采用关键字INNER JOIN来代替上面的JOIN，或举例8中的括号可以省略。

•外连接用于连接表时，不满足连接条件的行也能被显示用于提供信息的情况（即可以看到哪些行没有匹配到）。虽然SQL-92标准中不包含它，但是大多数DBMS支持。只是不同的DBMS语法不一。

举例9：外连接

SELECT ProjectName, EmployeeNumber, HoursWorked

FROM PROJECT LEFT JOIN ASSIGNMENT

ON PROJECT.ProjectID = ASSIGNMENT.ProjectID;

//即便没有ASSIGNMENT中对应的ProjectID值，PROJECT表的指定列也会全部选中

•外连接分为左右两种，关键字为LEFT JOIN和RIGHT JOIN，用法是对称的。使用LEFT JOIN时，位于关键字左侧的表即便不满足连接条件也会被选中，显示结果中不满足连接条件的行中其他表的列值为NULL。换句话说，位于左侧的表的指定列无论是否匹配都会全部显示。

## 3.5 修改和删除关系数据的SQL语句

3.5.1 修改数据

**Update语句**

•一般格式：

UPDATE TABLENAME

SET ColumnName = newvalue, AnotherColumn = newvalue2, …

WHERE conditions

•注意如果不加WHERE条件限制，则所有的指定列的值全部会被改变。

3.5.2 删除数据

**Delete语句**

•一般格式：

DELETE

FROM TABLENAME

WHERE conditions

•注意如果不加WHERE条件限制，则全表会被删除。

## 3.6 修改和删除表和约束的SQL语句

**DROP TABLE**

3.6.1 DROP TABLE语句

•一般格式：drop table

DROP TABLE TABLENAME

•drop table相当于删去整个table的定义和内容，而delete只是删去条目，table的定义框架仍然存在

•drop table和delete操作都受到参照完整性约束限制

**ALTER TABLE**

3.6.2 ALTER TABLE语句

•ALTER TABLE语句是用来修改table的一些配置的

举例1：取消名为ASSIGN\_EMP\_FK的约束（外键约束/参照完整性约束）

ALTER TABLE ASSIGNMENT

DROP CONSTRAINT ASSIGN\_EMP\_FK;

3.6.3 CHECK约束

举例2：添加约束，ADD后的语法和创建表的表约束相同

ALTER TABLE PROJECT

ADD CONSTRAINT PROJECT\_Check\_Dates

CHECK(StartDate<EndDate);

•CHECK约束的意义比较直观，即表列必须满足括号内的条件。条件表达式的用法与选择和WHERE是一样的，详见前文条件语句详解

举例3：添加列

ALTER TABLE PROJECT

ADD CurrentTotalHours Numeric(8,2) NULL;

•添加列的格式和create table的格式顺序类似，但是注意不能直接添加NOT NULL的列。因为添加后，该列必然没有值，而其他列因为已经建立往往有值。如果需要NOT NULL约束，可以先创建NULL列，赋值之后再行更改约束。

举例4：修改列属性

ALTER TABLE PROJECT

ALTER COLUMN CurrentTotalHours Numeric(8,2) NOT NULL;

举例5：删除列

ALTER TABLE PROJECT

DROP COLUMN CurrentTotalHours;

•ALTER TABLE也可以用于修改数据类型，但是必须小心，容易造成数据丢失，应参考DBMS文档。

## 3.7 SQL视图

## 3.8 Access工作台：第三部分——在Access中使用查询

## 3.9 SQL视图的创建

**SQL视图**

•SQL视图是SQL的功能，并非DBMS或软件功能。它是一个从其他表和视图中构建的虚拟表。定义视图后，就可以像表一样使用视图。

•视图就是通过SELECT语句创建的虚拟表。但是该SELECT语句不能含有ORDER BY子句。如果想对视图的内容排序，只能再次用SELECT语句从视图中读取并排序。

•一般格式：创建视图

CREATE VIEW ViewName AS

SELECT语句

举例1：

CREATE VIEW EmployeePhoneView AS

SELECT FirstName, LastName, Phone AS EmployeePhone

FROM EMPLOYEE;

•有的DBMS不能识别带分号的CREATE VIEW语句

•可以使用ALTER VIEW语句修改视图，使用方法和CREATE VIEW ViewName AS相同。相当于使用新的视图定义来覆盖原来的。

•一般格式：删除视图

DROP VIEW(ViewName)

## 3.10 SQL视图的使用

3.10.1 用视图隐藏列和行

3.10.2 用视图显示计算列的结果

3.10.3 用视图隐藏复杂的SQL语法

3.10.4 计算和内置函数的分层

•SQL Server要求创建视图时，于SELECT语句中，在内置函数或GROUP BY子句中使用在SELECT语句第一部分指定的所有列。虽然有这样的要求，但是只有GROUP BY之后第一个列有作用。

•视图实际上就相当于SQL自带的对自身的封装。将一个特别的SELECT查询封装为一个视图，之后再次调用便十分轻便。还可以理解为将一次SELECT的返回结果（一个表）存储起来并且赋予一个名称，成为一个新“表”。新表中将原来本是内置函数或算数计算的结果作为一个带名称的列存储起来，可以被后续的SELECT语句中WHERE子句调用。

## 3.11 Access工作台：在Access中使用视图

**第II部分数据库设计**

# 第四章 数据建模与实体-关系模型

•数据库开发过程包括3个主要步骤：需求分析（和数据建模），组件设计（数据库设计）和实现。这是系统开发生命周期（systems development life cycle, SDLC）模型的一个子集。

## 4.1 需求分析

## 4.2 实体-关系数据模型

•用于将需求转换成数据模型，才能进一步用来设计数据库。

•实体关系数据模型，又称E-R模型，现在的E-R模型都是扩展的，即extended entity-relationship model。E-R模型中最重要的元素是实体、属性、标识符和关系。

4.2.1 实体

•实体是用户希望跟踪的对象，给定类型的实体分组到实体类（entity class）中。用大写字母表示实体类。实体类的实体实例是特定的实体。这里的概念类似编程中的类与对象。实体类对应表名。

•用数据模型的术语和数据库术语比较只是便于理解。实际上数据模型和数据库设计分属不同的阶段（无法比较相同不同），在数据模型阶段无需考虑很多具体的约束问题，实体类是否会在后面被创建为表也是不一定的问题。

4.2.2 属性

•属性即对应表的列，用首字母大写的词组命名。

4.2.3 标识符

•标识符顾名思义，为实体起到标识作用的属性，一般为实体的名称或标号。两个以上的属性组成的标识符称为复合标识符（composite identifier）。对应表的键，即也可唯一或不唯一。

**数据模型-实体类**

①第一种表示法

**实体类**

|  |
| --- |
| 标识符 |
| 其他属性 |

②第二种

**实体类**

|  |
| --- |
| 标识符 |

③第三种

|  |
| --- |
| **实体类** |

4.2.4 关系（relationship）

•定义：E-R模型包括关系类和关系实例。关系类反应实体类之间的关联，关系实例则反应实体实例之间的关联。关系不可以有属性。

•度数（degree）：关系中实体类的数目，度数为2的关系较常见，也称为二元关系（binary relationship）。度数为3的称为三元关系（ternary relationship）

•创建数据模型时，首先标识实体，然后考虑其中关系，最后确定其属性以及标识符。另外，业务规则不是物理或逻辑规则，只是必须考虑进入的参考规则，思考的出发点是数据库不是业务规则。换句话说，数据库不能违反业务规则，但是要包含业务规则无法顾及的情况。

(1)二元关系的三种类型

①1:1关系，一对一

②1:N关系，一对多

•一般会将1对应的实体称为父实体（parent），N对应的称为子实体（child）

③N:M关系，多对多

**数据模型-关系**

①二元关系与三元关系：p195/212

②二元关系的三种类型表示：p195/212

③具有最小基数的关系：p196/213

(2)最大基数（maximum cardinality）

•基数（cardinality）：即一个关系连接的实体中出现的实体实例数，最大基数即能允许的最大实例数。比如1:N关系中，N一侧的实体最大实体实例数为N，但是1<x<N的任何一个数也可以作为实体实例数。

(3)最小基数（minimum cardinality）

•同理，表示必须参数关系实例的最小实体实例数。用0表示关系中可以不存在该实体，用1表示关系中至少包含一个该实体。

•如果最小基数是零，则关系中的实体参与是可选的（optional），如果为1，则表示该实体参与是强制性的（mandatory）。

•注意：写法说法上，某实体的最小基数表示的是和它产生关系的实体的最小数量，而不是它自己的最小数量。

## 4.3 实体-关系图

4.3.1 E-R模型的版本

•版本很多，符号多种，因软件不同而产生的区别也有。

①Information Engineering(IE), James Martin, 1990

也称为鸟足模型（crow’s feet）

②IDEF1X(Integrated Definition 1, Extended), NIST, 1993, 美国国家标准

③Unified Modeling Language(UML), OOP

4.3.2 数据建模产品中的E-R变体

**数据模型-鸟足版本**

p198/215

4.3.3 弱实体（weak entity）

•除非另一种实体也存在，否则弱实体就不存在。非弱实体称为强实体（strong entity）

4.3.4 ID依赖实体（ID-dependent entity）

•ID依赖实体是弱实体的一个特殊类型。该实体的标识符总是包含另一个实体的标识符的复合标识符。被包含的实体显然就是被依赖的实体。

•因为有ID依赖关系限制，因此ID依赖实体往往存在父实体（多对一的情况）且往往父实体最小基数为1。而没有父实体便没有ID依赖实体，有后者则必须先有前者。

**数据模型-ID依赖实体**

ID依赖实体使用圆角方框表示，强实体用直角方框表示。用实线表示ID依赖关系，也称作标识关系（identifying relationship）。以虚线表示强实体之间关系，也做非标识关系（nonidentifying relationship）。

4.3.5 非ID依赖的弱实体

•弱实体：逻辑上依赖另一个实体的实体（即物理上依赖另一个实体存在而存在，不考虑因为业务规则而产生的依赖关系，比如学生必须有老师，但逻辑上不依赖老师存在而存在，因此不是弱实体）。并非所有最小基数为1的实体都是弱实体，而弱实体的最小基数都为1。

**数据模型-非ID依赖弱实体**

弱实体仍采用圆角方框表示，但是关系用虚线表示，和强实体一样。

4.3.6 子型实体

•定义：子型实体（subtype）是另一种名为父型（supertype）实体的特例，由扩展的E-R模型引入。比如学生可以分为本科生与研究生。子型实体与父型实体使用同一个标识符，因为二者的实体实例是同一物体。

•子型实体与父型实体有相同的标识符，但是其属性不同。而父型实体也不必须包含子型实体的属性，即二者属性可以完全不同。子型实体的作用是找到父型实体内部的精细划分，并区别对待不同的父型实体，以避免诸如NULL值（如本科生和研究生有不同的考核方式，全部作为一个实体的属性来处理便会引入NULL值）等问题。

•辨别器（discriminator）：实质上是父型实体的属性，可以根据该属性的取值来决定父型实体实例可以划分进入哪个子型实体。不是每个父型实体都有辨别器，一般只有排他子型实体的情况才可能有。

•IS-A关系：连接父型实体和子型实体的关系

•HAS-A关系：表示其他的一般关联关系，关系所连接的实体不是同一实体

**数据模型-子型实体**

p203/220

子型实体固然是ID依赖的，使用实线连接关系

4.3.7 递归关系（recursive relationship）

•定义：某个实体与自身同类实体可能存在某种关系。比如客户之间可能存在关系，也称为一元关系，也可能是1:1,1:N,N:M。

**数据模型-递归关系**

图5-21，p244/261

## 4.4. 开发E-R图示例

4.4.1 Heather Sweeney Designs公司的数据库

4.4.2 培训课的客户列表

•一般常见的实体包括“客户customer”，“交易deal”，“活动activity或各种形式的会议”等等，也包括一些虚拟的事物用来管理，比如“调度分配assignment”，“联络contact”等

•可以使用代理键来代替一些必须使用较长复合键的情况。

4.4.3 给客户的信函模板

•实体的属性并非一次性添加完成，因为获取某个实体类的信息的渠道不同，可能在分析时先后加入不同的属性，而有可能之前加入的属性因为渠道不同而无法再提供。比如通过电子邮件和电话记录来添加客户实体时，先添加的客户实体有电子邮件但电话为NULL。

•对于所有最小基数为1的情况都有必要检查一下是否为弱实体。

4.4.4 销售发货单

•【经验】交易或者货单实体代表一次交易记录，而每次交易中涉及多个商品。一般这些商品的信息不会存储在交易实体中（数目多变而且复杂），而是单独作为一个实体。但是交易中的商品（历史记录）和当下实际商品可能存在区别，比如涨价等。所以将产品PRODUCT作为一个强实体，而每种交易商品（包含每次交易的数量）作为一个弱实体分开创建。

•【经验】有时候用户无法或没有经验提供一些信息，虽然不影响项目的工作，但是这些信息是非常重要的，并且也许以后用户会发觉。可以根据我们自己的经验在不大量增加工作成本的前提下，优先为其考量。比如为一些实体增添一些重要属性来存储未来可能会提供的信息。

4.4.5 属性说明

•进一步细化的数据模型中，实体属性应包括：列名，数据类型（长度），是否为键，是否可NULL，默认值，注释等。

4.4.6 业务规则

•业务规则的实现有时候靠数据库或DBMS实现（依靠SQL），但是更多复杂的规则仍需应用程序来实现。

4.4.7 验证数据模型

•验证数据模型的最常见做法是建立数据模型的原型呈现用户获取反馈。如果数据模型较复杂，可以分部分来验证。

•原型即简化版的数据库，只存储基本的数据（多数是样本填充数据）。然后创建原型表单与报表给用户验证。

•验证需要使用全部用例来验证。

## 4.5 Access工作台：第四部分——使用Access开发原型

# 第五章 数据库设计

## 5.1 把数据模型转换为数据库的设计方案

•数据库设计方案的纸面表示和E-R模型图类似（有方块表示的内容和连线表示的关系），但是添加了更多数据库内部结构的细节。

•转换流程：

(1)为每个实体创建一个表

①指定主键（考虑代理键是否合适）

理想的主键是不会变化的简短数字；但代理键无法保证多个数据库之间的相同类型实体的键值不重复（因为没有实际意义）

②为每个列指定属性

数据类型，空值状态（一般只有创建行时注定已知的信息才会设为NOT NULL），默认值，数据约束

③验证规范化

反规范化

有些函数依赖并不是常识性的，而是带有业务特性甚至个人习惯的。所以需要向客户调查清楚是否有一些这样的“约束”

(2)通过设置外键或表来创建关系

•可以看出，有的实体不是在数据模型建立时创建的，而是规范化表之后创建的。当然有经验以及完美的约束下，数据模型设计时也应该做到这一点。

(3)实现参照完整性约束

## 5.2 使用关系模型表示实体

•三个示例

5.2.1 ITEM实体的表示

**数据库设计-实体和表**

图5-2，p228/245

完整的数据库设计表：图5-3，p230/247

5.2.2 CUSTOMER实体的表示

5.2.3 反规范化（denormalization）

•反规范化：即有时根据实际情况，不需要将表的某些列单独提取作为新表来维护的情况。新表的维护有性能代价，但换取的是数据库信息的完备，精确以及简易。但有一些信息由于现实中的特殊特性，虽然不满足规范化但是不会造成后果。

•举例：客户信息中地址和邮编（地址和邮编有依赖关系且不为候选键，不满足规范化）

•反规范化的一般条件：验证一个规范化问题或表的结构问题，常见的方法就是分别验证增删改基本操作的处理情况。以某一表内反规范化依赖关系A→B为例，增：不需要存储过多的A到B实例，只保留原表所需的最后一次更新的即可；删：任何一个A到B的实例从数据库中删除并不重要，不需要存储这些信息；改：A到B的映射几乎不会更改（因为若改一对映射，在原表内则可能需多处更改）

5.2.4 SALES\_COMMISSION实体的关系设计

5.2.5 弱实体的表示

•弱实体的大部分关系设计需要通过应用程序代码来实现，因为它必须依赖父实体的创建而创建。

## 5.3 关系的表示

5.3.1 强实体中的关系

(1) 1:1强实体关系的表示

•只需将两个强实体创建为表，然后将一个表的主键放入另一个表中作为外键。而作为外键的列需要被约束为UNIQUE以反映1:1的关系，而不是1:N

•最小基数反应在数据库设计中的NULL值上（即数据是否必须），是否允许NULL值取决于关系另一端实体的最小基数。

•两个主键到底哪个作为外键？概念上等价但要考虑性能

①实体的关系对应表的连接JOIN，而显然表的连接是需要消耗性能的。实际应用时，哪一种查询更加常用，则应该优先使该查询更加便利

②同理，如果两个表的尺寸差异较大，那么不同方向的连接性能也不同

③底层结构中，要考虑哪个表有更多更全的索引

(2) 1:N强实体关系的表示

•将两个强实体创建为表，然后将父表（1端）的主键放入子表（N端）中作为外键，外键列不约束UNIQUE。

(3) N:M强实体关系的表示

•对于N:M（不知道是不是多值依赖）关系，将两个强实体创建为表，然后创建一个新表，名为交集表（intersection table）来表达这种关系。一般这个新表注定主键为复合键，且是拥有N:M关系的两个强实体表的主键的组合。交集表的主键同时也是两个父表的外键。

•更多的，交集表实际上是一个ID依赖弱实体（严格的说它不存在实体，只是一个表达关系时创建的表），它同时依赖两个强实体。而两个强实体各自和它的关系均为1:N，等同于将N:M拆分成两个1:N。

5.3.2 使用弱实体的关系

•关联实体（associative entity）和关联关系（association relationship）：类似N:M关系的交集表（注意交集表不是实体）的弱实体，连接两个被它ID依赖的实体。和交集表不同，这样的弱实体往往也有自己的属性，而不是除了主键没有其他属性。

•混合实体模式：类似关联关系，但是只有一个ID依赖关系，另一个关系是非标识关系。

**数据库设计-关系**

图5-12，p239/256

图5-18，p243/260

5.3.3 子型实体关系的表示

**数据库设计-子型实体**

图5-20，p244/261

5.3.4 递归关系的表示

•递归关系的处理和一般关系类似，只是将同一个表看做两个表来处理。仍然使用外键应对1:1与1:N的情况，使用交集表处理N:M。

•使用SQL连接来处理递归关系时，可以为同一个表赋予不同的别名来进行操作。可以看成是使用了两个相同结构的表

举例：N:M递归关系，医生医治医生

SELECT \*

FROM DOCTOR A, TREATMENT-INTERSECTION, DOCTOR B

WHERE A.Name = TREATMENT-INTERSECTION.Physician

AND TREATMENT-INTERSECTION.Patient = B.Name;

## 5.4 Heather Sweeney Designs公司的数据库设计

5.4.1 弱实体

5.4.2 关系

5.4.3 实现参照完整性约束

**数据库设计-参照完整性约束**

图5-27，p249/266

•级联行为CASCADE的讨论：

级联行为往往针对的是主键外键之间的操作，确切的说是主键发生变化（更新或删除）时，外键完全配合的响应行为。反过来NO ACTION代表不会对被外键“占用”的主键进行操作（即不级联），没有被外键“占用”的主键可照常修改（也不会有级联的可能）。

①级联更新：

a.代理键不会改变，不需要级联更新，甚至不会遇到更新（×）

b.根据业务规则不会改变的主键，也不会遇到更新（×）

c.一般主键改变时，外键也需要配合改变（√）

②级联删除：

a.有外键对应值的主键，表示某种意义的被“占用”，主键不能删除，也不会级联（×）

b.因为业务规则原因，尽管没有外键对应值，也不会删除主键（×）

b1.如果是b情况，则a一定满足，但没有外键对应值时，对主键的保护需要其他手段实现

c.因为业务规则，主键所在的实体对于外键所在实体有管理其“命运”的机能，删除主键则所有外键需配合改变（√）

## 5.5 Access工作台：第五部分——Access中的关系

# 第六章 数据库管理

## 6.1 Heather Sweeney Designs数据库

•数据库管理（database administration，DBA）：DBA也可以指database administrator。负责数据库及其应用程序的开发，运行与维护。

## 6.2 控制，安全和可靠的必要性

•三个数据库管理功能：控制并发，安全，备份恢复

## 6.3 并发控制

•目的：保证一个用户的工作不对另一个用户造成不合理的影响。另一种情况下，用户的工作按照预定的方式被其他用户影响。

6.3.1 使用原子事务的必要性

•事务（transaction）：也称逻辑工作单元（logical units of work，LUW）或称为原子（atomic），实质上是一系列数据库基本操作的集合，由应用程序规划实现。这些操作集合实际上是业务工作上的基本单位（因为很少有用户仅仅需要单独的基本操作，往往都是一些常见事务，由一组基本操作组成）。从基本操作层面上看，这一组操作要么都成功，要么全部不执行（即出现任何一处错误时，回滚至事务开始之前状态）。

6.3.2 并发事务处理

•虽然处理数据库的中央处理器一次只能执行一个命令，但是它是交叉执行并发事务的，也就是在两个同时事务间来回切换执行每个部分。切换的时间点一般选取在事务处理中断处，比如说等待A用户的操作或其他和A有关的非CPU操作时，CPU会去处理B的事务以顺便等待A端进展。

6.3.3 丢失更新问题

•丢失更新问题（lost update problem）：也称并发更新问题（concurrent update problem），在两个用户同时针对同一个数据库对象进行事务操作时，其中A用户对该对象的修改没有反馈到B用户的工作区造成错误（一般CPU为A读取数据后会等待A响应并为B读取数据，二者的初始数据相同，但之后的不同操作需要兼顾处理）

•非一致读取问题（inconsistent read problem）：A读取的数据已经被B的部分事务处理过，A读取了错误数据。

6.3.4 并发问题：脏读取，不可重复读取和幻象读取

•这里总结并发处理问题的标准化名称：

①脏读取（dirty read）：当一个事务读取了一个未提交到数据库但已修改的记录时，就会发生脏读取，亦即丢失更新问题。

②不可重复读取（nonrepeatable read）：如果事务重新读取以前读取过的数据，并发现另一个事务对其进行了修改和删除，就会发生不可重复读取。

③幻象读取（phantom read）：当事务重新读取数据，但发现上次读取该数据后，已有另一事务插入了新的数据行，就会发生幻象读取。

6.3.5 资源锁定

•定义：修改某些数据行或表时，禁止多个应用程序同时获取它们的副本。

•DBMS执行的锁定称为隐式锁定（implicit lock），而命令执行的锁定称为显式锁定（explicit lock）

•锁定的范围有级别划分，页面或表或数据库级别。锁定的大小称为锁定粒度（lock granularity）。对于粒度较大的锁定，DBMS容易管理但常引发冲突。反之不易管理，增加了大量的跟踪与细节，但是冲突较少。

•锁定的类型也有区别。排他锁定（exclusive lock）禁止任何形式的对数据项的访问。共享锁定（shared lock）禁止修改数据项，但是不拒绝对数据的读取。

6.3.6 可串行化的事务

•串行模式：即使用串行的方式来处理并发事务。

•方法：使用二段锁定（two-phased locking），一旦使用一个锁定，就不能使用其他任何锁定。事务有获取锁定的增生阶段（growing phase）和释放锁定的收缩阶段（shrinking phase）。

6.3.7 死锁（deadlock）

•定义：也叫致命包含（deadly embrace），因为并发事务多个锁定同时发生导致两个事务均无法完成的情况。

•解决办法：1. 允许死锁发生，然后打开，2. 阻止其发生

(1) 允许发生：当发生时，回滚其中一个事务，取消它对数据库的改动。

(2) 阻止发生：a.允许事务一次只发出一个锁定请求，一次锁定所有需要的资源。b. 规定所有应用程序以相同的顺序锁定资源。

6.3.8 乐观锁定和悲观锁定

•乐观锁定（optimistic locking）：先处理事务前期操作（比如读取数据和计算等，不包括写入及更新数据），然后锁定资源准备更新，此时检查是否发生冲突（读取数据后锁定前数据库中该数据被其他事务改变）。如果有，则需要重复事务，直到没有冲突。

•悲观锁定（pessimistic locking）：开始运行前，先锁定资源，处理事务全过程，最后释放锁定。

•两种方式区别在于乐观锁定假定没有冲突，而悲观锁定假设有冲突（更保险的做法）。

•乐观锁定的锁定时间比悲观短，如果锁定粒度大，事务很复杂（不是基本操作多而是需要大量计算和程序逻辑），那么乐观锁定的时间优势更加明显。

•事务如果需要很多数据库基本操作，那么冲突的可能性大大增加，乐观锁定可能一直无法完成事务。

6.3.9 声明锁定特征

•SQL事务控制语句标记事务边界：p277/294

BEGIN TRANSACTION 标记事务开始点

COMMIT TRANSACTION 完结事务点

ROLLBACK TRANSACTION 回滚事务的命令

•数据库应用程序一般不显式使用锁定，而是标记事务边界后，声明DBMS使用乐观或悲观锁定。之后如果需要改变锁定行为，无需重写应用程序。

6.3.10 一致事务

•事务的有些特点可概括为ACID，不是所有的事务都要满足，ACID事务满足如下

①atomic原子性：原子事务顾名思义业务范围内不能继续划分，要么执行全部事务操作，要么完全不执行。

②durable持久性：事务中所有提交的修改都是永久的，即便发生错误时也是如此。当然DBMS还是在需要时提供辅助措施来恢复修改。

③consistent一致性：一致性即另一种角度定义的锁定行为，包括两种层面的一致性，语句级一致性（statement-level consistency）和事务级一致性（transaction-level consistency）。语句级表示一个SQL语句执行期间，其他的事务不许修改该SQL语句涉及的行；事务级则表示事务中所有SQL语句所涉及的所有行在事务运行期间不许被其他事务修改。

•但是要注意，有的事务在运行期间前半部分改变了一些数据库内容，而后半部分受到前面的操作结果的影响，后面的操作不知道前面的结果如何，可能会造成一些问题需要查正。

6.3.11 事务隔离级别

•隔离级别（isolation level）分为4级，指定允许产生哪种并发控制问题。隔离级别的作用是让应用程序编程人员声明需要的隔离级别，然后由DBMS管理锁定来实现。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 隔离级别  问题类型 | 隔离级别 | | | |
| 读取未提交 | 读取已提交 | 可重复读取 | 可串行化 |
| 脏读取 | 可能 | 不可能 | 不可能 | 不可能 |
| 不可重复读取 | 可能 | 可能 | 不可能 | 不可能 |
| 幻象读取 | 可能 | 可能 | 可能 | 不可能 |

• “读取未提交”的含义是，某个事务读取完数据，在提交修改之前，就允许其他事务读取该事务要处理的数据，自然也就是脏读取的定义。“读取已提交”表示，某个事务读取完，提交修改之后才允许其他事务读取该数据，虽然避免了脏读取，但是还是允许其他事务“干扰”了该事务，会发生不可重复读取的情况。

•显然，事务一致性与隔离级别有重复性联系，一致性的事务注定在“可重复读取”级别以上。

## 6.4 游标类型

•游标（cursor）是一种指针，指向SQL SELECT语句的结果集，定义方法如下

DECLARE CURSOR CursorName AS

SELECT…

应用程序打开某个游标后，就可以把游标放到结果集中的某个地方。

•游标需要相当大的内存消耗，因此同时打开很多游标时，不放考虑使用缩容（reduced-capability）游标

•游标按照移动方式可分为：“只向前游标（forward only cursor）”，应用程序只能在记录中向前移动，即后面的修改和操作不可见；“可滚动游标（scrollable cursor）”，可在记录中前后移动。游标的类型概念对于不同的DBMS几乎一致，只是除了SQL Server的DBMS将只向前游标设为第四类，前三类都是严格可滚动游标。

•游标的功能分类有三种：

|  |  |  |
| --- | --- | --- |
| 游标类型 | 说明 | 注释 |
| 静态游标  static cursor | 应用程序在打开游标时才看到数据 | 由静态游标所做的修改是可见的，来自其他源的修改是不可见的。就好比获取了结果集的一个快照。 |
| 键集游标  keyset cursor | 打开键集游标时，每行的主键值都保存到记录集中。当应用程序访问行时，该键用来取回该行的当前值 | 来自任何源的更新都是可见的。来自键集游标外部的源的插入是不可见的（因为键集中没有这些源的键）。来自键集游标的插入显示在该记录集的底部。来自任何源的删除都是可见的。以行的顺序进行的修改都是不可见的。如果隔离级别是读取未提交，那么未提交的修改是可见的。否则只有提交的修改才是可见的。 |
| 动态游标  dynamic cursor | 任何类型的修改以及来自任何源的修改都是可见的 | 所有按记录集顺序进行的增删改操作都是可见的。如果隔离级别是读取未提交，那么未提交的修改是可见的（因为未提交修改之前即可被其他源读取，但提交后仍会动态更新）。否则只有提交的修改才是可见的。 |

•静态游标性能开销最小，动态游标开销最大。游标位于客户机上时开销小，在服务器端时开销大。后者比前者决定性更大。

## 6.5 数据库安全

•数据库安全的目的：确保只有授权用户才能在授权时间内进行授权的操作。包含两个部分，第一是身份验证，一般由用户提供用户名和密码；第二是授权，授予用户DBMS特定的许可。即只有身份验证通过，还没有得到DBMS授权的用户仍然无法访问数据库。

•数据库安全的目标实现之前，必须确定的内容：

①哪些用户可以使用数据库

②每个用户的处理权限和责任

6.5.1 用户账户

•用户账户是创建在DBMS中的，而不是特定的数据库中，是DBMS内部的用户账户。DBMS账户和密码安全的术语，特性和功能取决于DBMS产品。比如，SQL Server除了可以使用内部用户名和密码，也可以使用Windows系统进行登录验证；而SQL Server允许在数据库级别创建用户（来管理该DBMS用户对该数据库的访问权限），显然这个用户是基于DBMS级别用户的，而且密码共用。

6.5.2 用户处理权限和责任

•DBMS安全模型，基于E-R关系模型：图6-17，p282/299

一个用户可以是一个或多个角色，一个角色可以有一个或多个用户，角色（或用户）和对象之间的关系由权限或许可来描述，每个许可赋予一对角色（用户）和对象。

**数据库管理-用户组处理权限**

用户组处理权限划分范例，图6-18，p282/299

SQL Server中固定的数据库角色，图6-20，p283/300

注意：角色的名称显然是计划时任意取定的，并不具有数据库操作级含义。在SQL Server中，DBMS自带一些固定的基本数据库角色，并赋予了他们基本的相关权限。因此无需再定义角色，只需按照计划所需要的将特定的用户赋予一个或多个具有所需权限的固定角色即可。

• “系统管理员”之类的角色可以授予权限，但不能直接访问数据，虽然看似限制较弱，但所有改变安全系统的操作会在DBMS日志中留下踪迹。

•数据库安全管理（以及网络管理）的一个非常重要的规则：除非绝对需要，否则各种权限类型是授予用户组或用户角色的，而不是用户个人的。

•用户的权限包含着责任，即创建数据库和应用程序后，赋予一批用户不同的权限进行使用。也需要赋予对应的用户对应的责任令其遵守和参照。责任是系统培训和用户手册的内容，不是DBMS或应用程序的逻辑。而长期跟踪管理权利和责任的任务，是DBA的。

•权限的具体实现有很多途径，分属于不同的级别：操作系统，网络目录服务，Web服务器，DBMS和应用程序。实际上就是自然人与数据库应用程序之间所有涉及的纽带都可以实现权限管制。

6.5.3 DBMS级别的安全

•DBMS安全指南：

①在防火墙后运行DBMS：不允许从企业网络外部进行任何与DBMS或数据库应用的通信

②应用最新的操作系统，DBMS服务包与补丁

③将DBMS的功能限制为所需的功能：所有未使用的功能与协议等都必须禁止和删除

④保护运行DBMS的计算机（服务器）：禁止任何用户在DBMS服务器上工作，控制访问服务器（物理上或网络上）的人员与权限

⑤管理账户和密码

⑥加密在网络上传输的敏感数据

⑦加密在数据库上存储的敏感数据

6.5.4 应用程序级别的安全

•一般而言，应先利用DBMS的安全特性。只有不满足要求时，才需要添加应用程序代码。当然DBMS的安全特性仅限于常规的安全保护。和业务逻辑密切相关的一些安全措施，DBMS无法实现。

## 6.6 数据库备份与恢复

•发生系统崩溃，硬件故障等重大错误时，需要尽快恢复数据库。然而要想完全恢复到中断处，实际上并不现实。两种可行的方法如下：

6.6.1 通过重新处理来恢复

•原理：通过定期备份并记录自该备份后处理的所有事务，故障后恢复备份并重新处理后续事务。缺点是，如果任务过多（不一定是备份间隔过长导致的），系统可能无法在短时间内恢复到故障点。如果并发处理事务，但是事件本身是异步的，也会造成不一样的结果。因此重新处理通常不是多用户系统故障恢复的方法。

6.6.2 通过回滚和前滚来恢复

•原理：同样需要定期备份数据库，但是记录的是备份后事务对数据库做的改动。实际上，只有对数据库做的改动才会改变数据库本身，事务本身可能含有大量不会更改数据库的内容。遇到故障时，第一种方法前滚（rollforward），顾名思义即恢复备份后，重新应用记录中的改动（显然比重新处理事务更快）；第二种方法回滚（rollback），即根据记录取消错误地执行或者部分完成的事务的修改，然后重新启动所需事务即可。

•在把事务应用到数据库之前，必须将它们写入日志。具体来说，是某事务的每次改动数据库的操作之前都需要先记录下来。这样一旦发生崩溃，最多只有一个未应用事务的记录。

•数据库变化前所有的记录称为前像（before-images），用来撤销事务；数据库改变后的所有记录称为后像（after-images），用来恢复事务。

**数据库管理-事务日志**

事务日志示例，图6-24，p288/305

示例日志是一个表格，表格中每件事务都有一个独特的ID，每件事务都包含若干个数据库操作。由于是并发处理的，每件事务的操作不一定前后紧邻，而是通过类似链表的前后指针指示。

•检查点（checkpoint），顾名思义是DBMS产品自带的自动备份和事务与日志同步机制。1小时内记录3-4个检查点并不费事，最多只需15-20分钟处理即可恢复故障。

## 6.7 DBA的其他职责

(1)DBA需要确保采用一个系统来收集和记录用户报告的错误及其他问题，对这些问题进行优先级排序，确保它们能得到解决。DBA需要创建并管理一个控制数据库配置的过程。这个过程包括：记录修改请求、让用户和开发人员评估该请求，创建用来实现获批准的改动的项目和任务。

(2)DBA还需要负责维护文档，记录数据库结构、并发控制、安全、备份和恢复、应用程序的使用，以及大量与数据库的管理和使用相关的细节。

## 6.8 分布式数据库的处理

•分布式数据库是在多台计算机上存储和处理的数据库。

6.8.1 分布式数据库的类型

•可以通过分区（partitioning）和复制（replication）来分布数据库，两种方式也可以并用，将数据库分割/复制到多台计算机中。具体的，每个表的不同行或不同列都可以赋给不同的计算机。

•优点：在多台计算机上放置数据库可以提高吞吐量，或者缩短用户和计算机的距离以减少通信延迟。分布也可以改进控制能力。

6.8.2 分布式数据库面临的挑战

•分布式数据库面临的问题基本上是多个关系型数据库面临问题的复杂版。使用这种数据库面临问题很多，而且需要考虑数据通信方面的因素与知识。

## 6.9 对象-关系数据库

•在OOP领域里，有一种需求就是将对象的属性（数据成员）存储起来，这个操作称为对象持久化（object persistence）。使用一般的关系型数据的表结构无法很有效的完成这一工作，因为对象的数据结构往往比表的行更加复杂。因此开发了针对这一需求的数据库类型，更多知识可以搜索OODBMS和ODBMS。

## 6.10 Access工作台：第六部分——Access中的数据库管理

# 第七章 数据库处理应用程序

## 7.1 数据库处理的环境

•数据库根据规模不同需求不同，有不同的处理方式和处理环境。比如：

①由采用Internet技术（如Active Server Pages.NET，ASP.NET和Java Server Pages，JSP）的应用程序来处理

②由VB，Java，C#等语言开发的应用程序来处理

③利用存储过程和触发器来处理

7.1.1 查询、表单和报表

7.1.2 客户机/服务器和传统的应用程序处理

7.1.3 存储过程和触发器

•企业级DBMS产品允许开发人员创建逻辑模块（触发器）和数据库操作（存储过程）。前者即类似拦截器的程序代码，在指定的数据库操作的前或后或其他时机调用。后者即类似DBMS中的一个函数。

## 7.2 Web数据库处理

•应用程序访问数据库服务器需要使用DBMS提供的标准接口（即应用程序接口API），API是应用程序通过代码执行DBMS功能的对象、方法和属性的集合。

•计算机行业开发了数据库访问的标准，用以规范不同DBMS的API。如：ODBC（Open Database Connectivity）标准，OLE DB标准和ADO（Active Data Objects）活动数据对象。

•ASP.NET（HTML和.NET组合）用以创建Web页面或Web应用程序和IIS（Microsoft的Web服务器产品）组合；JSP（HTML与Java的组合）和Apache Web Server；AMP（LAMP或WAMP）组合，即Apache-MySQL-PHP/Pearl组合。这些都是常见的基于Web的数据库应用程序开发工具组合。

•如果Web服务器和DBMS可以在同一台计算机上运行，该系统就有双层结构（一层浏览器，一层Web服务器/DBMS计算机）。反之，就有三层结构。

7.2.1 ODBC

•原理：开放数据库互联（ODBC）标准允许编程人员使用该标准的语句编写各种ODBC产品的代码指令。这些指令传递到ODBC驱动程序，被翻译成所用DBMS的API。驱动程序接收DBMS的结果并翻译，作为ODBC标准的一部分。

(1)ODBC体系结构

•数据源：根据ODBC标准，数据源是数据库与其相关的DBMS，操作系统以及网络平台。

•应用程序，ODBC驱动程序管理器以及ODBC DBMS驱动程序都位于Web服务器上。

•ODBC指定了两种驱动程序：单层驱动和多层驱动；前者处理ODBC调用以及SQL语句，后者只处理ODBC调用，把SQL语句直接传递给数据库服务器。

(2)创建ODBC数据源名称

•数据源（data source）是一个ODBC数据结构，换句话说是ODBC标准定义的一个包含了数据库本身和其他相关内容（比如数据库使用的DBMS链接）的对象。每个数据源有一个名称（DSN）用来引用该数据源，有三种数据源：

①文件数据源可以在数据库用户之间共享的文件，唯一的要求时用户必须拥有相同的DBMS驱动和访问数据库的特权。

②系统数据源对应一台计算机，操作系统和该系统上的所有用户都可以使用系统数据源。这是一般的企业级应用程序常用的数据源类型。

③用户数据源只提供给创建它的用户。

7.2.2 使用IIS进行Web处理

•也可以使用Apache HTTP服务器，Web服务器就是用来存储要建立和使用的Web页面，也可以理解为Web应用程序。

7.2.3 HTML Web页面入门

•HTML（Hypertext Markup Language）是最基本的Web页面创建语言。超文本标记语言中的超文本表示可以包含对其他对象的链接，比如Web页面，图片或音频视频文件等。HTML是一组标准的HTML语法规则和文档标记，Web浏览器可以解释并根据他们创建特定的显示内容。

•W3C（World Wide Web协会）是指定HTML标准的协会。

7.2.4 Web页面index.html

•index.html是一个Web服务器专门的默认名称。在没有指明特定文件的情况下发出URL请求时，大多数Web服务器都会自动显示该文件。

7.2.5 创建Web页面index.html

7.2.6 使用PHP对Web数据库进行处理

•使用Web开发环境扩展网站功能，以便将页面连接到数据库上，有几种技术都可以实现：

①Microsoft：.NET Framework和ASP.NET技术

②Apache Web：JavaScript脚本语言创建JSP文件，或者使用Java EE

(1)PHP脚本语言

PHP是可以嵌入Web页面（HTML）的脚本语言（scripting language），HTML语句在用户的工作站上执行（一般就是浏览器，指明浏览器所需的工作，因此也被称为静态页面），PHP在Web服务器上执行，也就是指挥Web服务器的工作（比如访问数据库等）。

(2) Eclipse集成开发环境IDE

使用Notepad这样的编辑器就可以创建简单的Web页面，但是复杂的页面最好使用专用的IDE工具。比如Visual Studio， NetBeans和Eclipse等。Eclipse PDT（PHP Development Tools）专门用于在Eclipse中提供PHP开发环境。

(3) 创建与数据库的连接

(4) 创建RecordSet对象

(5) 显示结果

PHP中使用echo指令来使用HTML语法，常用作在PHP代码块中使用HTML页面显示所需结果。

(6) 与数据库断开连接

7.2.7 Web数据库处理面临的挑战

•HTTP（超文本传输协议）的特征是无状态的，无法维护请求之间的会话。浏览器中的客户端使用HTTP向Web服务器发出请求，服务器处理该客户端请求，将结果返回浏览器，然后就会“忘记”和客户端的交互，不保存任何数据以维护会话或与客户端的连接。因此，无论何种Web服务器，都必须向数据库应用程序添加代码来启动事务处理或者称为会话管理，因为事务往往包含一组数据库操作。

7.2.8 SQL注入攻击

•概念：实际上是一种代码欺骗，将本应传递某种类型数据的变量（如字符串变量）赋予了含有SQL代码的字符串值（实际上原本的SQL语句也是字符串）导致改变了原本的SQL语句含义。

•应对措施：执行输入数据检查，执行日志检测，确保只授予必须的数据库权限。

## 7.3 数据库处理和XML

7.3.1 XML的重要性

7.3.2 作为一种标记语言的XML

7.3.3 XML文档类型声明

7.3.4 用XSLT实体化XML文档

实体化即将XML文档实体化为其他有特定用途的文档，比如HTML等。

7.3.5 XML模式

7.3.6 XML和数据库处理

•简单而言就是XML格式文档和数据库表数据之间的转换与传递。

7.3.7 XML Web Services

•是一种标准，作用是共享数据库应用程序，即其他计算机可以通过标准的接口访问并使用某个计算机上的数据库应用程序。

7.3.8 NoSQL Movement

•相对于关系数据库模型和SQL，对应的即是NoSQL和非关系DBMS，也称为结构化存储（structured storage）。NoSQL DBMS一般是一个分布式的复制数据库。NoSQL数据库的一种实现方式可以使用XML文档结构来存储数据。

•非关系数据库是一种不同的方法来存储数据和处理信息的。但并不是意味着它没有关系的概念，是一团杂糅在一起的信息。

## 7.4 Access工作台：第七部分——使用Microsoft Access进行Web数据库处理

# 第八章 商业智能系统的数据库处理

## 8.1 商业智能系统

•商业智能系统（BI）也称为决策支持系统（DSS），实际上就是协助管理员和上层用户分析数据，预测事件，并支持决策的系统。显然，它可以是数据库应用程序的一部分，但是并不是对数据库操作的一部分。

## 8.2 可操作系统与BI系统的关系

•可操作系统支持主要的业务操作，实际上就是数据库应用程序操作数据的本体。也称为事务处理系统或在线事务处理系统（OLAP）。

•BI获取用于分析的数据来源有三：1.使用可操作系统的DBMS获取数据，但是不能使用该DBMS对数据修改；2.使用自己的BI DBMS来处理可操作数据库中的数据；3.直接读取购买来的数据。

## 8.3 报表系统和数据挖掘应用程序

•BI系统分为两大类：报表系统和数据挖掘应用程序。二者的功能与特性为：

①报表系统

分类，挑选，分组和进行基本计算（要比数据挖掘的数据处理简单）

汇总当前状态

比较当前状态与过去或预测状态

给实体（客户，产品，雇员等）分类

报告重要的交付活动（重在将结果汇报给用户）

②数据挖掘

常常使用复杂的统计和数学技术

用于假设分析，预测和决策

结果常常合并到其他报表或系统中（直接使用数据分析的结论而不是重在汇报）

8.3.1 报表系统

8.3.2 数据挖掘应用程序

## 8.4 数据仓库和数据集市

•BI系统从可操作的数据库中提取可操作数据，这一方式很难应用在大型数据库和复杂的数据库应用程序上：

①为BI系统服务会给可操作DBMS带来额外负担

②BI系统的创建和维护需要一般不能从操作中获得应用程序，设备和技术

③可操作数据本身可能并不适合提供给BI分析

8.4.1 数据仓库的组成

•数据仓库（data warehouse）是一个数据库系统，包含数据、专门用于给BI处理过程准备数据的程序与人员。数据仓库使用Extract， Transform，and Load，ETL系统从可操作的数据库中读取数据。ETL系统为BI系统准备数据（可操作数据库中的数据格式与表达不一定直接适合BI系统）

•数据仓库使用的数据库系统可以和可操作数据库不是一个公司产品或不是一个种类。

•元数据涉及数据的源，格式，假设和约束以及其他事实，这些元数据保存在数据仓库元数据库中。

8.4.2 数据仓库和数据集市

•数据仓库可以看成供应链上的发布者，数据集市（data mart）是一个比数据仓库小的数据集合，专注于某个特定的组件或业务的某个功能区域。后者类似于供应链上的零售商。也就是说数据集市是从数据仓库提取一部分数据用于分析特定业务数据的部分BI工具系统。这个系统称为企业数据仓库结构（enterprise data warehouse architecture， EDW）。

8.4.3 多维数据库

•数据仓库的数据库采用的设计类型称为多维数据库，不同于关系数据库。二者的区别在于：

|  |  |
| --- | --- |
| 可操作数据库 | 多维数据库 |
| 用于结构化的事务数据处理 | 用于非结构化的分析数据处理 |
| 存储当前的数据 | 存储当前和历史数据 |
| 数据由用户插入，更新和删除 | 数据由系统加载和更新 |

(1) 星型模式（star schema）

•多维数据库采用星型模式结构，星星中心是一个完全规范化的事实表，从中心向外辐射的是不一定规范化的维度表。星型模式的升级版本为雪片模式，雪片模式中维度表已经规范化，维度表的下一级分支为附加表。p375/392

•事实表用于存储业务操作的度量值，度量值是事实表所代表实体的数量或真实值，比如每天销售给每个客户的每个商品数与价格。维度表用于记录事实表每个属性的值和相关信息，比如日期信息，客户信息，商品信息等。事实表是所有维度表之间的一个交集表，其复合主键为所有维度表的外键（但是不是所有的事实表注定如此）。

(2) 演示维度模型

•一个用于演示数据的二维矩阵表或三维立方体图

(3) 多个事实表和一致维度表

•一致维度表：一个维度表连接到两个或多个事实表上。

## 8.5 报表系统

8.5.1 RFM分析

•RFM分析：根据客户的购买模式分析客户，并给他们排名。是一种简单的客户分类技术。考虑客户最近（recently）何时下了订单，客户下单的频率（frequently）以及客户给每个订单支付多少钱（money）。

•生成RFM分数需要两个数据：客户数据，每位客户的每次购买行为的销售数据（销售日期和销售总量）

•计算RFM分数和排名方法：

①R：按照最近一次购买日期为所有客户排序，订单日期最近的20%客户得分1，订单最远的20%客户得分5，其他类推。

②F：按照下单频率排序，频率最高的20%客户得分1

③M：按照订单量排序，订单最大的20%客户得分1

•RFM分数报表：图8-15，p381/398

8.5.2 报表系统的组成

•报表系统的组成：图8-16，p382/399

•报表特性：图8-17，p382/399

(1) 报表类型

静态：仅从底层数据中准备一次，且不会改变。如历年的销售报表。

动态：创建时报表系统会读取最新数据。如当日销售报表和当日股价。

查询报表（Query report）：响应用户输入的信息来提取数据。如搜索引擎。

OLAP：见后面部分

(2) 报表媒介

(3) 报表模式

推：根据预定义的时间主动发送给用户

拉：用户必须请求获取报表

8.5.3 报表系统的功能

①报表管理（report management）：

定义谁何时，以何种方式接收什么报表。允许报表系统管理员定义用户账户和用户组。

②报表生成（report authoring）：

涉及连接需要的数据源，创建报表结构和格式化报表。根据报表分配元数据（包括用户组信息，对应的格式，发布渠道，发布模式等）来执行。

③报表发布（report delivery）

8.5.4 OLAP（Online Analytical Processing）

OLAP reports也称为OLAP cube，其显著特征是动态的，即报表的格式可以由查看人员根据习惯和个人意愿进行修改切换。

## 8.6 数据挖掘

p387/404

8.6.1 无监督的数据挖掘（unsupervised data mining）

群集分析（cluster analysis）

8.6.2 有监督的数据挖掘

8.6.3 购物篮分析

## 8.7 Access工作台：第八部分——使用Access的商业智能系统