数据库

第一章 数据库系统概论

数据的完整性：数据的正确性、有效性和相容性

数据模型**三要素**：

- 数据结构
- 数据操作
- 数据完整性约束

数据库三级模式：

- 外模式
- 模式
- 内模式

两层映像与数据库独立性：

- 外模式/模式映像 -> 逻辑独立性
- 模式/内模式映像 -> 物理独立性

第二章 关系模型与关系代数

关系模型的几个基本概念：关系模型的数据结构、关系操作和关系完整性约束

关系数据结构

关系数据结构为：

- **关系**
  - 关系模型的数据结构非常简单，它就是二维表，亦称为**关系**
  - **关系数据库**就是表的集合，即**关系的集合**
- **关系模式**
  - 表的**表头部分**对应于关系模式，关系模式是型的概念
- **码**
  - 超码：属性集A能唯一地标识关系r中的一个元组
  - 候选码：属性集A是关系r的超码，且属性集A的任意真子集都不能成为关系r的超码
    候选码是最小的超码
  - 主码：若一个关系有多个候选码，则可以选定其中一个候选码作为该关系的主码
  - 外码：设F是关系r的一个属性（集），K_s是关系s的主码。如果F与K_s相对应，则称F是关系r**参照关系**s的外码
- **关系数据库模式**
  - 关系数据库也有值和型之分，关系数据库的**型**就是**关系数据库模式**；值就是关系数据库实例

关系操作

关系模型中的关系操作有**查询操作**和**关系操作**两大类

5个基本关系操作：**选择、投影、集合并、集合差和笛卡尔积**

关系完整性约束

- 实体完整性
  - 若属性集A是关系r的主码，则A不能取null
- 参照完整性
  - 若关系r的外码参照关系s的主码，则对于关系r中的每一个元组在属性F上的取值，要么取null，要么等于关系s中某个元组的主码值
- 用户自定义完整性

第三章 SQL查询语言

1. SQL特点

SQL语言由4部分组成：

- **数据定义语言 DDL (Data Definition Language)**
  - 定义数据库的逻辑结构，包括数据库、基本表、视图和索引等
  - DDL包括3类语言，即定义、修改和删除
  - **CREATE、ALTER、DROP**
- **数据操纵语言 DML (Data Manipulation Language)**
  - 对数据库的数据进行**检索**和**更新**，其中更新操作包括**插入、删除和修改数据**
  - **SELECT、INSERT、DELETE、UPDATE**
- **数据控制语言 DCL (Data Control Language)**
  - 对数据库的对象进行授权、用户维护、完整性规则定义
  - **GRANT、REVOKE**
- 其他
  - 主要是嵌入式SQL语言和动态SQL语言定义
  - 扩展SQL还包括数据库数据的重新组织、备份和恢复功能

2. 单表查询

SQL查询语句的基本结构包括3个子句：SELECT（投影运算）、FROM（笛卡尔积）和WHERE（选择运算）

投影运算 SELECT语句实例：

- 查询指定列

```
SELECT classNo, className
FROM Class

# 从Class表中依次取出每个元组，选取classNo, className属性的值形成一个新元组
```

- 消除重复元组 **distinct DISTINCT**

```
SELECT DISTINCT institute
FROM Class

# 使用关键字 DISTINCT
```

- 别名

```
SELECT institute 所属学院, classNo 班级编号
FROM Class

SELECT institute AS 所属学院, classNo AS 班级编号
FROM Class

# 该查询可使用AS关键字取别名，也可以不用
```

- 计算列

```
SELECT courseNo, lower(courseName) 课程名, courseHour/16 AS 周课时

# 函数lower()将大写字母改为小写字母 /16直接运算
```

选择运算 WHERE语句实例：

- 比较运算

```
\>、>=、<、<=、=、<>(或!=)
```

- 范围查询

```
[NOT] BETWEEN <值1> AND <值2>
```

- 集合查询

```
SELECT studentName, native, classNo
FROM Student
WHERE native [NOT] IN ('上海', '北京')
```

- 空值查询

```
SELECT *
FROM Course
WHERE priorCourse [IS] NULL

# IS NULL是判断空值的重要手段
```

- 字符匹配

```
[NOT] LIKE <匹配字符串> [ESCAPE<换码字符>]

# %表示任意长度的字符串
# _表示匹配任意一个字符
```

- 逻辑查询
  复合条件查询，使用AND、OR、NOT进行逻辑与、逻辑或、逻辑非运算

```
SELECT *
FROM Class
WHERE courseNo='001' OR courseNo='002'

# AND、OR、NOT
```

排序运算 ORDER BY实例：

- 排序运算
  - ```
    SELECT *
    FROM Student
    WHERE sex = '女'
    ORDER BY classNo, month(birthday) DESE

    # ORDER BY <表达式1>[ASC|DESC][,<表达式2>[ASCIDESC] ... ]
    ```
  - 缺省值为**ASC**，按**升序**排列
  - 如果**降序**排列，则必须指明**DESC**
  - 先按照<表达式1>的值进行排序；当<表达式1>值相同时，再按照<表达式2>值排序

聚合查询 实例：

SQL提供的聚合函数包括：

- 运算函数
    ```

```
count([DISTINCT | ALL] {*|<列名>}) 统计关系的元组个数或一个列中的值

sum([DISTINCT | ALL] <列名>) 统计一列中值的总和(必须为数值型)

avg([DISTINCT | ALL] <列名>) 统计一列中值的平均值(必须为数值型)

max([DISTINCT | ALL] <列名>) 统计一列中值得最大值

min([DISTINCT | ALL] <列名>) 统计一列中值得最小值
```

如果指定**DISTINCT**，则会在计算时消除<列名>中的重复元组

- 分组计算

```
SELECT studentNo, count(*) 门数, avg(score) 分数
FROM Score
GROUP BY studentNo
```

按照学号进行分组，将具有相同studentNo值的元组作为一组，然后对每组进行相应的计数和求平均值

3. 连接查询

连接实例：

- **等值/非等值连接**
  - >、=<、 <=、=、 <>(或!=)
  - 当比较运算符为=表示**等值连接**，其他为**非等值连接**
  - ```
    SELECT a.studentNo, studentName, b.courseI o, b.score
    FROM Student a, Score b, Score c  // 表a与表b的连接条件WHERE
    a.studentNo=b.studentNo  // 表a与表c的连接条件
    AND a.studentNo=c.studentNo  // 表b上的选择条件
    AND b.courseNo=001 AND c.courseNo=002  // 表c上的选择条件
    ORDER BY a.studentNo
    ```
- **自然连接**
  - SQL不直接支持自然连接，完成自然连接的方法是在等值连接的基础上消除重复列
  - ```
    [例3.36] 实现成绩表Score和课程表Course的自然连接
    SELECT studentNo, a.courseNo, score, courseName,creditHour, courseHour,
    priorCourse
    FROM Score a, Course b // 表a与表b的连接条件
    WHERE a.courseNo=b.courseNo
    ```
- **自表连接**
  - 某个表与自己进行连接，成为自表连接

- ```
  [例3.37]查找同时选修编号001和002课程的同学 (B, C表查询)
  SELECT a.studentNo, studentName, b.courseo, b.score, c.courseNo, c.score
  FROM Student a, Score b, Score c
  WHERE b.courseNo='001' AND c.courseNo='002'
  AND a.studentNo=b.studentNo AND b.studentNo=c.studentNo
  ORDER BY a.studentNo
  ```

- **外连接**
  - 在例3.39中，出现了查询结果没有“金融管理15-01班”的情况，因为该班没有学生。
    但是在实际应用中，往往需要将不满足连接条件的元组也要查询出来，只是在相应地方用空值替代
  - 左外连接：**对于左关系没有连接上的元素，右关系用空值替代** LEFT OUTER JOIN...ON...

    ```
    SELECT className
    FROM Class a
    LEFT OUTER JOIN Student b ON a.classNo=b.classNo
    WHERE grade=2015
    ```

	className	institute	studentNo	studentName
1	计算机科学与技术15-01班	信息管理学院	1500001	李小勇
2	计算机科学与技术15-01班	信息管理学院	1500004	张可立
3	计算机科学与技术15-02班	信息管理学院	1500005	王红
4	金融管理15-01班	金融学院	NULL	NULL
5	信息管理与信息系统15-01班	信息管理学院	1500002	刘方晨
6	信息管理与信息系统15-01班	信息管理学院	1500003	王红敏

  - 右外连接：RIGHT OUTER JOIN...ON...
  - 全外连接：FULL OUTER JOIN

4. 嵌套子查询

在SQL查询中，一个SELECT-FROM-WHERE查询语句成为一个查询块，将一个查询块嵌入到另一个查询块的WHERE子句或HAVING子句中，成为嵌套子循环

SQL允许多层嵌套子查询，**但在子查询中，不允许使用ORDER BY子句**，该子句仅用于最后结果排序

嵌套查询分为**相关子查询**和**非相关子查询**：

- 非相关子查询指子查询的结果不依赖于上层查询
- 相关子查询指当上层查询的元组发生变化时，其子查询必须重新执行

嵌套子查询实例：

- **使用IN的子查询**
  - IN的含义为“属于”
  - ```
    # 在学生表Student中，将学号出现在成绩表Score中的学生姓名查询出来
    SELECT studentName
    FROM Student
    WHERE Student.studentNo IN (SELECT Score.studentNo FROM Score)
    ```

    - 首先执行最底层的子查询块，将该子查询块的结果作为中间关系
    - 执行上一层(即外一层)查询块，对于得到的每个元组，判新该元组是否在它的子查询结果中间关系中
        - 如果在，取出该元组中的相关属性作为最终输出结果的一个元组
        - 否则舍弃该元组
    - 不断重复直至执行完最上层查询块
- **使用比较运算符的子查询**
    - 在比较运算符中，经常会使用谓词**ANY(或SOME)和ALL**
    - =ANY，>=ANY（大于等于任意值），!=ANY；<=ALL（小于等于任意值），<>ALL....
    - ```
      # 查询所选课程成绩大于所有002课程成绩的所有同学信息
      SELECT *
      FROM Score
      WHERE score > ALL(SELECT score FROM Score WHERE courseID='002')
      ```
    - **=ANY等价于IN谓词，!=ALL等价于NOT IN谓词**

聚合函数注意点：

作业

1. 若要撤销数据库中已经存在的表S,可用(C).

选择一项：

A. DELETE TABLE S

B. DELETE S

C. DROP TABLE S

D. DROP S

delete只删除表中的数据，表结构不会被删除，而drop的执行结果是销毁整个表，包括表结构

```
# DELETE用法
DELETE FROM table_name
WHERE condition;

# DROP TABLE用法
DROP TABLE table_name
```
```

2. 设关系数据库中一个表S的结构为S(SN,CN,grade),其中SN为学生名,CN为课程名,二者均为字符型;grade为成绩,数值型,取值范围0-100.若要把"张二的化学成绩80分"插入S中,则可用(D).

A. ADD INTO S VALUES('张二', '化学', '80')

B. INSERT INTO S VALUES('张二', '化学', '80')

C. ADD INTO S VALUES('张二', '化学', 80)

D. INSERT INTO S VALUES('张二', '化学', 80)

```
# INSERT插入用法
INSERT INTO table_name
VALUES (value1,value2,value3,...);
```

3. 设关系数据库中一个表S的结构为:S(SN,CN,grade),其中SN为学生名,CN为课程名,二者均为字符型;grade为成绩,数值型,取值范围0-100.若要更正王二的化学成绩为85分,则可用(A) .

选择一项:

A. UPDATE S SET grade = 85 WHERE SN = '王二' AND CN = '化学'

B. UPDATE S SET grade = '85' WHERE SN = '王二' AND CN = '化学'

C. UPDATE grade = 85 WHERE SN = '王二' AND CN = '化学'

D. UPDATE grade = '85' WHERE SN = '王二' AND CN = '化学'

```
# UPDATE更新用法
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

4. 若要在基本表S中增加一列CN(课程名),可用(C).

选择一项:

A. ADD TABLE S CN CHAR(8)

B. ADD TABLE S ALTER CN CHAR(8)

C. ALTER TABLE S ADD CN CHAR(8)

D. ALTER TABLE S ADD (CN CHAR(8))

```
# ALTER增加列用法
ALTER TABLE table_name
ADD/DROP column_name datatype;
```

5. 在视图上不能完成的操作是(C) .

选择一项:

A. 更新视图

B. 查询

C. **在视图上定义新的表**

D. 在视图上定义新的视图

6. SQL语言是(B)的语言,容易学习

选择一项:

A. 过程化

B. 非过程化

C. 格式化

D. 导航式

第四章 数据库建模

数据库设计过程

- 需求分析
- 概念设计
  - 设计E-R模型
- 逻辑设计
  - 将E-R模型转化为关系数据库模式
- 模式求精
  - 减少数据冗余、消除更新、插入与删除异常
- 物理设计
  - 包括数据库文件组织格式、内部存储结构、建立索引、表的聚集
- 应用和安全设计

E-R模型基本概念及表示

E-R模型采用实体集、属性集和属性三个基本概念来分别描述事物、联系和特征

- **实体和实体集**
  - 实体是客观世界中可区别于其他事物的"事物"和对象
  - 实体集是指具有相同类型即相同性质（或属性）的实体集合
- **属性**
  - 属性是实体集中每个实体都具有的特征描述
  - 可分为：简单属性和复合属性、单值属性和多值属性（双椭圆表示）、派生属性（虚线椭圆表示）
- **联系与联系集**
  - 联系是指多个实体间的相互关联
  - 多联系：实体之间有多种不同的联系
  - 实体的角色：实体在联系中的作用叫做实体的角色
  - 联系集的度：**参与联系集的实体集数目**称为联系集的度

E-R模型转化方法

从E-R模型转化为关系模式：

- **强实体集转化方法：**将实体集的每个属性对应为关系模式的属性，实体集的码作为关系的码
- **弱实体集转化方法：**弱实体集对应的关系模式属性由弱实体集本身的描述属性**（部分码）**加上所依赖的**标识实体集的主码**属性组成
- **属性集转化方法：**一个联系集映射而成的关系模式的属性，由来自参与联系的所有实体集的主码属性和联系本身的描述属性组成。
    - 生成模式的主码：
        - 一对一联系集：主码可选择任何一个参与实体集的主码
        - 一对多(多对一)联系集：主码由"多"的一方的实体集的主码组成
        - 多对多联系集：主码由所有参与实体集的主码的**并集**组成
    - 特别地，当为一对多和一对一联系集时可以不转化为单独的关系模式
        - **一对多，多的一方加少的外码和联系的属性**
        - 一对一，将某一方的主码属性及联系集属性的描述增加到另一方实体集所转化的关系模式中去

- **复合属性及多值属性转化方法：**
    - 对于复合属性，对每个子属性创建一个单独的属性，而不是复合属性自身创建一个单独的属性。例如**地址应该转化为省、城市、街道**
    - 对于多值属性，则必须为其**创建一个新的关系模式**，其属性为多值属性所在的实体集（或联系集）的**主码属性**和**该多值属性对应的属性**组成，**主码为全部属性**
        - PhoneNumber(studentNo, pNumber)
- **类层次转化方法：**父类实体集和子类实体集分别转化为单独的模式。其中，父类实体集对应的关系模式属性为父类实体集的属性，而各个子类实体集对应的模式由该子类的特殊属性和父类实体集的主码属性组成。**子类的主码与父类实体集的主码相同**
    - Student(**stuNo**, name, sex, birthday)
    - Undergraduate(**stuNo**, interest)
    - Graduate(**stuNo**, direction)

第五章 模式求精

对于给定的关系R（A1,A2,……,An）和其上的函数依赖集F，可将其属性分为如下四类：

- **L类**：仅出现在F的函数依赖左侧的属性
- **R类**：仅出现在F的函数依赖右侧的属性
- **N类**：在F的函数依赖左右两侧均未出现的属性
- **LR类**：在F的函数依赖左右两侧均出现的属性

码=超键：能够唯一标识一条记录的属性或属性集

候选键=候选码：能够唯一标识一条记录的**最小**属性集

主键=主码：某个能够唯一标识一条记录的最小属性集（是从候选码里人为挑选的一条）

主属性：**包含在任一候选码中的属性**称主属性。简单来说，主属性是候选码所有属性的并集

非主属性：不包含在候选码中的属性称为非主属性。 非主属性是相对于主属性来定义的

基于函数依赖理论，关系模式可以分成：

- 第一范式（1NF）：所有属性都是原子的
- 第二范式（2NF）：不存在**非主属性**对**候选码**的**部分依赖**
- 第三范式（3NF）：不存在**非主属性**对**候选码**的**传递依赖**
- BC范式（BCNF）：排除了任何属性（包括主属性和非主属性）对候选码的部分依赖以及传递依赖，并且排除了**主属性之间的传递依赖**

第七章 SQL数据定义、更新及数据库编程

SQL数据定义语言

表定义及约束定义：

其中注意：

- int(10)这里的“10”指**显示的域宽**，只是显示的数据长度而已
- decimal(p,n)这种就是指，p为几位**有效数字**，n是**小数点后有几位**

基本表的创建

```
CREATE TABLE <tableName>(
    /*定义表的列*/
    <columnName> <dataType> [DEFAULT <defaultValue>] [NULL|NOT NULL],
    /*举个例子*/
    courseHour    tinyint    DEFAULT 0                NOT NULL,

    /*定义约束*/
    [CONSTRAINT<constraintName>] {UNIQUE|PRIMARY KEY}(<columnName>, ...),

    /*定义外键约束*/
    [CONSTRAINT<constraintName>] FOREIGN KEY(<columName>, ...)
REFERENCE [<dbName>.owner.]<refTable>(<refColumn>, ...),

    /*定义自定义约束*/
    CHECK()
```

```
)
```

```
CREATE TABLE Course(
    courseNo char(3) NOT NULL,
    courseName varchar(30) UNIQUE NOT NULL,
    priorCourse char(3) NULL,
    studentNo char(7) NOT NULL CHECK(studentNo LIKE '[0-9][0-9][0-9]')

    /*建立命名的主码约束和匿名的外码约束*/
    CONSTRAINT CoursePK PRIMARY KEY (courseNo),
    FOREIGN KEY (priorCourse) REFERENCES Course(courseNo)  /*自表连接*/
    constraint cNoCK CHECK (courseNo LIKE '[0-9][0-9][0-9]') /*用户自定义完整性*/
)
```

基本表的修改

ALTER 是一种用于**修改数据库表结构**的关键字

使用**ALTER TABLE**来**修改**基本表的结构，**不可以删除列，一次只能进行一次操作**

```
/*增加一列*/
ALTER TABLE <tableName>
    ADD <columnName> <dataType>

ALERT TABLE TempTable
    ADD xsex int DEFAULT 0

/*增加约束*/
ALERT TABLE <tableName>
    ADD CONSTRAINT <constraintName>

ALERT TABLE Temptable
    ADD CONSTRAINT UniqueXname UNIQUE(xname)

/*删除约束*/
ALTER TABLE <tableName>
    DROP <constraintName>

ALERT TABLE Temptable
    DROP TermPK

/*修改列的数据类型*/
ALERT TABLE Template
    ALERT COLUMN xname char(10)
    MODIFY COLUMN xname char(10)
```

基本表的删除

```
DROP TABLE <tableName> [RESTRICT]
DROP TABLE TempTable
/*RESTRICT:删除有限制，该表不能有视图、触发器，不能被其他表引用。默认*/
/*CASCADE:删除没有限制，删除表的同时也会删除建立在该表上的索引、完整性规则等。*/
```

☆SQL数据更新语言

数据操纵语言DML，用于对数据库的数据进行**检索（查询）和更新**

插入数据

1.插入一个元组

```
INSERT INTO <TableName>[(<columnName>,...)]
    VALUES(<value1>,...)
/*如果有[(<columnName>,...)]，就按照[(<columnName>,...)]的顺序将VALUE的值插入，如果没有
[(<columnName>,...)]，就按照table本身的顺序来插入。如果数值没有列数那么多，那么会把剩下的列
数值取空。*/
/*最好抄一下属性*/

INSERT INTO Student
    VALUES('0700006','李相东','男','1991-10-21 00:00','云南','撒呢族','CS0701')

INSERT INTO Student(studentName,birthday,studentNo)
    VALUES('李章立','1991-10-12 00:00','0700007')
/*本例按照指定属性的顺序和属性的个数向学生表Student插入一个新元组，没有列出的属性列自动取空值
NULL或默认值。插入新元组时，数据的组织可不按照基本表结构定义的属性个数和顺序进行插入。*/
```

2.插入多个元组

即将一个查询结果插入到一个基本表中

```
INSERT INTO<tableName>[(<columnName1>,...)]
    <subquery> /*由SELECT引出的子查询/*
```

例如：将少数民族的选课信息插入基本表StudentNation中

```
/*创建新表*/
CREATE TABLE StudentNation (
    studentNo char(7) NOT NULL,
    courseNo char(3) NOT NULL,
    termNo char(3) NOT NULL,
    score numeric(5, 1) DEFAULT 0 NOT NULL CHECK( score BETWEEN 0.0
AND 100.0),
    CONSTRAINT StudentNationPK
    PRIMARY KEY (studentNo, courseNo, termNo)
)

/*插入查询结果*/
INSERT INTO StudentNation
    SELECT *
    FROM Score
    WHERE studentNo IN (SELECT studentNo FROM Student WHERE nation<>'回族')
```

将汉族同学的选课信息插入到StudentNation表中：

```
INSERT INTO StudentNation(studentNo, courseNo, termNo)
    SELECT studentNo, courseNo, termNo
    FROM Score
    WHERE studentNo IN (SELECT studentNo FROM Student WHERE nation='汉族' )
```

这里只把学号、课程号和学期号插入到基本表StudentNation中了，成绩列取0值，因为有默认值

3.删除数据

```
DELETE FROM <tableName> [WHERE <predicate>]
```

`WHERE <predicate>`指出了被删除元组满足的条件，可以省略，**省略就表示删除表中的所有元组。**
WHERE子句中可以包含子查询

```
/*删除学号为1600001同学的选课记录*/
DELETE FROM Score
WHERE studentNo='1600001'

/*删除选修了“高等数学”课程的选课记录*/
DELETE FROM Score
WHERE courseNo IN ( SELECT courseNo FROM Course WHERE courseName='高等数学' )

/*删除平均分在60到70分之间的同学的选课记录*/
DELETE FROM Score
WHERE studentNo IN (SELECT studentNo FROM Score HAVING avg(score) BETWEEN 60 AND
70 )
```

4.修改数据

```
UPDATE <tableName>
SET <cloumnName1>=<expr1>
```

实例：

```
/*将王红敏同学在151学期选修的002课程的成绩改为88分*/
UPDATE Score
SET score=88
WHERE courseNo='002' AND termNo='151'
    AND studentNo IN
                (SELECT studentNo FROM Student WHERE studentName='王红敏')

UPDATE Score
SET score=88
FROM Score a, Student b
WHERE a.studentNo = b.studentNo AND courseNo='002' AND termNo='151'


/*将每个班级的学生人数填入到班级表的ClassNum列中*/
UPDATE Class
SET classNum=sCount
FROM Class a, ( SELECT classNo, count(*) sCount FROM Student GROUP BY classNo )
```

```
b
WHERE a.classNo = b.classNo
```

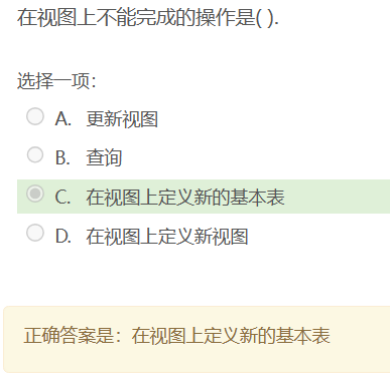
☆视图

☆视图的概念：视图是**虚表**，是从**一个或几个基本表（或者视图）中导出**的表。当基本表中的数据发生变化时，从视图中查询出的数据也随之改变。**视图**实现了数据库管理系统**三级模式中的外模式**

☆视图的主要作用是：

- 简化用户的操作
- **使用户能以多种角度看待同一数据库模式**
- 对重构数据库模式提供了一定程度的逻辑独立性
- **能够对数据库中的机密数据提供一定程度的安全保护**
- 适当的利用视图可以更清晰的表达查询

视图是一个虚表，自然和删除表DROP TABLE类似



游标

游标概念：是系统为用户开设的一个数据缓冲区，用于存放SQL语句的执行结果(元组集合)

使用游标的五个步骤：

- 定义游标：DECLARE
- 打开游标：OPEN
- **逐行提取游标集中的行：FETCH**
- 关闭游标：CLOSE
- 释放游标：DEALLOCATE

存储过程

概念：存储过程（Stored Procedure）是为了完成特定功能汇集而成的**一组命名了的SQL语句集合**

- 该集合**编译后存放在数据库**中，可根据实际情况重新编译
- 存储过程可直接在服务器端运行，也可在客户端远程调用运行，**远程调用时存储过程还是在服务器端运行**

存储过程的优点：

- 将业务操作封装
- 便于事务管理
- 实现一定程度的安全性保护
- 减少网络通信量，特别适合统计和查询操作

修改存储过程使用的语句是**ALTER PROCEDURE**

创建存储过程使用的语句是**CREAT PROCEDURE**

删除存储过程使用的语句是**DROP PROCEDURE**

下面几组命令,将变量count值赋值为1的是( ).

选择一项:

- A. DECLARE @count SELECT @count=1
- B. DIM count=1
- C. DECLARE count SELECT count=1
- D. DIM @count SELECT @count=1

在SQL Server 编程中,程序块的范围使用()来限定.

选择一项:

- A. { }
- B. BEGIN-END
- C. ()
- D. []

正确答案是：BEGIN-END

关于存储过程的描述正确的一项是( ).

选择一项:

- A. 存储过程的存在独立于表,它存放在客户端,供客户使用
- B. 存储过程只是一些T-SQL语句的集合,不能看作数据库系统的对象
- C. 存储过程可以使用控制流语句和变量,大大增强了SQL的功能
- D. 存储过程在调用时会自动编译,因此使用方便.

正确答案是：存储过程可以使用控制流语句和变量,大大增强了SQL的功能

触发器

触发器是用户定义在关系表上的一类由事件驱动的存储过程，由**服务器自动激活。是一种特殊的存储过程**，用于保证完整性

例题

在使用命令INSERT NTO <表名> [(列名...)] VALUSE(<值>)时,下列描述错误的是().

选择一项:

- A. INSERT语句中列名的顺序可以与表定义时的列名顺序一致
- B. INSERT语句中列名的顺序可以与表定义时的列名顺序不一致
- C. INSERT语句中值的顺序可以与列名的顺序不一致
- D. INSERT语句中值的顺序必须与列名的顺序一致

正确答案是：INSERT语句中值的顺序可以与列名的顺序不一致

第八章 数据库存储结构与查询处理

索引和散列

☆索引的基本概念、作用

两种基本索引类型：

- 顺序索引：**索引**中的**记录（索引项）基于搜索码值顺序排列**
  - 组织结构：在索引中按照**搜索码值的顺序存储索引项**，并将**索引项**与包含该**索引项中搜索码值**的文件记录关联起来（通过指针）
  - 用于支持快速地对**文件**中的记录进行**顺序**或**随机**地访问

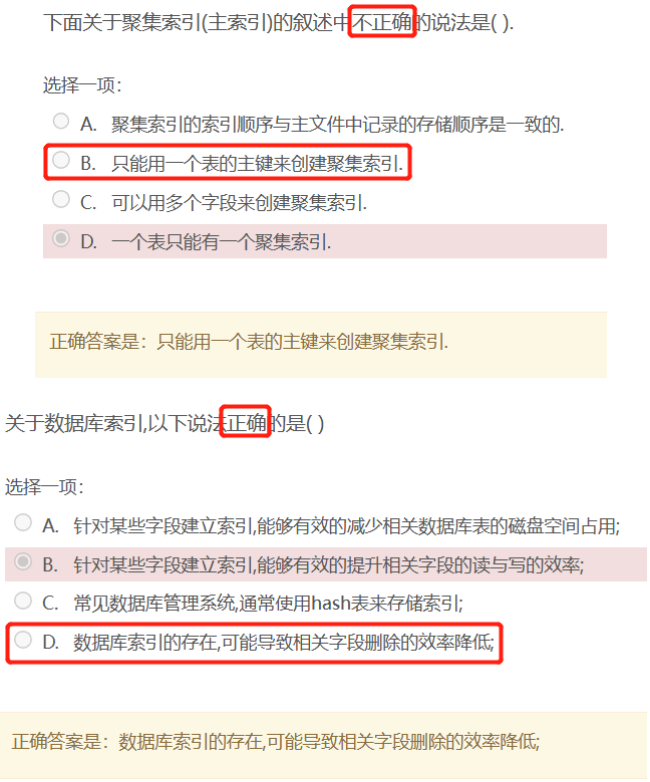
- 散列索引：索引中的**记录（索引项）**基于**搜索码值的散列函数（即哈希函数）的值**平均、随机地分不到若干个散列桶中

建立了索引的文件称为**索引文件（指数据文件）**。索引文件中的**记录**自身可以按照某种排列顺序存储。

一个索引文件可以有**多个索引**，分别对应于**不同的搜索码**

如果**索引文件**中的**记录**按照某个**搜索码值指定的顺序**物理存储，那么该**搜索码**对应的**索引**就称为**主索引**，也叫**聚集索引**

与此相反，**搜索码值**顺序与索引文件中记录的**物理顺序不同**的那些索引称为**辅助索引**或**非聚集索引**



查询处理

☆查询处理过程

查询处理过程包括：

- **语法分析和翻译**
  - **检查用户查询的语法**，并利用**数据字典**验证查询中的关系名、属性名等是否正确
  - **构造**该查询语句的**语法分析树**，并将其翻译成关系代数表达式
- **查询优化**
  - 执行查询，不仅需要提供关系代数表达式，还要对关系代数表达式加上**注释**来说明如何执行每个关系运算，**生成查询执行计划**
  - **不同的查询执行计划有不同的代价**，**构造具有最小查询执行代价的查询计划称为查询优化**，由查询优化器来完成
  - **查询优化是影响RDBMS性能的关键因素**
- **查询执行**
  - 查询执行引擎根据输入的**查询执行计划**，调用相关算法实现查询计算，并将计算结果返回给用户
  - 有效地对**内存缓存区**进行管理是影响查询执行性能的非常重要的方面

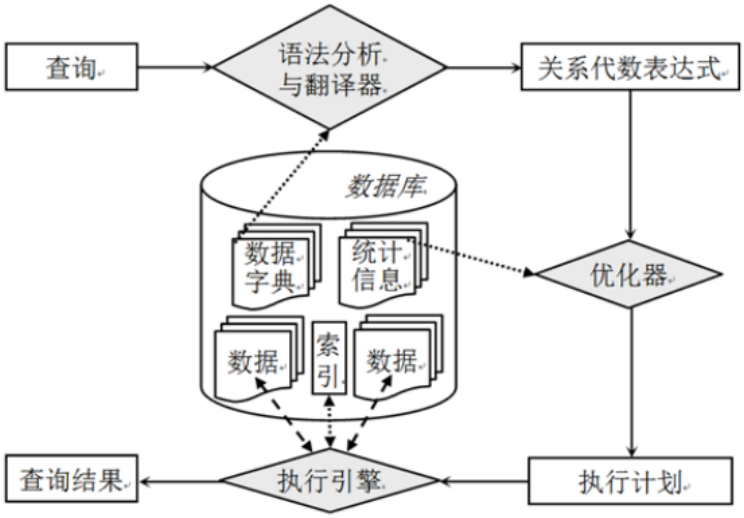


图 8-22　查询处理的过程

☆查询代价的度量

查询处理的代价可以通过该查询对各种资源的使用情况进行度量，主要包括：

- **硬盘存取时间**
- 执行一个查询所用的**CPU时间**
- 以及在并行/分布式数据库系统中的**通信开销**等

对于**大型数据库系统**而言，在**磁盘上存取数据的代价**通常是**最重要的代价**，可以通过传输磁盘块数以及搜索磁盘次数来度量

在代价估算时，通常假定是**最坏的情形**

查询优化

关系表达式的转换

■ 等价规则

- 合取选择运算的级联分解

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

- 选择运算满足交换律

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

- 系列投影的最后有效性

$$\prod_{A_1}(\prod_{A_2}(\ldots(\prod_{A_n}(E))\ldots)) = \prod_{A_1}(E)$$

- 选择操作与θ连接相结合

$$\sigma_{\theta}(E1 \times E2) = E1 \bowtie_{\theta} E2$$

$$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$$

☆查询优化的过程

查询优化分为3步进行：

- **逻辑优化**，即**产生**逻辑上与给定关系代数表达式**等价**的**关系代数表达式**
- **代价估计**，**即估计每个查询执行计划的代价**
- **物理优化**，即对所产生的表达式以不同方式作**注释**，产生不同的**查询执行计划**
- 查询优化器中第①步和第③步时交叉进行的

☆启发式优化规则

为了减少优化本身的代价，查询优化器通常都会或多或少地使用一些启发式规则

- **尽早执行选择操作**
- **尽早执行投影操作**【在尽早选择后再投影】
    - 投影运算像选择运算一样，也可以减少关系的大小，只要有可能就立即执行投影
    - **选择运算减小关系的潜力通常会比投影运算大**，因此，“尽早执行投影运算”的前提是先采用“尽早执行选择运算”的启发式规则

物理数据库设计

数据库在物理设备上的**存储结构**和**存取方法**称为**数据库的物理结构**，为一个给定的逻辑数据模型选取一个最适合应用环境的物理结构的过程，就是数据库的**物理设计**

目标：

- 提高数据库性能，以满足应用地性能需求
- 有效利用存储空间
- 在性能和空间之间做出最优平衡

内容：

- 确定数据库存储结构
    - 确定数据存放位置：将易变部分和稳定部分、经常存取部分和存取频率较低部分分开存放
    - 确定数据库存储结构：确定数据库存储结构要综合考虑**存取时间、存储空间利用率和维护代价**三个因素
- 确定数据库存取路径：主要指确定如何建立索引
    - 目前最常用的存储方法是**B + 树索引**
    - 索引存取方法的选择
        - 属性经常在查询条件中出现，则这一属性建立索引
        - 属性经常作为最大值和最小值等聚集函数的参数，则建立索引
        - 属性经常在连接操作的连接条件中出现，则建立索引
- 确定系统配置
    - 系统配置参数包括：同时使用数据库的用户数，同时打开数据库对象数，使用的缓冲区长度、个数，时间片大小，数据库的大小，装填因子，锁的数目等。这些参数影响存取时间和存储空间的分配
- 物理结构评价
    - 对时间效率、空间效率、维护代价和用户要求进行权衡
- 影响物理设计的主要因素

- 应用处理需求
- 数据特征
- 运行环境
- 物理设计的调整

第九章 数据库安全性与完整性

数据库安全性

概述

数据库安全保护的目标是确保只有**授权用户**才能访问数据库，未被授权的人员则无法接近数据

安全措施是指计算机系统中用户直接或通过应用程序访问数据库索要经过的**安全认证过程**

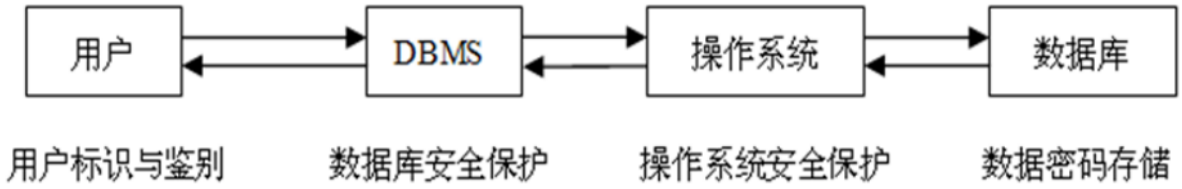


图 9-1　数据库安全认证过程

安全认证过程：

- 用户标识与鉴别：用户访问数据库时，要先将其**用户名和密码**提交给数据库管理系统进行认证
- 数据库安全保护：用户在数据库中执行什么操作，需通过**"存取控制"**或**视图**进行**权限分配**
  - **存取控制：**决定用户对数据库中**哪些对象**进行操作，进行**何种操作。**主要包括两部分：
    - 定义用户权限
    - 合法权限检查
  - **视图：**为不同用户定义不同的视图，达到**限制用户访问范围**的目的
    - **视图机制能够隐藏用户无权存取的数据**
    - 视图主要功能在于提供数据库的**逻辑独立性**
    - **将视图与存取控制机制结合使用。**先通过视图屏蔽一部分保密数据，然后进一步定义存取权限
  - **审计：**用于**跟踪并记录**有关数据的访问活动
    - 审计追踪把用户对数据库的所有操作自动记录下来，存放在审计日志中
    - 利用审计日志信息，可找出非法存取数据库的人、时间和内容等
    - 审计是可选特征
- 操作系统安全保护：通过操作系统提供的安全措施来保证数据库的安全性
- 数据密码存储：对敏感数据进行**"加密存储"**
  - 数据加密：防止数据库中数据存储和传输失密的有效手段
  - 加密的基本思想：将原始数据加密为不可直接识别的密文，然后数据以密文的方式存储和传输

数据库安全标准：

- 主体和客体
  - 主体：数据库的访问者，包括用户、进程、线程等
  - 客体：数据库的数据和载体，如基本表、视图、存储过程和数据文件等
- **自主存取控制：**基于**存取矩阵**的存取控制模型

    - 由**主体、客体、存取操作**三种元素组成，主体按存取矩阵的要求访问客体
- **强制存取控制：**适用于对数据由严格而固定密级分类的部门，如军事部门或政府部门。对数据本身进行密级标记，只有**符合密级标记要求的用户才可操纵数据**
    - 对主体和客体，为每个实例值指派了一个**敏感度标记**
    - 敏感度标记被分成若干级别，例如绝密、机密、可信、公开等
    - 主体的敏感度标记称为**许可证级别**，客体的敏感度标记称为**密级**
    - **对比主体和客体的敏感度标记**，确定主体是否能够存取客体
        - 当主体许可证级别大于等于客体的密级才能读客体；当主体的许可证级别等于客体密级才能写客体

目标权限授权和回收 （GRANT、REVOKE）

自主存取控制通过SQL的**GRANT**和**REVOKE**语句实现

使用GRANT和REVOKE语句，**向用户授予或收回对数据的操作权限**

例子：

```
/*授权 GRANT*/
GRANT{all | <command_list>} ON [(<columnName_list>)]
TO{public| <username_list>}[WITH GRANT OPTION]

/*SELECT 权限给用户*/ /*并且可以转授权限*/
GRANT SELECT ON <table_name> TO <user_name> WITH GRANT OPTION
GRANT all ON Score TO u1 WITH GRANT OPTION


/*回收 REVOKE*/
REVOKE {all | <command_list>} ON [(<columnName_list>)]
FROM {public | <username_list>} [CASCADE | RESTRICT]

/*回收用户SELECT权限*/
REVOKE SELECT ON <table_name> FROM <user_name>;
```

对象创建者自动拥有该对象的**更新（插入、删除和修改）和查询权限**；**过程创建者**自动拥有创建过程的**执行权限**

- CASCADE：级联收回；
- RESTRICT：赋予了权限就不能收回，默认
- WITH GRANT OPTION：将指定对象上 的目标权限授予其他用户

数据完整性

概念

数据库完整性是针对数据库中的数据进行正确性的维护，防止数据库中存在**不符合语义、不正确的数据。**

为维护数据库的完整性，数据库管理系统必须提供如下功能：

- 完整性约束条件定义机制
    - 完整性约束条件也称**完整性规则**，是数据库中数据必须满足的语义约束条件

- 完整性检查方法
  - **检查**数据**是否满足已定义的完整性约束条件**称为完整性检查
- 违约处理措施
  - 用户操作违背完整性约束条件，就采取一定措施

☆实现方法

- **实体完整性要求基本表的主码值唯一且不允许为空值**（强调主码）
  - 实体完整性定义使用CREATE TABLE语句中的**PRIMARY KEY**语句实现或者使用ALTER TABLE语句中的ADD PRIMARY KEY短语实现
  - 对单个属性构成的主码可定义为列约束，也可定义为元组约束；对多个属性构成的主码，只能定义为元组约束
- **参照完整性为若干个基本表中的相应元组建立联系**（强调外码）
  - 参照完整性定义使用CREATE TABLE语句中的**FOREIGN KEY和REFERENCES**短语来实现；FOREIGN KEY指出定义哪些列为外码，REFERENCES指明外码参照哪些关系
  - 给出FOREIGN KEY定义的关系称为参照关系；由REFERENCES指明的基本表称为被参照关系
  - 外码约束只能定义为表约束
- 用户自定义完整性就是定义某一具体应用中数据必须满足的语义要求，由RDBMS提供，而不必由应用程序承担
  - 列约束：包括数据类型、列值非空、列值唯一、设置默认值和满足**CHECK定义**等
  - 元组约束：设置不同属性之间的取值的相互约束条件，它也是用CHECK引出的约束

第十章 事务管理与恢复

事务

☆概念

事务的概念：对于**用户**而言，事务是具有完整逻辑意义的数据库操作序列的集合；对于**数据库管理系统**而言，事务则是一个读写操作序列，这些操作是一个不可分割的逻辑工作单元，**要么都做，要么都不做。**

☆特性

为了保证事务并发执行或发生故障时数据库的一致性（完整性），事务应具有以下ACID特性：

- **原子性**（atomicity）：事务的**所有操作**要么全部都被执行，要么都不被执行
- **一致性**（consistency）：一个单独执行的事务应保证其**执行结果的一致性**，即总是将数据库从一个一致性状态转化到另一个一致性状态
- **隔离性**（isolation）：当多个事务并发执行时，一个事务的执行不能影响另一个事务，即**并发执行的各个事务不能互相干扰**
- **持久性**（durability）：一个**事务成功提交**后，**它对数据库的改变必须是永久的**，即使随后系统出现故障也不会受到影响

在DBMS中实现事务持久性的子系统是( ).

选择一项:

- A. 安全管理子系统
- B. 完整性管理子系统
- C. 并发控制子系统
- D. 恢复管理子系统

正确答案是：恢复管理子系统

并发控制

事物并发可能产生的问题：

- **读脏数据**：如果T2读取到T1修改但未提交的数据后，如果T1撤销了操作，就会导致T2读取到不一样的数据
- **不可重复读**：指Ti两次从数据库中读取的结果不同
- **丢失更新**：两次以上的事物都读取了同一数据值并修改，最后提交的事务的执行结果覆盖了前面事务提交的执行结果

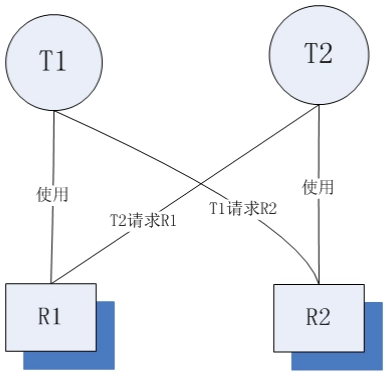
下列不属于并发操作带来的问题是( ).

选择一项:

- A. 丢失修改
- B. 不可重复读
- C. 死锁
- D. 脏读

正确答案是：死锁

死锁：是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去



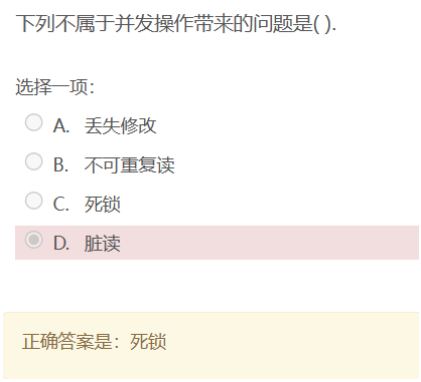
封锁概念

封锁：**当事务T需访问数据对象Q时，先申请对Q的锁。**如批准获得，则事务T继续执行，**且此后不允许其他任何事物修改Q**，直到事务T释放Q上的锁为止。

基本锁类型：

- **共享锁**：事务T获得数据对象Q的共享锁，则**事务T可以读Q但不能写Q**

- **排它锁**：事务T获得了数据对象Q上的排它锁，则**事务T既可以读Q又可以写Q**



基于封锁的协议

两阶段封锁协议：要求每个事务分两个阶段完成封锁操作，**增长（申请锁）**阶段和**缩减（释放锁）**阶段：

- 增长阶段：事务可以获得锁，但不能释放锁
- 缩减阶段：事务可以释放锁，但不能获得新锁

两阶段封锁协议可以**保证冲突可串行化**

对于级联回滚可以通过将两阶段封锁修改为**严格两阶段封锁协议**加以避免。严格两阶段封锁协议出了要求封锁是两阶段之外，还要求**事务持有的所有排它锁**必须在**事务提交之后**方可释放

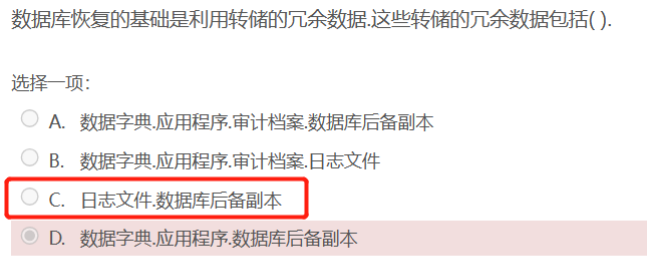
另一个两阶段封锁的变体是**强两阶段封锁协议**，它要求事务提交之前**不得释放任何锁**（包括共享锁和排它锁）

恢复与备份

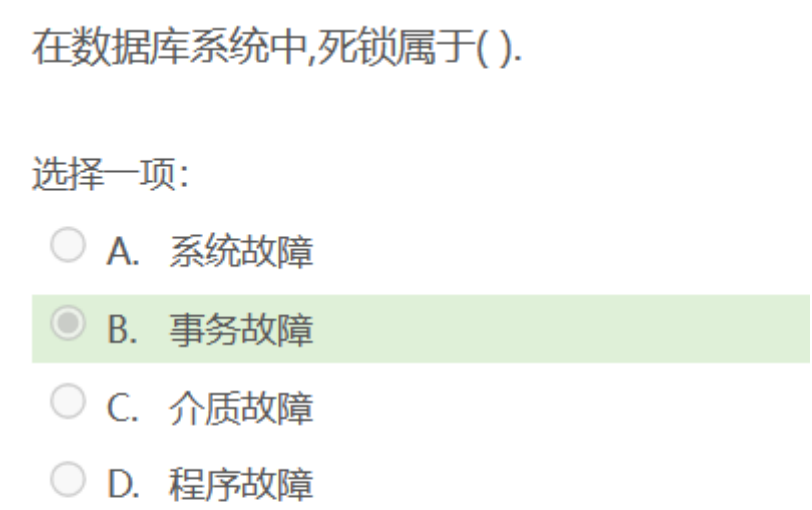
故障分类及恢复策略

- 事务故障
  - 概念：例如**某些应用程序的错误**以及**并发事务发生死锁**等，使事务未运行至正常终止点就夭折了，这种情况就是事务故障
  - 恢复策略：强行**回滚**夭折事务，清除其对数据库的所有修改，**使得该事务好像根本没有启动过一样**，称这类恢复操作为**事务撤销（UNDO）**
- 系统故障
  - 概念：由于某种原因造成**系统停止运行**，致使所有正在运行的事务都以非正常方式终止
  - 恢复策略：
    - **事务撤销（UNDO）所有未完成的事务——原子性要求**
    - **重做（REDO）所有已提交的事务—— 持久性要求**
- 介质故障
  - 概念：由于某种硬件故障，致使存储在外存中的数据部分丢失或全部丢失
  - 恢复策略：**装入**发生介质故障前某个时刻的**数据库数据副本**，并**重做（REDO）**自备份相应副本数据库之后的**所有成功事务**，将这些事务已提交的更新结果重新反映到数据库中去，**无需UNDO操作**
- 其他故障

- 黑客入侵、病毒、恶意流氓软件等引起的事务异常结束、篡改数据等不一致性
- 恢复策略：通过数据库的安全机制、审计机制等实现对数据的授权访问和保护



正确答案是：日志文件.数据库后备副本



正确答案是：事务故障

日志

☆概念

日志是DBMS记录**数据库全部更新操作**的序列文件

- 日志文件记录了数据库的**全部更新顺序**
- 日志文件是一个**追加文件**
- DBMS允许事务的**并发执行**导致**日志文件是"交错的"**
- **单个事务的日志顺序**与**该事务更新操作的执行顺序**是一致的
- 日志记录通常先是写到日志缓冲区中，然后写到稳固存储器（如磁盘列阵）中

事务日志用于保存( ).

选择一项:

A. 程序运行过程

B. 程序的执行结果

C. 对数据的更新操作

D. 数据操作

正确答案是：对数据的更新操作