《数据库设计与开发》

北京邮电大学软件学院
郭文明
(guolyz@sohu.com)
2003.06

0.前言

0.1 数据库发展与应用

0.2 本课程内容

0.3本课程学习方法

0.4本课程参考资料

0.1 数据库发展与应用

- 数据库定义：数据库是数据管理的技术，数据管理是指对各种数据进行收集、存储、加工和传播的一系列活动的总和。包括对数据的分类、组织、编码、存储、检索和维护。
- 数据处理的发展：数据处理的发展过程经历了手工管理、文件管理、数据库三个阶段。
- 数据模型：数据库是建立在数据模型基础上的数据集合。数据模型有：层次模型、网状模型、关系模型、面向对象模型。比较成熟完善、市场占用率较大的是基于关系模型基础上的关系数据库。

0.1 数据库发展与应用

- 数据库管理系统：数据库管理系统是位于用户与操作系统之间的一层数据管理软件，属系统软件。
- 数据库管理系统的功能：数据定义(create、drop等)；数据操纵(select、insert等)；数据库运行(安全性、完整性、并发控制、故障恢复)；数据库建立与维护(数据转储、数据恢复、性能监视、重组织)。
- 目前的一些数据库管理系统（DBMS），小型桌面数据库系统FoxPro，ACCESS，PRADOX等，大型数据库系统 ORACLE，INFORMIX，SYBASE，DB2，Microsoft SQLserver等 。

0.1 数据库发展与应用

- 数据库的应用已越来越广泛。从小型的单项事务处理系统到大型复杂的信息系统大都用先进的数据库技术来保持系统数据的整体性、完整性和共享性。
  - 1)电脑抽奖：采用数据库技术,不必修改程序,只变更奖项设置,即可应用于不同的抽奖活动。
  - 2)招生考试：通过对数据库排序和索引,完成考生按志愿从高分到低分排列,根据学校招生计划录取,录满为止。
  - 3)银行储蓄：为每个用户建立能够唯一识别的帐户,同时记录用户的往来明细,做到存储量大,运算速度快,准确程度高。

0.1 数据库发展与应用

- 企业(生产管理、质量管理、客户管理、财务管理、电子商务等)；机关(办公自动化)；金融(银行电子结算)；证券(股票交易、行情分析)；通讯(话费查询、费用结算)；教育卫生(网上录取、学籍管理)；社会保障(住房公积金、医疗保险、养老保险)；政府(电子政务)；交通(道路信息系统)等。
- 国民经济各个领域都离不开数据库技术，目前，一个国家的数据库建设规模(指数据库的个数、种类)、数据库信息量的大小和使用频度已成为衡量这个国家信息化程度的重要标志之一。

0.1 数据库发展与应用

- 数据库系统：数据库系统是指在计算机系统中引入数据库后的系统，一般由数据库、数据库管理系统、应用系统、DBA和用户构成。
- 数据库设计是建立数据库及其应用系统的技术，是信息系统开发和建设中的核心技术，具体说，数据库设计是指对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求(信息要求和处理要求)。这个问题是数据库在应用领域的主要研究课题。
- 数据库技术研究领域：DBMS的研制、数据库设计、数据库理论。

0.2 本课程内容

- 1）关系数据库与对象关系数据库：建立在关系模型基础上的数据操纵，融合了面向对象思想的对象关系数据库和对象关系SQL。
- 2）数据存储和数据库管理：数据库常用的文件存储、索引技术、散列技术和多键访问技术等数据存储技术，保证数据库正常运行的安全性、完整性控制和数据库恢复。
- 3）数据库应用程序体系结构：访问数据库的程序，事务处理技术，Client/Server体系结构，多层体系结构，分布式数据处理和中间件。

0.2 本课程内容

- 4）ORACLE数据库：ORACLE数据库的体系结构，存储管理，实例管理；ORACLE数据库的规划与实施。
- 5）数据库设计：需求分析、概念模型设计、数据库逻辑设计和物理设计、功能设计、数据库实施、CASE技术；ER模型、规范化设计方法。
- 另外，通过实验环节进一步理解所讲内容，实验即选择一DBMS平台、设计一实际的数据库应用系统。完成实验需具备DBMS知识、程序设计开发能力、适当的专业领域知识。（实验要求另发）。

0.3本课程学习方法

- 数据库设计与开发这一门课集理论性、技术性、艺术性为一体，学习过程中应根据这一些特点，针对性的采取一些措施。
- 1）从学习一个具体的数据库管理系统（DBMS）开始。理解数据管理的含义、理解数据库原理中的理论和方法。——离开具体的DBMS进行数据库设计是不完整的设计。
- 2）从熟悉一个具体的数据库应用系统开始。了解具体应用的领域知识和背景，结合DBMS的功能，探索利用DBMS为实际解决问题的过程。——离开深厚的领域知识，不可能设计出用户满意的数据库。

0.3本课程学习方法

- 3）从使用一个具体的编成语言开始。明确理解在数据库设计和开发过程中，哪些功能由DBMS完成，哪些由程序完成。——没有编程经验，不可能设计出编程人员满意的数据库。
- 4）从DBA的角度学。数据库应用系统的运行离不开DBA的大力支持，数据库设计开发除满足一般用户要求之外，还应方便DBA的数据管理。——数据库系统的运行没有DBA的参与，数据库系统的设计开发可能会感觉特别费劲。
- 5）多学习、多实践、多思考才能集理论性、技术性、艺术性为一体。

0.4本课程参考资料

- 1.数据库—原理、编程与性能, Patrick O'Neil著, 周傲英等译,机械工业出版社，2002.01

  (DATABASE Principles,Programming, and Performance. Patrick O'Neil Elizabeth O'neil,高等教育出版社,2001.05)
- 2.数据库设计教程, Thomas M.Connolly著,何玉洁等译,机械工业出版社,2003.06
- 3.Unix 和Linux 下的Oracle数据库管理, Michael Wesster 著,王华驹等译,人民邮电出版社, 2002.8
- 4.Oracle9i DBA Fundamentals I，ORACLE University
- 5.数据库实用教程, 丁宝康 董健全, 清华大学出版社,2001.09

1.关系数据库与对象关系数据库

- 关系数据库管理系统（RDBMS）使用灵活，即使用户不是程序员，也可轻松快捷地写出一般的查询语句。关系数据库管理系统建立在关系模型基础之上。最近几年，一种更新的数据模型——对象-关系模型在许多产品中正逐渐取代关系模型。建立在对象—关系模型基础之上的数据库管理系统称为对象-关系数据库管理系统（ORDBMS）。对象-关系数据库管理系统也支持关系数据库管理系统中的数据。

1.关系数据库与对象关系数据库

- 本章将介绍多年来支配数据表达方式的关系模型的概念和规则，从关系代数固有的查询能力方面介绍关系模型的特性，还要深入介绍工业标准SQL（SQL-99）语言。本章还将介绍对象-关系模型的特性和对象-关系SQL语法。

  1.1关系模型

  1.2关系代数

  1.3 SQL

  1.4对象关系数据库

1.1关系模型

- 1.1.1 关系数据库举例
- 1.1.2关系数据模型
  - 1.1.2.1域和数据类型
  - 1.1.2.2表和关系
  - 1.1.2.3关系规则
- 1.1.3数据模型
- 1.1.4最常用的数据模型

1.1.1 关系数据库举例

- 数据库是为了特殊目的存储在一起的相关数据记录的集合。关系数据库是按照关系模型组织存放数据的数据库。我们给出几个关系数据库的例子。
  - 1）产品—代理—销售数据库(CAP)

    顾客从代理商那里批发大量商品然后自己转销，顾客其实是零售商。顾客向代理商要求购买商品，每次订货要求有完整的记录。
  - 2）网上录取数据库(SCT)
  - 3）房屋销售数据库(HEC)

1) 产品-代理-销售数据库(CAP)

CUSTOMERS 顾客信息表

cid	顾客ID
cname	顾客姓名
city	顾客所在城市
discnt	顾客可能会有的折扣

AGENTS 代理商信息表

aid	代理商ID
aname	代理商名称
city	代理商所在城市
percent	代理商每笔交易所能获得的佣金百分比

PRODUCTS 商品信息表

pid	商品ID
pname	商品名称
city	商品库存所在城市
quantity	商品库存数量
price	商品批发价

ORDERS 订单信息表

ordno	订单ID
Month	订单月份
cid	顾客ID
aid	代理商ID
pid	商品ID
qty	数量
dollars	商品总价

2）网上录取数据库(SCT)

STUDENTS 学生信息表

Skh	考号	Sxm	姓名
Sxb	性别	Scsny	出生年月
Sdw	生源地	Syw	语文
Ssx	数学	Swy	外语
Sx	X	Szof	文化总分
Szg	照顾加分	Szgf	投档分
Szy1	一批志愿	Scy1	一批参志愿
Szy2	二批志愿	Scy2	二批参志愿
Sbz	标志(已投|在阅|录取|退档|死档等)		
Syx	院校码		
Ssj	时间		

COLLEGE 院校信息表

Cno	院校码
Cname	院校名称
Clb	院校类别
Cjh	院校计划数
Clq	院校录取数
Czy	院校在阅数
Ctd	院校投档数

TRACK 投档轨迹表

Skh	考号
Cno	院校码
Tbz	标志
Tsj	时间

2）网上录取数据库(SCT)运行机制

- 投档程序将考生按志愿从高分到低分排列，根据院校计划数和统一划定的重点院校、普通院校分数线进行投档：满足条件的考生其标志字段（Sbz）改为已投、其院校码字段(Syx)改为投档院校的编码；
- 院校根据投挡结果下载已投档到本学校的考生信息，此时考生其标志字段（Sbz）改为在阅；
- 院校通过阅读学生信息决定每一考生命运（录取、退档）；
- 考生标志字段（Sbz）的每一次修改，都会相应修改COLLEGE中的相关字段值，并且在TRACK中留下痕迹。

3）房屋销售数据库(HEC)问题

- 假设某房地产公司聘用多名业务员负责房地产的销售业务，每名客户可以多次或一次购买多套住房，每套住房有唯一的标识（房间号），每套房产交易都要签定合同，合同事例如下图所示：

合同号：01409090　日期：02/02/03　付款方式：一次性　总金额：495000.00元
客户身份证号：110102701104271　客户姓名：王刚　联系电话：88626868
地址：北京市白石桥路188号　邮政编码：100081

房间号	居室数	建筑面积(m^2)	使用面积(m^2)	单价(元/m^2)	金额（元）
102	3	110	90	3300.00	297000.00
103	2	78	60	3300.00	198000.00

务员代码：23　业务员姓名：李平　电话号码：84128996

- 试为该房地产公司设计一个关系数据库。

3）房屋销售数据库(HEC)问题

HOUSES　房屋表 房间号 居室数 使用面积 单价 销售合同号	EMPLOYEES　雇员表 业务员代号 业务员姓名 电话号码
CUSTOMERS 客户信息表 客户身份证号 客户姓名 客户住址 客户电话号码	ORDERS 合同表 合同号 日期 客户身份证号 业务员代码

1.1.2关系数据模型

- 单一的数据结构——关系

  关系模型的数据结构非常单一。在关系模型中，现实世界的实体以及实体间的各种联系均用关系来表示。在用户看来，关系模型中数据的逻辑结构是一张二维表。

- 关系操作

  关系模型给出了关系操作的能力，但不对RDBMS语言给出具体的语法要求。关系模型中常用的关系操作包括：选择(Select)、投影(Project)、连接(Join)、除(Divide)、并(Union)、交(Intersection)、差(Difference)等查询(Query)操作和增加(Insert)、删除(Delete)、修改(Update)操作两大部分。查询的表达能力是其中最主要的部分。

  关系操作的特点是集合操作方式，即操作的对象和结果都是集合。

1.1.2关系数据模型

- 关系的三类完整性约束

  关系模型允许定义三类完整性约束：实体完整性、参照完整性和用户定义的完整性。其中实体完整性和参照完整性是关系模型必须满足的完整性约束条件，应该由关系型数据库管理系统自动支持。用户定义的完整性是应用领域需要遵循的约束条件，体现了具体领域中的语义约束。

- 关系模型与以往的模型(层次模型、网状模型)不同，它是建立在严格的数学概念的基础上的。在用户观点下，关系模型中数据的逻辑结构是一张二维表，它由行和列组成。关系模型中称为关系、属性、元组。数据库中有两套术语：表、列、行；关系、属性、元组。

1.1.2.1域和数据类型

- 在目前的DBMS产品中（ACCESS、FOX、ORACLE、DB2 UDB、INFORMIX），创建数据库首先确定该数据库有多少表，每一表的名称是什么，表中包含哪些列，表的列必须确定类型。比如说，表CUSTOMERS中的discnt列的类型是real(实型)，city列的类型为char(20)（字符串）。
- 为什么在定义表的时候，必须给表的列加一个特定的类型呢？我们知道，在C或者Java等编程语言中，对于程序中使用的变量或常量等数据，事先应当申明其类型，具有相同类型的数据才可以进行一系列的比较和运算。数据库中给表的列加一个特定类型的原因也在于此。

1.1.2.1域和数据类型

- 在数据库原理或理论性文献中，一般地说，一个表的列的取值范围是在一集合D上，我们称D为域(Domain)。
- **定义1.1.1域(Domain)** 域是一组具有相同数据类型的值的集合。

  例如：自然数、整数、实数、字符串、日期、时间、{0,1}、{‘男’，‘女’}、{民族名|中国56个民族的名字}等等，都可以是域。

1.1.2.2表和关系

- **定义1.1.2 笛卡尔积(Cartesian Product)**

  给定一组域D1，D2，…，Dn，这些域中可以有相同的。D1，D2，…，Dn 的笛卡尔积为：

  $D1\times D2\times\ldots\times Dn=\{(d1,d2,\ldots,dn)\mid di\in Di,i=1,2,\ldots,n\}$

  其中每一个元素(d1,d2,…,dn)叫作一个n元组(n-tuple)或简称元组(Tuple)。元素中的每一个值di叫作一个分量(Component)。

  若Di(i=1,2,…,n)为有限集，其基数为mi(i=1,2,…,n)，则D1×D2×…×Dn的基数M为：

  $m1\times m2\times\ldots\times mn$

- 笛卡尔积可表示为一个二维表。表中的每行对应一个元组，表中的每列对应一个域。

1.1.2.2表和关系

- **定义1.1.3关系(Relation)** D1×D2×…×Dn的子集叫作在域D1，D2，…，Dn 的关系，表示为

  R(D1，D2，…，Dn)

  R这里只表示关系的名字，n是关系的目或度(Degree)。
- 关系是笛卡尔积的有限子集，所以关系也是一个二维表，表的每行对应一个元组，表的每列对应一个域。由于域可以相同，为了加以区分，必须对每列起一个名字，称为属性(Attribute)。n目关系必有n个属性。

1.1.2.2表和关系

- 若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为候选码(Candidate key)。若一个关系有多个候选码，则选定其中一个为主码(Primary key)。主码的诸属性称为主属性(Prime attribute)。不包含在任何侯选码中的属性称为非码属性(Non-key attribute)。
- 笛卡尔积中的许多元组是***无实际意义的***，从中取出有***实际意义***的元组来构造关系。在域的笛卡儿积中，我们称那些符合条件的元素为相关的，由此得到数学上的术语——关系。

1.1.2.2表和关系

- 表的列相对是稳定的，它描述表的结构。增加或删除列都不是常见的，我们也不希望在日常业务中这样。设计表的列布局就是数据库的逻辑设计。另一方面，表的内容是经常变化的，新的行被添加或被删除在日常业务中随时可能发生。数据库是关系的集合。组成关系的属性(列)的集合称为关系模式。数据库所有关系模式的集合构成数据库模式。
- 关系可以有三种类型：基本关系(基本表或基表)、查询表和视图表。基本表是实际存在的表，它是实际存储数据的逻辑表示。查询表是查询结果对应的表。视图表是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据。

1.1.2.3关系规则

- 基于笛卡尔积的子集的关系定义，在各种不同的商业数据库产品中又有一些规则，这些规则告诉我们在表结构中哪些变化是允许的，哪些检索操作是受限的，这些规则也是我们在进行数据库设计时需要充分考虑的。
- 关系规则是在关系模型的发明者E.F.Codd的一系列论文中提出，这些规则反映了特定的数学假设，目的是为了建立一个良好的关系结构。
- 出现新的对象-关系模型是因为人们认为从数学出发的关系规则有点太严格，无法容纳一些有价值的思想，特别是面向对象的系统分析思想出现之后。

规则1 列是同质的

- 每一列中的分量是同一类型的数据，来自同一个域。每一列为一个属性，要给予不同的属性名。该规则对应：设计表时，确定了表名之后，给每一个字段命名，并确定字段类型、长度。
- 注：不同的DBMS所支持的数据类型，能够处理的数据长度不一定相同。不同的DBMS当字段类型发生变化时，对已有数据的处理规则也不相同。例如：字段学号原来是字符型，现在该为数值型，那末已存在的字符型学号是否直接变成数值型。不同DBMS下的数据进行移动交换时，处理规则也不相同，数据导入导出时要特别注意此问题。对于来自于枚举型数值域的列，可通过参照表和被参照表的方式实现。

规则2 第一泛式规则

- 关系模型要求关系必须是规范化的，即要求关系中不允许含有多值属性和含有内部结构，遵守这样规则的表称为第一范式。关系的每一个分量必须是一个不可分的的数据项，不允许表中还有表。
- 第一范式规则是关系模型的基本规则，但在对象-关系数据库系统中，这一规则将不在是数据库设计的一个约束。以后我们会讨论对象-关系模型时知道，打破第一范式规则的途径是该模型中允许表中对象列的值是包含复杂类型的集合。用户可以自定义数据类型，某一字段可以是用户定义的复杂类型。

规则2 第一泛式规则

- 该规则表明重复字段在关系中是不允许的，这是关系数据库设计过程中的一个最基本的约束。例如，人事管理中常见的职工登记表有简历这一列，如果我们将简历这一列设计为备注型字段，则对曾经学习工作过的单位的查询基本不能完成，但如果将该列放入职工表的行中，则我们需要给表建立一定数目的列，这个数目要达到某一职工可能有的最多的学习工作单位数，但是这是不切实际的。一个有效的方法是将职工表分解成两部分，建立单独的职工表和简历表，职工表包括职工号等基本信息，简历表包括职工号、起始年月、终止年月、单位4个字段。

规则3 只能基于内容存取行规则

- 行的顺序可以任意交换，或者说，行是没有次序的，我们只可以通过行的内容即每一行中所存在的属性值来检索。因为关系是元组的集合，元组没有第一行、第二行之分。实际中，查询职工号为10001号职工的信息，没有查询第一行职工的信息。值得注意的是，关系模型要求关系中列也是没有次序的，列的顺序可以任意交换。这一点在“只能基于内容存取行”规则中已体现。
- 有些商业数据库系统打破了第三项规则，通过行标识（row identification，ROWID）提供一种用户检索表中行的方法。但其主要目的是帮助DBA检查在什么地方表中的行记录没有按正确方式存放。

规则4 行唯一性规则

- 任意两个元组不能完全相同，关系模型中关系是元组的集合，集合中完全相同的两元素只能被看作一个元素，集合不可以包含相同的元素。而且，规则3只能按列值存取要求元组必须唯一。
- 商业数据库系统中，很多并不是自动维护规则4的。

规则5 实体完整性规则

- 任意行在主键列上的取值都不允许为空值（NULL Value）。空值解释为未知的或者尚未定义的，当以后知道的情况下会重新填写该值。空值不同于数字0和空串。表的主键是用来唯一区分表的行的，不难想象，未知的或者尚未定义的实体我们是无法区分他们的。

1.1.3数据模型

- 数据模型是数据库的框架，这个框架表示了信息及其联系的组织方式和表达方式，同时反映了存取路径。数据模型是数据库管理系统分类的原因。一般的讲，任何一种数据模型都是严格定义的概念的集合。这些概念必须能够精确地描述系统的静态特性、动态特性和完整性约束条件。
- 1)数据结构：数据结构用于描述系统的静态特性。数据结构是所研究的对象类型的集合,是指对实体类型和实体间联系的表达和实现。通常按照数据结构的类型来命名数据模型。层次结构、网状结构和关系结构对应层次模型、网状模型和关系模型。

1.1.3数据模型

- 2)数据操作：数据操作是对系统动态特性的描述。数据操作是指对数据库中各种对象(型)的实例(值)允许执行的操作的集合。数据模型必须定义这些操作的确切含义、操作符号、操作规则(如优先级)以及实现操作的语言。
- 3)数据的约束条件：数据的约束条件是一组完整性规则的集合。是给定的数据模型中数据及其联系所具有的制约和依赖规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效、相容。此外，数据模型还应该提供定义完整性约束条件的机制。

1.1.4最常用的数据模型

- 目前，数据库领域中最常用的数据模型有四种：
  - 层次模型(Hierarchical Model)
  - 网状模型(Network Model)
  - 关系模型(Relational Model)
  - 面向对象模型(Object Oriented Model)

层次模型和网状模型统称为非关系模型。非关系模型的数据库系统在20世纪70年代至80年代初非常流行，在当时的数据库系统产品中占据了主导地位。关系模型的数据库系统在70年代开始出现，且发展迅速，并已逐渐取代了非关系模型的数据库系统的统治地位，现在流行的数据库系统大都是基于关系模型的。

1.1.4最常用的数据模型

- 20世纪80年代以来，面向对象的方法和技术在计算机各个领域，包括程序设计语言、软件工程、信息系统设计、计算机硬件设计等各方面都产生了深远的影响，也促进数据库中新一代数据模型——面向对象数据模型的研究和发展。目前没有实现。
- 在面向对象数据模型的启发下，目前许多关系型数据库管理系统扩展了部分对象模型功能——对象关系数据库。

作业：

1.试述数据库系统的组成和DBA的职责.

2. a)找出下面表的三个候选键.

A	B	C	D
a1	b1	c1	d1
a2	b3	c1	d2
a3	b4	c2	d2
a4	b2	c2	d1

b)给出一个四行四列的表,要求只有一个由前三列组成的候选键.

3.根据您的认识设计一个人事管理数据库,要求说明设计的背景及理由.

1.2关系代数

- 关系模型的重要部分是关系操纵，关系代数是一种抽象的查询语言，是关系数据操纵语言的一种传统表达方式，它是用对关系的运算来表达查询的。利用关系代数可以演示一个查询语言从关系数据库系统中检索信息的潜力，可以用最简单的形式来表达所有关系数据库查询语言必须完成的运算的集合，这些基本的运算对解释标准查询语言SQL如何被执行很有帮助，同时也有利于培养关系运算的思维能力。

1.2关系代数

- 关系代数的运算对象是关系，运算结果亦为关系。关系代数用到的运算符包括四类；集合运算符、专门的关系运算符、算术比较符和逻辑运算符。关系代数的运算按运算符的不同可分为传统的集合运算和专门的关系运算两类。其中传统的集合运算将关系看成元组的集合，其运算是从关系的“水平”方向即行的角度来进行。而专门的关系运算不仅涉及行而且涉及列。比较运算符和逻辑运算符是用来辅助专门的关系运算符进行操作的。

关系代数运算符

运算符	符号	含义	键盘格式	示例
集合运算符	∪	并	UNION	R∪S,或 R UNION S
	∩	交	INTERSECT	R∩S，或 R INTERSECT S
	-	差	MINUS	R-S，或 R MINUS S
	×	乘	TIMES	R×S,或 R TIMES S
专门关系运算符	σ	选择	R where C	$\sigma_{姓名=“张三”}(S)$或 S where 姓名=‘张三’
	π	投影	R[ ]	$\pi_{考号,姓名}(S)$ 或S[考号,姓名]
	∞	连接	JOIN	R∞S，或R JOIN S
	÷	除	DIVIDEBY	R÷S，或R DIVIDEBY S

关系代数中,这些运算经有限次复合后形成的式子称为关系代数表达式。

1.2.1传统的集合运算

- 传统的集合运算是二目运算，包括并、差、交、广义笛卡尔积。
- **定义1.2.1** 设关系R和关系S具有相同的目n(即两个关系都有n个属性)，且相应的属性取自同一个域，则可以定义并、差、交运算如下：
  - **并(Union)**：关系R与关系S的并记作：$R \cup S=\{t \mid t \in R \vee t \in S\}$

    其结果仍为n目关系，由属于R或属于S的元组组成。
  - **差(Difference)**：关系R与关系S的差记作：$R-S=\{ t \mid t \in R \wedge t! \in S \}$

    其结果仍为n目关系，由属于R而不属于S的所有元组组成。

1.2.1传统的集合运算

- 交(Intersection):关系R与关系S的交记作:
  $R \cap S=\{ t \mid t \in R \wedge t \in S \}$

  其结果仍为n目关系，由既属于R又属于S的元组组成。关系的交可以用差来表示，即$R \cap S=R-(R-S)$。

- 广义笛卡尔积(Extended Cartesian Product):两个分别为n目和m目的关系R和S的广义笛卡尔积是一个(n+m)列的元组的集合。元组的前n列是关系R的一个元组，后m列是关系S的一个元组。若R有k1个元组，S有k2个元组，则关系R和关系S的广义笛卡尔积有$k1 \times k2$个元组。记作:
  $R \times S=\{trts \mid tr \in R \wedge ts \in S \}$

传统集合运算举例

R

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1

S

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

R∪S

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1
a1	b3	c2

R∩S

A	B	C
a1	b2	c2
a2	b2	c1

传统集合运算举例

R

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1

S

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

R-S

A	B	C
a1	b1	c1

R×S

A	B	C	A	B	C
a1	b2	c1	a1	b2	c2
a1	b2	c1	a1	b3	c2
a1	b2	c1	a2	b2	c1
a1	b2	c2	a1	b2	c2
a1	b2	c2	a1	b3	c2
a1	b2	c2	a2	b2	c1
a2	b2	c1	a1	b2	c2
a2	b2	c1	a1	b3	c2
a2	b2	c1	a2	b2	c1

1.2.2 专门的关系运算

- 专门的关系运算包括选择、投影、连接、除等。为了叙述上的方便，先引入几个常用记号。
  - 设关系模式为R(A1，A2，…，An)。它的一个关系设为R。t∈R表示t是R的一个元组。t[Ai]则表示元组t中相应于属性Ai的一个分量。
  - 若A={Ai1，Ai2，…，Aik}，其中Ai1，Ai2，…，Aik是A1，A2，…，An中的一部分，则A称为属性列或域列。

1.2.2 专门的关系运算

- R为n目关系，S为m目关系。tr∈R,ts∈S,trts称为元组的连接。它是一个n+m列的元组，前n个分量为R中的一个n元组，后m个分量为S中的一个m元组。
- 给定一个关系R(X，Z)，X和Z为属性组。定义t[X]=x时,x在R中的象集(Images Set)为：

  $Zx=\{t[Z] \mid t\in R, t[X]=x\}$

  它表示R中属性组X上值为x的诸元组在Z上分量的集合。

学生-课程数据库

S 学生表

学号 Sno	姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
2000101	张明	男	19	CS
2000102	李华	女	20	IS
2000103	王强	男	18	MA
2000104	秦永	男	19	CS

C 课程表

课程号 Cno	课程名 Cname	学分 Ccredit
1	数据库	3
2	数学	4
3	信息系统	3
4	操作系统	3

SC 学生选课表

学号课 Sno	程号 Cno	成绩 Grade
200101	1	92
200101	2	85
200101	3	88
200102	2	90
200102	3	80

选择(Selection)

- **定义1.2.2 选择** 选择又称为限制(Restriction),它是在关系R中选择满足给定条件的诸元组,记作:

$$\sigma F(R)=\{t \mid t\in R \wedge F(t)=\text{‘真’}\}$$

其中F表示选择条件,它是一个逻辑表达式,取逻辑值‘真’或‘假’。逻辑表达式F由逻辑运算符¬,∧,∨连接各算术表达式组成。算术表达式的基本形式为:X1 θ Y1,其中 θ 表示比较运算符,它可以是>、≥、<、≤、=或≠;X1,Y1等是属性名,或为常量,或为简单函数;属性名也可以用它的序号来代替。

选择(Selection)

- 选择运算实际上是从关系R中选取使逻辑表达式F为真的元组。这是从行的角度进行的运算，是对行的水平分解。

例1 查询信息系(IS)全体学生

$\sigma_{Sdept=”IS”}(S)$

或 $\sigma_{5=”IS”}(S)$

其中下角标“5”为Sdept的属性序号。

或 S where Sdept=”IS”

例2 查询年龄在19到20岁之间的学生

$\sigma_{Sage\geq 19 \wedge Sage\leq 20}(S)$

或 $\sigma_{4\geq 19 \wedge 4\leq 20}(S)$

或 S where Sage>=19 and Sage<=20

投影(Projection)

- **定义1.2.3 投影** 关系R上的投影是从R中选择出若干属性列组成新的关系，是对关系的垂直分解。记作：

$$\pi A(R)=\{t[A]\mid t\in R\}$$

其中A为R中的属性列集合。

- 投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组，因为取消了某些属性列后，就可能出现重复行，应取消这些完全相同的行。

投影(Projection)

例3 查询学生的姓名和所在系。

π Sname, Sdept(S)

或 π 2，5(S)

或 S[Sname, Sdept]

例4 查询学生关系Student中都有哪些系。

π Sdept(S)

或 S[Sdept]

连接(Join)

- **定义1.2.4 连接** 连接也称为θ连接。它是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。记作：

$$R \infty S=\{trts \mid tr \in R \wedge ts \in S \wedge tr[A]\ \theta\ ts[B]\}$$
$$A\ \theta\ B$$

其中A和B分别为R和S上度数相等且可比的属性组。θ是比较运算符。连接运算的结果是从R和S的广义笛卡尔积R×S中选取R关系在A属性组上的值与S关系在B属性组上值满足比较关系θ的元组。

连接(Join)

- 连接运算中有两种最为重要也最为常用的连接，一种是等值连接(equijoin)，另一种是自然连接(Naturaljoin)。另外还有外连接、左连接、右连接等。
- θ 为“=”的连接运算称为等值连接。它是从关系R与S的广义笛卡尔积中选取A，B属性值相等的那些元组.
- 自然连接(Natural join)是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉。

外连接

- **定义1.2.5 外连接** 表R(A1,A2,…,An,B1,B2,…,Bk)和S(B1,B2,…,Bk,C1,C2,…,Cm)的外连接R∞oS,行t属于表R∞oS，如果下列情况之一发生：

  1）可连接的行u,v分别在R和S中，有u[Bi]=v[Bi]（0<=i<=k）成立，此时t[A]=u[A],t[B]=u[B],t[C]=v[C]。

  2）表R中的一个行u使得S中没有一个可以与之连接的行，此时，t[A]=u[A],t[B]=u[B],t[C]=null。

  3）表S中的一个行v使得R中没有一个可以与之连接的行，此时，t[A]=null,t[B]=v[B],t[C]=v[C]。

外连接

- 由定义知，外连接保留了未匹配的行。也就是说，在外连接一端的表上的行，即使在另一端上的表没有与之相匹配的连接列值也会出现在外连接的结果中。左连接和右连接运算只是因为我们需要在某一边上保留未匹配的行而已。左连接保留了在操作符左边的未匹配行，右连接保留了在操作符右边的未匹配行。
- 一般的连接操作是从行的角度进行运算。但自然连接还需要取消重复列，所以是同时从行和列的角度进行运算。

关系代数连接运算符

符号	含义	键盘格式	示例
∞Ai θ Bj	θ 连接	JOIN(Ai θ Bj)	R∞A1>B2 S
∞o	外连接	OUTER JOIN	R ∞o S
∞Lo	左连接	LOUTER JOIN	R ∞Lo S
∞Ro	右连接	ROUTER JOIN	R ∞Ro S

连接运算举例

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

θ 连接R ∞C<E S

A	R.B	C	S.B	E
a1	b1	5	b2	7
a1	b1	5	b3	10
a1	b2	6	b2	7
a1	b2	6	b3	10
a2	b3	8	b3	10

等值连接R ∞R.B=S.B S

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

自然连接R∞S

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

连接运算举例

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

外连接R ∞o S

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2
a2	b4	12	null
null	b5	null	2

左连接R ∞Lo S

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2
a2	b4	12	null

右连接R ∞Ro S

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2
null	b5	null	2

除(Division)

- **定义1.2.6 除** 给定关系R(X，Y)和S(Y，Z)，其中X，Y，Z为属性组。R 中的Y与S中的Y可以有不同的属性名，但必须出自相同的域集。R与S的除运算得到一个新的关系P(X)，P是R中满足下列条件的元组在X属性列上的投影：元组在X上分量值x的象集Yx包含S在Y上投影的集合。记作：

$$R \div S = \{tr[X] \mid tr \in R \wedge \pi_y(S) \in Yx\}$$

其中Yx为x在R中的象集，x=tr[X]。

除(Division)

- 除操作是同时从行和列角度进行运算。
- 除运算是乘运算的逆运算，给定两个表T和S，如果表R是通过R=T×S定义的，那么有T=R÷S成立。在这个意义上可解释为什么称作除运算。

除运算举例

R

A	B	C
a1	b1	c2
a2	b3	c7
a3	b4	c6
a1	b2	c3
a4	b6	c6
a2	b2	c3
a1	b2	c1

S

B	C	D
b1	c2	d1
b2	c1	d1
b2	c3	d2

R÷S

A
a1

除运算举例

- 在关系R中，A可以取四个值{a1,a2,a3,a4}。其中：

  a1的象集为{(b1，c2)，(b2，c3，)，(b2，c1)}

  a2的象集为{(b3，c7)，(b2，c3)}

  a3的象集为{(b4，c6)}

  a4的象集为{(b6，c6)}

  S在(B，C)上的投影为{(b1，c2)，(b2，c1)，(b2，c3)}

显然a1的象集包含了S在(B，C)属性组上的投影，所以R÷S={a1}.

除运算举例

R

A	B	C
a1	b1	c1
a2	b1	c1
a1	b2	c1
a1	b2	c2
a2	b1	c2
a1	b2	c3
a1	b2	c4
a1	b1	c5

1)给出S，求得 T=R÷S。
注意： T×S都在R中。

S

C
c1

T

A	B
a1	b1
a2	b1
a1	b2

2)给出S，求得 T=R÷S。
注意： T×S都在R中。

S

C
c1
c2

T

A	B
a1	b2
a2	b1

3)给出S，求得 T=R÷S。
注意： T×S都在R中。

S

C
c1
c2
c3
c4

T

A	B
a1	b2

4)给出S，求得 T=R÷S。
注意：T×S都在R中。

S

B	C
b1	c1

T

A
a1
a2

1.2.3关系代数综合例子

- 以下例子建立在下面CAP数据库上。

C顾客

cid	cname	city	discnt
c001	李广	天津	10.00
c002	王开基	北京	12.00
c003	安利德	北京	8.00
c004	曹士雄	天津	8.00
c006	曹士雄	广州	0.00

P商品

pid	pname	city	quantity	price
p01	梳子	天津	111400	0.50
p02	刷子	成都	203000	0.50
p03	刀片	西安	150600	1.00
p04	钢笔	西安	125300	1.00
p05	铅笔	天津	221400	1.00
p06	文件夹	天津	123100	2.00
p07	盒子	成都	100500	1.00

A代理

aid	aname	city	percent
a01	史可然	北京	6
a02	韩利利	上海	6
a03	杜不朗	成都	7
a04	甘瑞	北京	6
a05	敖斯群	武汉	5
a06	史可然	天津	5

O订单

ordno	month	cid	aid	pid	qty	dollars
1011	01	c001	a01	p01	1000	450.00
1012	01	c001	a01	p01	1000	450.00
1019	02	c001	a02	p02	400	180.00
1017	02	c001	a06	p03	600	540.00
1018	02	c001	a03	p04	600	540.00
1023	03	c001	a04	p05	500	450.00
1022	03	c001	a05	p06	400	720.00
1025	04	c001	a05	p07	800	720.00
1013	01	c002	a03	p03	1000	880.00
1026	05	c002	a05	p03	800	704.00

1.2.3关系代数综合例子

例1 查询所有定购了至少一个价值为0.50的商品的顾客的名字。

$$\pi_{cname}((\pi_{pid}(\sigma_{price=0.50}(P))\infty O)\infty C)$$

例2 找出全部没有在代理商a03处订购商品的顾客cid值。

$$\pi_{cid}(O)-\pi_{cid}(\sigma_{aid='a03'}(O))$$

或 $\pi_{cid}(C)-\pi_{cid}(\sigma_{aid='a03'}(O))$

例3 找出只在代理商a03处订购商品的顾客。

$$\pi_{cid}(O)-\pi_{cid}(\sigma_{aid《》'a03'}(O))$$

1.2.3关系代数综合例子

例4 找出没有被任何一个在北京的顾客通过在上海的代理商订购的所有商品。

在北京的顾客通过在上海的代理商订购的商品：

$$\pi_{pid}((\pi_{cid}(\sigma_{city='北京'}(C)) \infty O) \infty \sigma_{city='上海'}(A))$$

从所有商品中剔除以上商品：

$$\pi_{pid}(P) - \pi_{pid}((\pi_{cid}(\sigma_{city='北京'}(C)) \infty O) \infty \sigma_{city='上海'}(A))$$

例5 找出订购了所有价格为0.50的商品的顾客名字。

$$\pi_{cname}((\pi_{cid,pid}(O) \div \pi_{pid}(\sigma_{price=0.50}(P))) \infty C)$$

例6 找出订购所有被任何人订购过一次的商品的顾客cid。

$$\pi_{cid,pid}(O) \div \pi_{pid}(O)$$

例7 找出所有接到至少顾客c004订购的商品集合的订单的代理商的aid。

$$\pi_{aid,pid}(O) \div \pi_{pid}(\sigma_{cid='c004'}(O))$$

1.2.3关系代数综合例子

例8 找出订购了p01和p07这两种商品的顾客的cid。

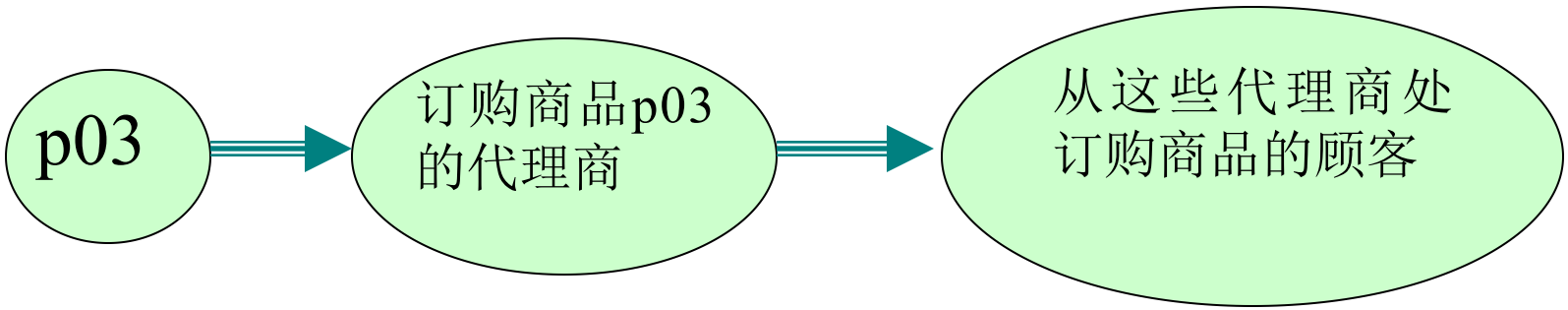
错误 π_{cid}（$\sigma_{pid='p01' \text{ and } pid='p07'}$（O））

错误 π_{cid}（$\sigma_{pid='p01' \text{ or } pid='p07'}$（O））

正确 π_{cid}（$\sigma_{pid='p01'}$（O））$\cap$ π_{cid}（$\sigma_{pid='p07'}$（O））

例9 找出从至少一个接受订购商品p03订单的代理商处订购过商品的顾客cid。

解题思路：



π_{cid}（π_{aid}（$\sigma_{pid='p03'}$（O））∞ O）

1.2.3关系代数综合例子

例10 找出所有具有和在北京和天津的顾客相同的折扣率的顾客的cid。

本题可理解为：找出那些具有和在北京和天津的顾客相同的折扣率的顾客的cid。

在北京和天津的顾客的所有折扣率：

$$\pi_{discnt}(\sigma_{city='北京' \text{ or } city='天津'}(C))$$

所以有：$\pi_{cid}(\pi_{discnt}(\sigma_{city='北京' \text{ or } city='天津'}(C))) \infty C)$

1.2.3关系代数综合例子

例11 列出通过以下代理商订购的商品的pid：代理商从以下的顾客处接受订单，而这些顾客从一个接受过顾客c001订单的代理商处订购了至少一个商品。

接受顾客c001订单的代理商：

$$\pi_{aid}(\sigma_{cid='c001'}(0))$$

从这些代理商处订购过商品的顾客：

$$\pi_{cid}(\pi_{aid}(\sigma_{cid='c001'}(0)) \infty 0)$$

从这些顾客处接受过订单的代理商：

$$\pi_{aid}(\pi_{cid}(\pi_{aid}(\sigma_{cid='c001'}(0)) \infty 0) \infty 0)$$

通过上述代理商订购的商品pid：

$$\pi_{pid}(\pi_{aid}(\pi_{cid}(\pi_{aid}(\sigma_{cid='c001'}(0)) \infty 0) \infty 0) \infty 0)$$

作业：

1. 如果关系R和S没有共同的列属性,根据定义说明表R×S等于表R∞S。

2. 对CAP数据库，用关系代数完成下列查询。

   1）找出顾客、代理商和商品都在同一个城市的三元组（cid,aid,pid）。

   2）找出顾客、代理商和商品两两不在同一个城市的三元组（cid,aid,pid）。

   3）列出所有在同一个城市代理商的aid对。

   4）找出折扣率最大和最小的顾客cid。

   5）取出销售过所有曾被顾客c002订购过的商品的代理商的　　名字。

   6）找出只从一家代理商处订购过商品的顾客cid。

1.3 SQL

- 自从20世纪80年代以来，SQL就是一个通用的、功能极强的关系数据库语言。现在，SQL语言正从关系形式（ANSI SQL-92标准）转向对象-关系形式（ANSI SQL-99标准，1999年颁布）。
- SQL语言是1974年由Boyce和Chamberlin提出。1986年10月美国国家标准局(ANSl)批准了SQL作为关系数据库语言的美国标准。同年公布了SQL标准文本(简称SQL-86)。1987年6月国际标准化组织(ISO)也采纳了此标准。1989年，美国国家标准局（ANSI）采纳了新的规范SQL-89标准，取代SQL-86，同时SQL-89标准也被国际标准化组织(ISO)采纳。1992年，ANSI/ISO颁布了SQL2版本，标准的名称为SQL-92。SQL-92分称几个顺序级别：从代表SQL-89最小扩展集的“Entry”到“Intermediate”和“Full”。完成于己于1999年的SQL-99具有更加高级的特征（包括对象-关系特性），亦称SQL3。

1.3 SQL

- 主要的几个数据库生产厂商并不可能完全遵守SQL-99（以及更老的SQL-92）。我们一般更加关注在产品中已经实现的SQL-99特征。

1.3.1 SQL特点
1.3.2数据定义
1.3.2.1 SQL模式的创建和删除
1.3.2.2 SQL提供的基本数据类型
1.3.2.3 定义、删除与修改基本表
1.3.2.4 建立与删除索引
1.3.2.5 视图的创建和删除
1.3.3 数据查询
1.3.3.1 简单查询
1.3.3.2 子查询
1.3.3.3 UNION运算和FOR ALL条件
1.3.3.4 高级SQL语法
1.3.3.5 集合函数
1.3.3.6 行分组
1.3.4 数据更新
1.3.4.1 插入数据
1.3.4.2 修改数据
1.3.4.3 删除数据

1.3.1 SQL特点

- 1)综合统一
- 2)高度非过程化
- 3)面向集合的操作方式
- 4) 以同一种语法结构提供两种使用方式
- 5)语言简捷，易学易用

和关系代数相比，就查询能力而言，SQL并没有根本的改进，在关系代数查询方面的经验可以成为用SQL来实现查询的良好借鉴。在构造查询时SQL的select语句比关系代数要灵活.

1.3.2数据定义

- SQL的数据定义功能包括对模式(Schema)、表(关系，Table)、视图(View)和索引(Index)的创建、删除和修改操作。如下表所示。

操作对象	操　作　方　式		
	创　建	删　除	修　改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

1.3.2.1 SQL模式的创建和删除

- 在SQL-99中，模式是表、索引及其他数据库对象的集合。模式名通常是一个用户名。在Core SQL-99和当前大多数产品中，当用户的数据库帐户建立时，其模式在用户名之后给出，他们不能再建立其他模式。SQL-99的扩展特性允许用户建立附加模式，一个SQL模式由模式名和模式拥有者的用户名或账号来确定。
- SQL模式的创建可用CREATE语句实现，其句法如下：

  CREATE SCHEMA 〈模式名〉 AUTHORIZATION 〈用户名〉

  目前只有DB2 UDB允许用户建立附加模式。其它产品中模式用用户名替代。

1.3.2.1 SQL模式的创建和删除

- 在SQL中还有一个“目录”概念。目录是SQL环境中所有模式的集合。包含数据库中定义的对象的信息的表，由系统维护。ORACLE叫数据字典，DB2 UDB叫目录表，INFORMIX叫系统目录。目录表在建立数据库时建立，用户不能更新，但DBA可以用select获取这些信息。
- 当一个SQL模式及其所属的基本表、视图等元素都不需要时，可以用DROP语句撤消这个SQL模式。DROP语句的句法如下：

  DROP SCHEMA <模式名> [CASCADE|RESTRICT]

1.3.2.2 SQL提供的基本数据类型

SQL-99	ORACLE	INFORMIX	DB2 UDB	说明	C
char(n)	char(n) n<=4000	char(n) n<=32767	char(n) n<=254	定长字符型	char array[n+1]
varchar(n)	varchar(n) varchar2(n)	varchar(n)	varchar(n)	变长字符型	char array[n+1]
numeric(p,d) decimal(p,d)	numeric(p,d) decimal(p,d) number(p,d)	numeric(p,d) decimal(p,d)	numeric(p,d) decimal(p,d)	定点数，由p位数字(不包括符号、小数点) 小数点后面有d位数字	无
smallint	smallint	smallint	smallint	短整数	short int
integer	integer	integer	integer	长整数	int,long int
real	real	real	real	浮点数	float
double precision, float, float(n)	double precision, number,float float(n)	double precision, float	double precision, double,float, float(n)	取决于机器精度的双精度浮点数 至少为n位精度	double

1.3.2.3 定义、删除与修改基本表

- 建立数据库最重要的一步就是定义一些基本表。SQL语言使用CREATE TABLE语句定义基本表，一般格式如下：

  CREATE TABLE <表名>
  (<列名><数据类型>[列级完整性约束条件]
  [，<列名><数据类型>[列级完整性约束条件]]...
  [，<表级完整性约束条件>]);

- 例:CREATE TABLE Student
  (Sno CHAR(5) PRIMARY KEY,
  Sname CHAR(8) NOT NULL,
  Sage SMALLINT CHECK(Sage BETWEEN 17 AND 22),
  Ssex CHAR(2) CHECK(Ssex IN(’男’,’女’)),
  Sdept CHAR(20) DEFAULT(‘软件学院’));

  创建学生表:Sno为主键(非空唯一),Sname非空,Sage在17到20之间取值,Ssex只能取‘男’或‘女’,Sdept默认值为‘软件学院’.

1.3.2.3 定义、删除与修改基本表

- 实际使用时要有用户ID和密码,进入交互式环境，才能完成数据库操作。
- 修改基本表

  ALTER TABLE<表名>

  [ADD<新列名><数据类型>[完整性约束]]

  [DROP<完整性约束名>]

  [MODIFY<列名><数据类型>];

  其中<表名>是要修改的基本表，ADD子句用于增加新列和新的完整性约束条件，DROP子句用于删除指定的完整性约束条件，MODIPY子句用于修改原有的列定义，包括修改列名和数据类型。

1.3.2.3 定义、删除与修改基本表

- 删除基本表

  DROP TABLE〈表名〉

  基本表一旦删除，表中的数据、此表上建立的索引和视图都将自动被删除掉。因此执行删除基本表的操作一定要格外小心。

- 注意：有的系统，如Oracle，删除基本表后建立在此表上的视图定义仍然保留在数据字典中。但是，当用户引用时就报错。

1.3.2.4建立与删除索引

- 索引的功能表现在以下3方面。
  (1)使用索引可以明显地加快数据查询的速度
  (2)使用索引可保证数据的唯一性
  (3)使用索引可以加快连接速度
- 建立索引的原则
  (1)索引的建立和维护由DBA和DBMS完成
  (2)大表应当建索引，小表则不必建索引
  (3)对于一个基本表，不要建立过多的索引
  (4)根据查询要求建索引

1.3.2.4建立与删除索引

- CREATE [UNIQUE] [CLUSTER] INDEX <索引名> ON <表名>

  (<列名>[<asc|desc>[,<列名>[<asc|desc >]]);

- 如果数据增加删改频繁，系统会花费许多时间来维护索引。这时，可以删除一些不必要的索引。

  DROP INDEX <索引名>;

1.3.2.5视图的创建和删除

- 视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制。视图一经定义，就可以和基本表一样被查询、被删除，我们也可以在一个视图之上再定义新的视图，但对视图的更新(增加、删除、修改)操作则有一定的限制。
- CREAT VIEW <视图名>[(列名>[,<列名>]...)]
  AS <子查询> [WITH CHECK OPTION];
  其中子查询可以是任意复杂的SELECT语句，但通常不允许含有ORDER BY子句和DISTINCT短语。WITH CHECK OPTION表示对视图进行UPDATE，INSERT和DELETE操作时要保证更新、插入或删除的行满足视图定义中的谓词条件(即子查询中的条件表达式)。

1.3.2.5视图的创建和删除

- DROP VIEW <视图名>;

  视图删除后视图的定义将从数据字典中删除。但是由该视图导出的其他视图定义仍在数据字典中，不过该视图已失效。用户使用时会出错，要用DROP VIEW语句将它们一一删除。

- DBMS执行CREATE VIEW语句的结果只把视图的定义存入数据字典，并不执行其中的SELECT。在关系数据库中，并不是所有的视图都是可更新的,因为有些视图的更新不能的有意义的转换成对基本表的更新。行列子集视图是可更新的。各个DBMS对视图的更新有自己的规定。

1.3.3 数据查询

- 数据查询是数据库的核心操作。SQL语言的数据查询只有一条SELECT语句:

  SELECT[ALL|DISTINCT]{*|<目标列表达式[[as]别名]>[,<目标列表达式[[as] 别名]>]...}

  FROM<表名或视图名>[,<表名或视图名>]...

  [WHERE<条件表达式>]

  [GROUP BY<列名>[,<列名>][HAVING<条件表达式>]]]

  [ORDER BY<列名>[ASC|DESC][,<列名>[ASC|DESC]] ...];

1.3.3 数据查询

- Select语句的执行过程可以理解为：

  首先，对FROM子句中的所有表做关系乘积

  接着，删除不满足WHERE子句的行

  根据GROUP BY子句对剩余的行进行分组

  然后删除不满足HAVING子句的组

  求出SELECT子句选择列表的表达式的值

  若有关键词DISTINCT存在，则删除重复的行

- Select中的标识符：一般的SQL标识符是大小写无关的。实际上SQL在解释以前会把它们转化为大写形式。一个标识符必须以一个字母打头，Entry SQL-92和Core SQL-99将一个标识符字节数限制在18个以内。

1.3.3 数据查询

- 表达式可以是数值表达式、字符串表达式和日期表达式等。

  数值表达式由常数、表属性、算术运算符、算术函数所组成。

  字符串表达式由常数、表属性、字符串运算符、字符串函数所组成。

  日期表达式由常数、表属性、日期运算符、日期函数所组成

- WHERE中使用谓词来表示条件。一般情况谓词运算结果为TRUE或FALSE,但如果遇到空值时,可能为UNKNOWN。
- SQL查询的一个争议点即：对于同一个查询会存在众多不同的构造方法。

1.3.3 ORACLE、INFORMIX、DB2 UDB 中的一些数学函数

名称	描述	结果
abs(n) mod(n,b) sqrt(n)	n的绝对值，n为数值型 n被b除后得到的余数。n,b为整数 n的平方根，n为整数或浮点数 另外，三角函数、指数函数、对数函数、幂函数和round(n)	数值类型 整数 浮点

1.3.3一些标准的和特定产品的串处理函数

SQL-99中的描述	ORACLE	DB2 UDB	INFORMIX
返回串长度(整数个字符) CHAR_LENGTH(str)	length(str)	length(str)	length(str), char_length(str)
返回子串,从m个开始取n个 SUBSTRING（str） FORM m FOR(n)	substr(str,m[,n])	substr(str,m[,n])	substr(str,m[,n]), substring(str from m for n)
返回去掉左或右空格后得到的包含空格的串 TRIM（[[LEADING|TRAILING|BOTH] [SET]FROM]str）	trim([[leading|trailing|both] [set] from] str), ltrim(str[,set]), rtrim(str[,set])	ltrim(str); rtrim(str)	trim([[leading|trailing|both] [set] from]str);
返回子串str2在str1中位置，如果指定n,则从n开始 POSITION（str1 IN str2）	instr(str1,str2[,n])	posstr(str1,str2[,n])	
字母小写 LOWER（str）	lower(str)	lcase(str)	lower(str)
字母大写 UPPER（str）	upper(str)	ucase(str)	upper(str)

1.3.3 SQL中的标准谓词

谓词	形式	例子
比较谓词	expr1 θ {expr2|(subquery)}	p.price>(subquery)
BETWEEN谓词	expr1[NOT] BETWEEN expr2 and expr3	c.discnt between 10.0 and 12.0
量化谓词	expr θ [SOME|ANY|ALL] (subquery)	c.discnt>=all (subquery)
IN谓词	expr [NOT] IN (subquery)	pid in (select pid from orders)
EXISTS谓词	[NOT]EXISTS(subquery)	exist(select * …)
IS NULL谓词	colname IS [NOT]NULL	c.discnt is null
LIKE谓词	cloname [NOT] LIKE val [ESCAPE val]	cname like ‘A%’

1.3.3.1 简单查询

- 例：检索定货记录中所有pid值：

```
select pid from orders  --结果中有重复的pid
select distinct pid from orders  --结果中pid唯一
```

select缺省为all,distinct没有出现时允许重复行，缺省情况不遵守行唯一性规则。

- 例：在orders表上生成每笔业务的利润profit(收入减去60%的成本、顾客的折扣以及代理商的酬金)：

```
select ordno,x.cid,x.aid,x.pid,.40*(x.qty*p.price)-
       .01*(c.discnt+a.percent)*(x.qty*p.price) as profit
from orders as x,customer as c,agents as a,products as p
where c.cid=x.cid and a.aid=x.aid and p.pid=x.pid;
```

投影 π

选择 σ

乘积×

1.3.3.1 简单查询

注：1）SQL-99规定FROM子句中执行连接运算，但大多数产品通过笛卡尔积运算并且在WHERE子句中包含表示参与连接的列值相等的条件来模拟连接运算，具体实现方法(执行计划，查询优化)各不相同。

2）FROM中AS被省略，SQL仍能识别表别名。ORACLE和INFORMIX使用*别名*或*表别名*，DB2 UDB使用*相关名*。

3）列表达式的列名可以通过AS指明，ORACLE中称*列别名*，INFORMIX中称*显示标签*，DB2 UDB中简单称为*列名*。如果没有AS，ORACLE中将完整表达式文本作为列名。

1.3.3.2 子查询

- 每个Select查询都会生成一张表，但我们不能像关系代数表达式那样任意将一个Select语句嵌入另一个Select语句。这是SQL与关系代数的一个很重要的不同点。例如：from子句中不能出现select(SQL-99标准已去掉该限制，但数据库产品中并未完全实现)，where子句中显然可以出现select。
- 出现在另一个select语句之内的select语句形式称为子查询。一个子查询能以许多种方式出现在另一个select语句的WHERE子句条件中。

1.3.3.2 子查询

- 1）IN谓词(NOT IN)

例:求通过住在北京或上海的代理商订货的顾客的姓名和折扣.

```
select cname,discnt from customers where cid in
(select cid from orders where aid in
(select aid from agents where city in (‘北京’,’上海’)));
```

例:求由住在北京的顾客和住在北京的代理商组成的所有订货ordno.

```
select ordno from orders where (cid,aid) in
(select cid,aid from customers c,agents a
 where c.city=‘北京’ and a.city=‘北京’);
```

SQL-99允许,有些系统不允许.

以上两例为不相关查询:内层子查询独立于外层的select.

例:找出订购了产品p05的顾客的名字.

```
select distinct cname from customers where ‘p05’ in
(select pid from orders where cid=customers.cid)
```

本例为相关查询:子查询使用外层select语句提供的数据.

1.3.3.2 子查询

- 2)量化比较谓词：expr θ [SOME|ANY|ALL] (subquery)

  θ 为比较运算符<,<=,=,<>,>,>=。SOME和ANY含义相同，SOME是最新版本的推崇形式。

例：找出佣金百分率最小的代理商aid。

```
select aid from agents where percent <=all
  (select percent from agents);
```

例：求出满足以下条件的顾客cid:该顾客的discnt小于任一住在北京的顾客的discnt.

错误：
```
select cid from customer where discnt <any
    (select discnt from customers where city= '北京' );
```

正确：
```
select cid from customer where discnt <all
    (select discnt from customers where city= '北京' );
```

注意any不是任意

1.3.3.2 子查询

- 3)EXISTS谓词:EXISTS (Subquery)为真当且仅当子查询返回一个非空集合;NOT EXISTS(Subquery)为真当且仅当返回集合为空.

交运算

例:求出既订购了产品p01有订购了产品p07的顾客cid.

关系代数:$\pi_{cid}(\sigma_{pid='p01'}(O)) \cap \pi_{cid}(\sigma_{pid='p07'}(O))$

```
select distinct cid from orders x where pid='p01' and exists
    (select * from orders where cid=x.cid and pid='p07');
或 select distinct x.cid from orders x,orders y
    where x.pid='p01' and x.cid=y.cid and y.cid='p07';
```

EXISTS的查询一般能找到等价的其他查询形式。

使用和不使用exists

1.3.3.2 子查询

- NOT EXISTS确实为我们带来了一些新的功能。not exists能被用来实现关系代数的MINUS运算。如果R和S是两个兼容表(属性相同A1…An),R-S用SQL计算:

差运算

```
select A1…An from R
  where not exists (select * from S
  where  S.A1=R.A1 and …… and S.An=R.An);
```

例:找出没有通过代理商a03订货的顾客cid.

关系代数： $\pi_{cid}(0) - \pi_{cid}(\sigma_{aid='a03'}(0))$

```
select distinct cid from orders x where not exists
  (select * from orders where cid=x.cid  and   aid='a03');
```

1.3.3.3 UNION运算和FOR ALL条件

∪运算

- 为了提供关系代数的∪运算，SQL使用UNION：

  subquery UNION [ALL] subquery

例：包含了顾客所在的或代理商所在或两者皆在的城市名单。

```
select city from customers union
    select city from agents;
```

行不重复

或

```
select city from customers union all
    select city from agents;
```

行重复

1.3.3.3 UNION运算和FOR ALL条件

÷运算

- SQL中没有等价的÷运算。如果面临的查询“**要求被检索的对象集合必须符合‘所有’这类关键词的条件**”(FOR ALL)时，关系代数要用到除运算。SQL中可以：1.表述要检索的候选对象的一个反例(至少一个对象不符合条件)，.并建立select语句(选出所有反例)；2.建立表示这类反例不存在的条件(not exists)；3.建立最终select。

例：求通过住在北京的所有代理商订了货的顾客cid.

1.反例：住在北京但没有为所求顾客c.cid订货的代理商：

```
select * from agents where a.city='北京' and not exists
  (select * from orders x where x.cid=c.cid and x.aid=a.aid)
```

2.反例不存在：not exists (反例)

3.最终： select c.cid from customers where not exists

```
(select * from agents where a.city='北京' and not exists
  (select * from orders x where x.cid=c.cid and x.aid=a.aid));
```

1.3.3.3 UNION运算和FOR ALL条件

÷运算

例：找出订购了所有被顾客c006订购的商品的顾客的cid.

反例：被c006订购但没有被c.cid订购的商品：

被c006订购

```
select  p.pid  from  products p
where  p.pid in (select pid from orders x where x.cid='c006')  and
   not exists  (select * from orders y where y.pid=p.pid and y.cid=c.cid)
```

没有被c.cid订购

反例不存在： not exists

```
(select  p.pid  from  products p
where  p.pid in (select pid from orders x where x.cid='c006')  and
   not exists (select * from orders y where y.pid=p.pid and y.cid=c.cid))
```

最终： select cid from customers where

```
                not exists  (select  p.pid  from  products p
where  p.pid in (select pid from orders x where x.cid='c006')  and
   not exists (select * from orders y where y.pid=p.pid and y.cid=c.cid)) ;
```

1.3.3.4 高级SQL语法

- 以下介绍的高级SQL运算符不是Entry SQL-92的部分,但几乎都属于SQL-99,目前产品中不一定支持，但可能出现在未来的数据库产品中。
- 1)INTERSECT(∩)和EXCEPT(—)运算符

ALL考虑重复行及数目

Subquery[UNION [ALL]|INTERSECT [ALL]|EXCEPT [ALL] Subquery]

注：Core SQL-99只有EXCEPT而没有EXCEPT ALL。

两个子查询从左到右的列类型是兼容的,即可以并、交、差。如类型char(5)和类型char(10)的列运算,结果为char(10).

ORACLE提供UNION、UNION ALL、INTERSECT、MINUS(EXCEPT)。但不支持INTERSECT ALL或MINUS ALL。DB2 UDB都支持。

1.3.3.4 高级SQL语法

例: (Q UNION ALL Q UNION ALL Q)
INTERSECT ALL (Q UNION ALL Q)

结果是(Q UNION ALL Q),包含两个重复行。

(Q UNION ALL Q UNION ALL Q)
INTERSECT (Q UNION ALL Q)

结果是(Q),没有包含重复行。

(Q UNION ALL Q UNION ALL Q)
EXCEPT ALL (Q UNION ALL Q)

结果是(Q),没有包含重复行。

1.3.3.4 高级SQL语法

- 2)连接形式

可以是视图名

ORACLE不能有AS

通用形式：FROM tablename [[AS] corr_name [,……]……]

SQL-99：FROM后面还可以是：

一般形式：

允许为表的列重新命名

tablename[[AS] corr_name[(colname[,colname……])]]

子查询：

允许是子查询,此时别名必须

(subquery) [AS] corr_name[(colname[,colname……])]

显式连接：

内连,左连,右连,外连

table1 [INNER|{LEFT|RIGHT|FULL}[OUTER]] JOIN table2

ON condition

连接条件

显然SQL-99中允许子查询出现在FROM子句中。以上三种形式在产品中并没有完全实现。使用时可以试用或帮助。

1.3.3.4 高级SQL语法

例：检索至少订购了一件价格低于0.5的商品的顾客姓名。

```
select distinct cname from
  (orders o join products p on o.pid=p.pid)
   join customers c on o.cid=c.cid
where p.price<0.5;
```

注：ORACLE仅提供左连和右连，而且语法与标准SQL也不同：

```
SELECT ... FROM T1,T2
WHERE [T1.c1[(+)]=T2.c2|T1.c1=T2.c2[(+)]] AND condition
```

T2保留所有行

与T1连不上的用NULL

1.3.3.5 SQL中的集合函数

- SQL中称集合函数(set function),ORACLE称组函数(group function),DB2 UDB称列函数(column function),INFORMIX称聚集函数(aggregate function).

名称	参数类型	结果类型	描述
COUNT	任意(*)	数值	出现次数
SUM	数值	数值	参数和
AVG	数值	数值	参数均值
MAX	字符,数值	字符,数值	最大值
MIN	字符,数值	字符,数值	最小值

- 集合函数语法:SET_FUNCTION ([ALL|DISTINCT] col)|COUNT(*)

注:ALL时包括重复行，DISTINCT不包括重复行。

WHERE子句比较操作中不能使用集合函数.如discnt<max(...)

集合函数忽略了所有空值.

集合函数不允许嵌套使用.如AVG(select MAX(dollars).....)

1.3.3.6 SQL中行的分组

- SQL报表功能:根据某些列值的共性把一个表所包含的全部行分成若干个子集,然后对每个子集执行集合函数.

  例:打印每个代理商为顾客c002和c003订购产品及产品总数量.

```
select a.aid,aname,p.pid,pname,sum(qty)
 from orders x,products p,agents a
 where x.pid=p.pid and x.aid=a.aid
                and x.cid in (‘c002’,’ c003’)
 group by a.aid,a.aname,p.pid,p.pname;
```

- GROUP BY对象的列上的空值会被分在同一组里.

1.3.3.6 SQL中行的分组

- 如果要去掉分组后的某些行,不能用where,只能用HAVING子句.

  例:求被至少两个顾客订购的产品pid.

```
select pid from orders
group by pid having count(distinct cid) >=2;
```

- 如果没有GROUP BY只有HAVING,则整个结果为一组.
- 基本SQL select语句的通用形式不允许集合函数的嵌套.但可以有变通的GROUP BY形式.

  例:求所有代理商的最大销售额的平均值.

1.3.3.6 SQL中行的分组

- 例:求所有代理商的最大销售额的平均值.

  错: select avg(select max(dollars) from orders
  group by aid);

  对: SQL高级形式,ORACLE中可行:

```
select avg(t.x) from (select aid,max(dollars) as x
  from orders group by aid) t;
```

  常用作法:

```
creat view t as (select aid,max(dollars) as x
            from orders group by aid);
select avg(t.x) from t;
```

1.3.4 SQL数据更新

- SQL中数据更新包括插入数据(Insert)、修改数据(Update)和删除数据(Delete)三条语句.
- **1.3.4.1 插入数据**

  INSERT INTO <表名>[(<属性列1>[,<属性列2>...)]

  [VALUES(expr1|NULL[,expr2|NULL]...)|Subquery]

  将新元组插入指定表中,其中新记录属性列1的值为expr1，属性列2的值为expr2...。INTO子句中没有出现的属性列，新记录在这些列上将取空值。还可通过子查询批量插入数据。
- 在表定义时说明了NOT NULL的属性列不能取空值,否则会出错。

1.3.4 SQL数据更新

- 如果INTO子句中没有指明任何列名，则新插入的记录必须在每个属性列上均有值，给定值对应于表定义的字段顺序。
- 子查询不需要用括号括起来。

```
insert into my_c select * from customers where city=‘北京’;
insert into my_c (select * from customers where city=‘北京’);
```

与customers表结构一样

错误

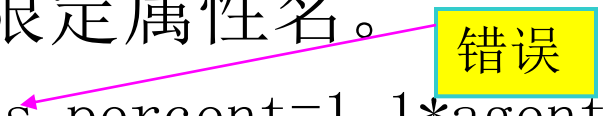
- 1.3.4.1 修改数据

```
UPDATE  <表名>
 SET <列名1>=<expr1|NULL|(Subquery)>
    [,<列名2>=<expr2|NULL|(Subquery)>...]
 [WHERE <条件>];
```

1.3.4 SQL数据更新

- 其功能是修改指定表中满足WHERE子句条件的元组。其中SET子句给出expr的值用于取代相应的属性列值。如果省略WHERE子句，则表示要修改表中的所有元组。
- 只有一个表可以作为UPDATE的对象。有些系统中不允许在SET子句中使用限定属性名。

  update agents set agents.percent=1.1*agents.percent



- Entry SQL-92中SET子句中不能用子查询，但SQL-99及ORACLE、DB2 UDB等都支持。

  用customers表中最新discnt更新my_c中discnt

  update my_c set discnt=(select discnt from customers where cid=my_c.cid)

1.3.4 SQL数据更新

- 删除语句的一般格式为：

  DELETE FROM〈表名〉 [WHERE <条件>]

  从指定表中删除满足WHERE子句条件的所有元组。如果省略WHERE子句，表示删除表中全部元组，但表的定义仍在字典中。DELETE语句删除的是表中的数据，而不是关于表的定义。

- 删除条件中可以有子查询

  删除总订货金额小于600的代理商：

```
delete from agents where aid in (select aid from
  orders group by aid having sum(dollars)<600);
```

- 增删改操作只能对一个表操作。因此在执行增删改操作时，要注意数据库中数据的一致性。

作业：

1. 检索佣金百分率大于最小百分率的代理商aid.
2. 求通过住在北京或上海的代理商订货的顾客cid.(使用子查询和不使用子查询两种)
3. R、S和T具有相同的属性列A1.....An,不用高级SQL来实现关系表达式(R UNION S) MINUS T.
4. 求出没有为任何住在北京的顾客订购任何在天津生产的产品的代理商的aid.
5. 没有一个包含GROUP BY子句的查询会返回重复行,这一命题为真吗?如果为真,解释原因;否则,给出反例.
6. 说出你所用到DBMS和其中的SQL语句,他们与讲课时的不一样或者不能用.

1.4 对象关系数据库

- 传统关系型仅支持简单数据类型；不支持数组、嵌套和递归的数据结构，给新型数据模型-面向对象的数据模型提供了应用背景。
- 面向对象的程序设计思想和技术促进了面向对象的数据模型的实现和应用-OODBS(面向对象数据库)。
- 习惯于‘填表’方式的客户不太愿意接受对象模型，而且面向对象数据库系统与编程语言集成度高,没有统一可行标准； OODBS只占很小的市场分额。
- 以关系数据库和SQL为基础扩展关系模型-对象关系数据库(ORDBMS)的发展满足了许多应用需求。

1.4 对象关系数据库

- ORACLE、DB2 UDB、INFORMIX都推出ORBMS，ANSI/ISO 发布的SQL-99标准是一个对象关系标准。

1.4.1 面向对象数据模型
- 1.4.1.1对象与标识
- 1.4.1.2类和类层次
- 1.4.1.3继承和嵌套
- 1.4.1.4面向对象数据库

1.4.2 ORACLE对象关系库
- 1.4.2.1ORACLE中对象类型
- 1.4.2.2ORACLE中汇集类型
- 1.4.2.3PL/SQL过程、用户定义函数和方法

1.4.3 对象关系数据库
- 1.4.3.1对象联系
- 1.4.3.2ORDB数据定义
- 1.4.3.3ORDB数据查询
- 1.4.3.4三种产品的对象关系特征

1.4.1 面向对象数据模型

- 面向对象数据库系统(Object Oriented Database System, OODBS)是数据库技术与面向对象程序设计方法相结合的产物；面向对象程序设计方法的主要活动在建立对象和对象之间的联系；面向对象数据库系统一般依赖于一个面向对象的程序设计语言.一个面向对象数据库系统是一个持久的、可共享的对象库的存储和管理者；而一个对象库是由一个00模型所定义的对象的集合体。

面向对象程序设计语言中建立的对象自动保存在磁盘上即OODBS

- 面向对象数据模型是用面向对象观点来描述现实世界实体的逻辑组织、对象间限制、联系.

面向对象数据库的用户主要是应用软件系统软件开发专业程序员

1.4.1 面向对象数据模型

- 面向对象数据模型的主要观点:

对象：对象是基本的数据结构,现实世界的任一实体都被统一模型化为一个对象，每个对象有一个永久标识；对象可以被指定为一个给定类型,还可以定义为其它对象的子类型;

封装：每一个对象是其状态与行为的封装。封装的状态为属性，行为为方法。对象与外部通信一般只能通过显式的消息传递。

继承：子类型继承父类型的所有特性和行为.

面向对象的核心观点构成00模型的基础，没有像关系模型那样的规范说明。
如果与面向对象数据库相比,RDBMS可以称为面向表的数据库。

1.4.1.1对象与标识

- 对象是由一组数据结构和在这组数据结构上的操作的程序代码封装起来的基本单位，包括：属性、方法和消息。
- 面向对象的数据库系统在逻辑上和物理上从面向记录上升为面向对象、面向可具有复杂结构的一个逻辑整体。
- 面向对象数据库中的每一个对象都有一个唯一的不变的标识(OID)。创建对象时系统就给它赋予一个OID，直到它被删除。
- 外部与对象的通信只能通过消息，隔离了对象的实现与对象的应用，提高数据独立性。
- 查询属性值通过调用方法,不象关系系统那样使用SQL,不够方便灵活。

ORDBMS中不支持封装,可即席查询

1.4.1.2 类和类层次

- 相似对象的集合称为类,每个对象称为它所在类的一个实例,一个类中的所有对象共享一个定义,它们的区别仅在与属性的取值不同.类本身也可以看作一个对象(类对象).

对相似对象的重复定义十分浪费,将相似对象分组形成一个“类”

- 面向对象数据模型提供类层次结构,一个面向对象数据库模式可能有多个类层次(超类和子类),在一个类层次中,一个类继承其所有(直接和间接的)超类的全部属性、方法和消息.超类是子类的抽象,子类是超类的特殊化.

不同类的对象可以共享他们公共部分的结构和特性

1.4.1.2 类和类层次

- 类层次可以动态扩展，一个新的子类能从一个或多个已有类导出。
- 面向对象数据库中,类是“型”,对象是某一类的一个“值”.类属性的定义域可以是任何类(基本类或包含属性和方法的一般类)。
- 面向对象数据库模式是类的集合，类可能有多个类层次。

例如：一个学校应用的面向对象数据库，其中有教职员工和学生两个类，这两类都有身份证号、姓名、性别、住址等属性，也有相同的方法和消息。统一定义教职员工和学生的公共属性、方法和消息部分，称为一个类“人”；分别定义各自的特殊属性、方法和消息部分，分别称为教职员工类和学生类，教职员工类和学生类定义为人的子类。

1.4.1.3 继承和嵌套

- 子类继承超类的特性,避免许多重复定义.这种继承性有两优点:是建模的有力工具;提供信息重用机制.封装和继承导出多态性.
- 子类可以定义自己特殊的属性、方法和消息,在定义这些特殊的属性、方法和消息时可能与超类或多个超类之间发生冲突.这种冲突由系统解决,不同系统使用不同的冲突解决方法.
- 一个对象属性可以是另一个对象称为对象的嵌套.这样一来,不仅类之间具有层次结构,而且某一个类内部也具有嵌套层次结构,形成对象横向和纵向的复杂结构.

不像RDBMS中的平面结构

1.4.1.4 面向对象数据库

- 在OODB中，与对象模型密切相关的是面向对象数据库语言。OODB语言主要包括对象定义语言（ODL）和对象操纵语言（OML），对象操纵语言一个重要子集是对象查询语言（OQL）。

RDBMS中有数据定义DDL、数据操纵DML、数据控制DCL

- OODB语言一般应具备类的定义与操纵、操作/方法的定义、对象的操纵功能。
- 面向对象数据库语言的研制是OODB系统开发中的重要部分，人们试图扩充面向对象程序设计语言OOPL的查询语言或者扩充SQL的功能，目前还没有像SQL那样的关于面向对象数据库语言的标准。不同的OODBMS其具体的数据库语言各不相同。

1.4.2 ORACLE对象关系库(对象关系模型)

- 对象关系模型支持用户类型定义(ORACLE称对象类型;INFORMIX称行类型;DB2 UDB称用户定义类型).一个表是包含用户定义类型的多个行.
- 对象关系模型允许一个行包括一个行值汇集(如数组或单个列自身是一个表).

显然,对象关系模型违反第一泛式规则

- 面向对象数据库中,只能通过对象的方法操纵对象,在对象关系模型中,所有对象被看作是公共的而并非私有的,通过SQL即可操纵对象,当然也可使用用户定义的函数.

显然,对象关系模型没有面向对象数据模型和关系模型那样严格

1.4.2 ORACLE对象关系库(一个例子)

家属属性有内部数据结构,对象类型

职工号	姓名	职务	家属	
			关系	姓名
0001	张家口	职员	妻子	李小玫
			儿子	张冠李
0002	柴米油	经理	妻子	陆续琴
0003	郝成功	职员	妻子	朱那亚
			儿子	郝乃军
			女儿	郝佳欣

每个行中包含多个值,汇集类型

1.4.2.1 ORACLE中对象类型

- 一个对象类型有多个类型属性,ORACLE中对象类型由Create type语句创建.

例:create type namesex_t as object
(name char(8),sex char(2));/

create table employees (eid char(4),
ename namesex_t,job char(6));

insert into employees value('0001',
namesex_t('赵子龙','男'),'经理');

select e.eid,e.ename.name,e.ename.sex
from employees e
where e.job='业务员';

SQL*plus中创建对象必须有/

typename(属性值)称为对象构造器

前面'表名.'不能省略

1.4.2.1 ORACLE中对象类型

- 创建对象的一般格式:

CREATE TYPE typename AS OBJECT

(属性名 类型 [,属性名 类型…]);

可以是其他用户已经定义的类型,由此形成对象的嵌套和依赖

(informix中用create row type rowname(),行类型)

- 若ORACLE中表的行包含对象类型,则称为对象表(informix中称类型表).采用如下形式定义表:

CREATE TABLE tablename OF typename

[([属性名 NOT NULL][,属性名 NOT NULL…]

[,PRIMARY KEY (属性名[,属性名…])])];

typename对象中属性即成为表中属性

- **DROP TYPE** typename [FORCE];

  DROP TABLE tablename;

1.4.2.1 ORACLE中对象类型

- 例:create type employee_t as object

(eid char(4),ename namesex_t,job char(6));

create table employees of employee_t

(primary key(eid));

- 对象表(包含行对象employee_t和列对象ename)

无名顶层列	employees			
顶层属性	eid	ename		Job
ename属性		ename. name	ename.sex	
行1	0001	赵子龙	男	经理
行2	0002	张翌德	男	职员
行3	0003	孟可飞	女	职员

1.4.2.1 ORACLE中对象类型

- 可用嵌套的点号访问一个对象的属性，无名顶层列可以用VALUE()形式引用行对象.

select * from employees e where e.job='经理';

eid	ename(name,sex)	job
0001	namesex_t('赵子龙','男')	经理

select value(e) from employees e where e.job='经理';

行对象

value(e)	(eid	ename(name,sex)	job)
employee_t	(0001	namesex_t('赵子龙','男')	经理)

列对象

select e.eid,e.job from employees e

where e.ename=namesex_t('孟可飞','女');

对象构造器

1.4.2.1 ORACLE中对象类型

- INSERT和UPDATE同样可以使用对象构造器.

```
insert into employees
  values('0004',namesex_t('胡作为','男'),null);
update employees e set e=employee_t('0004',
  namesex_t('胡有为','男'),'职员') where eid='0004';
```

- ORACLE为每个行对象提供一个唯一标识(对象标识符).一个表的列可被定义为REF(引用)的内部数据类型,允许它‘指向’一个对象表的行对象.对象的REF与对象自身具有不同类型.

1.4.2.1 ORACLE中对象类型

想着外码,理解REF

- 例:

```
create type c_t as object (...);
create type a_t as object (...);
create type p_t as object (...);
create type o_t as object
 (ordno int,month char(2),
  cid char(3),aid char(3),
  pid char(3),qty int,
  dollars double precision,
  oc ref c_t,
  oa ref a_t,
  op ref p_t);
```

```
create table c of c_t
      (primary key (cid));
create table a of a_t
      (primary key (aid));
create table p of p_t
      (primary key (pid));
create table o of o_t
     (primary key (ordno),
      scope for (oc) is c,
      scope for (oa) is a,
      scope for (od) is p);
```

1.4.2.1 ORACLE中对象类型

- 一般通过REF访问比通过连接访问的效率高.

超过200元的订单号和顾客名:

select x.ordno,x.oc.cname from o x where x.dollars>200;

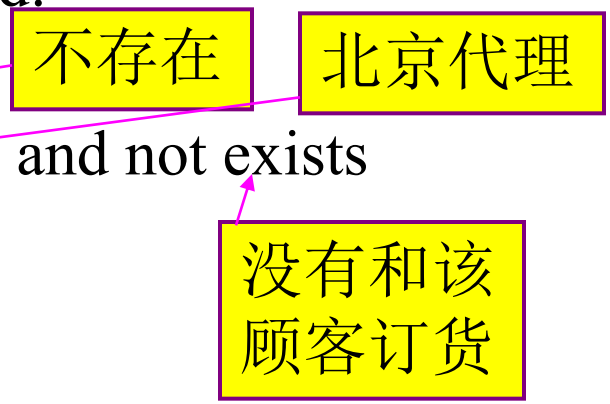
- **获得引用(REF)对象值的函数REF().**

没有通过代理商a05 订货的顾客名:

select x.cname from c x where not exists (select *
from o y where y.oc=ref(x) and y.aid='a05');

通过北京的所有代理商订货的顾客cid:

select x.cid from c x where not exists
(select * from a y where y.city='北京' and not exists
(select * from o z where
z.oc=ref(x) and z.oa=ref(y)));



1.4.2.1 ORACLE中对象类型

- 一个对象类型不能嵌套包含与自己类型相同的成分(递归),但可以包含对同样类型的引用.

  建一个配偶对象表:

  ```
  create type mate_t as object
    (mid char(4),mname namesex_t, partner  ref mate_t);
  create table mates of mate_t
    (primary key (mid), scope for (partner) is mates);
  ```

  检索其配偶为女性的人名:

  ```
  select mname.name from mates m where
  m.partner.mname.sex='女';
  ```

- 一个表集合可以具有一个用ref表示的复杂关系集.如雇员有部门,部门有经理,经理又是雇员.回路的依赖中删除时用**DROP TYPE … FORCE**.

1.4.2.2 ORACLE汇集类型

- ORACLE有两种汇集类型:表类型和数组类型.汇集类型是数据库成熟的数据类型,经过适当的转换,可以在查询中解释为表.
- 表类型和嵌套表创建:

1. 4. 2. 2 ORACLE汇集类型

- 例:

```
create type depe_t as table of namesex_t;
create table employees
   (eid char(4),ename namesex_t,dependents depe_t,
    primary key (eid) )
    nested table dependents store as d_tab;
```

eid	ename	Dependents
0001	namesex_t(‘张家口’,‘男’)	namesex_t(‘李小玫’,‘女’)
		namesex_t(‘张冠李’,‘男’)
0002	namesex_t(‘柴米油’,‘男’)	namesex_t(‘陆续琴’,‘女’)
0003	namesex_t(‘郝成功’,‘男’)	namesex_t(‘朱那亚’,‘女’)
		namesex_t(‘郝乃军’,‘男’)
		namesex_t(‘郝佳欣’,‘女’)

1.4.2.2 ORACLE汇集类型

- 子表的数据只能通过父表来访问.

检索0001号职工的家属:

对象列必须加限定非对象列不加

```
select dependents from employees where eid='0001';
```

检索有2个家属以上的职工eid:

```
select eid from employees e where 2<(select count(*) from
  table(e.dependents));
```

Table()将e.dependents转换为一表

检索0003号职工的家属数目:

错误:

```
select count(*) from (select e.dependents from
  employees e where e.eid='0003');
```

错误:

```
select count(*) from (select table(e.dependents) from
  employees e where e.eid='0003');
```

正确:

```
select count(*) from table (select e.dependents from
  employees e where e.eid='0003');
```

1.4.2.2 ORACLE汇集类型

- 利用表的乘积可以消除表的嵌套.

显示职工号和其家属名:

```
select e.eid,d.name from employees e,
    table(e.dependents) d;
```

没有家属的eid不显示

```
select e.eid,d.name from employees e,
    table(e.dependents) (+) d;
```

(+)表示外连，没有家属的eid也显示

显示具有最大家属名的职工号和其家属名:

```
select e.eid,d1.name from employees e,
    table(e.dependents) d1 where d1.name=
      (select max(d2.name)
          from table(e.dependents) d2);
```

1. 4. 2. 2 ORACLE汇集类型

- ORACLE的另一种汇集类型即**数组类型,以VARRAY声明.**

```
create type phone_t as varray(4) of int;
create table employees
 (eid char(4),ename namesex_t,
   telephone phone_t,  primary key (eid)) ;
```

eid	ename	telephone
0001	namesex_t(‘张家口’,‘男’)	phone_t(12345678,23456789)
0002	namesex_t(‘柴米油’,‘男’)	phone_t(34567890)
0003	namesex_t(‘郝成功’,‘男’)	phone_t(45678901,56789012, 67890123)

1.4.2.2 ORACLE汇集类型

- SQL语句不能访问数组的下标. 在嵌入主语言后, 从一行检索的数组可付给编程语言的数组.

  TABLE()可以应用于VARRAY.

  例:列出电话号码为45678901的人:

  select eid,ename from employees e where

  45678901 in (select * from table(e.telephone));

- 汇集类型(表类型和数组)的更新用汇集构造器.

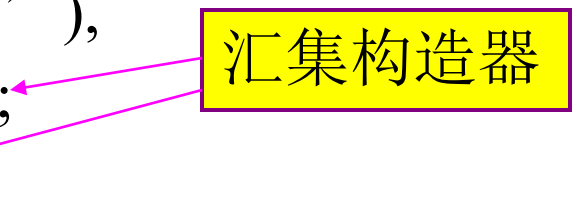
  Insert into employees values

  (‘0006’,namesex_t(‘孙子兵’,‘男’),

  phone_t(68888888,96666666));

  Update employees set

  telephone=phone_t(80166666,80166667) where eid=‘0001’;



1.4.2.3 PL/SQL、用户定义函数和方法

- 非过程的SQL不具备计算的完整性。ORACLE引入PL/SQL,INFORMIX引入SPL,SQLSERVER引入T-SQL.过程SQL支持内存驻留变量、条件和循环结构、过程和函数等。
- 一个用过程SQL编写的函数在服务器上执行时被称为存储过程，由Create Function语句记录在服务器上的数据库目录表中。
- 触发器是可以在任意事件发生时执行的过程性SQL语句块。触发器用于实现定制的约束或缺省行为。
- 用户定义函数(UDF)是用过程性SQL(或C、Java)写的函数，可以像交互式SQL中的内部函数一样被调用。
- 方法是与用户定义类型一起定义的用户定义函数。

1.4.2.3 PL/SQL、用户定义函数和方法

- 例:对象关系数据库中利用对象类型定义表、利用用户定义函数定义对象的方法。
- 1)定义对象点(point_t)和矩形(rectangle_t)及其方法inside(point_t)(点是否在矩形对象内)和area()(矩形面积).

定义点对象

```
create type point_t as object (x int,y int);
```

定义包含方法的矩形对象

```
create type rectangle_t as object (pt1 point_t, pt2 point_t,
    member function inside(p point_t) return int,
    member function area return int);
```

成员函数，实际是对象的方法

1.4.2.3 PL/SQL、用户定义函数和方法

```
create type body rectangle_t as
  member function area return int is  begin  --成员函数area定义
    return (self.pt2.x-self.pt1.x)* (self.pt2.y-self.pt1.y);
  end;                                --成员函数area没有参数,返回长乘宽.
  member function inside(p in point_t) return int is  begin
    if (p.x>=self.pt1.x) and (p.x<=self.pt2.x) and
       (p.y>=self.pt1.y) and (p.y<=self.pt2.y) then return 1;
    else return 0;
    end if;
  end; --成员函数inside参数为点类型的p,当p在矩形中时返回1,
  否则返回0.
end;
```

1.4.2.3 PL/SQL、用户定义函数和方法

- 2) 创建point_t和rectangle_t类型的对象表.

```
create table points of point_t (primary key (x,y));
create table rects of rectangle_t (primary key
 (pt1.x,pt1.y,pt2.x,pt2.y));
```

插入两个矩形：

```
insert into rects values (point_t(1,2),point_t(3,4));
insert into rects values (point_t(1,1),point_t(6,6));
```

插入三个点：

```
insert into points values (2,3);
insert into points values (1,4);
insert into points values (4,4);
```

1.4.2.3 PL/SQL、用户定义函数和方法

- 3) 使用方法area和inside.

计算rects表中矩形面积：

```
select value(x),x.area() from rects x;
```

判断点(4,2)是否在每个矩形中：

```
select value(x),x.inside(point_t(4,2)) from rects x;
```

查询在所有矩形内的所有点：

```
select value(p) from points p where not exists
  (select * from rects r where  r.inside(p)=0);
```

计算覆盖各点的最小矩形的面积：

```
select p.x,p.y,min(r.area()) from rects r,points p
   where  r.inside(value(p))>0 group by p.x,p.y;
```

将面积小于24的矩形的pt2横纵坐标加1:

```
update rects r set pt2=point_t(r.pt2.x+1,r.pt2.y+1) where r.area()<24;
```

1.4.2.3 PL/SQL、用户定义函数和方法

- 定义包含方法的对象

```
CREATE TYPE typename AS OBJECT
    (attrname datatype [,attrname datatype…]
  MEMBER FUNCTION methodname
      [(param  type [,param type…])]
      RETURN datatype,
  [,MEMBER FUNCTION methodname
      [(param  type [,param type…])]
      RETURN datatype,…]);
```

在前面对象类型的基础上增加了MEMBER FUNCTION语句，定义对象的成员函数，即方法。

1.4.2.3 PL/SQL、用户定义函数和方法

- 对象成员函数的定义

```
CREATE [OR REPLACE] TYPE BODY typename [AS|IS]
 MEMBER FUNCTION methodname [(param type...)]
    RETURN type IS
    BEGIN      --PL/SQL语句开始
       statements
    END;       --PL/SQL语句结束
 [MEMBER FUNCTION methodname [(param type...)]
    RETURN type is
    BEGIN.....]
END;
```

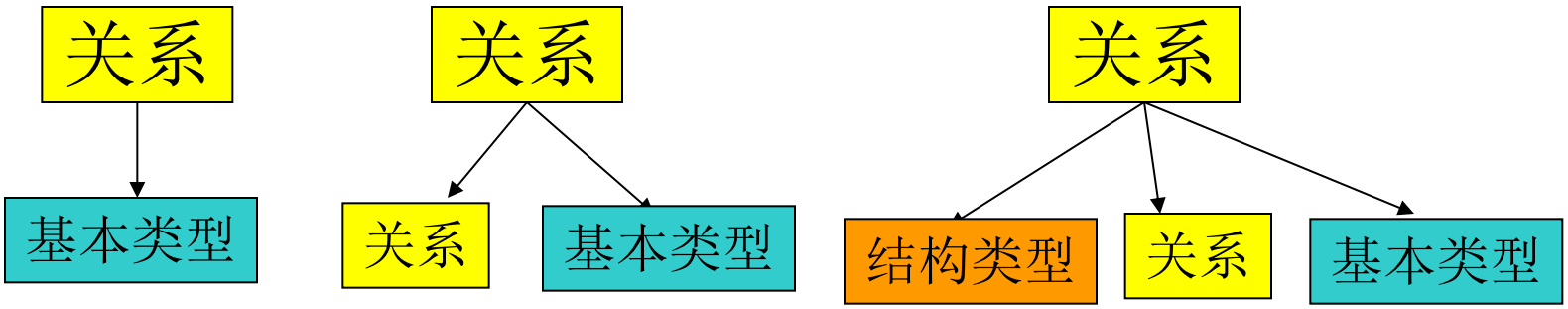
对于不同的DBMS，对象成员函数的处理过程使用的语句可能不同，带来移植的通用问题。

1.4.3 对象关系数据库

- 对象关系数据库系统除具有原来关系数据库的各种特点外，还应该提供以下特点：
- 1）扩充数据类型：允许用户根据自己应用需求定义自己的数据类型、函数和操作符。例如可以定义数组、向量、矩阵、集合以及这些数据类型上的操作。
- 2）支持复杂对象：复杂对象指由多种基本类型或用户定义类型构成的对象。
- 3）支持继承：支持子类、超类的概念，支持属性数据和函数及过程的继承。
- 4）提供通用的规则系统：规则在DBMS及其应用中十分重要，传统DBMS中的触发器可以看作规则的一种形式。

1.4.3.1 对象联系

- 关系模型中基本的数据结构层次是关系-元组-属性, 属性类型是基本类型.
- 嵌套关系模型是平面关系模型的扩展, 允许关系的属性又是一个关系.
- 嵌套关系模型进一步扩展, 出现复合对象模型. 关系中的属性可以是基本类型, 也可以是结构类型或集合类型.



关系模型　　嵌套关系模型　　复合对象模型

1.4.3.1 对象联系

- 嵌套关系和复合对象类型定义时不能允许递归,否则可能造成无穷嵌套,语义混乱.采用‘引用’技术(Reference)可解决类型定义中的递归问题.在属性类型中有基本类型、结构类型、关系类型和引用类型.引用类型相当于程序设计中指针概念.
- 数据的泛化和细化引出对象类型的层次.较高层对象类型为超类,较低层对象类型为子类,子类型继承超类型的特征.而子类型又有本身的其他特征.

1.4.3.2 对象关系数据定义

- 对象关系模型中,属性可以是复合类型:
  1)结构类型:不同类型元素的有序集合;
  2)数组类型:同类元素的有序集合;
  3)集合类型:同类元素的无序集合.

  数据类型可以嵌套.
- 继承性表现在类型的继承和表的继承.子类型继承超类型的特征;子表继承父表的特征.
- 数据类型可以嵌套定义,但要实现递归,就要采用‘引用’类型.

1.4.3.3 对象关系数据查询

- 当属性值为单值或结构值时, 属性的引用方式在层次间加圆点‘.’
- 聚集函数(max、min、count等)可以应用于任何以关系为值的表达式中,以关系作为参数,返回单个值.
- 当属性值为一个集合时,应当将嵌套的集合转换为常规的表的形式(解除嵌套).
- 在数据更新时,充分注意数据定义的形式,利用‘对象构造器’.也可以利用将常规关系转换为嵌套集合形式的函数完成(解除嵌套的逆过程).

1.4.3.4 三种产品的对象关系特征

	ORACLE	INFORMIX	DB2 UDB
用户定义类型(UDT)	有，结构化的‘对象类型’	有，结构化的‘行类型’	有，结构化的UDT
继承	无	有	有
UDT引用	有	无	有
集合类型	有，嵌套表、VARRAY	有，集合，多集合，列表	无
用户定义函数(UDF)	有，PL/SQL,C,Java	有，SPL，C，Java	有,C,Java
封装的UDT	BLOB(Binary Large OBject)对象	BLOB对象	BLOB对象
打包UDT	Cartridge	DataBlad	Extender
BLOB尺寸	4G-1	4T	2G-1
BLOB子串	UDF访问	UDF访问	UDF访问

作业：

1. 对象关系数据库特点.
2. 行对象与列对象有何不同？
3. Value(),Ref(),Table()的作用？
4. student_t类型中有namesex_t类型以及学号id、e-mail和入学年份(整数),其中namesex_t不空.
   定义student_t类型和表student,学号id.
5. 对对象关系数据库CAP,查找姓李的代理商aid和aname，这些代理商没有销售过北京的商品.
6. 检索在三个以上矩形中出现的点,显示点和覆盖这些点的矩形数.

2. 数据存储与数据管理

- 前面重点讲数据库的逻辑模式,本章介绍数据库物理模式设计中的数据存储技术和保证数据库正常运行的安全性、完整性控制和数据库恢复技术。

  2.1数据存储

  2.2数据管理

2.1数据存储

- 数据存储技术的重要目标就是尽可能减少读写数据所需的磁盘访问(I/O操作)次数，尽可能使数据驻留在内存中。

  均衡负载,提高效率！

  2.1.1数据的磁盘存储

  2.1.2索引

  2.1.3聚簇索引与非聚簇索引

  2.1.4散列(HASH)簇存储

2.1.1 数据的磁盘存储

2.1.1.1磁盘访问是面向页面(数据块)的

2.1.1.2ORACLE的磁盘资源分配

2.1.1.1磁盘访问是面向页面(数据块)的

- 基本表中的行和索引是存储在磁盘上的。磁盘由若干盘片组成，盘片有磁道、扇区，若干盘片的磁道组成柱面。
- 一次磁盘页面访问包括：

  寻道时间：磁盘臂移动到指定柱面的时间；

  旋转延时：磁盘旋转到指定扇区的时间；

  传输时间：读写磁盘页面数据的时间。

  磁盘访问时间主要是移动磁盘臂到指定位置所需时间。
- 如果两个要连续读取的数据块在磁盘上紧挨着，则寻到时间很短，如果两数据在同一柱面上，那末寻到时间为零。

2.1.1.1磁盘访问是面向页面(数据块)的

- 在读写磁盘的一个页面的时间里，可以执行百万条的程序指令与内存交换数据。相对于内存而言，磁盘访问速度是很慢的，我们要尽量减少磁盘访问的次数。
- 磁盘访问基本都是“面向页面的”，磁盘页面也称数据块。磁盘页面的页面地址可以是连续的整数，也可以由设备号、柱面号、磁盘表面号和开始扇区地址组成。
- ORACLE一个页面(块)为2KB,DB2 UDB标准页面为4KB（DB2 UDB还支持8KB、16KB和32KB）。

2.1.1.1磁盘访问是面向页面(数据块)的

- 数据库系统按照给定的磁盘页面(块)地址读取磁盘页面，把数据放到内存的缓冲区(缓冲区是在数据库系统初始化时候建立的)中。每读入一个页面都在散列后备表中记录该页面在缓冲区中位置，每一次读取页面时，首先在散列后备表中查询该页面是否已在缓冲区中，如果在缓冲区则忽略磁盘访问。
- 缓冲区采用最少使用算法(LRU)管理可用空间，当缓冲区需要自由空间时,最少使用的页面将被移出.最频繁使用的数据被保存在缓冲区中。
- 为了提高效率,扩大内存的同时,有必要对访问进行组织(表的磁盘空间分配),以使所需信息都在同一个页面上。

2.1.1.2 ORACLE的磁盘资源分配

- 不同商业数据库系统的磁盘空间分配体系结构不大相同.以下说明ORACLE中的磁盘资源分配。
- CREATE TABLESPACE tsname
  DATAFILE dfname1[,dfname2…]
  [DEFAULT STORAGE storage]
  [ONLINE|OFFLINE];

联机|脱机(立刻能不能用)

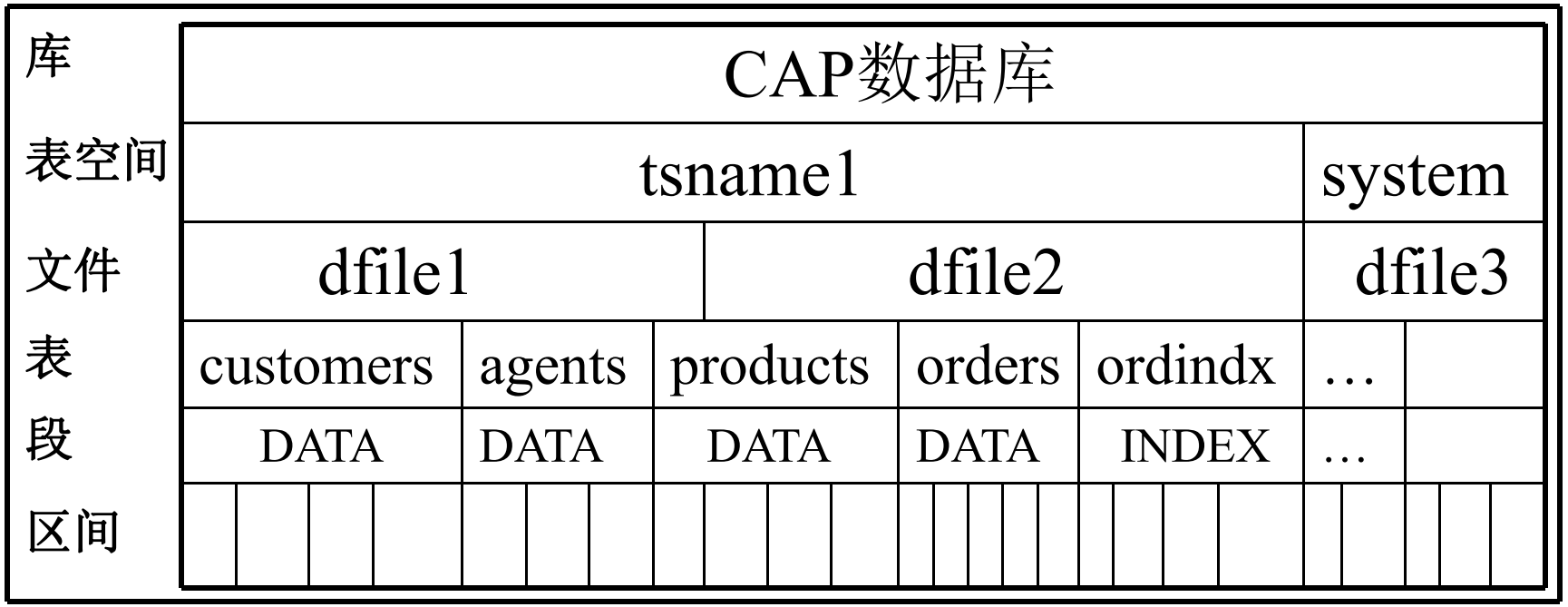
- 表空间是ORACLE数据库基本的分配介质,所有请求磁盘空间的表、索引和其它对象都在表空间中有对应的磁盘空间。表空间对应于一个或多个操作系统文件，可跨越磁盘设备。
- 所有数据库产品都有类似于表空间的结构来隔离用户和操作系统，它代表一块可以使用的磁盘空间。DB2称为表空间，INFORMIX称为数据库空间。

2.1.1.2 ORACLE的磁盘资源分配

- ORACLE数据库应包含多个表空间.SYSTEM表空间是在Create Database时自动创建的,SYSTEM表空间包含数据字典,也可包含用户对象.但DBA应创建几个表空间分别存储相应的对象.
- 当CREATE TABLE时可使用子句TABLESPACE指定表空间.
- 创建表和索引时,其表空间分配是以数据段对象和索引段对象标识的.
- 创建数据段和索引段时,将从表空间中分配一个初始的磁盘空间,称为初始区域(缺省10KB).当写满该区域后,再分配一块区域,称为下一区域.

2.1.1.2 ORACLE的磁盘资源分配

数据库存储结构示意图



- 每一块区域都在一个文件上, 一个段可以由来自多个文件的区域组成. 每一块区域大小都是块大小的整数倍。

2.1.1.2 ORACLE的磁盘资源分配

- 表空间由一个或多个在DATAFILE子句中指定的操作系统文件组成.

```
DATAFILE dfname [SIZE n [K|M]][REUSE]
   [AUTOEXTEND OFF|
    AUTOEXTEND ON [NEXT n [K|M]
    [MAXSIZE UNLINITED|n [K|M]]]]
   [,dfname …]
```

没有SIZE且数据文件存在,ORACLE使用该文件;

有SIZE,ORACLE创建一新文件(以前有则覆盖);

有SIZE且有REUSE,ORACLE重用旧文件.

AUTOEXTEND子句决定可否自动扩展文件大小;

NEXT定每次申请区域大小,MAXSIZE定最大尺寸.

2.1.1.2 ORACLE的磁盘资源分配

- DEFAULT STORAGE指定该表空间上的段的缺省磁盘分配区域方案，所包含段可以不使用该方案而自行确定.

```
DEFAULT STORAGE([INITIAL [K|M]][NEXT n [K|M]]
  [MINEXTENTS n][MAXEXTENTS [n|UNLIMITED]]
  [PCTINCREASE n])
```

INITIAL初始区域(缺省为10KB);NEXT下一区域大小(缺省为10KB);PCTINCREASE后续区域比前一区域大的百分比,0为不增长,缺省为50.

MINEXTENTS,MAXEXTENTS最大最小区域数.

- 区域大小是一个数据块中字节数的整数倍(最小为2048B,最大为4095MB).

2.1.1.2 ORACLE的磁盘资源分配

- 创建了表并分配了区域,记录总是一个接一个地插在第一个区域的第一个块里,直到第一个区域用完,系统再分配一个新的区域.
- 每一个行是由包含了列值的连续字节组成,在块中的布局如下图:

块头包含块地址段类型等,行目录有行在块中位置,可用空间留在中间.

- 行可用块地址及行目录唯一标识,此为行指针,。ORACLE中称ROWID,DB2中称RID.

2.1.1.2 ORACLE的磁盘资源分配

- ORACLE中ROWID有两种形式(限制的/扩展的):
  BBBBBBBB.RRRR.FFFF(块号.行号.文件号)
  OOOOOOFFFBBBBBBRRR(段号文件号块号行号)
- ROWID可以查看表中行是怎样存储在磁盘上的;
  通过ROWID访问某一行是最快的方法.
- DB2中行是不能超过一个页面的(4KB页面最大行长度4005字节),ORACLE中行允许跨越块存放.如果块中无法放下一行时,该行被切成片段.
- ORACLE中CREATE TABLE的PCTFREE子句可以减少出现行片段的概率.
- 减少片段是进行数据库重组织的原因之一.

2.1.1.2 ORACLE的磁盘资源分配

- CREATE TABLE tablename (……)
  [TABLESPACE tsname][STORAGE(……)]
  [PCTFREE n][PCTUSED n]

  STORAGE子句:针对该表不用表空间的区域分配方案,而专门由该STORAGE存储子句指定.

  PCTFREE决定每个块上可以有多少空间用于行的插入(缺省10,n为0到99):当块中有(100-n)%的空间使用后,新的插入被停止;留出的空间以备Varchar类型用或Alter table时插新列.

  PCTUSED决定当已用空间降到n%时,新的行可继续插入(缺省40,n为1到99).

2.1.1.2 ORACLE的磁盘资源分配

ORACLE磁盘分配总结：

- 表空间包含一系列数据库对象(表、索引等)，对应若干操作系统文件，多个表空间可以让不同磁盘设备起不同作用，减少I/O冲突，另外数据备份可在表空间一级进行.
- 表对应表空间中数据段，可以自行定义自己的磁盘存储方案，也可用所在表空间的存储方案.
- 段包含一系列区域，区域是块的整数倍，段的存储方案主要由初始区域、下一个区域和区域增长率最大最小区域数决定.
- 块是基本的I/O操作单元，尽可能将相临数据存放在一个块中，避免碎片的产生是好的存储模式的评判标准.

2.1.2 索引

- 数据库索引与驻留内存的数据结构有相似之处,但数据库索引包含数据量大,它是存放在磁盘上的,只有访问时才调入内存.
- 索引由一系列存储在磁盘上的索引项组成,索引项第一列是索引键(keyval),第二列是行指针(ROWID).索引项按索引键排序.当数据发生更新时,索引也更新.索引键和表的主键不是一回事.
- 数据库索引设计的重要目标是减少读数据所需的磁盘访问次数.
- B树是当今数据库中使用最广泛的索引结构,它是DB2唯一的索引结构;ORACLE的主要索引结构,ORACLE还有散列聚簇(Hash Cluster).

2.1.2 索引

2.1.2.1 B树(B+树)结构

2.1.2.2 索引节点布局和可用空间

2.1.2.3 ORACLE和DB2 UDB的Create Index

2.1.2.4 ORACLE位图索引

2.1.2.1 B树(B+树)结构

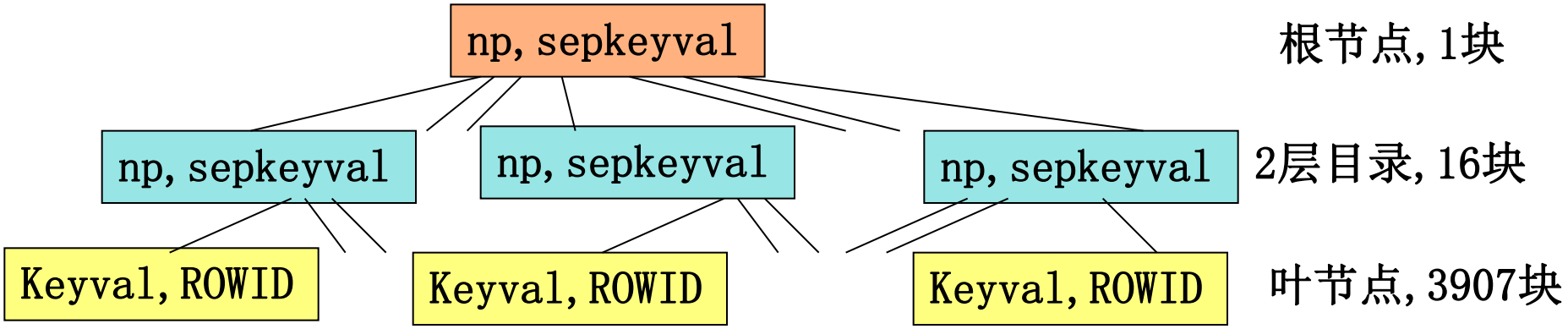
- 例:假设表中有一百万条记录要建索引,假设索引项占8个字节(ROWID占4个字节,键值占4个字节),一个2KB的块可存放2048/8=256个索引项,共需1000000/256=3907个数据块.

二分法查找: 3907个数据块顺序存放,第一次读入第3907/2=1954块数据,检查keyval,第二次读入1954块数据中的第1954/2=977块数据,检查keyval,……,第12次的I/O操作才能将包含所找键值的数据块调入内存.再根据ROWID去提取数据,再加一次I/O操作.

二分法查找不停地I/O操作.挺费劲!

2.1.2.1 B树(B+树)结构

B树结构:np为指向下层节点块的指针,sepkeyval为分隔符键值,假定目录项所需空间与叶子相同.



B树结构存储索引:一百万个索引项建起了三层的B树(叶节点的层号称为B树的深度),第一次调入根块,比较后第二次调入2层目录的某一块第3次I/O操作即可找到键值数据块.当然根据ROWID去提取数据,还得再加一次I/O操作.

B树结构存储索引使查找变得:好多啦!

2.1.2.1 B树(B+树)结构

- B树(B+树结构)是从根到叶多个扇出的树形结构,具有下列属性:

  1)每一节点和磁盘块大小一致,存放在磁盘上.

  2)目录层包含目录项(有n-1个分隔符键值和n个指向下一层B树节点的磁盘指针).

  3)叶节点包含形式为(keyval,ROWID)的项.

  4)根节点以下节点不能是满的(备插入用)但至少是半满的(删除后合并节点).

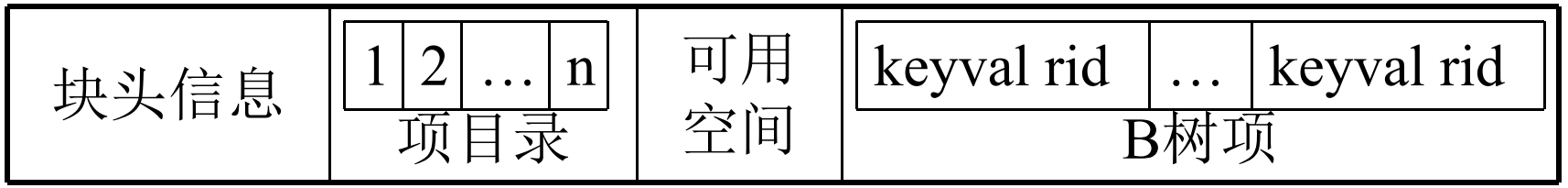
  5)根节点至少有两个项(只有一行记录时除外).

B树结构随数据的增加而增长,随数据减少而降低.这一过程由系统自动完成,但不同系统采用不同的策略.

许多数据库产品提供对B树进行重组织的实用程序,整理数据更新后的B树索引结构.

2.1.2.2 索引节点布局和可用空间

- B树节点的块结构为:



唯一键值的B树叶节点块结构

重复键值时DB2的叶节点块结构

(ORACLE有专门的BITMAP)

因为B树的生长过程中可能会发生分裂,所以其节点可以在半满到满之间变化,我们可以指定其初始可用空间的百分比.

块中块头信息也要占用空间,实际使用空间不是满块大小.

2.1.2.2 索引节点布局和可用空间

- B树的深度就近似于要访问到叶节点的I/O操作次数. B树的深度和行的数目是对数关系:

  $$K=CEIL(\log_n(M))$$

  K为B树的层数,M为行数,n为每层的扇出数,CEIL(s)函数返回大于等于s的最小整数.
- 一般经常使用的B树的上面层都是放在内存缓冲区中,以减少磁盘访问次数,提高效率.
- 建索引过程是首先取出索引项,然后按键值排序,最后顺序加到B树叶子层中.实际使用中应当先装数据,再建索引,这样才能保证数据是连续分布的.

2.1.2.3 ORACLE和DB2 UDB的Create Index

- CREATE [UNIQUE|BITMAP] INDEX indexname
  ON tablename (col[ASC|DESC][,col…])
  [TABLESPACE tsname][STORAGE(…)]
  [PCTFREE n] [NOSORT];

以上为ORACLE创建索引的语句.

PCTFREE决定索引第一次创建时B树节点块将不被填充的百分比,用于插入新行时的索引项,其缺省值为10,对以后不更新的表索引可以设为0.

NOSORT表明表中的行已经按索引键值排序并按此顺序存放在磁盘上(建索引省时间).如果有此子句但表中行没有排序将报错.

2.1.2.3 ORACLE和DB2 UDB的Create Index

- CREATE [UNIQUE] INDEX indexname
  ON tablename(col[ASC|DESC][,col…])
  [INCLUDE(col[ASC|DESC][,col…])
  [CLUSTER]
  [PCTFREE n] [MINPCTUSED n]
  [ALLOW REVERSE SCANS];

DB2中索引的表空间与表的表空间一样.

INCLUDE指定索引中存储的其他非索引列.

MINPCTUSED设置一个阈值,删除行达到该阈值后,块自动合并.

ALLOW REVERSE SCANS创建双向叶子指针(快).

2.1.2.4 ORACLE位图索引

- 如果有重复键值,ORACLE中可以使用位图索引(bit map index).
- 位图索引的思想:在块中不存放重复键的ROWID,而存放ROWID转换成的位图(表中有n条记录则使用n位的0和1序列,第k条记录在n位位图的第k位置为1).例如:5个职员,键值‘男’的位图01001说明第2个和第5个职员为男性.
- 位图索引有很好的I/O性能,压缩后的位图占空间少;位图索引在WHERE后有多个谓词时很有效.
- 当频繁的更新时,位图须先解压,再修改,使得更新费时.频繁更新时,不要使用位图索引.

想想DB2中重复键值的块结构

2.1.3 聚簇索引与非聚簇索引

- 根据索引键值把记录行存放在磁盘上称为聚簇(clustering),所引用的行和键值顺序一样的索引称为聚簇索引(clustered index).
- B树中,如果叶子节点上直接存放行的信息而不是行的ROWID,则称索引为主索引(primary index).如果叶子节点上存放行的ROWID,则称索引为辅助索引(secondary index).显然B树主索引就是聚簇索引.
- 当所查询行彼此很靠近时使用聚簇索引很有效,非聚簇索引则不能节省I/O操作.
- 一张表只能有一个聚簇索引.DBA和程序员商量决定应该如何使用聚簇索引.

2.1.3 聚簇索引与非聚簇索引

- CREATE [UNIQUE] INDEX indexname ON tab…
  [CLUSTER][PCTFREE n][MINPCTUSED n];
  DB2 UDB创建索引时用CLUSTER子句说明该索引为聚簇索引.(一张表只能有一个聚簇索引)

- CREATE TABLE tablename (…)
  [ORGANIZATION HEAP|ORGANIZATION INDEX]
  ORACLE用ORGANIZATION INDEX创建B树主索引,但B树键值必须为表的主键(primary key);ORGANIZATION HEAP为缺省值,表示行可随便放,不一定是B树主索引.

2.1.3 聚簇索引与非聚簇索引

- ORACLE中的聚簇是包含了在某些特定列上进行连接的多张表中的行的结构,进行连接的列称为聚簇键.两张表中具有相同聚簇键值的行在磁盘上是存放在一起的.
- CREATE CLUSTER clustername
  (colname datatype [,col…])
  [PCTFREE n][PCTUSED n][STORAGE(…)]
  [SIZE n [K|M]][TABLESPACE tsname]
  [INDEX|
  [SINGLE TABLE]HASHKEYS n[HASH IS exp]];

SIZE为聚簇键值对应行的空间大小(缺省为1块).

2.1.3 聚簇索引与非聚簇索引

- 聚簇创建好后, 可以在该聚簇上建表. 例:

```
create cluster deptemp (deptno int) size 2000;
create table emps(empno int primary key,
        ename varchar(10),deptno int)
        cluster deptemp(deptno);
create table depts (deptno int primary key,
        dname varchar(9)) cluster deptemp(deptno);
```

- 聚簇上还要创建索引, 这样即可按聚簇索引键访问行了. 索引聚簇使相同列只存储一次节约空间; 索引聚簇使连接运算更加快捷; 聚簇上的索引适用于其中各个表减少开销.

2.1.4 散列(HASH)簇存储

- HASH与聚簇都是把若干有共用列的表聚簇在一起,但聚簇建立专用索引,用索引确定各聚簇键对应行的存储位置;HASH不建索引,通过HASH函数生成HASH值,由HASH值确定键值对应行的存储位置,一步到位.
- CREATE CLUSTER clustername …
  (colname datatype [,col…]) SIZE n
  [SINGLE TABLE]HASHKEYS n [HASH IS expr];
  SINGLE TABLE表示只供一张表优化用.
  HASHKEYS说明该散列簇至少包含的键数(应该INITIAL的值大于SIZE*HASHKEYS,从而使初始区域放下该散列簇,避免碎片).

HASH关键字

2.1.4 散列(HASH)簇存储

- expr是计算数字值的表达式,称为HASH函数.HASH函数负责键值与空间存储位置的转换,HASH函数的优劣决定散列存储方式的成败.HASH函数有两种表达方式:

  1)缺省方式(没有HASH IS子句):系统自动转换.

  2)expr可以为:函数(列名,...).函数应保证得到唯一性较好的HASH值.

例:create cluster deptemp (deptno int) size 2000

hashkeys 10000 hash is mod(deptno,10003);

- 大型应用中,表的长度稳定,磁盘空间需求可以把握,查询一般为‘关键字=...’时,利用散列簇优势十分明显.

作业：

1. ORACLE中执行：

   create table customers(cid…)

   storage (initial 20480,next 20480, maxextents 8,minextents3,pctincrease 0);

   该文件第一次创建时，会分配多少字节的磁盘空间？

   该表最大可以容纳多大的空间？

2. 职工表emp中有200000行，每行长度100字节.

   create table emp (eid int not null…) pctfree 25;

   设块中可使用2000个字节，估算emp中行所需的块的个数.

   create unique index on emp (eid) pctfree 20;

   ROWID占6个字节，eid占4个字节，每个键的列附加一个字节.估算B树索引中每一块的索引项个数，叶子层块数？假设目录层目录项(sepkeyval,np)与叶子层中的项相等，估算每一目录层的块数?eid的值从1到200000连续，每秒可进行80块磁盘操作，估算在没有聚簇和有聚簇情况执行下列查询所需时间？

   select * from emp where eid between 10000 and 20000;

3. 解释散列簇.

2.2 数据管理

- 数据库运行过程中,有一系列工作要作:我们希望没有授权的用户不能操纵数据库,此为的**安全性**;我们希望被授权的用户在出现更新错误时能保持数据的**完整性**,此为数据库的完整性控制;我们希望了解数据库的整体情况,可以查询**系统目录**;我们希望数据库遇到故障时,把损失降低到最小程度,此为**数据库恢复**。

  2.2.1安全性

  2.2.2完整性约束

  2.2.3系统目录

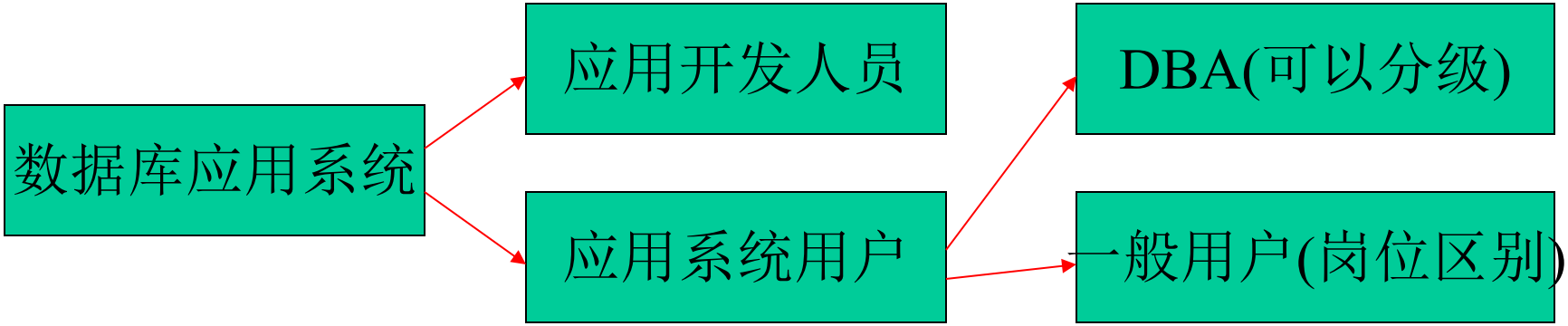
  2.2.4数据库恢复

2.2.1 安全性

- 安全性控制保护数据库以防止不合法的使用所造成的数据泄露和破坏。其基本措施是**存取控制**：确保只授权给有资格的用户访问数据库的权限，同时令所有未被授权的人员无法接近数据。
- Grant语句是表所有者授予一个或一类用户访问表的各种权利的SQL命令。访问列的安全性可通过视图实现。Revoke语句撤消对一个表的权限。
- 表的所有者自动拥有所有权限，而且不能被取消。

2.2.1.1用户管理
2.2.1.2权限管理
2.2.1.3角色
2.2.1.4权限授予
2.2.1.5资源管理
2.2.1.6审计

2.2.1.1 用户管理



- 数据库应用系统有合法的**用户名单**，为了访问一个数据库，用户须用该数据库确认有效的用户名和口令运行应用程序，连接到数据库中。对不同用户可以授予不同权限，每个用户只能在自己权限范围内活动。**不同用户有不同的权限,责任明确，保障安全**。
- 用户数:同一时刻登录到数据库的用户数(ORACLE中初始化参数LICENSE_MAX_SESSIONS)；数据库可容纳的不同的用户名总数(ORACLE中初始化参数LICENSE_MAX_USERS).

2.2.1.1 用户管理

- **CREATE USER** username
  IDENTIFIED BY password
  [DEFAULT TABLESPACE tsname]
  [TEMPORARY TABLESPACE tsname]
  [QUOTA [n [K|M]] [UNLIMITD] ON tsname]
  [PROFILE filename];

DEFAULT TABLESPACE用户创建的对象的表空间，TEMPORARY TABLESPACE用户暂存段表空间，缺省为SYSTEM表空间； QUOTA指定表空间的限制配额。每个用户可以设置一个配置文件，表明自己的操作范围，PROFILE用来指定用户配置文件。

2.2.1.1 用户管理

- ALTER USER username …; 修改用户
- DROP USER username [CASCADE]; 删除用户[及所属对象]。
- ORACLE初始系统的用户:

  SYS CHANGE_ON_INSTALL

  SYSTEM MANAGER

  INTERNALL ORACLE (DBA用户)
- 一个系统中用户名唯一,各个用户的信息可以在DBA_USERS数据字典中查到。

2.2.1.2 权限管理

- 权限分为两大类：系统权限和对象权限。
- **系统权限**用于划分DBA的职责(多个DBA),有：CREATE对象，ALTER对象，DROP对象等80余种。如果将系统权限授予某一用户，则该用户应当被看作DBA。
- **对象权限**是为控制用户使用数据库对象而设立的，数据库对象主要有表、视图、序列、过程/函数和快照等，权限有ALTER、 INSERT、DELETE、UPDATE 、SELECT、INDEX、EXECUTE、REFERENCES，对象权限与SQL语句有直接的对应关系，在某个数据库对象上拥有某种权限，往往意味着可以执行相应的SQL语句。

2.2.1.3 角色

- 角色介于权限和用户之间,**是一组系统权限和对象权限的集合,把它们组合在一起赋予一个名字**,就使得授予权限变得简单.用户被授予某个角色,则拥有该角色的所有权限.引入角色的概念可减轻DBA的负担.
- 数据库应用设计者可以不再理会各个工作站的具体用户是谁,只以抽象的角色代替.
- ORACLE 8预先定义了8种角色,用来限制系统权限,数据库应用设计者可以以此为基础规划和定义一般用户.
- **CREATE ROLE** rolename [NOT IDENTIFIED]
  [IDENTIFIED [BY password|EXTERNALLY]]

  如果角色有口令,用户要知道口令才可扮演该角色,EXTERNALLY表示由操作系统检验口令.

2.2.1.3 角色

- ORACLE预先定义的8种角色:

  CONNECT:连接到数据库,最终用户角色.

  RESOURCE:申请资源创建对象,开发人员角色.

  DBA:具有全部系统权限,可以创建用户.

  IMP_FULL_DATABASE:装入全部数据库内容.

  EXP_FULL_DATABASE:卸出全部数据库内容.

  DELE_CATALOG_ROLE:能删除审计表中记录.

  SELECT_CATALOG_ROLE:查询数据字典.

  EXECUTE_CATALOG_ROLEE:执行过程和函数.

2.2.1.4 权限授予

- GRANT [系统权限|角色]
  TO [用户|PUBLIC|角色]
  [WITH ADMIN OPTION];

  WITH ADMIN OPTION允许权限复制给其他用户.
- GRANT [ALL |对象权限[,对象权限…]|角色]
  ON [TABLE] tablename|viewname
  TO [PUBLIC|username[,username…]|角色]
  [WITH GRANT OPTION];
- 系统权限、对象权限和角色均可用响应的REVOKE语句收回.

2.2.1.5 资源管理

- PROFILE为用户规定了动态资源限制,如果创建用户时不指定资源限制文件,则采用一个叫做DEFAULT文件的配置(缺省配置).资源限制文件可以限制一个单独的调用或整个会话所需要的资源.
- **CREATE PROFILE** filename LIMIT

  SESSIONS_PER_USER n|UNLIMITED --用户可同时进行几个会话
  CPU_PER_SESSION n|UNLIMITED --每个会话用多少1%秒CPU
  CONNECT_TIME n|UNLIMITED --连接数据库时间(分钟)
  LOGICAL_READS_PER_SESSION n|UNLIMITED --会话中调用块数
  …… ;
- **限制用户动态资源的目的是为了安全**,分配给用户的动态资源应与用户的正常业务相当.如果用户资源占用超越分配值,意味着用户在干‘份外’的工作了.系统阻止这类行为通常是结束会话,回退当前事务,返回错误信息.

2.2.1.6 审计

- DBA需要了解用户对表作了那些操作,用户登录多少次等数据库使用情况,以便对系统进行管理。**审计是DBMS用于监控用户活动,保障系统安全的重要方法**。ORACLE将审计记录写在审计痕迹(Audit trail)表,可以查看此表审查数据库的活动。
- **审计用户**：AUDIT 语句选项 [BY 用户]
  [BY ACCESS| SESSION]
  [WHENEVER [NOT] SUCCESSFUL];

BY 用户 对指定用户审计,缺省为对所有用户审计;

BY ACCESS审计全部SQL，BY SESSION一次会话相同SQL只审计一次(节约)，缺省为全部审计;

WHENEVER [NOT] SUCCESSFUL审计成功或不成功的，缺省为不管成功不成功均审计。

2.2.1.6 审计

- 例:AUDIT select table,update table BY user1
  对用户user1的查询或修改表和视图操作审计。
- **审计对象**：AUDIT 选项|ALL ON [模式.]对象
  [BY ACCESS| SESSION]
  [WHENEVER [NOT] SUCCESSFUL];
- 例:AUDIT insert,update ON user1.student
  审计在用户user1的student表上的插入修改操作。
- **审计查看**：USER_AUDIT_TRAIL -定义的审计信息
  DBA_AUDIT_TRAIL -审计结果
  AUDIT_ACTIONS -可审计的命令
- **撤消审计**: NOAUDIT...

2.2.2 完整性约束

- 数据库应用设计者和DBA可以事先为每个数据项确定一些规矩,要求进入数据库的数据必须符合规矩,这些规矩即为完整性约束.
- **实现数据完整性的方法有三种**:应用接口编程、表定义时定义约束、触发器编程描述完整性.后两种方法由DBMS提供,与客户端无关,可降低网络负担.

  2.2.2.1Create table的完整性约束

  2.2.2.2Alter table修改完整性约束

  2.2.2.3触发器(Trigger)

2.2.2.1 Create table的完整性约束

- CREATE TABLE [schema.]tablename
  (colname dadatype [DEFAULT 值|NULL]
  [列级约束|表级约束][,colname…]);

列级约束:NOT NULL 或[CONSTRAINT 约束名]
UNIQUE|PRIMARY KEY|CHECK(条件)
|REFRENCES 表[(col,…)][ON DELETE CASCADE]]

表级约束:[CONSTRAINT 约束名] UNIQUE(col,col…)
|PRAMARY KEY (col,col…)|CHECK(条件)
|FOREIGN KEY (col,col…) REFRENCES 表 [(col,…)]
[ON DELETE CASCADE]

为了修改约束,可用此子句为列或表约束定一名字

没有此说明时,参照表的主键

2.2.2.1 Create table的完整性约束

- 通过not null,unique,primary key可实现实体完整性,通过发foreign key refrences可定义参照完整性,on delete cascade表示连带删除.
- 例:

```
create table orders (ordno integer not null, month char(2),
 cid char(4) not null,aid char(4) not null, pid char(4) not null,
 qty integer not null constraint qck check(qty>=0),
 dollars float default 0.0 constraint dck check(dollars>=0.0),
 primary key (ordno),
 constraint cidref foreign key (cid) references customers,
 constraint aidref foreign key (aid) references agents,
 constraint pidref foreign key (pid) references products);
```

2.2.2.2 Alter table修改完整性约束

- Alter Table语句允许DBA改变表的结构,加入或改变列,加入或删除各种约束.(注意权限)
- ALTER TABLE tablename
  [ADD (col 类型 [DEFAULT 值|NULL] [列级约束|表级约束][,colname...])]
  [DROP COLUMN col|(col,col...)]
  [MODIFY (col 类型 [DEFAULT 值|NULL])]
  [DROP CONSTRAINT 约束名]
  [DROP PRIMARY KEY];
- **如果表约束被命名,可以用该名字drop该约束,用add子句加一新的表约束.**注意加入新列影响行的物理存储,可能使他们不能存储在磁盘的当前位置.

2.2.2.3 触发器(Trigger)

- 触发器类似于过程和函数,**是一个程序块**,但过程是被显式调用,而触发器是当事件发生时,被自动调用(触发),且触发器不接受参数。
- 触发器能引起一个动作的执行,它可以用来实现**过程性约束**.建表时创建的约束称为**非过程性约束**。
- Sybase数据库最早支持触发器,现在Oracle、DB2 UDB、Informix、SQL Server都支持,但Core SQL-99不支持。
- 定义触发器动作时,各个产品用自己的过程性语言。以下以Oracle为例说明。

2.2.2.3 触发器(Trigger)

- CREATE [OR REPLACE] TRIGGER trigname
  BEFORE|AFTER|INSTEAD OF --触发时间
  INSERT|DELETE|UPDATE [OF col,…]--触发事件
  ON tabname --只能对表
  FOR EACH ROW|STATEMENT --行级触发
  [WHEN (条件)] BEGIN ..PL/SQL块.. END;

触发时间说明事件前或后或只触发不执行事件;

触发事件可以是增删改或三者的组合;

行级触发对满足WHEN条件的每一行都单独执行触发体,
语句级触发只在事件发生时触发一次;

行级触发器中可以区分新值和旧值,事件之前的值为old,
事件之后的值为new,old和new之前必须有冒号(:);

触发器中可使用inserting,deleting,updating三个逻辑变量,
如果INSERT行为发生,则inserting变量为真,否则为假.

2.2.2.3 触发器(Trigger)

- 例:create or replace trgger t_c
  after update on coustomers
  for each row when (new.discnt>15.0)
  declare v1 number;
  v2 number;
  begin
  v1 :=:old.discnt;
  v2 :=:new.discnt;
  dbms_output.put_line('old:'||to_char(v1)
  ||'new:'||to_char(v2));
  end;

说明:对表coustomers的修改操作发生后,把折扣率大于15的旧折扣率和新折扣率列出.

2.2.2.3 触发器(Trigger)

- Create table定义的**非过程性约束**的种类是有限的,而**过程性约束**留出程序接口,可以实现任何想法的约束.
- 完整性约束是为了防止错误的SQL意外地破坏数据完整性.当约束不满足时,非过程性约束只能是简单地拒绝接受数据,而通过过程性约束还能实现应该做什么.
- 显然非过程性约束比过程性约束更容易理解,非过程性约束没有过程性约束那样灵活,相对更成熟稳定.
- 触发器还可用于面向对象的想法的实现.

2.2.3 系统目录

- 所有的关系数据库都有系统目录,它包含数据库中定义的对象的信息,这些对象有用户、表、列、索引、视图、权限、约束等等,它是**由系统自动维护,用户不能直接更新,但可以查询**.
- ORACLE的数据字典即为系统目录.根据前缀可分为三种不同形式:

  USER_… 用户视图(当前用户的,不列所有者)

  ALL_… 扩展的用户视图(列所有者)

  DBA_… DBA视图
- ORACLE数据字典的另一种形式是动态性能视图, V$…

2.2.3 系统目录

- ALL_TABLES 表信息
- ALL_TAB_COLUMNS 表的列信息
- ALL_CONSTRAINTS员 表的约束信息
- ALL_TAB_COMMENTS 表的描述信息
- ALL_COL_COMMENTS 列的描述信息
- ALL_TAB_GRANTS 表的权限信息
- USER_INDEXES 索引信息
- USER_IND_COLUMNS 索引的列信息
- USER_CLUSTERS 聚簇信息
- USER_VIEWS 视图信息

2.2.3 系统目录

- USER_SEQUENCE　　同义词信息
- ALL_USERS　　用户信息
- USER_SYS_PRIVS　　用户系统权限信息
- USER_TAB_PRIVS　　用户对象权限信息
- DBA_ROLES　　角色信息
- ROLE_SYS_PRIVS　　角色的权利信息
- DBA_PROFILES　　用户资源配置文件信息
- USER_TRIGGERS　　触发器信息处理

……

2.2.3 系统目录

- DICTIONARY可列出所有数据字典表和视图的名字及描述信息,DICT_COLUMNS列出这些字典列和视图的所有列及描述信息.
- 通过对数据字典的查看,可以了解整个数据库的组成.数据库对象本身就是一对一、一对多、多对多的关系,我们可进一步体会关系数据库的特点.

2.2.3 系统目录

- **面向对象的目录表**:当用户在数据库中创建用户定义类型和用户定义函数时,不仅要记录这些类型和函数的存在,还要记录它们的相互依赖关系.
- ORACLE中对象关系的目录表:

  USER_OBJECT_TABLES

  USER_TYPES

  USER_DEPENDENCIES

2.2.4 数据库恢复

- **事务**是数据库的基本工作单位,每个运行事务对数据库的影响要么在数据库中,要么不反映在数据库中.这样才能保证数据库的一致完整.
- 数据库运行过程中的故障可能破坏数据的完整一致,**故障类型**有:

  事务故障:违反完整性约束引起事务夭折.

  系统故障:软硬件错误断电引起事务夭折.

  介质故障:磁盘损坏部分或全部数据丢失.
- 恢复的技术:**数据转储(备份)和日志**.

2.2.4 数据库恢复

- 为了提高效率,对数据库对象的操作的结果不是立刻写入磁盘,而是保留在缓冲区中,从而可以被不同事务一次又一次地使用,如果它变得不常用,则写入磁盘并清出缓冲区.**缓冲区采用最少使用算法(LRU)决定最长时间内未被使用的缓冲区被写入磁盘的数据文件中.**缓冲区是否写入磁盘数据文件与其中的数据是否提交无关,也即未提交的数据有可能被先期写入磁盘上.后台进程DBWR只服从于LRU,不服从于事务.
- 为了防止因故障原因使**留在缓冲区但已提交的数据反映不到磁盘上,或者写入磁盘的数据没有提交**,利用**日志**来记录对数据库的更新操作.日志项记录更新操作的足够信息(事务开始、更新前更新后的值、事务结束).

2.2.4 数据库恢复

- 日志记录首先保存在日志缓冲区中,在合适的时刻被写入磁盘上的日志文件中,该文件包括了在过去某个时段中的所有日志项.
- 数据缓冲区和日志缓冲区各自向数据文件和日志文件转移数据,系统采用**先写日志原则**协调这两项行为:

  数据库系统通常在日志缓冲区创建一个日志序列号(LSN),LSN_BUFFMIN为日志缓冲区被写回到磁盘后的最小序列号.

  每个数据缓冲区页面记录该页最近一次更新操作的日志序列号,记为LSN_PGMAX.

  数据缓冲区页面只有当LSN_PGMAX比LSN_BUFFMIN小时,才能被LRU到磁盘.

2.2.4 数据库恢复

- 先写日志原则保证了写入磁盘的更新操作肯定在日志文件中作了记录,没有写入磁盘的更新操作在日志文件中也可能作了记录(事务提交和日志缓冲区满也会引起日志缓冲区写入日志文件).。
- 对事务故障和系统故障的恢复需要根据日志文件作两种不同类型的处理:

  前卷(ROLL FORWARD):对已提交(已结束)的事务重新运行并提交,保证已提交的事务的数据更改正确地写入数据库中.

  回退(ROLLBACK):对未提交(未结束)的事务做撤消处理,用日志记录中更新前值依次替换数据库中的数据,使数据库回到该事务执行之前的状态.

2.2.4 数据库恢复

- 恢复过程的时间和ROLLBACK和ROLL FORWARD过程中要读取的日志文件的长度成正比.为了减少恢复过程的时间,记录系统经过一段合理时间之后的一个一致状态,恢复操作即可不用在此之前发生的日志数据项.
- 将所有已修改过的数据缓冲区均写入数据文件的特定操作称为**检查点**.
- 一个检查点的规则:检查点操作开始,新的事务不能开始,现有事务继续执行直到提交并写入磁盘,系统写入一个特殊的日志文件项(CKPT)到日志文件中,检查点操作结束.
- 以上规则可能消耗很多时间,阻塞系统操作.不同系统实现检查点的规则不一定相同,使用复杂的检查点规则的系统能够维护一个平稳的事务输出.

2.2.4 数据库恢复

- 激发检查点出现的方式有两种:

  在写完指定的日志文件块数之后检查点出现,控制检查点出现的频率.ORACLE初始化文件的LOG_CHECKPOINT_INTERVAL可以指定块数,频率越高,例程恢复时间越短,但增加后台进程DBWR的I/O开销,影响系统整体性能.

  当联机日志文件写满时检查点被激发.在归档方式下,一旦日志文件填满,系统就运行检查点,在检查点结束后,进行日志文件归档,归档完成之后,联机日志才能复用.

2.2.4 数据库恢复

- 数据库应用中,日志文件是不可缺少的.为了确保可以恢复,可做日志镜像.
- 恢复的策略:

  事务故障:系统反向扫描日志,撤消此事务对数据库的修改(UNDO).无需DBA介入,**例程恢复**.

  系统故障:系统正向扫描日志,对未执行完的事务撤消(UNDO),对执行完的事务(数据可能在缓冲区中)重做(REDO).无需DBA介入,系统重新启动时**例程恢复**.

  介质故障:DBA装入最新的后备数据库副本,DBA装入有关的日志文件副本,DBA**执行系统提供的恢复命令**,重做已完成的事务.

作业：

1.利用非过程性约束和过程性约束完成完整性定义:保证表agents的行的城市值必须是表coutomers中的城市之一.

2.CAP数据库中的四张表对应各自的用户,彼此不能相互操作,请给出让彼此之间可以相互查看的措施.

3.结合数据库恢复的内容谈谈如何保证数据库的可靠性.

3. 数据库应用程序体系结构

- 数据库系统是指在计算机系统中引入数据库后的系统,一般由数据库、数据库管理系统、应用系统、DBA、用户组成。
- 本章介绍数据库应用程序的数据访问与数据库应用系统的体系结构。

3.1嵌入式SQL

3.2事务处理技术

3.3数据库应用系统体系结构

3.4中间件

3.1 嵌入式SQL

- 为什么使用嵌入式SQL？
  - —有些数据访问任务对于交互式的非过程的SQL是无法完成的任务。
  - —使用交互式SQL，必须知道表名、列名并且能够写出符合语法的SQL语句。
  - —实际的应用系统是非常复杂的，数据库访问只是其中一个部件。有些动作如与用户交互、图形化显示数据等只能用高级语言实现。
- 嵌入到过程性主语言中使用的SQL称为嵌入式SQL。主语言可以是C或Java(不一定是Windows环境).也可以是Visual Basic, Delphi (Windows环境)等。
- ORACLE的Pro*C即是使用嵌入式SQL的平台。

3.1 嵌入式SQL

- 把SQL嵌入主语言使用时必须解决的三个问题：

1．区分 SQL语句与主语言语句,用EXEC SQL开始.

2．数据库工作单元和程序工作单元之间的通信.

SQL语句可以使用主语言的程序变量(简称主变量),这些变量名前加冒号(:)作标志，以区别于字段名。这些变量由BEGIN DECLARE SECTION与END DECLARE SECTION语句之间说明.

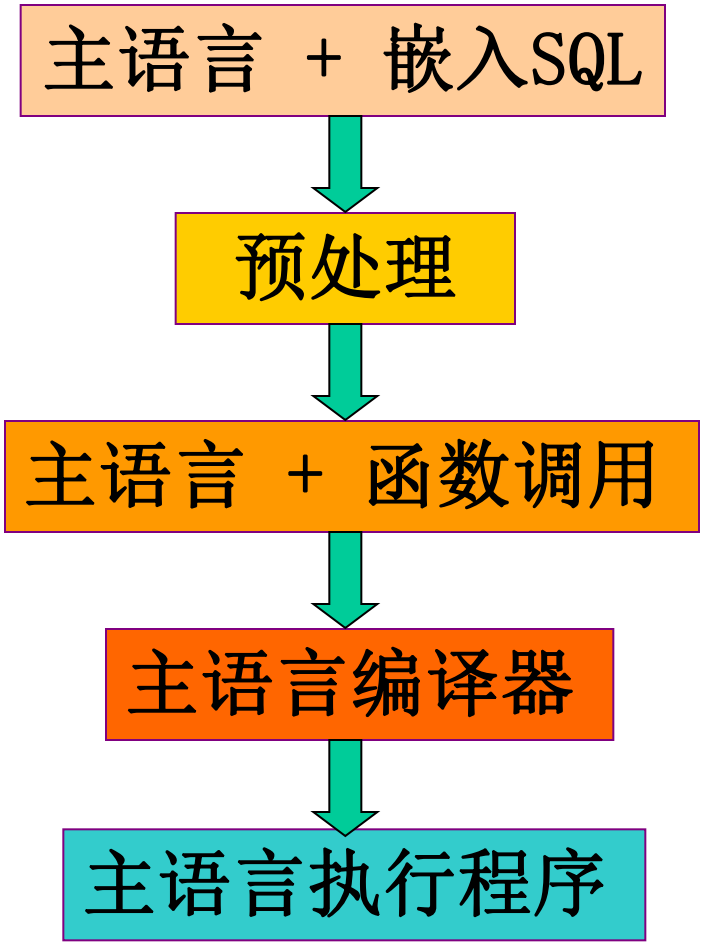
主语言中不能引用数据库中的字段变量.

SQL语句执行后，系统要反馈给应用程序若干信息,这些信息送到SQL的通信区SQLCA。SQLCA用语句EXEC SQL INCLUDE加以定义。

3．一个SQL语句原则上可产生或处理一组记录，而主语言一次只能处理一个记录，为此必须协调两种处理方式。这是用游标来解决的。

3.1 嵌入式SQL

- 嵌入式SQL的执行



预处理器把嵌入的SQL语句从主程序命令中分离出来,转换成相应的主语言语句. 例:

1.UNIX环境下,程序员创建包含SQL的main.pc的源程序.

2.Oracle程序员调用预编译器:

proc iname=main.pc

生成main.c,其中SQL语句被纯c语句(Orcale运行期库函数)替换.

3.主语言编译

cc -c main.c,生成main.o目标文件.

4. 连编生成可执行文件.

3.1.1 C语言中嵌入式SQL

```
#include <stdio.h>
#include “prompt.h”
exec sql include sqlca;   /*声明通讯区*/
char cid_prompt[]=“请输入顾客号:”;
int main( )
{
  exec sql begin declare section;
       char cust_id[5],cust_name[14];
       float cust_discnt;
       char user_name[20],user_ped[20];
  exec sql end declare section;
       /*声明变量*/
  exec sql whenever sqlerror goto report_error;
  exec sql whenever not found goto notfound;
       /*出错处理*/
  strcpy(user_name,”mytest”);
  strcpy(user_pwd,”test”);
  exec sql connect :user_name
       identified by :user_pwd; /*连接*/
while((prompt(cid_prompt,1,cust_id,4))>=0)
{
     exec sql select cname,discnt
        into :cust_name,:cust_discnt
           from customers where cid=:cust_id;
     exec sql commit work;
     printf(“顾客名: %s 折扣率: %5.1f\n”,
           cust_name,cust_discnt);
     continue;
notfound: printf(“没找到 %s,继续\n”,cust_id);
}   /*根据输入的顾客ID查询顾客信息*/
exec sql commit release; /*正常释放连接*/
return 0;

report_error:
    print_dberror(); /*出错信息*/
    exec sql rollback release; /*释放连接*/
    return 1;
}
```

3.1.1 C语言中嵌入式SQL

- 区分SQL语句与C语言语句

  嵌入的SQL语句以**EXEC SQL**开始，以分号(;)结束。

  例:exec sql select came,discnt

  into :cust_name,:cust_discnt

  from customers where cid=:cust_id;

- 嵌入SQL语句与C语言之间的数据传递

  宿主变量(宿主变量出现于SQL语句中时，前面加（:）以区别列名)

  C变量既可以用在C语句中，也可用在SQL语句中，用来在两者之间传递数据。

3.1.1 C语言中嵌入式SQL

- 宿主变量的声明

在嵌入式SQL语句中使用宿主变量,必须先声明它们:声明为通常的C变量，并将其放在下列标识语句之间，

EXEC SQL BEGIN DECLARE SECTION

EXEC SQL END DECLARE SECTION

例: exec sql begin declare section;

char cust_id[5]='c001';

char cust_name[14];

float cust_discnt;

exec sql end declare section;

3.1.1 C语言中嵌入式SQL

- 在SQL中建立连接和释放连接

  在一个嵌入式SQL程序开始,程序面临着和交互式用户同样的问题：怎样与数据库建立连接.

例: exec sql begin declare section; /*声明变量*/
char user_name[10],user_ped[10];
exec sql end declare section;
strcpy(user_name,'mytest'); /*变量赋值*/
strcpy(user_pwd,'test');
exec sql connect :user_name
identified by :user_pwd; /*Oracle连接*/

例:exec sql commit release;/*Oracle断开连接*/
exec sql rollback release;/*Oracle断开连接*/

3.1.2 游标

- SQL与主语言之间操作方式的协调

  SQL：一次一集合。

  C语言：一次一记录。

  – 游标：在查询结果的记录集合中移动的指针。

  若一个SQL语句返回单个元组，则不用游标。

  若一个SQL语句返回多个元组，则使用游标。

  – 不需要游标的数据操作

    - 结果是一个元组的select语句

```
exec sql select came,discnt
 into :cust_name,:cust_discnt
 from customers where cid=:cust_id;
```

3.1.2 游标

– 需要游标的数据操作

当select语句的结果中包含多个元组时，使用游标可以逐个存取这些元组。

活动集：select语句返回的元组的集合。

当前行：活动集中当前处理的那一行。游标即是指向当前行的指针。

游标分类：

- 滚动游标：游标的位置可以来回移动，可在活动集中取任意元组。
- 非滚动游标：只能在活动集中顺序地取下一个元组。
- 更新游标：数据库对游标指向的当前行加锁，当程序读下一行数据时，本行数据解锁，下一行数据加锁。

3.1.2 游标

- 定义一个游标，使之对应一个select语句

  DECLARE 游标名 [**SCROLL**] **CURSOR**

  FOR select语句 [**for update** [**of**列表名]]

  for update任选项，表示该游标可用于对当前行的修改与删除。

- 打开一个游标，执行游标对应的查询，结果集合为该游标的活动集。

  OPEN 游标名

在游标被声明之后,它仍然不是活动状态,在程序开始检索信息之前,执行SQL的OPEN语句打开这个游标，游标被打开之后，可使用取（fetch）操作从游标中每次检索一行，程序完成检索后，应关闭游标.

3.1.2 游标

- 在活动集中将游标移到特定的行，并取出该行数据放到相应的宿主变量中。

  FETCH [**NEXT** | **PRIOR** | **FIRST** | **LAST** |
  CURRENT | **RELETIVE n** | **ABSOLUTE n**]
  游标名 **INTO** [宿主变量表]

- 关闭游标，释放活动集及其所占资源。需要再使用该游标时，执行open语句。

  CLOSE 游标名

- 删除游标，以后便不能再对该游标执行open语句了.

  FREE 游标名

3.1.2 游标

```
#define ture 1
#include <stdio.h>
#include “prompt.h”
exec sql include sqlca;    /*声明通讯区*/
int main( )
{
  char cid_prompt[]=“请输入顾客号:”;
  exec sql begin declare section;
      char cust_id[5],agent_id[4];
      double dollar_sum;
      char user_name[20],user_ped[20];
  exec sql end declare section; /*声明变量*/
  exec sql declare agent_dollars cursor for
      select aid,sum(dollars) from orders
      where cid=:cust_id group by aid;/*游标*/
  exec sql whenever sqlerror goto report_error;
  exec sql whenever not found goto finish;
       /*出错处理*/
  strcpy(user_name,”mytest”);
  strcpy(user_pwd,”test”);
exec sql connect :user_name
        identified by :user_pwd; /*连接*/
while((prompt(cid_prompt,1,cust_id,4))>=0){
     exec sql open agent_dollars;
     while (ture) {
          exec sql fetch agent_dillars into
            :agent_id,:dollar_sum; /*游标取值*/
          printf(“ %s   %11.2f\n”,
           agent_id,dollar_sum);
      }
      continue;
finish: exec sql close agent_dollars;
         exec sql commit work; /*游标关闭*/
}   /*输入顾客ID的代理商和金额*/
   exec sql commit release; /*正常释放连接*/
   return 0;
report_error:
    print_dberror(); /*出错信息*/
    exec sql rollback release; /*释放连接*/
    return 1;
}
```

3.1.3 通讯区

- SQL语句执行信息反馈
  - 良好的应用程序必须提供对错误的处理,应用程序需要知道SQL语句是否正确执行了,发生错误时的错误代码，执行时遇到特殊情况时的警告信息。
  - SQL通讯区SQLCA是一个已被声明过的内存结构,每一嵌入SQL语句的执行情况在其执行完成后写入USERCA结构中的各变量中，根据SQLCA中的内容可以获得每一嵌入SQL语句执行后的信息,应用程序就可以做相应的处理。
  - 为了说明 (USERCA),必须在应用程序中包括:

EXEC SQL INCLUDE SQLCA;

3.1.3 通讯区

- SQLCODE：SQLCODE是SQLCA的结构一个成员。它是一个每次执行完SQL语句都被更新的整型变量。
  - 如果执行成功，SQLCODE为0；
  - 如果语句没有产生错误，但遇到了没有数据，SQLCODE为100；
  - 除去成功返回0或没有数据返回100的情况，其他都是某种错误并返回一个负数。
- SQL-99和X/OpenSQL等标准提倡用SQLSTATE取代SQLCODE检测SQL的执行情况，并将SQLSTATE设置为标准，但在产品中还不一定支持。例如ORACLE中,预处理程序运行时必须设置MODE=ANSI才能支持SQLSTATE.

3.1.3 通讯区

- SQLSTATE：SQLSTATE作用与SQLCODE作用相同，但它是在标准方式下起作用。SQLCODE是一个整数，SQLSTATE是一个长度为5的只能由A到Z字符和0到9数字组成的字符串。5个字符分为两组，前两个为类码，后三个为子类码。SQL标准规定：分类码的第一个字符可为A到H或0到4，这样分类主要为了统一，第一个字符为I到Z或5到9的分类码留给软件开发者去定义。

  分类码00指成功,01指成功但产生一个警告,02指没有数据,等价于SQLCODE的100,非00、01和02指语句没有成功完成。可以查阅数据库系统产品的嵌入式SQL参考指南得到SQLSTATE、SQLCODE、SQLCA，确定主要的出错条件，以它们的报告方式。

3.1.4 错误处理

- Whenever语句使我们在遇到出错和其他情况时，控制程序的运行:

  EXEC SQL WHENEVER 条件 动作

  条件可以是
  - SQLERROR：执行错误，如连接不成功等；
  - NOT FOUND：没有数据，如游标指针到头或到尾等；
  - SQLWARNING：警告错误。

  动作可以是
  - CONTINUE：继续正常流程；
  - GOTO 标号：转移到标号行继续执行；
  - STOP：结束程序，撤消当前事务，并断开数据库连接；
  - DO 函数：引发一个对已经命名的函数的调用。

3.1.4 错误处理

- 没有WHENEVER语句时缺省动作是Continue.
- 如果SQLCODE为负,则SQLERROR为真,如果SQLCODE为100或SQLSTATE为02000,则NOT FOUND为真。抽取错误信息的确切代码依赖于所使用的数据库系统。
- WHENEVER语句的主要价值在于减少处理错误的代码行数，另外WHENEVER的语法可以在不同数据库系统之间进行最大限度的移植。例如利用检测条件NOT FOUND在不同数据库产品中可能有不同的sqlca.sqlcode值，但NOT FOUND却始终表示没有发现记录这一错误。

3.1.5 指示变量

- 如果一个宿主变量所对应的数据库字段允许空值，或字符串类型的宿主变量的长度可能小于所对应的数据库字段的长度，则需要一个指示变量来指明数据库访问的返回状态。
- **指示变量**: 是一个C变量，用来指示返回给宿主变量的值是否为null值，以及返回给宿主变量的字符串是否发生了截断。

  指示变量的返回值：

  = 0：取到主变量的值不空，没有发生截断。

  = –1：取到主变量的值为空值。

  > 0：取到主变量的值发生了截断，指示变量的值是截断前的字符串的实际长度。

3.1.5 指示变量

- 指示变量的用法：声明与宿主变量的声明方式一样,在数据操纵语句中,在宿主变量和指示变量之间加(:)或关键字**indicator**。

例:exec sql begin declare section
float cust_discnt;
short int cd_ind;
exec sql end declare section
exec sql select discnt into :cust_discnt
[indicator]:cd_ind
from customers where cid=:cust_id;

在检索之后，如果cd_ind的值为-1,则变量cust_discnt得到了一个空值；如果为0，则cust_discnt中的值是正常的。

3.1.5 指示变量

- 不管读取数据还是更新数据,都可以用指示变量的值-1代表空值。

  例:要将表customers的某一行的discnt值设置为空值，可以这样写:

```
cd_ind=-1
exec sql update customers
  set  discnt=:cust_discnt
       indicator :cd_ind
  where cid=:cust_id;
```

- 实际上还可以进一步扩展指示变量的值，例如DB2 UDB的产品中，如果指示变量的值为-2，意味着是由于一个错误而不是数据库中存储的值而使检索到的值为空值。

3.1.6 嵌入式SQL语句

- 检索:EXEC SQL SELECT [ALL|DISTINCT] 表达式 INTO 宿主变量 FROM 表名 WHERE 条件;
- 游标定义:EXEC SQL DECLEARE 游标名 CURSOR FOR 子查询 [ORDER BY…] [FOR READ ONLY |UPDATE [OF 列名]];
- 游标打开与关闭:EXEC SQL OPEN|CLOSE 游标名;
- 游标取值:EXEC SQL FETCH [NEXT|PRIOR|FIRST|LAST |CURRENT|RELETIVE n|ABSOLUTE n] 游标名 INTO 宿主变量;
- 删除:EXEC SQL DELETE FROM 表名 [WHERE条件 |WHERE CURRENT OF 游标名];
- 修改:EXEC SQL UPDATE 表名 SET 列=表达式 [WHERE 条件|WHERE CURRENT OF 游标名];
- 增加:EXEC SQL INSERT INTO 表名 [(列名)] VALUES(表达式);

3.2 事务处理技术

- 保证数据的一致性和数据库的并行性，是衡量一个数据库的最基本和最重要的指标之一。

  一致性：以一致性规则为基础的数据逻辑关系。如转帐任务不能造成帐户金额的不平衡。

  并行性：各个用户能够实现对数据库资源的共享。

  数据并发是多用户系统的基本要求，数据一致是因为有数据并发存在，为了适应多用户系统，协调管理数据库中的数据，保证各个用户的任务能够顺利准确地完成。

  事务处理是实现数据库一致性和并行性的重要手段。在保证一致性的前提下最大限度地提高并发度。

3.2 事务处理技术

- 数据操纵以SQL为基础，但单条SQL语句不一定能完成所有的用户请求，许多用户请求需要若干条SQL语句，而且这些语句有关联，要么全做，要么全不做。
- 如果许多用户的众多SQL语句不加控制，随意执行，必然破坏数据的一致完整，出现丢失修改、不可重复读、读脏数据的错误。
- 数据库系统要对用户的操作进行控制(事务控制和封锁技术)，保证并发的同时保证数据的一致。

3.2.1 事务概念

- **事务是由一系列操作序列构成的程序执行单元，这些操作要么都做，要么都不做，是一个不可分割的工作单位。**

  事务是数据库提供的一种手段，应用程序员将一系列操作组合在一起作为一个整体以便数据库系统提供保证。事务才是对数据库系统进行操作的最小合法单位。DBMS在事务级而不是在语句级确保数据的一致性。

- 事务以Begin transaction开始，以Commit work或Rollback work结束。

  嵌入式SQL: EXEC SQL COMMIT WORK;

  EXEC SQL ROLLBACK WORK;

3.2.1 事务概念

- Commit work表示提交，事务正常结束。

  Rollback work表示事务非正常结束，撤消事务已做的操作，回滚到事务开始时状态。

  事务的内容只限于数据更新操作(增删改),不包含其他SQL命令。

  在标准SQL中，没有Begin transaction语句，上一条Commit或Rollback或SQL修改命令即为事务的开始。

  有的DBMS使用Begin transaction作为事务的开始。

3.2.1 事务概念

- 事务特性(ACID)
  - 原子性(Atomicity)

    事务的更新操作必须作为一个整体，其中包含的所有操作要么全做，要么全不做。

    原子性由系统**恢复机制**实现。
  - 一致性(Consistency)

    事务的成功完成将数据库从一个一致状态转变到另一个一致状态。

    事务开始前，数据库处于一致性的状态；事务结束后，数据库必须仍处于一致性状态。

    数据库的一致性状态由**用户**来负责。

3.2.1 事务概念

- 事务特性(ACID)
  - 隔离性(Isolation)

    系统必须保证事务不受其它并发执行事务的影响。

    对任何一对事务T_1，T_2，在T_1看来，T_2要么在T_1开始之前已经结束，要么在T_1完成之后再开始执行。

    隔离性通过**并发控制机制**实现。
  - 持久性(Durability)

    一个事务一旦提交之后，它对数据库的影响必须是永久的，无论发生何种系统故障。

    持久性通过**恢复机制**实现。

事务的原子性、隔离性、持久性由数据库系统加以保证，一致性一般由程序员在编写程序中予以保证。

3.2.2 事务的调度

- 事务的调度：DBMS在处理用户提交事务时的策略，即事务调度。事务的执行顺序称为一个调度，表示事务的指令在系统中执行的时间顺序。
  - 一组事务的调度必须保证
    - 包含了所有事务的操作指令
    - 一个事务中指令的顺序必须保持不变。
  - 串行调度
    - 在串行调度中，属于同一事务的指令紧挨在一起。
    - 对于有n个事务的事务组，可以有n！个有效调度。
  - 并行调度
    - 在并行调度中，来自不同事务的指令可以交叉执行。
    - **当并行调度等价于某个串行调度时，则称它是正确的。**

3.2.2 事务的调度

- ANSI SQL-99标准中，一致性级别(隔离级别)的定义
  - serializable：一个调度的执行必须等价于一个串行调度的结果。
  - repeatable read：只允许读取已提交的记录，并要求一个事务对同一记录的两次读取之间，其它事务不能对该记录进行更新。
  - read committed：只允许读取已提交的记录，但不要求可重复读。
  - read uncommitted：允许读取未提交的记录。
- 在SQL-99的语法中允许在启动事务的SQL语句之前设置隔离级别。

3.2.3 事务的数据封锁

- 数据库管理系统采用**数据封锁技术**和事务技术解决并发性和一致性问题.数据封锁有两种封锁模式:

  专用锁:当一个事务以专用方式封锁某个资源时,只有此事务可以改变其中的内容,在该事务结束或释放其资源后,其它事务才能使用;

  共享锁:允许已封锁的资源被其它事务适度共享,多个事务可以在同一资源上申请共享锁.

- 根据封锁的类型、持锁时间、何时释放的规则,将封锁协议分为一级封锁协议、二级封锁协议、三级封锁协议。**不同封锁协议在不同程度上保证数据一致性和系统并发性**。

3.2.3 事务的数据封锁

- 在SQL语句执行时,所有必要的**封锁均由DBMS自动完成(也可显式加锁),事务中所有为执行SQL语句建立的封锁均要延续到事务结束时才能释放**,否则一个数据库对象在同一时间段内被多个事务交替操作,可能造成对数据库数据的破坏.
- 当参与封锁的事务较多,而每个事务封锁的资源也较多时,可能出现‘死锁’情况。
- 预防死锁的方法有一次封锁法、顺序封锁法。
- 死锁的诊断有超时法、有向图法。死锁发生后,DBMS通常选择一个处理死锁代价最小的事务将其撤消,并恢复该事务。

3.2.3 事务的数据封锁

- 两段锁协议(Two-phase Locking)
  - 内容：

①在对任何数据进行读写之前，事务首先要获得对该数据的封锁。

②在释放一个封锁之后，事务不再获得任何其它封锁。

即事务分为两个阶段：

生长阶段：获得封锁。

收缩阶段：释放封锁。

  - 定理：若所有事务均遵从两段锁协议，则这些事务的所有并行调度都是可串行化的。

3.2.4 ORACLE事务控制-回退段

- ORACLE为了适应事务控制设置了**回退段**这一数据库对象.系统利用回退段来确保诸如读一致性、数据库恢复等管理功能。
- ORACLE在缺省情况下,读数据不加锁,通过回退段(Rollback Segment)保证用户不读脏数据和可重复读.
- 表空间中的数据按段来组织,数据段、索引段、暂存段和回退段，回退段是一块磁盘存储区域，**回退段可以由用户创建**，但只能由系统进程使用。

3.2.4 ORACLE事务控制-回退段

- 回退段中的数据是为事务服务的，每执行事务时，系统先在指定回退段上记录将要对数据进行的更改，以事务为单位，各个事务的回退信息链接在一起。
- 当事务要回退时，利用回退信息将数据块中的数据恢复到先前的状态。
- 事务成功提交后，回退信息逐渐失效(在提交之前申请的查询需要这些信息保证**读一致性**，事务提交前对数据的变动不会为其他用户所知晓)。回退段可以循环使用。

3.2.4 ORACLE事务控制-回退段

- 用回退段保证读一致性(不读脏数据和可重复读):

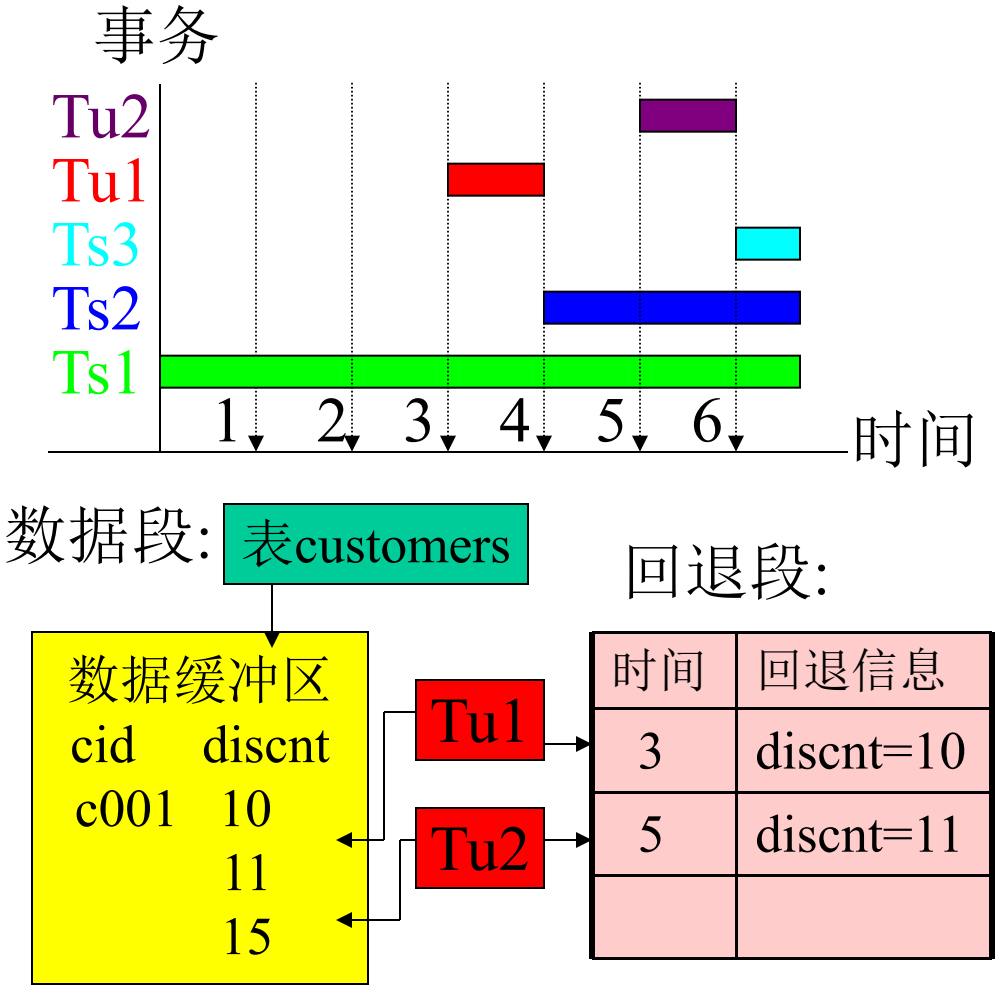
例: 有5个事务并发:
Ts1,Ts2,Ts3为查询事务,
Tu1,Tu2为更新事务.

Tu1:UPDATE customers
SET discnt=discnt*1.1
WHERE cid='c001';

Tu2: UPDATE customers
SET discnt=15 WHERE
cid='c001';

事务开始和完成时间段如右上图:

事务对回退段影响如右下图:



3.2.4 ORACLE事务控制-回退段

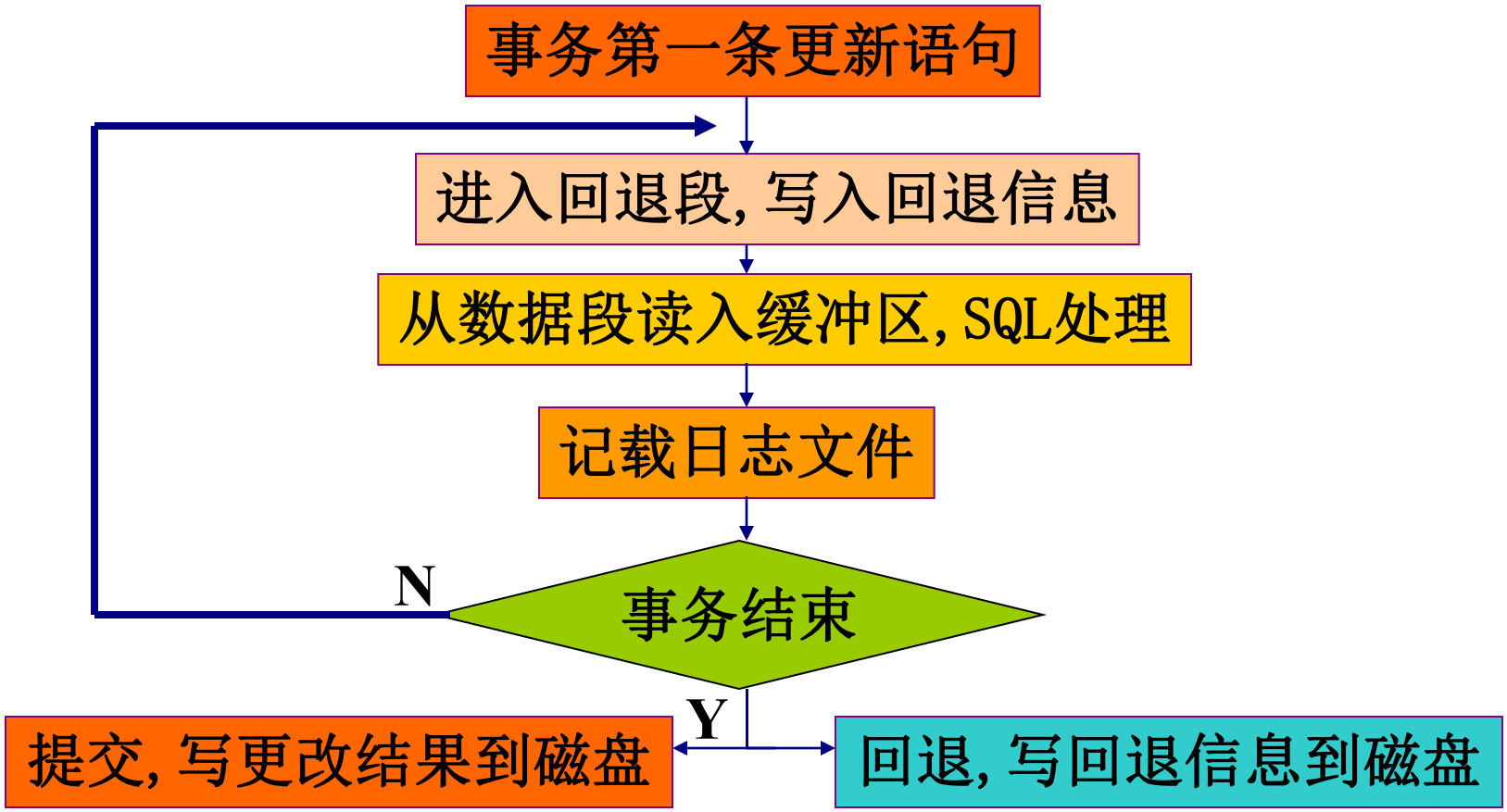
- 用回退段保证读一致性(不读脏数据和可重复读):
  事务的开始顺序和基准时间为:Ts1=0 – Tu1=3 – Ts2=4 – Tu2=5 – Ts3=6;
  事务的完成顺序和基准时间为:Tu1=4 – Tu2=6 – Ts1>6 – Ts2>6 – Ts3>6;
  数据缓冲区的内容是最新更改的结果, 回退段按时间顺序记录了每次更改以前的数据;
  尽管Tu1和Tu2两次更改了discnt的值, 但Tu1开始之前Ts1已开始,所以在时间4结束的Tu1回退信息并不马上消失,保证Ts1从回退段中查得discnt为10,Tu2开始之前Ts2已开始,所以在时间6结束的Tu2回退信息并不马上消失,保证Ts2从回退段中查得discnt为11, Ts3基准时间是6,从缓冲区中查得discnt为15.
  Ts1和Ts2结束后才清除回退段信息.

3.2.4 ORACLE事务控制-回退段

- 在数据库创建时SYSTEM表空间中建立了SYSTEM回退段。每个数据库可以有一个或多个回退段，最好只将SYSTEM回退段用于系统事务，将用户事务分配在其他回退段上，以提高资源利用率。
- 创建**回退段时可同时设定其存储参数**，回退段至少要包括两个区间(MINEXTENTS)，回退段的区间总数受到回退段存储参数MAXEXTENTS的限制。
- 一个事务的回退信息按顺序在指定的回退段中写入。创建多少个回退段及给每个回退段分配多少区间，要考虑可能出现的最大并发事务数，每个回退段所能同时服务的事务数。

3.2.4 ORACLE事务控制-回退段

- 事务的执行过程（采用日志和回退段双重记录事务活动）：



3.2.5 ORACLE事务控制-语句

- ORACLE事务控制语句:
  - **COMMIT [WORK];**

    清除本事务的全部保留点,清除为执行事务建立的封锁机制,结束本事务,提交数据.
  - **SAVEPOINT 保留点名;**

    保留点是一事务范围内的中间标志,经常用于将长事务划分为短小的部分,保留点可标志在长事务的任何点,允许回退该点之后的工作.保留点主要作为一种调试手段使用.在一个事务中如果用相同的保留点名创建第二个保留点,则前一个保留点被删除掉,每个事务最大的活动保留点由初始化参数文件中的参数SAVEPOINTS设定,极限值为255.

3.2.5 ORACLE事务控制-语句

- **ROLLBACK [WORK] [TO [SAVEPOINT] 保留点名]**;

  清除本事务的全部保留点,清除为执行事务建立的封锁机制,结束本事务,回退数据.

  数据定义语句CREATE、DROP总引发提交，系统正常关闭时，未提交事务将隐式提交，一个进程或例程非正常终止，未提交事务将隐式回退。

- **SET TRANSACTION [READ ONLY] [READ WRITE] [USE ROLLBACK SEGMENT 回退段名]**;

  设置事务是对事务的一种控制，建立当前事务为只读事务或读写事务，控制事务使用指定的回退段空间。只读事务不生成回退信息，所以不必分配回退段.缺省状态下事务为读写事务。

3.2.6 ORACLE事务控制-数据锁

- ORACLE提供了各种类型的封锁机制保证并发操作时数据的一致，对数据库对象的封锁由系统自动完成。数据库应用设计者可以干预自动封锁过程以提高整体效率.
- ORACLE主要封锁类型有:
  - **数据锁**: 目的是保护数据,用于封锁表和表中的某些行;
  - **字典锁**: 目的是保护数据库对象的结构,用于表对象的定义;
  - **内部锁**: 用于保护数据库的内部结构,如文件等.
- 数据锁有两种级别的封锁方式:
  - **表封锁**:封锁表中的所有行,数据并行度低.
  - **行封锁**:行级封锁在对应表上加字典锁.

3.2.6 ORACLE事务控制-数据锁

- ORACLE表上的数据锁类型有:
  - RS:行共享,用于让事务封锁表中的部分行,以便于数据修改,但在建立封锁时并不立即修改数据.当事务在一表持有行共享锁时,允许其他事务并行查询、插入、修改、删除。
  - RX:行专有,事务可对表中部分行建立专有锁，使事务可以在以后的操作中多次修改这些行。具有行专有锁的表允许其他事务并行查询、插入、修改、删除。
  - S:共享,共享锁将阻止其他事务修改表中的内容，具有共享锁的表拒绝其他事务插入、修改、删除。
  - SRX:共享行专有,每个表上的共享行专有锁只能由一个事务建立，有这种类型锁的表只能允许其他事务查询。

3.2.6 ORACLE事务控制-数据锁

- X:专有,事务在表上建立专有锁之后，可以独占此表进行写入操作。

- ORACLE中的SELECT不用任何数据锁阻塞其他操作，也称无阻塞查询。INSERT、UPDATE、DELETE引发数据封锁，事务所申请的全部数据锁在事务提交或回退时一起释放，表上的数据锁也可以通过显式语句完成(如下页表)。
- 表上的数据锁在事务执行过程中可能发生变化，如事务为执行带UPDATE子句的SELECT语句，需在选中行上建立行共享锁，如果稍后事务决定修改其中的某些行，则这些行上的行共享锁将自动转换为行专有锁。

3.2.6 ORACLE事务控制-数据锁

SQL语句	封锁	允许模式 RS	RX	S	SRX	X
Select	无	Y	Y	Y	Y	Y
Insert	RX	Y	Y	N	N	N
Delete	RX	Y	Y	N	N	N
Update	RX	Y	Y	N	N	N
Select for update of	RS	Y	Y	Y	Y	N
Lock table in row share mode	RS	Y	Y	Y	Y	N
Lock table in row exclusive mode	RX	Y	Y	N	N	N
Lock table in share mode	S	Y	N	Y	N	N
Lock table in share row exclusive mode	SRX	Y	N	N	N	N
Lock table in exclusive mode	X	N	N	N	N	N

3.2.6 ORACLE事务控制-数据锁

- 数据库设计者应该利用各种方法减少数据对象的封锁时间，以提高系统并发能力：

  在事务中尽可能晚地发出封锁申请；

  在备份的数据库对象上操作以减少封锁时间；

  利用索引减少封锁时间，查询操作可以根本不访问索引关联的表，而直接给出查询结果。

- 数据库设计者应该按照约定顺序封锁表减少出现死锁的可能性，以提高系统并发能力。

作业：

1.解释游标.

2.针对CAP数据库,编写一应用程序:给出任一产品的pid,列出购买该产品的前5名顾客(cid,cname)及其购买数量,即这些顾客购买该产品的数量最大.

3.说明数据库管理系统保证数据一致性和并发性的机制.

4.ORACLE为什么设置回退段?

5.数据库设计者在保证数据一致性和并发性方面可以有哪些措施?

3.3 数据库应用系统体系结构

- 数据库应用系统的发展经历了主机/终端模式、客户机/服务器模式之后，随着Internet的发展，又出现了浏览器/服务器模式，以及分布式数据库体系结构。

  3.3.1简介

  3.3.2客户机/服务器结构

  3.3.3 C/S结构的数据库管理系统

  3.3.4 C/S结构的数据库系统

  3.3.5浏览器/服务器模式

  3.3.6分布式数据库体系结构

3.3.1简介

- 早期数据库系统是**集中式的体系结构**，所有访问数据库的应用程序以及用户终端发送并接受数据的通信都在一个宿主计算机(UNIX大型或小型机)上运行。
- 随着PC机的兴起,**单用户的DBMS**出现,DBMS的功能和数据库应用功能结合在一个应用程序中，数据库应用处理用户输入和屏幕输出的同时,也处理对数据库中数据的访问。
- 基于PC的单用户DBMS不支持事务处理和回退恢复,不能保证数据的安全和完整，但查询速度并不一定慢。

3.3.1简介

- 基于PC的**多用户DBMS建立在文件服务器前提下**，客户端运行服务器的应用程序，文件服务器将客户申请的文件传送给客户机器，数据处理过程在客户机器上进行。这种方式网络传输的是数据文件，传输量大，数据锁定在解决并发时存在困难。
- 事务处理的性质使数据库系统从集中式、PC系统过渡到客户机/服务器系统，并使分布式系统成为可能。
- **客户机/服务器**系统最本质的特点在于：客户PC运行数据库应用(界面处理)，数据库服务器运行全部或大部分DBMS(数据处理)。服务器运行SQL，将查询结果传送到客户端，减少了网络信息的传输,系统采用数据锁定、事务技术、存储过程等数据库技术保证数据完整一致准确。

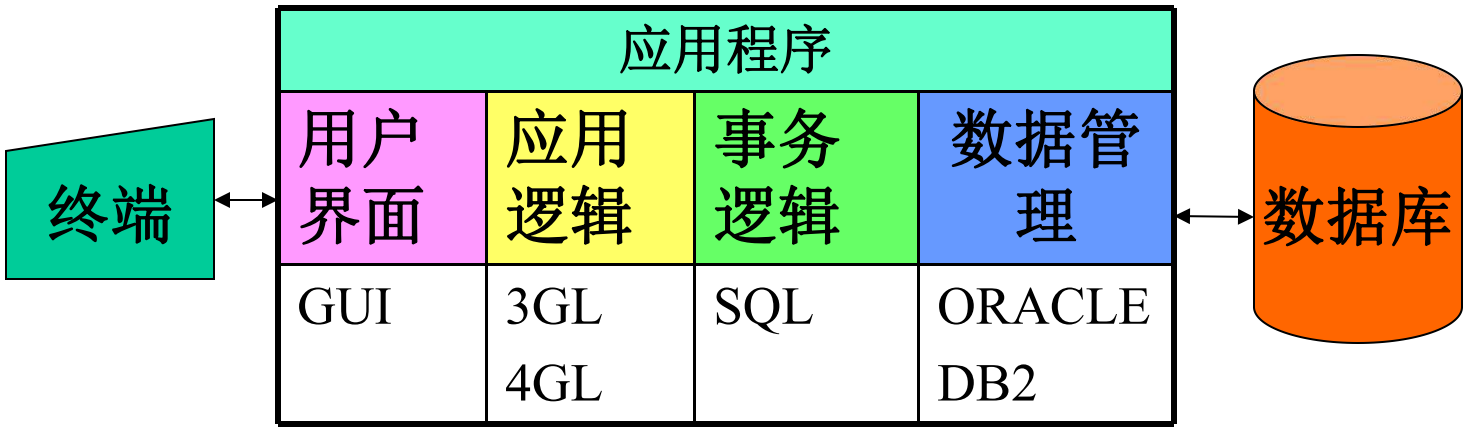
3.3.2客户机/服务器结构

- 客户机/服务器(Clien/Server)结构可以指硬件结构,也可以指软件结构。
  - **硬件结构**指某项任务在两台或多台计算机之间进行分配,客户机来运行提供用户接口和前端处理的应用程序,服务器提供可供客户机使用的各种资源和服务。

    客户机在完成某一项任务时，通常要利用服务器上的共享资源和服务器提供的服务。在一个客户机/服务器体系结构中可以有多台客户机、多台服务器。

3.3.2客户机/服务器结构

- **软件结构**指把一个应用系统按照逻辑功能分成四个组成部分:用户界面、应用表示逻辑、事务逻辑、数据管理，按照其相对角色的不同区分为客户端和服务器端软件。客户软件能够请求服务器软件的服务。客户软件和服务器软件可以分布在网络中不同的计算机节点上，也可以放置在同一台计算机上。



3.3.2客户机/服务器结构

- 常见C/S体系有两层结构和三层结构。
- **两层C/S结构**的基本工作方式：客户程序运行用户的应用程序,向数据库服务器发送SQL请求，数据库服务器接受客户机的请求,并将处理结果返回客户端。

  一个功能强大的客户应用开发语言和一个多用途的用于传送客户请求到服务器的机构是整个两层结构的核心。在一个数据存取事件中，数据由服务器实施存储和访问，数据库引擎负责处理从客户端发来的请求。

  把SQL语言从客户机传送到服务器上必须能识别服务的标识符或由一个应用程序接口来完成，还必须知道服务器的位置、数据组织形式以及数据如何定义。在服务器中，请求将得到存储逻辑和处理的进一步优化，例如使用权限、完整性、并发控制等。

3.3.2客户机/服务器结构

- 两层C/S结构具体又分为两种实现方式：
  - 一种是客户端完成界面显示和应用逻辑，服务器完成事务逻辑和数据管理。这种情况是以客户为中心的。

    这种方式下，表示部分和应用逻辑耦合紧密，比较适用于应用相对简单、数据访问量不大的情况。
  - 另一种以服务器为中心，一些重要的应用逻辑放在服务器上，充分利用服务器的计算能力，通常以存储过程和触发器出现，减少网络压力，提高系统性能。

    这种方式下，存储程序依赖于特定数据库，不同数据库间的移植不太容易。

3.3.2客户机/服务器结构

- 把两层结构中服务器部分和客户端部分的应用单独划分出来，即形成三层C/S结构。
- 在X/Open DTP标准中描述了三层C/S模型：由应用程序定义各种操作来执行完成特定任务，它定义事务的范围并把服务要求提交给事务管理器、通信管理器及一个或多个资源管理器。

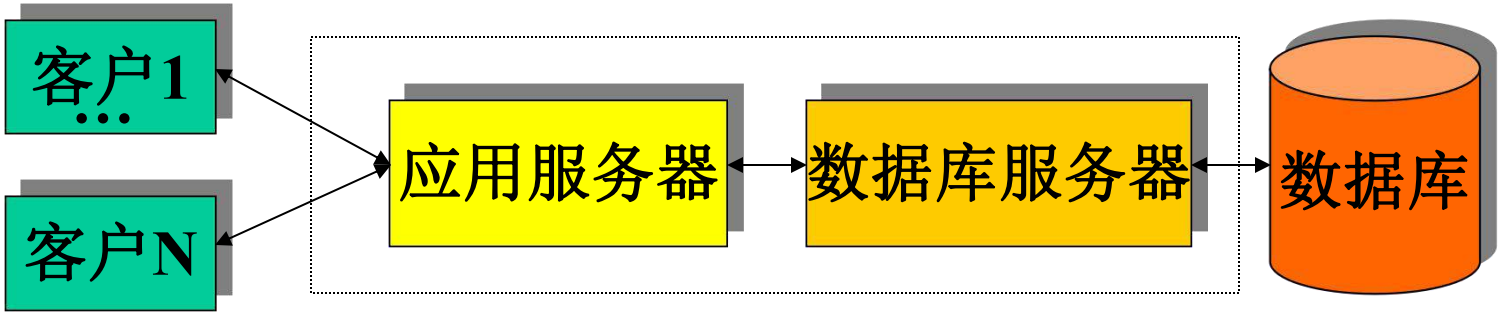
  资源管理器提供应用程序的事务间的服务，通常是一个数据库。

  事务管理器提供事务的服务。如果事务跨越多个平台，那么其中一个的事务管理器为该事务的事务管理器。

  通信管理器提供通信服务的接口以及管理应用层协议。

3.3.2客户机/服务器结构

- 数据库应用的三层C/S结构将应用分成表示部分、应用逻辑(或称商业逻辑)、数据访问部分。三层C/S结构使各部分相互独立并单独实现，分别称为客户、应用服务器和数据库服务器。
- 三层C/S结构中数据在发送到网络之前由功能服务器加以过滤，网络流量会减少，另外客户端并不是直接同数据库打交道，而是通过中间层的统一调用来实现，在**灵活性和独立性方面较好，适合于不同数据库的互联**。

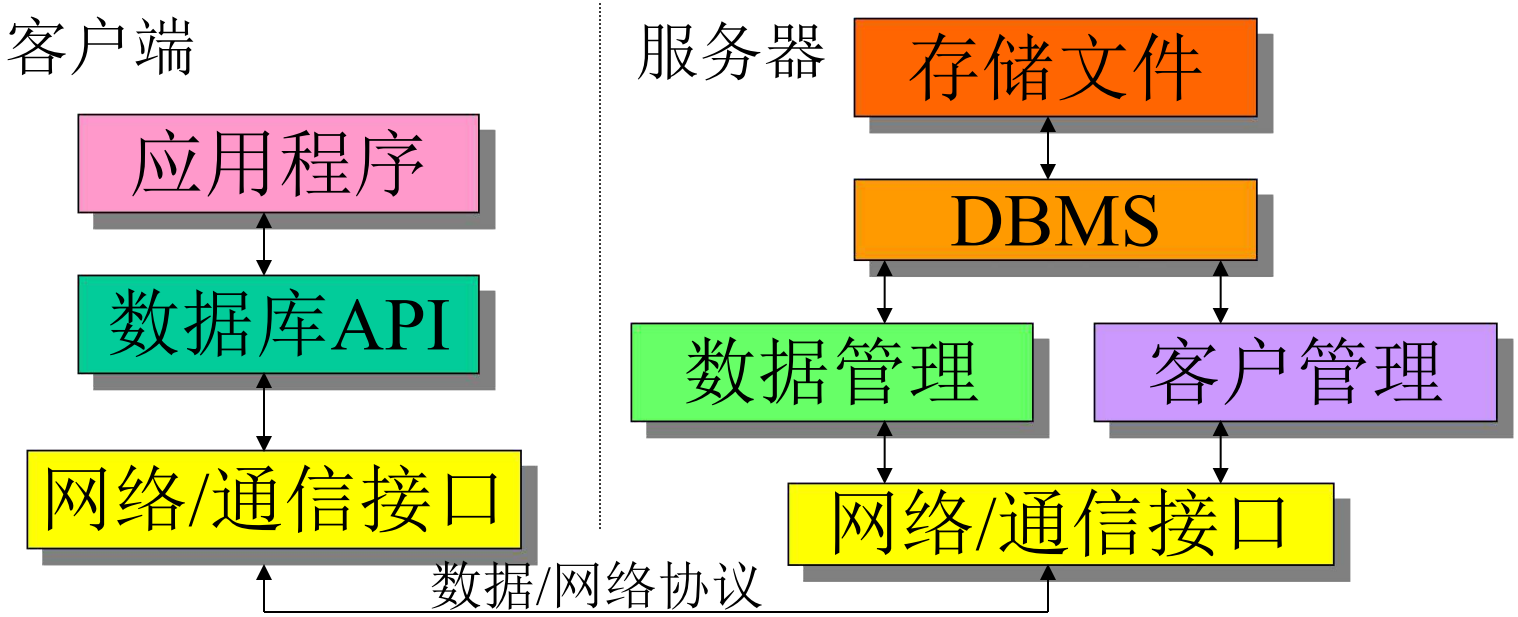


3.3.2客户机/服务器结构

- 基于三层模型的应用系统：
  - 应用逻辑层：最普遍的处理方案是使用存储过程和触发器。从客户机角度看存储过程是一个单独的事件，可以在服务器上执行复杂的操作，事务的完成或取消最终控制权交给用户；使用存储过程可避免通过网络传送SQL语句，也不必将数据通过网络传回来进行处理。
  - 数据层：定义正确的数据表和选择合适的存储方案，优化数据库的性能，使数据库快速、可靠、准确的响应用户的请求。如建立索引，优化设置。
  - 表示层：表示层主要承担人机界面的任务
- 应用系统的性能优化设计是一项系统的工作，**三层模型为系统的开放性扩充和性能优化提供了很大的空间和灵活性**，但单纯的三层结构并不一定是最优的。

3.3.3 C/S结构的数据库管理系统

- C/S数据库管理系统通常通过高性能的锁定、事务提交、参照完整性、存储过程和触发器等数据管理技术实现多用户下良好的数据完整性和并发控制。
- C/S结构数据库系统的分层模型



3.3.3 C/S结构的数据库管理系统

- 客户端网络接口：是在客户端**负责客户应用与数据库服务器通信**，将数据信息编码或解码,提供发送/接受的逻辑通道。网络接口层协议的目的是为了减少网络传送的字节数目，它建立在基础网络协议(TCP/IP、IPX、SAN等)之上。

  编码协议主要有两种:通信与远程过程调用(Remote Procedure Call，RPC)。通信是指前端应用与DBMS之间的信息传输采用的报文形式，编码协议有: TABULAR (Microsoft)、SQL*Net (Oracle)、DRDA(IBM)。远程过程调用是一种由客户制定而由服务器来执行的函数调用。
- 客户端数据管理层：用于客户应用与服务器数据库进行交互时所必需的登录管理和数据库连接以及提交语句、返回结果和处理错误。**客户端数据管理层实际上就是开发人员或应用程序使用的数据库API。**如DB_Library(Microsoft、Sybase)、OCI(Oracle)和SQLRuntime (IBM).

3.3.3 C/S结构的数据库管理系统

- 服务器端的客户管理：客户管理程序的作用是协调客户端的通信要求，为每个客户联系建立合法的数据通信渠道，为在网络上传送而使信息格式化，管理多种客户请求线程。
- 服务器端的数据管理：**数据管理程序处理每个客户的数据服务请求**，验证和解析SQL请求，优化数据操作，生成存取计划，建立不同形式的锁控制并发，对数据库中数据的访问(存取计划)。

• 客户对数据库的访问可以理解为：服务器端正确解析客户发出的SQL请求，并作出回应或发回出错信息。

3.3.3 C/S结构的数据库管理系统

- 基于不同DBMS的差异和对不同SQL语言版本的支持，客户应用必须考虑采用何种数据库访问策略：**选择开发客户应用的开发环境和选择适当的应用程序接口(API)**。
- **所有的客户机服务器数据库厂商都有自己支持的程序设计工具箱**，他们可以独立建立定制的客户应用。特定于专门数据库的设计工具，提供了对本数据库最完美的支持，但可能忽略对其他数据库的全力支持，选择第三方厂商的开发应用集成环境，是对该空隙的有力补充，他们往往注意数据库共性的同时，通过各种技术为异质数据库提供不同支持，如ODBC。

3.3.3 C/S结构的数据库管理系统

- 为了使客户应用能同时访问多种数据库，（既包括网络数据库，又包括单机数据库），应用开发环境所采用的技术往往有两种途径：
  - 支持诸如ODBC这样的公共数据库接口API
  - 专门设计同时驱动多种安装在客户端的数据库驱动程序管理层。
- 许多应用开发集成环境同时采用这两种途径，以便提高产品的通用性。如Delphi的BDE（Borland Database Engine）。

3.3.4 C/S结构的数据库系统

- **C/S系统的并发控制**：C/S系统必须提供封锁机制，事务编程要尽可能避免死锁，并允许死锁的发生及提供解决死锁的方案，保证并发执行的同时维护数据的一致。封锁机制可以由DBMS自动控制，也可以通过显式加锁完成。
- **C/S系统的完整性约束**：在C/S系统中，数据完整性约束是在服务器上定义，并由服务器来检查约束，这样能方便地实现对数据库的完整性和一致性控制。如果由客户机检查约束，检查逻辑必须包含在每一个应用程序中，既浪费又容易出错。

3.3.4 C/S结构的数据库系统

- **C/S系统的安全性控制**：DBMS通常运行在后台服务器上，自身已具备安全管理功能。应用程序运行在前端客户机上，安全性问题由开发者自己设计。如何将后台DBMS的安全机制与前端应用程序的安全机制有机结合起来，形成统一的安全保密机制。可以选择以下几种方案：
  - **内核级透明代理**：每个数据库应用只建立一个真正的数据库帐号（Root），它具有对系统应用所涉及的数据库实体进行操作的全部权限。为每一个系统操作人员分别创建一个‘应用系统帐号’，放在数据库中的Users表中。每次应用程序在客户端执行时，首先以Root登录数据库，然后执行登录程序，与Users表结合，实现应用系统登录。Users表中的内容需加密保存，数据的加密和解密通过应用程序完成。

3.3.4 C/S结构的数据库系统

- **用户授权机制**：后台服务器系统除了操作系统具有严格的用户等级机制外，DBMS也具有严格的用户授权管理机制。前端应用程序的安全机制与DBMS的安全机制统一起来，可以增强安全保密功能。

  具体做法：从功能出发将整个系统细分为若干个可分配的最小权限单元，这些权限即对数据库中所涉及的表、视图的增删改查。然后运用角色或工作组的概念，结合各种系统使用人员的工作性质，为系统创建各种操作等级，并为每个等级相应地授予不同的权限。用户等级及每种等级所对应的默认权限组合建立对照字典，管理员可以方便地增加等级或改变某一等级的默认权限。

  在统一管理下，既方便又可防止用户绕过应用逻辑直接操作数据库的可能。

3.3.4 C/S结构的数据库系统

- **智能型日志**：DBMS的日志系统是为了保障事务故障、系统故障和介质故障的恢复。智能型日志是数据库应用系统设计的为了跟踪系统使用情况的记录，相当于飞机的‘黑匣子’。在系统中，智能型日志将记录：自某用户登录时起，到其退出系统时止，这段时间中执行的所有操作，包括登录失败操作，具体内容有执行某操作的用户名、执行操作的计算机IP地址、操作类型、操作对象、执行时间等。一旦系统发生故障或受到非授权用户的恶意攻击，通过查询日志系统的记载可追溯到事件发生的所有过程。另外，日志系统还可以使系统管理员分类检索日志内容，通过智能推理日志内容，寻找系统中可能存在的不安全因素。如对同一帐号的连续三次登录失败，系统自动冻结该帐号或封锁工作站。

3.3.4 C/S结构的数据库系统

- **备份及恢复机制**：数据库管理系统提供了事务故障、系统故障的恢复例程，介质故障的恢复要靠DBA来进行。DBMS提供的数据恢复机制并不能满足所有的应用情形。

  为了防止存储设备的异常损坏，可以采用可热插拔的磁盘容错阵列或双机热备份等技术。为了防止人为的失误或破坏，可**建立强大的数据库触发器以备份重要数据的更新操作**，对删除操作，将被操作的记录全部存储在备份库中，对更新操作，可以备份执行过的SQL语句等等。保证在任何情况下，重要数据均能有效地得到恢复。

  只有将系统日志与备份数据有机地结合在一起，才能实现系统安全的‘万无一失’。

3.3.5 浏览器/服务器模式

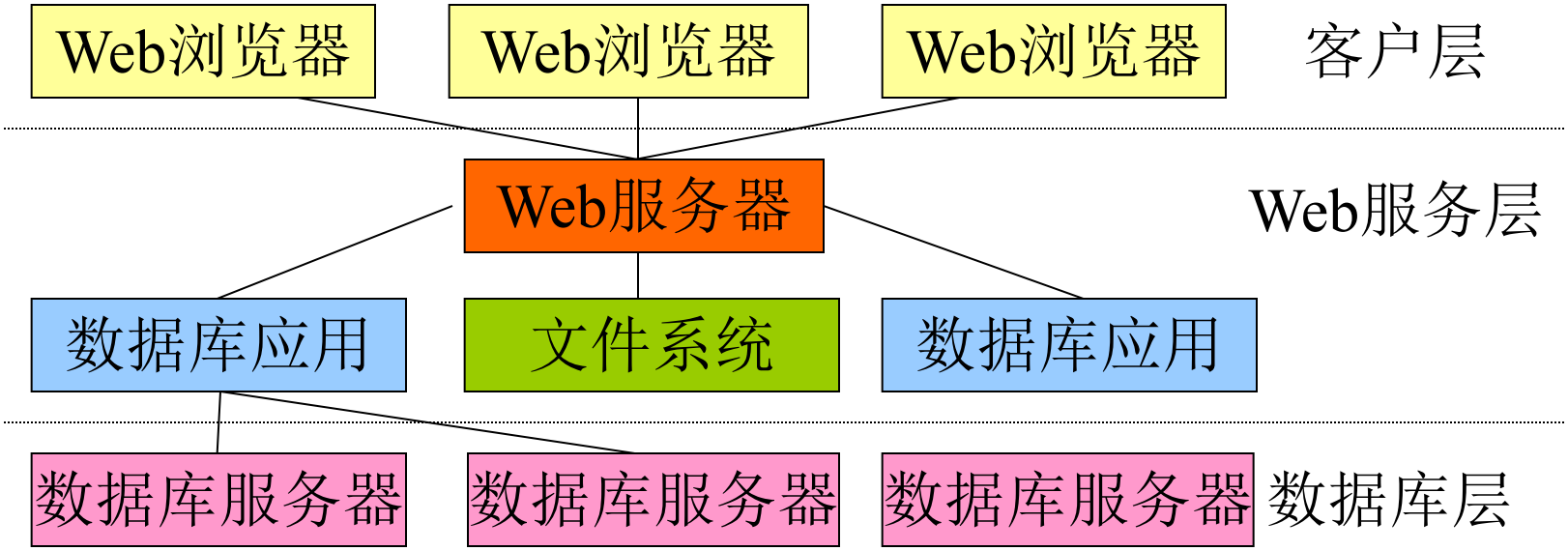
- 随着Internet和Web技术的发展,大量的商业活动在Internet展开。在典型的C/S体系中,为Internet中的每一个用户安装一前端应用程序已不再现实，在自己的Web站点上同时保存某一应用程序的多个特定平台的版本也不可行。
- 客户端安装某种具有一定‘翻译’功能的统一的构件，充当用户与Internet之间的一个接口，同时是大部分应用程序运行的主要环境外壳，这一构件即浏览器。对于浏览器而言,程序和数据的具体位置并不重要，只需知道它们的URL(通用资源定位符)。

3.3.5 浏览器/服务器模式

- 基于浏览器/服务器计算模式的系统应运而生，它继承了C/S模式的优点，十分容易地实现了不同网络间的连接。
- 目前应用广泛的浏览器/服务器计算模式主要是浏览器以超文本的形式向Web服务器提出访问数据库的请求，Web服务器接到客户请求后，激活对应的CGI程序将超文本HTML语言转化为SQL语法，将这个请求交给数据库，数据库服务器得到请求后，验证其合法性，并进行数据处理，然后将处理结果集返回CGI程序。CGI再将结果集转化为HTML，并由Web服务器转发给请求方的浏览器。

3.3.5 浏览器/服务器模式

- 在浏览器/服务器模式中，客户端的标准配置是浏览器，业务处理功能处理从C/S代码中分离出来，由独立的应用服务器处理，Web服务器成为应用处理的标准配置，数据处理仍然由数据库服务器处理. 浏览器/服务器模式是三层分布结构，即浏览器-Web服务器-数据库服务器.



3.3.5 浏览器/服务器模式

- 浏览器/服务器的三种工作方式:
  - 简单式：Web浏览器需要一个HTML页面时就提交一个URL地址到Web服务器，Web服务器从Internet上检索到所需的本地或远程的网页,并将页面返回到浏览器。也可使用Java Applet、ActiveX和Java Bean来加强表达。

    该模型只限于使用HTTP协议进行通信。
  - 交互式：在打开与服务器连接及传输数据以前，HTML获取用户输入的表单、文本域、按钮，通过这些内容取得与用户的交互。HTTP服务器将输入信息传递服务器程序或某个脚本进行处理，Web服务器再从DBMS服务器中检索数据，然后返回浏览器，最后中断浏览器和服务器连接。

    该模型已经是三层结构，但每一个浏览器和服务器间的通信都要建立一个连接，造价昂贵。

3.3.5 浏览器/服务器模式

- 分布式：客户程序是由可下载的Java编写，当HTTP服务器将含有Java小应用程序(Java Applet)的页面下载到浏览器时，小应用程序在浏览器中运行并通过构件与传输服务器上的小服务程序(Servlet)通信会话，小服务程序收到信息后，经过JDBC、ODBC或本地方法向数据库服务器发出请求，数据库服务器接到命令后，再将结果传给Servlet,最后送至浏览器。

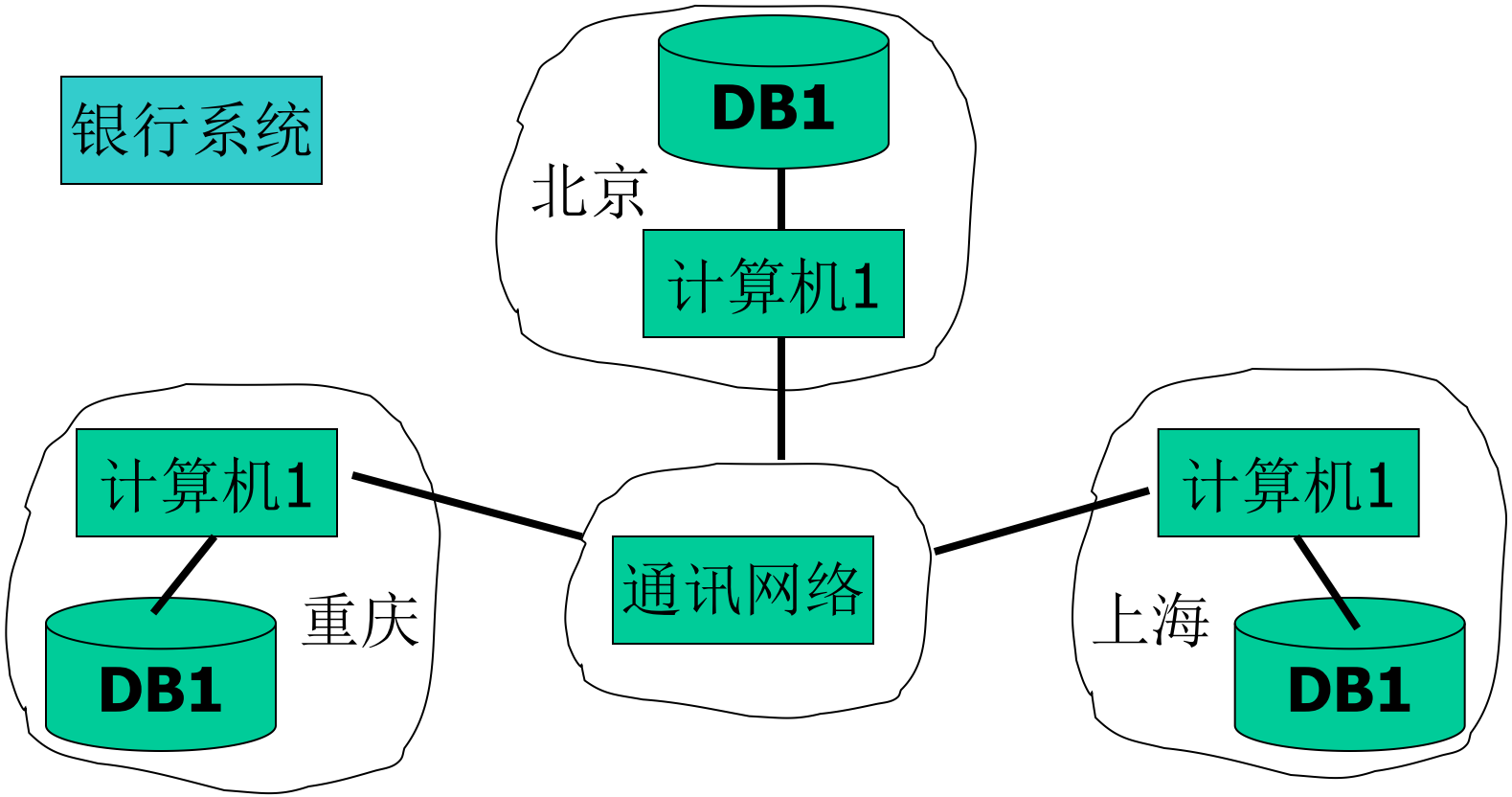
  这种方式引入了分布式计算构件概念，使得函数也可以在网络上传递。这样，在网络环境下，不同电脑、不同操作系统之间的应用程序就可以彼此沟通，达到“平台无关”。

  该模型已不再局限于Web服务器和HTTP协议，还有DCOM，CORBA协议等。

- 浏览器/服务器模式引发了另外一个专门的技术领域Web数据库。

3.3.6 分布式数据库体系结构

- 背景：数据库系统+计算机网络



3.3.6 分布式数据库体系结构

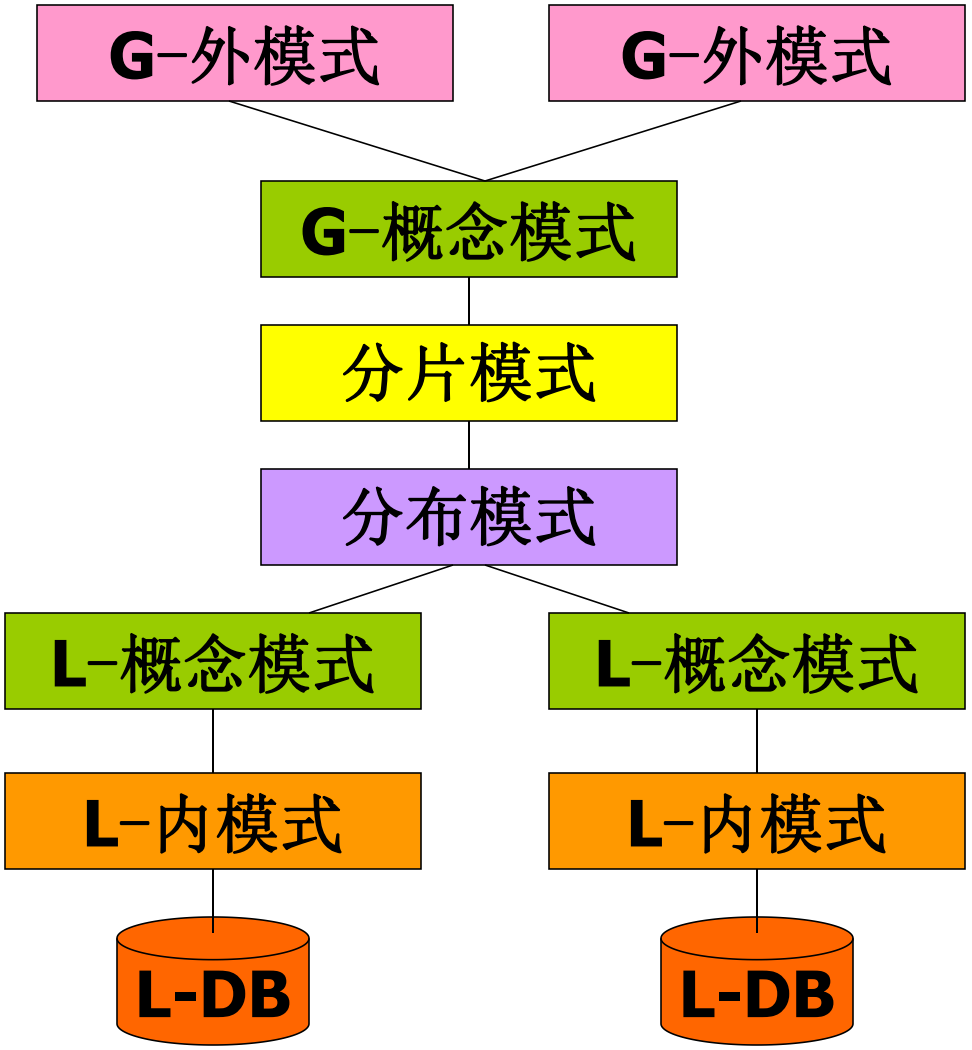
- 基本特性
  - 分布性：数据存储在不同场地上。与集中式数据库不同。
  - 逻辑整体性：数据逻辑上是相互联系的一个整体。与分散在计算机网络不同站点上的一组没有相互联系的本地数据库区别开来。
- 定义
  - DDBS(Distributed Data Base)是一个数据集合，这些数据，分布在计算机网络的不同计算机上，网络中每个结点具有独立处理的能力，可以执行局部应用，同时每个结点也能通过网络通讯支持全局应用。

    分布式数据库强调场地自治性（局部应用）以及自治场地之间的协作性（全局应用）。

    “在自治的结点之间协同工作”

3.3.6 分布式数据库体系结构

- 体系结构
  - G-概念模式：定义DDBS中数据的整体逻辑结构，数据如同没有分布一样。
  - 分片模式：每一个关系可以分为若干互不相交的部分，每一部分称为一个片段。
  - 分布模式：定义片段的存放地点。



3.3.6 分布式数据库体系结构

- 数据在分布式数据库的存储途径
  - 复制

    系统维护关系的几个完全相同的副本，这些副本存储在不同的结点上。
  - 分片

    关系被划分为几个片段，各个片段存储在不同的结点上。分片方式
    - 水平分片：将关系r按行分为若干子集r_1，r_2，…，r_n，每个子集r_i称为一个水平片段。一个水平片段可以看成是关系上的一个选择。
    - 垂直分片：将关系r按列分为若干属性子集r_1，r_2，…，r_n，每个子集r_i称为一个垂直片段。一个垂直片段可以看成是关系上的一个投影。

3.3.6 分布式数据库体系结构

  - 导出分片：导出水平分片，分片的条件不是关系本身属性条件，而是其它关系的属性条件。
  - 混合分片：关系按某种方式分片后，得到的片段再按另一种方式继续分片。
- 复制+分片

  关系被划分为几个片段，系统为每个片段维护几个副本。

- 数据项的命名

  每个数据项（关系、副本、片段）必须有唯一的名字，在分布式数据库系统中必须保证在不同的结点上不会用同一个名字来代表不同的数据项。

3.3.6 分布式数据库体系结构

- 分布（网络）透明性
  - 分片透明性：用户或应用程序只对全局关系进行操作而不必考虑关系的分片。如果分片模式改变了，通过调整全局模式与分片模式之间的映象关系来保持全局模式不变。
  - 位置透明性：用户或应用程序不必了解片段的存储位置。
  - 局部数据模型透明性：用户或应用程序不必了解局部场地上使用的是哪种数据模型。
- 分布式数据库系统(DDBS)的组成:全局用户、局部用户；全局数据库(GDB)、局部数据库(LDB)；全局数据库管理系统(GDBMS)、局部数据库管理系统(LDBMS)；全局数据分配(GDD)、局部数据分配(LDD)。

3.4 中间件

- 在分布式计算环境中，有两层的C/S模式、三层的C/S模式、三层的B/S模式以及多层的计算模式，不同层次之间需要相互通信。
- 在两层系统中，关系数据库开发商提供一些专用库和驱动程序来访问数据库。由这些专用程序对不同网络协议进行抽象，在写客户应用程序时无需考虑数据库的实际位置，客户层应用程序可以链接适当的动态或静态库与数据层进行通信，这些动态或静态库保证了请求和数据格式的一致、访问优化、分布式服务管理，称为中间件(如ODBC)。

3.4 中间件

- 多层系统中需要更复杂的关系实现跨网络、跨层次的通信，客户应用程序需要一种通信模型，来协调应用程序的不同平台。当这种模型实现后，就出现了为这样的客户应用程序提供中间件基础的标准。多层系统中中间件是建立在一种通信模型基础之上。
- 中间件定义：中间件(Middleware)是分布式环境中保证操作系统、通信协议、数据库之间进行对话、互操作的软件系统。

  在C/S环境中，中间件同时存在于客户端和服务器端，负责客户和服务器间的请求和回答。在分布式环境中，中间件允许不同服务器之间相互通信。

3.4 中间件

- 中间件作用：保证网络中各部件之间透明地连接，即隐藏网络部件的异构性，保证不同网络、不同DBMS和某些访问语言的透明性。
  - 网络透明性：能支持所有类型网络。
  - 服务器透明性：不管服务器的DBMS是何种型号，一个好的中间件都能通过SQL语言连接起来。
  - 语言透明性：客户机可用任何语言进行请求和接受回答，并保证数据类型的相互转换。

3.4 中间件

- 由于用途不同，存在不同种类的中间件，主要有数据访问中间件、远程过程调用中间件、分布式事务处理中间件、对象调用以及面向消息的中间件等。
- 不同种类的中间件基于不同的通信模型。

为了满足不同客户实现（编程语言）的需要，中间件技术在近10年中得到极大发展。第一个获得广泛认可的中间件技术是远程调用中间件(RPC)。使用远程过程调用，客户可以在远程计算机上执行C语言函数。对于远程过程调用体系结构来说，ONCRPC(Open Network Computing RPC)和DCE(Open Group's Distributed Coputing Environment)是主流标准。

3.4 中间件

- 随着面向对象语言的流行，分布式中间件也朝面向对象方向发展，OMG(对象管理组织)提出了CORBA(Common Object Request Broker Architecture)，微软提出了COM(Component Object Model)，都是规范分布式对象体系结构的尝试。许多开发商推出了基于这些规范的中间件。

  IBM早在20世纪90年代初就提出了系统对象模型SOM(System Object Model)。为了避开与微软竞争，IBM也转向了CORBA。

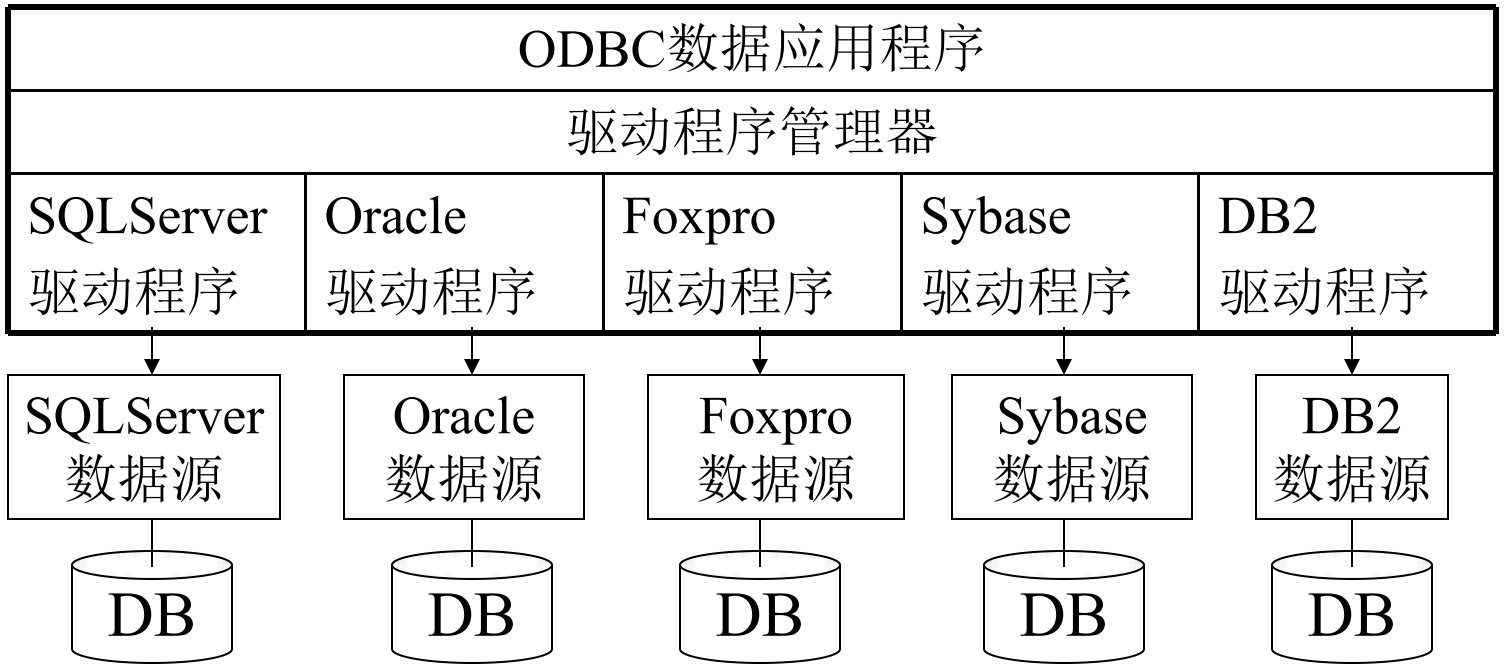
3.4 中间件

- ODBC (Open Database Connect)它是由Microsoft公司于1991 年提出的一个**用于访问数据库**的统一界面标准，是应用程序和数据库系统之间的中间件。它通过使用相应应用平台上和所需数据库对应的驱动程序与应用程序的交互来实现对数据库的操作，避免了在应用程序中直接调用与数据库相关的操作，从而提供了数据库的独立性。

  传统的数据库编程方式是‘主语言+DML’。由于各厂商的SQL版本不同，不同厂商的DBMS在客户机与服务器之间使用的通信协议不同，使得特定的前端应用不能访问不同的数据库。因此有必要建立一个公共的、与数据库无关的应用程序设计接口(Application Programming Interface, API)。最先推出的公共API是微软的ODBC。它以C/S结构为设计基础。

3.4 中间件

- 使用ODBC开发数据库应用程序时,在应用程序中调用标准的ODBC函数和SQL语句,通过加载的驱动程序将逻辑结构映射到具体的DBMS,即连接数据库和存取数据库的底层操作由驱动程序驱动各个数据库完成.



3.4 中间件

- 驱动程序管理器是一个动态连接库(DLL)，用于连接各种DBS的DBMS驱动程序，管理应用程序和DBMS驱动程序之间的交互作用。驱动程序也是一个动态连接库(DLL)，当应用程序调用SQLConnect时，驱动程序管理器加载驱动程序。
- ODBC规范定义的驱动程序有两种类型：单层驱动程序和多层驱动程序。
  - 单层驱动程序：单层驱动程序不仅要处理ODBC函数调用，还要解释执行SQL语句，执行DBMS功能。单层驱动程序本身是一个数据库引擎，尽管该数据库可能位于网络的任何地方，但由它直接可完成对数据库的操作，Xbase DBS的驱动就属于这种。应用程序把存放数据库的服务器作为文件服务器使用，在网络中传输的是整个数据库文件。

3.4 中间件

- 多层驱动程序：多层驱动程序只处理应用程序的ODBC函数调用和数据转换，它将SQL语句传递给数据源服务器，由DBMS解释执行SQL语句。SQL Server等数据库的DBS就属于这种。使用多层驱动程序，在数据库服务器上实现对数据库的各种操作，在网络中只传输用户请求和数据库处理结果。

- 应用程序的主要功能是:调用ODBC函数，递交SQL语句给DBMS，检索出结果并进行处理。应用程序的操作可包括：连接数据库；向数据库发送SQL语句；为SQL语句执行结果分配存储空间，定义所读取数据的格式；处理错误；向用户提交处理结果；请求事务的提交和回退操作；断开与数据源的连接。

作业：

1. 一个典型的数据库应用程序有哪四部分组成?
2. 客户机和服务器的任务各是什么？
3. 从C/S的结构看,其发展趋势如何?
4. 举例说明C/S系统开发中的并发控制、安全控制等管理问题。
5. 分布式数据库系统的优点缺点是什么?
6. 结合实际谈谈B/S模式中的中间件。

4. ORACLE数据库

- Oracle公司1977年在加利福尼亚的Redwood成立，在IBM的System/R(关系模型)的基础上，推出了第一个使用IBM的结构化查询语言SQL的RDBMS。今天，Oracle RDBMS可用于几乎所有的操作环境，包括IBM大型机、DEC VAX小型机、基于UNIX系统的小型机、Windows NT以及一些专用硬件操作系统平台；该公司是世界上最大的RDBMS供应商，也是世界上最主要的信息处理软件供应商，是仅次于微软公司的世界第二大软件公司,在信息高速公路中扮演着一个重要的角色。

4. ORACLE数据库

- 本章介绍ORACLE数据库管理系统。

  4.1 ORACLE数据库体系结构

  4.2 ORACLE数据库实例管理

  4.3 ORACLE数据库存取管理

  4.4 ORACLE数据库规划和实施

4.1 ORACLE数据库体系结构

- ORACLE产品结构：
  - **Oracle纵向产品**：例如 Oracle DBMS、Oracle应用服务器。另外，Oracle还提供了许多可用的开发工具，这些工具包括Designer/2000计算机辅助系统工程(CASE)工具和Developer/2000 开发包。随着INTERNET的发展，这些工具正逐渐变得基于Web。
  - **Oracle横向产品**：Oracle的应用软件为Oracle带来了大量收益。这些产品包括：

    Oracle财务软件(Oracle Financial)；

    Oracle制造业软件(Oracle Manufacturing)；

    Oracle人力资源软件(Oracle Human Resources)；

    Oracle自动控制软件(Oracle Automotive)；

    Oracle保健系统；

    Oracle基于不同客户环境的Oracle RDBMS服务器工具，包括Discoverer、Express及其他的软件。

4.1 ORACLE数据库体系结构

- ORACLE产品结构：

1.数据库：Oracle数据库服务器(对象选件、分区、空间、分布式、并行、多媒体)。

2.开发工具：

1）C/S：developer2000（对应微软VB）

2）B/S：服务器：IAS（对应微软IIS）

开发工具：PL/SQL、Portal

Jdeveloper(Java开发工具)

3）CASE：Designer(可生成60%以上的Form和Report)

4）数据仓库：Discoverer(关系数据库)

Express(多维数据库)

3.ORACLE应用：ERP:财务管理、企业内部管理

CRM:客户关系管理、企业外部管理

4.1 ORACLE数据库体系结构

- Oracle信息技术的显著特征：
  - **决策支持系统(DSS)**：以Oracle Server为基础数据仓库，使用Express及Oracle Discoverer等决策支持工具，Oracle确立了其在DSS领域的地位。
  - **海量数据管理**:Oracle对海量数据的管理非常重视，版本7.3中采用了数据分区的办法。采用数据分区后，海量数据分成很多可管理的块，当系统操作或用户会话处理查询时又能透明地将分块的数据组织起来。
  - **保密机制**:Oracle的高级保密机制通过各种各样的特权，控制对敏感数据的存取。用这些机制来保证某些用户能查看敏感数据，而有的用户被禁止。
  - **备份与恢复**:Oracle提供了高级备份和恢复的子例程。备份创建Oracle数据的一个副本，恢复把备份的数据恢复出来。Oracle的备份和恢复把数据丢失的可能性降到最小，并使出现故障时的排错时间最少。

4.1 ORACLE数据库体系结构

- Oracle信息技术的显著特征：
  - **空间管理**:Oracle提供了灵活的空间管理。用户可以为存放数据分配所需磁盘空间，也可以通过指示Oracle为以后的需求留下多少空间来控制后继的分配。
  - **开放式联接**:Oracle提供和其他软件联接的开放式接口。使用Oracle Access Manager，用户很容易就能将别的软件商开发的软件所运行的系统集成起来。
  - **开发工具**:Oracle Form和Oracle report是Oracle提供开发工具的核心。与Web相连进行发布，Oracle企业开发套件中捆绑了一些组件，使得发布灵活、操作性强、易于维护，很容易开发出不同层次的应用。套件中有如下四个主要组件：

    Oracle Designer;　　Oracle Developer;

    Oracle Developer Server;　　Oracle Application Server。

4.1 ORACLE数据库体系结构

- DBA应当了解ORACLE数据库是如何工作的和为什么要那样工作，只有这样，DBA才能自如地调整ORACLE数据库的对象，满足不同工作的需要。首先介绍ORACLE数据库的体系结构。

  4.1.1 ORACLE数据库

  4.1.2 系统数据库对象

  4.1.3 用户数据库对象

  4.1.4 数据库段

  4.1.5 数据字典

  4.1.6 其他对象

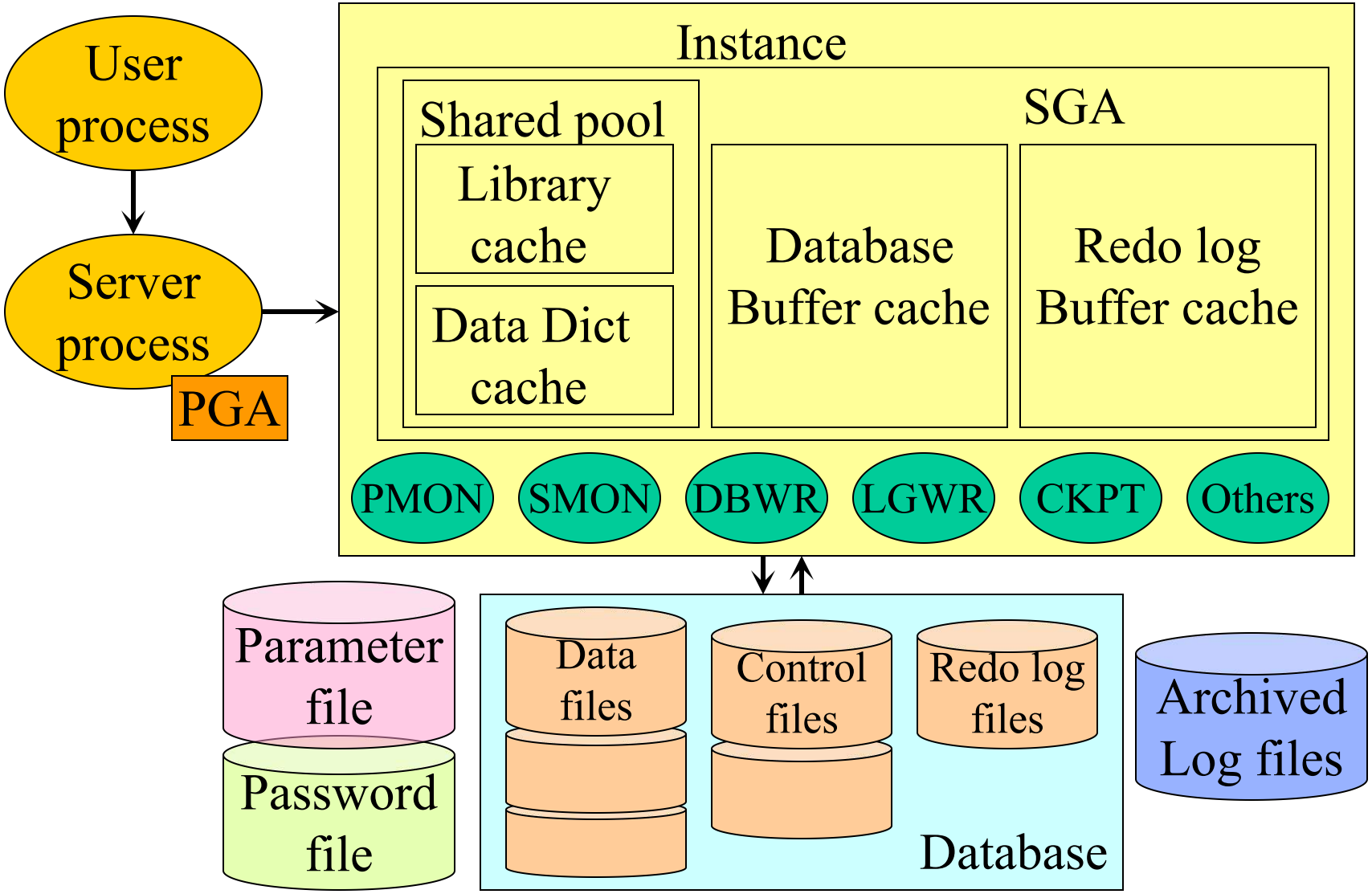
4.1.1 ORACLE数据库

- **Oracle数据库定义**：Oracle数据库为共同组成数据处理环境的配置文件、数据文件、控制文件以及重做日志文件的集合；也可以理解为处理数据文件的一批程序。Oracle数据库用来描述建立关系型数据库管理系统(RDBMS)的逻辑的和物理的数据结构。
- 从逻辑上讲，一个数据库由若干个**表空间**组成，每个表空间中有若干个表或其他数据库对象，表空间又分为数据段、索引段等**段**，每个段中又分为若干**区间**，每个数据区间由若干个**数据块**组成。
- 从物理上讲，一个数据库由若干个**物理文件**组成，物理文件有数据文件、日志文件、控制文件、初始化文件、归档文件、口令文件等。
- ORACLE数据库系统由**物理文件、内存结构、Oracle进程**三部分组成，内存结构、Oracle进程又组成**实例**。没有例程的数据是死数据。

4.1.1 ORACLE数据库

Oracle体系结构	Inatance	内存	共享池	SGA
			数据库缓冲区	
			日志缓冲区	
		进程	DBWR：数据库写入	其他进程：恢复、快照、锁等
			LGWR：日志写入	
			CKPT：检查点	
			SMON：系统进程	
			PMON：进程监控	
			ARCH：归档进程	
	Database	物理存储	数据文件	其他文件：参数文件、口令文件、归档文件
			日志文件	
			控制文件	
		逻辑存储	表空间	
			段	
			区间	
			块	

4.1.1 ORACLE数据库



4.1.2 ORACLE系统数据库对象

- **数据库对象**分两种类型：一类是由RDBMS内部使用的对象，称为系统数据库对象(system database object)，另一类是可以通过任何程序访问的对象，称为用户数据库对象(user database object)。
- **系统数据库对象**：是指RDBMS用于支持内部数据库功能的数据库对象。这些对象是由数据库系统管理员或服务器本身配置和创建的，并且不显式地用于用户数据库事务。系统数据库对象有：
  - 初始化参数文件
  - 控制文件
  - 联机和归档重做日志文件
  - 追踪文件
  - ROWID（行内部地址）
  - Oracle 块

4.1.2.1 初始化参数文件

- 初始化参数文件：也称为启动参数文件，文件缺省名为init.ora，是DBMS主要的配置点，包括定义名称的参数、设置静态限制的参数、影响动态性能的参数，控制或修改数据库和实例操作的某个方面。**当数据库启动时，在创建实例或读取控制文件之前，先读取init.ora文件**。

  init.ora文件中的值决定着数据库和实例的特性，例如共享池、高速缓存、重做日志缓存分配、后台进程的自动启动、控制文件的读取、自动联机回滚段等等。直到数据库被关闭并重新启动，对init.ora文件中参数的更改才被承认。

  init.ora是一个ASCII文本文件，默认条件下，文件的名字为init SID.ora，SID相当于它所控制的数据库的标识符。每个Oracle数据库和实例都有它自己唯一的init.ora文件。

4.1.2.1 初始化参数文件

- 初始化参数文件：

在Oracle RDBMS中，缺省的init.ora文件位于UNIX服务器上的$ORACLE_HOME/dbs目录下或NT服务器上的$ORACLE_HOME/database目录下。当创建新数据库和实例时，这个文件可被复制和重新命名。

通过查询V$PARAMETER视图，可以从数据库内部观察init.ora文件中的配置参数集。V$PARAMETER视图把所有的init.ora 参数和它们的值都列出来，并且每一个值都有一个标记符，用以指明参数值是否为服务器默认值。

通过调整初始化参数可以改善数据库系统的整体性能。

下表是init.ora参数的一些表述，详细的参数设置可参看使用手册。

4.1.2.1 初始化参数文件

参数	缺省	范围	说明
audit_trail	none	none,DB	使行写入审计跟踪接通或断开;
background_dump_dest			Oracle后台进程LGWR,DBWR所使用的追踪文件的路径;
control_files		文件名...	数据库的控制文件;
db_block_buffers	32	4..无限	包含在高速缓存中的数据库块数目;
db_block_size	2048	依赖OS	Oracle 数据库块的大小。在数据库建立起来后，这个值就不能改变了;
db_files			能够打开的数据库文件的最大数目;
db_name	空值	库名	可选择的数据库的名字。如果使用该参数，它必须与用在CREATEDATABASE语句中的数据库名称相一致;
db_file_multiblock_read_count			在顺序扫描中,一次I/O操作所能读取数据库块的最大数。这是用于顺序搜索的，是非常重要的;
dml_locks			DML封锁的最大数;
log_archive_dest			归档重做日志文件的最终位置;
log_buffer;			分配给重做日志缓冲区的字节数;
log_checkpoint_interval			触发一个检测点需要填充的重做日志文件块数;

4.1.2.2 控制文件

- 控制文件：控制文件是数据库的心脏，**存储着数据库的结构信息**，它包含以下信息：属于数据库的数据文件和重做日志文件的位置信息、数据库中的数据应该以何种字符集存储的信息、数据库中每个数据文件的状态和版本信息、检查点信息、表空间信息、回退段信息、以及其他的重要信息。

  包含在控制文件中的大部分参数是在数据库创建过程中设定的，相对来说是静态的，它们不是经常改变的。**控制文件采用二进制格式，并且是不可读或手工编辑的**。

  数据库可以操作多个控制文件。特定控制文件的创建是在init.ora参数CONTROL_FILES中指定的。

4.1.2.2 控制文件

- 控制文件：

如果没有正确的控制文件或控制文件毁坏，数据库将无法启动，并且存储在数据库中的数据信息将无法访问。正是由于这个原因，**控制文件的镜像备份功能是被Oracle服务器内部支持的，并且也是大力推荐的**。

如果要将控制文件镜像备份到一个新的数据库中，只需要在发出CREATE DATABASE命令之前，为CONTROL_FILES指定一个以上的参数值即可。

如果要将控制文件镜像备份到一个现有数据库中，必须关闭数据库，将当前的控制文件拷贝到你想要备份的目录当中，编辑init.ora中的CONTROL_FILES参数，以指定新的控制文件的位置，然后启动数据库。

注意：应至少提供两个控制文件，并存放在不同的磁盘上。

4.1.2.2 控制文件

- 改变任何控制文件参数，必须重新创建控制文件。重新创建控制文件的步骤如下：

1）备份数据库。修改控制文件过程中，出现的任何一个错误都会毁坏数据库，使之无法恢复。

2）执行ALTER DATABASE BACKUP CONTROLFILE TO TRACE命令,建立一个用户追踪文件（位于USER_DUMP_DEST），该文件带有重建当前控制文件所必须的命令。

3）编辑所产生的追踪文件。除了CREATE CONTROLFILE语句外，删除追踪文件中的所有行，设置新的参数值。

4）正常关闭（SHUTDOWN NORMAL）数据库。将旧的控制文件移放到备份目录里，确保Oracle在数据库启动时，不能在目录中找到控制文件的任何副本。

5）执行STARTUP NOMOUNT命令启动数据库，运行你所编辑的CREATE CONTROLFILE追踪文件，这将重建带有新参数值的控制文件。

6）执行ALTER DATABASE OPEN命令。

4.1.2.2 控制文件

- **在数据库创建时，把数据库参数值设为比你所需要的值高一些，可以避免重建你的控制文件**。可配置的系统控制文件参数:

  MAXLOGFILES: 联机重做日志文件的最大数;

  MAXLOGMEMBERS: 每个重做日志文件的最大成员数;

  MAXDATA FILES: 数据文件的最大数;

  MAXINSTANCES: 能够安装到这个数据库上的最大实例数（并行服务器）;

  MAXLOGHISTORY: 用于恢复实例的归档重做日志文件组的最大数（并行服务器）;

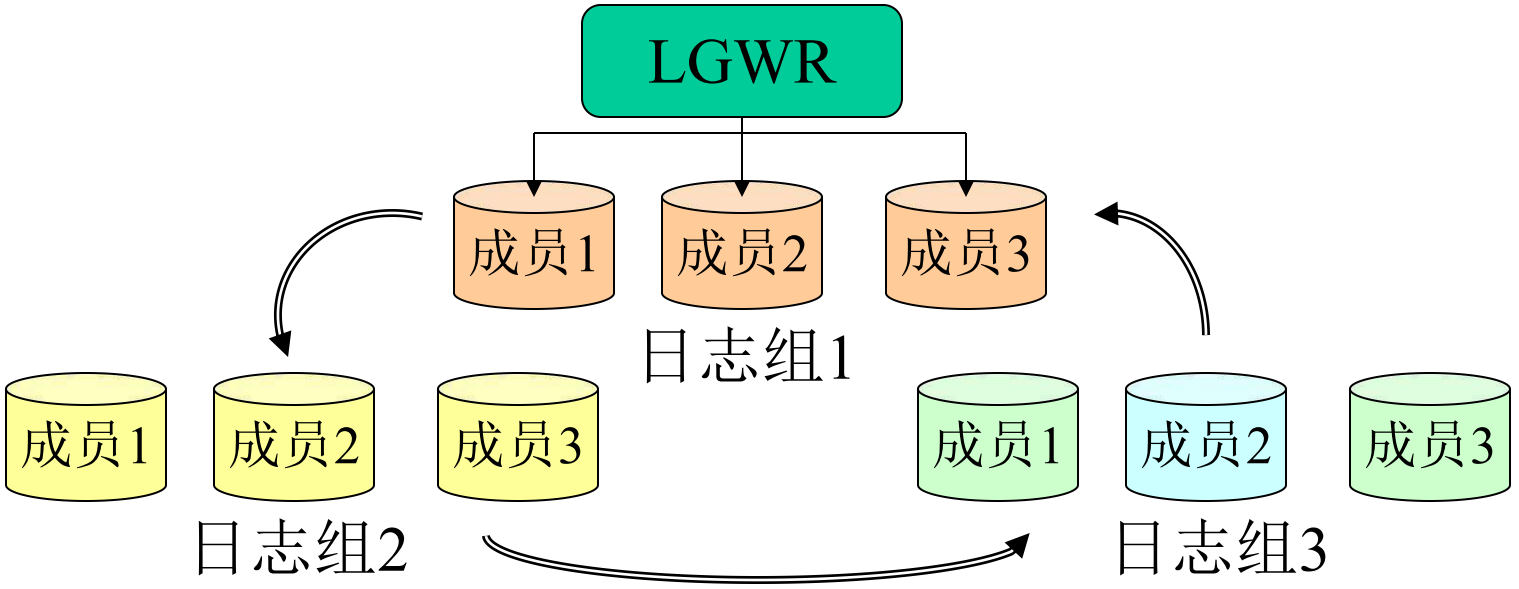
  注意要改变数据库名，要像前面所描述的那样重新创建控制文件。

  V$CONTROLFILE视图列出了Oracle服务器当前正在读写的控制文件。

4.1.2.3 联机重做日志文件

- 联机重做日志文件:日志写后台进程（LGWR）将重做日志缓冲区的内容写入联机重做日志文件中，重做日志贮存了数据库的所有变化信息，并在数据库的恢复期间被Oracle使用。联机重做日志文件为二进制文件，结构不公开。

联机重做日志文件至少由两组重做日志文件组成，并且按照一种循环的特性写入。



4.1.2.3 联机重做日志文件

- 联机重做日志文件:

当前日志组被填满时，LGWR进程停止对它的写入而转到下一个日志组。当日志切换发生时，前面写入的重做日志组被归档(ARCH)进程锁定，并被拷贝到磁盘或磁带上。

如果LGWR追上ARCH进程，并且需要实时写入由ARCH正在写入的组时，所有的数据库活动将被挂起，直到ARCH进程写完日志。**日志组不够用则影响数据库性能**。

每个日志组可由多个成员组成，**日志组的每个成员是其他成员的精确镜像**，并且重做日志项被并行地写入每个成员中。通过在每个组中使用多个成员，能防止数据库由于丢失重做日志而导致失效。只要组里有一个成员是可用的，数据库就能继续工作。

V$LOG和V$LOGFILE视图保存联机重做日志文件的信息。

4.1.2.4 追踪文件

- 追踪文件：所有的**Oracle数据库都至少有一个文件用于记录系统信息、错误及主要事件**。这个文件叫做sidALRT.log（sid是数据库系统的标识符），存储在由init.ora参数BACKGROUND_DUMP_DEST指定的位置。

  当调查数据库故障时，应首先查看该文件，关键的错误总是在这里被记载。

  后台进程和用户进程也建立它们自己的追踪文件，记载出现故障的信息。后台进程追踪文件贮存在BACKGROUND_DUMP_ DEST所指定的位置，用户进程追踪文件贮存在由USER_DUMP_ DEST参数设置所指定的目录里。

  后台进程追踪文件被命名为sidPROC.trc，PROC是后台进程的名字。用户会话追踪文件带有一个ora前缀，后面是一系列带有.trc文件扩展名的唯一标识数字。

  可以在V$PARAMETER视图中获得BACKGROUND_DUMP_DEST和USER_DIMP_DEST参数的当前设置。

4.1.2.5 ROWID

- ROWID：**Oracle数据库能够用ROWID(行内部地址)唯一识别数据库中的每一行**。

  ROWID的格式：BBBBBBBB.RRRR.FFFF，BBBBBBBB是数据文件中行所在的块号(采用16进制)。RRRR是数据行所在块中的行号(采用16进制)，FFFF是块所在的文件号(采用16进制)。

  通过查询DBA_DATA_FILES视图，可以把RDWID中的文件号和文件名相匹配。

  当每行在第一次创建时，都会被分配一个ROWID。除非该行被删除或该行所在的段被重组(通过导入/导出工具等等)，否则这个ROWID保持不变。

  在数据库中使用ROWID 是找到一行的最快方法。数据库的每个表都有一个名为ROWID 的伪列(pseudocolumn)，通过查询它可以列出表中每一行的ROWID。

4.1.2.5 ROWID

- ROWID：

因为ROWID的唯一性，它可以被用来创造性地解决许多不同的难题。

例如：下面的SQL显示一个在其中含有表的行的数据库文件数目：

```
select
count(distinct(substr(rowid,15,4)))
from 表;
```

4.1.2.6 ORACLE块

- Oracle块： Oracle块是数据库服务器能够访问的最小存储单元。Oracle块的大小是指一次I/O过程中，RDBMS从数据文件中读写的字节数、数据库对象大小和高速缓存中的块也是以Oracle块的形式设置的。

  一个Oracle块是由操作系统块组成的，但它们并不相同。**Oracle块是操作系统块的整数倍**。例如：在UNIX中，操作系统块通常为8KB，那么将db_block_size设置为8192、16384等，使它成为8KB的整数倍。

  Oracle块的大小是在数据库创建时为数据库设置的，是不能改变的。如果在创建数据库后，需要大一些的(或小些的)块，那么就必须重新建立整个数据库。

  每个Oracle块都包含头信息空间、块中数据将来的更新空间以及块中实际存储行所占空间。块头保存着诸如数据库段、同时可以有多少事务访问块等信息。

4.1.2.6 ORACLE块

- Oracle块

每个块还分配了一定量的空间，用于块中贮存行的将来更新(PCTFREE和PCTUSED存储设置来控制)。

因为数据库块决定着在每一次I/O操作中能从数据文件中读取的字节数，所以它的大小是非常重要的调整因素。**一个过大的数据库块将会把一些不必要的数据读进缓冲区，而一个太小的块则会导致行链接**。

联机事务处理（OLTP）应用有时可以从较小的块（4KB或8KB）上获益。

对于数据仓库或决策支持系统(DSS)数据库来说，较大的数据库块能够极大地提高性能。这些类型的应用在一个事务过程中，常常需要处理大量的数据，并且通常考虑系统的响应时间以分钟来计时，而不以秒来计时。

4.1.3 ORACLE用户数据库对象

- **用户数据库对象**：用户数据库对象是指那些不是Oracle RDBMS专用的对象。

  当然，用户数据库对象是受Oracle内部管理的，但是它们能够提供一组用户建造块，利用这些块用户可以创建自己的数据库。用户数据库对象包括数据文件、区间、表空间以及数据库段。

  - 数据文件
  - 区间
  - 表空间

4.1.3 ORACLE用户数据库对象

- **数据文件**：Oracle数据文件是操作系统文件。每个数据文件被分配给一个表空间，并且拥有存储在那个表空间里的实际数据。**存储在数据文件中的数据采用Oracle二进制格式**，这样除了Oracle外，不能被其他任何东西读取。

  数据文件是使用CREATE TABLESPACE或ALTER TABLESPACE创建的。一个数据文件的大小是根据CREATE语句中指定的大小来建立的，而不是由它所贮存的数据量决定的。

  访问DBA_DATA_FILES和V$DATAFILE视图，可获得数据文件中为数据库定义的有关信息。

4.1.3 ORACLE用户数据库对象

- 区间：区间(extent)是一个存储单位，由一个或多个逻辑连续的Oracle块组成，每个数据库段都是由一个或多个区间组成，在一个数据库段中的每个区间的大小可以相同或不同。

一个数据库段在对象创建时，完全按照CREATE命令的存储子句中所指定的来分配区间。当一个段无法把新数据填充到它当前分配的区间中时，必须为它分配其他的区间。下一个新区间的分配管理是数据库空间管理主要涉及的内容。

区间信息贮存在DBA_EXTENTS视图中。

4.1.3 ORACLE用户数据库对象

- 表空间：表空间(tablespace)是一个数据结构，用于组合被相似地访问的数据。**每个表空间都是由一个或多个数据文件组成的，所有的数据库对象必须被指定一个表空间，它们在那里被创建**。这样一来，组成对象的数据就被贮存到分配给指定表空间的数据文件中。

  表空间被用于分隔涉及数据访问的I/O，例如：可以创建一个表空间来存储数据对象，也可以创建另外一个表空间来存储索引对象。通过给驻留在不同物理磁盘上的表空间分配数据文件，可以确保对索引数据的访问不会影响对索引指向的数据的访问。

  在数据库备份和恢复中，表空间也扮演了一个重要的角色。因为一个表空间直接映射着一个或多个数据文件，**备份和恢复数据通常是在表空间（数据文件）级下进行的**。

  在DBA_TABLESPACES可查看表空间的信息。

4.1.4 ORACLE数据库段

- 数据库段（database segment）是贮存在数据库中的用户建立的对象。
- 除了用户创建的数据和索引段外，通常还有临时段和回滚段。尽管拥有合法权限的用户可以创建这些段,但最好由DBA创建，然后让应用用户和程序共享这些段。
  - 表:表（table）是存储数据的数据库段。每个表是由一个或多个列组成的，每个列都被指定一个名字和数据类型，每个列的数据类型为贮存在表中的数据定义了类型和精度。

    DBA_TABLES和DBA_TAB_COLUMNS包含数据库中关于表的信息。

4.1.4 ORACLE数据库段

- Oracle表的数据类型：

CHAR 固定长度的字符,尾部使用空格填充,<=255字节
VARCHAR 变长字符,<=2KB
VARCHAR2 变长字符,<=2KB
LONG 变长字符,<=2GB
NUMBER 变长数字
DATE 日期和时间,从公元前到公元后4712年12月31日
RAW 变长原始二进制数据,<=255字节
LONG RAW 变长原始二进制数据,<=2GB

Oracle建议所有的变长字符定义为VARCHAR2,而不是VARCHAR。Oracle保证永远不对VARCHAR2做出导致需要对应用程序修改才能使其向上兼容的功能性变化。因为VARCHAR的功能是由ANSI标准委员会授权管理的,Oracle不能保证这些功能在新版本里不发生巨大的变化。

4.1.4 ORACLE数据库段

- 索引:**索引(index)是为了加速对特定表数据的访问而创建的数据段**。一个索引拥有表的一列或多列的值以及与这些列值相对应的行内部地址(ROWID)。当Oracle服务器需要在表中查找某一指定行时，它在索引中查找ROWID，然后直接从表中提出数据。

  在Oracle中有几种可用的索引类型。最常用的是B-树索引。簇索引是在簇中被表共享的列的索引,不同于常规的索引，簇索引在索引中只存储一次索引键值，而不管索引键在表中重复多少次。在簇上能够执行任何数据操作语言前，必须在簇上创建簇索引。

  最新的索引类型是位映射（bitmap）索引，在基数较低（不同值数量较小）的列中，位映射索引可以比传统的B-树索引更小，更有效。

  DBA_INDEXES和DBA_IND_COLUMNS含有关于数据库中全部索引的信息。

4.1.4 ORACLE数据库段

- 回滚段:**回滚段（rollback segment）是存储在数据库事务中发生改变的原始数据块的数据库对象。它们用于提供数据的已经改变但尚未提交的读一致性视图**。

多个用户会话可以共享一个回滚段，每个回滚段至少由两个区间组成。当一个事务开始时，用户会话在一个可用回滚段中得到一个可用区间的专用锁，如果事务填满了第一个区间，它会被分配另一个区间。如果另一个区间不可用，回滚段会自动地给自己分配其他的区间，叫回滚段扩展(rollback segment extension)。

如果回滚段不能分配其他的区间（因为已经达到回滚段最大区间数，或者在回滚段表空间上没有更多的自由区间），那么就会出现错误，而且事务将被回滚。

回滚段区间的分配影响数据库的性能，因此尽可能在没有分配新区间的情况下，使所有事务能够执行。

DBA_ROLLBACK_SEGS视图中含有关于回滚段的信息。

4.1.4 ORACLE数据库段

- 表簇:表簇（table cluster）是一个数据库对象，它**可以将那些经常在相同数据块中一起使用的表进行物理分组**。当处理那些经常连接在一起进行查询的表时，表簇是特别有效的。一个表簇存储簇键以及簇表中的列值。
- 哈西簇:哈西簇（hash cluster）是数据库存储的最后选项。**在一个哈西簇中，表是基于哈西值组织的，在表的主键值上使用哈西函数可以得到这个哈西值**。在从哈西簇中提取数据时，哈西函数被用于要求的键值上，结果值给出Oracle哈西簇中的存储数据的块。

4.1.5 ORACLE数据字典

- 数据字典（data dictionary）是存储在数据库中的所有对象信息的知识库，**Oracle使用数据字典获取对象信息和安全信息，而用户和数据库系统管理员用它来查阅数据库信息**。它保存着数据库中数据库对象和段的信息，例如表、视图、索引、包以及过程，它还保存着关于如用户、权限、角色、审计和约束等的信息。**数据字典是只读的**，由四部分组成：内部RDBMS（X$） 表、数据字典表、动态性能(V$)视图和数据字典视图。
  - 内部RDBMS（X$）表:Oracle数据库的心脏就是所谓的内部RDBMS（X$）表，这些表被Oracle RDBMS用于跟踪内部数据库信息。X$表是加密命名的且几乎是无法解密的。**X$表被设计成不能被数据库系统管理员或用户直接使用**。

4.1.5 ORACLE数据字典

- 数据字典表:**数据字典表**（data dictionary table）存储表、索引、约束以及所有其他数据库结构的信息。它们属于SYS，在数据库创建时自动发生。**在数据字典视图中可以找到数据字典表中的大部分信息**。
- 动态性能视图:动态性能（V$）视图（dynamic performance（V$）view）是Oracle数据库系统管理员的主要依靠，**这些视图包含了大量数据库函数运行时的性能和统计信息**。它们还具有相当的可读性（与X$表相反），也就是说能够被数据库系统管理员用于诊断和解决问题。关于大多数V$视图的文档能够在Oracle Reference Manual中查到。

4.1.5 ORACLE数据字典

- 数据字典视图:**数据字典视图是在X$和数据字典表上创建的视图**，也就意味着它们能被终端用户和数据库系统管理员使用和查询，它们被分成三类：**DBA_、ALL_和USER_视图**。DBA_视图包含了数据库所有对象的信息。例如，DBA_TABLES包含所有已创建表的信息，ALL_视图包含了用户查询表时可以访问的所有对象的信息，USER_视图包含了用户查询表时表所拥有的全部对象的信息。

4.1.6 ORACLE其它对象

- 在数据库中还贮存着一些其他的对象，这些对象没有被准确地按段分类，它们包括视图、序列、同义词、触发器、数据库链以及存储包、过程和函数。
  - 视图(View):视图是存储的能够被查询的SQL语句。视图用于隐藏一些数据，并使复杂的查询变得易于理解和使用。通过在远程数据库表上创建视图，视图也能被用于隐藏分布式数据库对象。任何可以做为SQL查询而执行的语句，都能被创建成为一个视图。

    在设计应用程序时，视图是非常有用的，因为它们能够以表的形式隐藏复杂的查询逻辑，使得更易于查询。它们可以和嵌在它们内部的优化器提示一起被创建，以确保最佳的查询性能。

    DBA_VIEWS视图存有在数据库中创建的视图的信息。

4.1.6 ORACLE其它对象

- 序列(Sequence)：序列是用于产生唯一数码的数据库对象。序列创建时带有初始值、增量值以及最大值。每次从序列中返回一个数字，当前序列值是一个一个递增的，每个序列产生的数字可达到38位长度。

  可以通过选择来自序列的NEXTVAL或CURRVAL伪列使用序列。例如：一个名为EMP_SEQ的序列，

  执行 SELECT EMP_SEQ.NEXTVAL FROM DUAL;

  返回序列的下一个整数值，并且当前序列值加1；

  执行 SELECT EMP_SEQ.CURRVAL FROM DUAL;

  返回当前序列的整数值。

  序列最通常的用途是提供唯一的数字作为表主键列的代用品。

  有关序列的信息贮存在DBA_SEQUENCES视图中。

4.1.6 ORACLE其它对象

- **触发器(Trigger)**:触发器是存储过程，当针对一个表发生特定的动作时，就会激活它。触发器可以被编码。当针对一个表进行插入、更新、删除或三种操作的结合时，激活触发器，也可以在某行被影响或某条语句出现时被激活。触发器经常用于加强数据完整性约束和业务规则。

  关于触发器的信息可在DBA_TRIGGERS视图中找到。

- **同义词(Synonym)**:同义词是指向其他数据库表的数据库指针。**当创建一个同义词时，就指定了一个同义词名字和同义词所引用的对象**。当你引用同义词名字时，Oracle服务器会自动地用同义词定义的对象名字来代替同义词的名字。

  同义词有两种类型：私有（private）和公共（public）。私有同义词是在指定的模式中创建的，并且只允许拥有它的模式访问。公共同义词由PUBLIC模式所拥有，所有的数据库模式都可以引用它们。

4.1.6 ORACLE其它对象

- 同义词:

如果执行SQL语句“SELECT * FROM EMP;”，Oracle服务器中对象名字的解析顺序是:

1）首先，服务器查看在发出命令的用户模式中是否有名为EMP的表或视图。

2）如果这个表或视图不存在，Oracle检查名为EMP的私有同义词是否存在。

3）如果这个私有同义词存在，这个同义词所引用的对象将取代EMP。

4）如果这个私有同义词不存在，检查名为EMP的公共同义词是否存在。

5）如果这个公共同义词不存在，Oracle返回消息“ORA-00942，table or view does not exist”。

关于公共同义词的信息贮存在DBA_SYNONYMS中。

4.1.6 ORACLE其它对象

- 数据库链（database link）:数据库链是与远程数据库连接的存储定义，它们用于查询分布式数据库环境中的远程表。因为它们存贮在Oracle数据库中，它们被归入数据库对象类别。关于数据库链的详细信息可以在DBA_DB_LINKS数据字典视图中得到。
- 存储包(Package) :存储包是一个过程、变量和函数的集合，包按照功能被逻辑地分组。包是有目的地将一些过程、函数及程序块集合在一起的“大”程序。
- 过程(Precedure) :对无名的PL/SQL块，只能使用文件名的方式调用，在程序中略显不便。将程序块命名就是过程。一个存储过程是一个操作的代码单元，可以被传递参数，并能够返回值。

4.1.6 ORACLE其它对象

- 函数(Function):一个存储函数是一个代码单元，可以被传递参数，并能够返回一个值。

  存储包、过程和函数与它们的源代码一起存在数据字典中。可以通过DBA_OBJECTS和DBA_SOURCE视图查看有关存储包、过程和函数的有关信息。

- 快照(Snapshot):分布式数据库中，对于那些实时性不强而又要经济查询的信息来说，远程查询并不是良策。基于此，Oracle提供一种快照对象。快照就是在远程机器放置本地数据库信息的一个只读副本。这样，查询远程数据库就像查询本地数据库一样，缺点可能不实时，但减少了通信线路的影响。通过指定快照的刷新方式，Oracle自动完成快照的更新。

作业：

1. 简述ORACLE数据库的组成。
2. 初始化参数文件的作用是什么？
3. 如何为控制文件做镜像？
4. ORACLE日志能否替代第三章介绍的智能型日志？说明理由。
5. 试从ORACLE数据库的存储结构谈谈ORACLE的先进性、灵活性？

4.2 ORACLE数据库实例管理

- Oracle服务器就是数据库管理系统(DBMS)，由**数据库(Oracle database)**和**实例(Oracle instance)**组成。
- 实例是一系列复杂的内存结构和操作系统进程，它为Oracle客户提供所期望的不同程度的服务。一个实例只能打开一个数据库，或者说一个数据库被唯一的一个实例装载。

4.2.1 实例组成

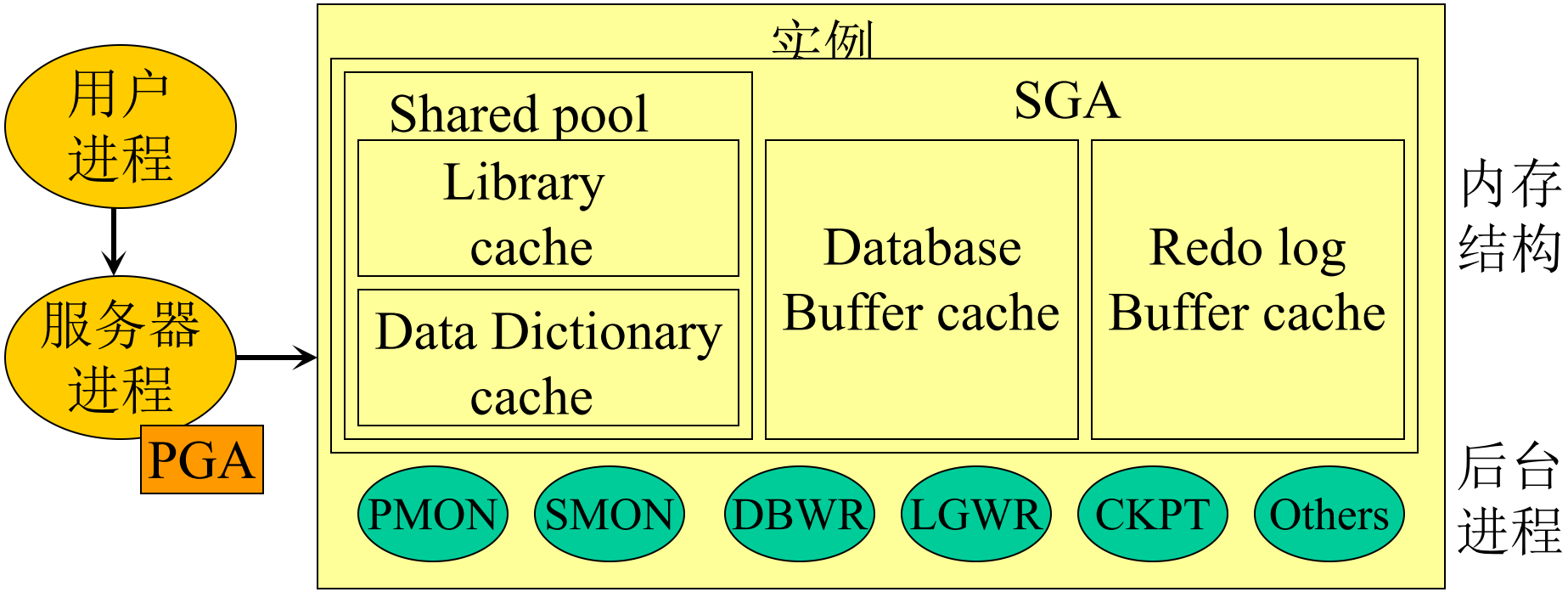
4.2.2 实例创建

4.2.3 监控实例

4.2.1 ORACLE数据库实例组成

- 实例中**每一个进程都有各自的一个内存块**，该内存块用于保存私有变量、地址堆栈和其他运行时的信息。
- **进程间使用公共共享区**并在公共共享区内完成它们的工作。公共共享区是能够在同一时间内被不同程序和不同进程读写的一块内存区。该内存块称为**系统全局区(SGA)**。因为SGA驻留在一个共享内存段中，所以它经常被称作共享全局区。
- 后台进程就像数据库的手，是直接处理数据库的组件；SGA就像大脑，在必要时间接地调度手处理它们的信息与存储检索。SGA参与发生在数据库中全部的信息和服务器的处理。
- 单用户的Oracle配置(Personal Oracle Lite)，不使用多进程执行数据库的功能。相反，所有的数据库功能由一个Oracle进程完成。由于这个原因，单用户也称为单进程Oracle。

4.2.1 ORACLE数据库实例组成



- ORACLE实例是指有自己的内存结构和相关的服务器进程及后台进程的DBMS。

4.2.1.1 内存结构

4.2.1.2 进程

4.2.1.1 ORACLE内存结构

- ORACLE的内存结构主要有两个内存区域组成：

  SGA(System Global Area):系统全局区，是ORACLE实例的基础，是一个共享区域，存放最近使用的SQL语句、最近使用的数据、最近使用的数据字典等信息，供各个进程调用。

  PGA(Program Global Area):程序全局区，该区域是一非共享区域，存放会话、排序、游标、主变量等信息，被服务器进程调用，保证谁的请求处理结果给谁，谁的排序给谁，用户声明的、打开的、使用的、关闭的游标是私有的，用户的主变量不被其他用户干扰。

4.2.1.1 ORACLE内存结构

- 系统全局区：系统全局区是实例的主要部分。它含有数据维护、SQL语句分析与重做缓存所必须的所有内存结构。
- **系统全局区的数据是共享的**，也就是说，多个进程可以在同一时间对SGA中的数据进行访问和修改。所有数据库操作都使用包含在SGA中某点上的结构。
- 当实例被创建时，分配SGA；当实例关闭时，释放SGA。

4.2.1.1 ORACLE内存结构

- SGA组成如下：
  - 共享池。
  - 数据库缓冲区高速缓存。
  - 重做日志缓冲区。

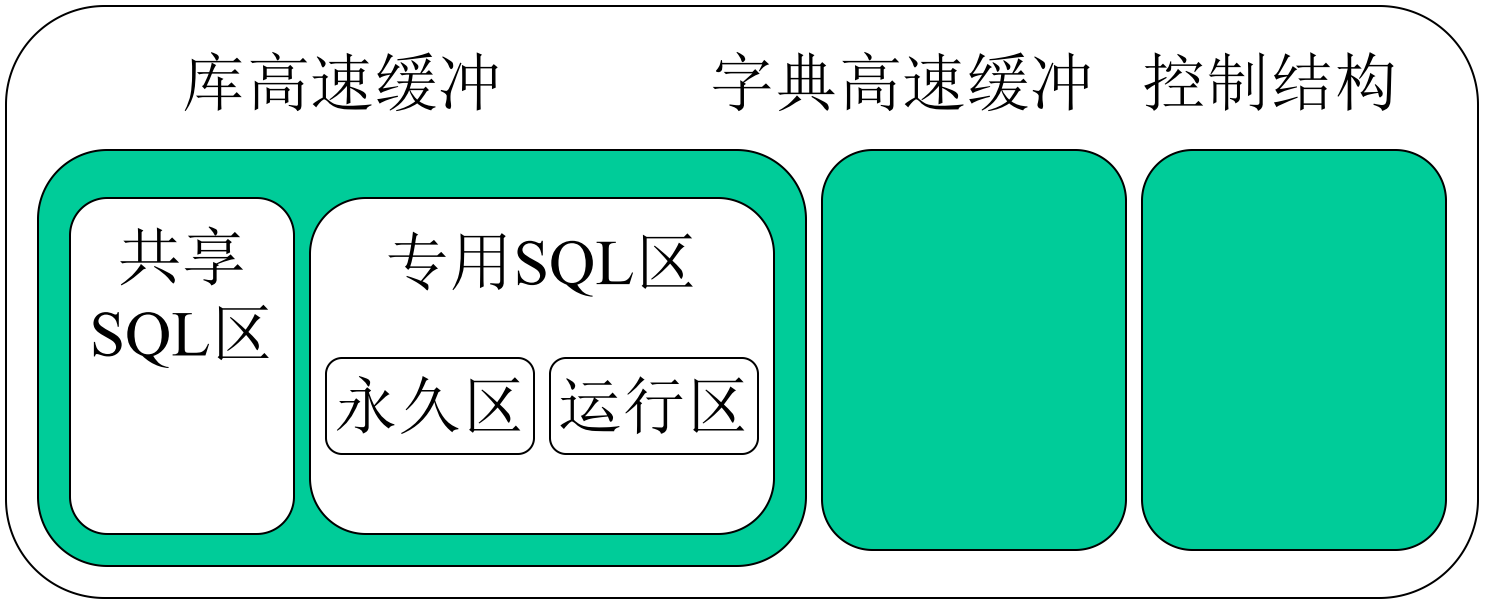
另外oracle9i还有Java pool，8i有Java虚拟机，解释Java；Java pool是利用数据库技术解释Java.

oracle9i的SGA可以通过SGA_MAX_SIZE被动态的设置大小，8i不能动态设置。

4.2.1.1 ORACLE内存结构

- 共享池：共享池包括库高速缓存、数据字典高速缓存和服务器控制结构（例如数据库字符集）。

  参见下图



4.2.1.1 ORACLE内存结构

- 共享池：

库高速缓存存储已提交给RDBMS的SQL语句文本、分析过的格式与执行计划，以及已被执行的PL/SQL包头与过程等。数据字典高速缓存存储用于分析SQL语句的数据字典行。

Oracle服务器用库高速缓存来提高执行SQL语句的性能。当一条SQL语句提交时，服务器首先查找高速缓存，查看相同的语句是否已被提交或缓存过。如果有，Oracle使用存储的语法分析树和执行路径来执行该语句，使用存储代码可以获得明显的性能提高。

4.2.1.1 ORACLE内存结构

- 共享池：

对于使用以前缓存的SQL语句，它必须在所有方面都与缓存版本完全相同，包括标点符号和字母的大小写。

库高速缓存包括共享和专用SQL区。共享SQL区包括SQL语句语法分析树和执行路径；专用SQL区存储特定的会话信息，一个用户会话能够一次打开的专用SQL区的数量由init.ora参数OPEN_CURSORS决定。

库高速缓存中的专用SQL区可更进一步分为永久区和运行区。永久区中包含合法的信息，并可应用于SQL语句的多个执行中，而运行区中仅包含正在被执行的SQL语句的数据。

4.2.1.1 ORACLE内存结构

- 共享池：

字典高速缓存含有RDBMS引擎分析SQL语句所使用的数据字典信息。在这个区域中含有段信息、安全性、访问权限和在可用的自由存储空间等信息。

共享池的大小取决于init.ora文件参数SHARED_POOL_SIZE，它是以字节为单位的。ORACLE 9i可以通过ALTER SYSTEM SET SHARED_POOL_SIZE动态设置。

共享区经过长期装卸和卸载数据对象会产生许多碎片，如果在共享池中没有足够的连续空间用来装载目标数据，会产生错误。这个问题可以通过运行SQL命令ALTER SYSTEM FLUSH SHARED_POOL解决。但是如果在数据库操作时，经常遇到共享池错误，就必须增大共享池。

4.2.1.1 ORACLE内存结构

- 数据库缓冲区高速缓存：是影响整个数据库系统运行的重要因素之一。**数据库缓冲区高速缓存是由与Oracle块相同大小的内存块组成。所有Oracle操作的数据在使用前被装入到数据库缓冲区高速缓存中。数据的更新在内存块中完成**。

  Oracle根据最近最少被使用(LRU)列表将数据清出缓冲区高速缓存。LRU列表记录数据块被访问的频繁程度。当服务器在缓冲区中需要更多空间来从磁盘读入一个数据块时，它去访问LRU列表，以确定可以清出哪些块，使用这种方法，**保证最频繁使用的块保存在内存中**。

4.2.1.1 ORACLE内存结构

- 数据库缓冲区高速缓存：

被修改过的缓冲块称为脏的，脏列表记录所有在内存中被修改而又尚未写入磁盘中的数据。当Oracle接收到修改数据的请求时，对高速缓存中的块进行数据修改，同时写入重做日志中，然后该块被放入脏的列表中，对这些数据的随后访问从高速缓存中读取改变的数据的新的值。

Oracle服务器对更新的一块数据，并不立即更新数据文件中的数据。RDBMS等到一定条件时才将改变的数据刷新写入数据文件（参见数据库写入进程）。

4.2.1.1 ORACLE内存结构

- 数据库缓冲区高速缓存:

DB_BLOCK_SIZE和DB_BLOCK_BUFFERS是决定缓冲区高速缓存大小的两个初始化参数。

DB_BLOCK_SIZE设置Oracle块大小，DB_BLOCK_BUFFERS决定分配给缓冲区高速缓存的块的数量。两参数相乘就可得出缓冲区高速缓存的内存总数(以字节为单位)。

ORACLE 9i可以通过ALTER SYSTEM SET DB_CACHE_SIZE动态设置。

4.2.1.1 ORACLE内存结构

- 重做日志缓冲区：重做日志缓冲区用于在内存中存储未被刷新写入联机重做日志文件的重做信息。

  它是循环使用的缓冲区，当重做日志缓冲区填满时，将它的内容写入联机重做日志文件。

  重做日志缓冲区的大小是由LOG_BUFFER初始化参数决定，以字节为单位，决定在内存中保留多少空间缓存重做日志项。如果这个值设置得过低，进程之间相互竞争，LGWR进程读出和写入缓存，有可能会导致性能问题。

4.2.1.1 ORACLE内存结构

- 重做日志缓冲区：

为强迫重做日志顺序写入，Oracle服务器使用闩控制对缓存的访问。闩是一个Oracle进程对一个内存结构的锁定，一个进程必须持有重做分配闩，才能写入重做日志缓冲区。当一个进程持有分配闩时，其他任何进程都不能使用这个分配闩写入重做日志缓冲区。

Oracle服务器使用LOG_SMALL_ENTRY_MAX_SIZE初始化参数限制一次写入的重做的总量。这个参数以字节为单位，其缺省值随操作系统和硬件不同而不同。

4.2.1.1 ORACLE内存结构

- 重做日志缓冲区：

对具有多个CPU的服务器而言，Oracle服务器不允许使用重做分配闩所书写的重做日志项所需空间比参数LOG_SMALL_ENTRY_MAX_SIZE大。相反，进程必须持有一个重做复制闩。可获得的重做复制闩的数量等于LOG_SIMULTANEOUS_COPIES初始化参数的值。LOG_SIMULTANEOUS_COPIES的缺省值是系统中CPU的数量。使用重做复制闩，多个进程能同时写入重做日志缓冲区。

可以使用V$LATCH动态性能视图监控重做分配闩与重做复制闩。

4.2.1.2 ORACLE进程

- ORACLE进程可以理解为一系列执行一定任务、提供不同服务信息的程序。主要有用户进程、服务器进程、后台进程三种类型。
- 用户进程：当客户机向ORACLE服务器发出一个连接请求时，就产生一个用户进程。

  数据库用户操纵数据前，首先要建立一个与数据库服务器的连接(可通过Oracle工具,如SQL*Plus)，连接的同时产生用户进程，用户进程并不直接作用于ORACLE服务器，而是与一个服务器进程进行通信。

4.2.1.2 ORACLE进程

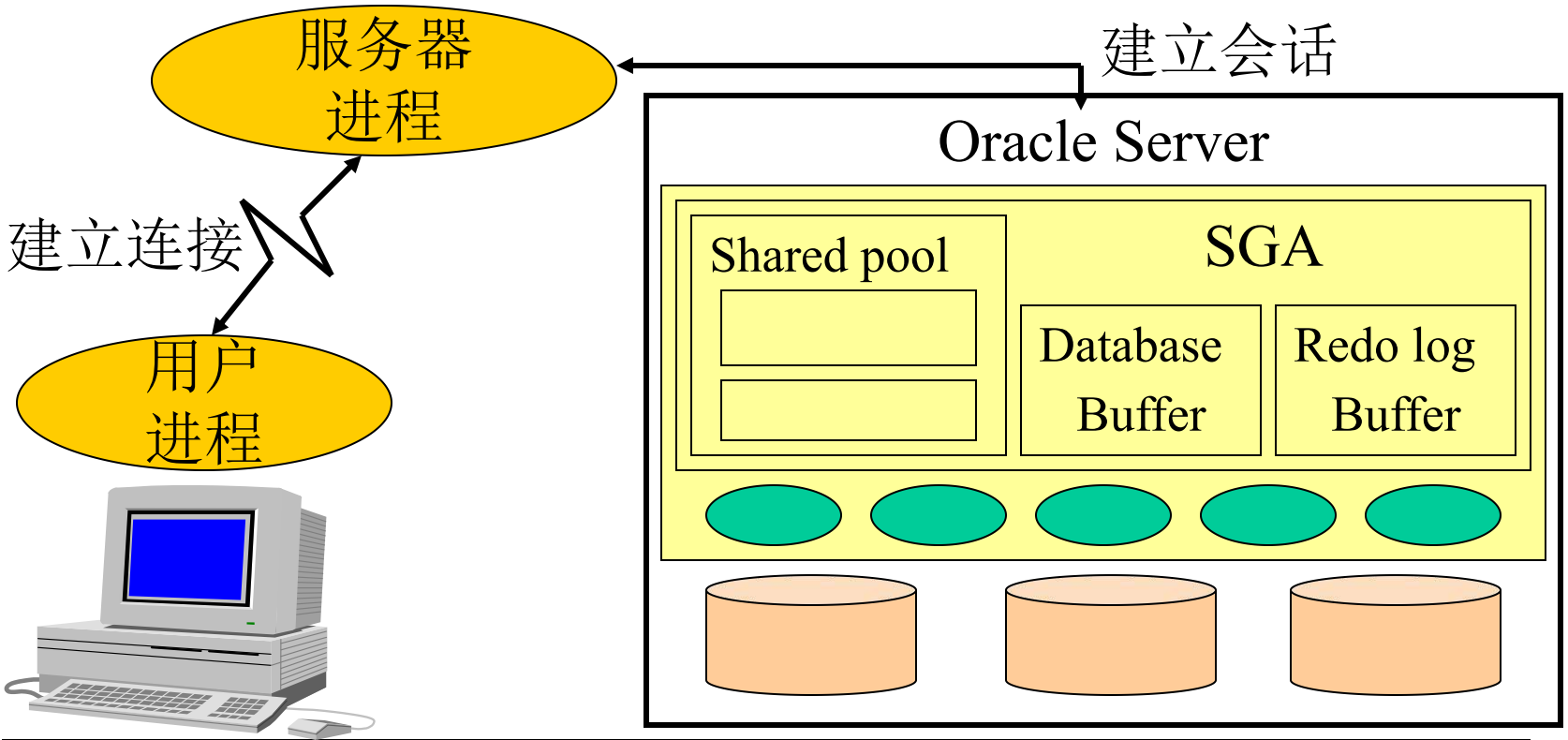
- 用户进程：**每个用户进程被分配一部分内存区，称为进程全局区(PGA)。**

  当一个用户进程通过专用服务器方式连接数据库时，用户的会话数据、堆栈空间和游标状态信息存储在进程全局区中。用户的会话数据包括安全和资源使用信息；堆栈空间含有为用户会话指定的本地变量；游标状态区包括运行时的游标信息、返回的行和游标返回的代码。

  如果用户进程通过共享服务器进程方式进行连接，游标和会话信息被存储在系统全局区(SGA)中。尽管对整个数据库而言，这并不增加对内存空间的要求，但是它需要一个更大的系统全局区来存放这些附加的会话信息。

4.2.1.2 ORACLE进程

- 服务器进程：服务器进程接受用户发出的请求，并根据请求与数据库通信，通过这些通信完成用户进程对数据库中数据的处理要求，同时完成对数据库的连接操作和I/O访问。



4.2.1.2 ORACLE进程

- Oracle后台进程：在任意瞬间，Oracle数据库可以处理许多行信息、处理几百个同步用户请求、进行复杂的数据操作，与此同时提供最高水平的性能和数据的完整性。

  为了完成这些任务，Oracle数据库将一项大的工作分散到多个程序中，其中每个程序的大部分操作都是相互独立的，并扮演一个特定的角色。这些程序称为Oracle后台进程。

4.2.1.2 ORACLE进程

- Oracle数据库的物理文件和内存结构之间的关系由后台进程来维持。
- 理解后台进程和它们担负的任务，将有助于分析性能问题、指出瓶颈和诊断数据库中的故障点。
- 数据库拥有多个后台进程，其数量取决于数据库的配置。每个后台进程创建一个跟踪文件，在实例操作期间保存跟踪文件。可以设置init.ora文件的BACKGROUND_DUMP_DEST参数来规定后台进程跟踪文件的位置。排除数据库故障时，跟踪文件就显得非常重要。

4.2.1.2 ORACLE进程

- Oracle后台进程有：
  - 系统监控（SMON）
  - 进程监控进程（PMON）
  - 数据库写进程（DBWR）
  - 日志写进程（LGWR）
  - 调度进程（Dnnn）
  - 归档进程（ARCH）
  - 检查点（CKPT）
  - 恢复进程（RECO）
  - 快照进程（SNPn）
  - 锁进程（LCKn）
  - 并行查询进程（Pnnn）

4.2.1.2 ORACLE进程

- 系统监控(System Monitor,SMON)

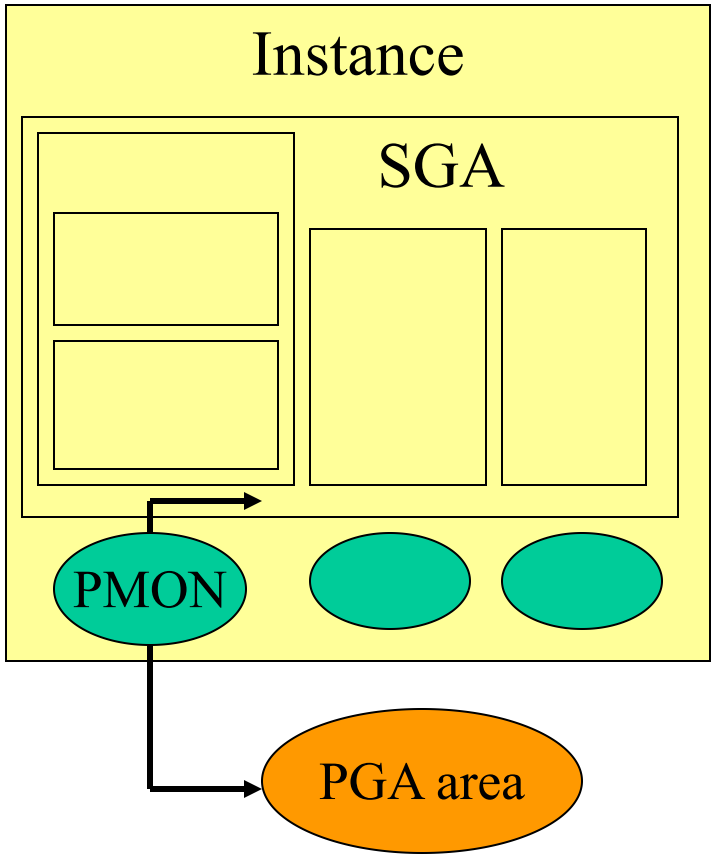
进程监控(Process Monitor,PMON)：

由于种种原因，对Oracle数据库的连接可能会发生崩溃、挂起或其他非正常终止。例如，网络断线、用户提出不合理的要求遭到系统拒绝，操作系统出错等等，都会引起连接中断或系统挂起，Oracle服务器必须有能力去处理由于各种原因而引起的失败。

系统监控和进程监控都是自动解决数据库系统问题的后台进程。

4.2.1.2 ORACLE进程

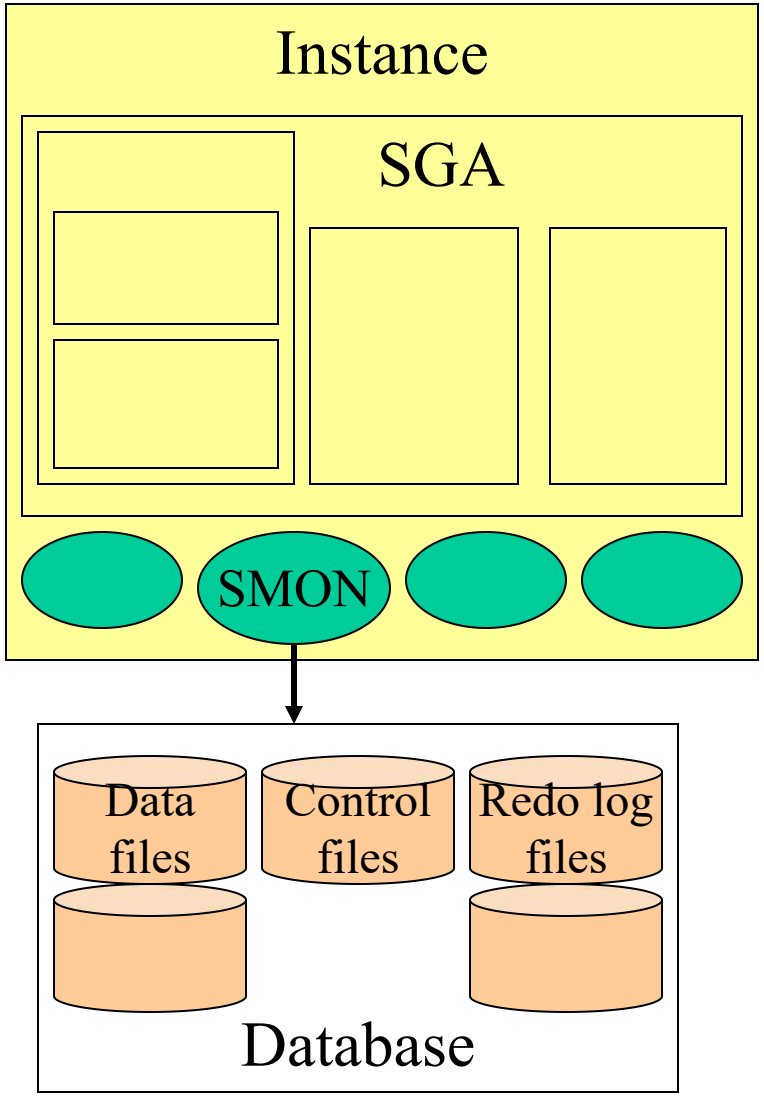
- 进程监控(PMON)自动清除中断或失败的进程，包括清除非正常中断的进程留下的孤儿会话、回滚未提交事务、释放被断开连接的进程占有的锁、释放被失败进程占有的系统全局区(SGA)资源，它同时监控服务器和调度进程，如果它们失败则自动重启它们。



4.2.1.2 ORACLE进程

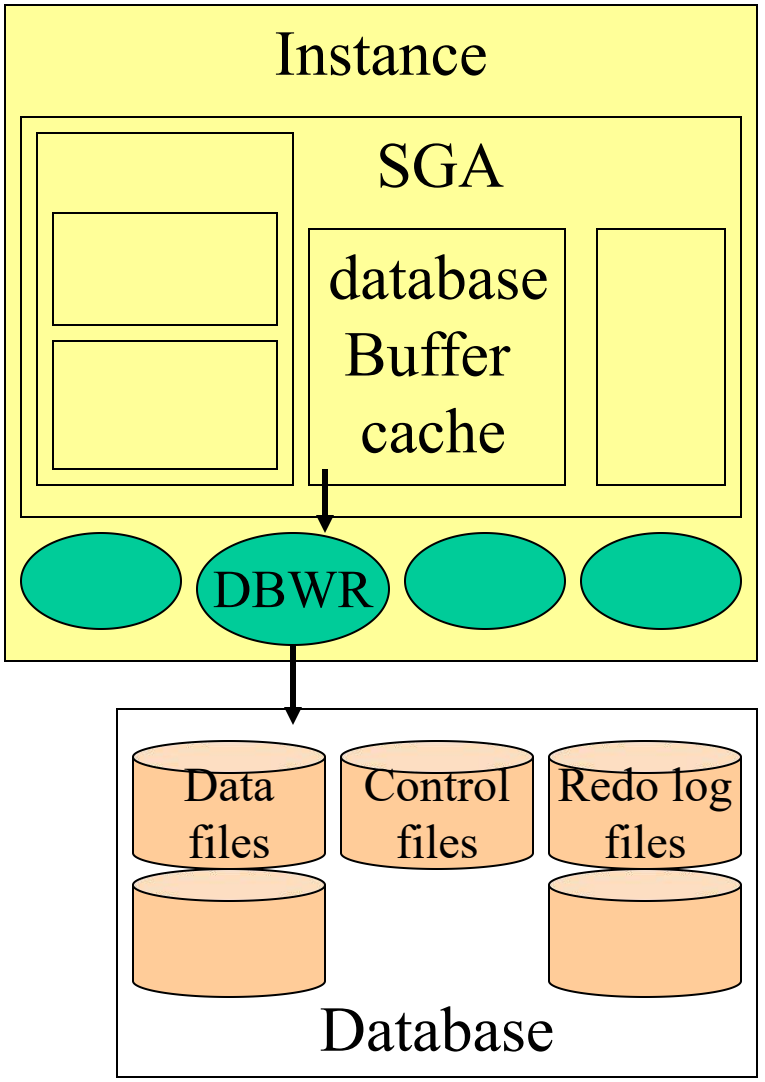
- 系统监控(SMON)作为恢复自动实例的进程，回滚尚未提交的事务或前卷重做日志。SMON也是管理某些数据库段的进程，收回不再使用的临时段空间，并自动合并在数据文件中相邻的自由空间块。

  SMON仅当创建表空间或表时的缺省存储参数pctincrease不是0时，合并表空间中的自由空间。如果想让SMON自动地处理这个操作，至少将pctincrease设置为1。



4.2.1.2 ORACLE进程

- 数据库写进程(Database Writer, DBWR)：负责将缓冲区中脏的数据块写入到数据文件中。



4.2.1.2 ORACLE进程

- 数据库写进程(Database Writer,DBWR)

DBWR等到下列情况发生时，才成批地读脏列表，并将在脏列表中发现的所有块刷新写入数据文件。

1）发生一个检查点。

2）脏列表的长度达到init.ora文件中DB_BLOCK_WRITE_BATCH参数值的一半。

3）使用的缓冲区数量达到init.ora参数DB_BLOCK_MAX_SCAN。

4）DBWR后台进程发生超时（大约每3秒）。

4.2.1.2 ORACLE进程

- 数据库写进程(Database Writer,DBWR)

在大多数安装中，有一个DBWR进程处理所有数据库的所有写入活动。但是当发现这个DBWR进程不能满足数据库要求时，可以启动一个以上的DBWR进程。

init.ora文件中DB_WRITES参数可以设置在启动时创建的DBWR进程数(缺省值是1)。建议使用与存储数据文件的物理磁盘一样多的DBWR进程或者将这个数量设为与数据库中数据文件的数量相等。

4.2.1.2 ORACLE进程

- 数据库写进程(Database Writer,DBWR)

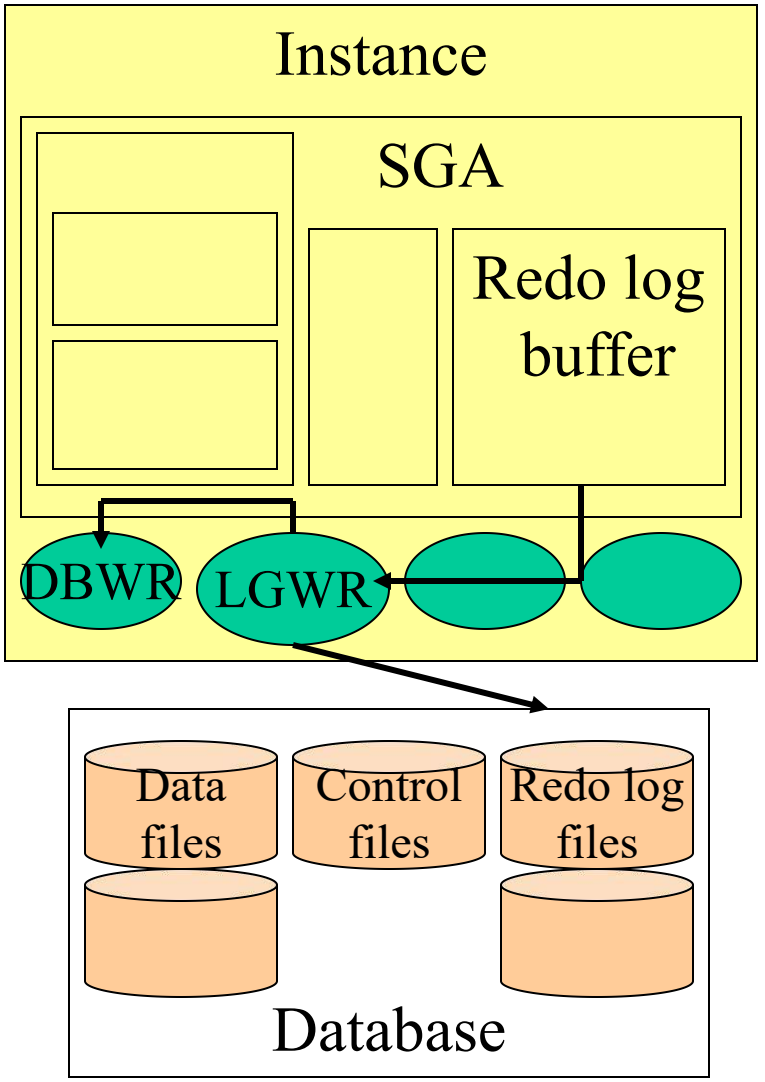
另一个参数是DB_BLOCK_CHECKPOINT_BATCH，设置在每个检查点DBWR写入的块的最大量（参看检查点进程）。增大该参数，可以减少DBWR刷新缓冲区的次数。但将这个数值增得过大，当DBWR最终刷新缓冲区时，可能产生不能接受的延迟。

第三个参数是DB_BLOCK_CHECKSUM，这是个布尔参数，启用这个参数时，导致每个数据库块被写入时附加一个校验和值。当随后读取该块时，计算校验和值并与存储在数据库中的数值进行比较，如果值不同，将产生错误。当查找数据毁坏问题时，这是一个有价值的参数，但是在全部时间都启用这个值也会有代价。

4.2.1.2 ORACLE进程

- 日志写（Log Writer,LGWR）：LGWR是将在系统全局区中重做日志缓冲区的重做日志条目写入到联机重做日志文件的进程。

  LGWR执行写入操作的条件是：发生提交、重做日志缓冲区满度达到三分之一、超时(3秒)、DBWR进程开始之前。



4.2.1.2 ORACLE进程

- 日志写（Log Writer,LGWR）：

直到Oracle在LGWR完成将重做信息从重做缓冲区刷新到联机重做日志文件之后，Oracle才认为一个事务已完成。在LGWR成功地将重做日志项写入联机重做文件时（并不是改变数据文件中的数据时），将一个成功码返回给服务器进程。

用于配置LGWR进程的选项很少，大多数的配置涉及重做日志缓冲区和支持该缓冲区的内存结构，而不是LGWR进程自身。

可以建立多个LGWR的I/O口，以改善向日志文件的写入性能，init.ora中LGWR_IO_SLAVES参数决定。

4.2.1.2 ORACLE进程

- 日志写（Log Writer,LGWR）：

检查点导致LGWR和DBWR。检查点间隔时间越短，发生数据库故障时需要的恢复时间越短，同时减少了必须执行每一检查点所需的工作。当决定正确的检查点间隔时，必须权衡所有这些因素。

LOG_CHECKPOINT_INTERVAL参数即一定数量的操作系统块(不是Oracle块)写入重做时，引起数据库中的一个检查点被触发。
LOG_CHECKPOINT_TIMEOUT参数指定的时间间隔(以秒为单位)发生一个检查点。

4.2.1.2 ORACLE进程

- 日志写（Log Writer,LGWR）：

另外当一个重做日志组写满时，一个检查点被触发。要注意不要设置不必要的检查点，或者迫使不需要的检查点发生。例如，如果一个重做日志组大小为3MB，而LOG_CHECKPOINT_INTERVAL设置为2.5MB。当有2.5MB的数据写入重做日志时，导致发生一个检查点。另外，当重做日志组写满时（仅在又写入0.5MB的数据后），发生另一个检查点。事实上，这两个检查点将相继发生。**频繁的检查点将浪费大量的进程活动和I/O次数以执行相关的检查点**。

最后一个参数是布尔值LOG_CHECKPOINTS_TO_ALERT。每当检查点发生时，它为数据库在alert.log文件中设置一个标记，并用于试图指出确切的检查点间隔。

4.2.1.2 ORACLE进程

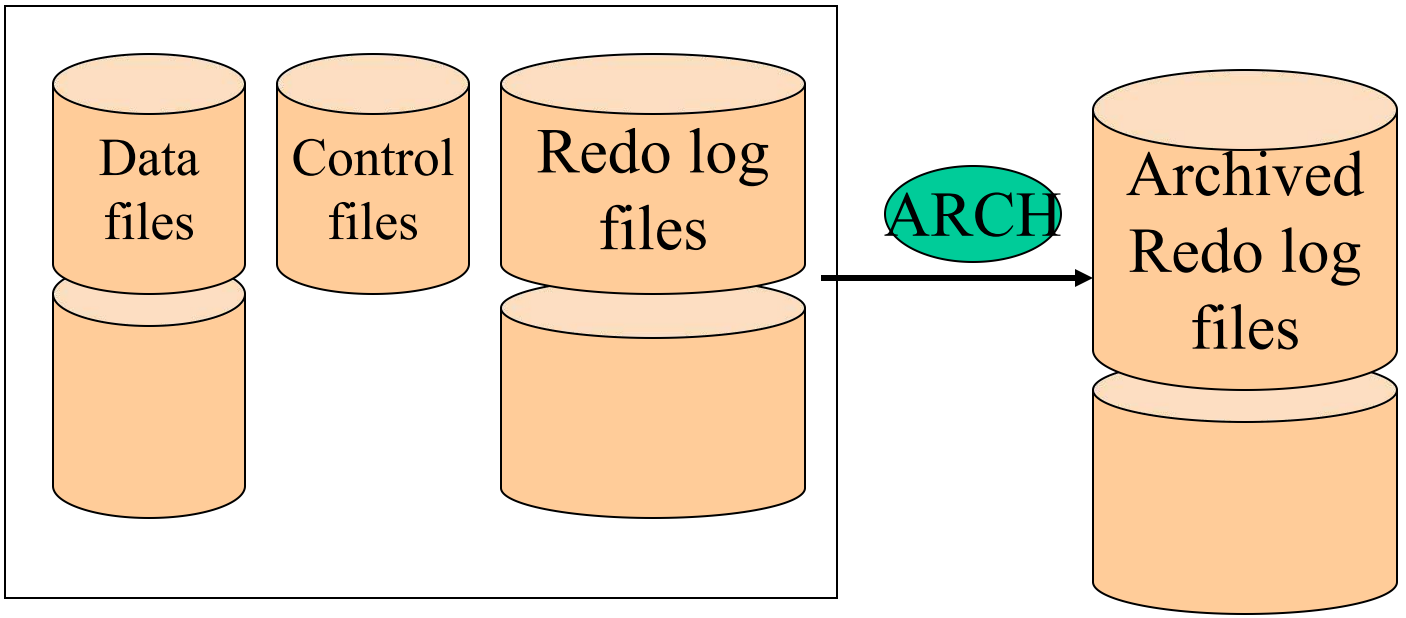
- 调度进程：正如前面所提到的，服务器进程既可以是一个用户进程专有的，也可以在多个用户进程之间共享。使用共享服务器要求配置多线程服务器。当使用共享服务器进程时，至少必须存在一个调度进程（Dnnn），在环境需要时也可能有多个调度进程。**调度进程将用户请求传送到系统全局区的请求队列，并将服务器的响应信息返回给正确的用户进程。**

  参数MTS_DISPATCHERS指定调度进程使用的协议及开始使用该协议的调度进程的数量。

  参数MTS_MAX_DISPATCHERS控制RDBMS允许的调度进程的最大数量。

4.2.1.2 ORACLE进程

- 归档进程(Archiver,ARCH):ARCH负责将全部联机重做日志复制到归档重做日志文件。这仅在数据库运行在归档模式（ARCHIVELOG）下才发生。



4.2.1.2 ORACLE进程

- 归档进程(Archiver,ARCH):

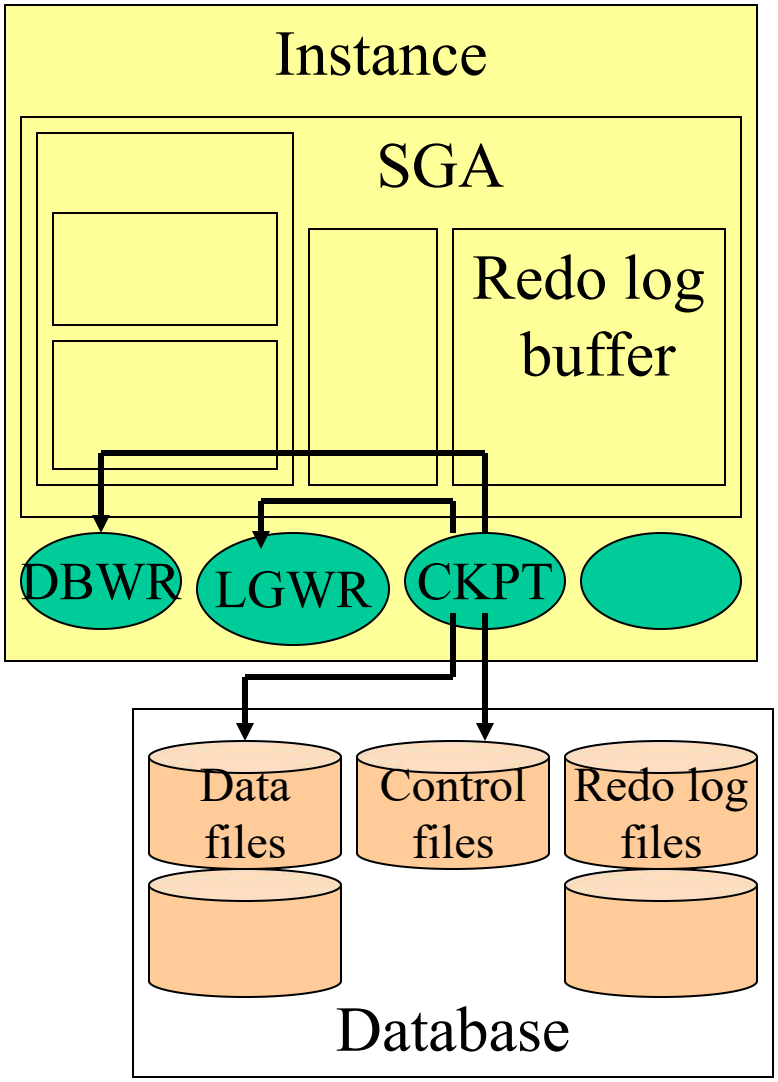
当ARCH正在复制归档重做日志时，没有其他进程能够写入这个重做日志。重做日志是按顺序循环使用的,如果数据库需要转换重做日志，但是ARCH还正在按其顺序复制下一个日志，所有数据库的活动将终止，直到ARCH完成。如果归档由于某些原因不能完成复制日志，它将等待直到引起不能写入的错误得到解决为止。

在init.ora文件中ARCHIVE_LOG_START参数必须设置为TRUE，当数据库启动时，才会自动开始归档。如果设置了归档模式，但不自动启动ARCH，当所有联机重做日志写满时，数据库将会挂起，等待你手工归档联机日志。

4.2.1.2 ORACLE进程

- 检查点进程(CKPT):CKPT是可选的后台进程，执行LGWR进程通常会执行的检查点任务——即用当前版本信息更新数据文件和控制文件头。

  CHECKPOINT_PROCESS参数设置为TRUE可以启用CKPT进程。注意在Oracle 8.x中，该已被集成入RDBMS中并设置为TRUE。如果在Oracle 8.x的init文件中包括这个参数，实例启动将会失败。



4.2.1.2 ORACLE进程

- 恢复进程(RECO):负责在分布式数据库中恢复失败的事务。分布式事务配置数据库时(设置DISTRIBUTED_TRANSACTIONS值大于0)，RECO会自动启动。
- 快照进程(SNPn):处理数据库快照的自动刷新，并运行预定的数据库过程。参数JOB_QUEUE_PROCESS设置启动的快照进程数，参数JOB_QUEUE_INTERVAL决定快照进程在被唤醒以处理挂起的作业或事务之前休眠的时间(以秒为单位)。

4.2.1.2 ORACLE进程

- 锁进程(LCKn):在并行服务器环境中,多个实例安装在同一个数据库上，锁进程(LCKn)负责管理和协调每个实例占有的锁。
- 并行查询进程(Pnnn):根据数据库的活动和并行查询选项的配置，Oracle服务器启动和停止查询进程。这些进程涉及并行索引的创建、表的创建及查询。启动的进程的数量与参数PARALLEL_MIN_SERVERS指定的数量相同，但决不能超过参数PARALLEL_MAX_SERVERS指定的进程数。

4.2.1 ORACLE数据库实例组成

- 为更好地理解前面讨论的实例组件是如何相互交互的，看一个典型的事务在实例结构中的移动。
- 当一个用户会话使用SQL*Net驱动程序连接到服务器会话时，开始一个事务。这个连接可以使用它自己的一个服务进程进行专用连接，或通过调度进程处理的一个共享连接。服务器会话对传递给它的SQL语句与已经保存在共享SQL区中的语句进行比较，如果在共享池中发现有完全一样的语句，使用该语句早已存储的做过语法分析的形式与执行计划。如果在共享池中没有发现匹配语句，服务器进程对这些语句进行语法分析。

4.2.1 ORACLE数据库实例组成

- 接下来，服务器会话查看在数据库缓冲区中是否已经存储了完成该事务所必须的数据块。如果在缓冲区中没有，服务器会话就从数据文件中读取必要的数据，将它们拷贝到缓冲区中。如果事务是一个查询，服务器会话将查询的结果返回给用户进程（执行必要次数的数据块读和拷贝以返回所有数据）。
- 对一个修改数据的事务，有更多的步骤需要做。例如，假设事务是一个更新。在将必要的数据块读入缓冲区高速缓存中之后，修改内存中的数据块。修改的缓存块标记为脏的，并把它们放入脏列表中。还产生重做信息，并将重做信息存储在重做日志缓存中。

4.2.1 ORACLE数据库实例组成

- 如果事务是相对短期的，事务完成，用户提交，发出信号给LGWR进程让它将重做日志缓冲区刷新到联机重做日志文件。如果事务相对持续时间很长和很复杂，下列事情有可能发生：

  1）产生的重做引起重做日志缓冲区写满三分之一的空间，这会触发LGWR进程刷新重做日志缓冲区。

  2）放入脏列表中的块数达到限定的长度，这会触发DBWR进程将数据库缓冲区所有脏列表项刷新写入到数据文件中，它反过来也使LGWR进程向磁盘刷新重做日志缓冲区。

  3）遇到一个数据库检查点，这将触发数据库缓冲区高速缓存和重做日志高速缓存刷新。

4.2.1 ORACLE数据库实例组成

4）可获得的缓冲区高速缓存中的自由缓存空间下降到规定的门限以下，这也会引起数据库高速缓存缓冲区的刷新。

5）产生一个不可恢复的数据库错误，这迫使该事务中断、回滚并向服务器会话报告错误。

- 当事务正在处理向重做缓存生成的重做并刷新时，联机重做日志逐渐被填满。当前日志被填满后，LGWR进程开始写入下一个日志组；与此同时，归档进程将重做日志复制到磁盘或磁带。因为直到所有重做日志信息从重做缓冲区写入到联机重做日志之后，事务才被记录为执行成功的，所以LGWR和ARCH必须各自有能力在无错误的情况下完成自己的任务。

4.2.2 ORACLE数据库实例创建

- 通过前面介绍，我们知道**ORACLE数据库服务器的启动事实上是创建实例的过程**。打开一个Oracle数据库的过程具体包括以下三步：

  1）创建一个Oracle实例（非安装阶段）

  2）由实例安装数据库（安装阶段）

  3）打开数据库（打开阶段）

  Oracle实例在数据库启动的非安装阶段创建。当数据库经过非安装阶段时，读取init.ora参数文件，启动后台进程，初始化系统全局区（SGA）。init.ora文件定义了实例的配置，包括内存结构的大小和启动后台进程的数量和类型等。

4.2.2 ORACLE数据库实例创建

- 下一阶段称为安装阶段。init.ora文件中的控制文件参数值决定数据库的安装实例。在安装阶段，读取控制文件并使其成为可访问的，可以对控制文件内存储的数据进行查询和修改。

  最后的阶段就是打开数据库。在这一阶段，存储在控制文件中的数据库文件以排它使用方式被实例锁定，使数据库能够被普通用户访问。打开是数据库的正常操作状态，在数据库打开之前，只有DBA能访问数据库，且只能通过服务器管理器对其进行访问。

4.2.2 ORACLE数据库实例创建

Oracle启动进程	0	开始状态		SHUTDOWN
	1	启动实例	读参数文件；分配SGA；启动后台进程；打开跟踪文件和警告。	NOMOUNT (Start up nomount;)
	2	装载数据库	打开控制文件(控制文件的位置在参数文件中约定)。	MOUNT (Alter database mount;)
	3	打开数据库	打开数据文件；打开日志文件；(数据库包含的文件名及位置信息在控制文件中约定)。	OPEN (Alter database open;)

4.2.2 ORACLE数据库实例创建

- 为了改变数据库的操作状态，必须拥有SYSDBA特权。当数据库从关闭状态到打开状态时，可以单步调试数据库的每一个操作状态，例如，可以在服务器管理器工具中执行STARTUP NOMOUNT命令，使数据库处在非安装阶段，接下来可以运行ALTER DATABASE MOUNT或者ALTER DATABASE OPEN命令以单步调试到操作阶段。

  无论是在何种操作状态下，如果执行SHUT DOWN命令，将完全关闭数据库。当关闭数据库时，只能从当前运行状态转到完全关闭状态，数据库不能从打开状态转到安装状态。

4.2.2 ORACLE数据库实例创建

- 没有安装数据库的实例称为空闲的——它使用内存，但不做任何工作。一个实例只能唯一地与一个数据库连接，而且对一个数据库也只分配一个实例(使用并行服务器除外)。
- 启动数据库命令：

  STARTUP [FORCE][RESTRICT][PFILE=filename]
  [OPEN [RECOVER]|MOUNT|NOMOUNT];

  FORCE 强制关闭数据库，然后正常启动，

  RESTRICT 启动数据库，只允许具有特权的用户访问，

  RECOVER 启动数据库同时开始介质恢复，

  PFILE 说明启动实例使用的参数文件及其位置。

4.2.2 ORACLE数据库实例创建

- 关闭数据库过程与启动相反，关闭数据文件，关闭日志文件，关闭控制文件，关闭实例并释放资源。关闭数据库命令：

  SHUTDOWN
  [NORMAL|TRANSACTIONAL|IMMEDIATE|ABORT];

SHUTDOWN	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
允许新的连接	NO	NO	NO	NO
等待用户结束会话	NO	NO	NO	YES
等待事务结束	NO	NO	YES	YES
产生检查点并关闭数据文件	NO	YES	YES	YES

4.2.2 ORACLE数据库实例创建

- 关闭数据库的方式中可以看出，

对于NORMAL、 IMMEDIATE、TRANSACTIONAL的关闭数据库方式，下次启动数据库时不需做例程恢复;

对于ABORT的关闭方式，下次启动要做例程恢复。

4.2.3 ORACLE数据库监控实例

- 一般情况下，系统全局区和后台进程在没有DBA干预的情况下运行。然而，有时可能会发生问题，这时DBA必须能诊断和修复发生的问题，即DBA应当能够有效地监控和管理实例。以下介绍的**三种方法可以监控和追踪实例及其相关的结构**。
  - 使用诊断追踪文件管理实例
  - 通过操作系统追踪管理实例
  - 使用V$表监控实例结构

4.2.3 ORACLE数据库监控实例

- 使用诊断追踪文件管理实例

诊断追踪文件中包含有实例运行过程中的事件发生信息，充分利用它可以发现和解决许多问题。诊断追踪文件有三类：警告文件(alertSID.log)、后台进程跟踪文件、用户跟踪文件。

查找关于实例的问题最好的地方是在进程自己的追踪文件中。根据特定的进程和所遇到的错误，这些追踪文件被写在由参数USER_DUMP_DEST或者BACKGROUND_DUMP_DEST所指定的位置。

4.2.3 ORACLE数据库监控实例

- 使用诊断追踪文件管理实例

后台进程失败经常在数据库的alert.log文件中写入一项，或者写到它们各自的追踪文件，当一个后台进程被终止或者非正常中断一个操作时，通常产生一个追踪文件，包含导致失败的错误信息、当前进程堆栈的转储、当前执行的游标和与问题有关的其他信息。

DBA应当收集这些追踪文件，将它们提供给Oracle全球范围客户支持顾问，他们会帮助你诊断问题。

4.2.3 ORACLE数据库监控实例

- 通过操作系统追踪管理实例

后台进程也能通过使用操作系统命令进行追踪。在UNIX环境中，每一个后台进程是一个分立的任务，可以分立追踪。通过查看操作系统进程的内存和CPU的使用（通过使用sar、ps、vmstat和top等工具）指明性能问题和失控的查询。有时解决挂起、中断服务器进程或用户进程的唯一方法是在操作系统级中断它们。但只能对有十足经验的用户使用。

4.2.3 ORACLE数据库监控实例

- 通过操作系统追踪管理实例

在NT服务器环境中，全部的Oracle实例在NT操作系统作为一个称为服务的后台进程实现。各个后台进程作为属于这个服务的线程实现。使用与NT OS的Performance Monitor工具进行监控，它监控许多事情，包括属于这个服务的线程的内存的消耗和环境的切换。详细情况可查看有关使用手册。

4.2.3 ORACLE数据库监控实例

- 使用V$表监控实例结构

DBA可以使用许多动态性能视图显示实例的信息。这些视图对发现当前数据库实例的状态和与实例相关的故障问题有很大的价值。

1. 监控数据库的连接

所有连接到实例的用户和后台进程都能够使用V$视图被监控。

4.2.3 ORACLE数据库监控实例

1. 监控数据库的连接

V$process视图显示所有连接到数据库的进程的信息，包括后台进程和用户进程；

V$bgprocess含有所有可能的后台进程的列表和一个附加的PADDR列，PADDR列包含后台运行进程的十六进制地址；

V$session视图包括连接到数据库的用户会话的信息。

V$mts视图含有共享服务器进程的追踪信息。这个视图包含的列有最大的连接数、服务器启动、服务器中止和服务器的最高水位。

4.2.3 ORACLE数据库监控实例

2. 监控共享SQL区

视图V$sqlarea包含共享SQL区中的SQL语句的信息，包括被执行SQL语句的文本、访问该语句的用户数、执行语句时访问的磁盘块和内存块以及其他信息。在V$sqlarea中，disk_reads和buffer_gets列追踪从磁盘中读入的块数和从缓冲区中读入的块数。

V$open_cursor视图可以找出尚未关闭的游标。

4.2.3 ORACLE数据库监控实例

2. 监控共享SQL区

视图V$sqltext也能用于确定传递给数据库引擎的SQL语句。V$sqlarea视图仅存储SQL语句的前80个字符，而视图V$sqltext保存完整的SQL语句。

除了在SQL语句中适时加入换行符之外，V$sqltext_with_newlines视图与V$sqltext视图是完全相同的。在V$sqltext视图中，SQL语句是分块存储的。要获取整个语句，你必须按照PIECE列排序获取SQL语句的所有部分。

4.2.3 ORACLE数据库监控实例

3. 监控系统全局区

V$sga视图显示每一个SGA主要组成部分的大小(以字节为单位),包括重做日志缓冲区、数据库缓冲区高速缓存和共享池。V$sgastat视图包括更多的相关信息，在这个视图中能查找在系统全局区包含的每个单独的内存结构的大小，包括为堆栈空间、PL/SQL变量和堆栈预留的内存。也可以查询这个视图，找出SGA区中可获得的自由内存的总数。

4. 监控库和字典缓冲区

V$librarycache视图中包含库缓冲区中的每种类型对象的性能信息；V$rowcache视图中包含数据字典缓冲区的性能信息。

4.2.3 ORACLE数据库监控实例

5. 监控并行查询进程

查询视图V$pq_sysstat显示并行查询服务器的当前运行信息，例如忙的和空闲的查询服务器的数量与动态服务器创建和终止的统计数。V$pq_tqstat视图包含以前使用并行查询服务器运行的查询的信息。

6. 监控归档进程

归档进程活动信息存储在V$archive视图中，可以从这个视图中查询由ARCH进程写入的归档日志信息。

4.2.3 ORACLE数据库监控实例

7.监控多线程服务器进程

视图V$mts、V$dispatcher和V$shared_server包含多线程服务器（MTS）和内存结构的状态信息。V$mts视图包含共享服务器进程的追踪信息，例如启动和终止的服务器数量和运行服务器的最高水位；V$dispatcher视图包含运行的调度进程的信息，从这个视图中，可以查询调度进程的名字、支持的协议、处理的字节数、处理的消息数、当前的状态和与调度进程相关的其他运行信息；V$shared_server视图为运行的共享服务器进程提供相同类型的信息。

作业:

1. 何谓ORACLE实例?
2. 为什么ORACLE有SGA?
3. 服务器进程的功能是什么?
4. 叙述ORACLE数据库的启动和关闭过程.
5. 监控ORACLE实例的方法有哪些?

4.3 ORACLE数据库存取管理

- 数据库管理员的任务是使所有运行Oracle数据库的系统能够有效的工作。数据库管理员为每一个用户提供技术支持，并且应当对Oracle软件中出现的所有问题都相当熟悉。除上一节介绍的ORACLE实例设置外，ORACLE数据库的存储对数据库的性能也有很大影响，另外ORACLE的安全性、备份与恢复机制也在本节中介绍。

  4.3.1 管理ORACLE存储

  4.3.2 ORACLE安全管理

  4.3.3 ORACLE备份与恢复

4.3.1 管理ORACLE存储

- 在2.1的数据库数据存储介绍中，我们已经知道ORACLE的数据存取是面向数据块(页)的；在4.1中，我们知道ORACLE数据库的存储体系是表空间、段、区间、块，下面将集中讨论ORACLE存储对性能的影响。

  4.3.1.1管理数据库对象

  4.3.1.2理解数据库碎片

  4.3.1.3管理回滚段

  4.3.1.4鉴别存储问题

4.3.1.1 管理ORACLE数据库对象

- 管理Oracle块

Oracle服务器寻址的最小存储单元是Oracle块。无论段是一个表、索引、簇或者其他对象，块结构是相同的。**设计一个性能最优的数据库要从对Oracle块适当的配置和管理开始**。

Oracle块是由三部分组成的：块头、数据存储区、自由空间区。块头包含有关块的信息(什么类型的段数据存储在块中，什么段在块中有数据，块地址以及指向存储在其中的实际行的指针)。块头由一个固定部分和一个可变部分组成，在块中块头通常使用85到100字节。

在Oracle块中，主要关注管理数据存储区域和自由空间区域。数据区域是块中实际存储行的地方。自由空间区是一个保留的区域，被定义为总的可用空间的百分数，用于存储有关在块中的行将来更新的信息。

4.3.1.1 管理ORACLE数据库对象

- 理解PCTFREE与PCTUSED

PCTFREE和PCTUSED是应用于段的两个存储参数，当Oracle向数据库中写信息时，它必须首先在一个段的分配区中找到一个或更多块来存储信息。

如果在块中自由空间的百分数比PCTFREE参数大得多，它就可以用来存储新信息。当自由空间的百分数低于PCTFREE时，块就被认为是“满的”。

当块中使用空间的百分数低于PCTUSED时，它就可以被用来存储新信息。

4.3.1.1 管理ORACLE数据库对象

- 理解PCTFREE与PCTUSED

使Oracle保持足够的额外空间用于行增长(修改结构)，而不需要跨块存储。保持行被限制在一个单独的块中有助于使数据库以最高性能运行。

PCTFREE和PCTUSED的值不能等于100%。由数据库引擎处理引起的系统开销可以通过在PCTFREE和PCTUSED之间留至少20%的裕量而得到解决。缺省PCTFREE为10，PCTUSED为40。

PCTFREE和PCTUSED在数据库段的存储子句中被指定。可以用dba_tables、dba_clusters或者dba_indexes数据字典视图查询它们目前的值。

4.3.1.1 管理ORACLE数据库对象

- 索引与PCTUSED/PCTFREE

索引使用PCTFREE和PCTUSED与表相比不同。**只有当一个索引最初建立时这些参数适用**。PCTFREE的设置应让索引结构最初在叶块中建立时就有足够的空间，这样就可以增加辅助关键字，而不必拆分已存在的叶块提供给新关键字。**索引不必设置PCTUSED**。

当为一个已存在的表建立新索引时，如果预料到表中的数据较稳定，修改不多时，可选择较小的PCTFREE，反之则应选择较大的值。

4.3.1.1 管理ORACLE数据库对象

- 管理表存储区

创建表时，系统自动在表空间中分配一个区间，构成数据段。表的空间需求是平均行长与行数的乘积。确定表的容量应该在表建立之前的设计阶段完成。预期的增长和存储参数值等问题应该在建表时处理。**ORACLE用如下方式来为未来的数据预留空间**(参看2.1内容)：

1）为表的数据段设置存储参数，控制每次分配的区间大小。

2）用存储参数PCTFREE和PCTUSED控制构成数据段的区间中各个块的空间利用率。

4.3.1.1 管理ORACLE数据库对象

- 管理索引

当查询语句中的WHERE子句表达式含有被索引的关键字时，ORACLE将自动启动索引。

许多应用程序员(以及DBA)通过申请更多索引的办法解决数据库的性能问题。但**更多的索引不仅会影响修改性能，而且在一个查询中并不是子句中涉及的索引都启用**。

过多的索引并不会一直提供比较好的SELECT性能。

4.3.1.1 管理ORACLE数据库对象

- 管理索引：关于索引的管理和使用的几点技巧：

1）组合索引是提高性能的有力工具，特别是针对只需从索引段读出而不用从数据段的查询。

2）如果在查询子句中包含‘IS NULL’或‘IS NOT NULL’，索引将不在启用。

3）在查询条件中涉及多个索引，而又无明显优先可选项的查询中，ORACLE一般按索引类型及列的性质选择启动索引，当有唯一性和非唯一性索引均可使用时，选择只启用唯一性索引。

4）在关键字列上进行运算可以屏蔽定义在此列上的索引。

5）在WHERE子句中若包含不等于(‘!=’或‘NOT’)运算，ORACLE将不在启用索引，但可对其他列使用索引。

4.3.1.1 管理ORACLE数据库对象

- 监控回滚段

回滚段实际上是动态的，DBA应该集中精力**保证每个回滚段及与之联系的表空间有足够的空间用于增长**。

关心回滚段的目的在于合理地配置系统资源，提高回滚段的利用率，关键是各个事务要占用多大的存储空间，其活动时间有多长。

4.3.1.1 管理ORACLE数据库对象

- 管理临时表空间和段

临时段是SELECT语句在需要的时候向ORACLE申请的存储资源，随SELECT语句结束而注销。只有查询的直接结果与查询的显示结果有差异时才会申请临时段，保存中间结果，比如DISTINCT、ORDER BY等。

在缺省情况下，临时段创建在系统表空间中，而临时段频繁的创建和删除势必造成系统表空间的效率降低，还会使系统表空间中的‘碎片’增加，影响存储器的有效使用。

应该设计一个专门的表空间来存放临时段，最好将其分配在与系统表空间和数据段不同的存储器上，减少I/O冲突。在创建用户时，应同时为其指定临时段所使用的表空间。**临时段表空间最起码的容积应有两倍的单个数据表信息量，还要考虑用户数**。

4.3.1.2 ORACLE数据库碎片

- DBA面对的最常见问题就是**处理数据库对象的碎片**。碎片浪费空间，导致性能问题，并给数据库对象的管理带来困难。数据库碎片通常是行被插入、修改和删除以及对象被建立和删除的结果，一个新建立的数据库不会有碎片。只有当在数据库上运行了一个或更多的活动应用程序后，才开始看到数据库碎片的迹象。
- **数据库碎片包括分裂成碎片的数据库对象，分裂成碎片的表空间、链接行和转移行**。DBA应该,注意数据库应用程序是如何执行任务的，尽量避免数据库应用程序运行过程中产生碎片，并且监测数据库运行状态，对产生的碎片进行必要的处理。

4.3.1.2 ORACLE数据库碎片

- 分裂成了碎片的表空间

表空间变成碎片是由于运行错误或者表空间中的数据库对象的无计划撤消和重建造成的。表空间不会由于任何行级动作而变成碎片。

Oracle尽力帮助DBA进行空间分配和管理。例如假定一开始表空间有四个10MB表段，现在删除其中的两个表，这两个表都位于表空间的“中间”位置，SMON后台进程自动地把两个10MB的区间聚合为一个20MB区间。如果要删除的两个表不是位于表空间的“中间”位置，则形成表空间碎片。

表空间碎片会导致：表空间中的空间被分离且不能有效地使用。

DBA_FREE_SPACE和DBA_EXTENTS数据字典可以查询数据库的每个表空间中有多少自由空间是可用的和最大自由区间的容量以及每个表空间中自由区间的数目。

4.3.1.2 ORACLE数据库碎片

- 处理分裂成碎片的表空间

处理表空间碎片的最好方法是避免它，这需要仔细地计划和管理。避免碎片的几种提示方法：

1）把对象用相似的空间和增长特性聚合在一起。

2）如果可能，把它们的区间设成相同的容量，这样所有的对象可以共享释放对象回收的区间。

3）使每个段的PCTINCREASE保持为0(注意Oracle7.3后，设置PCTINCREASE比0大会导致SMON自动聚合相邻的自由区域)。

对确实需要整理的分裂成碎片的表空间，简单的方法是在表空间中导出损坏了的对象，删除该对象，再把它们导入回来。这不仅能够聚合自由空间，而且导出操作也可以把数据库对象聚合到一个区域中。

4.3.1.2 ORACLE数据库碎片

- 对象碎片

碎片问题可能发生在对象级。对象碎片会导致：由于在表和索引块中的自由空间空洞导致空间的浪费；读性能下降，因为数据不再被紧紧地排在一起，物理磁盘驱动器必需从一个比需要的大的磁盘表面区域中搜寻和读数据。

对象碎片的三个特殊碎片类型：行转移、行链接和过分扩展的段。

DELETE操作可能会产生**稀疏的块**，因为这些块不满足PCTUSED条件，以至没有成为自由空间。处理对象碎片的操作也会处理这类稀疏的块问题。

DBA在一个经常改变的表空间中发现空间分配问题，并重建了表空间和它的所有对象，随后数据库性能提高了。这种在性能上的提高是消除表空间碎片的结果，但也可能是消除对象级碎片带来的性能的提高。

4.3.1.2 ORACLE数据库碎片

- 行转移：**行转移在对行的修改引起行的长度比块中可利用空间大时发生**。当发生行转移时，行的一部分转移到一个新的数据块中，这个新数据块的地址存储在原始行单元中。当Oracle需要读这个行时，它首先读原始单元并找到一些数据和指向另一个块的指针，再在这个块中找到剩下的行数据。

  为什么Oracle执行行转移而不是把整个行移到一个新块中？因为移动一个行的单元会引发ROWID的变化，而ROWID的变化引发一系列的变动，代价太大。

  行转移存在两个问题：

  1）Oracle必须读每一个包括行数据的一部分或者是指向另一个包含着行数据的块的指针的块，无疑这会导致相当大地全系统的额外开销。

  2）Oracle必须和行数据一起存储额外的指针来供给行转移机制，浪费了一些磁盘空间。

4.3.1.2 ORACLE数据库碎片

- 行链接：**行链接在行太长以致于不能放入任何一个数据块时发生**。这导致该行被存储在一个或多个数据块的链中。行链接伴随着包含LONG、LONG RAW或者LOB数据类型的大行时发生。
- 消除链接和转移的行

链接和转移的行会对数据库的整体性能造成一系列的问题。可以通过下面的SQL语句检查链接和转移的行：

```
analyze table customer list chained rows
into cha;
```

这个命令把ROWID和数据库中所有链接和转移行的表存储到cha表中。直接查询cha表，即可找出行和表的特定信息。

4.3.1.2 ORACLE数据库碎片

- 消除链接和转移的行

消除转移的行的方法是直接查询cha表，把转移的行从表中移出放入一临时表中，然后再集中将临时表中数据添加进表，即可消除转移的行，该方法可能使原来相邻存放的数据变得不再相邻，并不一定能提高性能；另一个方法是使用全表导出，删除表，之后使用导入重建该表，该方法最简单、最全面，肯定会消除行转移引起的性能问题，但无疑是消耗时间最长的。

行链接是非常难调整的。唯一的**解决办法是缩短表的行长或者增加数据库块的大小**。选择前一种方法通常需要对表结构进行重新设计，但是后一种方法需要导出整个数据库，重新创建数据库然后全部导入，因为块的大小调整必须重构数据库。

4.3.1.3 管理回退段

- 像其他段一样，回滚段被定位在表空间中并且包含区间。每个数据库有一个SYSTEM回滚段，它位于系统表空间中并在数据库建立时建立，用户不允许使用它并且它不能被删除。正象第三章所述，ORACLE数据库应该建立附加回滚段。
- 在有许多并行事务的系统上需要有更多区间的更多回滚段。事务越大，回滚段也就需要越大。因为**回滚段通常允许增长，然后收缩回到存储参数OPTIMUM的值**，一定要特别小心以确保一个回滚段需要的空间裕量。
- 当创建的回滚段空间太小或者创建的回滚段太少时，Oracle通常出现“快照太旧”的错误信息。

4.3.1.3 管理回退段

- 回滚段操作

事务可以通过显式或者隐式方式被指定到一个特殊的回滚段中。Oracle用扩展(存储子句)和循环(释放的空间可以重用)的方式使用回滚段。**如果达到了一个回滚段的maxextents，或者表空间中没有更多的空间分配给回滚段的话，扩展就失败了**。应该给回滚段分配预留扩展的表空间。

包含在任何给定的表空间中的所有回滚段应该被精确地分配成相同的大小，如果设置了不匹配的区间大小，那么可能出现前面讨论过的碎片问题。也即**尽量为活动时间大致相同的事务创建一个回滚段，这样可以使回滚段的区间利用率达到最好水平**。

特殊回滚段是为特殊事务类(如DSS和OLTP)创建的,这些可能被设置为不通用大小的回滚段，它们应该存储在它们自己的表空间中以使碎片减到最小。

4.3.1.3 管理回退段

- 设定回滚段大小

数据库设计者对回滚段通常考虑如下问题：

1）使用多少个回滚段；

2）给每个回滚段分配多少空间；

3）各个回滚段是专用的还是共享的；

4）各个回滚段分别住留在哪些表空间中。

当设定回滚段大小时，应该被设置得足够大，这样系统就不必马上重新使用刚被一个完成的事务释放的块，同时有足够的空间来存储非活动的或者被提交的数据事务，而这些数据事务可能是其他长时间运行的查询所需要的(读一致性)。

使用SET TRANSACTION USE ROLLBACK语句在事务开始时明确地使用某个回滚段。可以建立一个只被长时间运行的批量工作使用的大回滚段。

4.3.1.3 管理回退段

- 使用OPTIMAL参数

当建立回滚段时，回滚段的目标容量可以用存储参数OPTIMAL定义。OPTIMAL参数指定了回滚段要缩小到的尺寸。

例：
```
create rollback segment r05 tablespace rbs
    storage (initial 10M next 10M optimal 20M
             minextents 2 maxextents 100);
```

ORACLE尽力把回滚段的尺寸维持在指定的最佳值20MB上。当回滚段开始写入另一个区间时，Oracle总是检查段是否比OPTIMAL的尺寸大，如果大，并且下一个区域是完全空闲的，Oracle就释放该区间。

如果回滚段的尺寸设置不适当，回滚段会由于OPTIMAL子句的缘故持续地收缩，这种不断的区间分配和释放会影响到性能。

4.3.1.3 管理回退段

- 进行装载测试以获得回滚段估算

为了正确地设置回滚段的尺寸，需要对将由较大事务产生的撤消信息的数量有个了解。可以在样本事务中执行一个**装载测试**：

1）建立两个表保存统计信息：undo_begin和undo_end。

2）在与数据库连接后，发出一系列事务处理命令，使用动态性能视图v$rollstat把写入指定回滚段的原始数据捕获到表undo_begin中。

3）在事务处理结束后，再把写入指定回滚段的数据放到表undo_end中。

4）在表undo_begin和undo_end中已经有了开始和结束值，将这两个值相减便可得到写入指定回滚段的数据。这些值可以用于确定你的回滚段的尺寸需求。

4.3.1.4 鉴别存储问题

- 当数据库启动和运行时，也需要查看数据库的每个主要组件有多大。目的是要**确保表空间足够大，能够存储它们所包含的段**。
  - 开发表空间

    表空间的开发包括了解表空间的总尺寸以及表空间使用什么样的数据文件两方面。**DBA_DATA_FILES表包含所有关于数据文件的信息**，表空间的尺寸是它的数据文件的总和。**DBA_FREE_SPACE表中保存数据库中所有自由区间的清单**，通过合计表空间中自由区间的尺寸，会看到还剩多少自由空间、每个表空间中的最大自由区域多少。**DBA_SEGMENTS表包含了数据库的区间信息**。应该通过这三个表定期评价你的表空间被分裂成碎片的程度。

4.3.1.4 鉴别存储问题

- 表检查

表检查包括估计总的容量以及实际使用的空间。一个已用空间与分配空间的比值很低的区间意味着INITIAL和NEXT区域值可能被设置得太高了。可以在DBA_EXTENTS表中找到区间信息。

- 优化簇存储和检查索引

理想情况下，每个簇刚好有一个区间。可以查询DBA_EXTENTS计算每个簇的区间数目。可能的情况下，重新计算最佳的存储参数，并重建一个簇。

同簇一样，理想情况下每个索引刚好有一个区间。

- 监控临时表空间

排序、链接和GROUP BY子句等操作要使用临时表空间。要监控临时段，可以检查**DBA_SEGMENTS中类型为TEMPORARY的所有段信息**。

4.3.2 ORACLE安全管理

- Oracle安全模型原理是基于最小特权原则。此原则认为用户应该仅有完成他的任务所必需的特权。ORACLE的安全性主要通过用户(USER)管理、权限(PRIVILEGE)管理、角色(ROLE)管理来实现。ORACLE的安全管理在第二章中已经作过介绍，下面的内容是第二章的继续。

  4.3.2.1 用户验证

  4.3.2.2 权限管理

  4.3.2.3 审计

4.3.2.1 用户验证

- 数据库验证

当创建用户时，可以为其选择资源限制文件。

资源限制文件(profile)可以使用企业管理器(OEM)或SQL*Plus创建环境资源文件，使资源限制文件生效需在init.ora中设置resource_limit=TURE。

资源限制文件可以做到口令限制和资源限制。

资源限制包括会话级限制(当达到限制要求时则断开会话)和调用级限制(当达到限制要求时回退语句)。

具体有：限制会话的数量、每个会话使用CPU时间(100秒为单位)、CPU调用的次数、逻辑读的次数、每次调用的逻辑读的次数、空闲时间以及连接时间等等。

4.3.2.1 用户验证

- 数据库验证

口令限制包括

用户帐户的锁定:当一个用户多次注册失败后，该帐户能够被锁定一段指定的时间;

口令使用期与到期:一个给定的口令在使用一段时间后过期，此时必须修改该口令;

口令历史:口令历史选项检查每一个新近设定的口令以确保口令在指定长度的时间内或指定次数的口令修改中未重复使用;

口令复杂性验证:复杂性验证检查口令的长度以使它更难被计算机破坏者所破解(长度上至少为4个字符，不要与用户标识相同，至少包含一个字母字符、一个数字字符和一个标点符号，不要与简单单词的一个内部列表上的任何单词匹配，与以前的口令至少有3个字符不同）。

4.3.2.1 用户验证

- 数据库验证

例：CREATE PROFILE d_prof LIMIT

SESSIONS_PER_USER 2 --限制会话的数量

CPU_PER_SESSION 10000 --会话使用CPU时间(100秒为单位)

IDLE_TIME 60 --空闲时间

CONNECT_TIME 480; --连接时间

例：CREATE PROFILE grace LIMIT

FAILED_LOGIN_ATTEMPTS 3 --3次失败则锁定

PASSWORD_LIFT_TIME 30 --可以使用30天

PASSWORD_GRACE_TIME 5; --30天到期后再提示5天

4.3.2.1 用户验证

- 外部验证

外部验证依赖于操作系统或网络验证服务。此类登录不需要数据库口令。要使用外部验证， init.ora参数REMOTE_OS_AUTHENT必须被设置为TRUE（缺省值是FALSE）。另外在init.ora文件中设置OS_AUTHENT_PREFIX参数表示任何同此值有相同前缀的用户将在外部得到验证。比如，该值被设置为ops那么所有以ops打头的用户将在外部得到验证。

- 企业验证

企业验证启用一个中央资源用于口令管理，该用户被称为全局用户（global user）并且必须在每个使用全局口令的数据库中创建。假如口令得到验证，用户便能够访问Oracle数据库。Oracle安全服务与Oracle企业管理器的接口集中了安全角色管理和企业验证。这使得当前正在被管理的用户具有全局同一性。

4.3.2.2 权限管理

- ORACLE权限有系统权限和对象权限，通过将权限直接授予用户或通过将角色(一定权限的集合)赋予用户完成用户的权限管理(见第二章)。
- 两种特权：

Category	Examples
SYSOPER	STARTUP SHUNTDOWN ALTER DATABASE OPEN|MOUNT ALTER DATABASE BACKUP CONTROLFILE TO RECOVER DATABASE ALTER DATABASE ARCHIVELOG
SYSDBA	不仅具有SYSOPER权限，而且 CREATE DATABASE ALTER DATABASE BEGIN/END BACKUP RESTRICTED SESSEION RECOVER DATABASE UNTIL

4.3.2.3 审计

- Oracle能够审计并记录数据库中发生的活动。可以使用AUDIT命令允许审计操作或使用NOAUDIT命令禁止审计操作。有三种类型的审计操作：登录尝试、对象存取(具体对象上的具体语句)和数据库操作(具体的系统特权和语句，不考虑对象)。
- 任何一条命令，不管是成功的还是不成功的都可以进行审计。要创建审计系统视图，以用户SYS的身份运行CATAUDIT.SQL文件脚本，并设置init.ora参数AUDIT_TRAIL，该参数可以设置为写入数据库或一个操作系统文件：
  AUDIT_TRAIL=DB 写入数据库，所有已审计的活动被写入SYS.AUD$表中；AUDIT_TRAIL=OS 写入一个操作系统文件。

4.3.3 ORACLE备份与恢复

- 数据丢失分为物理丢失和逻辑丢失两类。

  物理数据丢失是指操作系统级的数据库组件的丢失，例如数据文件、控制文件、重做日志以及存档日志。

  逻辑数据丢失是诸如表、索引或表记录的数据库级组件的丢失。

  Oracle能够实现物理数据备份与逻辑数据备份。有效的备份计划必须把两种方案都包含在内才能完全避免数据丢失。

4.3.3 ORACLE备份与恢复

- 物理数据备份即为数据文件、控制文件、归档重做日志的拷贝。

物理备份应用于相同的Oracle版本以及引起物理备份的实例上的恢复，因此，物理备份被认为是不可移植的，它们通常仅用于保护同一台机器和实例中的数据免遭数据丢失。物理备份通常按照预定的时间间隔运行以防止数据库的物理丢失。

物理备份可以进一步分为冷备份(脱机备份offline backup)与热备份(联机备份online backup)。

4.3.3 ORACLE备份与恢复

- 逻辑数据备份通常是SQL语句的集合，这些SQL语句重新创建数据库对象(表、索引、授权、角色等)和记录。

当需要在不同实例之间移动指定的数据，或者需要在不同的系统结构、操作系统版本或Oracle版本之间拷贝一个实例的全部数据时可以使用Oracle的逻辑备份系统。逻辑备份用于实现数据库对象的恢复，用来保证数据库能够从无意中的修改(例如DELETE、DROP等)中恢复出来。

在开发者环境中，应该设计一个逻辑备份策略。一般存在两个数据库拷贝，一个是成品数据库，另一个由开发者使用，用来调试他们的程序。开发数据库将与成品数据库定期地保持同步。

4.3.3.1 ORACLE逻辑备份

- Oracle提供了逻辑备份工具EXP（IMP用于逻辑恢复）。命令格式如下：

  EXP [keyword=value[, keyword=value...]]

  - 完全逻辑备份

    EXP可用于完全逻辑数据库备份。但条件是必须有EXP_FULL_DATABASE权限和足够的存储空间。

    写入缺省输出文件中的完全逻辑备份的例子：

    exp USERID=SYSTEM/MANAGER FULL=Y

  - 指定用户模式的逻辑备份

    EXP支持指定用户模式的备份。导出一个指定的用户组会导出用户定义的所有表、索引等相关的数据。

    把TGASPER模式导出到标准输出文件中的示例：

    exp USERID=SYSTEM/MANAGER OWNER=TGASPER

4.3.3.1 ORACLE逻辑备份

- 指定表的逻辑备份

导出一组表的处理方法与用户导出的方法一样。用TABLES关键字列举表的名字，可以导出一个或一组表。

下面的例子把ABC表与XYZ表从BGE模式中导出到缺省输出文件中：

```
exp USERID=SYSTEM/MANAGER OWNER=BGE TABLES=XYZ,ABC
```

注意EXP缺乏一些主要的功能，EXP使你不能导出一条基于SQL WHERE子句的信息。Oracle的导出文件实际上是一种专有格式。市场上有许多第三方的工具提供了这些缺少的功能。

运行exp HELP=Y会显示EXP的所有关键字及其描述。

ORACLE逻辑备份工具EXP的对应恢复工具是IMP，其用法可以参照EXP。

4.3.3.2 ORACLE冷物理备份

- 冷物理备份是数据库管理员用来保护数据库系统整体上免遭各种类型数据破坏的工具。定期的冷备份以及一组好的归档重做日志与当前重做日志能够把数据库恢复到失败前的最后一次提交状态。
- 在NT桌面环境出现以前，往往借助操作系统工具执行冷物理备份。NT环境是图形化界面，所以Oracle创建了一个叫做备份管理器的图形化产品，备份管理器使用一个标准的NT图形用户界面程序执行物理备份与恢复。从Oracle 8开始，Oracle引入了一个称为恢复管理器（简称为RWAN）的新的备份与恢复系统。

4.3.3.2 ORACLE冷物理备份

- 无论采用什么方法，冷物理备份都应包含如下数据库元素：控制文件、数据文件、重做日志文件、归档日志文件。
  - 命令行驱动冷物理备份

    在备份过程开始以前，要确保数据库已有足够长的时间停留在脱机状态，这样才能得到完全备份。

    执行一个冷备份所需要的基本步骤如下：

    1）建立一个需要备份的操作系统级文件的列表。

    2）选择使用NORMAL或IMMEDIATE方式关闭数据库。注意仅在使用IMMEDIATE或NORMAL模式关闭数据库后才对它进行备份。

    3）执行步骤1中文件列表的操作系统级备份。

    4）正常启动数据库。

4.3.3.2 ORACLE冷物理备份

- 命令行驱动冷物理备份

建立一个需要备份的操作系统级文件的列表并不是简单的过程，下面提供一个方法。

1）init.ora文件必须备份。还要查看一下此文件是否有一个ifile行，假如有一个使用ifile关键字列出的文件名，还必须备份那个文件。

2）查看可用的init.ora文件，生成由数据库使用的控制文件列表。

3）使用DBA_DATA_FILES获得所有数据库数据文件。

4）下面的语句可生成重做日志文件的清单：

```
SELECT MEMBER FROM V$LOGFILE;
```

5）归档日志可由归档进程自动写到一个目录中。init.ora文件中的log_archive_dest关键字被设置为目的目录，归档日志将被写入这里。

4.3.3.2 ORACLE冷物理备份

- 桌面驱动冷备份

Oracle的备份管理器作为一个工具用于Oracle数据库的备份。备份管理器的界面操作相当容易。执行一个冷物理备份步骤：

1）确保Oracle数据库运行正常。（因为备份管理器需要在关闭数据库以前对该数据库进行访问以便它能建立一个需要备份的数据库文件列表。）

2）备份管理器中选择备份方式(offline_full database)。

3）确定备份目标:数据备份到磁盘或磁带中。

4）开始备份,备份管理器完成工作后重启数据库。

一般不把数据备份在数据库所驻留的物理设备中。备份管理器不备份init.ora、config.ora或归档日志文件。

4.3.3.3 ORACLE热物理备份

- 热备份要求对Oracle和操作系统具有较高的专业知识。由于热备份的复杂性，必须事先测试备份策略。只有从一个测试系统中的大量失败中恢复出来时，热备份系统才被认为是可靠的。
- Oracle提供了两种热备份方法，一种用于命令行UNIX环境，另一种用于桌面（Windows）环境。
  - 热备份复杂性

    Oracle基本存储单元是数据块。数据块中存储数据，数据块头存储包含在块中的数据的信息。数据库处于联机状态时，备份程序和Oracle可能都工作于同一个块上。假定当Oracle写新的数据和更新数据块头时，备份程序仅完成了对块头部分的备份，那么完成备份之后，该块具有老的块头和新的数据。这个数据块成为**不纯的数据块**。

4.3.3.3 ORACLE热物理备份

- 热备份复杂性

针对不纯的数据块，Oracle的解决方案是实现一种称为备份模式的特别模式，该模式在各个表空间上依次被启用或禁用。**如果当前处于备份模式，那么Oracle更新的数据块被完整地拷贝到重做日志中**。这样一来，Oracle数据文件的联机备份可能仍然存在不纯的数据块，但有效的拷贝存在于重做日志中。

对于联机备份的数据，在恢复状态时，可能存在不纯数据块的数据文件得到恢复，并且Oracle使用可能存在完整数据块(有效块)的重做(或归档)日志来“修补”数据文件。这样，最终的结果是一个有效且一致的数据库。

可以理解，联机备份时，日志文件(联机日志和归档日志)是不可缺少的。否则恢复过程可能得不到一个一致的数据库。

4.3.3.3 ORACLE热物理备份

-命令行驱动热物理备份

UNIX环境下的热备份是一系列命令。热备份操作顺序：

1）建立需要备份的表空间的列表(DBA_Tablespaces)。

2）对于该列表中的每一个表空间，a.建立一个数据文件列表(DBA_data_files)；b.让表空间处于备份模式下(ALTER TABLESPACE tsname BEGIN BACKUP;)；c.把数据文件列表中的数据文件作备份；d.使该表空间脱离备份模式(ALTER ... END BACKUP)。

3）生成一个“可备份的”控制文件(ALTER DATABASE BACKUP CONTROL FILE TO fname REUSE;)。

4）强制执行日志切换(ALTER SYSTEM SWITCH LOGFILE;)

5）延时等待重做日志得到归档。

6）把归档日志目录备份到一个备份设备中(文件拷贝)。

显然应该在数据库中的活动最少时进行联机备份。

4.3.3.3 ORACLE热物理备份

- 桌面驱动热物理备份

使用Oracle的备份管理器不仅可进行脱机备份，而且也可以进行联机备份。如下步骤执行联机备份：

1）确保Oracle数据库运行正常，打开备份管理器。

2）选择备份方式(Online_selected tablespace)。

3）选择希望备份的表空间。

4）确定备份目标是到磁带还是磁盘中。

5）单击Backup按钮，完成备份。

6）选择备份方式(Online_Control File only)。

7）确定存储控制文件备份的目录名称。

在此过程结束后，备份管理器已经完成了对数据库的备份并且已经使所有的表空间脱离备份模式。

备份管理器不能备份init.ora、config.ora或归档日志文件。

4.3.3.4 用恢复管理器进行物理备份

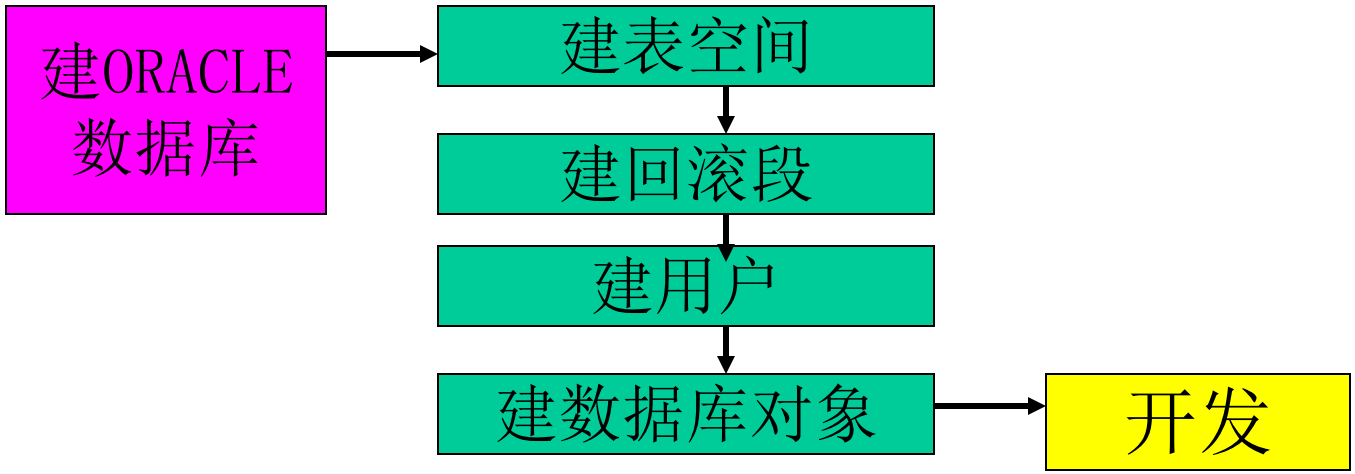
- 恢复管理器（RMAN）是对Oracle 8备份、恢复和重建进行管理的一个软件。
- RMAN的特点：备份能发生在多级上：数据库、表空间或者数据文件级；支持增量备份；备份能够很容易地建立时间表和控制。另外，RMAN能够与已经建立起来的第三方厂商硬件或软件集成。
- RMAN有两个主接口：命令行解释(CLI)接口或OEM图形用户接口(GUI)。CLI接口是一个与SQL*Plus相似的应用。GUI集成在OEM中，并提供能够与第三方应用相链接的API库。
- RMAN提供直接使用BACKUP、RECOVER和RESTORE命令或在脚本中嵌入的能力。

作业:

1. 引起碎片的原因有哪些?如何处置?
2. 管理回退段有哪些工作?
3. 简述资源限制文件的作用.
4. 将没有任何权限的角色授予用户会起什么作用?
5. ORACLE数据备份的方式有哪几种?
6. 简述ORACLE联机备份的机制.

4.4 ORACLE数据库规划与实施

- ORACLE为信息系统提供了一个开放、综合、集成的方法。使用ORACLE数据库进行信息系统开发的步骤可以归纳为如下过程：



- 本节介绍ORACLE数据库的规划与实施：

4.4.1 数据库应用程序类型
4.4.2 ORACLE空间设计
4.4.3 ORACLE安全设计
4.4.4 ORACLE对象设计
4.4.5 DBA工具

4.4.1 数据库应用程序类型

- 规划和配置数据库，需要知道将要操作数据库的事务有什么，这些事务访问数据库的频率有多大。针对不同的事务访问情形，对内存的要求不同，磁盘I/O操作的情形也不同。**只有设置合适的ORACLE实例，才能使数据库应用系统获得快速、准确、可靠的事务响应。**ORACLE既可以设置成处理长的密集型事务的系统，也可以设置成处理大量的小型快速事务的系统。有三种通用的数据库设置类型：
  - 联机事务处理（OLTP）
  - 决策支持系统（DSS）
  - 批量作业系统
  - 混合系统

4.4.1 数据库应用程序类型

- 联机事务处理（OLTP）

OLTP系统是一个包含繁重DML的应用，其面向事务的活动主要包括更新，但也包括一些插入和删除。典型的例子是用于航空公司和旅馆的预定系统。OLTP系统可以允许有很高的并发性。

OLTP的特点是有大量短的事务高频率地访问数据库，每一个事务操作的数据量不是很多，但事务比较频繁，而且要求数据库响应事务的速度快。一般来说OLTP系统需要24*7地响应对数据库的请求。根据这些要求，我们应该从以下几方面考虑：

1）大量的小回滚段。因为事务都不太大，对回滚段空间的需求不可能很大，但事务数量多，引发回滚段数量的增大。应该创建大量小的回滚段，把事务对回滚段的争用减到最小。标准的配置可以考虑每四个活动事务用一个回滚段，每个事务接受一个区间。

4.4.1 数据库应用程序类型

- 联机事务处理（OLTP）

2）OLTP应用的表可能插入或者删除记录，所以存放易变表和索引的表空间应被分配到它们最大的期望容量。

3）适当的估计重写日志缓冲区和日志文件大小，减少日志写和日志切换的频率，尽可能降低对响应事务请求的影响，另外频繁的检查点也可能影响事务响应速度。

4）拥有大型共享池。不同客户可能执行同样的SQL语句，SQL在共享池中缓冲，性能可提高（应用程序的SQL语句应当统一，另外WHERE中使用绑定变量查询而不是直接的值查询，使不同值的查询共享SQL的执行计划）。

5）数据拥有单独的索引。OLTP的事务请求基本在规划设计范围之内，建立单独有效的索引，并在独立的表空间中创建主键和其他索引，且存放在独立的存储器上。

6）使用小型临时段。需要小的排序应在内存中完成，尽量避免OLTP系统对临时段的请求进入磁盘。

4.4.1 数据库应用程序类型

- 决策支持系统（DSS）

DSS系统通常是一个大型的、包含历史性内容的只读数据库，通常用于简单的固定查询或特别查询。夜间处理任务被认为是DSS，**查询（选择）是DSS的主要活动**。根据这些要求，我们应该从以下几方面考虑：

1）拥有少量的较大的回滚段。大型工作要创建大型的回滚段，使用SET TRANSACTION USER ROLLBACK SEGMENT语句使事务强制使用专用回滚段。

2）为分类排序创建大型临时表空间，DSS经常超出内存的使用空间而从内存转入磁盘（临时表空间）。

3）使用较小的共享SQL区域而采用较大的数据缓冲区高速缓冲。DSS中SQL的执行频率并不大，无需考虑SQL语句的共享，但应增加数据库缓冲区高速缓冲的容量，使得更多的数据块和回滚段在内存中高速缓冲。

4.4.1 数据库应用程序类型

- 决策支持系统（DSS）

4）如前所述，DSS希望Oracle块一次读取尽可能多的行。因为DSS查询一般触发整个表扫描，所以希望通过读取多个连续的块使系统读取的块达到最大值。把DB_BLOCK_SIZE和DB_FILE_MULTIBLOCK_READ_COUNT参数设置得尽可能高些。

5）运行的SQL应当删除不必要的排序并充分利用索引，以减少对临时表空间和回滚段的压力。2种方法：在没有可以选择的良好索引时不使用任何索引或屏蔽使用某些索引（参见4.3）；使用位图索引。

6）不要在DSS应用代码中使用绑定变量。在OLTP下，希望最小化应用进程开销(语法分析)。然而，DSS的查询，语法分析占用整个查询时间的比例会更小。假如使用绑定变量，优化程序不能调用它所存储的统计信息（通过ANALYZE命令）以选出存取数据的最好方法。

4.4.1 数据库应用程序类型

- 批量作业系统

批作业处理系统是作用于数据库的非交互性的自动应用。它通常含有繁忙DML语句并有较低的并发性。

另外还有一些其他的应用类型：

OLAP（联机分析处理）系统可提供分析服务。该系统在数学、统计学、集合以及大量的计算方面区别于OLTP或DSS模型。可以把OLAP看作是在OLTP系统或DSS之上的一个扩展或一个附加的功能层次。地理信息系统（Geographic Information Systems，GIS）或有关空间的数据库和OLAP数据库相集成，提供图表的映射能力。例如用于社会统计的人口统计数据库。

VCDB（可变基数数据库），这类数据库通常被用作一个处理系统的数据库后端，这样就会导致在数据处理期间，数据库中的表显著地增长或收缩。基数是指在一个给定时间里一个表中行的数目。

4.4.1 数据库应用程序类型

- 混合系统

混合系统混合了前面介绍的几种类型。许多实际应用系统往往是OLTP和DSS甚至其他模型的集成。

从前面OLTP和DSS的需求对比来看，OLTP和DSS的基本特征似乎相反，如何协调这两种需求的矛盾？一般有三种解决方法：

1）**在一个系统中的OLTP和DSS之间节流**，白天运行OLTP，而夜间进行批量处理。即白天运行OLTP配置的数据库，夜间跳回DSS配置的实例。

这种方法的实际操作模式可以是以下两种：OLTP不支持24*7工作方式，DSS时不支持OLTP；DSS操作的系统反跳之后，OLTP可以使用，但系统性能将受影响。DBA需要测试DSS和服务器反跳对Web用户的影响，必要时选择其他解决办法来调整系统服务模式。

4.4.1 数据库应用程序类型

- 混合系统

2）**同时运行两个数据库，一个服务于OLTP，一个服务于DSS。**OLTP数据库进行实时更新，在有规则的时间间隔内，将变化传递给DSS数据库。

这种方法的实际操作模式可以多种。DBA可以夜间执行部分或全部OLTP系统的输出，并输入到DSS系统。较好的解决方法是对实时方式下传递变化的连接数据库使用复制技术，将变化的数据复制记录下来，OLTP数据库只传递变化的数据给DSS数据库。第三种方法是使ORACLE中OLTP数据库的归档日志自动应用于DSS数据库，保证DSS系统紧跟在OLTP系统之后，DSS数据库不仅作为OLTP的备用数据库，而且自己又独立地服务于决策支持。尽管这种方式可能不能保证OLTP和DSS数据库的绝对同步，但在DSS的实时性要求不是很强时，是一种很好的方法。

4.4.1 数据库应用程序类型

- 混合系统

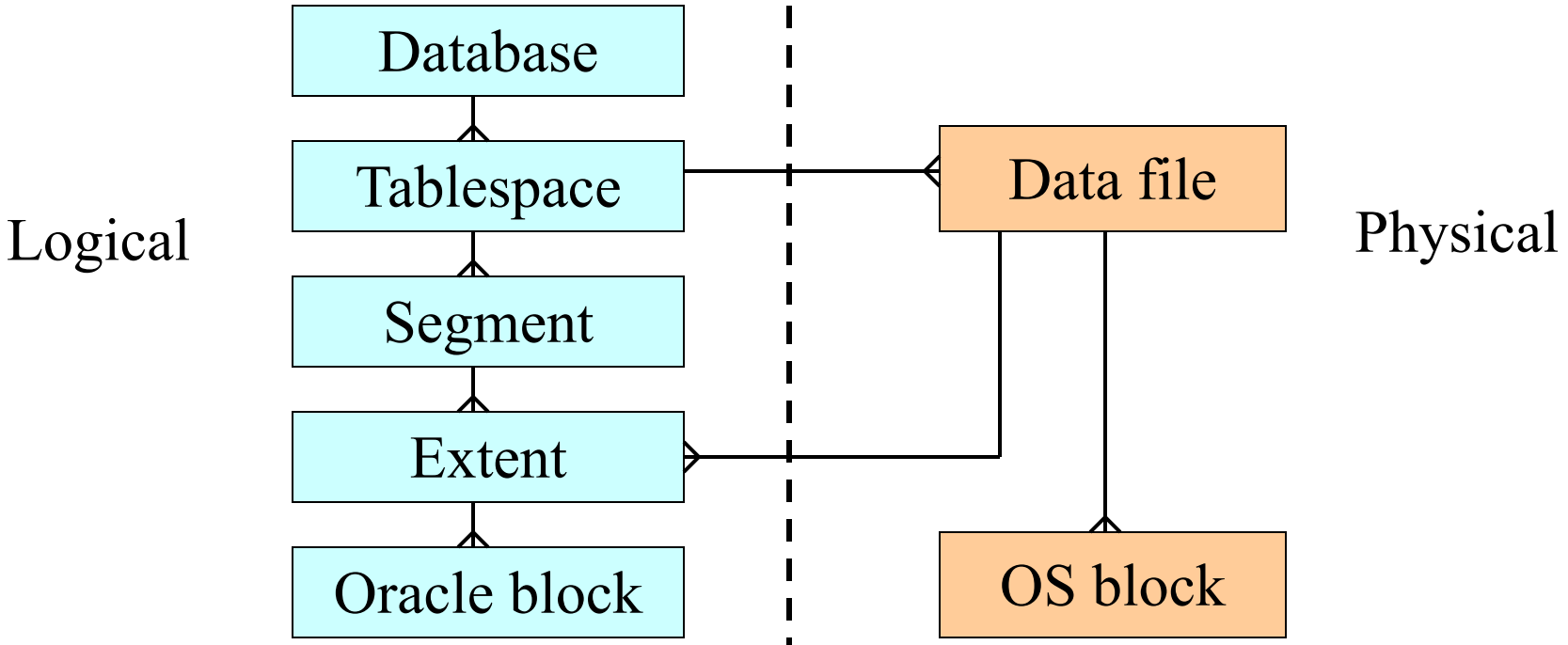
3）**在一个系统中同时运行OLTP和DSS**。这是最普遍的解决方法。系统经常作为OLTP开始活动，逐渐加入DSS需求使系统渐渐成为混合系统。

在这种情况下，DBA必须在OLTP和批量处理之间寻找平衡，并且应该偏向于OLTP用户，创建与OLTP同样多的回滚段，而且要创建少量专门为大型事务指定的大型回滚段，同时将高速缓冲和共享池保持合理的容量，使系统既支持OLTP对共享池的需求又支持DSS对数据库缓冲区高速缓冲的需求。还要确保临时表空间够批量排序使用，同时将OLTP排序控制在内存中。

推荐在这种情况下使用专用服务器，并且配置ORACLE并行服务器(Oracle Parallel Server,OPS)，因为OPS允许多个实例访问同一个物理数据库，一个实例为OLTP配置，另一些为批量处理配置。

4.4.2 ORACLE空间设计

- ORACLE数据库的结构包括逻辑结构和物理结构。在进行应用系统数据库设计时，空间设计是非常重要的，它与今后运行系统的性能、可用、维护、管理等有密切的关系。物理结构与逻辑结构之间的关系如下图：



4.4.2.1 ORACLE表空间设计

- ORACLE表空间分为两大类：系统表空间和非系统表空间。系统表空间是建库时自动创建的主要用于存储数据字典、系统回滚段；非系统表空间是由用户创建的表空间，主要用于分离段，方便用户的数据管理。ORACLE推荐一种最佳灵活结构OFA(Optimal Flexible Architecture)可以用于表空间的设计。**OFA是ORACLE软件安装和数据文件的推荐目录结构**。OFA只是一组建议，并不是绝对原则。下面介绍基本OFA和扩展OFA。
  - 基本OFA

    1）系统表空间（SYSTEM）：系统表空间不具有重建性，只用于存放数据字典，其他内容如数据段信息等都应从系统表空间中移出。

4.4.2.1 ORACLE表空间设计

- 基本OFA

2）分离应用程序数据段（DATA）：应用程序数据段应是一个独立的表空间，即数据表空间。建立单独数据表空间的理由是：应用程序与数据文件分离有利于减少资源争用，简化文件管理；另外，当数据段产生碎片时利于管理。

3）分离应用程序索引段（INDEX）：索引段不应与相关的数据段存储在一个表空间，以免产生资源争用。由于表增长和不正确的尺寸设置可能产生索引段的碎片，分离应用程序索引段，可以减少整理数据表或索引碎片所需的管理代价。从数据表分离相应的索引也可以通过ALTER INDEX命令来实现。

4）分离工具段（TOOLS）：许多第三方工具会将数据存储在系统表空间中，为了避免这种情况，可将它们的对象移出系统表空间。

4.4.2.1 ORACLE表空间设计

- 基本OFA

5）分离回滚段（RBS）：回滚段会针对巨型事务动态扩展尺寸，也会动态收缩到指定的优化尺寸（参见4.3）。回滚段的I/O通常与数据和索引表空间的I/O同步。将回滚段分离出来是出于减少I/O冲突的考虑。

6）分离临时段（TEMP）：临时段是数据库中动态生成的对象，用来存储巨型排序操作（如SELECT DISTINCT，UNION，CREATE INDEX等）的数据。由于动态特性，临时段不应与其他类型段一起存储。通常，在建立用户时，将这些用户使用的临时数据段设置到临时表空间。

7）分离用户（USER）：为用户帐号建立一个USER表空间，在建立用户时将用户的操作数据放在用户表空间。

以上7个表空间是最基本的表空间，也是传统OFA的基本组成部分。随着应用的深入，又有扩展的OFA。

4.4.2.1 ORACLE表空间设计

- 扩展的OFA

1）分离低使用数据段（DATA_2）：在基表中，有一些是动态数据，一些是静态数据，而动态数据和静态数据常常被并发操作。静态数据在实际使用时只是被提取信息，可以把这些静态数据表设置到一个专用表空间（DATA_2），从而避免并发操作冲突。

2）分离低使用索引段（INDEX_2）：数据表分为静态表和动态表，则索引亦如此。对于静态表的索引表应建立静态索引表空间（INDEX_2）。如果索引已经生成，可以使用ALTER INDEX语句来移动这些索引到一个新的表空间中。

3）分离工具索引（TOOLS_1）：如果数据库对TOOLS表空间有较多的操作，则这些工具表索引可以移到另一个表空间（TOOLS_1），这样TOOLS表空间就可看成工具数据表空间。

4.4.2.1 ORACLE表空间设计

- 扩展的OFA

4）分离特殊回滚段（RBS_2）：RBS表空间的回滚段必须有适当的大小和数量，以适应应用的使用。但大型事务可能超过回滚段的配置。为了避免这种情况，可以设置一个特殊回滚段，使大型事务只使用特殊回滚段。建立一个特殊表空间来容纳这个特殊回滚段。

5）分离特殊用户临时段（TEMP_USER）：和特殊回滚段一样，一些特殊用户在做特殊操作时，可以设置一个特殊用户临时表空间，这样便于系统管理。

- 附加的特殊应用OFA扩展

1）SNAPS：用于快照(Snapshots)。快照表是分布式数据库处理的一种特殊表，为此类表建立相应的表空间。

2）PARTITIONS：用于分区(Partition)。分区是为了分配I/O并且简化巨型表的管理。

3）TEMP_WORK：用于大型数据装载。

4.4.2.2 数据文件设置

- 根据ORACLE数据库结构特点，每个表空间至少要有一个数据文件，因此，应用系统的每个表空间均具有各自的数据文件。Oracle 7.2以后可以支持数据文件大小的重新定义，当表空间需要扩充时，不必为其增加数据文件，而简单地将其扩充即可，避免了预先分配过大的存储空间及过多数据文件的问题。同时可以支持数据文件的自动扩展，避免当表空间已满时，DBA手工扩充表空间对用户使用的影响。
- 对于一个不太大的数据库，可以将数据库软件和数据文件都放置在一个磁盘上，将它们设计成不同的子文件即可。对于大型数据库，需要指定哪些文件放置在哪个磁盘上。

4.4.2.2 数据文件设置

- 表空间的磁盘分配：

一般情况下，90%以上的I/O操作集中在系统表空间、数据表空间、回滚表空间、索引表空间这4个表空间上，所以**这4个表空间应单独存储在不同的磁盘上**。

- 系统文件的规划：

ORACLE有联机日志文件、归档日志文件、控制文件和系统软件文件等，如何放置这些系统文件也是一件重要的任务。

1）联机日志文件：每个数据库至少要有两个联机日志文件，由于联机日志文件中保留了数据库的当前事务，所以它无法从备份中恢复，他们是ORACLE无法用备份工具恢复的唯一文件类型。这类文件应单独存储。

4.4.2.2 数据文件设置

- 系统文件的规划：

2）控制文件：前面章节已经说明了控制文件的重要性，每个数据库应当至少有3个控制文件的拷贝，它们被分布在3个驱动器中。

3）归档日志文件：如果归档日志文件和联机日志文件放在同一个磁盘上，当LGWR在读盘时，ARCH也在读同一磁盘，势必造成I/O冲突，所以归档日志文件和联机日志文件应分盘存储。

4）ORACLE软件：Oracle软件是系统运行的核心，这些软件应放在独立的磁盘上。

- 磁盘规划设计：

磁盘规划设计可以依据下列准则：数据库必须可恢复；必须对联机日志文件进行镜像；必须最小化DBWR、LGWR、ARCH之间的冲突；必须最小化DBWR磁盘间的冲突；估计数据库文件的I/O量。

4.4.2.2 数据文件设置

- 磁盘规划设计：

在磁盘规划设计过程中需要做一些数据库测量，以估计其I/O量，以下给出一参照的7磁盘设计方案：

磁盘	存放内容
1	Oracle软件
2	SYSTEM表空间，控制文件1
3	RBS，TEMP，TOOLS表空间，控制文件2
4	DATA表空间，控制文件3
5	INDEX表空间
6	联机日志文件1，2，3，转储文件
7	应用软件，归档日志文件

4.4.2.2 数据文件设置

- 表空间与数据文件设置举例：

```
connect system/manager
create tablespace tsp_acct
datafile  'd:\oracle\oradata\orcl\apacct.dbf'
size 32M  reuse autoextend on next 4M maxsize unlimited;
create tablespace tsp_public
datafile  'd:\oracle\oradata\orcl\appublic.dbf'
size 100M reuse autoextend on next 16M maxsize unlimited;
create tablespace tsp_temp
datafile  'd:\oracle\oradata\orcl\aptemp.dbf'
size 500M reuse autoextend on next 32M maxsize unlimited;
```

连接数据库

创建tsp_acct表空间

创建回滚段表空间

创建临时表空间

4.4.2.3 段存储分配设置

- 在数据库中，大量使用的段包括表段和索引段，还有回滚段、临时段。表的逻辑定义信息放在DBA_TABLES和DBA_TAB_COLUMNS中，索引的定义信息放在DBA_INDEXS和DBA_IND_COLUMNS中。从存储管理角度看，表和索引都属于段，只是类型不同而已，有关段及存储定义的系统数据字典有DBA_SEGMENTS和DBA_EXTENTS等。
- **在应用系统中，对大量使用的段的规划主要是它们的存储方案的设计**，每个段由区间组成。每个段使用的空间由它的存储参数决定，在CREATE TABLE、CREATE INDEX、CREATE CLUSTER、CREATE ROLLBACK等命令中都涉及到STORAGE参数。

4.4.2.3 段存储分配设置

- 如果在创建段的命令中不指定STORAGE参数，则使用它所在表空间的缺省参数。表空间的存储参数或段的存储参数设计不合理将直接影响数据库的性能，甚至直接影响到数据库的正常运行。

  对存储参数的确定问题，前面的讨论中已经说明，应根据实际初始数据量的估算和数据的动态变化情况以及段的性质等多方面来决定。

  注意，ORACLE数据库在创建回滚段之后，状态为脱机，因此必须由DBA将其状态改为联机，方可使用。

4.4.2.3 段存储分配设置

- 段存储分配设置举例:

```
create table pat_master
(pat_id varchar2(10),name varchar2(8),...)
pctfree 10 pctused 60
storage (initial 16M next 16M minextents 1 maxextents
   unlimited pctincrease 0)
tablespace tsp_acct;
create index pat_master_index
on pat_master (pat_id) pctfree 20
storage (initial 4M next 4M minextents 1 maxextents
   unlimited pctincrease 0)
tablespace tsp_acct;
create public rollback segment rbs_public
tablespace tsp_public storage (initial 4M next 4M);
Alter rollback segment rbs_public online;
```

创建pat_master表

创建pat_master_index索引

创建回滚段

4.4.2.4 数据块利用设置

- 数据块空间利用参数可以控制表段和索引段中的空间使用。分两类:控制并发性的参数(INITRANS和MAXTRANS)、控制数据空间使用的参数(PCTFREE和PCTUSED)。
- 可以使用INITRANS和MAXTRANS来确定一个数据库块上活动事务的个数。INITRANS是每个块的初始化事务入口数，数据段和索引段的INITRANS的缺省值分别为1和2，增大其值则允许多个事务同时操作数据块。MAXTRANS是在每个块的最大事务入口，缺省值为最大值255。

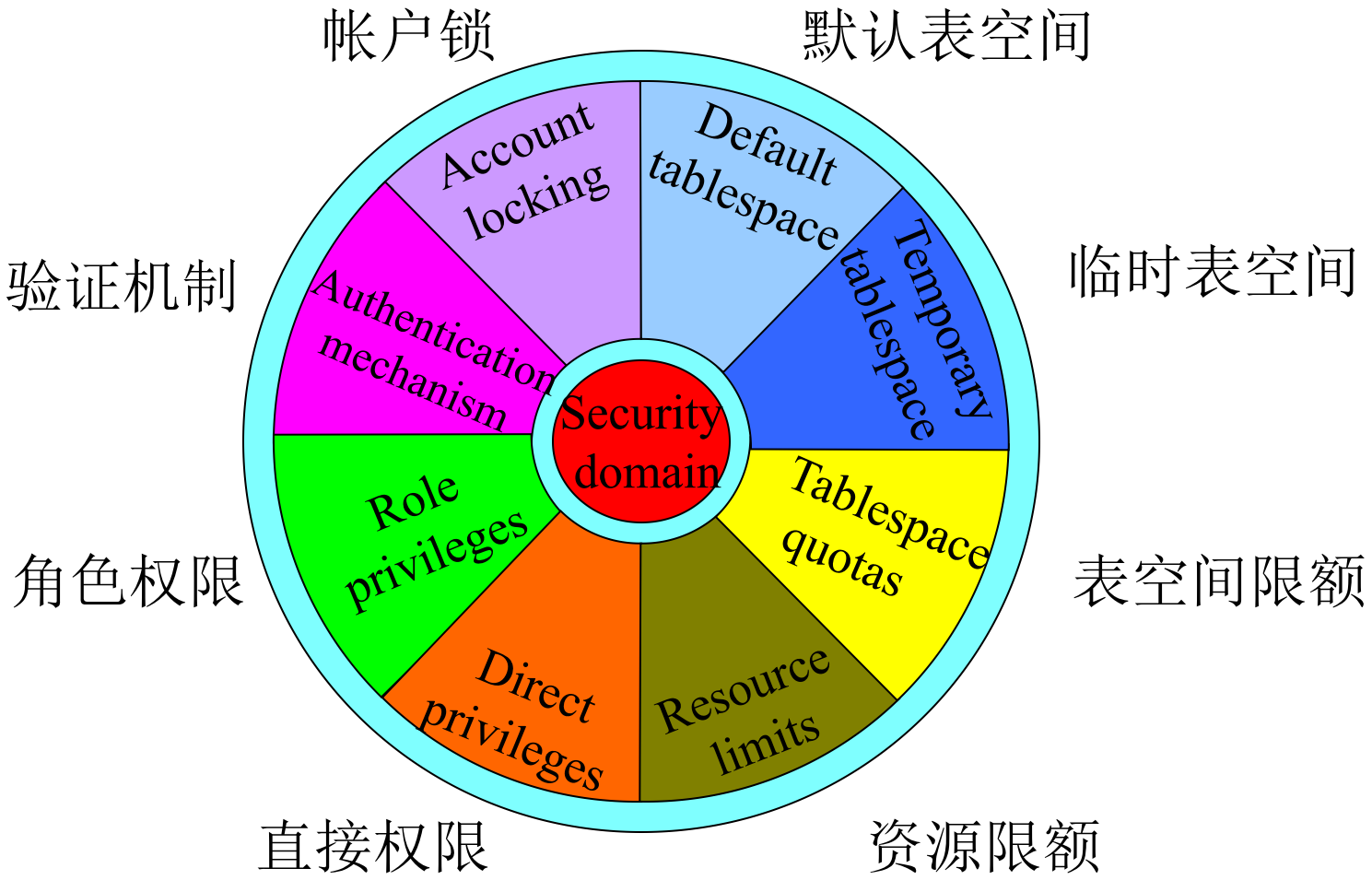
4.4.2.4 数据块利用设置

- 通过选择适当的PCTFREE和PCTUSED可以提高性能和空间利用。对于改动较少的表，在设置数据块空间利用参数时，PCTFREE适当降低。对于经常改动的表，尤其是数据行长度增加的改动，应该适当提高PCTFREE的设置，以免发生行转移。数据行长度增加这类应用的一个例子：医疗信息系统中，病案首页表中行的信息开始长度可能不大(许多字段值为空)，但随着病人住院的过程，行的信息不断增加。

4.4.3 ORACLE安全设计

- 在ORACLE中，DBA定义访问数据库的用户名，安全域定义作用于用户的有关安全方面的设置。
- 安全域包括：验证机制、表空间限额、默认表空间、临时表空间、帐户锁、资源限制、直接权限、角色权限8个方面，如下页图。ORACLE的安全性设计围绕用户的安全域展开。

4.4.3 ORACLE安全设计



4.4.3 ORACLE安全设计

- 安全域内容

1)验证机制：用户访问数据库可以通过数据库、操作系统、网络这三种方式中的一种进行验证，一个用户究竟使用何种验证方式，根据具体需求决定。

2）表空间限额：表空间限额，可以控制一个数据库的用户对数据库中表空间的物理存储空间的分配数量。

3）默认表空间：定义用户在创建段时的存储位置，当用户创建段时，如果指定存储表空间，则使用定义段时指定的表空间，否则使用这一默认表空间。

4.4.3 ORACLE安全设计

- 安全域内容

4）临时表空间：当用户执行排序操作时，临时表空间定义临时段分配的位置。

5）帐户锁：帐户锁可以防止非法用户对系统的登录。

6）资源限制：对用户使用资源进行一定的限制。

7）直接权限：通过GRANT直接授予用户的系统权限和对象权限。

8）角色权限：通过GRANT将系统权限和对象权限授予角色，再将角色授予用户，使用户间接获得权限。

4.4.3 ORACLE安全设计

- 口令和资源

配置文件是命名的一组口令和资源限制的设置集合。Oracle中存在一个名为DEFAULT的配置文件，默认配置文件中的所有项目都不受限制，如果创建用户时没有指定配置文件子句，则系统将DEFAULT指定给该用户。

Oracle中允许DBA修改DEFAULT和用户定义的配置文件，用ALTER PROFILE命令。允许删除DEFAULT和定义的配置文件，用DROP PROFILE CASCADE命令。

创建用户时需确定用户名和验证机制、指定默认表空间和临时表空间、确定有关表空间的限额、指定用户的配置文件。

ORACLE应用系统中一般定义一系列的用户和配置文件、定义一系列的角色，然后将角色和配置文件授予（或指定给）用户。

4.4.3 ORACLE安全设计

-权限和角色

Oracle大约有100多种的系统权限，包括诸如创建表空间的系统范围级操作权限、诸如创建表的用户模式内对象管理权限和诸如创建任意表的任意模式内对象管理权限。系统权限和对象权限可以通过GRANT或REVOKE语句授予用户或收回。

角色是被命名的一组相关的权限，角色是属于系统范围的，既不属于某个用户，也不属于某个模式。可以把系统权限和对象权限授予角色，也可以把角色授予另一个角色，最终把角色授予用户，由此方便Oracle的安全管理。

使用角色的步骤是：创建角色、为角色授予相应的权限、将角色授予特定的用户。

Oracle中有一些预定义角色可以使用，例如DBA、SELECT_CATALOG_ROLE等，DBA可以直接使用。

4.4.3 ORACLE安全设计

- 应用系统的用户

应用系统的安全设计一般要对用户进行分类，然后根据用户类型的不同，对其进行设置、维护及管理。

数据库管理员（DBA）：默认的数据库管理员internal、sys和system均具有DBA权限。数据库打开、关闭、恢复等操作时，需使用internal；修改数据库重要参数时使用sys；其他一般管理应使用system。

所有者（OWNER）：对大型的应用系统，往往根据不同的用户所属将对象进行划分，这样不同的用户拥有不同的数据库对象，创建数据库对象的用户称为对象的所有者（模式）。此类用户需要有RESOURCE的权限。

一般用户（USER）：一般用户即应用系统的最终用户。此类用户对数据库而言仅需要有CONNECT的权限即可，但此类用户的角色权限和对象使用权限需要根据具体工作和应用进一步规划和设计。

4.4.3 ORACLE安全设计

- 应用系统的一般用户

应用系统中除创建一系列数据库对象外，还要创建一系列的应用程序，不同的应用程序执行不同的操作。对于一般用户来讲，使用数据库系统是通过使用用户管理应用程序开始的，往往限制其使用一些应用程序的权力。对于这种情形，处理办法是：在数据库和应用程序之间建立用户管理应用程序层次，对每一个用户，授予相应的数据库权限和使用应用程序的权限。

一般用户的配置：系统级权限仅限于角色CONNECT，对象级权限仅限于相关应用系统的自定义角色，应用程序使用权限仅限于相关应用程序的使用级别。

4.4.3 ORACLE安全设计

- 应用系统的一般用户

在应用系统内部进行多个层次的检查和验证：

1）第一层：Oracle数据库系统层，判断其能否建立会话。

2）第二层：应用程序的使用权限，检查验证用户能够使用哪些应用程序。

3）第三层：Oracle数据库的对象级权限，判断用户能否操纵其相应的对象，对象级权限是通过自定义Oracle角色来设置的。

4.4.3 ORACLE安全设计

- 在ORACLE数据库中，当一个数据库用户被创建时，系统就为该用户自动创建一个与用户同名的模式。用户和模式的概念在某种程度上是一样的。模式或用户可以拥有的对象如下图。用户的定义应与数据库的对象定义结合起来。

Schema Objects	
Tables	表
Triggers	触发器
Constraints	约束
Indexes	索引
Views	视图
Sequences	序列发生器
Stored program units	存储过程单元(PL/SQL写)
Synonyms	同义词
User_defined data types	用户定义数据类型(Oracle8i以后)
Database links	数据库连接(分布式数据库时使用)

4.4.3 ORACLE安全设计

- 安全设计举例：

```
create profile grace_5 limit
failed_login_attempts  3
password_lock_time     unlimited
password_lift_time     30
password_grace_time    5
sessions_per_user      2
cpu_per_session        10000
idle_time              60
conenct_time           480;
create user comm identified by comm
default tablespace data
temporary tablespace tsp_temp
quota 15M on data  profile grace_5;
grant connect,resource to comm;
```

创建配置文件

创建用户

授权

4.4.4 ORACLE对象设计

- 数据库应用系统中肯定要建立大量的模式对象，这些对象应作出合理的规划，尽量减少因设计不周导致性能下降、表空间不足、用户权限不够等问题的出现。
  - 表

    在创建表时，需要注意的事项如下：

    1）对于表名、列名、约束名、索引名和聚集名使用符合命名规则的并带有描述意义的名称。

    2）建立相应完全的、详细的文档资料。

    3）所有表都应当规范(规范化至少是第三泛式)。

    4）定义允许为空的列，并保留一定的存储空间。

    5）适当地使用聚集表，保留一定的存储空间。

    6）表段应放在单独的表空间中。

4.4.4 ORACLE对象设计

- 表

7）为了减少存储碎片产生，区间大小最好是数据块的整数倍。

8）对于应用系统中频繁使用的数据行较少的表，考虑使用CACHE选项，即允许其长期保持在内存中，避免反复读取，提高性能。

- 触发器

触发器作为Oracle标准功能的补充提供给用户。使用它通常实现如下功能：

1）自动生成可以派生的列值。

2）防止无效的处理。

3）增强复杂的安全验证。

4.4.4 ORACLE对象设计

- 触发器

4）增强分布式环境中跨节点的引用完整性。

5）增强复杂的业务规则。

6）提供透明的事件登记。

7）提供高级的审计。

8）维护表复制的同步。

9）采集关于访问表的统计信息。

尽管触发器可实现许多功能，但是，**能用其他方式实现的功能尽量避免使用触发器**。过多的触发器导致复杂的内部相关性，维护和管理变得复杂。特别注意触发器的语句不要太多,不要创建递归调用的触发器(如一个对象的AFTER UPDATE的活动中又有该对象的UPDATE操作)。

4.4.4 ORACLE对象设计

- 索引

索引一旦建立，其使用和维护都是由Oracle自动实现的。使用它应注意：

1）索引可以加速查询，但会降低修改速度，频繁改动的表，尽量少建索引。

2）索引段应存储在独立的表空间中。

3）为减少存储碎片，区间大小应是数据块的整数倍。

4）为了避免索引的建立和改动操作生成大量的重做日志，影响系统的性能，在创建较大索引时，使用NOLOGGING选项。

5）由于索引项相对于数据项要短，使一个数据块中容纳索引的数量要比数据行多，为了提高索引在数据块一级上操作的并发性，可适当提高INITANS参数的值。

4.4.4 ORACLE对象设计

- 索引

6）索引中PCTFREE只在创建索引时使用。对于单调增加的索引列值，可以将PCTFREE设置得较低。因为新的索引项始终是对已经存在的索引项的追加，不需要在两个已经存在的索引项之间插入，所以没必要预留过多的空间。如果索引列值是任意值时，则可能需要预留较多的空间。

- 视图

对视图的操作类似表的使用，表可以使用的地方，视图一般也可使用。视图可用于安全控制和简化查询的目的，视图只能创建或删除，不能修改。

4.4.4 ORACLE对象设计

- 序列

应用系统中，经常使用顺序号，并发用户中如何保证顺序号的连续且不重复。序列发生器是数据库的一种模式对象，可以解决上述问题，为应用系统开发人员提供顺序号的自动管理与维护。Init.ora中的参数SEQUENCE_CACHE_ENTRIES与序列发生器有关，影响内存中可以保留的顺序号个数。

- 存储过程

可以调用存储过程的环境有：另一个存储过程的过程体或一个触发器中，一个存储函数可以从一个SQL语句中调用，其使用方式与调用Oracle内置的SQL函数是一样的。创建存储过程时，应当注意该存储过程的模式归属。

- 同义词

利用同义词可以将应用系统中模式对象的模式屏蔽掉。

4.4.3 ORACLE对象设计

- 对象设计举例：

创建表

```
create table emp
(empno number(5) primary key,ename varchar2(15) not null,
 deptno number(3) not null constraint dept_fkey references dept)
 pctfree 10 pctused 40 tablespace users
 storage(initial 50K next 50K maxextents 10 pctincrease 25);
```

创建触发器

```
create or replace trigger inp_bill_detail_insert
after insert on inp_bill_detail for each row
begin
update pats_in_hospital set
 total_costs=decode(total_costs,null,0,total_costs)+:new.costs,
 total_charges=decode(total_charges,null,0,total_charges)+:new.charges
where patient_id=:new.patient_id and visit_id=:new.visit_id;
end; /
```

创建视图

```
create view exam_item_dict as
 select item_code,item_name from clinic_item_dict where
  item_class= 'D' ;
```

授权

```
grant select on exam_item_dict to public;
```

4.4.5 DBA工具

- 通常情况下，DBA的操作都是以命令行的方式进行，尤其对复杂任务。企业管理器(Oracle Enterprise Manager,OEM)给DBA提供了图形界面。

Database Administration Tools	
Tool	**Description**
Oracle Universal Installer(OUI)	**Used to install,upgrade,or remove software components**
Oracle Database Configuration Assistant	**A graphical user interface tool that interacts with the OUI,or can br used independently, to create,delete or modify a database**
Password File Utility	**Utility fro creating adatabase password file**
SQL*Plus	**A utility to access data in an Oracle database**
Oracle Enterprise Manager	**Agraphical interface used to administer, monitor,and tune one or more database**

4.4.5 DBA工具

DBA Tools (Standard application that can be launched from OEM)	
Tool	**Description**
Instance Manager	**Performs startup,shutdown and monitor databases**
Security Manager	**Used to manage users and privileges**
Storage Manager	**Maintains tablespaces,data files, rollback segments and log groups**
Schema Manager	**Used to create and maintain objects such as tables,indexes,and views**
SQL*plus Worksheet	**Provides the capability to issue SQL statements against any database**

作业:

1. 简述使用Oracle数据库的步骤.
2. 数据库应用程序的类型有哪些?混合系统的数据库规划应注意哪些方面?
3. 空间设计包括哪些方面?
4. 安全性设计包括哪些方面?
5. 表对象的设计应注意哪些方面?

5. 数据库设计

- 计算机信息系统以数据库为核心，在数据库管理系统（DBMS）的支持下，进行信息的收集、整理、存储、检索、更新、加工、统计和传播等操作。
- DBMS提供的功能可以细化为:
  - 数据存储、检索、更新
  - 用户可访问的目录
  - 事务支持
  - 并发控制服务
  - 恢复服务
  - 授权服务
  - 对数据通信的支持

5. 数据库设计

- 完整性服务
- 数据独立性服务（视图、表空间）
- 实用工具服务

- 数据库设计是指对于一个给定的应用环境，提供一个确定最优模型与处理模式的逻辑设计，以及一个确定数据库存储结构与存取方法的物理设计，建立起既能反映现实世界信息和信息联系，满足用户数据要求和加工要求，又能被某个数据库管理系统所接受，同时能实现系统目标，并有效存取数据的数据库。

5.1 数据库设计过程

5.2 实体联系模型

5.3 关系规范化

5.1 数据库设计过程

- 数据库系统设计是在现成的DBMS上建立数据库应用系统的过程。其特点是：数据量大、保存时间长，数据关联复杂，用户要求多样化。
- 从系统开发角度来看，数据库系统具有结构特性和行为特性两个方面。
  - 结构特性设计（数据库设计）

    1）是与数据模型所反映的实体及实体之间的联系的静态模型的设计。

    2）设计各级数据库模式，决定数据库系统的信息内容。
  - 行为特性设计（数据库应用软件的开发）

    1）与数据库状态转换有关，是改变实体特性的操作。

    2）决定数据库系统的功能，是事务处理等应用程序的设计。

5.1 数据库设计过程

- **现代数据库设计方法强调数据库的结构设计和行为设计相结合。**
- 数据库应用系统设计中的主要困难和问题
  - 懂计算机、数据库原理和熟悉DBMS，同时又具备企业业务知识和实际经验的人很少。
  - 企业或组织的数据库应用系统的目标和需求缺少明确的规定。
  - 缺乏完善的设计工具、方法和理论。
  - 随应用范围的扩大和深入，用户不断要求修改和增加新的功能。

5.1 数据库设计过程

- 对数据库设计人员的要求
  - 计算机科学基础知识和程序设计技术
  - **数据库基本知识和设计技术以及某一种DBMS**
  - 软件工程的原理和方法
  - **应用领域的知识**
- 开发方法（多种多样）
  - 快速原型法
  - 直接设计法

5.1 数据库设计过程

- 开发方法
  - 软件工程分步设计法(生命周期法)

软件：程序及其开发、使用和维护所需的所有文档的总和。

软件工程：是为了组织大型软件生产，克服手工作坊方式软件生产的缺点，而提出的一种科学的软件设计方法。是指导计算机软件开发和维护的工程学科。

对于大规模、十分复杂、要求较高的数据库应用系统，应当采用软件工程的方法。

5.1 数据库设计过程

- **数据库应用系统的开发是一项系统工程**，系统工程是为了合理进行开发、设计和运用系统而采用的思想、步骤、组织和方法的总称。按照系统工程的观点，数据库系统的设计与开发有如下指导方针。
  - 1) 系统的目的性：系统开发的成功与否取决于是否符合用户的需要，满足用户的要求是设计与开发工作的出发点和归宿。
  - 2) 系统的整体性：对各个环节的信息进行综合和抽象，得出现实世界业务环节的整体逻辑模型和整体物理模型。而不是各个环节信息的简单组合和拼凑。

5.1 数据库设计过程

- 数据库系统的设计与开发指导方针
  - 3) 系统的相关性：组成系统的各个子系统(模块)各有其独立功能，同时又相互依赖，相互作用，通过信息流把它们的功能联系起来。
  - 4) 系统的扩展性和易维护性：要适应外界环境的变化，对数据库的设计要充分考虑留有余地，可扩充。系统要能跟外界进行信息交换，有行业规范或国家标准的尽量采用，对没有标准的可以考虑建立标准；系统要有前瞻性，对可能提出的信息需求和功能需求，系统应留有接口；对可能发生的误操作或故意破坏，尽可能把损失降低到最小程度。

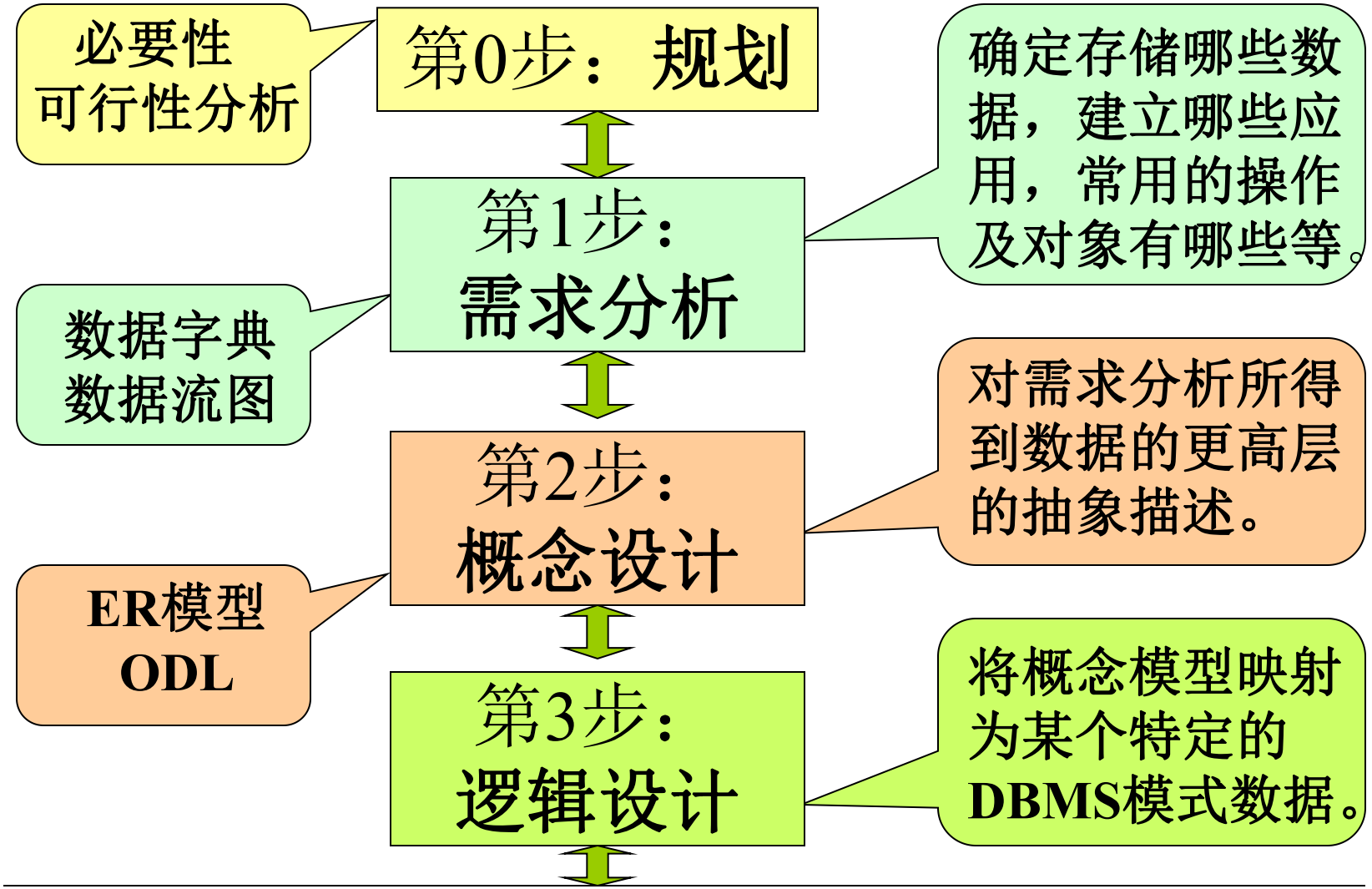
5.1 数据库设计过程

- 数据库系统的设计与开发指导方针
  - 5) 工作成果文档化，图表规范化：软件是程序和文档资料的集合，文档资料是系统的组成部分。因为系统的扩充、修改、交流是建立在文档资料基础之上，而不应该是拿程序进行交流。在文档资料的标准规范化方面可以参照软件工程规范。

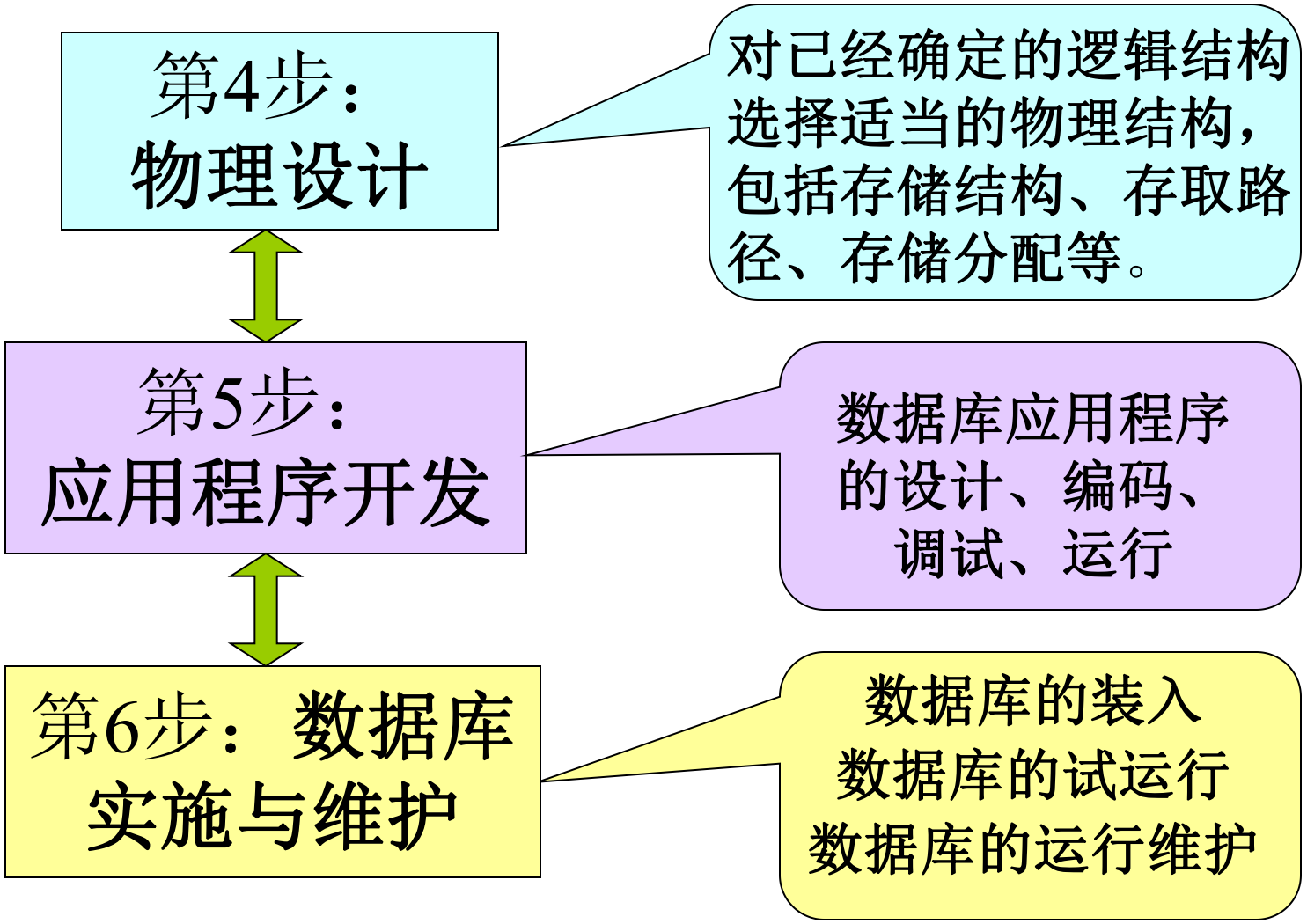
5.1 数据库设计过程

- 数据库设计准则
  - 数据库必须正确反映现实世界，能为某个DBMS所接受。
  - 应用系统有良好的性能，有利于实施和维护。
  - 数据库能满足当前和今后相当长时期内的数据需求，使数据库有较长的使用寿命。
  - 当软件和硬件环境发生变化时容易修改和移植。
  - 能满足安全性要求，当系统发生故障时，容易恢复数据库。
  - 数据库存取效率、查询效率要高。

5.1 数据库设计过程


必要性
可行性分析
第0步：规划
确定存储哪些数据，建立哪些应用，常用的操作及对象有哪些等。
第1步：
需求分析
数据字典
数据流图
对需求分析所得到数据的更高层的抽象描述。
第2步：
概念设计
ER模型
ODL
将概念模型映射为某个特定的DBMS模式数据。
第3步：
逻辑设计

5.1 数据库设计过程


第4步：
物理设计
对已经确定的逻辑结构选择适当的物理结构，包括存储结构、存取路径、存储分配等。
第5步：
应用程序开发
数据库应用程序的设计、编码、调试、运行
第6步：数据库实施与维护
数据库的装入
数据库的试运行
数据库的运行维护

5.1 数据库设计过程

- 数据库设计过程是有一个起点和几乎无止镜的逐步求精的反复过程，尽管前面我们把数据库设计过程表述为一个程序化的过程，但需强调：数据库设计过程并不一定非要以这种程序化方式进行，各个阶段之间的区分也并不是非常严格。
- 本讲义给出的数据库设计过程可以作为引导你进行有效的数据库设计的框架。

5.1.1 规划

- 在数据库设计的规划阶段主要进行建立数据库系统的必要性及可行性分析。
- 规划阶段必须完成下列任务：确定系统的范围，任务陈述定义数据库应用程序的主要目标，每个任务标识数据库必须支持的特定任务，确定开发工作所需的资源，估计开发成本，确定项目进度。
- 系统规划纲要内容包括：
  - 1) 系统目标与范围的描述：
    - 确定系统长期目标和近期目标；
    - 定义数据库应用程序的范围和边界（包括主要的用户视图，用户视图定义了根据要存储的数据和在数据上要执行的事务，也即用户要怎样使用数据，不仅考虑当前用户视图，而且也要考虑未来的用户视图）；

5.1.1 规划

- 确定系统与外部环境的信息联系和接口；
- 系统的主要功能和结构。

- 2) 系统运行环境描述：
  - 管理思想及管理方法的设想（新的数据库系统的运行可能是传统业务的彻底革新，新的思想、新的机制应当是完备而可行的，并且是高效而经济的）；
  - 说明系统运行的基本要求及条件（如何收集数据、如何指定格式、需要什么样的文档资料）；
  - 信息范围、信息标准（良好的信息标准为衡量系统质量和训练职工提供基础）、信息来源、人力资源、设备资源。
- 3) 确定计算机系统选型要求：包括主机及外部设备配置要求、性能指标、网络系统配置、系统软件配置要求。
- 4) 系统开发进度计划（初步）。

5.1.1 规划

- 可行性分析应从经济方面、技术方面、系统运行方面(管理体制、人员的适应性及法律法规)进行分析和评价。可行性研究报告的内容包括：
  - 1)原来系统概况。包括企业目标、规模、组织结构、人员、设备、效益等。
  - 2)原来系统存在的主要问题和主要信息要求。
  - 3)待开发系统的总体方案（如前所述）。
  - 4)经济可行性分析包括系统的投资、运行费用、经济效益及社会效益。
  - 5)技术可行性分析主要包括对技术的评估。
  - 6)系统可行性分析分析系统运行对管理思想、管理体制和变革的要求，系统运行和有关法律法规的相互作用。
  - 7)结论。对可行性研究的简要总结。

5.1.2 需求分析

- 需求分析是数据库设计成败的关键，没有对系统充分的需求分析，数据库设计很难取得成功。需求分析阶段应该对系统的整个应用情况作全面、详细的调查，收集支持系统总的设计目标的基础数据和对这些数据的要求，确定用户的需求，并把这些要求写成用户和数据库设计者都能够接受的文档。
- 设计人员还应该了解系统将来要发生的变化，收集未来应用所涉及的数据，充分考虑到系统可能的扩充和变动，使系统设计符合未来发展的趋向，以减少系统维护的代价。

5.1.2 需求分析

- 实际上，系统的需求包括着多个层次，不同层次的需求从不同角度与不同程度反映着细节问题。没有一个清晰、毫无二义的‘需求’存在，真正的‘需求’实际上在人们的脑海中。任何文档形式的需求仅仅是一个模型，一种叙述。
- 需求分析文档要尽可能地从**不同层次角度去描述**（有可能冗余），要注意描述的名词对用户、分析者、设计者在理解上务必达成共识。
- 在此阶段，从多方面对整个组织进行调查，收集和分析各个应用主要集中在对**信息**和**处理**两方面的需求。

5.1.2 需求分析

- **信息需求**是指用户要从数据库获得的信息内容和用户产生的信息内容，**处理需求**是指完成什么处理功能及处理方式，处理需求即事务需求，信息需求和处理需求组成用户视图。另外还有安全性和完整性要求。对收集到的数据进行抽象，抽取共同的本质特性，并用各种概念精确地加以描述。
- 需求分析阶段的方法：
  - 检查文档资料：收集检查与目前系统相关的文档资料、表格、报告、文件
  - 面谈：找出事实、确认事实、澄清事实
  - 观察业务处理：跟班作业

5.1.2 需求分析

- 研究：从计算机行业杂志、参考书和Internet上获取其他人解决问题的信息，也可能直接得到相关问题的解决方案或软件包。
- 问卷调查：调查建议和意见，诸如：“当前使用什么报表，是否存在问题”等等。

- 需求分析阶段的工作分为六个步骤：
  - 分析用户活动，产生业务流程图
  - 确定系统范围，产生系统范围图
  - 分析用户活动所涉及的数据，产生数据流图
  - 分析系统数据，产生数据字典
  - 功能分析
  - 功能数据分析

5.1.2 需求分析

- 分析用户活动，产生业务流程图

了解用户当前的业务活动和职能，理清其处理流程。把用户业务分成若干个子处理过程，使每个处理功能明确、界面清楚，画出用户活动图(业务流程图)。

- 确定系统范围，产生系统范围图

在和用户经过充分讨论的基础上，确定计算机所能进行数据处理的范围，确定**哪些工作由人工完成，哪些工作由计算机系统完成**，即确定人机界面。

- 分析用户活动所涉及的数据，产生数据流图

深入分析用户的业务处理，以数据流图(Data Flow Diagram,DFD)形式表示出数据的流向和对数据所进行的加工。**DFD有四个基本成分：数据流、加工或处理、文件、外部实体**。DFD可以形象地表示数据流与各业务活动的关系，它是需求分析的工具和分析结果的描述手段。

5.1.2 需求分析

- 分析系统数据，产生数据字典

仅仅有DFD并不能构成需求说明书，DFD只表示出系统有哪几部分组成和各个部分之间的关系，并没有说明各个成分的含义。数据字典提供对数据库时间描述的集中管理，它的功能是**存储和检索各种数据描述**(元数据Metadata)，数据字典是数据收集和数据分析的主要成果，在数据库设计中占有很重要地位。

数据字典编写的基本要求是：

a. 对数据流程图上各种成分的定义必须明确，易理解，唯一。

b. 命名、编号与数据流程图一致。

c. 符合一致性与完整性的要求，对数据流程图上的成分定义与说明无漏项，无同名异义或异名同义。

d. 格式规范，文字精炼，符号正确。

5.1.2 需求分析

- 分析系统数据，产生数据字典

数据字典中通常包括数据项、数据结构、数据流、数据存储、处理过程5个部分。

a. 数据项：数据项是数据的最小组成单位，若干个数据项可以组成一个数据结构，数据字典通过对数据项和数据结构的定义来描述数据流以及数据存储的逻辑内容。对数据项描述包括数据项名、含义、别名、类型、长度、取值范围以及与其他数据项的逻辑关系，必要时说明数据项的数值含义。

b. 数据结构：数据结构名、含义及组成部分。

c. 数据流：表示某一加工处理过程的输入或输出数据，包括数据流名、说明、来源、去向、包含的数据项或数据结构名称、单位时间传输次数(活动频率)。

5.1.2 需求分析

- 分析系统数据，产生数据字典

d. 数据存储：名称、输入、输出、数据量、存取频率和存取方式(批处理或联机处理；查询或更新；顺序或随机)。

e. 处理过程：名称、输入、输出、频率、数据量、处理逻辑说明和响应时间等。

- 功能分析

数据库的设计是与应用系统的设计紧密结合的过程，离开一定的功能，数据库就失去其存在价值。数据库设计的一个重要特点是结构(数据)和行为(功能)的结合。用户希望系统能提供的功能必须有一个清晰的描述。

5.1.2 需求分析

- 功能分析

功能分析是对数据流程图中的处理过程作详细的说明。用户的数据处理可以分为数据输入输出处理、数学处理、逻辑判断三类。对输入输出处理要确定输入输出的格式、输入输出方式；对数学处理要建立处理的数学模型(公式及数值分析)；对逻辑判断的描述通常采用决策树、决策表及自然语言或形式语言等方法。在这三类用户数据处理的基础上进行抽象和归纳，得出功能的层次体系结构。功能分析可以采用软件结构图或模块图来表示系统的层次分解关系、模块调用关系。

功能分析建立在用户需求和数据分析基础上，它通常是系统模块划分和应用程序菜单设置的依据。

5.1.2 需求分析

- 功能数据分析

反映系统全貌的数据流程图与数据、功能详细分析完成后，为保证总的系统描述和细节情况相一致，需要进行整理和审核，这一过程称为功能数据分析。

功能数据分析可以使用填写数据**功能格栅图**等方法，如果完成某功能所需的数据不存在，需在数据字典中添加项目；如果数据字典中的数据没有任何一个功能使用，那么它可能是多余的或者在功能分析中有遗漏。

通过功能数据分析的最后大检验，使需求分析报告中的内容祥实准确。用户需求的功能在需求分析报告中有所描述，完成功能所需的数据在数据字典中有所描述，为数据库的设计和应用程序的设计工作打下坚实基础。

5.1.3 概念设计

- 在进行数据库设计时，如果将现实世界中的客观对象直接转换为机器世界中的对象，注意力往往被转移到更多的细节限制方面，而不能集中在最重要的信息的组织结构和处理模式上。
- 通常将现实世界中的客观对象首先抽象为不依赖任何机器的信息结构，这种信息结构不是具体的DBMS的数据模型，而是基于现实世界和机器世界的中间层次，称为**概念模型**。逻辑设计再把概念模型转换成具体DBMS支持的数据模型。

5.1.3 概念设计

- 广泛被采用的能够很好表达概念模型的方法是设计实体联系模型(Entity Relationship Model, ER模型)。对象定义语言(Object Definition Language,ODL)是以面向对象的观点、方法说明数据库的概念结构，也可表达概念模型，特别是对象数据库的情形。
- 实体联系模型定义实体、属性和联系等数据对象，数据库设计者将需求分析过程中产生的数据项和数据结构归纳到这些类别中，通过对实体联系图的分析把握现实世界中信息的组织结构。

5.1.3 概念设计

- ER模型又比较容易转换为机器模型（ER模型内容参见5.2）。
- 设计概念模型的策略有自顶向下、自底向上、由里向外、混合策略等。
- 利用ER方法进行数据库的概念设计，采用**自底向上策略**，可以分成三步进行：
  - 设计局部ER模型
  - 设计全局ER模型
  - 全局ER模型的优化
  - 概念模型评审

5.1.3 概念设计

- 设计局部ER模型

一个数据库系统是为多个不同用户服务的，各个用户对数据的观点可能不一样，信息处理需求也可能不同。先分别考虑各个用户的信息需求，形成局部ER图。大体分为四个环节：确定局部结构范围、局部实体定义、局部联系定义、属性分配。

5.1.3 概念设计

- 设计全局ER模型

各个局部ER图设计完成之后，下一步是把它们综合成单一的全局ER模型。全局ER模型不仅支持所有局部ER模式，而且必须合理地表示一个完整、一致的数据库概念模型。设计过程为：确定公共的实体类型，合并局部ER图并且消除在局部ER图中的属性冲突、结构冲突和命名冲突。

5.1.3 概念设计

- 全局ER模型的优化

在得到全局ER图后，对它进行优化，一个好的全局ER模型，除能准确、全面地反映用户功能需求外，还应满足：实体类型尽可能少；实体类型所含属性个数尽可能少；实体间联系无冗余。但这些条件并不是绝对的，视具体情况而定。

5.1.3 概念设计

- 概念模型评审

评审分为用户评审与DBA及应用开发人员评审两部分。用户评审重点放在确认全局概念模式是否准确完整反映了用户的信息需求和现实世界事物的属性间的固有联系，DBA和应用开发人员评审侧重于完整性、一致性。被评审的文档资料应包括局部概念结构描述、全局概念模式描述、数据清单和业务功能清单。

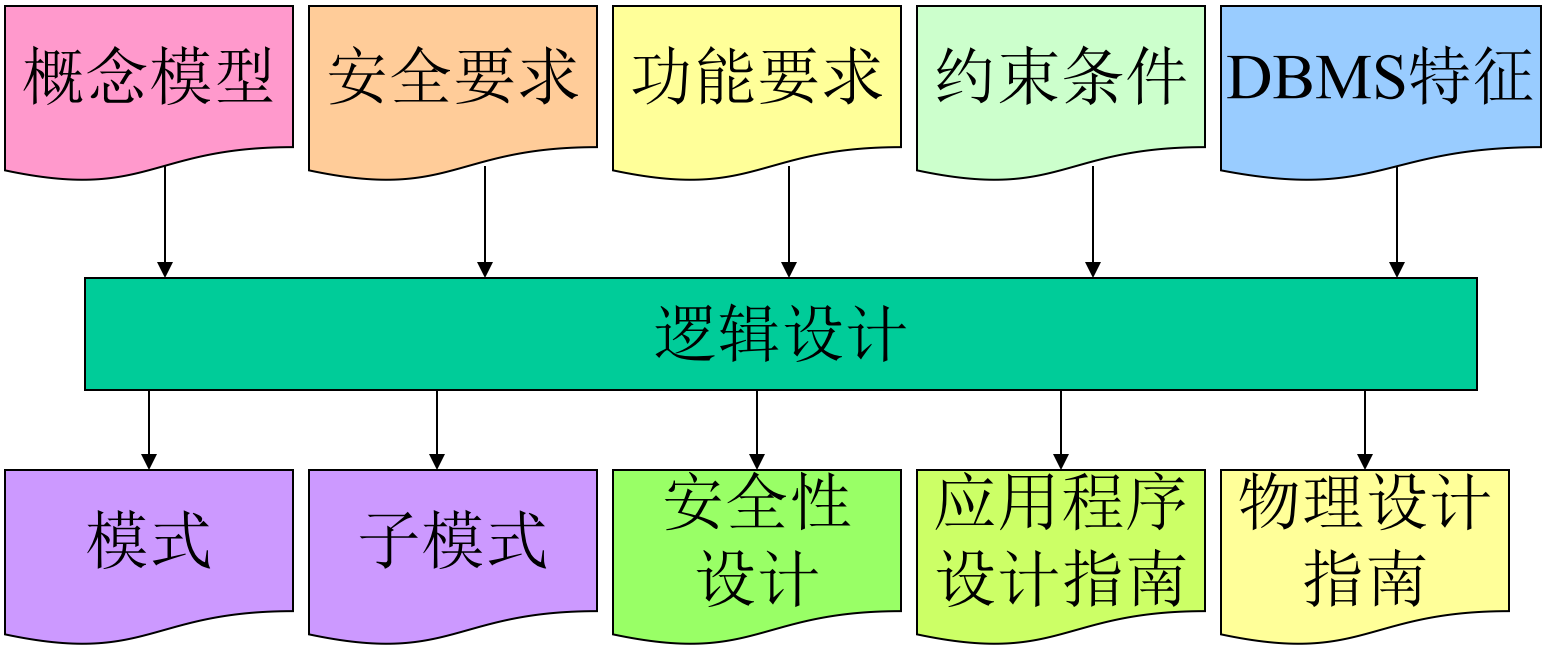
没有概念模型的评审过程，数据库设计人员和开发人员可能承担过多的责任。

5.1.4 逻辑设计

- 概念设计的结果是得到一个与DBMS无关的概念模式。而逻辑设计的目的是把概念模式设计阶段的全局ER模式转换成与选用的具体机器上的DBMS所支持的数据模型相符合的逻辑结构。
- 逻辑结构即数据库模式，包括数据库内模式、模式和外模式。逻辑设计主要设计模式和外模式，关系数据库可以说是数据库表和视图等。（例如Oracle的逻辑结构包括表空间及数据库对象）。

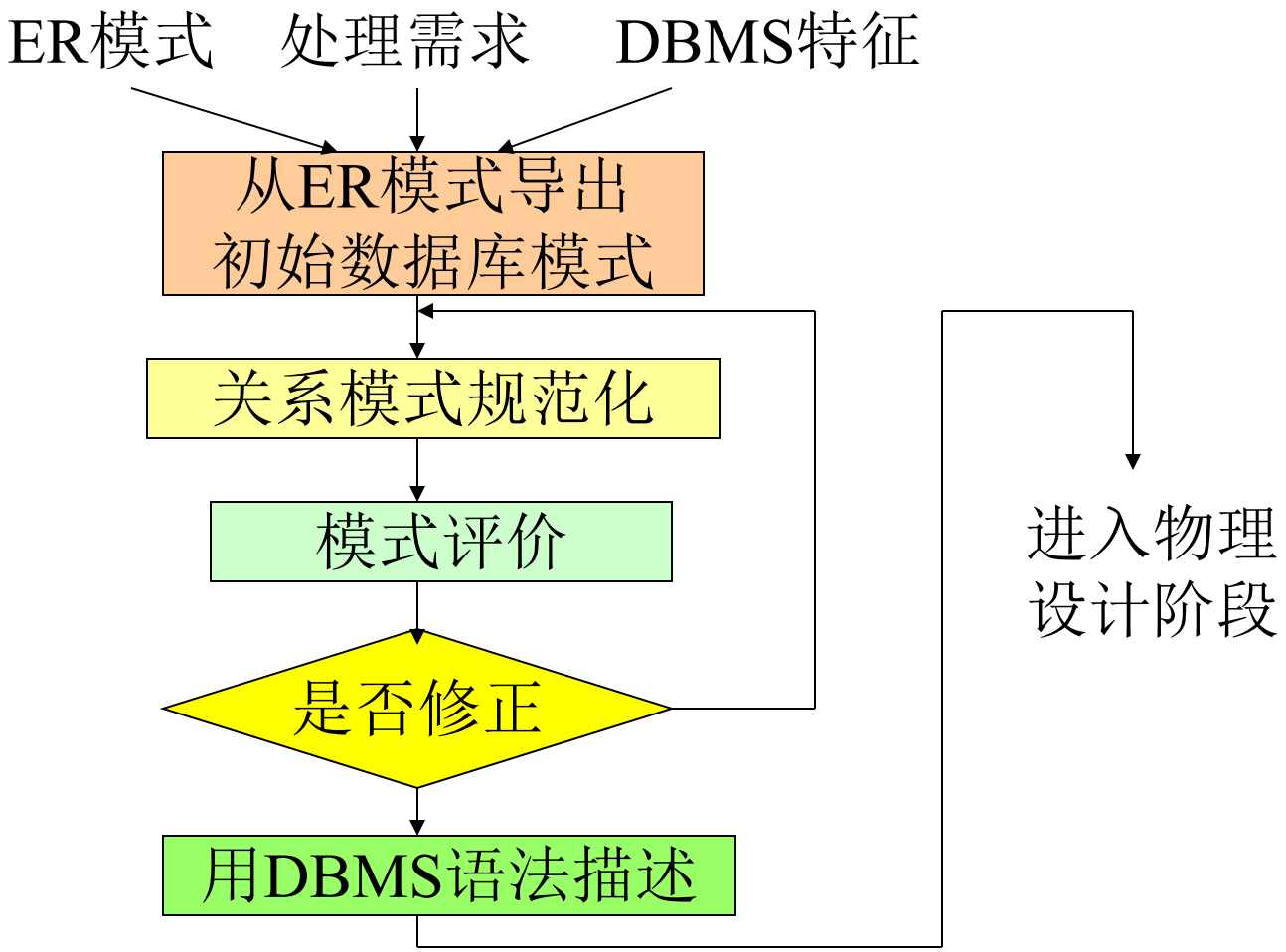
5.1.4 逻辑设计

- 逻辑设计可表示如下示意形式：



5.1.4 逻辑设计

- 逻辑设计的步骤：



5.1.4 逻辑设计

- ER模式向关系模式转换

一个实体型转换为一个关系模式，实体的属性即为关系模式的属性，实体的标识符即为关系模式的键。

联系类型的转换，根据不同情况做不同的处理：

1）若实体间的联系是一对一的，可以在两个实体类型转换成的两个关系模式中的任意一个关系模式的属性中加入另一个关系模式的键和联系类型的属性。也可能做合并关系模式处理。

例：学院与院长之间是一对一，在学院关系模式中加入院长关系模式的键。

5.1.4 逻辑设计

- ER模式向关系模式转换

联系类型的转换

2）若实体间的联系是**一对多**的，则在多端实体类型转换成的关系模式中加入一端实体类型转换成的关系模式的键和联系类型的属性。

例：学院与教师之间是一对多，在教师关系模式中加入学院关系模式的键。

5.1.4 逻辑设计

- ER模式向关系模式转换

联系类型的转换

3）若实体间的联系是**一对多**的，而且在多端实体类型为**弱实体**，转换成的关系模式中将一端实体类型(父实体)的键作为外键放入多端的弱实体(子实体)中。弱实体的主键由父实体的主键与弱实体本身的候选键组成。也可以为弱实体建立新的标识ID。

例：学生与社会关系之间是一对多，社会关系是弱实体，在社会关系关系模式中加入学生关系模式的键学号，由学号和称呼两属性组成社会关系关系模式的主键。

5.1.4 逻辑设计

- ER模式向关系模式转换

联系类型的转换

4）若实体间的联系是**多对多**的，则将联系类型也转换成关系模式，其属性为与该联系相连的各个实体的码以及联系的属性，而键是与该联系相连的各个实体的码的组合，或者是与该联系相连的各个实体的码和联系的附加属性的组合。

例：学生与课程之间是多对多，建立学生课程联系关系模式，学号和课程号组成学生课程联系关系模式的键。CAP数据库中Orders的码由顾客实体、代理商实体、产品实体的码和日期组成。

5.1.4 逻辑设计

- 规范化处理

规范化的处理的目的是减少乃至消除关系模式中存在的各种异常，改善完整性、一致性和存储效率。

对于有经验的数据库设计人员而言，在进行ER图的建立和关系模式转换时，已经考虑到规范化的要求，专门的规范化步骤往往指不能确定的实体类型、联系的关系转换。

一般的规范化过程是基于3NF或BCNF的标准进行的。在规范化模式合并和模式分解过程中，要特别注意保持依赖和无损连接要求（规范化内容参见5.3）。

5.1.4 逻辑设计

– 模式评价

模式评价的目的是检查已给出的数据库模式是否完全满足用户的功能要求，是否具有较高的效率。并确定需要加以修改的部分。

– 模式修正

根据模式评价结果对已生成的模式进行修改。

在逻辑设计阶段，还要设计出子模式，子模式体现各个用户对数据库的不同观点。

5.1.5 物理设计

- 数据库物理设计主要指数据库的存储记录格式、存储记录安排和存取方法，包括索引机制、空间大小、块的大小等，物理设计的目的是以合适的存储空间得到用户事务的快速响应（**时间和空间的效率**）。
  - 是在计算机的物理设备上确定应采取的数据存储结构和存取方法，以及如何分配存储空间等问题。
  - 物理设计与特定硬件系统、DBMS等相关，难以形成统一的设计方法。

5.1.5 物理设计

- 索引选择

  索引文件对存储记录进行内部链接，从逻辑上改变了记录的存储位置。

- 记录聚簇

  不改变逻辑数据库的模式结构，根据不同的使用要求，将属性记录或文件集中存放在不同的位置。

- 第四章Oracle中探讨了许多的数据存储问题，即物理设计问题。

5.1.5 物理设计

- 物理设计的过程：
  - 使用DBMS的可用功能完成基表的设计和完整性约束和业务规则。
  - DBMS为数据提供了许多可选择的文件组织方式，基于对事务的分析，选择合适的文件组织方式，何时选择索引。
  - 考虑放宽规范化约束，改善系统性能。怎样处理派生数据，历史数据。考虑重复列或连接表。
  - 设计安全措施来避免对数据未经授权的访问，如何实现每个用户的视图，以及表上需要的访问控制。
  - 不断通过监听操作系统来标识和解决由设计引起的性能问题，并实现新的或改变的需求。

5.1.6 数据库应用程序开发

- 数据库应用程序开发即应用程序的设计、编码、调试和试运行过程。其中应用程序设计是对用户界面和使用数据库的程序的设计。应用程序设计是数据库应用程序开发的重要环节。
- **功能分析的结果**是应用程序设计的重要依据。
- 根据逻辑设计和物理设计的结果，在计算机系统上建立起实际的数据库结构，装入部分实验数据，应用程序的开发建立在实验数据基础上，实验数据应尽量采用实际数据，并尽可能覆盖现实世界的各种情况。

5.1.6 数据库应用程序开发

- 应用程序开发的主要工作：
  - 应用程序设计：应用程序设计主要包括事务设计和用户界面设计。
    - 事务代表了现实世界的事件，事务设计包括事务使用什么数据，事务要做什么，事务的输出，事务的使用频度。事务有检索事务、更新事务、混合事务之分。
    - 用户界面设计要易于掌握、操作直观。界面设计可考虑：
      - 有意义的标题
      - 可视化空间布局与边界
      - 常用字段标签
      - 不可接受值的错误信息提示
      - 便利的游标活动
      - 一致性的术语和缩略语
      - 一致的颜色和字体
      - 一致的操作快捷方式（尽可能地非鼠标化操作）
      - 一致的鼠标操作方式

5.1.6 数据库应用程序开发

- 应用程序开发的主要工作：
  - 应用程序编写
  - 组织数据入库
  - 应用程序的调试与试运行

    应用程序开发完成后，要进行装入实际数据的试运行，试运行的过程是进一步检测系统功能的过程。对于重要系统的试运行过程中，原系统(可能是手工方式)的运行应当平行进行，以防止由于系统错误带来用户不可弥补的损失。
  - 整理文档

5.1.6 数据库应用程序开发

- 数据库系统正式运行，标志着数据库设计与应用开发工作的结束和维护阶段的开始，运行**维护**阶段的任务包括：
  - 维护数据库的安全性和完整性：检查系统安全性是否受到侵犯，及时调整授权和密码，实施系统转储与后备，发生故障及时恢复。
  - 监测并改善数据库性能：对数据库的存储空间状况及响应时间进行分析评价，结合用户反应确定改进措施。
  - 必要时对数据库进行重新组织和重新构造。
  - 根据用户要求对数据库现有功能进行扩充。
- 数据库运行和维护的工作就是**DBA的工作**。

作业：

1. 数据库系统设计人员应具备哪些知识?
2. 数据库系统设计的方法有哪些?
3. 需求分析的内容是什么?
4. 简述概念模型评审的必要性.
5. 结合第四章内容说明逻辑设计和物理设计的内容.

5.2 实体联系模型

- 实体联系模型(Entity Relationship Model)是P.P.Chen于1976年首先提出的，此后此模型不断扩展和完善，成为被广泛采用的概念模型设计方法。这个模型直接从现实世界中抽象出实体类型及实体间联系，然后用实体联系图(ER图)表示数据的抽象和数据的联系。设计ER图的方法称为ER方法。

  5.2.1 ER模型的概念

  5.2.2 ER图的绘制

  5.2.3 ER模型的转换

  5.2.4 数据库设计工具(CASE)

  5.2.5 ER模型实例分析

5.2.1 ER模型的概念

- **实体**(entity)就是具有公共性质的可区别的现实世界对象的集合。例如CAP数据库中的客户、代理商、产品都为实体，分别表示不同对象的集合。数学表述中通常用一个大写字母代表一个实体，一个实体E由一个现实世界对象的集合构成，使用小写字母加下标表示这些对象：
  $E=\{e_1, e_2, \ldots, e_n\}$。
- **属性**(attribute)是描述实体或者联系的性质的数据项。在实体的定义中说，属于一个实体的所有实体实例具有共同性质，这些性质就是属性。在一个实体中，能够唯一标识实体的实例的属性或属性集合称为实体标识符(主键)。属性域是属性的可能取值范围，也称为属性的值域。

5.2.1 ER模型的概念

- 属性的分类：

  基本属性和复合属性

  单值属性和多值属性

  导出属性和空值属性

  - 基本属性和复合属性

    基本属性是不可再分割的属性，复合属性是可再分解为其他属性的属性。例如性别、年龄为基本属性；地址属性为复合属性，因为地址可以分解为邮编、省(市)、县(区)、街道等子属性。

5.2.1 ER模型的概念

- 单值属性和多值属性

单值属性指的是同一实体的属性只能取一个值，多值属性指同一实体的某些属性可能取多个值。例如年龄属性只能取一个值，是单值属性；学位是多值属性，可以取学士、硕士、博士多个值，爱好也是多值属性。

- 导出属性和空值属性

通过具有相互依赖的属性推导而产生的属性称为导出属性，例如年龄可以由出生年份导出；当实体的实例在某个属性上没有值时应使用空值(Null),Null还可用于值未知，可以使用Null的属性称为空值属性。

5.2.1 ER模型的概念

- **联系**(relationship):给定m个实体的有序列表:$E_1, E_2, \ldots, E_m$(列表中同一个实体可以出现多次),一个联系R定义了这些实体实例之间的对应规则。联系表示一个或多个实体之间的关联关系,联系是实体之间的一种行为,一般用动词(英语用动名词)来命名联系。
  - 联系的元数

    一个联系涉及到的实体个数,称为该联系的元数或度数(degree)。同一个实体的实例之间的联系称为一元联系,也称递归联系;两个不同实体的实例之间的联系称为二元联系;三个不同实体实例之间联系称为三元联系;依此类推。

5.2.1 ER模型的概念

- 联系的属性

联系也可以有附加的属性。经常先不考虑ER图中联系的属性，集中精力考虑实体的联系。

- 联系中实体的基数

两个有联系R的实体E和F，E中每个实例可能与F中的实例联系，(联系实例数目大于0)，也可能没有与与F中的实例联系(联系实例数目等于0)，E中每个实例与F中有联系实例数目的最小值和最大值，称为E的基数。记作mincard(E,R)和maxcard(E,R)。同理有mincard(F,R)和maxcard(F,R)。

5.2.1 ER模型的概念

- 联系中实体的基数

例如学生实体E和课程实体F有选修联系R，每位学生至少选1门课，最多选10门课；每门课程最多有100人选，最少可以没人选。则有：

mincard(E,R)=1，maxcard(E,R)=10。

mincard(F,R)=0，maxcard(F,R)=100。

一个实体E参与联系R，并且mincard(E,R)=x，maxcard(E,R)=y，那么在ER图中，E和R之间的连接线可以用标记：card(E,R)=(x,y)表示实体的基数。

5.2.1 ER模型的概念

- 联系的方式

联系涉及到实体之间实例的对应方式，二元联系的联系方式有四种：1:1,1:N,M:N,M:1。由于M:1是1:N的反面，通常不单独提及。

如果实体E和F在联系R中有maxcard(E,R)=1，maxcard(F,R)=1，那么E和F联系是1:1的。

如果实体E和F在联系R中有maxcard(E,R)=N，maxcard(F,R)=1，那么E和F联系是1:N的。

如果实体E和F在联系R中有maxcard(E,R)=M，maxcard(F,R)=N，那么E和F联系是M:N的。

5.2.1 ER模型的概念

- 联系的方式

当一个联系R中的实体E具有mincard(E,R)=1时，E称为强制参与R(mandatory participation)，或简单称E在R中是强制的。一个实体F在R中不是强制的，则称为可选的(optional participation)。

类似地，可以给出一元联系、三元联系的一对一、一对多、多对多定义。特别注意多元联系的多对多联系方式。

5.2.1 ER模型的概念

- **属性的基数**：仿照联系中实体的基数概念，有实体中属性的基数，给定一个实体E和隶属于它的属性A，

  当mincard(A,E)=0时，表示属性A是可选的；

  当mincard(A,E)=1时，表示属性A是强制的；

  当maxcard(A,E)=1时，表示属性A是单值的；

  当maxcard(A,E)=N时，表示属性A是多值的。

  一个属性A参与实体E，并且mincard(A,E)=x，maxcard(A,E)=y，那么在ER图中，A和E之间的连接线可以用标记：card(A,E)=(x,y)表示属性的基数。

5.2.1 ER模型的概念

- **弱实体**：如果实体的所有实例都通过一个联系R依赖于另一个实体的实例而存在，而且该实体标识码的部分或全部从其依赖的实体(父实体)中获得，那末这个实体就称为弱实体，而另一个实体称为强实体。

  例如人事系统中，社会关系实体是以职工实体存在为前提，社会关系实体是弱实体。

- **泛化层次**：泛化层次(generalization hierarchy)也称泛化联系(generalization relationship)，是对应于对象关系模型继承特性的一个概念。其思想是多个有公共属性的实体可以泛化为一个更高层次的超类型实体，相反一个一般化实体可以分解成低层次的子类型实体。

5.2.1 ER模型的概念

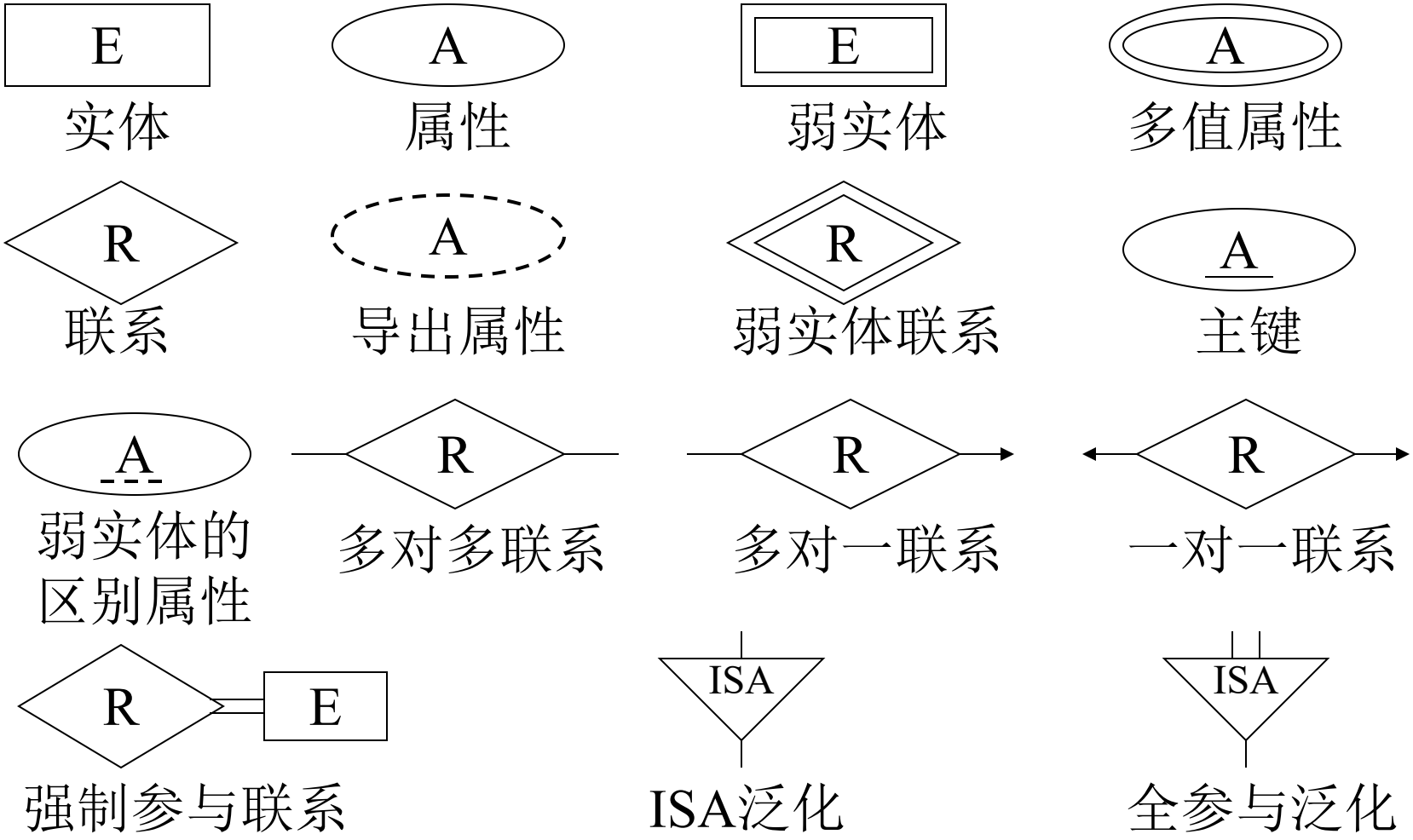
- **泛化层次**：例：

学生和教师都是人，学生实体和教师实体泛化出超类实体人。子类一个重要性质是继承性：子类继承其超类上定义的所有属性，其本身还可以包含其他另外的属性。子类型实体和超类型实体之间的联系经常称为ISA联系。

关系模型没有为泛化层次概念提供支持，关系模型中有**两种方法支持泛化**：可以保留超类型实体和所有子类型实体并创建显式联系来表示这个ISA联系；也可以把子类型实体的属性加入到超类型实体中并加入一个附加属性来区别这些不同的子类型。

5.2.2 ER图的绘制

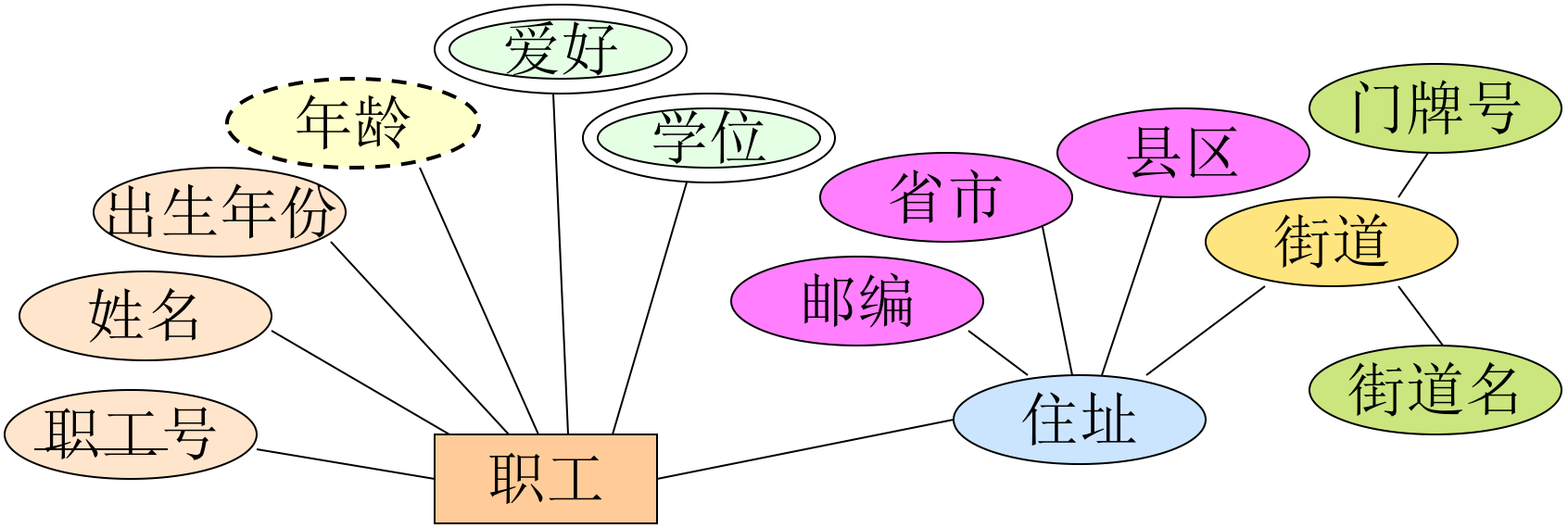
- 前面介绍的ER模型中的概念可以采用以下图例表示：



5.2.2 ER图的绘制

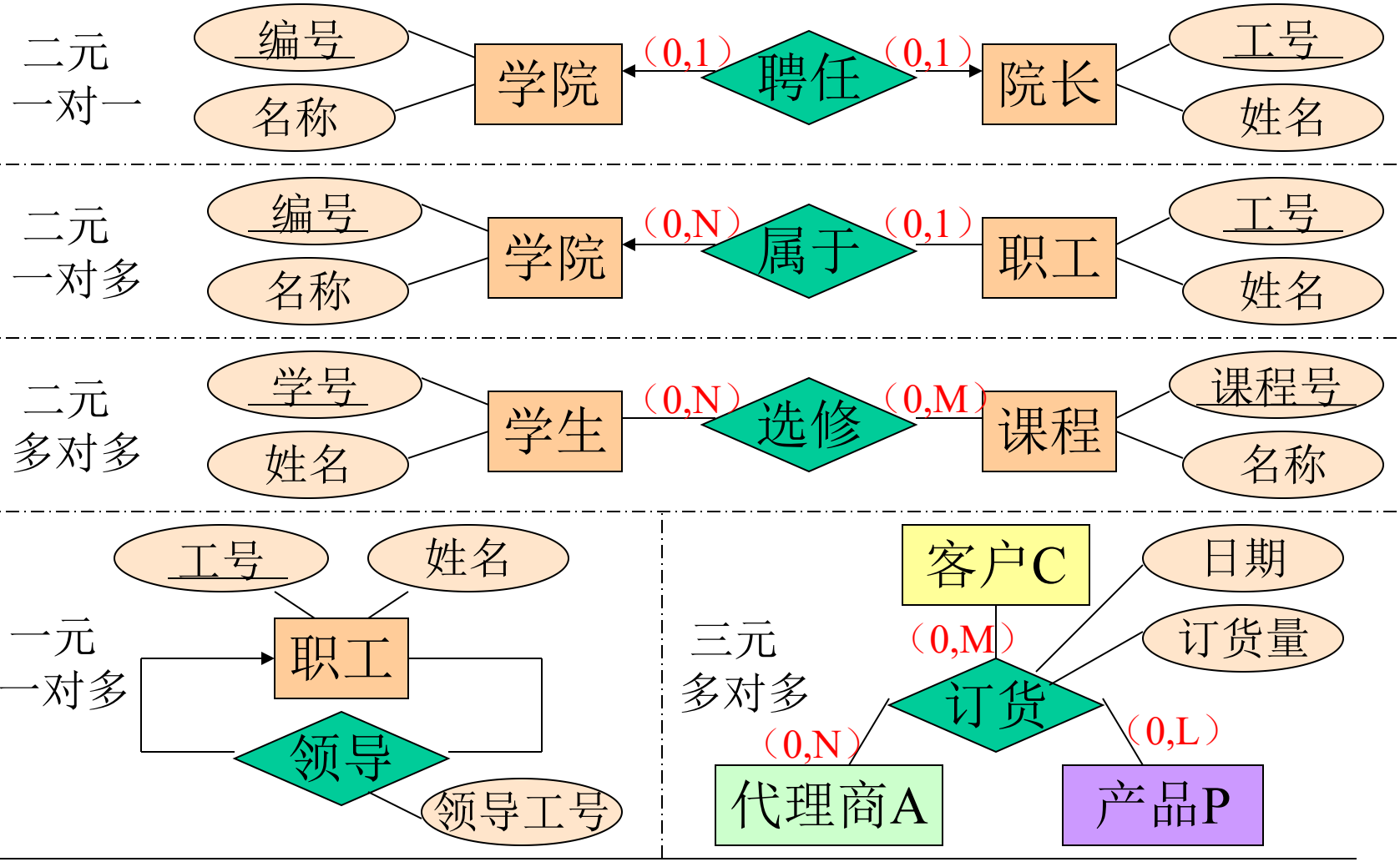
- 举例：

下图给出了包含主属性、基本属性、导出属性(年龄)、多值属性(学位、爱好)、复合属性(住址)的职工实体。说明：年龄可以由出生年份导出；学位(爱好)可能是一种学位(爱好)，也可能是多种学位(爱好)；住址是一个模糊的概念，它又可以分成省市、县区、街道以及楼层单元等子属性。



5.2.2 ER图的绘制

- 举例：下图给出了联系的类型：



5.2.2 ER图的绘制

- 前面举例中特别**注意**：多对一联系中的‘多’方是maxcard值取1的一方；‘一’方是实体实例能够参与多个联系实例的一方，这些实体实例‘射出多条连接线’连接到‘多’方的多个实体实例。

  比如学院实体和职工实体联系中，尽管maxcard(职工,属于)=1，但职工实体方是多方，学院实体的实例‘射出多条连接线’连接到职工实体的多个实例。

- maxcard(职工,属于)=1表示一个职工只能属于一个学院，mincard(职工,属于)=0表示一个职工可能不属于任何学院，同理maxcard(学院,属于)=N表示一个学院可能有多名职工，mincard(学院,属于)=0表示一个学院可能没有任何一个职工。

5.2.2 ER图的绘制

- 一般情况下关于联系中实体的基数不讨论0、1和N以外的其他数值，但如果能给出具体的数值，则可能产生更进一步的语义信息(参见下面的例子)。
- 进一步举例：
  - 人事系统中完成职工信息、部门信息、工资信息、培训信息、奖励信息、考勤信息的管理。分析如下：

    1）一个职工可能不属于任意一个部门，也可能属于一个部门，如果包含历史信息，则一个职工可能属于多个部门；一个部门包含多名职工。为了简化确定部门和职工是一对多联系，联系属性有调入日期。

5.2.2 ER图的绘制

– 人事系统分析

2）职工工资由其所聘任的岗位决定，类似于职工和部门关系，岗位和职工也是一对多联系，联系属性包括聘任日期。

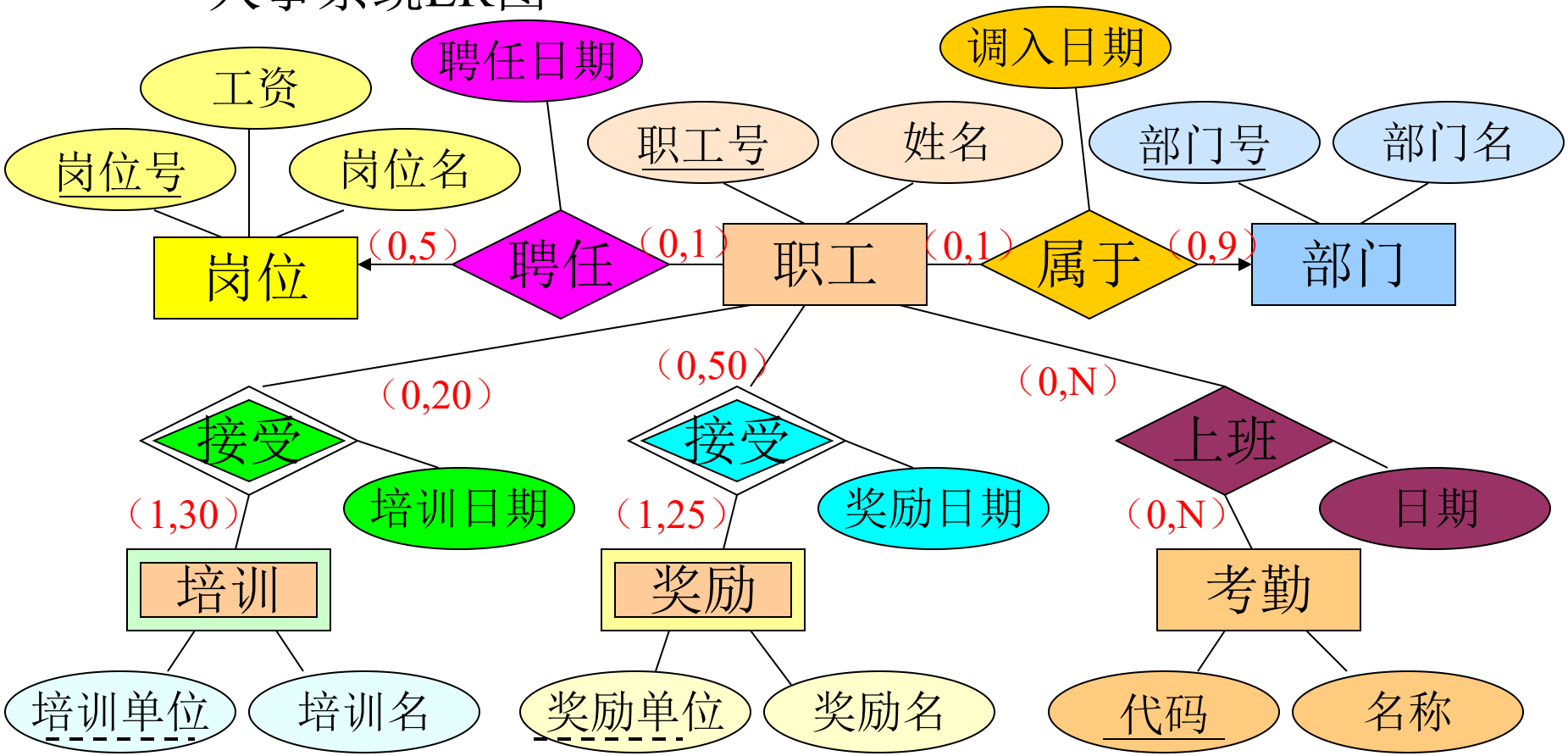
3）职工接受培训可能是本单位委派，也可能是自己决定，所以培训内容是随职工接受过培训才存在，培训内容是弱实体，职工与培训联系是多对多。联系属性有培训日期。

4）奖励信息管理如同培训情形。

5）考勤是建立在每天(当然是工作日)考勤状况基础上。考勤内容可以有迟到、早退、旷工、请假等，职工与考勤的联系是多对多，联系属性有日期。

5.2.2 ER图的绘制

– 人事系统ER图



说明：通过前面分析人事系统中包含职工、部门、岗位、培训、奖励、考勤7个实体，其中培训、奖励是弱实体。

5.2.2 ER图的绘制

– 人事系统ER图解释：

前面的ER图中，联系两边的连线上有数据对，它们的含义是(根据实际情况决定，在此仅是假设)：

一个岗位最少为无职工被聘任，最大只能有5个职工被聘任，一个职工最少是无岗位，最大只能被聘任一个岗位(不能兼任岗位)；

一个部门最少为无所属职工，最大只能有9个职工(部门不能太大)，一个职工最少是无所属部门，最大只能属于一个部门(不能兼任部门)；

一个培训最少要有职工被培训过，最大只能有30个职工被培训，一个职工最少是从未被培训，最大只能被培训20次；

5.2.2 ER图的绘制

– 人事系统ER图解释：

一个奖励最少要有职工被奖励过，最大只能有25个职工被奖励，一个职工最少是从未被奖励，最大只能被奖励50次；

一项考勤最少为无职工被使用，最大无限制，一个职工最少是从未使用考勤内容，最大无限制。(考勤内容中应无‘正常’项，否则不然)。

- 通过前面举例，可以清楚地看到，ER图通过实体、属性、联系三个方面的认知，基本包含了现实世界信息之间的关系描述(实体可以是物理的，也可以是逻辑的)。

5.2.2 ER图的绘制

- ER图是建立在对实际项目清晰的分析基础之上的，实体的确定以及联系的类型是根据实际语义确定的。它由数据库设计人员绘制，但对DBA或开发人员是透明的，对用户也是透明的。
- 我们可以通过对ER图的解释(如前面说明)，不断与用户进行沟通，逐步修改完善ER图，直至与用户在信息组织方面达成完全的一致。

5.2.2 ER图的绘制

- ER图是概念设计的一个很好的工具。需要说明的是，对于前面的举例，只是人事管理系统的一个侧面，具体用户所需的人事管理系统可以在此基础上修改和扩展。同样是人事管理系统，针对不同的用户需求，完全可能给出不大相同的ER图。
- 本单元给出的ER图图例并不是标准。关于ER图绘制的图例标准，目前暂无定论，但不同文献的介绍基本大同小异。

5.2.3 ER图的转换

- 转换规则1：ER图中的每一个实体映射到关系数据库中的一个表，并用实体名来命名这个表。表的列代表了连接到实体的所有简单单值属性(可能是通过复合属性连接到实体的，但复合属性本身并不变成表的列)。实体的标识符映射为该表的候选键，实体的主标识符映射为主键。实体实例映射为该表的行。
- 转换规则2：给定一个实体E，主标识是p，a是E的一个多值属性，那么a映射成自身的一个表，该表的列包含p和a，这个表的主键是p和a的组合。E当然地映射成一个表。

5.2.3 ER图的转换

- 转换规则3：当两个实体E和F参与一个二元的多对多的联系R时，联系R映射成一个表T。这个表包括从实体E和F转化而来的两个表的主键，这些列构成了表T的主键，也可能还得再加入联系的属性才能构成表T的主键。
- 转换规则4：当两个实体E和F参与一个多对一的二元联系R时，这个联系在关系数据库设计中不能被映射自身的一个表。假设F表示多方，具有maxcard(F,R)=1，那么从F转化成的表T中应当包括从E转化的表的主键，这被称为T的外键。因为maxcard(F,R)=1，T的每一行都通过一个外键值联系到实体E的一个实例。

5.2.3 ER图的转换

- 转换规则4(续)：如果F在R中是强制参与的，那么意味着T的上述外键不能取空值。如果F在R中是选择参与的，那么T中不与E的实例相联系的行的外键可以取空值。
- 转换规则5：如果给定实体E和F，它们的联系是一对一，二者的参与都是可选的，那么E和F分别转换为表S和T，并且在表S中加入T表的主键作为S的外键，在表T中加入S表的主键作为T的外键，有此表示E和F的可选参与一对一联系。
- 转换规则6：如果给定实体E和F，它们的联系是一对一，二者的参与都是强制的，那么最好将E和F对应的两个表合并成一个表，由此而避免使用外键。

5.2.3 ER图的转换

- 转换规则7：如果参与联系的实体数目多于两个，一般称为多元联系。对于多元联系，通常用多个不同的二元联系替代，再针对二元联系进行转换(Oracle的数据库设计工具Designer只处理二元联系的转换)。

  对于不能用多个不同的二元联系替代的情形，一般采用如下规则进行转换：除参与联系的实体转换为相应的表之外，联系也转换为独立的表，表中包含了参与联系的实体的主键。注：转换的表能够真实表达语义是最终目的。

5.2.3 ER图的转换

- 转换规则8：对于一元联系，一般不特别将联系转换成独立的表，而在参与联系的这个实体相应的表中加入联系的附加属性。例如职工实体的领导联系是一个一元联系，职工实体转换成相应的表，在此表中加入领导工号属性。联系转换为表中的一列。

5.2.4 数据库设计工具(CASE)

- 很多商业产品提供数据库设计工具，支持数据库设计人员进行数据库设计，例如Oracle提供Designer/2000和RationalRose都是数据库设计工具。通常这种工具有很多组件，它们由下面各种类型组件中的部分组成。
  - ER设计编辑器：设计者可以在其中构造ER图，一般采用图形化界面，通过拖放方式来编辑和修改ER图。
  - ER图到关系设计转换器：可以自动将ER图转换到关系数据库中的表。使用这种工具，设计数据库从ER图的设计开始，然后将ER图自动转换成关系表。注意，不同的CASE产品对ER图的图例的表示不一定相同，不同的CASE产品对ER图的理解也不一定相同，往往都作了一定的简化或扩展，一般不能处理前面所介绍全部内容。

5.2.4 数据库设计工具(CASE)

- 函数依赖到ER设计转换器：有时候CASE中提供另一种类型的组件，它从一个数据库的函数依赖集合生成有效的ER图来反应数据的规则。
- 设计分析器：分析所作的设计工作并给出报告，它可以帮助数据库管理员改正各种各样的错误。

  除上面的这些组件之外，还有些CASE提供数据库系统设计的全过程支持，从业务流程图的绘制、ER图的绘制开始，自动生成系统的功能菜单，数据字典、数据库关系表，并能够生成数据库系统设计过程中的全部文档资料。

- Oracle的Designer/2000和Developer/2000结合使用可以一条龙完成数据库系统的开发工作。

5.2.5 ER模型实例分析

- 对5.2.2中人事系统ER图，转换为如下几个数据库关系表(数据类型和长度略)：
  - 职工表：职工号，姓名，部门号，调入日期，岗位号，聘任任期。
  - 部门表：部门号，名称
  - 岗位表：岗位号，岗位名称，工资
  - 职工培训表：职工号，日期，培训单位
  - 培训表：培训单位，培训名称
  - 职工奖励表：职工号，日期，奖励单位
  - 奖励表：奖励单位，奖励名称
  - 职工考勤表：职工号，日期，考勤代码
  - 考勤表：代码，名称

5.3 关系规范化

- 规范化是关系数据库逻辑设计的另一种方法，它和ER模型的出发点不一样，但是一个基于规范化的关系设计和一个由ER模型转换成的关系设计几乎得到相同的结果。两种方法具有互补性。规范化方法中，从一个将被建模的现实世界的情形出发，分析数据项之间的相互影响，通过无损分解使数据库的逻辑设计趋于合理，把低一级的关系模式分解为若干个高一级的关系模式。

  5.3.1 关系模式的设计问题

  5.3.2 函数依赖

  5.3.3 关系模式的分解特性

  5.3.4 范式和关系模式规范化

5.3.1 关系模式的设计问题

- 关系模型要求关系必须是规范化的，即要求数据库表中不允许含有多值属性和含有内部结构，关系的每一个分量必须是一个不可分的的数据项，不允许表中还有表。遵守这样规则的表称为第一范式。这是关系数据库设计过程中的一个最基本的约束(见第一章)。
- 满足第一泛式不能保证数据库模式是良好的数据库模式。所谓良好的数据库模式的标准是什么？如何实现？这就是关系模式的设计问题。

5.3.1 关系模式的设计问题

- 存在大量数据冗余的数据库模式不是良好的数据库模式。因为大量的数据冗余不仅造成存储空间的浪费，存取效率的低下，而且使得数据信息的更新变得繁琐复杂，另外还隐藏着破坏数据一致性完整性的危险。
- 良好的数据库模式不存在数据冗余、插入异常、删除异常和更新异常等问题。重新分配数据项到不同的表中，能够消除这些问题，这恰是规范化过程将实现的任务。

5.3.1 关系模式的设计问题

- 更新异常(Update Anomaly)：如果更改表所对应的某个实体实例或者关系实例的单个属性时，需要将多行更新，那么就说这个表存在更新异常。数据冗余引起更新异常。
- 删除异常(Delete Anomaly)：如果删除表的某一行来反应某个实体实例或者关系实例消失时，会导致丢失另一个不同实体实例或者关系实例的信息，而这是我们不希望丢失的，那么就说这个表存在删除异常。
- 插入异常(Insert Anomaly)：如果某个实体或者实例信息随着另一个实体或实例信息的存在而存在，在缺少另一个实体或实例信息时，无法表示这个实体或者实例信息，而这是我们不希望看到的，那么就说这个表存在插入异常。

5.3.1 关系模式的设计问题

- 异常情况举例：
  - 下图模式1是管理职工工资的一个常见的简化的关系模式。

工资管理模式1

工号	姓名	级别	基本	津贴	考勤扣款	违纪扣款	实发额
001	张三	一级	800	2000	120	10	2670
002	李四	三级	1200	4000	20		5180

说明：应发工资额为基本工资和津贴之和，应发工资减去扣款项目为实发工资额，基本工资和津贴的额度由级别决定，扣款额根据实际情况决定。

5.3.1 关系模式的设计问题

工资管理模式1存在以下问题：

1）扣款项目不是每个职工都发生，数据冗余，浪费空间。

2）当公司出现其他扣款项目时，必须修改模式结构。

3）应发工资额由级别决定，当没有职工级别是二级时，二级的基本工资和津贴无法添加，插入异常。删除张三信息时，把一级与应发工资额的对应规则也删除了，删除异常。修改级别与应发工资额的对应规则时，必修修改多行，更新异常。

5.3.1 关系模式的设计问题

– 下图模式2是改进后的职工工资管理关系模式。

工资管理模式2

工号	姓名	级别	基本	津贴	扣款项目	扣款金额
001	张三	一级	800	2000	考勤	120
001	张三	一级	800	2000	违纪	10
002	李四	三级	1200	4000	考勤	20

说明：该模式解决了问题2，当公司出现其他扣款项目时，无须修改模式结构，比模式1合理。

5.3.1 关系模式的设计问题

工资管理模式2仍然存在问题：

1）职工信息和应发项目随扣款项目重复，数据冗余。

2）应发工资额由级别决定，当没有职工级别是二级时，二级的基本工资和津贴无法添加，插入异常。删除张三信息时，把一级与应发工资额的对应规则也删除了，删除异常。修改级别与应发工资额的对应规则时，必修修改多行，更新异常。

- 为了给模式2进一步优化，需对模式2做进一步分析。

5.3.2 函数依赖

- 函数依赖(FD)定义了数据库系统中数据项之间相关性性质最常见的类型。建立在函数依赖的基础之上，作模式的进一步规范化。
- 定义：设有关系模式R(U)，X和Y是属性集U的子集，函数依赖(Functional Dependency, FD)是形为X→Y的一个命题，对任意R中两个元组t和s，都有t[X]=s[X]蕴涵t[Y]=s[Y]，那么FD X→Y在关系模式R(U)中成立。X→Y读作‘X函数决定Y’，或‘Y函数依赖于X’。

5.3.2 函数依赖

- 函数依赖举例：

  工资管理模式2中的函数依赖：

  工号→姓名；工号→级别；级别→基本；

  级别→津贴；工号,扣款项目→扣款金额。

  以上函数依赖是根据实际语义，分析数据项之间相关性性得出的。

- 通常并不根据表的内容得出函数依赖，而是通过理解数据项和企业的规则来决定函数依赖。

5.3.2 函数依赖

- 定义：设F是在关系模式R(U)上成立的函数依赖集，X和Y是属性集U的子集。如果从F推导出X→Y也在R(U)上成立，那么称F逻辑蕴涵X→Y。
- 定义：设F是在关系模式R(U)上成立的函数依赖集，被F逻辑蕴涵的函数依赖集合，称为F的闭包，记为F^+ 。
- Armstrong公理：
  - 包含规则(Inclusion rule)：如果Y⊂X, 那么X→Y。
  - 传递规则(Transitivity rule)：如果X→Y且Y→Z, 那么X→Z。
  - 增广规则(Augmentation rule)：如果X→Y, 那么XZ→YZ。

5.3.2 函数依赖

- Armstrong公理的一些逻辑蕴涵：
  - 合并规则(Union rule)：如果X→Y且X→Z，那么X→YZ。
  - 分解规则(Decomposition rule)：如果X→YZ，那么X→Y且X→Z。
  - 伪传递规则(Pseudoransitivity rule)：如果X→Y且WY→Z，那么XW→Z。
  - 集合累积规则(Set accumulation rule)：如果X→YZ且Z→W，那么X→YZW。
- 在实际使用中，经常要判断能否从已知的FD集F推导出X→Y(比如在决定候选键的过程中)，那么可以先求出F的闭包F^+，然后再看X→Y是否在F^+中。

5.3.2 函数依赖

- 求解F的逻辑蕴涵问题是一个算法问题。
  - Armstrong公理是函数依赖的一个正确的和完备的推理规则集，正确性是指‘从FD集F使用推理规则集推出的FD必定在F^+中’，完备性是指‘F^+中的FD都能从F集使用推理规则集导出’。
  - 函数依赖集F中的FD很多，我们应该从F中去掉平凡的FD、无关的FD、FD中无关的属性，以求得F的最小依赖集F_{min}。
  - 从F求出F^+是一个复杂的困难的算法问题(见参考资料1)。引入属性集的闭包概念将使该问题简化。
  - 建立在属性集的闭包的基础上，使得F的最小依赖集F_{min}的求解算法变得简化可行(见参考资料1)。

注：得到函数依赖的逻辑蕴涵的目的是建立在函数依赖基础上进行关系的规范化。

5.3.2 函数依赖

- 定义：给定表T的函数依赖集F和属性集X，X函数决定的最大属性集合Y称为属性X的闭包X^+，X^+={属性Y | X→Y在F^+中}。从X求出X^+并不难。

例：工资管理模式2中的属性闭包（推导过程见后页）：

工号$^+$={工号, 姓名, 级别, 基本, 津贴}；

级别$^+$={级别, 基本, 津贴}；

{工号, 扣款项目}$^+$={工号, 姓名, 级别, 基本, 津贴, 扣款项目, 扣款金额}。

5.3.2 函数依赖

- 定义：F是属性集U上的FD集。如果F_{min}是F的一个最小依赖集，那么F_{min}应满足下列四个条件：
  1）$F_{min}^{+} = F^{+}$；
  2）每个FD的右边都是单属性；
  3）F_{min}中没有冗余的FD；
  4）每个FD的左边没有冗余的属性。
- 定义：设关系模式R的属性集是U，X是U的一个子集。如果X→U在R上成立，那么称X是R的一个超键。如果X→U在R上成立，但对于X的任一个真子集X_1都有X_1→U不成立，那么称X是R的一个候选键。

5.3.2 函数依赖

- 例：工资管理模式2中，因为{工号,扣款项目}$^{+}$={工号,姓名,级别,基本,津贴,扣款项目,扣款金额}，所以它的候选键是：{工号,扣款项目}。

  证明：按照Armstrong公理，

  工号⊂{工号,扣款项目}⇒{工号,扣款项目}→工号，

  又:工号→姓名⇒{工号,扣款项目}→姓名 ①

  同理有： {工号,扣款项目}→级别 ②

  级别→基本⇒ {工号,扣款项目}→基本 ③

  级别→津贴⇒ {工号,扣款项目}→津贴 ④

  {工号,扣款项目}→扣款项目 ⑤

  {工号,扣款项目}→扣款金额 ⑥

  综合① ② ③ ④ ⑤ ⑥有以上结论。证毕

5.3.3 关系模式的分解特性

- 规范化过程是建立在模式分解基础上的，将表分解成为两个或更多较小的表，可以通过把分解出的表连接起来重新获得原始表的详细信息。
- 定义：对于任何表R(U)，R的一个分解是一个表的集合$\{R_1(U_1), R_2(U_2), \ldots, R_k(U_k)\}$，该集合具有性质：$U_i$是U的一个子集且$U=U_1 \cup U_2 \cup \ldots \cup U_k$。
- R分解成$R_1, R_2, \ldots, R_k$的目的是为了消除数据冗余和操作异常现象。如果分解后的表表示了不同的内容，分解就没有什么意义。

5.3.3 关系模式的分解特性

- 可以从两个角度来考虑模式的分解：

  1）无损分解：分解后的表能够通过连接运算恢复分解前表的内容，保证分解前表的内容没有损失。

  2）保持依赖：在模式R上有一个FD集F，在R_i上有一个FD集F_i，那么$\{F_1, F_2, \ldots, F_k\}$与F应该等价，即$\{F_1, F_2, \ldots, F_k\}^+ = F^+$。

- 不是无损分解或没有保持依赖的分解很难说是一个好的模式设计。

5.3.4 范式和关系模式规范化

- 范式(Normal Forms,NF)是与函数依赖有直接联系的，评价模式好坏的衡量标准。建立在函数依赖基础上的范式有1NF,2NF,3NF,BCNF。经常称某一关系模式为第几范式，表明该关系模式是符合某一种范式级别的关系模式。
- 一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫关系模式规范化。
- 第一范式(1NF)：如果关系模式R的每个关系r的属性都是不可分的数据项，那么就称R是第一范式的模式。1NF是关系模式应具备的最起码的条件。关系数据库设计研究的关系规范化是在1NF之上进行的。

5.3.4 范式和关系模式规范化

- 只符合1NF标准的关系模式不是好的数据库模式。前面介绍的工资管理模式2即为1NF。
- 第二范式(2NF)：
  - 定义：对于FD W→A，如果存在X⊂W有X→A成立，那么称W→A是局部依赖(A局部依赖于W)；否则称W→A是完全依赖。
  - 定义：如果关系模式R是1NF，且每个非主属性完全函数依赖于候选键，那么就称R是第二范式。
  - 工资管理模式2属于1NF，但不属于2NF，因为非主属性‘姓名’、‘级别’、‘基本’、‘津贴’是部分函数依赖于候选键{工号,扣款项目}的，1NF存在数据冗余、操作异常。

5.3.4 范式和关系模式规范化

- 1NF模式通过分解可以达到2NF，方法是消除非主属性对候选键的部分依赖，把部分依赖的属性单独组建关系。
- 工资管理模式2分解成模式3(如下表)，模式3属于2NF。

工资管理模式3

表1

工号	姓名	级别	基本	津贴
001	张三	一级	800	2000
002	李四	三级	1200	4000

表2

工号	扣款项目	扣款金额
001	考勤	120
001	违纪	10
002	考勤	20

5.3.4 范式和关系模式规范化

- 第三范式(3NF)：
  - 定义：如果X→Y，Y→A且Y!→X和A!⊂Y，那么称X→A是传递依赖(A传递依赖于X)。
  - 定义：关系模式R中不存在这样的候选键X，属性组Y和非主属性Z(Z!⊂Y)使得X→Y,(Y!→X)Y→Z成立,即非主属性既不部分依赖也不传递依赖于R的候选键，则称R是第三范式。
  - 工资管理模式3属于2NF，但不属于3NF，因为表1中非主属性‘基本’、‘津贴’是传递依赖于候选键{工号}的，2NF存在数据冗余、操作异常。
  - 2NF模式通过分解可以达到3NF，方法是消除非主属性对候选键的传递依赖，把传递依赖的属性单独组建关系。

5.3.4 范式和关系模式规范化

- 工资管理模式3分解成模式4(如下表)，模式4属于3NF。

工资管理模式4

表1

工号	姓名	级别
001	张三	一级
002	李四	三级

表3

级别	基本	津贴
一级	800	2000
三级	1200	4000

表2

工号	扣款项目	扣款金额
001	考勤	120
001	违纪	10
002	考勤	20

5.3.4 范式和关系模式规范化

- BC范式(BCNF)：
  - 3NF模式中讨论非主属性，并未排除主属性对候选键的传递依赖，任有可能存在冗余和异常。
  - 定义：如果关系模式R是1NF，且每个属性都不部分依赖于候选键也不传递依赖于候选键，那么称R是BC范式。
  - 属于BCNF的模式一定属于3NF，但属于3NF的模式不一定是BCNF。
  - 例：R(Bno, Bname, Author)的属性表示书号、书名和作者名。调查实际情况后有如下规则：每个书号只有一个书名，不同书号可以有相同书名；每本书可以有多个作者合作，但每个作者参与编著的书名应该互不相同。这样得到两个FD：

    Bno→Bname和{Author, Bname}→Bno

5.3.4 范式和关系模式规范化

- 例(续)：

根据逻辑蕴涵可以推出R的候选键为{Bname, Author}或{Bno, Author}，R的属性都是主属性，R显然是3NF。但Bname传递依赖于候选键{Author, Bname}，所以R不是BCNF。

R存在冗余和操作异常：一本书由多个作者编写时，其书名与书号间的联系在关系中将多次出现，数据冗余；修改由多个作者编写的书的书名时必须修改多条记录，更新异常。

把R分解成R_1{Bno, Bname}和R_2{Bno, Author}，能解决上述问题，且都是BCNF。但这个分解把{Author, Bname}→Bno的函数依赖丢失了。

- 对于排除主属性对候选键的传递依赖或部分依赖的问题，模式分解不能保证保持函数依赖。

5.3.4 范式和关系模式规范化

- 关系规范化盘点：
  - 不好的数据库设计存在数据冗余，操作异常。函数依赖的分析提供了很好的数据库设计思路。
  - 对数据库设计过程中初始函数依赖关系的采集是通过理解数据项和企业的规则来决定函数依赖的。
  - 函数依赖存在大量的蕴涵逻辑，通过Armstrong公理可以得出许多蕴涵的函数依赖，根据所有这些函数依赖找出关系的候选码、主属性、非主属性。
  - 不好的数据库模式的原因是表达函数依赖的方式不妥，通过模式的分解，在保证无损连接的同时更好地表达函数依赖。研究并消除属性之间的部分依赖、传递依赖即可得到高级别范式的模式。

5.3.4 范式和关系模式规范化

- 关系规范化盘点：
  - 模式好坏的评价标准是范式，范式的高低不是规范化的步骤。从不同角度的分解都可以得到高级别的范式，一般不是从1NF到2NF，2NF到3NF的。
  - 实际使用时，一般是基于3NF进行，但根据实际情况，不达3NF的情况也很常见。
  - 有的数据库设计工具(CASE)支持基于函数依赖的规范化方法的算法，从函数依赖集合生成有效的ER图来反映数据的规则。
  - ER方法和基于函数依赖的规范化方法都有各自缺点，ER方法过分依赖于设计者的直觉，规范化方法又更多地基于数学方法，在应用时显得过分机械。结合两种方法，ER图从实体联系角度，规范化从函数依赖角度，一般能得到较好的数据库模式。

5.3.4 范式和关系模式规范化

- 以上规范化过程是建立在函数依赖基础上进行的。除函数依赖之外，还有其它的数据依赖(见参考资料5)。数据库设计的规范化问题研究是数据库领域的一个重要研究方向。

作业：

1. 数据库设计中如何处理单值属性、多值属性、导出属性？
2. 举例说明联系中实体基数的意义。
3. 某商业集团数据库中有三个实体。商店的属性有编号、商店名、地址等；商品属性有编号、商品名、规格、单价等；职工属性有编号、姓名、性别、业绩等。每个商店可销售多种商品，每种商品也可放在多个商店销售，有月销售量；每个商店有多名职工，每个职工只能在一个商店工作，商店聘用职工有聘期和月薪，试画ER图。
4. 设关系模式R(ABCD)，B值与D值之间是一对多，A值与C值是一对一。写出R的函数依赖集F，并求F^+。
5. 设关系模式R(ABCD)，F={A→B,C→B}，写出R的候选键。
6. 设有R(队员编号，比赛场次，进球数，球队名，队长名)记录球队队员每场比赛进球数，规定每个队员只能属于一个球队，每个球队只用一个队长。
   1）写出R的FD和关键码。
   2）说明R不是2NF的理由，并把R分解成2NF，再分解成3NF。