

PART I 基础篇

CHAPTER 1 数据库系统引论

What

1. 数据密集型应用 (data intensive application)

一类重要的计算机应用——数据密集型应用 (data-intensive applications), 其特点:

- 数据量大 (e.g., exceeding MB level)
- 持久数据 (persistent data)
- 共享数据 (shared data)

2. 数据管理 (data management)

数据密集型应用中的核心技术是数据管理 (data management), 其主要任务:

- 数据组织与编码 (data organization and coding)
- 数据存储、索引 (data storage and indexing)
- 数据访问/检索/查询 (data access/retrieval/query)
- 数据更新与维护 (data updating and maintenance)
- 数据安全 (data security) ...

数据管理是数据处理的基础

3. 数据库、DBMS&数据库系统 (database, DBMS & database system)

数据库: 一个数据库是一个逻辑上相关的可共享数据集。数据库方法借助特殊的软件系统——数据库管理系统 (DBMS) 来实现数据管理。

DBMS: 一种特殊的软件系统, 为数据库提供了数据抽象 (data abstraction)、程序-数据独立性 (independence) 以及一系列数据管理辅助功能 (见后文), 形成了有效的数据管理方法。

数据库系统 (database system): 由数据库、数据库管理系统 (DBMS)、数据库应用程序和创建、维护与使用数据库的人 (people) 所组成的系统。

4. 传统数据库&后关系数据库 (traditional database & post-relational database)

传统数据库:

- 层次模型
- 网状模型
- 关系模型

后关系数据库:

- 面向对象模型 (O-O)
- 对象-关系模型
- 实体-联系模型 (E-R)

5. 数据的语法/语义 (data syntax/semantics)

数据语法:

数据语义:

6. 数据模型/模式 (data model/schema)

数据模型：用于描述数据的静态特性和动态特性的一组概念和定义（反映数据库的某一时刻的状态）
数据模式：对某一类数据的结构、联系和约束的描述（反映数据库的各种数据的结构、属性、联系和约束）

7. 多级数据模型/模式 (multilevel data model/schema)

多级数据模型：

概念数据模型：概念化结构，面向现实世界/用户，与DBMS无关

逻辑数据模型：逻辑结构，面向用户、面向实现，与DBMS有关

物理数据模型：物理存储结构，面向机器世界/实现，与DBMS、OS、硬件有关

多级数据模式：

内模式（用逻辑数据模型描述）：描述数据库中数据是如何存储的，即数据库的物理表示（**physical representation**）

概念模式（用逻辑数据模型描述）：描述数据库中包含什么数据（以及数据之间的关系），即数据库公共视图（**community view**）

外模式（用物理数据模型描述）：描述数据库中特定用户相关的部分，即数据库的用户视图

Why

1. 作为数据管理技术，为什么数据库系统比文件系统优越？

DD（数据字典）使数据库具有自描述性（**self-describing**）

DBMS提供了数据抽象（**data abstraction**）、程序-数据独立性（**independence**）以及一系列数据管理辅助功能（见后文），形成了有效的数据管理方法。

2. 为什么数据库中要采用多级数据模型/模式？

将数据库的物理表示与数据库的用户视图进行分离，即提供数据独立性（**data independence**）。

CHAPTER 2 数据模型

What

1. 关系（数据）模型/模式 (relational data model/schema)

关系数据模型：是以集合论中的关系（**relation**）概念为基础的数据模型

2. 关系/表、元组/行、属性/列、域/数据类型 (relation/table, tuple/row, attribute/column, domain/data type)

关系：笛卡尔积 $D_1 \times D_2 \times \dots \times D_n$ 的某个子集称为定义在域 D_1, D_2, \dots, D_n 上的一个关系

r （**relation**），

元组： $D_1 \times D_2 \times \dots \times D_n = \{ (d_1, d_2, \dots, d_n) \mid d_i \in D_i, i=1, 2, \dots, n \}$ 。集合中的每个元素 (d_1, d_2, \dots, d_n) 称作一个 n -元组（**n-tuple**），简称元组（**tuple**）

属性（**attribute**）：事务的特征

域（**domain**）：属性的取值范围

3. 完整性约束及其类型与作用 (integrity constraints and their types)

完整性约束：语义约束

类型：

域完整性约束（**domain integrity constraints**）

属性值应在域中取值

属性值是否可为NULL？（由语义所决定）

实体完整性约束（**entity integrity constraints**）

每个关系应有一个PK，每个元组的PK值应唯一，且不能为NULL

引用完整性约束（**referential integrity constraints**）

一个关系中的FK值必须引用（另一个关系或本关系中）实际存在的PK值，否则只能暂时取NULL（称悬空引用）

一般完整性约束 / 业务规则（**business rules**）

由特定应用领域中的业务规则所决定，由用户明确地自定义

作用：

4. 键、超键、主键、外键 (key, super-key, primary key, foreign key)

键：关系中满足如下两个条件的属性（组）称为此关系的候选键（**candidate key**），简称为键（**key**）：

a. 决定性条件：这个属性（组）的值唯一地（**uniquely**）决定了其他属性的值（因而也决定/标识了整个元组）；

b. 最小性条件：这个属性（组）的任何真子集（**proper subset**）均不满足决定性条件。

超键：关系中包含（候选）键的属性（组）称为超键（**superkey, i.e., the superset of a key**）。超键 \supseteq 键 或 键 \subseteq 超键

主键：在关系模式机器实现时，从一个关系中（多个）键中选定一个作为此关系模式的键，称被选定者为**主键（primary key, PK）**；其他键称为**候补键（alternate key）**

外键：若一个关系A中某个属性（组）不是本关系的键，但它的值引用了其他关系（或本关系）B中某个键的值，则称此属性（组）为本关系的外键（**foreign key, FK**）。A称为**施引关系（referencing relation）**，B称为**被引关系（referenced relation）**

5. 关系代数操作&关系演算 (relational algebra operation & relational calculus)

关系代数操作：过程性的（**procedural**），由一组操作所组成：传统的集合运算（并、交、差、笛卡尔积，等）和关系专用操作（选择、投影、连接、除，等），每个操作以一个或多个关系为输入，以结果关系为输出

关系演算：非过程性的（**nonprocedural**），使用谓词逻辑（**predicate logic**）来定义所需的结果。根据变量是元组（**tuple**）还是域（**domain**），进一步区分为：元组关系演算 vs. 域关系演算

6. 选择、投影、连接/笛卡尔积、并、差 (select, project, join/Cartesian product, union, difference)

选择（**selection**）

选出关系r中满足<选择条件>的元组，构成结果关系（横向筛选，一元操作）

$\sigma_{\langle \text{选择条件} \rangle} (r) = \{ t \mid t \in r \text{ AND } \langle \text{选择条件} \rangle \}$

投影（**projection**）

选出关系r中<属性表>所列出的诸属性列的值，构成结果关系（纵向筛选，一元操作）

$\pi_{\langle \text{属性表} \rangle} (r) = \{ t [\langle \text{属性表} \rangle] \mid t \in r \}$

笛卡尔积（**Cartesian product**）

$r \times s = \{ \langle t, g \rangle \mid t \in r \text{ AND } g \in s \}$

序偶<t, g>称元组t与元组g的拼接（**concatenation**）

连接（**join**）：从两个关系 r 和 s 的笛卡尔积的所有元组拼接中选出满足<连接条件>者，构成结果关系：

$r \bowtie_{\langle \text{连接条件} \rangle} s = \sigma_{\langle \text{连接条件} \rangle} (r \times s)$

并（**union**）

$r \cup s = \{ t \mid t \in r \text{ OR } t \in s \}$

差（**difference**）

$$r - s = \{t \mid t \in r \text{ AND } (t \notin s) \}$$

7. 关系代数表达式 (relational algebra expression) 【上题】
8. 关系完备操作集&关系完备系统 (relationally complete operation set & relationally complete system)

关系完备操作集: 集合 $\{\sigma, \pi, \cup, -, \times\}$ 或 $\{\sigma, \pi, \cup, -, \bowtie\}$ 是关系完备 (relationally complete) 操作集。

关系完备系统: 支持关系完备操作集的DBMS称关系完备的 DBMS, 或者说DBMS具有关系完备性/relational completeness

9. E/R (数据) 模型/模式, E/R图 (Entity-Relationship data model/schema, E/R graph)

E-R模型: 三种抽象: 实体与弱实体、属性、联系

语义约束: 基数比约束、参与约束

E-R模式: 运用前述E-R数据模型对一个企业/机构的全体数据进行建模后所得的结果称为E-R数据模式, 通常简称为E-R模式 (E-R schema)

E-R图: E-R模式常用表示, 有各种符号体系 (notation),

矩形表示实体, 双线矩形表示弱实体

菱形表示实体间的联系, 用线来连接实体与联系

单线/双线表示实体的部分/全参与, 线上标注基数比

椭圆表示实体/联系的属性, 用单线来连接实体/联系与属性, 实体键 (属性) 进一步有横线标识

10. 实体、弱实体、属性、联系&联系的语义 (entity, weak entity, attribute, relationship & relationship's semantics)

实体集 (entity set): 对同类事物的一种抽象

实体 (entity): 实体集中的一个实例, 是对某类事物中某个具体事物的一种描述

弱实体: 不能独立存在, 依附于其他实体集中的某个实体 (称所有者实体)。弱实体键必须包含其所有者实体的键

属性 (attribute): 对事物 (或事物间联系) 特征的一种抽象

联系 (relationship): 对事物之间某种关系的一种抽象

语义约束: 基数比约束 (cardinality ratio constraints)

对联系 $R(E_1, E_2, \dots, E_n)$,

当 $n = 2$ 时, 二元联系 (binary relationship) 其基数比可以是: $1:1, 1:N, M:N$

当 $n > 2$ 时, 多元联系 (multiway relationship), 如三元联系其基数比可以是:

$1:1:1, 1:1:P, M:N:P$, 等

当 $n = 1$ 时, 自联系/递归联系 (recursive relationship) 其基数比可以是:

$1:1, 1:N, M:N$

参与约束 (participation constraints)

对联系 $R(E_1, E_2, \dots, E_n)$ 中的某个实体集 E_i ,

若所有实体 $e_i \in E_i$ 均参与联系 R , 则称实体集 E_i 是全参与的 (total participation)

若存在实体 $e_i \in E_i$ 不参与联系 R , 则称实体集 E_i 是部分参与的 (partial participation)

11. 传统数据模型&后关系 (数据) 模型 (traditional data model & post-relational data model)

传统数据模型 (traditional data models)

- 层次模型
- 网状模型
- 关系模型

非传统数据模型：后关系模型 (post relational data models)

- 面向对象 (object-oriented, O-O) 模型
- 对象-关系 (object-relational) 模型
- 实体-联系 (entity-relationship, E-R) 模型

Why

1. 为什么关系数据模型会取代层次和网状数据模型，成为数据库的主流数据模型？

关系模型 (relation data model) 的主要特征是用表格结构表达实体集，用外键表示实体间联系。与层次模型和网状模型相比，关系模型比较简单，容易为初学者接受。关系模型的层次、网状模型的最大差别是用键而不是用指针导航数据，其表格简单，用户易懂，用户只需简单的查询语句就可以对数据库进行操作，并不涉及存储结构、访问技术等细节。

2. 为什么要发展后关系数据模型？

传统数据模型不足：以记录为基础，不能很好地面向用户和应用。记录和实体不一定相对应。不能以自然的方式表示实体间联系。语义贫乏。数据类型少，难以满足应用需要

3. 为什么E/R模型能成为数据库概念设计的有力工具？

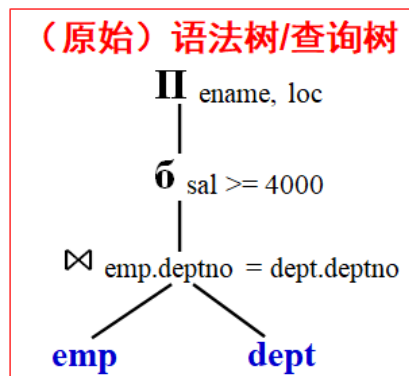
E-R (实体联系数据模型) 与传统数据模型的区别在于：E-R不是面向实现，而是面向现实世界的，因此，它能比较自然地描述现实世界。用E-R表示数据模式时，我们只关心有哪些数据 (即有哪些实体以及属性) 以及数据间的关系 (实体关系)，而不必关心这些数据在计算机内如何表示和用什么DBMS。

How

1. 如何计算一个关系代数表达式？

2. 一个关系代数表达式如何表示成一棵语法树？

$\Pi_{ename, loc} (\sigma_{sal \geq 4000} (\bowtie_{emp.deptno = dept.deptno} emp \text{ } dept))$



3. 如何求一个关系模式的键、超键？如何确定主键、外键？

键：a. 决定性条件：这个属性（组）的值唯一地（**uniquely**）决定了其他属性的值（因而也决定/标识了整个元组）；

b. 最小性条件：这个属性（组）的任何真子集（**proper subset**）均不满足决定性条件。

超键：关系中包含（候选）键的属性（组）

主键：在关系模式机器实现时，从一个关系中（多个）键中选定一个作为此关系模式的键，称被选定者为主键**primary key, PK**；其他键称为候选键（**alternate key**）

外键：若一个关系A中某个属性（组）不是本关系的键，但它的值引用了其他关系（或本关系）B中某个键的值，则称此属性（组）为本关系的外键（**foreign key, FK**）

■ 例1：

学生(学号, 姓名, 班级)

PK = {学号}

课程(课程号, 课程名, 学分)

PK = {课程号}

学生选课(学号, 课程号, 成绩)

PK = {学号, 课程号}

FK1 = {学号}, FK2 = {课程号}

■ 例2：

部门(部门号, 部门名, 地点)

PK = {部门号}

职工(职工号, 姓名, 工种, 主管经理, 所在部门号)

PK = {职工号}

FK1 = {主管经理}, FK2 = {所在部门号}

4. 如何评价传统数据模型？

肯定之处

向用户提供了统一的数据模型（如：关系模型）；

数据与程序之间具有相当程度的独立性；

向用户提供了统一的数据库语言（如：SQL）；

DBMS在数据共享性、安全性、完整性及故障恢复等方面提供了足够的保障。

不足之处

以记录（**record**）为基础，不能很好地面向用户和应用：记录以实现方便为出发点，刻板地描述各种实体（**entity**）

不能以自然的方式（**natural way**）表示实体之间的联系（**relationships between entities**）：实体间联系以面向实现的方式或非显式的方式来表示

语义贫乏（**semantically poor**）：无法明确、显式地描述实体间联系的语义

数据类型少（**few data types**），难以满足应用需要：不支持用户自定义（**user-defined**）数据类型、复杂数据类型、取值规则

CHAPTER 3 关系数据库语言SQL

What

1. 数据库操作&数据库语言（DDL, QL, DML, DCL）（database operations, database language: Data Definition Language, Query Language, Data Manipulation Language, Data Control Language）

数据定义语言（**data definition language, DDL**）：定义、撤销和修改数据模式对象：表、视图、索引...

查询语言（**query language, QL**）：查询数据

数据操纵语言（**data manipulation language, DML**）：插入/删除/修改数据，简单的数值计算与统计功能

数据控制语言（**data control language, DCL**）：控制数据访问权限

2. 面向记录vs面向集合的数据库语言 (record-oriented / set-oriented database language)

层次、网状数据库向用户呈现包含数据的逻辑属性+物理存储细节的数据模式，因此使用物理指针进行导航式访问 (**navigational access**)，一次操作一个记录，操作过程繁琐，效率低下。相应的数据库语言称面向记录的语言 (**record-oriented language**)。

关系数据库的数据模式抽象级别高，使用联想式访问 (**associative access**)，即按数据的内容 (属性值) 来访问数据，一次操作得到一个记录的集合。相应的数据库语言称面向集合的语言 (**set-oriented language**)。

3. 导航式访问vs联想式访问 (navigational access / associative access)

4. 交互式vs嵌入式数据库语言 (interactive/embedded database language)

SQL语言等数据库语言往往不是计算完备的 (**computationally complete**)。

在某些数据库应用中，要实现数据“管理”与“计算”的集成，可以将数据库语言嵌入 (**embedding**) 到程序设计语言 (e.g. Java, C) 中——这样的高级语言称宿主语言 (**host language**)。

因此，数据库语言就有两种或两种使用方式：交互式和嵌入式。

5. 过程性vs说明性数据库语言 (procedural/declarative database language)

早期的层次、网状数据库的语言是命令式/过程性的，使用这样的数据库语言进行数据库查询操作，用户 (程序员) 的负担很重；而且应用程序的可维护性差，程序与数据之间的独立性 (**independence**) 很差。

关系数据库提供了非过程性 (**non-procedural**) / 声明式 (性) 的数据库语言SQL，大大方便了用户对数据库的查询操作。某些过程性机制 (e.g. 流程控制、存储过程，等) 是有用的。因此，SQL语言进行了过程化扩充 (**procedural extensions**)，提供了SQL的扩充模块：SQL/PSM (**Persistent Stored Modules**)。

6. 计算完备vs不完备的语言 (computing complete/incomplete language)

7. 数据库用户接口&前端开发工具 (database user interface & front development tool)

用户接口 (**user interface**)：

用户接口是DBMS提供给用户操作数据库的界面

用户接口将用户对数据库的操作请求以数据库语言语句 (和命令) 的形式提交给系统，并接受系统的处理结果、将结果呈现给用户

用户接口提供了两种操作数据库的方式：交互方式和批处理方式 (即编写应用程序)

用户接口风格可有：文本的和GUI

前端开发工具 (**front-end development tools**)：

DBMS厂商或第三方提供的数据库应用集成化开发工具 (e.g. ORACLE Developer/2000, Sybase Powerbuilder)

8. 基表&视图 (base-table & view)

视图 (**view**)：视图是由其他表 (基表/视图) 导出的虚表 (**virtual table**)，又称为导出表 (**derived table**)，可用于定义外模式

视图与基表 (**base table**) 的区别与联系

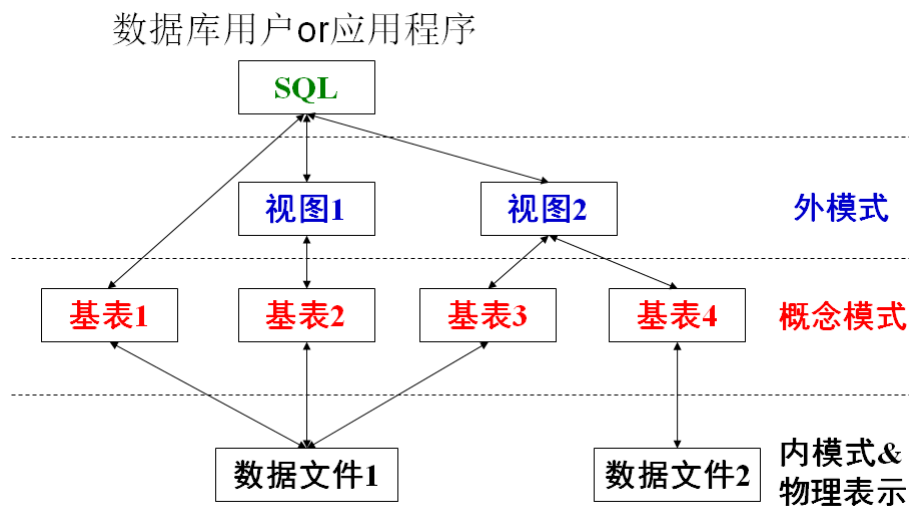
视图如同基表一样，是由行、列组成的二维表，在其中可查询数据、有限制地更新数据

视图中不直接包含数据 (即不对应物理数据文件)，其数据包含在导出它的基表中

基表中的数据发生了变化，由该基表定义的视图中的数据也会随之变化

数据库中只存放视图的定义，不会因为定义了视图而出现数据冗余

当用户在视图上的查询时，DBMS会根据视图定义将视图上的查询转换为用于定义视图的有关基表上的查询——视图消解



9. SQL语言标准 (SQL-89, SQL2, SQL3/SQL:1999) /实现 (SQL standard/implementation)

10. SQL DDL: CREATE TABLE, CREATE VIEW

表 3-3 SQL 模式对象定义语句

语句关键字	模式对象	说 明
CREATE	TABLE、VIEW、TRIGGER、INDEX、CLUSTER、STORED PROCEDURE	创建基表、视图、触发器、索引、簇集、存储过程。
DROP	TABLE、VIEW、TRIGGER、INDEX、CLUSTER、STORED PROCEDURE	删除基表、视图、触发器、索引、簇集、存储过程。
ALTER	TABLE	修改基表, 包括增加列、增加与删除主、外键等。

11. SQL QL: SELECT

表 3-5 SQL 查询语句

语句关键字	语法格式	说 明
SELECT	SELECT [ALL DISTINCT] <列表表达式> [{,<列表表达式>}] FROM <表标识> [<别名>] [{,<表标识> [<别名>]}] [WHERE <查询条件>] [GROUP BY <列标识>[{,<列标识>}] [HAVING <分组条件>]] [ORDER BY <列标识> <序号> [ASC DESC] [{,<列标识> <序号> [ASC DESC]}]];	SELECT 子句中<列表表达式>是算术表达式, 用于投影表中的列, 或对列值进行计算。ALL (默认) 表明返回查询结果的所有行, 不去掉重复行。DISTINCT 对重复行只返回其中一行。
		FROM 子句指明查询的数据来源 (基表或视图)。
		WHERE 中的<查询条件>是逻辑表达式。简单条件有比较、BETWEEN、LIKE、IN 和 EXISTS; 复合条件是由简单条件、逻辑运算符及括号所组成的逻辑表达式。条件中允许嵌入子查询。
		GROUP BY 子句对已选择的行进行分组; HAVING 子句进一步选择已分的组, 对每个已选中的组在查询结果中只返回其单行总计信息。
		ORDER BY 子句对查询结果的显示输出进行排序, ASC (默认) 为升序, DESC 为降序, 排序时遵循“NULL 值最大”原则。

注: “[]” 中内容可出现 0 或 1 次; “{ }” 中内容可出现 1~n 次; 有下划线的关键字可省略。

12. SQL DML: INSERT, DELETE, UPDATE

表 3-6 SQL 数据操纵语句

语句关键字	语法格式	说 明
INSERT	INSERT INTO <表名> [(<属性名> [{, <属性名>}])] {VALUES (<常量> [{, <常量>}]) <查询语句>;	向基表中插入数据。属性列的顺序可与基表定义中的顺序不一致; 属性列表可以被省略。
DELETE	DELETE [<表创建者名>.]<表名> [WHERE <删除条件>;]	从基表中删除数据。若 WHERE 子句缺省, 则删除基表中所有元组, 但基表仍作为一个空表存在于数据库中。
UPDATE	UPDATE [<表创建者名>.]<表名> SET <属性名=表达式> [{, <属性名=表达式>}] [WHERE <更新条件>;]	根据更新条件, 将基表中相应元组的属性值更新为表达式的值。

13. 嵌入式SQL (embedded SQL)

如何将包含外语 (SQL) 的宿主语言源程序编译、连接成可执行代码? ——借助预编译器

预编译器如何识别SQL语句? SQL语句前加特殊标志: EXEC SQL <SQL语句>

DBMS如何识别SQL语句中的宿主语言变量? 宿主语言变量前加标志 (冒号):

程序如何访问数据库? 合法的数据库用户, 通过CONNECT语句连接到数据库:

EXEC SQL CONNECT :username IDENTIFIED BY :password ;

程序工作单元与数据库工作单元之间如何通讯? 通过SQL通讯区SQLCA

程序中如何逐行处理SELECT返回的多行结果? 通过游标 (cursor) 机制 (扩充的SQL语句 EXEC SQL...):

定义游标: DECLARE <游标名> CURSOR FOR

<SELECT语句> ;

打开游标: OPEN <游标名> ;

逐行取数: FETCH <游标名> INTO <主变量列表>;

关闭游标: CLOSE <游标名> ;

14. SQL过程化扩充 (SQL procedural extension)

SQL/PSM主要包括过程化结构 (主要语句见表3-9)、存储过程与函数

存储过程是指使用CREATE PROCEDURE语句事先定义好的过程, 经编译后存储在DBMS中, 供应用调用
函数由CREATE FUNCTION语句定义

表 3-9 持久存储模块 (PSM) 中的主要语句类型

语句类型	语句说明
赋值语句	使用 “=” 运算将一个 SQL 值表达式的结果赋值到一个局部变量、表列或 UDT 属性。
条件语句	使用 IF 语句根据条件选择动作的执行。
选择语句	使用 CASE 语句进行多条件选择。
循环语句	使用 FOR、WHILE、REPEAT 三种语句来定义可根据条件重复执行的一组 SQL 语句。
调用语句	使用 CALL 语句进行过程调用; 使用 RETURN 语句指定 SQL 函数或方法的返回值。
条件处理	使用 DECLARE...HANDLER/CONDITION 和 SIGNAL/RESIGNAL 语句来定义例外或完成条件及其后处理动作。

Why

1. 为什么SQL会成为关系数据库的标准语言?
2. SQL为什么要进行过程化扩充?

为弥补SQL在流程控制方面的不足

How

1. 如何使用各种（交互式）SQL语句？

PART II 系统篇

CHAPTER 4 数据库管理系统引论

What

1. DBMS的组成结构&功能 (DBMS' architecture & functionality)
2. DBMS的进程结构&多线程DBMS (DBMS' process structure & Multithreading DBMS)
3. 数据库系统的体系结构&多层结构 (C/S, 三层结构) (Database system architecture & multi-tier architecture)
4. 事务&ACID性质 (transaction & ACID properties)
5. 显式/隐式的事务提交与回滚 (explicit/implicit transaction commit & rollback)
6. 元数据&数据字典 (metadata & data directory, catalog or dictionary)

Why

1. 事务为什么要具有ACID性质？
2. 数据库系统为什么要向多层结构演变？
3. 数据库系统中为什么要建立数据字典？

How

1. SQL中如何实现事务的提交&回滚？

CHAPTER 5 数据库的存储结构

What

1. 数据库的多级存储 (database multilevel storage)
2. 数据库物理结构&数据文件、日志文件、控制文件 (database physical structure & data file, log file, control file)
3. 基表的典型存储结构：表、索引表、索引簇表、散列簇表 (typical base-table storage structures: table, indexed table, indexed cluster, and hash cluster)
4. B树索引 (B-tree index)
5. 数据库的逻辑存储结构：逻辑存储空间&表空间，用户模式&模式对象 (database logical storage structure: logical storage space & table space, user's schema & schema object)

Why

1. 为什么数据库中要引入逻辑结构的概念？
2. 为什么基表数据的存储要使用索引、簇集等机制？

CHAPTER 6 查询处理和优化

What

1. 查询 (query)
2. 查询处理&求值, 查询引擎 (query processing & evaluation, query engine)
3. 查询执行计划&优化, 优化器 (query execution plan & optimization, optimizer)
4. 查询优化的方法: 层次、目标&策略 (optimization approach: level, objective & strategy)
5. 代数优化 (algebraic optimization)
6. 依赖于存取路径的规则优化 (access-path-dependent rule-based optimization)
7. 代价估算优化 (cost-evaluation-based optimization)

Why

1. 关系系统中为何要进行查询优化?

How

1. 如何用查询语法树表示代数优化的过程?
2. 连接操作如何实现?

CHAPTER 7 事务管理

What

1. 事务管理 (transaction management)
2. 数据库恢复 (database recovery)
3. 后备副本 (backup)
4. 日志及其结构 (log and its structure)
5. 前像/后像&撤消/重做 (before image/after image & undo/redo)
6. 向前恢复/向后恢复 (forward recovery/backward recovery)
7. 提交规则&先记后写规则 (commit rule & log ahead rule)
8. 三类数据库故障及其恢复对策 (3 types of failure and their recovery strategies)
9. 并发访问&并发控制 (concurrent access & concurrency control)
10. 并发控制的正确性准则 (correctness guideline of concurrency control)
11. 合式事务&两段事务 (well-formed transaction & two-phase transaction)
12. 加锁协议&两段封锁协议 (locking protocol & two-phase locking protocol, 2PL)
13. 各类锁: 排它锁, 共享锁, 共享更新锁 (exclusive lock, sharing lock sharing update lock)
14. 封锁粒度、多粒度封锁&意向锁 (locking granularity, multiple-granularity locking & intent lock)
15. 死锁vs活锁 (dead lock & live lock)

Why

1. 为什么数据库要定期备份? 不停地建日志?
2. 为什么更新事务执行要遵循提交规则&先记后写规则?
3. 为什么要进行并发控制?
4. 事务为什么要遵守两段封锁协议?

How

1. 更新事务如何执行?
2. 各类数据库故障如何恢复?
3. 如何保证并发控制的正确性?
4. 数据库系统中如何防止死锁发生?

CHAPTER 8 数据库的安全与完整性约束

What

1. 数据库安全 (database security)
2. 数据库用户名&口令 (database user name & password)
3. 访问控制 (access control)
4. 授权：集中式vs.分散式 (authorization: centralized vs. decentralized)
5. 特权&角色 (privilege & role)
6. 数据库审计&审计痕迹 (database audit & audit trail)
7. 数据加密 (data encryption)
8. 完整性约束及其类型 (integrity constraints and their types)
9. 完整性约束的SQL实现机制：非空、唯一列、主键、外键、检验、断言、触发器 (SQL mechanisms of integrity constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, ASSERTION, TRIGGER)
10. 用过程说明约束 (declaring constraints with procedure)

Why

1. 数据库中为何要引入安全机制？
2. 商用数据库产品为何不支持数据加密？
3. 数据库中为什么要引入完整性约束机制？
4. 为什么用过程说明完整性约束不好？

How

1. 如何用SQL DDL创建/删除角色？
2. 如何用SQL DCL授权/收权？
3. 如何用SQL实现关系数据库的各种完整性约束？

CHAPTER 9 触发子与主动数据库系统

What

1. 被动数据库系统 (passive database system) 与主动数据库系统 (active database system)
2. ECA规则/触发器 (Trigger) 及其各种类型
3. 触发器有哪些典型的用途？

Why

1. 数据库中为何要引入主动机制？

How

1. 如何用SQL's CREATE TRIGGER 语句定义各类触发器？

PART III 应用篇

CHAPTER 10 数据依赖与关系模式规范化

What

1. 冗余&更新异常 (redundancy & update anomaly)

数据冗余 (data redundancy) 重复数据
更新异常 (update anomalies): 修改异常 (modification anomalies) 删除异常 (deletion anomalies) 插入异常 (insertion anomalies)

2. 模式的规范化&模式分解 (schema normalization & decomposition)

通过分解关系模式R来消除其中不合适的数据依赖, 这样的模式分解过程称为关系模式规范化。
将一个关系模式按“语义单纯化”的原则进行合理分解——称模式分解 (decomposition of schema), 以最终达到“一事一地 (One Fact in One Place)”。

3. 函数依赖&决定子 (functional dependency& determinant)

在一个关系模式R(U)中, 如果一部分属性Y \subseteq U的取值依赖于另一部分属性X \subseteq U的取值, 则在属性集X和属性集Y之间存在着一种数据依赖关系, 称之为函数依赖。

设有关系模式R(U), U是其属性集, A、B是U的子集。若对于模式R(U)的任一关系实例 r 中的任意两个元组 t 和s,

t [A] = s [A] \rightarrow t [B] = s [B] 成立, 则称B函数依赖于A或A函数决定B, 记为: A \rightarrow B。A称为决定子。亦可记为: A₁, A₂, ..., A_n \rightarrow B₁, B₂, ..., B_m (A_i, B_j为单个属性)。

注: A不函数决定B, 记为A \nrightarrow B。若A \rightarrow B, B \rightarrow A, 则称两者一一对应, 记为A \leftrightarrow B。

4. 平凡/非平凡/完全非平凡函数依赖 (trivial/nontrivial/full nontrivial functional dependency)

设A、B是某个关系模式的两个属性子集, 对函数依赖A \rightarrow B,

若B \subseteq A, 则称此函数依赖为平凡依赖 (trivial dependency);

若B - A $\neq \Phi$, 则称此函数依赖为非平凡依赖 (nontrivial dependency);

若B \cap A = Φ , 则称此函数依赖为完全非平凡依赖 (completely nontrivial dependency)。

5. 完全/部分函数依赖 (full/part functional dependency)

设A、B是某个关系模式的两个不同属性集,

若有函数依赖A \rightarrow B, 且不存在 C \subset A, 使C \rightarrow B, 则称依赖A \rightarrow B为完全依赖 (full dependency), 记为 A \rightarrow (f) B; 否则称为部分依赖 (partial dependency), 记为 A \rightarrow (p) B。

6. 传递函数依赖 (transitive functional dependency)

A, B, C是某关系模式的三个不同属性集, 若有: A \rightarrow B, B \nrightarrow A, B \rightarrow C, 则称C传递函数依赖于A, 记为: A \rightarrow (t) C。

7. 分裂/合并规则, 平凡依赖规则, 传递规则 (splitting/combining Rule, trivial-dependency rule, and transitive rule)

分裂/合并规则 (the splitting/combining rule)

$X_1, X_2, \dots, X_n \rightarrow Y_1, Y_2, \dots, Y_m \rightarrow X_1, X_2, \dots, X_n \rightarrow Y_1$

...

$X_1, X_2, \dots, X_n \rightarrow Y_m$

平凡依赖规则 (the trivial-dependency rule)

$A \rightarrow B \leftrightarrow A \rightarrow B - A$

传递规则 (the transitive rule)

$(A \rightarrow B, B \rightarrow C) \rightarrow (A \rightarrow C)$

8. 范式, 1NF/非第一范式条件, 2NF, 3NF, BCNF (normal form, non-first normal form, NF2)

范式 (normal form): 是符合某种规范化程度的全体关系模式的一个集合

1NF: 设有一个关系模式 R , 若 R 的任一关系实例 r 中的属性值均是原子数据 (即: 属性都是不可再分的数据项), 则称 R 属于第一范式 (1NF), 记为 $R \in 1NF$ 。

2NF: 设有一个关系模式 $R \in 1NF$, 若 R 的每个非主属性均完全函数依赖于键, 则称 R 属于第二范式 (2NF), 记为 $R \in 2NF$ 。

3NF: 设有一个关系模式 $R \in 1NF$, 若 R 的任一非平凡函数依赖 $X \rightarrow A$ 满足下列两个条件之一: (1) X 是超键, (2) A 是主属性, 则称 R 属于第三范式 (3NF), 记为 $R \in 3NF$ 。

BCNF: 设有一个关系模式 $R \in 1NF$, 若 R 的任一非平凡函数依赖 $\bullet X \rightarrow A$ 均满足下列条件: 决定子 X 必是超键, 则称 R 属于 Boyce-Codd 范式 (BCNF), 记为 $R \in BCNF$ 。

9. 全键关系模式 (all-key relational schema)

10. 函数依赖集及其闭包 (functional dependency set and its closure)

11. 逻辑蕴涵 (logically implicating)

12. 关系模式的一个分解, 无损/保持依赖分解 (decomposition, lossless/ preserve-dependency decomposition)

Why

1. 关系模式为何要规范化?
2. 为什么规范化程度并非越高越好? 具体策略是什么?

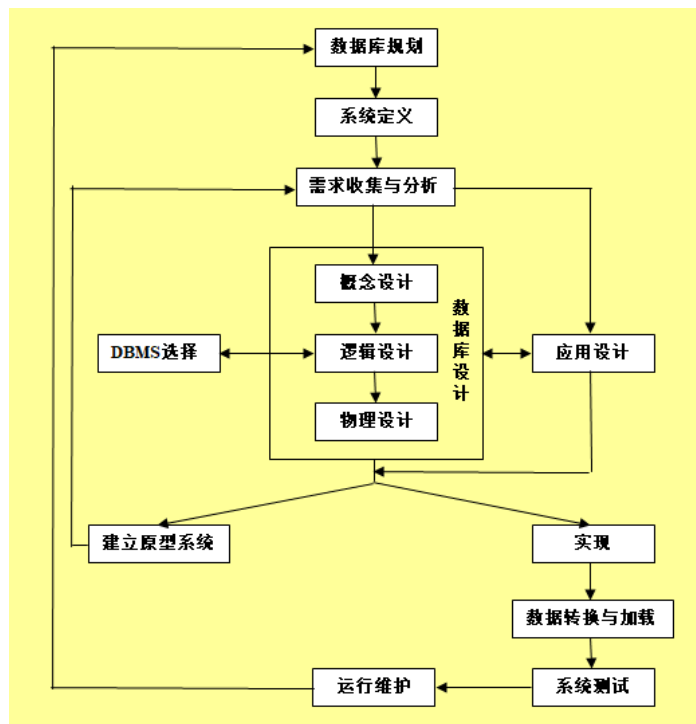
How

1. 如何判定一个关系模式是否属于某个范式?
2. 如何将一个关系模式无损分解到 BCNF/3NF?

CHAPTER 11 数据库设计

What

1. DB 的生命周期 (database's life cycle)



2. DB设计的任务、目标、方法、特点、步骤/阶段 (task, objective, approach, characteristic, stages/phases of database design)

任务：根据一个企业/单位/机构的信息（处理）需求及数据库系统的支撑环境（硬件、网络、OS、DBMS等）的特点，设计出满足用户需求的数据模式（外模式、概念模式、内模式）及典型应用。

目标：对常用的、大多数典型应用（主要是数据库查询）能使用方便、运行性能满意。

方法：(1) 面向过程的方法 (process-oriented approach)

(2) 面向数据的方法 (data-oriented approach)

特点：反复性 (iterative) 试探性 (tentative) 分步性 (multistage)

步骤/阶段：

需求分析 (Requirements Analysis)

定义需求说明 (specification of requirements)

构建元数据库 (metadata repository)

概念设计 (Conceptual Design)

概念建模 (conceptual (E-R/UML) modeling)

逻辑设计 (Logical Design)

模式转换 (E-R/UML to relational conversion)

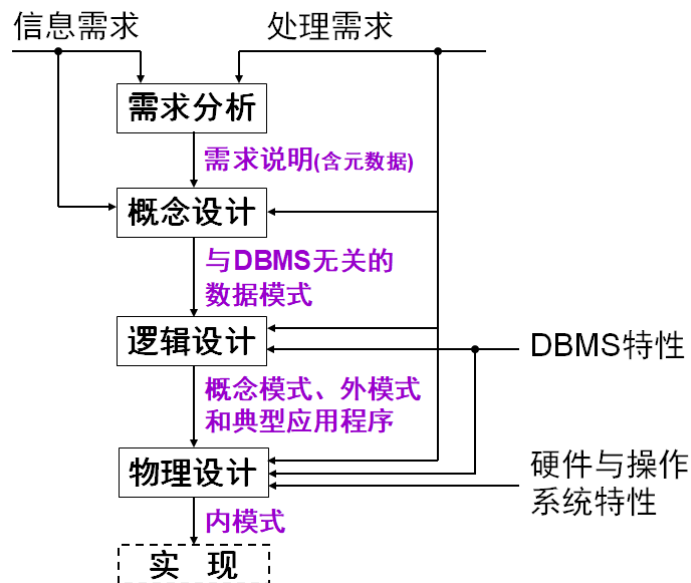
模式规范化 (normalization)

完整性约束设计 (integrity constraints design)

外模式设计 (external schema design)

物理设计 (Physical Design)

内模式设计 (internal schema)



3. 需求分析&需求规格说明 (requirements analysis & specification of requirements)

需求分析 (Requirements Analysis)

定义需求说明 (specification of requirements)

构建元数据库 (metadata repository)

需求规格说明:

功能-实体、功能-业务单位、实体-业务单位矩阵

初步的 (高层的) 实体联系 (E-R) 图

业务功能层次结构 (BFH) 图

数据流图 (DFD)

数据分布、约束、输出格式需求

数据量, 功能使用频率, 用户期望性能

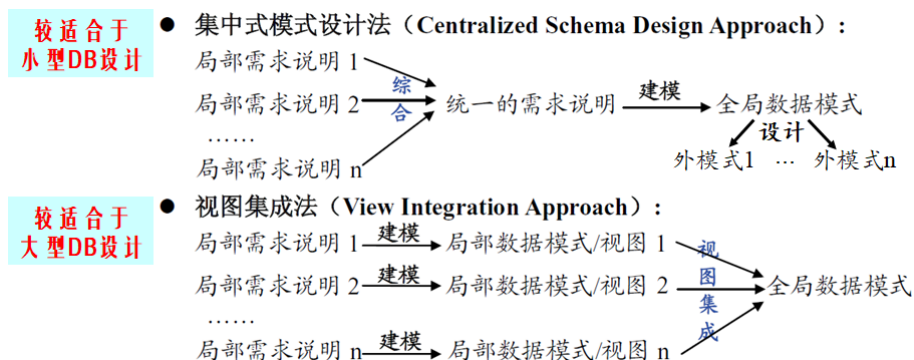
各种策略: 审计, 控制, 后备/恢复, 转换, etc.

4. DB概念设计&视图集成 (DB conceptual design & view integration)

任务: 对数据实体及其联系进行概念建模

技术: 实体联系 (E-R) 建模或UML类图建模

方法: (E-R建模方法--在第2章中已学过)



5. DB逻辑设计&模式转换 (DB logical design)

任务: 设计数据库的逻辑 (概念) 模式和外模式。


技术:

E-R模式向关系模式的转换 (主要技术)


关系模式规范化 (第10章中已学过)

模式的调整

外模式设计

实体及其属性  转换 → 创建关系模式（含属性）

根据规则，通过关系模式间的键引用（FK→PK），将联系及其属性

联系及其属性  转换 → ①尽量置于为参与该联系的某个实体所创建的关系模式中，
②或只能置于为该联系单独创建的关系模式中

6. DB物理设计&存储结构选择（DB physical design & storage structure selection）

任务：设计数据库的内（存储）模式，包括：选择合适的存储机制（存储结构和存取方法），合理安排存储空间。

目标：(1) 提高DB的查询性能（乃主要目的！）(2) 有效利用存储空间（有时与(1)矛盾）

技术：存储机制（存储结构和存取方法）的选择：

无簇表：(1) 普通表

簇表：(3) 散列簇表

(2) 索引的表

(4) 索引簇表

分区设计：数据在多个磁盘上合理分布，以合理地安排I/O，从而提高DB性能。

索引的选择：

建立：一般，PK及UNIQUE属性列上的索引是由DBMS自动建立的；

其他属性列上的索引需用户（一般DBA）手工建立。

使用：索引的使用由DBMS自动决定，即对用户是透明的。

建议：常用来连接（JOIN）相关表的列上宜建索引。

以读为主的表，其常用查询涉及的列上宜建索引。

对于只需访问索引块而不需访问数据块的查询是常用查询的表，其相应列上宜建索引。

不用：很少出现在查询条件中的列。（没意义）

属性取值很少的列。（没效果，e.g. 性别）

数据太少的表。（没必要，全表扫描很快）

属性值分布严重不均匀的列。（反而不如全表扫描）

经常频繁更新的表/列。（索引维护开销大，得不偿失）

属性值过长的列。（难以建索引，如：Oracle规定不能在LONG或LONG RAW类型的列上建索引）

簇集的选择：

建立：簇集需用户（一般是DBA）手工建立。

使用：簇集的使用由DBMS自动决定，即对用户是透明的。（实际中常用散列簇集，较少用索引簇集）

建议：更新较少的表，且对其只进行等值查询时，宜建散列簇集。●---即选用(3)散列簇表

更新较少的表，且对其既进行等值查询又进行范围查询时，可建索引簇集。●---即选用(4)索引簇表

表

不用：更新频繁的表，则宁可选用(1)普通表。

更新较少的表，但对其只进行范围查询时，则宁可用(2)索引的表。

Why

1. 为什么数据库设计要采用面向数据的方法？

以信息需求为主，兼顾处理需求；

较好地反映数据的内在联系，不但可以满足当前应用的需要，还可以满足潜在（将来）应用的需要；

与信息工程方法(IEM)一致：遵循James Martin于1980年代提出的数据中心原理，对业务过程目标进行分析和数据建模的基础上，分阶段开发信息系统的方法。

2. 为什么数据库设计要分阶段进行?

数据库设计往往由不同的人分阶段进行，一是技术上分工的需要，二是为了分段把关，逐级审查，保证审计的质量和进度。

3. 为什么数据库概念设计时要进行概念（e.g. E/R）建模?

在需求分析的基础上，通常用概念数据模型，如E-R数据模型，表示数据及其相互间的联系。概念数据模型是与DBMS无关面向现实世界的数据库模型，因而也易于为用户所理解。

4. 为什么数据库逻辑设计时要进行规范化与逆规范化?

5. 为什么数据库物理设计时要精心选择基表的存储结构?

数据库的内模式虽不直接面向用户，但对数据库的性能影响颇大。DBMS提供相应的DDL语句及命令，供数据库设计人员及DBA定义内模式之用。

How

1. 如何进行视图集成?

确认视图中的对应和冲突

对应是指语义上等价的概念（实体、属性或联系）。

冲突指有矛盾的概念。常见的冲突有：

（1）命名冲突。同名异义；同义异名。

（2）概念冲突。同一概念在一个视图中作为实体，在另一视图中作为属性或联系。

（3）域冲突。相同属性在不同视图中有不同的数据类型或取值范围（包括取值使用不同的度量单位）。

（4）约束冲突。不同视图可能有不同的联系语义约束。

修改视图，解决部分冲突

例如教材P227的图11-2中，【同义异名】“入学时间”和“何时入学”两个属性名可以统一成“入学时间”；

【同名异义】学生分为大学生和研究生两类，课程也分为本科生课程和研究生课程两类；学号一律用字符串表示，等。

合并视图，形成全局模式

尽可能合并对应的部分，保留特殊的部分，删除冗余部分；必要时对模式进行适当修改，力求使模式简明清晰。

2. 如何将E/R数据模式转换为关系数据库模式?

3. 如何为基表选择合适的存储结构?

CHAPTER 12 数据库管理

What

1. 数据库管理&数据库管理员（database management & Database Administrator, DBA）

数据库设计完成后，数据库的实现（创建及数据载入）、运行维护、性能监控、数据库扩充与修改等技术工作的全部统称为数据库管理。

负责数据库管理工作的人（特殊的DB用户）称数据库管理员。

2. DBA的职责 (responsibilities of DBA)

- 1) DBMS及其工具软件的安装与升级;
- 2) 为新系统建立DB及应用，并组织数据的载入;
- 3) 规划、分配与管理DB的存储空间;
- 4) 注册用户，并维护DB的安全;
- 5) DB的日常备份与恢复;
- 6) 维护DB的完整性;
- 7) 监控、审计用户对DB的访问;
- 8) 监控并调整优化DB的性能，必要时重组DB;
- 9) 根据新的用户需求重构DB;
- 10) 制定必要的规章制度，并组织实施。

3. DB调整、重组与重构 (DB adjustment, reorganization & restructuring)

调整 (adjustment)

为了改善DB的性能，对前期设计并实现的DB存储模式进行适当调整；为了适应需求的较小变化，对DB的概念模式、外模式作局部调整。

重组 (reorganization)

为了消除由于DB长期运行而引起的诸多不利因素、改善已恶化的DB性能，对DB物理组织进行一次全局调整。

重构 (restructuring)

为了满足用户新的应用需求，对DB的概念模式、外模式、存储模式进行扩充与修改，从而引起DB及应用程序的一系列改变。（在某种意义上可以认为是对DB的一次重新设计 / re-design）。

Why

1. 为什么数据库系统在运行过程中要由DBA实施管理？

数据库的数据量和数据模式会随应用的开展而不断地扩充和变化，数据库在运行时，会不可避免地要对数据进行更新操作，更新操作会导致存储器上数据结构的恶化。凡是种种变化，都会导致I/O性能下降。数据库设计只是定义数据库初始状态，不可能一劳永逸，为保证数据库能出库、正常地服务，对数据库进行经常地维护和管理是绝对必须地，DBA则负责管好数据库系统，向用户提供可靠、可信和及时地数据库服务。