**数据库基础**

## 一、基本知识

**1 不论做机器学习，还是后台开发，都需要了解数据库的基本知识**，这一篇blog主要是总结数据库的一些基本概念。并不是为了讲解，重点就是总结梳理。数据库主要是面向业务逻辑的，所以为了满足前端业务对数据的需求，数据库管理系统负责完成数据的定义、阻止、存储、管理、操作、事务管理等功能。所以数据库的学习内容里，最核心的就是数据库管理系统DBMS。



**2 相关概念**

1. Data：数据，是数据库中存储的基本对象，是描述事物的符号记录。

2. Database：数据库，是长期储存在计算机内、有组织的、可共享的大量数据的集合。

3. DBMS：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，用于科学地组织、存储和管理数据、高效地获取和维护数据。

4. DBS：数据库系统，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、数据库管理员（DBA）构成。

5. 数据模型：是用来抽象、表示和处理现实世界中的数据和信息的工具，是对现实世界的模拟，是数据库系统的核心和基础；其组成元素有数据结构、数据操作和完整性约束。

6. 概念模型：也称信息模型，是按用户的观点来对数据和信息建模，主要用于数据库设计。

7. 逻辑模型：是按计算机系统的观点对数据建模，用于DBMS实现。

8. 物理模型：是对数据最底层的抽象，描述数据在系统内部的表示方式和存取方法，在磁盘或磁带上的存储方式和存取方法，是面向计算机系统的。

9. 实体和属性：客观存在并可相互区别的事物称为实体。实体所具有的某一特性称为属性。

10.E-R图：即实体-关系图，用于描述现实世界的事物及其相互关系，是数据库概念模型设计的主要工具。

11.关系模式：从用户观点看，关系模式是由一组关系组成，每个关系的数据结构是一张规范化的二维表。

12.型/值：型是对某一类数据的结构和属性的说明；值是型的一个具体赋值，是型的实例。

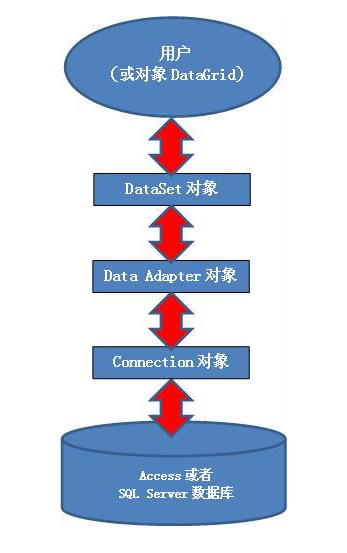
13.数据库模式：是对数据库中全体数据的逻辑结构（数据项的名字、类型、取值范围等）和特征（数据之间的联系以及数据有关的安全性、完整性要求）的描述。

14.数据库的三级系统结构：外模式、模式和内模式。

15.数据库内模式：又称为存储模式，是对数据库物理结构和存储方式的描述，是数据在数据库内部的表示方式。一个数据库只有一个内模式。

16.数据库外模式：又称为子模式或用户模式，它是数据库用户能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图。通常是模式的子集。一个数据库可有多个外模式。

17.数据库的二级映像：外模式/模式映像、模式/内模式映像。



**3 重点知识点**

（1）数据库系统由数据库、数据库管理系统、应用系统和数据库管理员构成。

（2）数据模型的组成要素是：数据结构、数据操作、完整性约束条件。

（3）实体型之间的联系分为一对一、一对多和多对多三种类型。

（4）常见的数据模型包括：关系、层次、网状、面向对象、对象关系映射等几种。

（5）关系模型的完整性约束包括：实体完整性、参照完整性和用户定义完整性。

（6）阐述数据库三级模式、二级映象的含义及作用。

（7）数据库三级模式反映的是数据的三个抽象层次：模式是对数据库中全体数据的逻辑结构和特征的描述。内模式又称为存储模式，是对数据库物理结构和存储方式的描述。外模式又称为子模式或用户模式，是对特定数据库用户相关的局部数据的逻辑结构和特征的描述。数据库三级模式通过二级映象在 DBMS 内部实现这三个抽象层次的联系和转换。外模式面向应用程序， 通过外模式/模式映象与逻辑模式建立联系， 实现数据的逻辑独立性。 模式/内模式映象建立模式与内模式之间的一对一映射， 实现数据的物理独立性。

**4 数据库对数据的组织主要分为层次模型、网状模型、关系模型**

层次模型从数据结构的角度看就是树状结构，而网状模型则是图结构。当然，网状模型更好地描述世界中事物之间的关系，而层次模型受限，但是容易导致数据结构的爆炸式增长。现在最常用的数据模型就是关系模型，即关系数据库。不论是哪一种数据结构组织成的数据库，都可以分为逻辑结构和具体数值两部分，具体存储的数据数值必须按照这种逻辑结构来存储，否则就会报错。而针对某一个应用将逻辑结构确定下来，就成了一个具体的模式。模式就是数据库中全体数据的逻辑结构和特征的描述。

**5 外模式、模式、内模式**

**（1）模式：**是数据库中全体数据的中间层，描述了数据的逻辑结构和特征，是所有用户的公共数据视图。DDL语言可用来描述模式中的逻辑，针对一个DB，只存在一个模式。

**（2）外模式：**是指数据库对外界的应用程序、外界用户呈现出来的一种视图，它并不代表真正的逻辑结构，仅仅是一种映像。如果应用程序不同、用户不同，则外模式就不同。

**（3）内模式：**又称存储模式，一个数据库只有一个存储模式。由于DBMS管理员在建立数据库模式的时候，并不关心底层细节，只关心逻辑通顺与否，所以这里需要构建一个内模式用来将逻辑模式和物理存储方式对应起来。当然，内模式也不仅仅是逻辑和存储的对应，还包括索引、压缩、加密等等。这里就有数据结构里提到的数据库的索引方法，B+树，用来加速数据库查询速度，当然也有hash索引，这种速度更快，但是对内存的消耗还是很大的。

## 二、关系数据库

**1 相关概念**

**（1）主键：** 能够唯一地标识一个元组的属性或属性组称为关系的键或候选键。 若一个关系有多个候选键则可选其一作为主键(Primary key)。

**（2）外键：**如果一个关系的一个或一组属性引用(参照)了另一个关系的主键，则称这个或这组属性为外码或外键(Foreign key)。

**（3）关系数据库：** 依照关系模型建立的数据库称为关系数据库。 它是在某个应用领域的所有关系的集合。

**（4）关系模式**： 简单地说，关系模式就是对关系的型的定义， 包括关系的属性构成、各属性的数据类型、 属性间的依赖、 元组语义及完整性约束等。 关系是关系模式在某一时刻的状态或内容， 关系模型是型， 关系是值， 关系模型是静态的、 稳定的， 而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。

**（5）实体完整性：**用于标识实体的唯一性。它要求基本关系必须要有一个能够标识元组唯一性的主键，主键不能为空，也不可取重复值。

**（6）参照完整性：** 用于维护实体之间的引用关系。 它要求一个关系的外键要么为空， 要么取与被参照关系对应的主键值，即外键值必须是主键中已存在的值。

**（7）用户定义的完整性：**就是针对某一具体应用的数据必须满足的语义约束。包括非空、 唯一和布尔条件约束三种情况。

**2 重要知识点**

（1）关系数据库语言分为关系代数、关系演算和结构化查询语言三大类。

（2）关系的 5 种基本操作是选择、投影、并、差、笛卡尔积。

（3）关系模式是对关系的描述，五元组形式化表示为：R（U，D，DOM，F），其中

R —— 关系名

U —— 组成该关系的属性名集合

D —— 属性组 U 中属性所来自的域

DOM —— 属性向域的映象集合

F —— 属性间的数据依赖关系集合

**3 笛卡尔乘积，选择和投影运算如下**

（1）关系数据库的数据组织形式就是关系模型，从实际应用角度，举一个学生选课数据库比较好理解，但是从理论角度来讲，关系数据模型主要包括域（相同数据类型的值的集合），笛卡尔积（域的集合运算，每一个分量都是一个元组），关系（笛卡尔积的子集叫做在域上的关系，具体到实例里面，一个关系就可以抽出来作为一张数据库的表）。

（2）在一个关系（或一张表）里，主码、候选码、主属性、非主属性（非码属性）等等概念就不赘述了，有时候码又称为键。全码意味着表里的所有属性都是候选码，即缺一不可，缺一则无法定义其中一个元组。需要注意的是，一张表中，候选码才是决定元组唯一的因素，而主码只是候选码其中的一个，主码是从经验角度对候选码的简化。在实际应用过程中，我们可以说整个表中的属性组合就是一个超码，因为每一个元组自身都可以描述自己的唯一性，但是存在一个最小超码，这个最小超码就是候选码，候选码可以有很多个，但是主码只有一个，那就是最方便数据库表操作的一个或一组属性。

（3）关系数据库的基本操作主要有关系演算（查询、插入、删除）；查询操作又包括关系代数（选择、投影、连接、除、并、差、交、笛卡尔积）。

**4 关系数据库的完整性**

**（1）关系数据库的完整性是指数据库中的数据一些语义**，只有保证这些语义正确，才能保证对数据库的操作不会出现错误。

**（2）实体完整性：**表中的每一个元祖都必须主属性完整，也就是候选码中的各个属性必须存在。如果违反了实体完整性，就会拒绝插入或修改，比如插入的主码和表中一个元组一样，或者插入的元组的主码其中一项为空。

**（3）参照完整性：**如果一个属性并非表中的码，但是它又和其他表中的主码，则称该属性为外码。外码可以有两种取值，一种是空值，另一种是对应的其它表中的主码的值。

**（4）如果违反了参照完整性，有若干种解决方案：**一是拒绝执行这个操作；二是级联修改，即外码与对应的表中的主码不一致时，就级联删除或者级联修改，强制它们一致；三就是设置该处为空，这样也可以保证参照完整。具体选择哪种方法视情况而定。

**（5）用户定义完整性：**即具体问题具体分析的完整性定义。SQL提供了CHECK关键字来帮助检查用户定义完整性，即对域的取值设置范围规定。如果对于一个表、或一个模式的完整性条件非常多，每次建立SQL语句都有很多的用户定义完整性需要操作，那么可以采用CONSTRAINT语句将CHECK和SQL语句独立出来，方便每次的修改。

## 三、SQL

**1 相关概念**

（1） SQL：结构化查询语言的简称， 是关系数据库的标准语言。SQL 是一种通用的、 功能极强的关系数据库语言， 是对关系数据存取的标准接口， 也是不同数据库系统之间互操作的基础。集数据查询、数据操作、数据定义、和数据控制功能于一体。

（2）数据定义：数据定义功能包括模式定义、表定义、视图和索引的定义。

（3） 嵌套查询：指将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询。

（4） SQL 数据定义语句的操作对象有：模式、表、视图和索引。

（5） SQL 数据定义语句的命令动词是：CREATE、DROP 和 ALTER。

（6）RDBMS 中索引一般采用 B+树或 HASH 来实现。

（7）索引可以分为唯一索引、非唯一索引和聚簇索引三种类型。

**2 SQL 创建表语句的一般格式为**

CREATE TABLE <表名>

( <列名> <数据类型>[ <列级完整性约束> ]

[，<列名> <数据类型>[ <列级完整性约束>] ] …

[，<表级完整性约束> ] ) ；

其中<数据类型>可以是数据库系统支持的各种数据类型，包括长度和精度。

列级完整性约束为针对单个列(本列)的完整性约束， 包括 PRIMARY KEY、 REFERENCES表名(列名)、UNIQUE、NOT NULL 等。

表级完整性约束可以是基于表中多列的约束，包括 PRIMARY KEY ( 列名列表) 、FOREIGN KEY REFERENCES 表名(列名) 等。

**3 SQL 创建索引语句的一般格式为**

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>

ON <表名> (<列名列表> ) ；

其中UNIQUE：表示创建唯一索引，缺省为非唯一索引；

CLUSTER：表示创建聚簇索引，缺省为非聚簇索引；

<列名列表>：一个或逗号分隔的多个列名，每个列名后可跟 ASC 或 DESC，表示升/降序，缺省为升序。多列时则按为多级排序。

**4 SQL 查询语句的一般格式为**

SELECT [ALL｜DISTINCT] <算术表达式列表> FROM <表名或视图名列表>

[ WHERE <条件表达式 1> ]

[ GROUP BY <属性列表 1> [ HAVING <条件表达式 2 > ] ]

[ ORDER BY <属性列表 2> [ ASC｜DESC ] ] ；

ALL／DISTINCT： 缺省为 ALL， 即列出所有查询结果记录， 包括重复记录。

DISTINCT则对重复记录只列出一条。

算术表达式列表：一个或多个逗号分隔的算术表达式，表达式由常量(包括数字和字符串)、列名、函数和算术运算符构成。每个表达式后还可跟别名。也可用 \*代表查询表中的所有列。

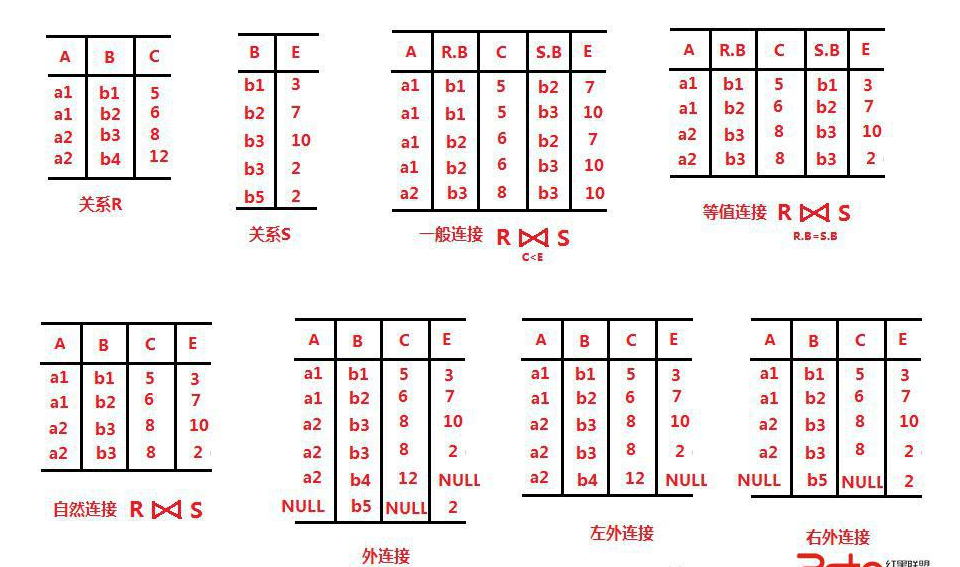
<表名或视图名列表>： 一个或多个逗号分隔的表或视图名。 表或视图名后可跟别名。

条件表达式 1：包含关系或逻辑运算符的表达式，代表查询条件。

条件表达式 2：包含关系或逻辑运算符的表达式，代表分组条件。

<属性列表 1>：一个或逗号分隔的多个列名。

<属性列表 2>： 一个或逗号分隔的多个列名， 每个列名后可跟 ASC 或 DESC， 表示升/降序，缺省为升序。



## 四、数据库的安全性

**1、 在进入数据库时有用户名和密码登录，这个是用户身份鉴定**

**2、 定义用户权限**，其中一种是满足计算机安全性等级C2的自助存取控制DAC：它是指不同的用户对数据库中的对象有不同的权限，且同一用户对不同的数据库对象有不同的权限，权限分为查看、更改和删除等等。另一种满足更高的安全等级B1，即对整个数据库中的对象都进行密级划分，只有满足一定级别权限的用户才能访问这些对象，否则不可以。用户权限的定义里最典型的就是应用程序可以访问的视图，以及DBMS管理员可以访问的模式。也就是说，外界用户只能查看外模式，只有管理员才可以访问修改逻辑模式。在SQL语言中，集成了DCL，所以安排了GRANT和REVOKE两个动词语句来实现对数据库对象的授权和撤回。

**3、 审计，这是一种监测手段，类似于摄像头，**DBMS提供监视操作，将所有的用户和应用程序对数据库的修改、查看等信息全部记录在审计log中，这样就可以对审计日志进行分析，找出哪些非法用户在何时操作了哪些数据。SQL里提供了AUDIT动词法语。

**4、 再有一个方法就是数据加密，**加密技术应用广泛，尤其是在通信中，使用了对称非对称的加密算法，一般对于机密性非常高的数据，以及要在网路上传输的数据才会使用加密技术，因为加密的代价比较大，时间和空间的压力都比较大。

## 五、关系数据库的理论

**1.数据依赖：**反映一个关系内部属性与属性之间的约束关系，是现实世界属性间相互联系的抽象，属于数据内在的性质和语义的体现。

**2. 规范化理论：**是用来设计良好的关系模式的基本理论。它通过分解关系模式来消除其中不合适的数据依赖，以解决插入异常、删除异常、更新异常和数据冗余问题。

**3. 函数依赖：**简单地说，对于关系模式的两个属性子集X和Y，若X的任一取值能唯一确定Y的值，则称Y函数依赖于X，记作X→Y。

**4. 非平凡函数依赖：**对于关系模式的两个属性子集X和Y，如果X→Y，但Y!⊆X，则称X→Y为非平凡函数依赖；如果X→Y，但Y⊆X，则称X→Y为非平凡函数依赖。

**5. 完全函数依赖：**对于关系模式的两个属性子集X和Y，如果X→Y，并且对于X的任何一个真子集X'，都没有X'→Y，则称Y对X完全函数依赖。

**6. 范式：**指符合某一种级别的关系模式的集合。在设计关系数据库时，根据满足依赖关系要求的不同定义为不同的范式。

**7. 规范化：**指将一个低一级范式的关系模式，通过模式分解转换为若干个高一级范式的关系模式的集合的过程。

**8. 1NF：**若关系模式的所有属性都是不可分的基本数据项，则该关系模式属于1NF。

**9. 2NF：**1NF关系模式如果同时满足每一个非主属性完全函数依赖于码，则该关系模式属于2NF。

**10. 3NF：**若关系模式的每一个非主属性既不部分依赖于码也不传递依赖于码，则该关系模式属于3NF。

**11. BCNF**：若一个关系模式的每一个决定因素都包含码，则该关系模式属于BCNF。

**12. 数据库设计：**是指对于一个给定的应用环境，构造优化的数据库逻辑模式和物理结构，并据此建立数据库及其应用系统，使之能够有效地存储和管理数据，满足各种用户的应用需求，包括信息管理要求和数据操作要求。

**13. 数据库设计的6个基本步骤：**需求分析，概念结构设计，逻辑结构设计，物理结构设计，数据库实施，数据库运行和维护。

**14. 概念结构设计：**指将需求分析得到的用户需求抽象为信息结构即概念模型的过程。也就是通过对用户需求进行综合、归纳与抽象，形成一个独立于具体DBMS的概念模型。

**15. 逻辑结构设计：**将概念结构模型（基本E-R图）转换为某个DBMS产品所支持的数据模型相符合的逻辑结构，并对其进行优化。

**16. 物理结构设计：**指为一个给定的逻辑数据模型选取一个最适合应用环境的物理结构的过程。包括设计数据库的存储结构与存取方法。

**17. 抽象**：指对实际的人、物、事和概念进行人为处理，抽取所关心的共同特性，忽略非本质的细节，并把这些特性用各种概念精确地加以描述，这些概念组成了某种模型。

**18. 数据库设计必须遵**循结构设计和行为设计相结合的原则。

**19. 数据字典主要包括数据项、**数据结构、数据流、数据存储和处理过程五个部分。

**20. 三种常用抽象方法是分类、聚集和概括**。

**21. 局部 E-R 图之间**的冲突主要表现在属性冲突、命名冲突和结构冲突三个方面。

**22. 数据库常用的存**取方法包括索引方法、聚簇方法和 HASH方法三种。

**23. 确定数据存放**位置和存储结构需要考虑的因素主要有： 存取时间、 存储空间利用率和维护代价等。

## 六、细说数据库三范式

**1 第一范式（1NF）无重复的列**

第一范式（1NF）中数据库表的每一列都是不可分割的基本数据项

同一列中不能有多个值

即实体中的某个属性不能有多个值或者不能有重复的属性。

简而言之，第一范式就是无重复的列。

在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。

**2 第二范式（2NF）属性完全依赖于主键[消除部分子函数依赖]**

满足第二范式（2NF）必须先满足第一范式（1NF）。

第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。

为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。

第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一

部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是属性完全依赖于主键。

**3 第三范式（3NF）属性不依赖于其它非主属性[消除传递依赖]**

满足第三范式（3NF）必须先满足第二范式（2NF）。

简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。

例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在的员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部

门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性。

**4 具体实例剖析**

下面列举一个学校的学生系统的实例，以示几个范式的应用。在设计数据库表结构之前，我们先确定一下要设计的内容包括那些。学号、学生姓名、年龄、性别、课程、课程学分、系别、学科成绩，系办地址、系办电话等信息。为了简单我们暂时只考虑这些字段信息。我们对于这些信息，说关心的问题有如下几个方面。

1）学生有那些基本信息

2）学生选了那些课，成绩是什么

3）每个课的学分是多少

4）学生属于那个系，系的基本信息是什么。

**（1）首先第一范式（1NF）：**数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。

在当前的任何关系数据库管理系统（DBMS）中，不允许你把数据库表的一列再分成二列或多列，因此做出的都是符合第一范式的数据库。

**（2）我们再考虑第二范式，**把所有这些信息放到一个表中(学号，学生姓名、年龄、性别、课程、课程学分、系别、学科成绩，系办地址、系办电话)下面存在如下的依赖关系。

1）（学号）→ (姓名, 年龄，性别，系别，系办地址、系办电话)

2） (课程名称) → (学分)

3）（学号，课程）→ (学科成绩)

**（3）根据依赖关系我们可以把选课关系表SelectCourse改为如下三个表：**

学生：Student(学号，姓名, 年龄，性别，系别，系办地址、系办电话)；

课程：Course(课程名称, 学分)；

选课关系：SelectCourse(学号, 课程名称, 成绩)。

**（4）事实上，对照第二范式的要求，**这就是满足第二范式的数据库表，若不满足第二范式，会产生如下问题

@数据冗余： 同一门课程由n个学生选修，"学分"就重复n-1次；同一个学生选修了m门课程，姓名和年龄就重复了m-1次。

@更新异常： 1)若调整了某门课程的学分，数据表中所有行的"学分"值都要更新，否则会出现同一门课程学分不同的情况。

2)假设要开设一门新的课程，暂时还没有人选修。这样，由于还没有"学号"关键字，课程名称和学分也无法记录入数据库。

@删除异常 ： 假设一批学生已经完成课程的选修，这些选修记录就应该从数据库表中删除。但是，与此同时，课程名称和学分信息也被删除了。很显然，这也会导致插入异常。

**（5）我们再考虑如何将其改成满足第三范式的数据库表，接着看上面的学生表Student(学号，姓名, 年龄，性别，系别，系办地址、系办电话)，关键字为单一关键字"学号"，因为存在如下决定关系：**

（学号）→ (姓名, 年龄，性别，系别，系办地址、系办电话)

但是还存在下面的决定关系

(学号) → (所在学院)→(学院地点, 学院电话)

即存在非关键字段"学院地点"、"学院电话"对关键字段"学号"的传递函数依赖。

它也会存在数据冗余、更新异常、插入异常和删除异常的情况（这里就不具体分析了，参照第二范式中的分析）。根据第三范式把学生关系表分为如下两个表就可以满足第三范式了：

学生：(学号, 姓名, 年龄, 性别，系别)；

系别：(系别, 系办地址、系办电话)。

1. **单看理论是比较抽象的，一旦结合例子就显得清晰多了。**

关系模型可以用五元组来表示：关系名、一组属性、属性对应的域、属性到域的映射、属性上的一组数据依赖。前四个都比较好理解，数据依赖是指属性之间的约束关系，主要包括函数依赖和多值依赖。

1. **函数依赖是指其中的一些属性能够唯一确定其他属性**，比如一个学生的学号确定下来之后，其姓名、学分等等都确定下来了。将函数依赖这一概念推广到关系数据库中的码，候选码即决定项，非候选码即依赖项。若候选码多于一个，则选择一个是主码，也就是说，关系数据库的模式都满足关系模型。多值依赖是指在一张表中，其中的某一些属性受到其他若干属性的影响进而产生依赖，这样非常容易造成数据冗余，难以处理。
2. **范式理论属于逻辑学的内容。**

**第一范式：**一个关系（或一张表）满足了每一个分量都必须是不可分的数据项，也就是说表中不能含有表，就可以称该关系数据库满足了第一范式。

**第二范式：**在第一范式的基础上，每一个非主属性完全依赖于码，即称满足了第二范式，也就是说，表可以构成一个整体了，数据的属性相互之间有关联。

**第三范式：**在第二范式基础上，数据表中如果不存在码传递决定非主属性，则称该关系模型满足第三范式。这一条有点抽象，解释一下，其含义是主属性都是直接决定非主属性，不存在中间的传递关系。

**BCNF范式：**又称扩展第三范式，指每一个决定因素都包含码在内。

**第四范式：**第四范式比BCNF更加严格，由于函数依赖是多值依赖的一种特殊情况，第四范式的本质含义就是限制关系模型不允许多值依赖出现，只许有函数依赖。消除多值依赖需要SQL投影操作完成。

## 七、数据库编程

**1 嵌入式SQL编程**

（1）其主要方式就是采用高级语言进行流程控制，SQL语言进行数据库操作。主流语言都提供了SQL语言编程接口，将要处理的SQL语句组织成字符串传递给高级语言的接口，高级语言自己调用SQL来传递给DBMS进行处理。

（2）—但是这其间有一个问题，就是嵌入式SQL语句和主语言之间的通信方式：一种是DBMS向主语言编译器、解释器等等传递SQL语句执行的控制状态信息，这样主语言即可根据这些信息来决定如何进行下一步操作，这种方法就是SQL通信区。另一种方法就是主语言向SQL语句提供参数，使用主变量来实现。

（3）在嵌入式SQL中，为了弥合SQL面向集合，每次语句产生或多条记录，而主变量则每次只能存放一条记录这种矛盾，诞生了CURSOR这种东西，使用游标可以方便地操作数据库。游标的本质就是用户自己开设的一个数据缓冲区，将所有的SQL语句的执行结果存放在里面，用户利用该缓冲区将记录逐个附在变量里，告知主语言。

当然，如果查询结果就是单条记录，自然不需要游标。

**2 动态SQL编程**

（1）其目的是使SQL查询语句能够根据运行状态的不同而变化，因为在数据库执行SQL语句的过程中，很可能主语言的需求就变化了，需要更改操作，这时候就需要更改SQL语句，执行新的操作，这个变量的更改就放在主变量里，以适应随时更改。

（2）除此之外，SQL语句本身提供了动态参数功能，即 ?符号，供开发者使用。

**3 存储过程**

在数据库编程中，除了嵌入式SQL，还有PL/SQL，这是SQL语言的过程化扩展，让其实现过程处理。PL/SQL语言中包含了赋值语句、控制结构、循环结构等等，和程序设计语言类似。PL/SQL语言编写的程序的基本结构就是块，块与块之间可以相互嵌套，每一个块完成一个独立的逻辑操作。这有点类似于高级程序语言中的函数概念。

**4 PL/SQL的块主要有两种类型**，命名块和匿名块。匿名块每次执行都需要编译，也不可以保存在数据库中。而命名块就可以在编译之后保存在数据库中，被反复调用，命名块包括存储过程和函数。将若干PL/SQL语句构成的操作过程编译和优化后存放在数据库中，就被称为存储过程，每次执行的时候就调用一次。由于是预先编译过的，所以执行起来不需要语法检查，运行效率很高，所以经常在服务器端使用。另外，存储过程降低了客户机和服务器之间的通信量。如果访问服务器，只要告知服务器若干参数，让服务器调用存储过程，这样执行的效率就很高。在SQL语言中，采用 Procedure 关键词进行存储过程的创建、执行、删除。同样的，也有游标可以用来对其操作。

**5 ODBC编程**

ODBC编程克服了各种不同类型的DBMS的异构性，提供了一个统一的操作接口，方便了程序的移植。使用ODBC开发应用程序，数据库的接口是ODBC api，ODBC程序的另一头是各种异构的数据库比如SQL Server、Oracle等等。我没有使用过ODBC，但是我感觉分布式开发的系统、或者更新换代的系统才会用到ODBC吧。

## 八、查询优化

**1 查询处理是数据库的核心操作**，主要分为查询分析、查询检查、查询优化、查询执行几个阶段。

**2 查询分析：对语句进行词法、语法分析**

**3 查询检查：**是指对语句中的数据对象进行检查，比如说要查询的表存在与否，属性是否存在，查询条件是否符合要求。检查通过后，就把SQL语句转换为等价的关系代数表达式

（一般都是查询树、语法分析树），DBMS会处理这个关系代数表达式。

**4 查询优化：**即代数优化（优化关系代数表达式，即等价变换）和物理优化（基于存取路径的优化）。

**5 在数据库优化中，主要是指选择SELECT的优化 和连接优化**

**6数据库选择SELECT方法**主要有以下几种：

**（1）全表扫描查询：**耗时，简单粗暴。当然如果数量比较小还是比较有效的。

**（2）索引、散列查询：**建立在B+数和哈希表基础上，时间复杂度大大降低，数据量越大越有效。但是索引并不是一定就能够优化，在数据量比较小的时候，构建一颗索引树的时间和空间代价也是比较大的，尤其是当数据库更新频繁的时候，索引树需要经过旋转、增添删改等平衡化操作，而且如果树保存在外存上，读取外存的时间和空间都是很大的开销，散列更是如此。所以，在数据量超级庞大的情况下，再考虑建树，当然了，这些问题DBMS都已经做过优化了。

**（3） 嵌套循环：简单粗暴耗时。**

**（4）排序-合并方法**：依然是采用有序的表来进行表的连接，效率也是大大增加。

## 九、事务

**1 事务：**指用户定义的一个数据库操作，操作可能只包含一条SQL语句，也可能包含多条，比如，一个存储过程就是一个事务，因为它已经编译过了，不可再细分。

**2 事务的特性就包含原子性，**即一个事务要么都做，要么都不做。另外、事务还要确保一致性，即事务处理前后的结果一定保证数据库的状态满足事务处理的预期要求。

**3 事务必须确保隔离性**，即事务的执行不能被其他事务干扰造成无法处理。事物的持续性即事务处理完成后，不会根据某种原因造成数据的改变或者丢失。

**4 数据库中的事务有点类似于操作系统中的进程和线程，**当然数据库的事务处理比进程和线程的处理略简单一些，毕竟处理的仅仅是数据库中的数据，而进程和线程涉及到的上下文很复杂。且事务的操作和并发控制紧紧联系在一起。

## 十、数据库故障恢复

**1 系统故障（DBMS运行的OS出现故障）**、物理故障（磁盘掉磁）、病毒木马（恶意篡改数据）是非事务故障，也就是说，单个DBMS自己无法解决的，DBMS可以克服的、机制最复杂的就是事务故障。

**2 故障解决办法**

**（1）数据转储：**即备份。静态转储即转储其间不允许别的事务插手进来，比如网站团队半夜维护网站；而动态转储允许转储过程中事务进来操作，此时就需要建立事务日志log，记录到底哪些数据动过了。另外，如果数据量太大，还需要增量转储，也就是只转储改变的那一部分。转储技术不仅仅是克服故障的好办法，同样也是实现分布式的手段，分布式不仅仅可以克服故障，还能在存取效率上有很大的提升。现在已经可以实现DBMS自动把刚刚接收到的数据修改记录立即传送到数据转储的镜像上的功能，即数据库镜像。

**（2）登记日志文件：**这是数据库故障恢复的核心，比如说磁盘掉磁了，登记日志文件都没了，那就没什么好说的了，损失已经形成，时光不能倒流。具有检查点的恢复技术其根本也是登记日志文件，只不过需要在检查点附近讨论到底哪些事务需要恢复，哪些事务不需要。

## 十一、事务并发

1 数据库中建立了封锁技术，和OS中的线程锁类似，确保事务的一致性。

2 排他锁：写锁，在增删数据库数据的时候不允许其它事务进来。

3 共享锁：读锁，如果一个事务在读取数据，那么其它事务也可以进来读数据，但是不能写数据。

4 尽管有了锁技术，但是还是会出现各种各样的错误和问题，比如活锁，解决活锁的方法是设置排队队列。而死锁则可以采用预防、诊断、检测等方法来处理，和OS处理死锁的方法一样。同样地，加锁技术还会出现惊群现象，这个也是靠排队队列来解决。

5 在处理并发的过程中，为确保事务的一致性和隔离性，还需要采用可串行化调度保证数据的正确性。基于此需求，诞生了两段锁协议：在事务读写之前，首先要申请锁；在事务处理完成释放封锁之后，不允许再申请锁。这是一个很强的条件，充分条件。

6 为了提高事务处理效率，需要进行多粒度封锁，其含义就是把所有的数据库的对象分为不同的粒度，库级别最高，模式次之，表再次之，最后是一个个的元组。如果一个事务想要申请锁，那么就需要从多粒度树的头结点开始询问，数据库有没有被加锁？表有没有被加锁？如果都没有被加锁，则可以对自己想要操作的数据加锁。意向锁则是对多粒度加锁协议的一种改进，避免访问过多的对象，提高加锁效率。

7 上述技术DBMS都已经广泛实现，也就是说，我们如果非必须只需要使用，而不需要知道其内涵到底是什么。

8 现阶段DBMS对应用程序的并行支持最高效的形式就是构建DBMS进程池，这样可以提高应用程序和DBMS之间的协调性。

## 十二、面向对象的数据库技术

SQL支持面向对象技术，现在的DBMS都提供面向对象功能。也就是说，在数据角度上看，面向对象的RDBMS提供了LOB、MOB、BOOLEAN、ARRAY、DISTINCT等数据类型。也就是说，属性的类型可以自定义了，也可以完成继承，参照也有独立的类型，即引用类型。讲真这些东西我都没有用过，等到用的时候再接触吧。

## 十三、XML数据库：

SQL支持XML，我记得在做文本处理的时候需要将文本转化成结构化的数据，看来这就是了，XML比SQL更加灵活，就是一个层级模型。

## 十四、数据仓库：

这两年大数据这么火，等随后有时间再学习一下数据仓库吧。这里只是列一下基本的概念。数据仓库的目的并非面向业务，数据库才是。所以数据库更追求对数据处理的实时性、便捷性、操作细碎、杂乱、更新速度快。而数据仓库则是面向分析的，目的不在于处理应用程序前端接收的业务，而是为了分析数据。所以数据不进行更新操作，数据量也较数据库大很多，一个事务采集的数据量非常大，主要目的是数据挖掘，供管理分析决策。