数据库——以SQl Sever为例

相关概念

（主要来源于：https://zhuanlan.zhihu.com/p/79552738，个别补充会分别注明引用）

数据库（DB）

- 数据库是长期储存在计算机内，有组织的、可共享的、统一管理的大量数据的集合。
- 数据库中的数据按一定的**数据模型**组织、描述和储存，具有较小的冗余度、较高的数据独立性和易扩展性，并可为各种用户共享。
- 数据库系统的核心：**数据模型。**
- 数据库数据的基本特点：**永久储存、有组织、可共享。**
- **数据库是一种文件的形式，由文件和文件组组成。**

文件

- 主要数据文件：.mdf结尾，用于存放数据和数据库的初始化信息。每个数据库有且只能一个主要数据文件。
- 次要数据文件：.ndf结尾 ，存放除了主要数据文件以外的所有数据的文件。次要数据文件并不是必须的，可以没有。如果有的话，可以是一个，也可以有多个。
- 事务日志文件：.ldf结尾，存放用于恢复数据库的所有日志信息。每个数据库至少要有一个日志文件，也可以有多个。

文件组

- 文件组：数据库文件的一种逻辑管理单位，它将数据库文件分成不同的文件组，方便对文件的分配和管理。
- 文件组分为主文件组和用户自定义的文件组。
- 主文件组：Primary，主要数据文件和没有明确指派给其他文件组的文件。
- 用户自定义的文件组：Create DataBase或 alter database 语句中，fileGroup关键字指定 的文件组。
- 设计原则：（暂时还没看到有什么运用的地方）

  1）文件只能是一个文件组的成员。
  2）文件或文件组不能由一个以上的数据库使用
  3）数据和日志信息不能属于同一个文件或文件组。
  4）日志不能作为文件组的一部分。

数据库管理系统（DBMS）

- 数据库管理系统：位于用户与操作系统（OS）之间的一层数据管理软件。
- 它为用户或应用程序提供访问DB的方法，包括DB的建立、查询、更新及各种数据控制。

数据库系统（DBS）

- 数据库系统：数据库和数据库管理系统组成。

数据管理技术

人工管理阶段（40年代末——50年代中）

文件系统阶段（50年代末——60年代中）

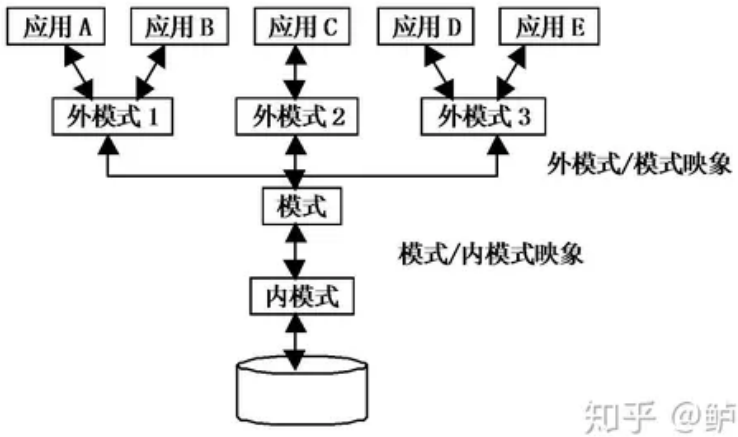
数据库系统阶段（60年代末——80年代）

高级数据库阶段（标志：80年代的分布式数据库、90年代的对象数据库和21世纪初的网络数据库）

关系与非关系数据库

- 关系数据库：建立在关系模型基础上的数据库 SqlServer、MySQL、Oracle 、Access
- 非关系数据库：不同点：不使用SQL作为查询语言。

数据库体系结构



三级模型

数据库的三级模式结构能够提高系统的安全性、可提供数据独立性。

模式（逻辑模式）

- 数据库中全体数据的逻辑结构和特征的描述，所有用户的公共数据视图，综合了所有用户的要求。
- **一个数据库只有一个模式。**
- 优点：数据库模式以某一种数据模型为基础，统一综合地考虑了所有用户的需求，并将这些需求有机地结合成一个逻辑整体。

外模式（子模式/用户模式）

- 数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述。
- 外模式是数据库用户的数据视图、是与某一应用有关的数据的逻辑表示。
- 优点：外模式是保证数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据，数据库中的其余数据是不可见的。

内模式（储存模式）

- 内模式是数据物理结构与储存模式的描述，是数据在数据库内部的组织方式。
- **一个数据库只有一个内模式。**
- 优点：内模式由内模式描述语言来描述、定义所有内部记录类型、索引和文件的组织方式，以及数据控制方面的细节，它是数据库的存储观。

二级映像

外模式/模式映像

- 数据库系统通常都有一个外模式都对应一个外模式/模式映像，它定义了外模式与模式之间的对应关系。
- 这些映像定义通常包含在各自的外模式的描述中。
- 当模式改变时（例如增加新的关系、新的属性、改变属性的数据类型等），由数据库管理员对各个外模式/模式的映像作相应改变，可以使外模式保持不变。应用程序是依据数据的外模式编写的。从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称**数据的逻辑独立性。**

模式/内模式映像

- 模式/内模式映像是唯一的，它定义了数据全局逻辑结构与储存结构之间的对应关系。
- 该映像定义通常包含在模式描述中。
- 当数据库的储存结构改变时（例如选用另一种存储结构），由数据库管理员对模式/内模式映像作相应改变，可以使模式保持不变，从而应用程序也不必改变。保证了数据与程序的物理独立性，简称**数据的物理独立性。**

数据独立性

数据独立性是指**应用程序和数据结构之间相互独立**, 互不影响，包括数据的物理独立性和逻辑独立性。

物理独立性

用户的应用程序与数据库中数据的物理存储是相互独立的。

逻辑独立性

用户的应用程序与数据库的逻辑结构是相互独立的。

数据库的主要功能

数据定义功能

DBMS提供相应数据语言来定义（DDL）数据库结构，它们是刻画数据库框架，并被保存在数据字典中。

数据存取功能

DBMS提供数据操纵语言（DML），实现对数据库数据的基本存取操作：检索，插入，修改和删除。

数据库运行管理功能

DBMS提供数据控制功能，即是数据的安全性、完整性和并发控制等对数据库运行进行有效地控制和管理，以确保数据正确有效。

数据库系统运行控制语言DCL

数据库的建立和维护功能

包括数据库初始数据的装入，数据库的转储、恢复、重组织，系统性能监视、分析等功能。

数据库的传输

DBMS提供处理数据的传输，实现用户程序与DBMS之间的通信，通常与操作系统协调完成。

数据模型

数据模型是数据库的核心。

类型

概念模型

也称信息模型，是按照用户的观点来对数据和信息建模，主要用于数据库设计。

逻辑模型

主要包括层次模型、网状模型、关系模型、面向对象数据模型和对象关系数据模型、半结构化数据模型等。

是按计算机系统的观点对数据建模，主要用于数据库管理系统的实现。

物理模型

物理模型是对数据最底层的抽象，它描述数据在系统内部的表示方式和存取方法，或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。

物理模型的具体实现是数据库管理系统的任务，数据库设计人员要了解和选择物理模型，最终用户则不必考虑物理级的细节。

三要素

数据结构

用于描述现实系统中数据的静态特性。

数据操作

用于描述数据的动态特性。

数据的完整性约束条

用于描述对数据的约束。

联系

描述具有共同特性的实体之间联系集的信息结构，通常包括联系的类型名、联系的属性等。

数据库数据保护功能

（来源于：http://bbs.chinaunix.net/forum.php?mod=viewthread&tid=188100）

数据库安全性

- 数据安全性是指数据库中为保护数据而具备的防御能力（非授权用户无法访问数据）
- 表现在用户标识和鉴别、存取控制、视图机制、审计、数据加密
- 数据库安全可分为二类：系统安全性和数据安全性。

系统安全性

系统安全性是指在系统级控制数据库的存取和使用的机制。
l 有效的用户名/口令的组合；
l 一个用户是否授权可连接数据库；
l 用户对象可用的磁盘空间的数量；
l 用户的资源限制；
l 数据库审计是否是有效的；
l 用户可执行哪些系统操作。

数据安全性

数据安全性是指在对象级控制数据库的存取和使用的机制。
l 哪些用户可存取一指定的模式对象及在对象上允许作哪些操作类型。
l 数据库的存取控制

数据完整性

（来源于：https://blog.csdn.net/gao_zhennan/article/details/78397717）

数据库完整性（Database Integrity）是指数据库中数据在逻辑上的一致性、正确性、有效性和相容性。

实体完整性

- 主键非空且唯一。
- 实体完整性是指在基本表中, 主属性不能取空值 。
- 若属性（只一个或一组属性）A是基本关系R的主属性，则A不能取空值（null value）所谓空值就是“不知道”或“不存在”或“无意义”的值。

参照完整性

- 外键null或者在主键中【注：类似集合外码取值为主码的子集，null类似空集】
- 参照完整性是指在基本表中, 外码是另一个关系主码的有效值或是空值 。
- 若属性（或属性组）F是基本关系R的外码，它与基本关系S的主码Ks相对应，则对于R中每个元组在F上的值必须为：取空值（F的每个属性值均为空值）、或者等于S中某个元组的主码值。

域完整性

- 是指数据库表中的列必须满足某种特定的数据类型或约束。
- 其中约束又包括取值范围、精度等规定。
- 表中的CHECK、FOREIGN KEY 约束和DEFAULT、 NOT NULL定义都属于域完整性的范畴。

用户定义的完整性

- 具体应用的数据符合语义要求。【注：即符合现实世界的要求。如：学生成绩取值1-100，职工年龄大于等于18等，我公司招收本科以上的毕业生。】

数据库设计

六大阶段

（来源于：https://blog.csdn.net/zhonghong666/article/details/88779710）

需求分析阶段

- 分析用户的需求，包括数据、功能和性能需求
- 得到用数据字典描述的数据需求，用数据流图描述的处理需求
- 是整个设计过程的基础，是最困难和最耗费时间的一步。

概念结构设计阶段

- 对需求进行综合、归纳与抽象，形成一个独立于具体DBMS的概念模型
- 主要采用E-R模型进行设计，包括画E-R图。
- 是整个数据库设计的关键，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数据库管理系统的概念模型。

逻辑结构设计阶段

- 将概念结构转换为某个DBMS所支持的数据模型（比如 关系模型 ），并对其进行优化
- 通过将E-R图转换成表，实现从E-R模型到关系模型的转换
- 将概念结构转换为某个数据库管理系统所支持的数据模型，并对其进行优化。
- **外模式、模式在逻辑结构设计阶段得到**

物理结构设计阶段

- 主要是为所设计的数据库选择合适的存储结构和存取路径
- 为逻辑数据结构选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。
- **内模式在物理设计阶段得到。**

数据库实施阶段

- 包括编程、测试和试运行
- 运用DBMS提供的数据语言（比如 SQL）及其宿主语言（例如 C、C++），根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行

数据库运行和维护阶段

- 系统的运行与数据库的日常维护

设计原则

一对一设计原则

在软件开发过程中，需要遵循一对一关系设计原则进而开展数据维护工作，通过利用此原则能够尽量减少维护问题的出现，保证数据维护工作顺利开展同时降低维护工作难度。

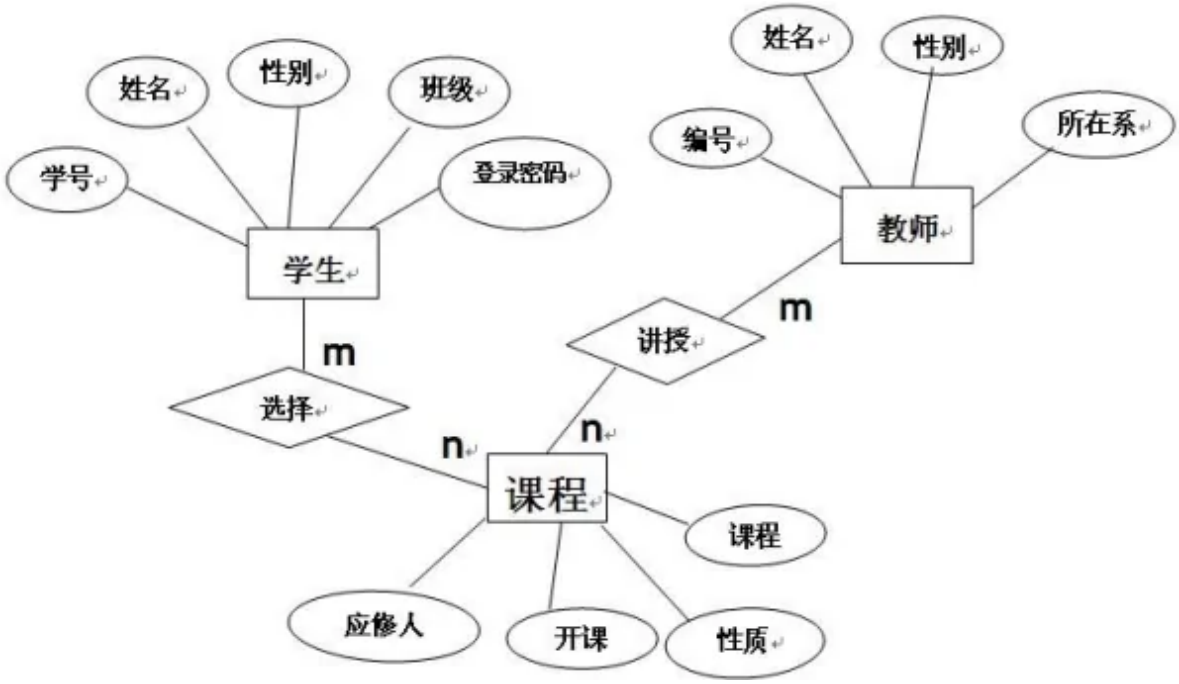
独特命名原则

独特命名原则的应用是为了减少在数据库设计过程中出现重复命名和规范命名现象出现。

双向使用原则

双向使用原则包括：事务使用原则和索引功能原则，软件市场常见的索引模式有：多行检索聚簇索引和单行检索非聚簇索引。

E-R图



组成部分

- 实体：矩形框
- 属性：椭圆形框
- 联系名：菱形框

连接

- 实体与属性之间，实体与联系之间，联系与属性关系连接：连线。

联系

- 实体的三种联系，包含一对一，一对多，多对多三种
- A 到 B 是一对多关系：带箭头的线段指向 B
- A 到 B 是一对一：两个带箭头的线段
- A 到 B 是多对多：两个不带箭头的线段

关系运算

关系的基本运算有两种：传统的集合运算（并、差、交等）和专门的关系运算（选择、投影、自然连接、除法、外连接）。

有些查询需要几个基本操作的组合，并且需要几个步骤才能完成。

传统集合运算

集合运算（并、差、交等）

专门关系运算

专门的关系运算（选择、投影、自然连接、除法、外连接）

数据库对象

（来源于：https://blog.csdn.net/jerry11112/article/details/78160771）

键

- 数据库中的键（key）也可以称为码，是关系模型中的一个重要概念，**它是逻辑结构，不是数据库的物理部分。**
- 键是用于建立和标识表之间的关系,也用于唯一标识表中的任何记录或数据行。
- 在一个关系中，存在一个或多个属性，可以在这个关系中将每个元组唯一标识。（可以粗略的理解：关系=表）

超键(super key)

- 在关系中能唯一标识元组的属性集称为关系模式的超键

候选键(candidate key)

- 不含有多余属性的超键称为候选键。也就是在候选键中，若再删除属性，就不是键了！

全键

- 是候选键的一种特殊情况，如果关系中只有一个候选码,且这个候选码中包含了全部属性,那么这个候选码为全码

主键(candidate key)

- 用户选作元组标识的一个候选键程序

外键(foreign key)

- 如果关系模式R中属性K是其它模式的主键，那么k在模式R中称为外键

例子如下：

学生信息（学号 身份证号 性别 年龄 身高 体重 宿舍号）和 宿舍信息（宿舍号 楼号）

超键：只要含有“学号”或者“身份证号”两个属性的集合就叫超键，例如R1（学号 性别）、R2（身份证号 身高）、R3（学号 身份证号）等等都可以称为超键！

候选键：不含有多余的属性的超键，比如（学号）、（身份证号）都是候选键，又比如R1中学号这一个属性就可以唯一标识元组了，而有没有性别这一属性对是否唯一标识元组没有任何的影响！

主键：就是用户从很多候选键选出来的一个键就是主键，比如你要求学号是主键，那么身份证号就不可以是主键了！

外键：宿舍号就是学生信息表的外键

表

包含数据库中所有数据的对象，行和列组成 ，用于组织和存储数据。

字段

表中的列，一个表可以有多个列，自己的属性——数据类型（决定了该字段存储哪种类型的数据），大小（长度）

视图

表（虚拟表）一张或多张表中导出的表，**用户查看数据的一种方式**，结构和数据是建立在对表的查询基础之上的。

索引

为了给用户提供**一种快速访问数据的途径**，索引是依赖于表而建立，检索数据时，不用对整个表进行扫描，可以快速找到所需的数据。

存储过程

一组为了完成特定功能的SQL语句的集合**（可以有查询、插入、修改、删除），编译后， 存储在数据库中，以名称进行调用， 当调用执行时，这些操作就会被执行。

触发器

在数据库中，属于用户定义的SQL事务**命令集合**，针对于表来说，当对表执行增删改操作时，命令就会自动触发而去执行。

约束

对数据表的列进行的**一种限制。**可以更好的规范表中的列。

缺省值

对表中的列可以指定一个**默认值**，当进行插入时，如果没有为这个列插入值，那么就会自动以预先设置默认值进行自动补充。

数据类型

整型数据类型

浮点型

字符型

Character 字符串

数据类型	描述	存储
char(n)	固定长度的字符串。最多 8,000 个字符。	n
varchar(n)	可变长度的字符串。最多 8,000 个字符。	
varchar(max)	可变长度的字符串。最多 1,073,741,824 个字符。	
text	可变长度的字符串。最多 2GB 字符数据。	

Unicode 字符串：

数据类型	描述	存储
nchar(n)	固定长度的 Unicode 数据。最多 4,000 个字符。	
nvarchar(n)	可变长度的 Unicode 数据。最多 4,000 个字符。	
nvarchar(max)	可变长度的 Unicode 数据。最多 536,870,912 个字符。	
ntext	可变长度的 Unicode 数据。最多 2GB 字符数据。	

日期类型

数据类型	描述	存储
datetime	从 1753 年 1 月 1 日到 9999 年 12 月 31 日，精度为 3.33 毫秒。	8 bytes
datetime2	从 1753 年 1 月 1 日到 9999 年 12 月 31 日，精度为 100 纳秒。	6-8 bytes
smalldatetime	从 1900 年 1 月 1 日到 2079 年 6 月 6 日，精度为 1 分钟。	4 bytes
date	仅存储日期。从 0001 年 1 月 1 日到 9999 年 12 月 31 日。	3 bytes
time	仅存储时间。精度为 100 纳秒。	3-5 bytes
datetimeoffset	与 datetime2 相同，外加时区偏移。	8-10 bytes
timestamp	存储唯一的数字，每当创建或修改某行时，该数字会更新。timestamp 基于内部时钟，不对应真实时间。每个表只能有一个 timestamp 变量。	

货币类型

smallmoney	**介于 -214,748.3648 和 214,748.3647 之间的货币数据。**	**4 字节**
money	介于 -922,337,203,685,477.5808 和 922,337,203,685,477.5807 之间的货币数据。	8 字节

二进制数据类型

数据类型	描述	存储
bit	允许 0、1 或 NULL	
binary(n)	固定长度的二进制数据。最多 8,000 字节。	
varbinary(n)	可变长度的二进制数据。最多 8,000 字节。	
varbinary(max)	可变长度的二进制数据。最多 2GB 字节。	
image	可变长度的二进制数据。最多 2GB。	

其他类型

数据类型	描述
sql_variant	存储最多 8,000 字节不同数据类型的数据，除了 text、ntext 以及 timestamp。
uniqueidentifier	存储全局标识符 (GUID)。
xml	存储 XML 格式化数据。最多 2GB。
cursor	存储对用于数据库操作的指针的引用。
table	存储结果集，供稍后处理。

数据库脚本

- 用于创建数据库对象的语句集合。
- 可以完成移植，提高数据访问效率，完成对数据的相关处理。

SQL语言

- 一种有目的的**编程语言**，用于存取数据、查询、更新和管理关系数据库，高级的非过程化的编程语言。
- SQL语言分为：DQL 数据查询语言、DML 数据操纵语言、DDL 数据定义语言、DCL 数据控制语言

DQL 数据查询语言

例如：

- Select 列　From 表名　where 条件

DML 数据操纵语言

例如：

- Insert 插入　insert into 表名　(列,列，...)　values (对应的值，对应的值)
- Update 更新　update 表名 set 列名=值, 列名=值 where 条件
- Delete 删除　delete from 表名 where 条件

DDL 数据定义语言

创建数据库及其对象，例如：

- Create/Alter/Drop Database/Table/View/Proc/Index

DCL 数据控制语言

用于授予或回收访问数据库某种特权，对数据实行监视等，例如：

- commit 提交　rollback 回滚 grant 授权

T-SQL

- Transact-SQL 基于SQL（Structured Query Language） 结构化查询语言，用于应用程序和数据库之间沟通的编程语言。
- Sql Server 支持的脚本语言。

表

- 唯一标识：一个表中，会存很多条记录，需要一个列来唯一标识一条数据。
- 标识列：一个列设置成标识列，它就不能再手动插入，插入时，自动生成的。这个列必须是整型。
- 标识种子：第一条记录标识列的值。

主键

- 设为主键的列，值不能为空，不能重复。
- 创建一个主键，同时自动创建了一个聚集索引
- 一个表只能有一个主键，也可以没主键，但一般都会设置一个主键。

外键

- 一般在两个表之间要建立关联时候，**一个列创建为外键，它在另一个表必须是主键**
- 一个表里可以有多个外键，也可以没有

约束

- 约束定义：规定表中的数据规则。如果存在违反约束的数据行为，行为就会被阻止。
- 创建约束的时机：

  可视化创建——在创建表之后；

  脚本创建——可以在创建的过程中，也可以在创建后再来建立约束。

主键

- Primary Key约束，唯一性、非空，不能修改

外键

- Foreign Key约束，加强两个表的一列或多列数据之间的连接的。 先建立主表的主键，然后再定义从表中的外键。
- 只有主表中的主键才能被从表用来作为外键使用。
- 主表限制了从表更新和插入的操作。
- 当删除主表中的某条数据，应该是先删除从表中相关的数据，再删除主表。

Unique约束

- 唯一性约束，确保表中的一列数据没有相同的值。
- 与主键约束相似，但又不同：

  主键只能有一个，但一个表中可以定义多个唯一约束。

  唯一键可以为NULL，但主键不可以

Check约束

- 通过逻辑表达式来判断数据的有效性，用来限制输入一列或多列的值的范围。

Default约束

- 默认值约束。
- 用户在插入新的数据行时，如果该行没有指定数据，那么系统将默认值赋给该列，如果没有设置默认值，系统就会默认为NULL。

脚本创建数据库

- master：系统数据库，它记录了SQL Server系统的所有系统级信息，还记录了所有其他数据库的存在，数据库文件的位置，SQL Server的初始化信息。

创建数据库

语法：CREATE DATABASE DatabaseName;

```
create database TestNewBase --数据库名称
on primary --主文件组
(
    name='TestNewBase_data', --数据库数据文件的逻辑名
    filename='C:\Users\14638\Documents\SQLStudy\DBase\TestNewBase.mdf', --主要数据文件路径（绝对路径）
    size=5MB, --数据库主要数据文件的初始大小
    filegrowth=1MB --主要文件的增量
)

log on  --创建日志文件
(
    name='TestNewBase_log', --数据库日志文件的逻辑名
    filename='C:\Users\14638\Documents\SQLStudy\DBase\TestNewBase_log.mdf', --日志文件路径（绝对路径）
    size=5MB, --数据库日志文件的初始大小
    filegrowth=10% --日志文件的增量
)
```

删除数据库

语法：DROP DATABASE DatabaseName;

```
drop database SuibianBase
go
```

修改数据库

（需要直接找）

T-SQL

表操作

创建表

语法：
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY(one or more columns)
);

```
create table ProductInfos
(
    Id int identity(1001,1) primary key not null,  --标识列  identity(标识种子，增量)，primary key设置主键
    ProNo varchar(50) not null,
    ProName nvarchar(50) not null,
    TypeId int not null,
    Price decimal(18,2) null,
    ProCount int default 0 null
)
```

删除表

语法：DROP TABLE table_name;

```
create table ProductType
(
    TypeId int identity(1,1) primary key not null,
    TypeName nvarchar(20) not null
)
```

修改表

重命名表

语法：
ALTER TABLE old_table_name RENAME TO new_table_name;
RENAME old_table _name TO new_table_name;

修改表数据

语法：

```
--添加一列 ProRemark
alter table ProductInfos add ProRemark nvarchar(max) null

--删除一列 ProRemark
alter table ProductInfos drop column ProRemark

--修改一列
alter table ProductInfos alter column ProNo nvarchar(50) null

--修改列名 一般要慎用
--exec sp_rename 'ProductInfos.ProCount','Count','column'
```

表数据操作

插入表数据

语法：

INSERT INTO table_name (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

INSERT INTO table_name VALUES (value1,value2,value3,...valueN);

单条插入

```
insert into 表名 (列名) value ('XXX')
insert into 表名 (列名) select 'XXX'
```

多条插入

```
--方法一
insert into 表名 (列名) values ('XXX'),('XXX'),('XXX'),('XXX')
--方法二
--union去重，union all允许重复 union all效率比union高
insert Test2(MName,Age)
--select 6,'admin',22 union
--select 7,'bbbb',22 union
--select 8,'cccc',21 union
select 'key',21 union  --重复插不进去
select 'key',21

select 'key',21 union all  --重复插得进去
select 'key',21
```

克隆表数据

- 将一张表数据复制到另一张表上

```
--方法一，目标表已经在数据库中存在
insert into 目标表名(目标表列名) --目标表
select 源表列名 from 源表表名 --源表

--方法二，目标表在数据库中不存在，自动创建一个目标表
select 源表列名 into 目标表名
from 源表表名
```

更新表数据

注意：

- 主键不可修改
- 几乎要用where条件定位，不然将把整张表修改

```
--update
update 表名 set 列名='XXX',列名=XXX
where 列名=XXX
```

删除表数据

注意：

- delete会造成标识列值不连续，标识列还是接着删除前的值自增，而不是从初始值开始
- truncate会清空表中的数据，标识列会从初始值开始
- 对比：

  truncate效率比delete高。因为delete每删除一条数据，都会在日志中记录，truncate不会记录，不激活触发器

  truncate是即时操作，不记录rollback，一旦删除,不能恢复，需要慎用。

  delete update insert记录在事务中，可以恢复。

```
--方法一：
delete from 表名 where 列名=XXX
--方法二：
truncate table 表名
```

约束

创建约束

创建表时创建约束

```
create table ProductInfos
(
    Id int identity(1001,1) primary key not null,  --标识列  identity(标识种子，增量)，primary key设置主键
    ProNo varchar(50) unique not null,  --unique约束（唯一约束）
    ProName nvarchar(50) not null,
    TypeId int foreign key references ProductType(TypeId) not null, --foreign key regerences 建表进行外键约束
    Price decimal(18,2) check(Price<10000) default (0.00) not null, --default defalut约束 , check check约束
    ProCount int default 0 null
)
```

建表后添加约束

```
--主键 Id
alter table ProductInfos add constraint PK_ProductInfos primary key(Id)

--外键 TypeId
alter table ProductInfos add constraint FK_ProductInfos foreign key(TypeId)
references ProductType(TypeId)

--unique约束 ProNo
alter table ProductInfos add constraint IX_ProductInfos_ProNo unique(ProNo)

--unique ProNo+ProName
alter table ProductInfos add constraint IX_ProductInfos_ProNo
unique(ProNo,ProName)

--check约束
alter table ProductInfos add constraint CK_ProductInfos_Price check(Price<10000)

--default约束
alter table ProductInfos add constraint DF_ProCount default (0) for ProCount
```

查询数据

单表查询

查询所有数据

```
//查询所有数据
select * from XXX(表名)
```

查询部分数据

```
//查询部分数据
select UserId,UserName,Age from XXX(表名) where 条件
```

列名别名

- 列名 as 别名
- 列名 别名
- 别名=列名

```
//列名别名（三种方式）
select UserId as 用户编号,UserName 用户名,年龄= Age
from XXX(表名)
```

排序

- 主键默认就有排序功能，如果不特意标明是哪种排序，默认升序。
- 降序：从大到小 desc
- 升序：从小到大 asc

```
//排序
select UserId,UserName,Age from XXX(表名)
order by Age,UserId
```

模糊查询

格式

```
select UserName,Age from XXX(表名) where UserName like XXX(匹配语句)
```

匹配模式

- % 0个或多个
- _ 匹配单个字符
- [] 范围匹配 括号中所有字符中的一个
- [^] 不在括号中所有字符之内的单个字符

```
--模糊查询，SQL提供的四种匹配模式

-- % 0个或多个，可以匹配任意类型、任意长度的，也因此效率不高
--种类一：包含于
select * from UserInfos where UserName like '%ad%'
--种类二：以匹配字符或字符串结尾
select * from UserInfos where UserName like '%in'
--种类三：以匹配字符或字符串开头
select * from UserInfos where UserName like 'a%'

-- _ 单个字符，可以限制表达式的字符长度
select * from UserInfos where UserName like '_a'
select * from UserInfos where UserName like '_dm__'
select * from UserInfos where UserName like '__'

-- [] 范围匹配，在括号范围内单个字符
select * from UserInfos where UserName like 'ad[mnd]in'
select * from UserInfos where UserName like '[a-z]a'
```

```
-- [^] 范围匹配，不在括号范围内单个字符
select * from UserInfos where UserName like '[^b-d]a'
```

范围查询

给定范围

- 比较运算符：> < >= <= <>
- 不在这个范围之内：in (2,3,4)　not in (2,3,4)
- between、and：等价于 >= and <=

```
--比较运算符 > < >= <= <>
select * from UserInfos
where Age <30 and DeptId>1

--in (2,3,4)    not in (2,3,4) 不在这个范围之内
select * from UserInfos where Age in (30,35,24)
select * from UserInfos where Age not in (30,35,24)

--between  and 等价于 >= and <=
select * from UserInfos
where Age between 20 and 33   --Age>=20 and Age<=33

 --子查询
 select * from UserInfos
 where DeptId in
 (
   select DeptId from DeptInfos where DeptId>1
 )
```

百分比、前n条

- top n/percent

```
--前面多少条、百比比
select top 10 * from UserInfos
select top 100 percent * from UserInfos
```

聚合函数

- 聚合函数：对一组值执行计算并返回单一的值，经常与select语句中group by结合使用。
- **聚合函数，就是用来输入多个数据，输出一个数据的**
- 五种聚合函数：count 记录个数 sum 求和　avg 求平均 max 最大值 min 最小值

```
--count 统计记录数，一般是用count(1)来统计所有
select count(1) Record from UserInfos  --伪造列,Record是列的别名
select count(*) from UserInfos

--sum 求和
select sum(Age) SUM from UserInfos
```

```
--avg 求平均
select avg(Age) AVG from UserInfos

--max 最大值
select max(Age) MAX from UserInfos

--min 最小值
select min(Age) MIN from UserInfos
```

分组查询

（参考：https://blog.csdn.net/qq_41059320/article/details/89281125）一定要仔细看，不记得就回去看看，很详细！

- 结合聚合函数，根据**一列或多个列对结果集进行分组。**
- having：对分组后虚拟表进行筛选，后跟筛选条件
- 各关键字的顺序：select... where... group by... order by...

```
--统计各部门有多少个用户
select DeptId,count(1) 用户数 from UserInfos where Age>26
group by DeptId
having DeptId>1  --having 对分组后的筛选条件
order by DeptId desc

--例1：查出来的就是有id重复数据的内容。
select id,count(1) from 表名 where 条件
group by id  --group by 字段
having count(1)>1  --having 条件判断
```

连接查询

- 连接查询根据两个或多个表之间的关系，从这些表中查询数据，目的是实现多表查询。
- 分类包括：内连接 、外连接、交叉连接。

内连接

- inner join，使用比较运算符 = > < >= <= <> 进行表间的比较，查询与条件相匹配的数据。
- 结果：相匹配的数据查询出来，显示匹配出来的结果，如果没有匹配上，就没有结果。

显示查询

格式：表 inner join 表 on 条件 where

```
--显示写法 inner join
select UserId,UserName,Age,u.DeptId,DeptName from UserInfos u
Inner join DeptInfos d on d.DeptId=u.DeptId  --d.DeptId>u.DeptId 是不同效果，关联条件不同
where Age>25
```

隐式查询

格式：select .... from 表,表 where 关联条件

```
--隐式写法
select UserId,UserName,Age,u.DeptId,DeptName
from UserInfos u,DeptInfos d
where d.DeptId=u.DeptId and Age>60
```

外连接

外连接分为：左外连接、右外连接 全外连接，可以简称为：左连接、右连接、全连接

左（外）连接

- 格式：left (outer) join on
- 返回左表的所有行，右表中没有匹配上，对应的列就显示NULL
- 输出结果：左表——所有行，右表——行数与左表相同，没有匹配上，显示NULL

```
--左连接：以左边的表为准，右表未匹配上的显示NULL
select * from UserInfos u
left join DeptInfos d  --left (outer) join
on u.DeptId=d.DeptId
```

右（外）连接

- 格式：right (outer) join on
- 与左连接相反，返回的右表的所有行，左表进行匹配，左表中没有匹配上的，对应的列显示NULL
- 输出结果：右表——所有行，左表——行数与右表相同，没有匹配上的，显示NULL

```
--右连接：以右边的表为准，左表未匹配上的显示NULL
select * from UserInfos u
right join DeptInfos d
on u.DeptId=d.DeptId
```

全（外）连接

- 格式：full (outer) join
- 返回左表和右表中所有行，当某一行在另一个表中没有匹配，另一个表中的列返回NULL
- 输出结果：左右表都显示，若没有互相匹配上的，显示NULL

```
--全连接：将左右表都显示出来，未匹配的显示NULL
select * from UserInfos u
full join DeptInfos d
on u.DeptId=d.DeptId
```

交叉连接

- 格式：cross join
- 不带where：返回被连接的两个表的笛卡尔积，返回的行数是两个表行数的乘积。
- 笛卡尔积：第一个对象是X的成员而第二个对象是Y的所有可能有序对的其中一个成员，如集合A={a, b}，集合B={0, 1, 2}，则两个集合的笛卡尔积为{(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)}。
- 带where：返回的是匹配的数据，等价于inner join。

```
--交叉连接：没有where限制时，用左表中的每一条和右表中的每一条匹配，从而显示出m*n条结果
--等价于inner join
select * from UserInfos u
cross join DeptInfos d
where u.DeptId=d.DeptId  --只显示匹配出来的条数
```

类型转换函数

convert

- 格式：convert(类型（长度），表达式)

cast

- 格式：cast(表达式 as 类型)

```
select 2+2
select 'a'+'abssd'

--convert 类型转换函数
select 'cssf'+convert(varchar,2)
select convert(varchar(10),getdate(),126)  --126是不同的日期格式

--cast 类型转换函数
select 'cbd'+cast(2 as varchar)
```

字符串操作函数

！！！数据库中的字符串，索引从1开始。

查找子串

charindex

patindex

```
--返回字符串中指定的字串出现的开始
select CHARINDEX('bc','abcdef')
--返回字符串中指定的字串出现的开始，字串前后必须带%
select PATINDEX('%bc%','abcdef')
```

获取子串

left

格式：left(字符串,i)

作用：**从左往右**选取字符串从第i位开始的字符，作为子串。

right

格式：right(字符串,i)

作用：**从右往左**选取字符串从第i位开始的字符，作为子串。

substring

格式：right(字符串,star,count)

作用：**从右往左**选取字符串从第star位开始（包括第star位）count个字符，作为子串。

注意：count=0，则返回空子串，count < 0会报错。

```
select LEFT('dbsdososx',4)
select RIGHT('deoccsoj',6)
select SUBSTRING('slosfojoxj',3,4)
```

大小写转换

upper

小写--->大写

lower

大写--->小写

```
select upper('abcdDefg')
select lower('ABCDEFG')
```

字符串长度

len

```
select len('ABCD')
```

除去空格

ltrim

作用：去掉第一个字符左边的空格

rtrim

作用：去掉最后一个字符右边的空格

```
select ltrim('    ab    cd      ')
select rtrim('     six  sdc     ')
```

重复字符串

replicate

格式：REPLICATE('string',count)

作用：将指定的字符串重复数遍后，生成新的字符串

```
--abcabcabcabc
select REPLICATE('abc',4)
```

字符串替换

replace

格式：REPLACE ('string1' , 'string2' , 'string3')

作用：用第三个表达式替换第一个字符串表达式中出现的所有第二个给定字符串表达式。

作用：

```
select REPLACE('abcdefg','cd','ss')
```

stuff

格式：stuff('string1' , start , length ,"string2")

作用：删除指定长度的字符，并在指定的起点处插入另一组字符。

```
 --attefg
 --从第二个字符开始的后3个字符被后面的字符串替换。
select STUFF('abcdefg',2,3,'tt')
```

字符串翻转

reverse

```
--gfedcba
select REVERSE('abcdefg')
```

授权语句