数据库概论

1. 前言

数据管理技术

人工管理阶段：50年代中期以前，计算机未普及，主要用于科学计算，外存为磁盘卡片纸带等。用户完全负责数据组织管理，数据完全面向特定应用程序，数据与程序没有独立性。

文件系统阶段：50年代后期到60年代中期，利用文件系统进行数据管理。数据与程序的独立性差，数据共享性差冗余度大，冗余数据不一致，文件系统对多用户并发控制和事务原子性的支持度低，对权限控制和其他安全性问题难以保证。

数据库方法：60年代后期至今，规定数据结构以及支持的操作，以简洁的方式表达不同应用的数据，对用户提供简单且组合起来功能强大的操作集合。数据库中数据和程序的独立性强，支持面向全组织的复杂的数据结构（支持全企业的应用而非像文件系统只支持某个单独的应用），数据冗余度小且易于扩充，便于进行安全管理和权限控制，方便进行并发控制。

数据库的问题：安装和维护成本高；数据集合必须具有足够的结构性，而实际上大部分数据不符合该结构性；支持联机业务数据但很难扩展支持Web级别海量数据。

NoSQL：强化海量数据的高效存储访问和对高度并发读写支持，进一步输的数据模式可以灵活扩展，弱化查询复杂性和数据的一致性。

数据模型

数据模型：数据库系统中用于提供信息标识和操作手段的形式构架；是用来描述数据间关系，数据操作和数据约束的工具。

结构数据模型三要素：**数据结构**（数据本身的特性和数据间的联系），**数据操作**（允许的对数据的操作），**约束条件**（完整性规则的集合，用于保证数据正确性、有效性和相容性）。

实体关系模型：由实体对象和实体间的关系组成，在数据库设计中被广泛应用。

层次模型：以树结构表示实体间的联系的模型，节点表示实体，连线表示两实体的关系。层次模型结构简单易于实现；但数据操纵复杂，并且真正满足树结构约束的数据很少。

网状模型：以图结构表示实体间的联系的模型，节点表示实体，有向边表示两实体间一对多的联系。表达数据的联系的种类丰富；结构复杂，语言复杂。

关系模型：以二维表的形式表示实体间的联系，使用SQL语言查询。表结构简单直观单一，易于理解，请求非过程化，数据独立性强；由于请求非过程化，优化策略必须高效。

数据库模式

模式Schema：某个模型之上对于一组数据的描述，物理模式和逻辑模式分别对应于数据库物理层和逻辑层的设计。模式是所有用户的公共数据视图，是数据库中全体数据的全局逻辑结构和特性的描述。

实例Instance：某个特定的时间点上数据库中存储的实际内容。

外模式Sub-schema：用户的数据视图，是数据的局部逻辑结构，是模式的子集。

内模式Storage-schema：存储模式，是数据的物理结构及存储模式。

外模式/模式映像：某个外模式与模式之间的对应关系。当模式改变，修改映像可以使得外模式保持不变，从而应用程序依然兼容，称作数据的**逻辑独立性。**

模式/内模式映像：数据逻辑结构和存储结构之间的对应关系。当存储结构改变，修改映像可以使得模式保持不变，从而应用程序可以保持不变，称作数据的**物理独立性。**

$$\text{应用} \Leftrightarrow \text{外模式} \overset{\text{外模式/模式映像}}{\Longleftrightarrow} \text{模式} \overset{\text{模式/内模式映像}}{\Longleftrightarrow} \text{内模式} \Leftrightarrow \text{数据库}$$

数据库的构成

数据定义语言DDL：定义数据模式。DDL编译器产生一组表，表的定义保存在数据字典中，数据字典记录了元数据（描述数据的数据）。

数据操纵语言DML：访问和操纵特定数据模型中数据的语言，也称作查询语言。

过程性语言：用户指定需要什么数据，同时指定如何获取这些数据。

非过程性语言：用户指定需要什么数据，而不指定如何获取这些数据。

数据库管理员DBA：协调数据库各种活动，必须对企业信息资源和需求有深刻了解。DBA负责建库（定义模式、外模式、存储结构和存取策略，装入和整理数据），用库（定义完整性约束条件、监视和控制数据库运行、建立后援和回复机制），改进（分析系统性能，重组织和重构造以提升性能）。

事务管理：一个事务是一组操作的集合，完成单一的逻辑功能，具有原子性。数据库并发控制管理并发事务之间的交互，以保证一致性。

2. 关系模型

关系模型的数据结构

域：一组值的集合，这一组值具有相同的数据类型。

笛卡尔积：一组域$D_1, \cdots, D_n$的笛卡尔积如下，笛卡尔积的每个元素称为一个n元组，元组的每个值id为一个分量。笛卡尔积可以写作二维表的形式。

$$D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \cdots, d_n) | d_i \in D_i, i = 1, \cdots, n\}$$

$$card(D_i) = m_i \Rightarrow card(D_1 \times D_2 \times \cdots \times D_n) = \prod_{i=1}^{n} m_i$$

关系：笛卡尔积$D_1 \times \cdots \times D_n$的子集为域$D_1, \cdots, D_n$上的关系，表示为$R(D_1, \cdots, D_n)$，R是关系的名字，n为关系的度。关系可以表示为二维表。

关系的性质：**列同质**，每一列中的分量必须来自同一域；**不同的列可以来自相同的域，但每列属性名不同**；**行列次序无关**；**任意元组不同**；**不可再分性（1NF）**，任何分量都是不可再分的数据。

NULL：每个属性域中的特殊值，表示不存在或值未知。NULL导致数据操作的复杂性。

候选码：关系中的一个属性组，其值可以唯一标识一个元组，并且如果从中去掉任何一个属性，其不再可以唯一标识元组。任何一个候选码中的属性都称作主属性。

主码：设置数据库时，从一个关系的多个候选码中选定一个作为主码。

外部码：关系R中的一个属性组，其不是R的码，但它与另一个关系S中的码相对应，则成这个属性组为R的外部码。

关系模式：关系的描述称作关系模式，记作$R(A_1, A_2, \cdots, A_n)$，其中包括关系名R，关系中的属性名$A_1, \cdots, A_n$，属性/域的映像，属性间的数据依赖等。

关系操作：关系操作都是集合操作，其对象和结果都是集合，可以用关系代数和关系演算两种方式表示。

完整性约束：**实体完整性**要求关系的主码的属性值非NULL；**参照完整性**要求关系的外部码必须为所参照关系中该属性的取值之一，或者为NULL；**用户自定义的完整性**使得用户针对具体应用环境定义其需要的完整性约束。

关系代数

关系代数是过程性语言。

基本操作：选择，投影，并，差，笛卡尔积，重命名。

选择运算：在关系R中选择满足给定条件的元组，F是选择的条件，$\forall t \in R$，$F(t) \in \{True, False\}$，F由逻辑运算（与或非）和算术运算（比较）连接而成。

$$\sigma_F(R) = \{t | t \in R, F(t) = True\}$$

投影运算：从关系R中取出若干列组成新的关系，并对投影的结果进行行去重。

$$\Pi_A(R) = \{t[A] | t \in R\},\ A \subset R$$

并运算：关系R和S相容，将其合并为一个新的关系并进行行去重。

$$R \cup S = \{t | t \in R\ or\ t \in S\}$$

R和S相容⇔R和S包含同样的目R和S对应的属性相容

差运算：两关系相容，所有出现在一个关系而没有出现在另一个关系中的元组的集合。

$$R - S = \{r | r \in R\ and\ r \notin S\}$$

交运算：两关系相容，所有同时出现在两个关系中的元组的集合。交运算可以通过两次作差完成。

$$R \cap S = \{r | r \in R\ and\ r \in S\}$$
$$R \cap S = R - (R - S)$$

笛卡尔积运算：对两关系进行笛卡尔积运算。

$$R \times S = \{concat(r, s) | r \in R, s \in S\}$$

更名运算：给一个关系表达式及其属性赋予名字。在同一关系多次参与同一运算时很有帮助。

$\rho_x(E)$返回表达式E的结果，并把名字x赋给E
$\rho_x(A_1, \cdots, A_n)(E)$返回表达式结果，把名字$x$赋与$E$，同时将各个属性更名为$A_1, \cdots, A_n$

θ连接：从两个关系的笛卡尔积中选取给定属性间满足一定条件的元组。

$$R \bowtie_{A\theta B} S = \sigma_{r[A]\theta s[B]}(R \times S)$$
$$= rs | r \in R\ and\ s \in S\ and\ r[A]\theta s[B]$$

等值连接：θ连接中使用=。

自然连接：$R \bowtie S$，从两个关系的广义笛卡尔积中选取相同属性列，在其上选择取值相等的元组并去除重复行。

除运算：R关系除S关系，要求R关系的属性集合真包含S关系的属性集合，计算时逐个考虑R关系中的元组r，求r在S关系属性列上的分量x，再求x在S关系中的象集C_x，若C_x中包含了所有的C的取值，则x是满足条件的一个元组。

象集：关系R(X, Z)中X和Z是属性组，x是X上的取值，定义x在R中的象集为Z_x，满足下式。

$$Z_x = \{t[Z] | t \in R \; and \; t[X] = x\}$$
$$R \div S = \{x | x = r[S] \; and \; r \in R \; and \; C \subset C_x\}$$
$$R \div S = \Pi_{R-S}(R) - \Pi_{R-S}((\Pi_{R-S}(R) \times S) - \Pi_{R-S,S}(R))$$

除运算的性质：$Q = R \div S$ ⇒Q是满足$Q \times S \subset R$的最大关系。除运算可以通过两次减运算实现。

赋值运算：将关系代数表达式的值赋与临时的关系变量，不支持递归定义，以此将复杂的关系代数表达式划分成几个小的易于理解的部分。

$$\text{临时关系变量} \leftarrow \text{关系代数表达式}$$

外连接：假定向参与连接的一方表中加入一个取值全部为空的行，它和参与连接的另一方表的任何一个未匹配上的元组都可以匹配，用于避免自然连接失配造成信息丢失。

$$\text{外连接} = \text{自然连接} + \text{未匹配元组}$$
$$\text{左外连接} = \text{自然连接} + \text{左侧表中未匹配的元组}$$
$$\text{右外连接} = \text{自然连接} + \text{右侧表中未匹配的元组}$$
$$\text{全外连接} = \text{自然连接} + \text{两侧表中未匹配的元组}$$

聚集函数和分组：聚集函数求一组值的统计信息，返回单一值；分组将一个元组分成若干个组，每个组上使用聚集函数。下式中G为用于分组的属性，F为聚集函数，A为属性名，G将E分为若干组且满足同一组中元组在$G_1, \cdots, G_n$上值相同，不同组值不同。

$$_{G_1,G_2,\cdots,G_n}G \; F_1(A_1), F_2(A_2), \cdots, F_m(A_m)(E)$$

广义投影：在投影列表中使用算术表达式对投影进行扩展。

$$\Pi_{F_1,F_2,\cdots,F_n}(E), \; F_1, \cdots, F_n \text{是算术表达式}$$

数据库删除：将满足条件的元组从关系中删除，是对永久关系的赋值运算。

$$R \leftarrow R - E$$

数据库插入：插入一个指定的元组，或者插入一个查询结果。

$$R \leftarrow R \cup E$$

数据库更新：利用广义投影改变元组某些属性上的值。

$$R = \Pi_{F_1,\cdots,F_n}(R)$$

NULL空值：标明值不存在或为止。**包含空值的算术操作结果也是空，聚集函数忽略空值，去重和分组认为两个空值相等。**

关系代数运算规则

选择运算级联：合取选择运算可以分解为单个选择运算的序列。

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

投影运算级联：投影运算的序列中仅有最后一个有效，其余可以直接忽略。

$$\Pi_{L1}(\Pi_{L2}(\cdots(\Pi_{L_n}(E))\cdots)) = \Pi_{L_1}(E)$$

选择运算交换律：$\sigma_{\theta_1}(\sigma_{\theta_2(E)}) = \sigma_{\theta_2}(\sigma_{\theta_1(E)})$

选择、笛卡尔积与θ连接：

$$\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$$
$$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1\ and\ \theta_2} E_2$$

连接交换律：θ连接和自然连接都满足交换律。

$$E_1 \bowtie_{(\theta)} E_2 = E_2 \bowtie_{(\theta)} E_1$$

连接结合律：自然连接满足结合律，θ连接满足有条件的交换律。

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$
$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2\ and\ \theta_3} E_3 = E_1 \bowtie \theta_1\ and\ \theta_3(E_2 \bowtie \theta_2 E_3)$$，其中θ_2只涉及E_2和E_3的属性

选择运算对θ连接的分配律：选择运算对θ连接具有有条件的分配律。

当θ_0的所有属性只涉及到E_1时，
$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$
当θ_1的所有属性只涉及到E_1，θ_2的所有属性只涉及到E_2时，
$$\sigma_{\theta_1\ and\ \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$

优化策略：尽可能地将选择运算先做（移向语法树叶节点）；合并可能的投影操作；尽可能地将投影运算先做（移向语法树叶节点）；尽可能后做连接和笛卡尔积（移向语法树根节点）。

视图

视图Views：所有不属于概念模型，但是会被用户所见的虚拟关系，称作视图，视图仅有定义被存储在数据库中。视图可以被视为基本表和外模式之间的映象；视图提供了个性化服务和安全性支持。

创建视图：create view VIEW_NAME as <query expression>，建立任意合法的关系代数表达式的视图，名称为VIEW_NAME。

通过视图更新：若需要通过师徒更新，则必须将更新视图的语句转换到对数据库中真实表更新的语句上，这一过程需要考虑更新操作是否符合视图定义，以及更新的语句是否可以转换成唯一的查询。

主码缺失：视图定义中不包括基本表的主码，插入时因为主码为NULL而出错。

聚集函数和算术运算：视图中存在来自聚集函数或算术运算的列，则这些列上不可以进行更新。

物化视图：视图的计算结果被实际存储起来，不仅仅包括定义，而且包括视图内容，物化视图可以看做数据库的Cache加快查询速度。

元组关系演算

元组关系演算：形式化定义如下。其中P为公式，由原子公式和运算符组成；t为元祖变量。

$$\{t|P(t)\}$$

约束变量：元组变量前有∀或∃则称作约束变量，约束变量实现自由变量和其他表中变量的关系描述，可以实现自由变量的"赋值"。

自由变量：不是约束变量的元组变量，查询结果中的变量都是自由变量，其他变量都是约束变量。

原子公式：原子公式包括三类。

$$\begin{aligned}&s \in R，s是关系R中的元组\\&s[x]\theta u[y]，s[x]和u[y]为元组分量且满足比较关系\\&s[x]\theta c，分量s[x]与常量c之间满足比较关系\end{aligned}$$

公式：公式使用递归定义如下。原子公式是公式；若P是公式，则非P也是公式；若P1和P2是公式，则P1 and P2，P1 or P2，P1⇒P2也是公式；若P(t)是公式，R是关系，则$\exists t \in R(P(t))$和$\forall t \in R(P(t))$也是公式。

$$\begin{aligned}P_1 \wedge P_2 &\Leftrightarrow \neg(\neg P_1 \vee \neg P_2)\\\forall t \in R(P(t)) &\Leftrightarrow \neg\exists t \in R(\neg P(t))\\P_1 \Rightarrow P_2 &\Leftrightarrow \neg P_1 \vee P_2\end{aligned}$$

不安全的元组关系演算：元组关系演算中不恰当的描述会产生无限的结果，因此必须使用安全的描述。¬∃就有可能引起不安全的描述。

元组关系演算与关系代数的等价性：关系代数的6种基本操作可以通过元组关系演算实现。

$$\begin{aligned}投影：&\Pi_A(R)=\{t|\exists s \in R(s[A]=t[A])\}\\选择：&\sigma_{F(A)}(R)=\{t|t \in R \wedge F(t[A])\}\\广义笛卡尔积：&R(A)\times S(B)=\{t|\exists u \in R \exists s \in S(t[A]=u[A] \wedge t[B]=s[B])\}\\并：&R \cup S=\{t|t \in R \vee t \in S\}\\减：&R-S=\{t|t \in R \wedge \neg t \in S\}\end{aligned}$$

3. SQL语言

概述

SQL特点：集DDL，DML，DCL于一体；面向集合的操作方式（一次一集合）；**非过程化**；具备自含式和嵌入式语言；语法简单易学。

SQL与关系代数：新标准的SQL语言是关系代数的超集；SQL中引入排序的符号，区分不同的数据行；SQL语言允许重复，并默认进行去重；单值集合可以直接同数值进行比较。

SQL查询基本结构

基本结构

$$SELECT \quad A_1, A_2, \cdots, A_n \quad FROM \quad r_1, r_2, \cdots, r_m \quad WHERE \quad P$$
$$\Leftrightarrow \Pi_{A_1,\cdots,A_n}(\sigma_P(r_1 \times \cdots \times r_2))$$

SELECT子句：可以为列名，*（表示所有属性），算术表达式，聚集函数。SELECT对应于关系代数的投影操作。

FROM子句：列出查询的对象表，并允许对表进行重命名。FROM对应于关系代数的连接操作。

WHERE子句：允许进行比较运算，并通过逻辑运算进行组合，此外可以通过BETWEEN进行范围判断。WHERE子句对应于关系代数的选择操作。

重复元组：SQL运算中允许进行多重集合运算，默认保留重复元组。通过ALL显示指明保留重复元组，通过DISTINCT显示指明去除重复元组。

元组显示顺序：ORDER BY 列名 ASC/DESC。

更名运算：旧名 (AS) 新名。为关系或属性重新命名，可以出现在SELECT和FROM子句之中。

字符串匹配：列名 (NOT) LIKE "字符串"。找出满足匹配规则的字符串，匹配规则中“%”表示匹配0或多个任意字符，“_”表示匹配任意单个字符。

字符串转义：ESCAPE "字符"。定义转义字符，之后可以用转义字符+"%"和转义字符+“_”去匹配“%”和“_”。

字符串范围：[]。任何在指定范围内的字符。[a-f]或[abcdef]。

关系连接：连接类型包括内连接（INNER JOIN），左外连接（LEFT OUTER JOIN），右外连接（RIGHT OUTER JOIN）和全外连接（FULL OUTER JOIN），连接条件包括自然连接（NATURE），条件连接（ON <谓词>）和USING（USING (A1, A2, ..., An)）。

USING：(A1, A2, ..., An)时两个连接关系的公共属性的子集，元组在其上取值相等，且只出现一次。

集合交（并）操作：INTERSECT/UNION。其左右各为集合，默认自动去重。相当于关系代数中的交和并运算。

集合差操作：EXCEPT。默认自动去重。相当于关系代数中的差运算。

聚集函数：AVG，MIN，MAX，SUM，COUNT。将一列中所有值汇聚为单个统计值。SELECT之后若存在聚集函数，则非聚集函数列必须在GROUP BY中出现。

分组操作：GROUP BY 列名 (HAVING 条件表达式)。GROUP BY将表中元组进行分组，每个分组上使用聚集函数得到单一至；HAVING对分组进行选择，只对符合条件的分组进行聚集。

空值测试：列名 IS (NOT) NULL。判断是否为NULL，若为NULL则返回TRUE，不能写作列名=NULL，因为空值不满足IS NULL之外的任何查找条件。

NULL参与算术运算，则算术表达式值为NULL；NULL参与比较运算，则结果试做FALSE；NULL参与聚集函数，则除去COUNT(*)之外所有聚集函数都忽略NULL。

空值填充：在SELECT子句中可以使用ISNULL(列名，替换值)对空值进行填充。当列为NULL时，则返回替换值。

SQL的嵌套子查询

集合成员资格：表达式 (NOT) IN (子查询)。判断表达式的值是否在子查询结果之中。

集合之间比较：表达式 比较运算符θ SOME/ALL 子查询。SOME要求表达式的值与子查询结果中的一个值相比满足比较运算；ALL则要求全部。

集合基数测试：(NOT) EXISTS/UNIQUE (子查询)。EXISTS测试集合是否为空，判断子查询结果中是否有任何的元组存在。UNIQUE测试子查询结果中是否具有重复元组。

派生关系：(子查询) AS 关系名(列名, 列名, ...)。在FROM子句中使用子查询表达式，使得该子查询的结果命名为一个临时关系表并加以引用。

临时视图：WITH 关系名(列名, 列名, ...) AS (子查询)。定义临时视图，该视图在语句运行结束之后撤销。

定义视图：CREATE VIEW 视图名(列名, 列名, ...) AS (查询表达式) (with check option)。with check option指明对当前视图进行修改时，是否需要检查满足视图定义中的条件。

撤销视图：DROP VIEW 视图名。

事务：事务具备原子性，是一个查询和更新的序列，这个序列要么都成功要么都失败，并且在外界看来，不可能将一个其余的操作插入事务序列之中。事物通过COMMIT进行提交，通过ROLLBACK进行回滚。

SQL数据修改

插入：INSERT INTO 表名[列名, 列名, ...] VALUES (值, 值, 值) / (子查询)。向表中插入指定好值的元组（或几个元组），或插入子查询的结果。

删除：DELETE FROM 表名 [WHERE 条件表达式]。从表中删除符合条件的元组，若没有WHERE子句，则删除整个表中所有元组。

Oracle和SQL Server上支持**TRUNCATE TABLE**直接删除表中所有行，并且比DELETE速度更快，占用资源更少。

更新：UPDATE 表名 SET 列名=表达式/子查询 列名=表达式/子查询 ... WHERE 条件表达式。指定更新哪些列，更新的条件，以及更新后的值。

SQL Server允许 UPDATE ... SET ... FROM ... WHERE ...的语句，从其他表更新本表的数据。

视图修改：视图在修改时，必须将语句转换为在真实表上的修改语句。但会存在主码缺失，聚集函数，Check option不满足等情况出错。

视图更新约束：SELECT子句中目标列不可以包含聚集函数；SELECT子句中不可以使用DISTINCT；不可以包括GROUP BY子句；不能包括算术表达式计算出的列。对于从单个基本表使用选择和投影导出，并包含主码的视图，允许更新。

数据定义语言DDL

域类型：相当于数据类型。

域定义：CREATE DOMAIN 域名 数据类型。相当于C语言中自定义的数据类型。

基本表定义：CREATE TABLE 表名 (列名 数据类型 (DEFAULT 缺省值) (NOT NULL), ..., (CHECK 条件))。其中CHECK条件定义了该表上的完整性约束条件。

基本表定义的修改：ALTER TABLE 表名 ADD/DROP/MODIFY 子句。增加新列或约束/删除列或约束/修改列定义。

插销基本表定义：DROP TABLE 表名。从数据字典中删除该表定义，在其上的所有数据，索引，触发器，约束条件，以及根据它建立的视图均被删除不可用。

索引定义：CREATE (UNIQUE) (CLUSTER) INDEX 索引名 ON 表名 (列名 ASC/DESC, ...)。UNIQUE唯一性索引，不允许表中不同行在索引列上取相同值。CLUSTER聚集索引，将表中元组按此索引项的值排序并物理地聚集在一起，一个表上至多一个聚集索引。

索引删除：DROP INDEX 索引名。

索引：提高查询速度（定位元组，以及表连接时），实现数据约束（主码，候选码），但会增加更新插入和删除的代价。一般在使用频率高，经常用于连接的列上建立索引。

主索引（聚集索引）：顺序文件的记录顺序正是索引搜索码的顺序。主索引通常是主码，一个关系上一般只能有一个主索引。

辅助索引（非聚集索引）：索引搜索码顺序与文件记录顺序不同。

4. 高级SQL

完整性约束

域约束：CREATE DOMAIN 域名 域 CONSTRAIN 约束名 CHECK 值域检查条件。完整性约束中最基本的约束，对插入，修改或删除的值进行检查，要求值满足域约束设定的条件。相当于高级语言中对结构体字段进行值检查。

引用约束：保证外键属性的取值组成的集合继续是其指向的主键属性的取值组成的集合的子集。插入时，系统必须保证外键取值在其指向的主键取值之中；删除被指向的表中元组时，系统检查外键是否指向该元组，之后可以选择拒绝执行，或级联删除（CASCADE）或设置为空；更新被指向表的元组的主键时，类似删除，检查是否有外键指向它，若有则拒绝更改或级联更新；更新外键的值是，类似插入，检查其值是否在主键取值之中。

码声明：在CREATE TABLE时可以使用PRIMARY KEY子句声明主码包括的属性，使用UNIQUE KEY子句声明候选码属性，使用FOREIGN KEY ... REFERENCES子句声明外键属性和外键引用的关系。

引用完整性的维持方式：**级联方式CASCADE**，将依赖关系中的所有外码值进行同主码值一致的操作；**RESTRICT**，只有依赖关系中没有一个外码值与要删除的基本关系的主码值对应时，才可以删除或更新；**置空方式SET NULL/DEFAULT**，删除或更新主码时，将依赖于其的外键置为NULL或缺省值。

断言：断言在整个数据库范围内发挥作用，域约束，主码约束和外键约束都是其特例。通过CREATE ASSERTION 断言名 CHECK 检查条件进行创建。

SQL数据分析和递归SQL

多维数据模型：以属性为轴形成的超立方体，可以在其上的某些维度上进行上钻和下钻操作来进行数据分析。

ROLLUP命令：增强GROUP BY子句，生成在指定属性列表的每个前缀上分组聚集的并集。ROLLUP(A, B, C)得到的属性集为{(), (A), (A, B), (A, B, C)}。可以与CUBE组合使用，组合使用时两个属性集做笛卡尔积。

CUBE命令：增强GROUP BY子句，生成在指定属性集合列表中所有属性之上聚集的并集。CUBE(A, B)得到的属性集为{(), (A), (B), (A, B)}。可以与ROLLUP组合使用，组合使用时两个属性集做笛卡尔积。

分析子句：函数名() OVER (PARTITION BY 列名, ...) ORDER BY 列名, ... ASC/DESC (NULLS FIRST/NULLS LAST)。函数对每个元组给出一个值，而传统聚集函数每个集合给出一个值；PARTITION BY按类似GROUP BY的规则，将元组按列分组；函数包括ranking（输出按某个值的排序列），dense_rank（输出按某个值的排序列，且出现相同排序时，其后不留空隙）等。

滑动窗口：按照滑动窗口的方式获得一些局部的统计值，同样的元组可以存在于多个窗口之中。

递归SQL：对于未知层数的查询，使用如下递归SQL的方式实现。一般临时表会有一列标明递归层数，用于进行递归层数的控制等。

$$
\begin{aligned}
&WITH\ \text{临时表名}(\text{列名},\text{列名},\cdots)\ AS\\
&(SELECT\ \ldots\ FROM\ \ldots\ WHERE\ \ldots \quad \%\text{给出递归开始的第一层结果}\\
&UNION\ ALL\\
&SELECT\ \ldots\ FROM\ \ldots\ WHERE\ \ldots) \quad \%\text{进行递归，该}SELECT\text{子句使用临时表}\\
&SELECT\ *\ FROM\ \text{临时表名};\quad \%\text{进行查询，获取结果}
\end{aligned}
$$

SQL增强功能

触发器TRIGGER：在数据库发生某种变化时，自动执行触发器中定义的语句。一个触发器包括执行的时机E，执行的条件C和完成的动作A。触发器无法在前台直接调用，而只能被动触发。

触发动作：INSERT，DELETE，UPDATE（可以在特定属性上进行限制）。触发时机：BEFORE，AFTER。引用新旧值：REFERENCING OLD ROW（删除和更新），REFERENCING NEW ROW（插入和更新）。

语句级触发器：上述触发器会在每个受影响的元组上执行动作，语句级触发器可以在特定语句后执行动作。元组级触发器使用FOR EACH STATEMENT，语句级触发器使用FOR EACH ROW。语句级触发器引用新旧值时，使用REFERENCING OLD/NEW TABLE，作为该语句所影响元组的临时表。

存储过程：相当于存储在数据库中的一段程序，系统对其进行编译，适合完成功能单一，逻辑复杂，相对集中的任务。通过CREATE PROCEDURE进行创建，参数包括IN，OUT，INOUT三种；通过CALL来启动存储过程。

优点：安全性强，可以授权用户使用存储过程而不授权访问数据表，可以通过参数化防止SQL嵌入；性能好，减少网络传输代价，运行时不再需要编译和优化；可维护性好，方便扩展和修改，避免客户端重复实现某些功能

缺点：开发复杂度高，难以调试。

SQL应用程序

嵌入式SQL：在高级语言中嵌入SQL语句，通常使用EXEC SQL和END EXEC来进行标记，在其中可以插入SQL语句。需要进行预编译，在预编译过程中检查SQL语法。

数据传递：宿主变量声明在高级语言中的SQL标识之间，可以用在高级语言中，也可以用在SQL语句中，用于进行数据传递。宿主变量前加":"用来和SQL语句的表名列名区分。

```
EXEC SQL SELECT A INTO :tmp_a FROM TABLE_A;
```

游标：SQL语言一次一集合，一般的高级语言（如C语言）一次一记录，因此需要使用游标来逐个存取遍历这些元组。不需要游标的情况包括INSERT，DELETE，UPDATE和返回单个元组的SELECT。

滚动游标：游标可以来回移动，可以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。

声明游标：DECLARE定义一个游标并使之对应一个SELECT语句。

打开游标：OPEN打开一个游标，执行游标对应的查询，结果集合作为该游标的活动集。

检索游标：FETCH在活动集中将游标移动到特定元组，并取出该行数据放到响应宿主变量中。

关闭游标：CLOSE关闭游标，释放活动集。允许再次OPEN。

释放游标：FREE删除游标，之后不能再对其进行OPEN。

SQL模块方法：将SQL和编程语言分离，通过使用API实现SQL语句查询等。

SQL安全性控制

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，破坏数据的安全性和有效性。

自主访问控制DAC：基于用户-对象权限访问矩阵和授权机制实现的安全性控制机制。

授权图：权限可以从一个用户向另一个用户传递，用授权图表示。图的节点的用户，有向边表示用户将某权限授予另一用户，图的根节点是DBA。在权限图中一个用户拥有权限的充分必要条件是权限图中有一条从根节点到该用户的路径。

权限收回的基本策略：收回用户U的权限，删除U的所有出边，删除图中所有不包括DBA的独立自环。

SQL权限：读取权限READ，允许读取但不允许修改；插入权限INSERT，允许插入但不允许修改；修改权限UPDATE，允许修改但不允许删除；删除权限DELETE，允许删除；索引权限INDEX，允许创建和删除索引；创建表权限CREATE，允许创建关系表并且拥有该表上所有权限；修改表权限ALTER，允许增加和删除关系中的属性；删除表权限DROP，允许删除关系表。

SQL授权：GRANT 表级权限 ON 表名/视图名 TO 用户 (WITH GRANT OPTION)。表级权限包括SELECT（可以指定列名），UPDATE（可以指定列名），INSERT，DELETE，INDEX，ALTER，DROP，RESOURCE以及ALL；WITH GRANT OPTION表示允许该用户将权限授予别的用户。

SQL回收权限：REVOKE 表级权限 ON 标明/视图名 FROM 用户。收回权限时需要符合权限回收的基本策略。

视图权限：表上的权限不能细化到某个元组，粒度太大；可以在表上定义各种粒度的视图，并进行权限控制，达到细粒度权限控制的效果。

角色ROLE：权限可以赋值给角色，角色被赋值给用户或者其他角色，可以从角色收回权限。角色相当于对象，更加实际地反映了现实中的组织运作情况。

DAC的问题：非法用户对合法用户植入木马，可以访问到其没有权限的数据。

MAC：通过主体，客体和敏感度标记实现安全性控制。主体为DBMS所管理的用户或代表用户的各个进程；客体为受主体控制的文件、基本表、索引、视图等。

敏感度标记：对敏感度标记的等级划分。从高到低为TS（绝密），S（机密），敏感（C），公开（P）。主体的的敏感度标记称作许可证级别，客体的敏感度级别称作密级。

强制存取控制规则：元组作为客体，带有密级标签；主体的许可证级别大于等于客体密级时，主体可以读取响应客体；主体的许可证级别等于客体密级时，主体可以写相应客体。

修正规则：主体许可证级别大于等于客体密级时，主题可以读取相应客体；主体许可证级别小于等于客体密级时，主体可以可以写相应客体。

MLS问题：低许可证等级用户向表中插入时，若存在主键相同的高密级元组，直接拒绝插入的话会泄露信息；因此必须允许插入，并将主键修改为主键+密级。

审计：建立对于某个用户或者某个表之上的操作的记录，所有操作写入一个公共表之中。

数据库安全三权分立：审计员负责数据库系统的审计，安全管理员负责MAC，DBA负责DAC。