**第一章 数据库系统的概念**

**1.1数据库系统的基本概念**

1. **数据**：
   * 是描述现实世界中具体事物或抽象概念的信息载体。
   * 类型包括数字、字符串、日期、图像、声音等。
2. **数据库**：
   * 一个长期存储在计算机中的、组织化的、可共享的数据集合。
   * 特点：**低冗余、高独立性、**组织良好、易于扩展。
3. **数据库管理系统 (DBMS)**：
   * 软件系统，用于高效地存储、维护、查询和处理数据。
   * 功能包括**数据存储管理、安全性和授权控制、事务管理、并发控制等**。
4. **数据库系统 (DBS)**：
   * **包括数据库、DBMS、数据库管理员、应用开发工具、用户和应用系统。**

**1.2数据库系统相比文件系统的优势**

1. **减少数据冗余和不一致性：**
   * 文件系统中，同一数据可能存在多份副本，易引发数据不一致。
   * 数据库系统通过集中管理减少冗余，保证一致性。
2. **提高数据共享与并发控制**：
   * 支持多个用户和应用程序同时访问数据，并通过并发控制机制防止冲突。
3. **简化数据访问**：
   * 提供高级查询语言（如SQL），使数据检索更高效和便捷。
4. **支持数据完整性和安全性**：
   * 数据库系统通过完整性约束保证数据质量，通过安全管理控制访问权限。
5. **支持数据独立性**：
   * 应用程序与数据的存储结构分离，使得数据的物理存储和逻辑设计更容易修改。
6. **可靠的事务处理**：
   * 确保数据操作的原子性（要么完全成功，要么完全失败），保证一致性。

**1.3数据抽象中的三级模式**

1. **子模式（视图层）**：
   * 定义用户视角下的数据表示。
   * 屏蔽底层逻辑和物理存储细节，提供用户所需的视图。
2. **逻辑模式（逻辑层）**：
   * 描述数据库存储什么数据及其之间的关系。
   * 是数据库的核心设计，独立于物理存储实现。
3. **物理模式（物理层）**：
   * 描述数据的物理存储结构，包括索引、文件组织方式等。

**1.4两级映像和数据独立性**

1. **两级映像**：
   * **子模式-逻辑模式映像**：
     + 当逻辑模式发生变化时，只需修改映像关系，子模式保持不变，保证了逻辑数据独立性。
   * **逻辑模式-物理模式映像**：
     + 当物理存储结构改变时，只需调整映像关系，逻辑模式和应用程序无需修改，保证了物理数据独立性。
2. **数据独立性**：
   * **逻辑数据独立性**：
     + 应用程序与逻辑模式分离，当逻辑模式改变时应用程序不受影响。
   * **物理数据独立性**：
     + 应用程序与物理存储分离，当物理模式改变时逻辑模式和应用程序不受影响。

**第二章 关系代数**

**2.1关系**

1. **定义**：
   * 关系是一个有序数据集合，用于表示多个实体及其属性之间的逻辑联系。
   * 从数学角度，关系是一个元组的集合，每个元组是从属性集合到其对应值域的映射。
2. **特点**：
   * 关系中的每个元组（行）是属性（列）及其取值的组合。
   * 数据是无序的，即元组和属性的顺序不影响关系的本质。

**关系模式**

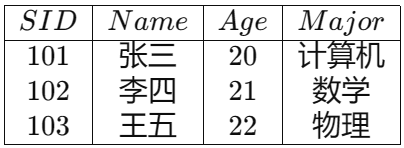
1. **定义**：
   * 关系模式是描述关系的结构和语义约束的集合。
   * **用于定义数据的逻辑结构，包括属性名、属性的类型及约束条件。**
2. **表示形式**：
   * 一个关系模式通常表示为： R(A1,A2,...,An)
   * 其中：
     + R是关系的名称；
     + A1,A2,...,An是关系的属性集合。
3. **特点**：
   * **稳定性**：关系模式是相对固定的，不随着数据的变化而改变。
   * **语义约束**：关系模式包含数据的完整性约束（如主键、外键、唯一性等）。

**关系实例**

1. **定义**：
   * 关系实例是一个具体的关系，在某一时刻存储的数据集合。
   * 它由元组的具体值组成，是现实世界状态的快照。
2. **特点**：
   * **动态性**：关系实例随着时间的变化而改变，因为数据的插入、更新、删除会影响实例的内容。
   * **时间相关性**：关系实例反映的是数据库在某一时间点的状态。
3. **关系模式与关系实例的对比**：
   * **关系模式是结构和规则，关系实例是数据内容。**
   * **关系模式定义了数据的格式，关系实例体现了现实中的实际数据。**

**示例**

1. **关系模式**：
   * 关系名称：学生(Student)
   * 模式定义： Student(SID,Name,Age,Major)
   * 其中：
     + SID：学号（主键）
     + Name：姓名
     + Age：年龄
     + Major：专业
2. **关系实例**（某一时刻的数据）：



**关系的变化：**

* 随着学生数据的增加或修改，关系实例可能变化，例如：
  + 插入新记录：(104,赵六,19,化学)。
  + 修改记录：将张三的专业改为数据科学。
  + 删除记录：移除李四的记录。

**2.2 主码 (Primary Key)**

1. **定义**：
   * **主码是一个关系中用来唯一标识元组（记录）的属性或属性组合。**
   * **在候选码中选择一个作为主码。**
2. **特点**：
   * **唯一性：每个元组在主码上的值是唯一的。**
   * **不可空性：主码的值不能为空（Null）。**
   * **最小性：主码是由最少数量的属性组成的，不存在冗余。**
3. **作用**：
   * 用于快速定位元组，保持数据的一致性和完整性。

**2. 外码 (Foreign Key)**

1. **定义**：
   * **外码是一个关系中的属性或属性组合，用来引用另一个关系的主码。**
   * 用于建立两个关系之间的关联。
2. **特点**：
   * 外码值必须在被引用关系的主码中存在，或为空。
   * 外码值的变更会受到被引用关系的约束。
3. **作用**：
   * 用于实现关系之间的关联，维持引用完整性。

**示例：**

* 表A（学生表）：主码是 StudentID。
* 表B（选课表）：外码 StudentID引用表A的主码 StudentID，表示选课的学生。

**3. 超码 (Super Key)**

1. **定义**：
   * **超码是能够唯一标识关系中元组的属性集合，可以包含冗余属性。**
   * **每个主码都是超码，但反之未必成立。**
2. **特点**：
   * **唯一性**：超码中包含的属性可以唯一标识每个元组。
   * **非最小性**：超码可能包含冗余属性。
3. **例子**：
   * 对于关系 R(StudentID,Name,Age)，以下都是超码：
     + {StudentID}
     + {StudentID,Name}
     + {StudentID,Name,Age}

**4. 候选码 (Candidate Key)**

1. **定义**：
   * **候选码是最小的超码（不能再去掉任何属性），也是主码的候选集合。**
   * **一个关系可能有多个候选码。**
2. **特点**：
   * **唯一性**：可以唯一标识元组。
   * **最小性**：不包含冗余属性。
3. **与主码的关系**：
   * 在候选码中，选定一个作为主码，其他的仍然是候选码。

**例子：**

* 关系 R(StudentID,Name,Email)中：
  + 候选码可以是 {StudentID}和 {Email}。
  + StudentID被选为主码时，Email是候选码。

**它们之间的关系**

* **超码**：所有能唯一标识元组的属性组合。
* **候选码**：最小的超码，没有冗余。
* **主码**：从候选码中选出的唯一标识符。
* **外码**：指向其他关系的主码，建立表与表之间的关联。

**总结表**

| **名称** | **定义** | **特点** | **示例** |
| --- | --- | --- | --- |
| 超码 (Super Key) | 唯一标识元组的属性集合，可以包含冗余属性。 | 包含冗余，不一定最小。 | {StudentID,Name} |
| 候选码 (Candidate Key) | 最小的超码，唯一标识元组，没有冗余属性。 | 唯一性和最小性。 | {StudentID},{Email} |
| 主码 (Primary Key) | 从候选码中选定的唯一标识元组的属性集合。 | 唯一且非空，关系的核心标识符。 | {StudentID} |
| 外码 (Foreign Key) | 一个关系中引用另一个关系主码的属性集合。 | 指向主码，用于建立表间联系。 | 选课表中的StudentID引用学生表 |

**2.3 完整性约束**

关系数据模型的完整性约束用于保证数据的准确性和一致性，是数据库系统中数据约束的重要部分。主要包括以下三类：

**1. 实体完整性约束 (Entity Integrity Constraint)**

1. **定义**：
   * 确保关系中每个元组都有唯一标识。
   * **关系的主码（Primary Key）上的属性值必须唯一且不能为 NULL。**
2. **规则**：
   * 主码值必须唯一，不能重复。
   * 主码值不能为空。
3. **作用**：
   * 保证每个元组在关系中的唯一性。
   * 防止丢失对元组的引用。

**示例：**

* 表：学生(Student) Student(SID,Name,Age)
  + SID是主码：
    - SID的值必须唯一，例如 {101,102,103}
    - SID的值不能为 NULL。

**2. 参照完整性约束 (Referential Integrity Constraint)**

1. **定义：**
   * **确保外码（Foreign Key）引用的值必须存在于被引用关系的主码中。**
   * **外码可以为空，但如果有值，必须与被引用关系中的主码匹配。**
2. **规则**：
   * 外码值必须存在于被引用关系的主码值中。
   * 删除或更新被引用主码时，需要保证外码的约束：
3. **作用**：
   * 维护关系之间的引用完整性。
   * 防止引用不存在的数据。

**示例：**

* 表1：学生(Student) Student(SID,Name)
* 表2：选课(Course) Course(CID,SID,Subject)
  + SID是选课表的外码，引用学生表的SID。
  + 如果学生表中没有 SID=104，那么选课表中SID不能取值为 104。

**3. 用户定义完整性约束 (User-defined Integrity Constraint)**

1. **定义**：
   * 根据具体应用场景，由用户自定义的数据约束条件。
   * 通常是某些属性值必须满足的业务规则。
2. **规则**：
   * 使用 CHECK 约束定义。
   * 定义约束条件，确保数据符合业务逻辑。
3. **作用**：
   * 约束属性或元组的取值范围，符合特定语义需求。
   * 增强数据的一致性和准确性。

**示例：**

* 表：学生(Student) Student(SID,Name,Age)
  + 约束1：Age≥18。
  + 约束2：Name≠NULL。

**完整性约束小结**

| **类型** | **描述** | **作用** |
| --- | --- | --- |
| **实体完整性** | 主码的值必须唯一且非空。 | 确保每个元组有唯一标识，防止数据丢失。 |
| **参照完整性** | 外码必须引用一个存在的主码，或外码值为空。 | 维护关系间的引用一致性，防止引用失效或无效数据。 |
| **用户定义完整性** | 自定义的业务规则（如 CHECK 条件）。 | 保证数据符合特定的业务语义要求。 |

**2.4 关系代数的运算**

关系代数是一种操作关系数据库的数学方法，提供了查询和操作数据的形式化语言。其运算分为基本运算和扩展运算，常见运算包括以下内容：

**1. 基本运算**

**(1) 选择（Selection）**

* **符号**：σ条件(R)
* **功能**： 从关系 R 中选出满足指定条件的元组（行）。
* **特点**： 选择操作的结果仍是一个关系。
* **示例**： σAge>20(Student)表示选出学生表中年龄大于 20 的元组。

**(2) 投影（Projection）**

* **符号**：π属性列表(R)\pi\_{属性列表}(R)π属性列表​(R)
* **功能**： 从关系 RRR 中选出指定的属性列，去掉重复的元组。
* **特点**： 结果只包含指定的列，其他列被忽略。
* **示例**： πName,Age(Student)表示选出学生表中的姓名和年龄列。

**(3) 并集（Union）**

* **符号**：R1∪R2
* **功能**： 返回两个关系 R1和 R2的并集，包含在 R1或 R2中的所有元组。
* **要求**：
  + **R1和 R2必须具有相同的属性。**
  + 结果中去除重复元组。
* **示例**： Student1∪Student2

**(4) 差集（Set Difference）**

* **符号**：R1−R2
* **功能**： 返回在 R1中但不在 R2中的元组。
* **要求**：
  + **R1和 R2必须具有相同的属性。**
* **示例**： Student1−Student2

**(5) 笛卡尔积（Cartesian Product）**

* **符号**：R1×R2
* **功能**： 返回 R1和 R2中所有元组的笛卡尔积。
  + 如果 R1有 m个元组，R2有 n个元组，则结果有m×n个元组。
* **特点**： 通常与选择操作结合使用，以表示关系之间的连接。
* **示例**： Student×Course

**(6) 交集（Intersection）**

* **符号**：R1∩R2
* **功能**： 返回 R1和 R2中都存在的元组。
* **要求**：
  + R1和 R2必须具有相同的属性。
* **示例**： Student1∩Student2

**2. 扩展运算**

**(1) 自然连接（Natural Join）**

* **符号**：R1⋈R2
* **功能**： 通过两个关系中具有相同属性的列，将元组进行组合。
* **步骤**：
  1. 选择具有相同属性值的元组。
  2. 合并元组，去掉重复的列。
* **示例**： Student⋈Course

**(2) θ-连接（Theta Join）**

* **符号**：R1⋈θR2
* **功能**： 通过指定的条件 θ\thetaθ（如 =,<,>,≠=, <, >, \neq=,<,>,=）连接两个关系。
* **示例**： Student⋈Age>20Course

**(3) 左外连接（Left Outer Join）**

* **符号**：R1⋈LR2
* **功能**： **返回自然连接结果和 R1​ 中未匹配到 R2​ 的元组（填充空值）。**
* **示例**： Student⋈LCourseStudent

**(4) 除法运算（Division）**

* **符号**：R1÷R2
* **功能**： 返回在 R1中所有属性的值，这些值与 R2的每个元组都相关。
* **示例**： 如果 R1是 (StudentID,Course)，R2是Course，则结果是**选出选修了 R2中所有课程的学生。**

**3. 关系代数的常用组合**

通过将基本运算和扩展运算组合，可以实现更复杂的查询，例如：

* **查询选修特定课程的学生**： πName(σCourse=′Database′(Student⋈Course))

**2.5 元组关系演算 (Tuple Relational Calculus, TRC) 与域关系演算 (Domain Relational Calculus, DRC)**

**1. 元组关系演算 (TRC)**

**定义**

* **元组关系演算是一种基于元组变量的查询语言，查询结果由满足给定条件的元组集合组成。**
* **查询以表达式形式表示： {t∣P(t)}**
  + t：元组变量，表示关系中的一个元组。
  + P(t)：谓词，表示条件表达式，用于描述 t 应满足的条件。

**特点**

1. **基于元组变量**：
   * 查询涉及整个元组，而不是单独的属性。
2. **结果为元组集合**：
   * 查询结果是由满足条件的元组组成的子集。
3. **使用逻辑运算**：
   * 支持逻辑操作符（如 AND、OR、NOT）以及量词（如 ∀和∃）。

**示例**

假设有关系 Student(SID,Name,Age)，查询年龄大于 20 的学生：

{t∣t∈Student∧t.Age>20}

**2. 域关系演算 (DRC)**

**定义**

* **域关系演算是一种基于属性值的查询语言，查询结果由满足给定条件的属性值集合组成。**
* 查询以表达式形式表示： {(x1,x2,...,xn)∣P(x1,x2,...,xn)}
  + x1,x2,...,xn ​：域变量，分别对应关系中的属性。
  + P(x1,x2,...,xn)：谓词，表示条件表达式，用于描述属性值应满足的条件。

**特点**

1. **基于域变量**：
   * 查询直接涉及属性的值，而不是整个元组。
2. **结果为属性值集合**：
   * 查询结果由满足条件的属性值组合组成。
3. **使用逻辑运算**：
   * 支持逻辑操作符和量词，与 TRC 类似。

**示例**

假设有关系 Student(SID,Name,Age)，查询年龄大于 20 的学生姓名：

{x∣∃y,z (x,y,z)∈Student∧z>20}

* x：对应学生的姓名。
* y,z：分别对应学号和年龄。

**TRC 与 DRC 的对比**

| **特性** | **元组关系演算 (TRC)** | **域关系演算 (DRC)** |
| --- | --- | --- |
| **基础单位** | 元组变量，表示关系中的完整元组 | 域变量，表示关系中的属性值 |
| **查询结果** | 元组的集合 | 属性值的集合 |
| **表示方式** | {t∣P(t)} | {(x1,x2,...,xn)∣P(x1,x2,...,xn)} |
| **适用场景** | 查询涉及多个属性或整个元组时 | 查询某些具体属性时更简单 |
| **复杂度** | 更直观，适合与元组相关的查询 | 更灵活，但表达复杂查询时可能较繁琐 |

**两者的关系**

* TRC 和 DRC 都是非过程性查询语言，关注结果而非查询过程。
* 它们等价于关系代数，能表达关系代数能实现的所有查询。

SQL（Structured Query Language，结构化查询语言）是关系数据库中用于定义、查询、操作和管理数据的标准语言。以下是对 SQL 各类语句的详细介绍，包括分类、语法及示例。

**1. SQL 的分类**

SQL 的功能主要分为以下几类：

| **类型** | **功能** |
| --- | --- |
| **DDL (Data Definition Language)** | 定义数据库对象（如表、视图、索引等）的结构和属性。 |
| **DML (Data Manipulation Language)** | 操作表中的数据，包括插入、删除和更新数据。 |
| **DQL (Data Query Language)** | 查询表中的数据，包括 SELECT 语句。 |
| **TCL (Transaction Control Language)** | 控制事务，包括提交和回滚。 |
| **DCL (Data Control Language)** | 管理用户权限，包括授权和回收权限。 |

**2. 数据定义语言 (DDL)**

**1) CREATE 语句**

用于创建数据库对象（如数据库、表、视图等）。

**语法**：

CREATE TABLE 表名 (

列名1 数据类型 [约束],

列名2 数据类型 [约束],

...

);

**示例**： 创建一个学生表：

CREATE TABLE Student (

SID INT PRIMARY KEY,

Name VARCHAR(50) NOT NULL,

Age INT CHECK (Age >= 18),

Major VARCHAR(50)

);

**2) ALTER 语句**

用于修改已有数据库对象的结构。

**语法**：

ALTER TABLE 表名

ADD 列名 数据类型 [约束];

ALTER TABLE 表名

DROP COLUMN 列名;

ALTER TABLE 表名

MODIFY COLUMN 列名 数据类型;

**示例**： 向学生表中添加列：

ALTER TABLE Student

ADD Email VARCHAR(100);

**3) DROP 语句**

用于删除数据库对象。

**语法**：

DROP TABLE 表名;

DROP DATABASE 数据库名;

**示例**： 删除学生表：

DROP TABLE Student;

**3. 数据操作语言 (DML)**

**1) INSERT 语句**

用于向表中插入数据。

**语法**：

INSERT INTO 表名 (列1, 列2, ...) VALUES (值1, 值2, ...);

**示例**： 向学生表插入数据：

INSERT INTO Student (SID, Name, Age, Major)

VALUES (101, '张三', 20, '计算机');

**2) UPDATE 语句**

用于修改表中的数据。

**语法**：

UPDATE 表名

SET 列1 = 值1, 列2 = 值2, ...

WHERE 条件;

**示例**： 修改学生的专业：

UPDATE Student

SET Major = '数据科学'

WHERE SID = 101;

**3) DELETE 语句**

用于删除表中的数据。

**语法**：

DELETE FROM 表名

WHERE 条件;

**示例**： 删除年龄小于 20 的学生：

DELETE FROM Student

WHERE Age < 20;

**4. 数据查询语言 (DQL)**

**SQL 查询的类型**

SQL 查询分为单表查询、连接查询、嵌套子查询和集合查询，这些查询方式用于满足不同的业务需求。以下是对这些查询的详细介绍：

**4.1 单表查询**

单表查询是针对单个表进行数据查询的操作。

**基本语法**

SELECT 列名1, 列名2, ...

FROM 表名

[WHERE 条件]

[GROUP BY 列名]

[HAVING 条件]

[ORDER BY 列名 [ASC|DESC]];

**示例**

假设有一个表 Student，结构如下：

Student(SID, Name, Age, Major)

1. 查询所有学生的信息：

SELECT \* FROM Student;

1. 查询年龄大于 20 的学生：

SELECT Name, Age FROM Student

WHERE Age > 20;

1. **按专业分组查询每个专业的学生人数：**

**SELECT Major, COUNT(\*) AS StudentCount**

**FROM Student**

**GROUP BY Major;**

1. **查询年龄最大的学生：**

**SELECT Name, Age**

**FROM Student**

**WHERE Age = (SELECT MAX(Age) FROM Student);**

**2. 连接查询**

连接查询用于从多个表中获取相关数据。常见的连接方式包括内连接、左外连接、右外连接和全外连接。

**内连接（INNER JOIN）**

**语法**：

SELECT 表1.列名, 表2.列名, ...

FROM 表1

INNER JOIN 表2 ON 表1.列名 = 表2.列名

[WHERE 条件];

**示例**： 查询学生及其选修的课程信息：

SELECT S.Name, C.CourseName

FROM Student S

INNER JOIN Course C ON S.SID = C.SID;

**左外连接（LEFT JOIN）**

返回左表中所有记录以及右表中匹配的记录，不匹配的用 NULL 填充。

**示例**： 查询所有学生及其选修的课程信息（包括未选课的学生）：

SELECT S.Name, C.CourseName

FROM Student S

LEFT JOIN Course C ON S.SID = C.SID;

**右外连接（RIGHT JOIN）**

返回右表中所有记录以及左表中匹配的记录。

**示例**： 查询所有课程及其对应的学生信息：

SELECT S.Name, C.CourseName

FROM Student S

RIGHT JOIN Course C ON S.SID = C.SID;

**全外连接（FULL JOIN）**

返回左右两表的所有记录，不匹配的部分用 NULL 填充。

**示例**：

SELECT S.Name, C.CourseName

FROM Student S

FULL JOIN Course C ON S.SID = C.SID;

**3. 嵌套子查询**

嵌套子查询是在一个 SQL 查询中嵌套另一个查询，用于实现复杂查询需求。

**基本语法**

SELECT 列名

FROM 表名

WHERE 列名 运算符 (SELECT 子查询);

**分类**

1. **标量子查询**： 返回单个值，用于比较。
   * 示例：查询选修课程最多的学生：

SELECT Name

FROM Student

WHERE SID = (SELECT SID

FROM Course

GROUP BY SID

ORDER BY COUNT(\*) DESC

LIMIT 1);

1. **多行子查询**： 返回多行结果，使用 IN、ANY、ALL 等操作符。
   * 示例：查询选修了“数学”或“英语”课程的学生：

SELECT Name

FROM Student

WHERE SID IN (SELECT SID

FROM Course

WHERE CourseName IN ('数学', '英语'));

1. **相关子查询**： 子查询依赖于外部查询。
   * 示例：查询选修的课程数大于 2 的学生：

SELECT Name

FROM Student S

WHERE (SELECT COUNT(\*)

FROM Course C

WHERE C.SID = S.SID) > 2;

**4. 集合查询**

集合查询用于合并两个或多个查询结果，常见操作包括 UNION、INTERSECT 和 EXCEPT。

**1) UNION（并集）**

将两个查询结果合并，并去除重复记录。

**语法**：

SELECT 列名 FROM 表1

UNION

SELECT 列名 FROM 表2;

**示例**： 查询所有选修“数学”或“英语”课程的学生：

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '数学')

UNION

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '英语');

**2) UNION ALL（并集，保留重复）**

与 UNION 类似，但保留重复记录。

**示例**：

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '数学')

UNION ALL

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '英语');

**3) INTERSECT（交集）**

返回两个查询结果中共有的记录。

**语法**：

SELECT 列名 FROM 表1

INTERSECT

SELECT 列名 FROM 表2;

**示例**： 查询同时选修“数学”和“英语”课程的学生：

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '数学')

INTERSECT

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '英语');

**4) EXCEPT（差集）**

返回第一个查询结果中有，但第二个查询结果中没有的记录。

**语法**：

SELECT 列名 FROM 表1

EXCEPT

SELECT 列名 FROM 表2;

**示例**： 查询选修了“数学”但未选修“英语”课程的学生：

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '数学')

EXCEPT

SELECT Name FROM Student S

WHERE SID IN (SELECT SID FROM Course WHERE CourseName = '英语');