数据库模式设计如果不好会导致的问题：

1．冗余

2．导致数据一致性出现问题

3．插入异常

4．更新异常

5．删除异常

函数依赖

函数依赖是指一个或多个属性的取值可以确定另一个属性的取值。具体地说，如果一个关系模式R中属性集合X的取值能唯一地确定属性集合Y的取值，那么我们称属性集合Y对于属性集合X具有函数依赖。这被表示为X → Y，其中X称为函数依赖的左部，Y称为右部。

例如，考虑一个包含学生信息的关系模式R，其中包含属性集合{学生ID，姓名，年龄，所在班级，班级导师}。假设我们观察到一个学生ID只对应一个姓名和年龄，那么我们可以说“学生ID函数依赖于姓名和年龄”，表示为{学生ID} → {姓名，年龄}。

函数依赖在数据库设计中非常重要，因为它们可以帮助我们识别重复数据和设计表结构。如果我们正确地理解和应用函数依赖，就可以减少数据冗余，提高数据完整性和一致性。

数据库的键

在关系型数据库中，键（key）是指一种特殊的属性或属性集合，用于唯一标识一个关系中的元组。键可以帮助我们在数据库中快速准确地定位、访问和修改数据。

关系数据库中的键可以分为以下三种类型：

1. 主键（Primary Key）：主键是一个关系中的一个或多个属性，用于唯一标识关系中的每个元组。主键具有以下特点：
    - 主键的值在关系中必须是唯一的。
    - 主键的值不能为空值（NULL）。
    - 一个关系只能有一个主键。
2. 外键（Foreign Key）：外键是一个关系中的属性，它引用了另一个关系的主键，用于建立两个关系之间的关系。外键具有以下特点：
    - 外键的值必须在另一个关系中存在，或者为空值（NULL）。
    - 一个关系可以有多个外键。
3. 候选键（Candidate Key）：候选键是一个关系中的一个或多个属性，它可以唯一标识关系中的每个元组，但不是主键。一个关系可以有多个候选键。**而且不同候选键的元素个数可能不同！**

4. 超键（super key）是指可以唯一标识一个关系中的每个元组的属性集合。换句话说，一个超键的值集合可以唯一确定关系中每个元组的值。超键可以包含关系中的所有属性，也可以只包含部分属性。超键不一定是最小的，也就是说，可能存在多个超键可以唯一标识关系中的每个元组。

键在数据库中非常重要，因为它们可以帮助我们维护数据的完整性和一致性。通过使用键，我们可以确保每个元组都具有唯一的标识符，从而避免数据冗余和不一致性。

寻找数据库的键

函数依赖集合的闭包

定义：在关系数据库中，函数集合的闭包是指一个函数集合中所有可能的函数依赖组合所得到的函数依赖集合。换句话说，函数集合的闭包包含了原函数集合中所有可能的函数依赖和派生函数依赖。通过寻找函数集合的闭包，我们可以了解关系模式中所有可能的函数依赖，从而更好地设计和优化关系模式。

以下是寻找函数集合的闭包的方法：

1. 初始闭包：将原函数集合中的所有函数依赖加入到闭包中。
2. 递归添加：对于闭包中的每个函数依赖，找到其右部属性集合所能推导出的所有函数依赖，并将其添加到闭包中。
3. 直到无法添加：重复步骤2，直到闭包中没有新的函数依赖可以添加为止。

函数依赖的规则包括以下几条：

1. 自反性规则：如果Y包含于X，则X → Y。
2. 扩展性规则：如果X → Y，那么XZ → YZ，其中Z是关系R中除X、Y之外的任意属性集合。
3. 传递性规则：如果X → Y，Y → Z，那么X → Z。

这些规则可以用来推导出函数依赖的闭包。通过这些规则，我们可以确定关系模式中的主键、候选键和冗余属性，从而优化关系模式的设计和性能。

通过函数依赖的闭包求解键的办法（暴力法）：

1. 确定关系中的属性集合：首先，确定关系中的所有属性集合，包括主属性和非主属性。
2. 确定函数依赖集合：根据实际情况，确定关系中的所有可能函数依赖。一个函数依赖是指一个属性集合能够唯一确定另一个属性集合。例如，如果属性集合A能够唯一确定属性集合B，则称为A → B。
3. 计算函数依赖的闭包：通过使用函数依赖的自反性、扩展性和传递性规则，计算关系中所有可能的函数依赖。这将给出一个包含原函数依赖和派生函数依赖的函数依赖集合，即函数依赖的闭包。
4. 确定候选键：对于关系中的每个属性集合，检查是否可以唯一确定每个元组。如果可以，则该属性集合是一个候选键。如果有多个候选键，则需要确定一个主键。
5. 检查主键是否在函数依赖闭包中：检查所选的主键是否在函数依赖闭包中。如果主键不在闭包中，则需要重新选择一个候选键作为主键。

启发式算法：

1. 不在任何函数依赖右边出现的属性**必然是**键的一部分。
2. 不在函数依赖左边出现的属性**一定不是**键的一部分

关系模式的分解

1. 无损分解（Lossless Decomposition）

   无损分解是指将一个大的关系模式拆分成多个小的关系模式，同时保持它们之间的函数依赖和信息完整性不变。这种分解方式的优点是可以提高数据库的性能和可维护性，同时保持数据的完整性和一致性。无损分解的目标是保证原始关系模式中的所有函数依赖都能在新的小的关系模式中得到满足，因此无损分解的结果仍然可以保持关系模式的完整性和一致性。

2. 有损分解（Lossy Decomposition）

   有损分解是指将一个大的关系模式拆分成多个小的关系模式，但不能保持它们之间的函数依赖和信息完整性不变。这种分解方式的优点是可以提高数据库的性能和可维护性，同时减少数据的冗余和复杂性。有损分解的目标是在满足一定的性能需求的前提下，尽可能地减少数据冗余和复杂性。有损分解的结果可能会导致数据的不一致性和丢失一些信息，因此需要谨慎选择分解方式和对数据进行适当的处理。

   需要注意的是，无损分解和有损分解都有其适用的场景和限制，具体的选择需要根据实际需求和性能要求进行综合考虑。在分解关系R时，我们希望分解能够：

   - 最小化冗余
   - 避免信息丢失
   - 保留依赖关系（即约束条件）
   - 确保查询性能良好

如何将一个模式分解为多个模式

> 当我们要将一个关系模式拆分为符合第一范式的多个关系模式时，需要遵循以下规则：

1. 确定主键：首先需要确定关系模式的主键，以确保每个元组都能唯一标识。主键可以由单个属性或多个属性组成。
2. 消除重复组：如果关系模式中存在重复组，即有两个或多个元组在所有属性上的取值都相同，就需要将这些元组合并为一个元组。
3. 拆分多值依赖：如果关系模式中存在多值依赖，即一个属性依赖于主键的某个子集而不是整个主键，就需要将这些属性拆分成新的关系模式，并与原来的关系模式建立外键关系。
4. 拆分部分依赖：如果关系模式中存在部分依赖，即一个非主属性依赖于主键的某个子集而不是整个主键，就需要将这些非主属性与主键组成一个新的关系模式，并在新的关系模式中建立外键关系。

example

例如，假设我们有一个包含以下属性的关系模式：

学生（学号，姓名，年龄，课程，成绩）

在这个关系模式中，学号是主键，课程和成绩是非主属性。由于一个学生可以选修多门课程，课程和成绩之间存在多值依赖关系。为了消除多值依赖，我们可以将课程和成绩拆分成新的关系模式，并与原来的关系模式建立外键关系。这样就可以得到以下两个关系模式：

学生（学号，姓名，年龄）

选课（学号，课程，成绩）

在新的关系模式中，每个关系模式都符合第一范式的要求，因为每个属性都是单值属性，不存在重复组和多值依赖的问题。

关系模式的范式

在关系型数据库中，范式（Normalization）是一种通过分解关系模式来减少数据冗余、提高数据一致性和数据可维护性的技术。范式分为不同的级别，从第一范式（1NF）到第五范式（5NF），每个级别都有其特定的规则和限制。以下是各个范式的详细说明：

1. 第一范式（1NF）

   第一范式要求一个关系模式中的每个属性都是原子的，即不能再分解成更小的数据项。例如，如果一个学生的电话号码是“123-456-7890”，则应将其分解成三个不同的属性：国家代码、区号和电话号码。

2. 第二范式（2NF）

   第二范式要求一个关系模式中的每个非主属性都完全依赖于候选码，而不是部分依赖于候选码。例如，如果一个关系模式中包含“订单号”、“商品编号”和“商品数量”这三个属性，其中“商品数量”只依赖于“订单号”，而不依赖于“商品编号”，则应将其拆分为两个关系模式，以确保每个非主属性完全依赖于主键。

3. 第三范式（3NF）

   第三范式要求一个关系模式中的每个非主属性都不依赖于其他非主属性，即不存在传递依赖关系。例如，如果一个关系模式中包含“订单号”、“商品编号”、“商品名称”和“供应商名称”这四个属性，其中“供应商名称”依赖于“商品编号”，而“商品名称”和“供应商名称”存在函数依赖关系，则应将其拆分为两个关系模式，以确保不存在传递依赖关系。

   一个模式违反第三范式:是A->B当且仅当A不是超键且B不是主属性

4. 巴斯-科德范式（BCNF）

   巴斯-科德范式是第三范式的扩展，要求一个关系模式中的每个属性都完全依赖于主键，而不是仅依赖于部分主键。这种范式适用于复杂的关系模式，其中存在多个主键。

对于任意一个函数依赖A->B，A必须是候选码。

5. 第四范式（4NF）

   第四范式要求一个关系模式中的每个多值依赖都被分解成独立的关系模式。例如，如果一个关系模式中包含“学生编号”、“课程编号”和“成绩”这三个属性，其中每个学生可能有多个课程的成绩，而每个课程可能由多个学生选修，则应将其拆分为三个关系模式，以确保每个多值依赖都被分解成独立的关系模式。

6. 第五范式（5NF）

   第五范式是指一个关系模式中的每个依赖关系都是基于超键而不是基于主键的。超键是指能够唯一标识一个关系模式中所有元组的一个或多个属性组合。第五范式适用于多维关系数据库和数据仓库，其中存在复杂的多重关系和多个维度。

   需要注意的是，虽然范式可以提高数据库的性能和可维护性，但过度范式化也可能会导致查询性能下降和数据更新的困难。因此，在实际应用中，需要根据实际需求和性能要求来选择适当的范式级别，以实现数据的高效管理和使用。

与范式结合的分解

已知关系模式和函数依赖集合保持FD和无损连接3NF模式分解的算法如下：

1. 求**极小函数依赖集**Sm和关系模式的所有键(keys)。
2. 在Smin中按函数依赖左部相同原则进行分组，每个组中的所有属性形成分解后的子关系模式R1, R2, ..., Rn。
3. 如果某个关系模式Ri的所有属性被另一个关系模式Rj所包含，删除关系模式Ri。
4. 判断是否有某个key出现在其中的一个关系模式中，如果出现则结束；如果没有出现，将任意一个key也作为子关系模式R加入到分解后的模式中。

最小函数依赖集

如果函数依赖集合F满足如下条件，则称F为一个极小函数依赖集，也称为最小依赖集或最小覆盖：

1. F中的任意函数依赖的右部仅含有一个属性。
2. F中不存在这样的函数依赖X→A使得F与F-X→A等价。
3. F中不存在这样的函数依赖X→A，X有真子集Z使得(F-X→A)∪(Z→A)与F等价。

如果G+ = F+，就称F与G等价。G = F的充分必要条件是F ⊆ G+ 且 G+ ⊆ F。

我们在求最小函数依赖集合时，事实上是将所有右边仅有一个属性的函数依赖的冗余消除掉，包括但不限于具有将具有传递性的几个函数依赖消除掉。最本质仍然是**消除冗余。**

> 一个性质：任意一个数据库的关系模式都可以保持函数依赖和无损链接到分解到第三范式！

第四范式

第四范式（4NF）是关系数据库中的一种规范化形式，它要求关系模式中的每个非平凡多值依赖都要被分解，以达到消除冗余和保证数据一致性的目的。

具体来说，第四范式的要求如下：

1. 关系模式中的每个非主属性必须与主键存在非平凡多值依赖，即非主属性不能完全依赖于主键中的任意一个属性。
2. 关系模式中的每个非主属性必须是一个数据项的集合，即不能包含重复值。

满足第四范式的关系模式可以消除非平凡多值依赖，避免数据冗余和更新异常，并且可以保证数据一致性。但是，第四范式的实现可能会导致关系模式的分解过于复杂，查询性能下降等问题，因此在实际应用中需要根据具体情况综合考虑。

多值依赖

> 多值依赖（Multivalued Dependency，MVD）是指一个关系模式中的某些属性集合对另一个属性集合存在依赖关系，但是这种依赖关系并不是函数依赖。在一个满足MVD的关系模式中，一个属性集合的取值可以对应多个其他属性集合的取值。
>
> 例如，假设我们有一个包含以下属性的关系模式：
>
> 学生（学号，姓名，选修课程，成绩）
>
> 在这个关系模式中，选修课程和成绩之间存在MVD，因为每个学生可以选修多门课程，每门课程的成绩都可能不同。因此，选修课程的取值可以对应多个成绩的取值，而不是像函数依赖那样只对应一个成绩的取值。
>
> MVD是关系数据库理论中一个重要的概念，可以帮助我们理解数据库设计和优化的一些问题。在实际应用中，如果一个关系模式中存在MVD，我们可以考虑将其拆分为符合第三范式或BCNF的多个关系模式，以提高数据库的性能和可维护性。

下面将对于多值依赖举一个例子：

假设有一个关系模式R，包含属性A、B和C，其中属性A为主键。现在，我们发现存在这样的依赖关系：对于任意的两个元组r1和r2，如果它们在A属性上的值相等，那么在B属性和C属性上的值都是多值依赖的。这就是一个MVD，可以表示为A →→ B,C。

举个例子，假设关系模式R中有如下数据：

A	B	C
a1	b1, c1	b2, c2
a1	b2, c2	b1, c1
a2	b3, c3	b4, c4

A	B	C
a2	b4, c4	b3, c3

我们可以看到，在A属性上有两个不同的值a1和a2，对于每个值，B和C属性上的值都是多值依赖的，即b1和b2、c1和c2，以及b3和b4、c3和c4。这就是一个MVD的例子。

需要注意的是，MVD不同于函数依赖，它是对于关系模式中的一组属性来说的，并且MVD的右侧可以包含多个属性，表示这些属性的组合是多值依赖的。MVD的出现可能导致数据冗余和更新异常，需要进行适当的规范化处理。

函数依赖和多值依赖的关系

> 结论：每个函数依赖都可以看作是一个多值依赖。具体来说，如果一个属性集合X决定了另一个属性集合Y，那么对于所有在属性集合X上取值相同的元组，它们在属性集合Y上的取值也必须相同。换句话说，如果我们交换在属性集合Y上的两个值，那么这些元组仍然是合法的。因此，可以将X和Y看作是一个MVD。
>
> 具体来说，它指出，如果一个FD X -> Y成立，那么对于在属性集合X上取值相同的任意两个元组，它们在属性集合Y上的取值必须相同。因此，交换在属性集合Y上的两个值不会改变这些元组的合法性。因此，我们可以将FD X -> Y视为MVD X - Y。
>
> 补集原理：如果一个属性集合X决定了另一个属性集合Y，那么对于属性集合Y的补集（即所有不在Y中的属性集合Z），X和Z之间必须存在MVD。这个原理是很有用的，因为它可以帮助我们从一个给定的函数依赖集合中推导出所有的MVD。具体来说，对于每个函数依赖X -> Y，我们可以通过补集原理推导出所有的MVD X - Z，其中Z是Y的补集。

important：对于多值依赖，如果AB->->C，那么我们无法推出A->->C且B->->C！对于A->->BC，也无法推出A->->B和A->->C！

我们在上文说过一件事情：任意一个函数依赖都可以看成是一个多值依赖，那么如果基于此，R(A,B,C)中有函数依赖：A->BC，R事实上应该是4NF，但是如果从任意一个函数依赖都是多值依赖的角度去思考：A->BC有A->B，所以A->->B且A->->C，那么他就不是第四范式，所以我们应该对此加以约束：

如果关系模式R满足以下条件：当X->->Y是一个非平凡的多值依赖（MVD）时，X是一个超键，那么R就符合第四范式（4NF）。其中，非平凡的MVD是指Y不是X的子集，且X和Y并不包含关系模式中的所有属性。

需要注意的是，这里的"超键"仍然基于函数依赖的定义。此外，该定义表示只有满足特定条件的非平凡MVD才需要被考虑，而不是所有MVD。

那么基于此，我们看到：由于A是一个超键，且A->->B和A->->C是非平凡的多值依赖，所以这个模式R是4NF。

与之类似的是R2(A.B,C),存在多值依赖：A->->B和A->->C；与上面的区别在于