1. 关系数据库让用户自行定义每一张表的 Primary Key（主键），用于唯一识别表中的每一行数据。例如学生表 student(sno, sname, birthday, gender)的主键可定义为为 sno，宿舍表 rooms(dorm_no, room_no, size, floor)的主键可定义为(dorm_no, room_no)（由宿舍号和房间号组成的复合主键）。在文档数据库中，用户无需定义主键，每一个文档都可以由系统自动产生的省缺 ID 进行识别。换句话说，文档数据库的 ID 属性起到了 Primary Key 的作用。请思考：关系数据库的 Primary Key 机制和文档数据库的 ID 机制有什么不同，各自的优缺点是什么？

关系数据库的 Primary Key 机制与文档数据库的 ID 机制的不同，以及各自的优缺点：

不同：

1、关系数据库的主键是由用户定义的，用于保证表中记录的唯一性。用户可以选择一个或多个列作为主键，而这些列的值必须是唯一的，且不可为空。

2、文档数据库中的 ID 通常是由系统自动生成的，保证每个文档的唯一性。这个 ID 是文档创建时自动分配的，用户无需事先定义。

优缺点：

关系数据库主键：

优点：用户定义的主键允许对数据模型有更严格的控制，可以根据业务规则选择合适的主键，这有助于确保数据的完整性和业务逻辑的一致性。

缺点：用户必须预先知道哪些字段可以唯一标识记录，这在某些情况下可能不是很明显。同时，主键的选择可能会对数据库性能产生重大影响，不当的选择可能导致性能问题。

文档数据库 ID：

优点：系统自动生成的 ID 减少了用户的工作量，使得用户不必担心唯一性和主键的选择问题。这也使得文档数据库可以更灵活地处理不同结构的数据。

缺点：自动生成的 ID 可能缺乏业务意义，不如用户定义的主键直观，且在需要维护数据之间的关系时可能不如手动定义的主键那样灵活。

2. 关系数据库要求用户在使用一张表之前用 DDL 对表进行事先定义，并且给出表需满足的各种约束（比如主键、外键等）。文档数据库则不同，它通常不要求用户对文档集的结构做事先定义，甚至允许用户往文档集中插入任意结构的文档。请思考：关系数据库和文档数据库为什么使用了两种不同的功能设计？背后的原因是什么？

关系数据库通过 DDL 定义表的结构和约束，因为它们通常用于更复杂的查询和事务处理，其中数据的完整性、一致性和关系是非常重要的。这种设计支持强类型和结构化查询，有利于维护大型、复杂的数据系统。

文档数据库不要求预定义结构，因为它们旨在提供更高的灵活性和扩展性，尤其是在处理半结构化或非结构化数据时。这种设计允许快速开发和迭代，适用于快速变化的数据模型和需求。

背后的原因是关系数据库和文档数据库服务于不同的使用案例。关系数据库强调 ACID 属性，适合需要严格数据完整性和预定义模式的应用。而文档数据库更适合于需要处理大量变化数据、不需要事先定义固定模式的应用场景。

3. 如果我们在属性 A 上定义了 Primary Key 或 Unique 这样的约束，那么数据库将要求表中的任何一个元组在 A 上的取值都是唯一的。也就是说，当我们往表中重复插入 A 值时，数

据库将立即感知到，并禁止我们这样做。请问：数据库是如何感知我们往表中重复插入 A 值的?
1、数据库通过在属性 A 上建立索引来感知重复值。当定义了主键或唯一约束时，数据库会自动为这些字段创建一个唯一索引。
2、当尝试插入新的记录时，数据库会在这个唯一索引中查找是否存在相同的 A 值。
3、如果索引查找表明已经存在一个相同的 A 值，数据库将拒绝新的插入操作并返回一个错误，通知用户违反了唯一性约束。
4、这种检查几乎是即时的，因为索引结构（如 B 树）是为快速检索和维护唯一性而优化的。

假设我们有三个关系（同课程中使用的例子一样），其模式分别为
Student(s_no, s_name, birthday, gender)
Course(c_no, c_name, credit)
SC(s_no, c_no, grade)请写出以下信息需求的 SQL 查询:
在数学课上成绩超过 90 分的男生姓名;

1. **在数学课上成绩超过90分的男生姓名**:

```sql
SELECT s.s_name
FROM Student s, SC sc, Course c
WHERE s.s_no = sc.s_no
  AND sc.c_no = c.c_no
  AND c.c_name = 'Math'
  AND sc.grade > 90
  AND s.gender = 'male';
```

数学课成绩超过历史课成绩的女生姓名;

1. **数学课成绩超过历史课成绩的女生姓名**:

```sql
SELECT DISTINCT s.s_name
FROM Student s, SC sc_math, SC sc_history, Course c_math, Course c_history
WHERE s.s_no = sc_math.s_no
  AND sc_math.c_no = c_math.c_no
  AND c_math.c_name = 'Math'
  AND s.s_no = sc_history.s_no
  AND sc_history.c_no = c_history.c_no
  AND c_history.c_name = 'History'
  AND sc_math.grade > sc_history.grade
  AND s.gender = 'female';
```

平均成绩超过 90 分的女生姓名。

1. **平均成绩超过90分的女生姓名**:

```sql
SELECT s.s_name
FROM Student s, SC sc
WHERE s.s_no = sc.s_no
  AND s.gender = 'female'
GROUP BY s.s_name
HAVING AVG(sc.grade) > 90;
```

假设我们有三个关系（同课程中使用的例子一样），其模式分别为
Student(s_no, s_name, birthday, gender)
Course(c_no, c_name, credit)
SC(s_no, c_no, grade)请写出满足以下访问需求的 SQL 语句:
删除所有平均成绩小于 70 分的课程;
找到在所有课程上的成绩都超过课程平均成绩的学生。

1. **删除所有平均成绩小于70分的课程**:

```sql
DELETE FROM Course
WHERE c_no IN (
  SELECT c_no
  FROM SC
  GROUP BY c_no
  HAVING AVG(grade) < 70
);
```

在这个查询中，子查询首先选择所有平均成绩小于70分的课程编号（`c_no`），然后外层的`DELETE`语句使用这些编号来删除`Course`表中相应的课程。

1. **找到在所有课程上的成绩都超过课程平均成绩的学生**：
   这个查询稍微复杂一些，因为我们需要找到每门课程的平均成绩，然后比较学生在每门课程上的成绩是否都超过了相应的平均成绩。我们可以使用如下查询：

```sql
SELECT s_no, s_name
FROM Student
WHERE NOT EXISTS (
  SELECT *
  FROM SC as sc1
  WHERE sc1.s_no = Student.s_no AND sc1.grade <= (
    SELECT AVG(grade)
    FROM SC as sc2
    WHERE sc2.c_no = sc1.c_no
  )
);
```

在这个查询中，外层查询选择了所有学生，内层的`NOT EXISTS`子查询检查每个学生是否有任何一门课程的成绩不超过该课程的平均成绩。如果不存在这样的课程（即学生在所有课程上的成绩都超过了平均成绩），那么这个学生就会被包含在最终结果中。