数据库SQL介绍

- Def：SQL是一种极其高效的数据库系统语言；可以实现对数据库中的数据进行增删改查等操作
- 增加操作：使用create命令：
    - 可以create table
    - 可以create View
    - 可以create Index
    - 可以create trigger/procedure
    - insert 用于增加单独一行
    - 如果我想单独向某个表中增加一列(增加一个属性)：Alter Table 表名 add 属性名 属性数据类型 +各种约束;
- 删除操作：使用drop命令
    - 相应的，也可以drop table
    - 可以drop View
    - 可以drop Index
    - 可以drop trigger/procedure
    - delete 用于删除行数据
    - 语法规则：drop 对象 对象名
    - 删除某一个属性，alter table 表名 drop column 属性名;
- 更改操作：用于更新数据，使用update命令
- 查找数据：使用select - from - where范式进行查找

数据库的数据类型

> 数据库中每一张表的每一个属性都可以人为将其定义为不同的数据类型，具体来说，有以下的数据类型：

- 整数类型（Integer Types）：用于存储整数值，包括整数、小整数和大整数等。常见的整数类型有：INT、TINYINT、SMALLINT、BIGINT。
- 浮点数类型（Floating-Point Types）：用于存储带有小数部分的数值。常见的浮点数类型有：FLOAT、DOUBLE。
- 字符串类型（Character Types）：用于存储字符和文本数据。常见的字符串类型有：CHAR、VARCHAR、TEXT。
- 日期和时间类型（Date and Time Types）：用于存储日期、时间和日期时间值。常见的日期和时间类型有：DATE、TIME、DATETIME、TIMESTAMP。
- 布尔类型（Boolean Type）：用于存储真值（True/False）。

- 二进制类型（Binary Types）：用于存储二进制数据，如图像、音频或视频文件。常见的二进制类型有：BINARY、VARBINARY、BLOB。
- 数组类型（Array Types）：用于存储多个值的集合。某些数据库系统提供了对数组类型的支持，如PostgreSQL的ARRAY类型。
- JSON类型：用于存储JSON（JavaScript Object Notation）格式的数据。某些数据库系统提供了对JSON类型的原生支持。

  此外，还有其他一些特殊的数据类型，如空值类型（NULL Type）用于表示缺失的或未知的值，以及几何类型（Geometry Types）用于存储几何数据（如点、线、多 边形等）。

  需要注意的是，不同的数据库系统可能具有不同的命名和语法规则，因此在具体使用时，还需要参考相应数据库的文档或规范。

数据库中的约束问题

> 我们在某一个数据库的某一张表中，往往会对一些属性有约束，例如在学生信息这个表中，学生的学号必须是唯一的，所以我们就有必要对数据进行一些约束，以保证我们插入的数据是合理合法的，满足我们现实世界对于数据库系统的要求。
>
> 在数据库中，属性的约束用于定义和保证属性值的有效性和一致性。以下是一些常见的属性约束：

1. 唯一约束（Unique Constraint）：用于确保属性值在表中是唯一的。唯一约束可以应用于一个或多个属性，但每个属性的值都必须是唯一的。
2. 主键约束（Primary Key Constraint）：用于唯一标识表中的每一行数据。主键约束要求属性值是唯一的，并且不能为NULL（空值）。
3. 外键约束（Foreign Key Constraint）：用于建立表与表之间的关系。外键约束要求属性值必须与另一个表中的主键值相匹配。它用于维护表之间的引用完整性，确保引用的表存在相关的记录。
4. 非空约束（Not Null Constraint）：用于确保属性值不能为空（非NULL）。非空约束要求属性值在插入或更新时不能为NULL。
5. 默认约束（Default Constraint）：用于为属性提供默认值。如果没有为属性提供值，则默认约束会自动为属性分配默认值。
6. 检查约束（Check Constraint）：用于定义属性值的条件约束。检查约束允许需要用户指定一个条件表达式，该表达式决定了哪些值是允许的。
7. 级联约束（Cascade Constraint）：用于指定当关联表中的行发生更新或删除操作时，对相关的表进行相应的级联操作。常见的级联操作包括级联更新（CASCADE UPDATE）和级联删除（CASCADE DELETE）。

   这些约束可以单独使用，也可以组合使用，以实现更复杂的数据完整性和业务规则。数据库管理系统提供了语法和功能来创建和管理这些约束，以确保数据的一致性和完整性。

SQL查询(Important 上机要考)

查询的基本范式是使用select子句，使用select ___ from ____ where _____的基本范式，意为什么位置找到满足什么条件的什么属性，如果我们想要查询所有元素，我们第一个____处可以填*号，就是将整张表中所有满足条件的元组全都返回

SQL（Structured Query Language）查询用于从数据库中检索数据。SQL查询语句通常使用SELECT语句来指定要检索的数据和检索的条件。下面是SQL查询的主要内容：

1. SELECT子句：用于指定要检索的列或表达式。可以使用通配符（*）检索所有列，也可以逐个列出要检索的列。例如：

```
SELECT column1, column2 FROM table_name;
```

2. FROM子句：用于指定要检索数据的表或视图。可以指定一个或多个表，并使用适当的连接条件进行关联。例如：

```
SELECT column1, column2 FROM table1, table2 WHERE table1.column =
table2.column;
```

3. WHERE子句：可选的，用于指定条件来筛选检索的数据。可以使用比较运算符（例如等于、大于、小于）、逻辑运算符（例如AND、OR、NOT）和通配符进行条件匹配。例如：

```
SELECT column1, column2 FROM table_name WHERE condition;
```

4. GROUP BY子句：可选的，用于根据一个或多个列对检索的数据进行分组。通常与聚合函数（如SUM、AVG、COUNT）一起使用，以便对每个分组计算汇总值。例如：

```
SELECT column1, SUM(column2) FROM table_name GROUP BY column1;
```

5. HAVING子句：可选的，用于在GROUP BY之后对分组结果进行过滤。可以使用聚合函数和比较运算符来指定过滤条件。例如：

```
SELECT column1, SUM(column2) FROM table_name GROUP BY column1 HAVING
condition;
```

6. ORDER BY子句：可选的，用于指定结果集的排序顺序。可以按升序（ASC）或降序（DESC）对一个或多个列进行排序。例如：

```
SELECT column1, column2 FROM table_name ORDER BY column1 ASC;
```

7. LIMIT子句（或类似的语法）：可选的，用于限制结果集返回的行数。不同的数据库系统可能有不同的语法来实现此功能。例如：

```
SELECT column1, column2 FROM table_name LIMIT 10;
```

除了以上主要内容外，还有其他高级功能和语法，如子查询、联合查询、连接（JOIN）操作、表达式、函数等，可以进一步扩展和优化SQL查询的能力。下面我们将会对这些内容进行说明

- 如果我们想要对查询的结果进行重命名，例如如果表中元素的属性名叫做Stu_ID和Stu_Name，我们想要查询所有Stu_ID满足最后一位数是1的学生的学号和姓名，最后展示查询结果用Number和Name的形式给出，那么我们的SQL语句是：

```
SELECT Stu_ID AS number,Stu_Name AS Name From Student where Stu_ID%10=1;
```

- 我们同样可以对查询结果进行一些处理，例如我们有一张表，查询的结果是美元，我们要换算成人民币，我们可以使用类似于以下格式的SQL语句：

```
SELECT Bank_id,dollar*5 AS RMB From Bank where Dollar>1000;
```

此语句将Bank表中所有满足Dollar>1000的所有的Bank_id和对应的余额(换算为人民币)以查询结果给出

- 模糊查找(查找部分)：Like运算符用于在 SQL 查询中进行模式匹配，它允许根据指定的模式或模式匹配符来筛选数据。LIKE 运算符通常与字符串列一起使用。LIKE 运算符使用以下两个通配符进行模式匹配：
    - 百分号（%）：表示零个、一个或多个字符的任意序列。
    - 下划线（_）：表示单个字符。
    - 使用百分号（%）进行模糊匹配：
        - 模式：'J%'，将匹配以字母 'J' 开头的任意字符串。
        - 模式：'%son'，将匹配以字母 'son' 结尾的任意字符串。
        - 模式：'%a%'，将匹配包含字母 'a' 的任意位置的字符串。

    ```
    SELECT column1, column2 FROM table_name WHERE column1 LIKE 'J%';
    ```

    - 使用下划线（_）进行精确匹配：
        - 模式：'H_l_'，将匹配以字母 'H' 开头，后跟一个任意字符，然后是字母 'l'，最后是任意字符的字符串。

```
SELECT column1, column2 FROM table_name WHERE column1 LIKE 'H_ll_';
```

值得注意的是，我们可以用LIKE运算符，也可以用NOT LIKE运算符代表不是这样的！

三值逻辑

在数据库中，Three-Valued Logic（三值逻辑）是一种逻辑体系，用于处理逻辑判断的结果。传统的二值逻辑中，逻辑判断的结果只有两个可能的取值：True（真）或 False（假）。而在Three-Valued Logic中，逻辑判断的结果可以是三个取值之一：True（真）、False（假）或 Unknown（未知）。

Three-Valued Logic的引入主要是为了处理缺失或不确定的数据情况。在数据库中，有时候某些属性的值可能是未知的或缺失的，这就导致对这些属性进行逻辑判断时无法得到明确的 True 或 False 结果。为了能够表示这种不确定性，引入了 Unknown 取值。

理解三值逻辑可以认为TRUE=1，FALSE=0，Unknown=0.5，然后我们的运算中，AND是MAX(a,b),OR是MIN(a,b),NOT(x)是1-x然后根据运算结果的数值来判断结果逻辑属于这三种状态中的哪一种

多关系查询

多关系查询（Multi-Table Queries）是在数据库中同时使用多个表进行查询的操作。它可以通过连接（Join）多个表来获取更丰富和综合的查询结果，从而实现更复杂的数据分析和数据提取。

但是相应的，我们也可以不使用join将多个表进行链接，而是在FROM中使用多个表来执行查询，而无需显式使用 JOIN 子句。这种方法称为隐式连接（Implicit Join）或逗号连接（Comma Join）。

- 隐式链接：例子：我想在查询某一门课程的学生的成绩，返回学生的姓名，学号，成绩，课程编号，其中Course表中存储了姓名，学号，课程编号，Courseinfo中存储了学号，成绩，课程编号，那么我们可以写出SQL：

  ```
  select Course.CourseID,Course.stu_Name,Course.stu_ID,Courseinfo.Grade
  From Course,Courseinfo
  where Course.CourID=Courseinfo.CourseID
  ```

- 显式链接：同样是上面的例子，我们可以写出：

  ```
  select Course.CourseID,Course.stu_Name,Course.stu_ID,Courseinfo.Grade
  From Course JOIN Courseinfo ON course.courseiD=Courseinfo.CourseID
  ```

  在这个例子里面，where子句被省略了，因为在内连接中，连接条件通常通过 ON 关键字在 JOIN 子句中指定，而不是在 WHERE 子句中进行筛选。需要注意的是，如果需要在内连接的基础上进一步筛选数

据，可以使用 WHERE 子句来添加其他条件。例如，可以添加额外的条件来过滤特定的行或应用其他约束。

- 链接的几种类型：
    - 内连接（Inner Join）：返回两个表中满足连接条件的匹配行。只有在两个表中都存在匹配的行时，才会返回结果。
    - 左连接（Left Join）：返回左表中的所有行，以及右表中与左表匹配的行。如果右表中没有匹配的行，则用 NULL 值填充右侧的列。
    - 右连接（Right Join）：返回右表中的所有行，以及左表中与右表匹配的行。如果左表中没有匹配的行，则用 NULL 值填充左侧的列。
    - 全连接（Full Join）：返回左表和右表中的所有行，并将匹配的行组合在一起。如果某个表中没有匹配的行，则用 NULL 值填充对应的列。

子查询

> 子查询（Subquery）是在 SQL 查询中嵌套在另一个查询中的查询语句。子查询允许我们在一个查询中使用另一个查询的结果作为条件、表达式或过滤条件。
>
> 子查询通常用于以下情况：

1. 过滤条件：子查询可以作为主查询的 `WHERE` 子句中的条件来过滤数据。子查询的结果将作为条件进行匹配。 `sql SELECT column1, column2 FROM table1 WHERE column1 IN (SELECT column1 FROM table2 WHERE condition);` 在这里，我们使用了IN运算符，IN 运算符用于在 SQL 查询中检查一个表达式是否与给定的值列表中的任何一个匹配。它可以用于过滤数据或作为子查询中的条件。IN 运算符和等号（=）运算符在 SQL 查询中都用于比较值，但它们之间有一些区别：
    - 匹配方式：
        - 等号运算符（=）用于检查两个值是否相等。它进行的是精确匹配，要求比较的两个值在类型和内容上完全相同。
        - IN 运算符用于检查一个表达式是否与给定的值列表中的任何一个值匹配。它进行的是值的匹配，而不要求严格的类型和内容相同。
    - 操作数：
        - 等号运算符需要两个操作数，分别是待比较的两个值或表达式。
        - IN 运算符需要一个操作数作为待比较的表达式，以及一个值列表作为要进行匹配的值。
    - 使用场景：
        - 等号运算符通常用于比较两个单独的值是否相等，例如在 WHERE 子句中进行筛选。
        - IN 运算符常用于将一个表达式与给定的值列表进行匹配，用于筛选多个值的情况，或者在子查询中使用

2. 列表或表达式：子查询可以作为 `SELECT` 语句中的列、计算字段或表达式的一部分。子查询的结果将作为列的值或表达式的一部分返回。

    ```sql
    SELECT column1, (SELECT AVG(column2) FROM table2) AS avg_value
    FROM table1;
    ```

3. 表连接：子查询可以在 `FROM` 子句中用作表，与其他表进行连接，以便生成更复杂的查询结果。

    ```sql
    SELECT t1.column1, t2.column2
    FROM (SELECT * FROM table1 WHERE condition1) AS t1
    JOIN (SELECT * FROM table2 WHERE condition2) AS t2 ON t1.id = t2.id;
    ```

    子查询可以嵌套多层，并且可以使用各种操作符和条件来生成复杂的查询逻辑。它们提供了一种灵活的方式来构建复杂的查询和数据分析。需要注意的是，子查询的性能可能会受到影响，特别是在处理大量数据和复杂查询时。合理使用子查询并优化查询语句是确保查询性能的关键。在实际使用子查询时，应注意子查询的语法和逻辑正确性，并确保查询的可读性和可维护性。根据具体的数据库系统和查询需求，可以参考相关文档和规范来了解更多关于子查询的详细信息。

exist运算符

`EXISTS` 运算符用于在 `SQL` 查询中检查子查询的结果是否存在。它返回一个布尔值，表示子查询是否返回至少一行数据。

`EXISTS` 运算符的语法如下：

```sql
EXISTS (subquery)
```

其中，`subquery` 是一个子查询，它可以是一个完整的 `SELECT` 语句或任何返回结果集的查询。

EXISTS 运算符的工作原理如下：

1. 对于主查询中的每一行，它会评估子查询并检查子查询的结果是否为空。
2. 如果子查询返回至少一行数据，则 EXISTS 运算符返回 True。
3. 如果子查询不返回任何行，即为空结果集，则 EXISTS 运算符返回 False。

以下是一个使用 EXISTS 运算符的示例：

```sql
select name
from beers b1
where not exists(
    select *
    from beers
    where b1.manf=manf AND b1.name<>name
);
```

在这个示例中，主查询根据 EXISTS 运算符的结果决定是否返回数据。子查询指定了另一个表和条件，如果子查询返回至少一行数据，则主查询将返回满足条件的行。具体而言，我们将beers表中进行遍历，对于每一行beers表的值，都进行一次遍历，判断exists()中是否有满足条件的元组，如果有，则exists的值是1，我们这里如果exists是0，那么就会在查询结果中显示出来。实现的功能是找到beers表中beers.manf只有一个name的所有name值。

EXISTS 运算符常用于以下情况：

- 确定一个表中是否存在满足特定条件的行。
- 在查询中使用 EXISTS 子查询来进行复杂的筛选和连接操作。

需要注意的是，子查询的结果并不会实际返回给主查询使用，而仅用于判断 EXISTS 运算符的结果。因此，子查询的选择和优化对查询性能和结果的正确性都非常重要。

在实际使用 EXISTS 运算符时，需要注意子查询的逻辑和语法，并确保查询的正确性和性能。根据具体的查询需求和数据比较的情况，选择合适的运算符和查询方法。

ANY运算符&all运算符

ANY 运算符在 SQL 查询中用于比较一个表达式与一组值中的任意一个值是否满足条件。它可以与比较运算符（如 >、<、= 等）结合使用。ANY 运算符的语法如下：

    ```sql
    expression operator ANY (subquery)
    ```

其中，expression 是要进行比较的表达式，operator 是比较运算符（如 >、<、= 等），subquery 是一个子查询，它返回一个结果集。

ANY 运算符的工作原理如下：

1. 子查询返回一个结果集，该结果集包含一组值。
2. 对于主查询中的每个行，它会将表达式与子查询结果集中的每个值进行比较。
3. 如果表达式与子查询中的任意一个值满足比较条件，则 ANY 运算符返回 True。

4. 如果表达式与子查询中的所有值都不满足比较条件，则 ANY 运算符返回 False。

以下是一个使用 ANY 运算符的示例：

```sql
SELECT column1, column2
FROM table_name
WHERE column1 > ANY (SELECT column3 FROM another_table WHERE condition);
```

在这个示例中，主查询使用 ANY 运算符将 column1 的值与子查询中的值进行比较。子查询指定了另一个表和条件，它返回一组值。如果 column1 的值大于子查询中的任意一个值，则主查询将返回满足条件的行。

ANY 运算符常用于以下情况：

- 比较一个表达式与一组值中的任意一个值，以确定是否满足特定条件。
- 在查询中使用 ANY 子查询来进行复杂的筛选和条件判断。

需要注意的是，子查询的结果集中的值应与待比较的表达式具有兼容的数据类型，以确保比较操作的正确性。同时，子查询的选择和优化对查询性能和结果的正确性都非常重要。

在实际使用 ANY 运算符时，需要注意子查询的逻辑和语法，并确保查询的正确性和性能。根据具体的查询需求和数据比较的情况，选择合适的运算符和查询方法。

事实上，我们可以把any理解为其中一个，也就是说我们对于any运算符，只需要满足待查或者待判断属性比集合中的一个元素大或者小就可以返回正常的元组。相应的，ALL运算符则是需要满足比集合中的元素都大或者都小才可以返回元组。(类似数学中的存在和任意)

用途：可以用于寻找到最大/最小的元组，例如寻找beers表中价格最高的啤酒： `sql select beers from beers where price>ANY(select price from beers);`

并叉交运算

在关系型数据库中，UNION、INTERSECTION和DIFFERENCE是用于组合和操作多个查询结果的集合运算符。

1. UNION（并集）：UNION 运算符用于合并两个或多个查询结果集，并返回唯一的、合并后的结果集。它会去除重复的行，并按照列的顺序进行合并。注意，UNION 运算符要求每个查询结果中的列数和数据类型必须一致。

   ```sql
   SELECT column1, column2 FROM table1
   UNION
   SELECT column1, column2 FROM table2;
   ```

以上示例将合并 table1 和 table2 的查询结果，返回一个包含合并后结果的唯一结果集。

2. INTERSECTION（交集）：INTERSECTION 运算符用于从两个查询结果集中获取共同存在的行，并返回结果集。它只返回在两个结果集中都存在的行。

```
SELECT column1, column2 FROM table1
INTERSECTION
SELECT column1, column2 FROM table2;
```

以上示例将返回 table1 和 table2 结果集中共同存在的行。

3. DIFFERENCE（差集）：DIFFERENCE 运算符用于从一个查询结果集中去除与另一个查询结果集相同的行，并返回结果集。它返回在第一个结果集中存在但在第二个结果集中不存在的行。

```
SELECT column1, column2 FROM table1
DIFFERENCE
SELECT column1, column2 FROM table2;
```

以上示例将返回 table1 结果集中存在但 table2 结果集中不存在的行。

这些集合运算符（UNION、INTERSECTION、DIFFERENCE）提供了一种方便的方式来组合和操作查询结果集。它们对于从多个查询中提取需要的数据以及进行数据分析和报表生成非常有用。

需要注意的是，这些运算符要求查询结果集的结构和数据类型相匹配，同时也需要考虑运算符的使用方式和数据操作的逻辑。

确定结果集返回类型

Forcing Set/Bag Semantics（强制集合/袋子语义）是指在 SQL 查询中显式指定结果集的语义，以确保查询结果按照集合语义（Set Semantics）或袋子语义（Bag Semantics）返回。

在关系型数据库中，Set Semantics 和 Bag Semantics 是两种不同的结果集语义：

1. Set Semantics（集合语义）：根据集合的数学定义，Set Semantics 要求查询结果中的行是唯一的，不会包含重复行。重复行将被自动去除，只返回一个实例。

2. Bag Semantics（袋子语义）：Bag Semantics 允许查询结果中存在重复行。查询结果将按照实际的出现次序返回，包括重复的行。

在大多数情况下，默认情况下 SQL 查询遵循 Set Semantics，即自动去除重复的行。然而，有时我们可能需要显式指定 Bag Semantics，保留重复的行。

为了在 SQL 查询中强制使用 Set Semantics 或 Bag Semantics，可以使用以下关键字：

- DISTINCT：使用 DISTINCT 关键字可以明确指示查询使用 Set Semantics，去除重复的行。

```
SELECT DISTINCT column1, column2
FROM table_name;
```

- ALL：使用 ALL 关键字可以明确指示查询使用 Bag Semantics，返回所有的行，包括重复的行。ALL 是默认的行为，通常不需要显式指定。

```
SELECT column1, column2
FROM table_name
ALL;
```

需要注意的是，对于大多数查询，Set Semantics 是默认和期望的行为，因为它符合集合的数学定义。Bag Semantics 通常用于需要保留重复行的特定场景。使用强制集合/袋子语义时，应谨慎考虑查询需求和语义的选择，并确保结果符合预期。了解默认的语义以及显式设置语义的选项，有助于正确处理查询结果。

聚合函数

> Aggregations（聚合函数）在 SQL 查询中用于计算和汇总数据的特殊函数。它们作用于一组行，并返回单个值作为结果。聚合函数可以用于从表中提取统计信息或计算总和、平均值、最大值、最小值等。

常见的 SQL 聚合函数包括：

1. COUNT：用于计算指定列或表中行的数量。

```
SELECT COUNT(column) FROM table;
```

2. SUM：用于计算指定列的总和。

```
SELECT SUM(column) FROM table;
```

3. AVG：用于计算指定列的平均值。

```
SELECT AVG(column) FROM table;
```

4. MAX：用于获取指定列的最大值。

```
SELECT MAX(column) FROM table;
```

5. MIN：用于获取指定列的最小值。

```
SELECT MIN(column) FROM table;
```

这些聚合函数可以在 SELECT 语句中使用，并结合其他查询条件和操作符进行更复杂的数据处理。

还有其他一些常用的聚合函数，如：

- SUM(DISTINCT column)：计算指定列的唯一值之和。
- AVG(DISTINCT column)：计算指定列的唯一值的平均值。
- COUNT(DISTINCT column)：计算指定列的唯一值的数量。

聚合函数也可以与 GROUP BY 子句结合使用，以对结果进行分组和计算聚合值。

需要注意的是，聚合函数通常会忽略 NULL 值，除非使用特殊的函数如 COUNT(*) 来计算包括 NULL 值的行数。

聚合函数提供了在查询中进行统计和计算的强大工具，使我们能够从数据中提取有意义的信息和汇总结果。使用聚合函数可以进行数据分析、生成报表以及获取有关数据集的有用摘要信息。

分组操作

> 在 SQL 查询中，Grouping（分组）是一种对结果集进行分组操作的机制。它结合了聚合函数和 GROUP BY 子句，用于将数据按照指定的列或表达式进行分组，并对每个分组应用聚合函数。

Grouping 的主要步骤如下：

1. 指定分组列或表达式：使用 GROUP BY 子句指定要进行分组的列或表达式。根据这些列或表达式的值，数据将被分成不同的组。
2. 应用聚合函数：对于每个分组，应用一个或多个聚合函数，如 SUM、AVG、COUNT 等，计算每个分组的汇总结果。

以下是一个使用 Grouping 的示例：

```
SELECT column1, COUNT(*) as count
FROM table_name
GROUP BY column1;
```

在这个示例中，我们使用 GROUP BY 子句按照 column1 列的值对数据进行分组。对于每个分组，使用 COUNT(*) 函数计算每个分组中的行数，并将结果命名为 count。最终的结果将返回每个分组的 column1 值和对应的行数。

Grouping 可以帮助我们进行数据分析、生成报表以及从数据中提取有意义的信息。它使我们能够根据不同的条件对数据进行分组，并计算每个分组的聚合值。通过分组，我们可以了解不同组之间的差异、趋势和汇总信

息。

需要注意的是，聚合函数只能应用于 SELECT 语句中的非分组列或表达式。如果需要筛选分组结果，可以使用 HAVING 子句进行条件过滤。总结而言，Grouping 是 SQL 中用于将数据按照指定列或表达式进行分组并应用聚合函数的机制。它为我们提供了从数据中提取有用信息的强大工具。

Having子句

```
HAVING 子句和 WHERE 子句在 SQL 查询中用于过滤数据，但它们有一些关键的区别：
```

1. 应用位置：
    - WHERE 子句用于在执行聚合之前对原始数据进行筛选。它出现在 SELECT 语句中的 FROM 子句之后和 GROUP BY 子句之前。
    - HAVING 子句用于在执行聚合后对分组结果进行筛选。它出现在 GROUP BY 子句之后。
2. 操作对象：
    - WHERE 子句用于筛选原始数据行，根据指定的条件过滤掉不符合条件的行。
    - HAVING 子句用于筛选分组后的结果集，根据指定的条件过滤掉不符合条件的分组。
3. 使用聚合函数：
    - WHERE 子句可以使用聚合函数，但它是在原始数据上运行的，而不是在分组后的结果上运行的。它可以用于筛选满足特定条件的聚合结果。
    - HAVING 子句必须使用聚合函数，它用于筛选分组后的结果集中的聚合结果，只有满足条件的分组才会包含在最终结果中。

以下是一个示例来说明 WHERE 子句和 HAVING 子句的区别：

```sql
SELECT column1, COUNT(*)
FROM table_name
WHERE column2 > 10
GROUP BY column1
HAVING COUNT(*) > 5;
```

在这个示例中，WHERE 子句用于在执行聚合之前筛选原始数据，只保留 column2 大于 10 的行。然后，数据按照 column1 进行分组，并计算每个分组的行数。最后，HAVING 子句用于筛选分组结果，只包含行数大于 5 的分组。