08_数据库高级和新特性

2024全新MySQL企业开发版

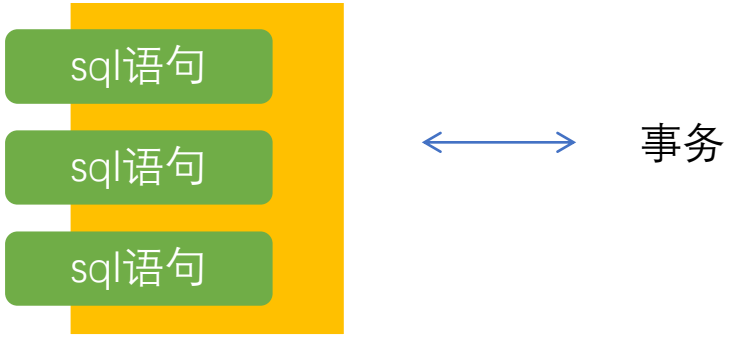
目录
Contents

1 数据库事务

1 数据库事务

1.1 数据库事务概述

数据库事务(transaction)是一套操作数据命令的有序集合，一个不可分割的工作单位。
事务中单个命令不会立即改变数据库数据,当内部全部命令执行成功,统一更新数据,当有任一命令失败,可以进行状态回滚。
事务由事务开始与事务结束之间执行的全部数据库操作组成。



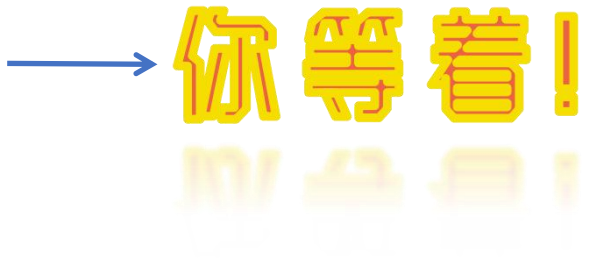
事务作用：

1. 为数据库操作序列提供了一个从失败中恢复到正常状态的方法，
2. 当多个应用程序在并发访问数据库时，以防止彼此的操作互相干扰。

1 数据库事务

1.1 数据库事务概述

生活场景：临近期末高考,你偷偷逃课出去上网吧,被父母当场抓住!!!!



逃学上网

高考临近

一个事务

程序场景：我们要实现一个转账业务,涉及两个具体数据修改动作(加钱|减钱)!!!!

结果1: 加钱执行成功,减钱执行成功,业务成功!

结果2: 加钱执行成功,减钱执行失败,业务失败!

结果3: 加钱执行失败,减钱执行成功,业务失败!

结果4: 加钱执行失败,减钱执行失败,业务失败!

发起者减钱

接收者加钱

一个事务

1.2 数据库事务ACID特性

原子性（Automicity）

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性（Consistency）

事务内部的操作的状态前后一致,成功都成功,失败都失败,避免部分成功出现错误数据。

隔离性（Isolation）

事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

持久性（Durability）

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响

1.3 事务的开启、提交、回滚

MySQL默认情况下是自动提交事务。
默认每一条语句都是一个独立的事务，一旦成功就提交了。语句报错失败就回滚。
我们的目标： 将多条语句加入到一个事务中

方案1: 手动提交模式

```
#开启手动提交事务模式（取消自动提交事务）
set autocommit = false;  或  set autocommit = 0;
上面语句执行之后，它之后的所有sql，都需要手动提交才能生效，直到恢复自动提交模式。
#恢复自动提交模式
set autocommit = true; 或 set autocommit = 1;
# 查看是否自动提交
show variables like 'autocommit';
```

例如：

```
set autocommit = false;#设置当前连接为手动提交模式

update t_employee set salary = 15000  where ename = '孙红梅';

commit;#提交 或 rollback# 回滚
```

1.3 事务的开启、提交、回滚

MySQL默认情况下是自动提交事务。
每一条语句都是一个独立的事务，一旦成功就提交了。一条语句失败，单独一条语句不生效，其他语句是生效的。
我们的目标：将多条语句加入到一个事务中

方案2: 自动提交模式下开启独立事务

```
# 也可以在自动提交模式下，开启一个事务。
start transaction;

# 添加多个sql命令

# 最后可以提交或者回滚
commit; 或 rollback;
```

例如：

```
start transaction; #开始事务

update t_employee set salary = 0  where ename = '李冰冰';

#下面没有写commit;那么上面这句update语句没有正式生效。
commit; 或 rollback; #提交
```

1.3 事务的开启、提交、回滚

MySQL默认情况下是自动提交事务。
每一条语句都是一个独立的事务，一旦成功就提交了。一条语句失败，单独一条语句不生效，其他语句是生效的。
我们的目标：将多条语句加入到一个事务中

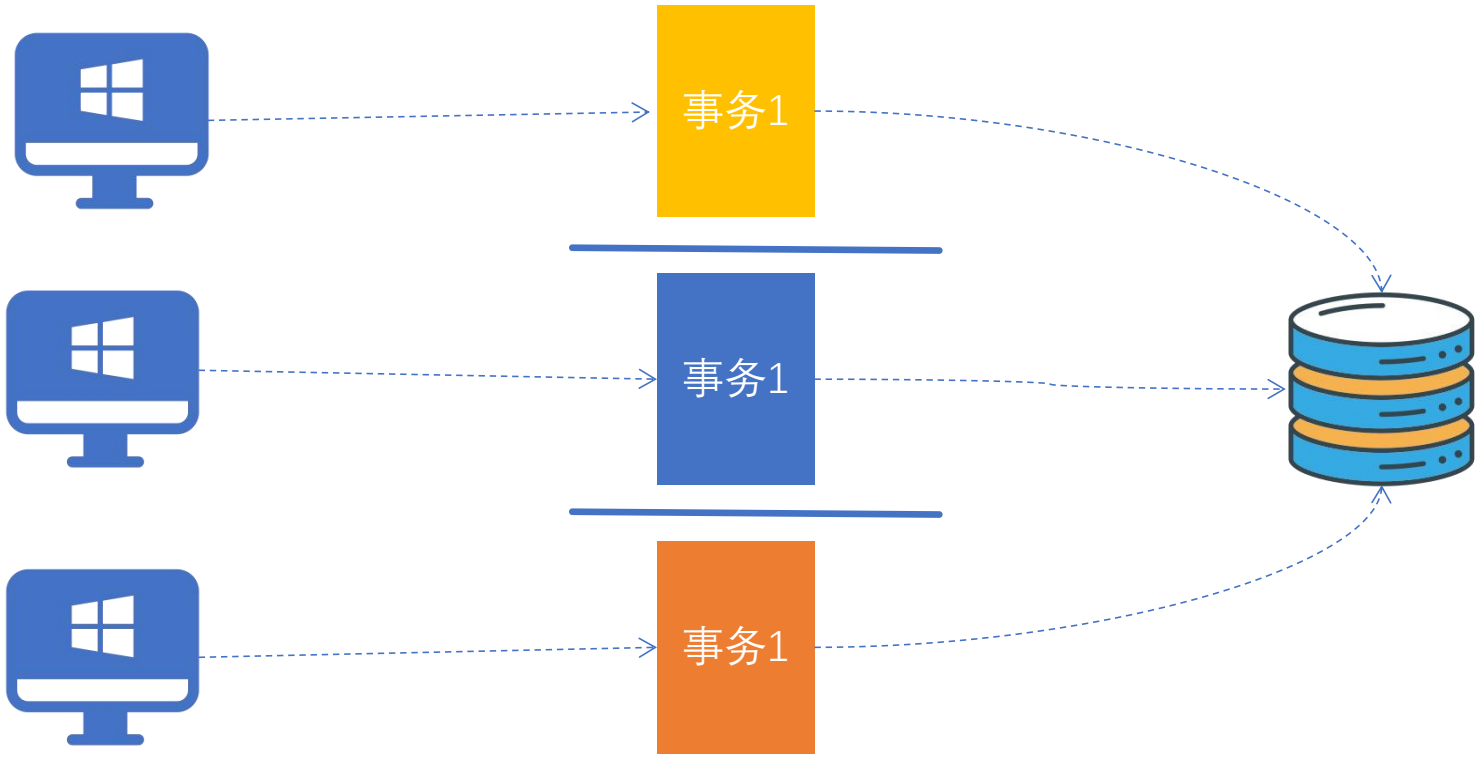
方案3: DDL语句不支持事务(注意注意)

```
#说明：DDL不支持事务
#DDL：create,drop,alter等创建库、创建表、删除库、删除表、修改库、修改表结构等这些语句不支持事务。
#换句话只对insert,update,delete语句支持事务。
TRUNCATE 表名称; 清空整个表的数据，不支持事务。 把表drop掉，新建一张表。

START TRANSACTION;
TRUNCATE t_employee;
ROLLBACK; #回滚 无效
```

1.4 事务隔离性概述和演示

一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。



1.4 事务隔离性概述和演示

一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

隔离级别	概述	脏读	不可重复读	幻读
read-uncommitted	读未提交事务数据	是	是	是
read-committed	读已提交事务数据(oracle)	否	是	是
repeatable-read	可重复度(mysql)	否	否	是(小概率)
serializable	串行化和序列化	否	否	否

数据库事务的隔离性：一个事务与其他事务隔离的程度称为隔离级别。数据库规定了多种事务隔离级别，不同隔离级别对应不同的干扰程度，隔离级别越高，数据一致性就越好，但并发性能越弱。

脏读：一个事务读取了另一个事务未提交数据；

不可重复读：一个事务读取了另一个事务提交的修改数据。

幻读：一个事务读取了另一个事务提交的新增、删除的记录情况，记录数不一样，像是出现幻觉。

```
#修改隔离级别:
set transaction_isolation='隔离级别';

#查看隔离级别:
select @@transaction_isolation;
```

目录
Contents

2 用户权限控制管理

2 用户权限控制管理

2.1 用户权限控制管理概述

MySQL使用权限来限制用户对数据库和表的访问。权限可以授予到数据库级别、表级别或特定操作上。

基本权限类型:

权限项	权限介绍
SELECT	允许用户查询表中的数据
INSERT	允许用户插入新数据到表中
UPDATE	允许用户更新表中的现有数据
DELETE	允许用户从表中删除数据
CREATE	允许用户创建新的数据库或表
DROP	允许用户删除数据库或表
GRANT OPTION	允许用户将其拥有的权限授予其他用户
ALL PRIVILEGES	允许用户执行所有操作

2 用户权限控制管理

2.2 用户权限控制管理实现

MySQL权限控制,具体现以下几个方向操作: **创建用户, 赋予权限, 回收权限 , 删除用户**

创建用户语法:

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
# 'username' 表示要创建的用户的用户名
# 'localhost' 表示用户可以通过连接到MySQL服务器的本地主机,其中'%'表示允许来自任何主机的连接。
# 'password' 表示要创建的用户的用户名
```

赋予权限语法:

```
# 1. 赋予全部权限
GRANT ALL PRIVILEGES ON database_name.table_name TO 'username'@'localhost';
# database_name:* 表示全部库权限
# table_name: * 代表全部表

# 2. 指定库和权限
GRANT SELECT, INSERT ON database_name.table_name TO 'username'@'localhost';
```

2 用户权限控制管理

2.2 用户权限控制管理实现

MySQL权限控制,具体现以下几个方向操作: **创建用户, 赋予权限, 回收权限 , 删除用户**

回收权限语法:

```
# 撤销全部权限
REVOKE ALL PRIVILEGES ON database_name.* FROM
'username'@'localhost';
# 撤销部分权限
REVOKE SELECT, INSERT, UPDATE ON database_name.table_name FROM
'username'@'localhost';
```

查看权限语法:

```
# 查看权限
SHOW GRANTS FOR 'username'@'localhost';
# username用户名
# 查看有用户列表
SELECT User, Host FROM mysql.user;
```

删除用户语法:

```
# 删除用户
DROP USER `用户名`;
```

目录
Contents

3 数据备份和还原

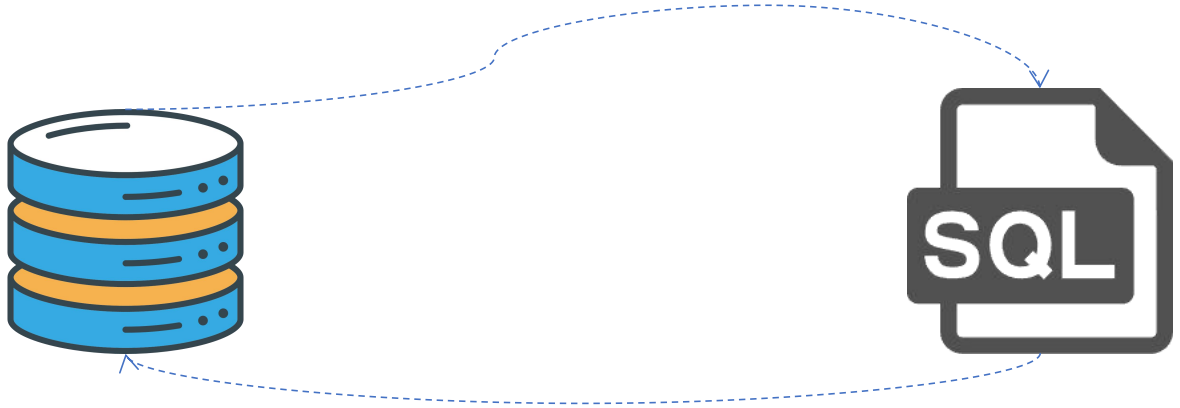
3 数据备份和还原

概述

MySQL的数据备份是指将数据库中的数据、表结构、以及可能的其他相关信息定期复制并保存到安全的位置，以防止数据丢失、数据库损坏或者其他意外事件导致的数据丢失情况。数据备份是数据库管理中至关重要的一环，它确保了数据库的可靠性、可恢复性和持久性。

应用场景

灾难恢复： 当数据库发生故障或数据丢失时，可以通过备份来快速恢复数据，以最小化业务中断。
数据迁移： 在服务器迁移、数据库升级或复制时，备份是迁移过程中的关键步骤。
数据复制： 将数据库备份复制到不同的地理位置或云存储中，以提高数据的安全性和可用性。
测试和开发： 在开发、测试或演示环境中使用备份数据，以避免影响生产环境中的实际数据。



3 数据备份和还原

全量备份实现

```
# 备份单库和单表
mysqldump -u username -p database_name  表名> backup.sql

# 备份单库和多表
mysqldump -u username -p database_name  表名1 表名2 ...> backup.sql

# 备份单库的所有表
mysqldump -u username -p database_name > backup.sql

# 例如
mysqldump -uroot -proot studb > d:/backup.sql
# -p如果写密码必须紧贴-p参数
# 以上命令必须在未连接mysql的状态下执行(打开cmd执行)
```

全量恢复实现

```
# 还原数据
mysql -u username -p database_name < backup.sql

# 注意: 需要提前准备数据库,导入已存在的库
# 注意: 在迁移过程中，确保源服务器和目标服务器的 MySQL 版本兼容。
```

3 数据备份和还原

开发中,不小心误删了数据怎么办。

Binlog运行日志介绍

MySQL的二进制日志记录,binlog记录数据库所有的增删改查的操作,同时也包括操作的执行时间。我们可以利用日志文件实现： 误删除数据恢复、增量复制、主从同步等!

开启Binlog配置(默认开启)

mysql配置文件: C:\ProgramData\MySQL\MySQL Server 8.2\my.ini (默认位置)

```
[mysqld]
log-bin=mysql-bin
datadir=C:/ProgramData/MySQL/MySQL Server 8.0/Data
```

log-bin表示启用二进制日志，并指定日志文件名, 命名规则: mysql-bin.000001开始
datadir 指定了 MySQL 数据目录的位置，日志文件缓存到此文件夹中

3 数据备份和还原

准备数据和日志文件

我们准备数据,并且在中间删除,并且产生日志文件!!

```
#1. 清空原有的日志文件
RESET MASTER;

#2. 准备数据,插入数据 -> 000001日志文件
CREATE DATABASE test08_binlog;
USE test08_binlog;
CREATE TABLE table_01(
  id INT PRIMARY KEY AUTO_INCREMENT,
  NAME VARCHAR(20) NOT NULL
);
INSERT INTO table_01(NAME) VALUES('二狗子'),('驴蛋蛋');

#3. 重启一个新的日志文件 -> 000002日志文件
FLUSH LOGS;

#4. 将删除数据和插入数据植入第二个日志文件 -> 000002日志文件
DELETE FROM table_01 WHERE id = 2; #删除驴蛋蛋
INSERT INTO table_01(NAME) VALUES('狗剩子');
SELECT * FROM table_01;
```

3 数据备份和还原

查看日志文件清单

```
#1.查看日志文件清单
SHOW BINARY LOGS;
# 结果
Log_name          File_size  Encrypted
----------------- --------- -----------
my_logbin.000001       1023  No
my_logbin.000002        787  No

#2. 查看日志文件信息
SHOW BINLOG EVENTS; #查看第一个日志文件 000001
SHOW BINLOG EVENTS [IN '指定查看日志文件名' FROM Pos值 LIMIT OFFSET,ROW_COUNT]; #查看指定日志文件
```

Log_name	Pos	Event_type	Server_id	End_log_pos	Info
my_logbin.000002	4	Format_desc	1	126	Server ver: 8.2.0, Binlog ver: 4
my_logbin.000002	126	Previous_gtids	1	157	
my_logbin.000002	157	Anonymous_Gtid	1	236	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
my_logbin.000002	236	Query	1	320	BEGIN
my_logbin.000002	320	Table_map	1	391	table_id: 124 (test08_binlog.table_01)
my_logbin.000002	391	Delete_rows	1	441	table_id: 124 flags: STMT_END_F
my_logbin.000002	441	Xid	1	472	COMMIT /* xid=43 */
my_logbin.000002	472	Anonymous_Gtid	1	551	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'

Pos：起始位置
Event_type： 命令的类型
End_log_pos: 终止位置
Info: 命令大致描述

3 数据备份和还原

查看日志文件详情

```
# 查看日志文件详情
# 可以查看详情
mysqlbinlog -v binlog日志文件
# 注意,mysqlbinlog不是sql命令,是mysql的工具
# mysql已经配置到环境变量,可以直接在cmd操作使用

# 例如:
cd 日志文件所在文件夹
> mysqlbinlog -v my_logbin.000002
```

3 数据备份和还原

恢复误删除的数据

目标: 找回删除的驴蛋蛋数据
思路: 找出删除语句之前和之后的一条命令pos,跳过删除,恢复之前和之后的数据即可

my_logbin.000002	391	Delete_rows	1	441	table_id: 124 flags: STMT_END_F

```
# 将删除前后的日志文件导成sql脚本
> mysqlbinlog my-logbin.000001> d:/my_binlog.000001.sql # 将其他的日志完整导出
> mysqlbinlog --stop-position=391 my-logbin.000002> d:/my_binlog.391.sql # 02日志删除之前
> mysqlbinlog --start-position=441 my-logbin.000002> d:/my_binlog.441.sql # 02日志删除之后
```

mysqlbinlog 常见的选项:
--start-datetime 从二进制日志中读取指定时间戳或者本地计算机时间之后的日志事件。
--stop-datetime 从二进制日志中读取指定时间戳或者本地计算机时间之前的日志事件。
--start-position从二进制日志中读取指定position 事件位置作为开始。
--stop-position 从二进制日志中读取指定position 事件位置作为事件截至。

```
# 利用脚本恢复数据,避免跑路
#1. mysql连接情况下执行,删除原有库
drop database test08_binlog;
# 2. 脚本恢复即可
mysql -uroot -proot < d:/my_binlog.000001.sql
mysql -uroot -proot < d:/my_binlog.391.sql
mysql -uroot -proot < d:/my_binlog.441.sql
# 不需要提前准备库表,因为日志生成的sql脚本带有建库和表语句
```

目录
Contents

4 数据库新特性(8+)

4 数据库新特性(8+)

4.1 窗口函数使用

MySQL 中的窗口函数是一种高级 SQL 功能，允许在结果集中执行聚合、分析和排序操作，而不会改变查询结果的行数。窗口函数通常与 OVER 子句一起使用，用于指定窗口的大小和行的排列顺序。它们对于分析数据、计算排名以及执行滑动窗口操作非常有用。

窗口函数总体上可以分为序号函数、分布函数、前后函数、首尾函数和其他函数，如下表：

函数分类	函数	函数说明
序号函数	ROW_NUMBER()	顺序排序
	RANK()	并列排序，会跳过重复的序号，比如序号为1、1、3
	DENSE_RANK()	并列排序，不会跳过重复的序号，比如序号为1、1、2
分布函数	PERCENT_RANK()	等级值百分比
	CUME_DIST()	累积分布值
前后函数	LAG(expr, n)	返回当前行的前n行的expr的值
	LEAD(expr, n)	返回当前行的后n行的expr的值
首尾函数	FIRST_VALUE(expr)	返回第一个expr的值
	LAST_VALUE(expr)	返回最后一个expr的值
其他函数	NTH_VALUE(expr, n)	返回第n个expr的值
	NTILE(n)	将分区中的有序数据分为n个桶，记录桶编号

语法1: 多行函数 OVER（[PARTITION BY 字段名 ORDER BY 字段名 ASC|DESC]）

语法2: 多行函数 OVER 窗口名 … WINDOW 窗口名 AS （[PARTITION BY 字段名 ORDER BY 字段名 ASC|DESC]）

4 数据库新特性(8+)

4.1 窗口函数使用

准备数据:

商品表: 商品编号、类别编号、类别名称、商品名、价格、库存、上架时间

序号函数: ROW_NUMBER()

ROW_NUMBER()函数能够对数据中的序号进行顺序显示。

```
# 举例1: 查询 goods 数据表中每个商品分类下价格降序排列的各个商品信息。
SELECT ROW_NUMBER() OVER(PARTITION BY category_id ORDER BY price DESC) AS
row_num, id, category_id, category, NAME, price, stock FROM goods;
# 窗口函数 over (partition by 分组 order by 排序 )

#举例2: 查询 goods 数据表中每个商品分类下价格最高的3种商品信息。
SELECT * FROM ( SELECT ROW_NUMBER() OVER(PARTITION BY category_id ORDER BY
price DESC) AS row_num, id, category_id, category, NAME, price, stock FROM goods) t
 WHERE row_num <= 3;
```

语法1: 函数 OVER（[PARTITION BY 字段名 ORDER BY 字段名 ASC|DESC]）

语法2: 函数 OVER 窗口名 … WINDOW 窗口名 AS （[PARTITION BY 字段名 ORDER BY 字段名 ASC|DESC]）

4 数据库新特性(8+)

4.1 窗口函数使用

准备数据:

商品表: 商品编号、类别编号、类别名称、商品名、价格、库存、上架时间

序号函数: RANK()

使用RANK()函数能够对序号进行并列排序，并且会跳过重复的序号，比如序号为1、1、3。

```
# 举例：使用RANK()函数获取 goods 数据表中各类别的价格从高到低排序的各商品信息。
SELECT RANK() OVER(PARTITION BY category_id ORDER BY price DESC) AS row_num,
    id, category_id, category, NAME, price, stock
    FROM goods;

#举例2：使用RANK()函数获取 goods 数据表中类别为“女装/女士精品”的价格最高的4款商品信息
 SELECT * FROM(
    SELECT RANK() OVER(PARTITION BY category_id ORDER BY price DESC) AS row_num,
    id, category_id, category, NAME, price, stock
    FROM goods) t
    WHERE category_id = 1 AND row_num <= 4;
```

4 数据库新特性(8+)

4.1 窗口函数使用

准备数据:

商品表: 商品编号、类别编号、类别名称、商品名、价格、库存、上架时间

序号函数: DENSE_RANK()

DENSE_RANK()函数对序号进行并列排序，并且不会跳过重复的序号，比如序号为1、1、2。

```
# 举例1: 使用DENSE_RANK()函数获取 goods 数据表中各类别的价格从高到低排序的各商品信息。
SELECT DENSE_RANK() OVER(PARTITION BY category_id ORDER BY price DESC) AS row_num, id, category_id, category, NAME, price, stock  FROM goods;

#举例2: 使用DENSE_RANK()函数获取 goods 数据表中类别为“女装/女士精品”的价格最高的4款商品信息。
SELECT * FROM( SELECT DENSE_RANK() OVER(PARTITION BY category_id ORDER BY price DESC) AS row_num,  id, category_id, category, NAME, price, stock FROM goods) t WHERE category_id = 1 AND row_num <= 3;
```

4 数据库新特性(8+)

4.1 窗口函数使用

准备数据:

商品表: 商品编号、类别编号、类别名称、商品名、价格、库存、上架时间

分步函数: PERCENT_RANK()

PERCENT_RANK()函数是等级值百分比函数。按照如下方式进行计算。

计算规则: (rank - 1) / (rows - 1)

```
# 举例1：计算 goods 数据表中名称为“女装/女士精品”的类别下的商品的PERCENT_RANK值。
#写法一：
SELECT RANK() OVER (PARTITION BY category_id ORDER BY price DESC) AS r,PERCENT_RANK()
OVER (PARTITION BY category_id ORDER BY price DESC) AS pr,id, category_id, category,
NAME, price, stock FROM goods WHERE category_id = 1;

#写法二：
SELECT RANK() OVER w AS r,
    PERCENT_RANK() OVER w AS pr,
    id, category_id, category, NAME, price, stock
    FROM goods
    WHERE category_id = 1 WINDOW w AS (PARTITION BY category_id ORDER BY price DESC);
```

4 数据库新特性(8+)

4.1 窗口函数使用

准备数据:

商品表: 商品编号、类别编号、类别名称、商品名、价格、库存、上架时间

分步函数: CUME_DIST()

CUME_DIST() 函数返回当前行的值在排序后结果集中的累积分布比例。它告诉你当前行的值在整个结果集中的相对位置。
计算规则: 即 rank / total_rows。

```
# 举例：查询goods数据表中小于或等于当前价格的比例。
SELECT CUME_DIST() OVER(PARTITION BY category_id ORDER BY price ASC) AS cd,
   id, category, NAME, price
   FROM goods;
```

4 数据库新特性(8+)

4.1 窗口函数使用

准备数据:

商品表: 商品编号、类别编号、类别名称、商品名、价格、库存、上架时间

前后函数: LAG(expr,n)

LAG(expr,n)函数返回当前行的前n行的expr的值。

```
# 举例: 查询goods数据表中前一个商品价格与当前商品价格的差值。
SELECT id, category, NAME, price, pre_price, price - pre_price AS diff_price
  FROM (SELECT  id, category, NAME, price,LAG(price,1) OVER w AS pre_price FROM goods
  WINDOW w AS (PARTITION BY category_id ORDER BY price)) t;
```

前后函数: LEAD(expr,n)

LEAD(expr,n)函数返回当前行的后n行的expr的值。

```
# 举例: 查询goods数据表中后一个商品价格与当前商品价格的差值。
 SELECT id, category, NAME, behind_price, price,behind_price - price AS diff_price
   FROM(SELECT id, category, NAME, price,LEAD(price, 1) OVER w AS behind_price
   FROM goods WINDOW w AS (PARTITION BY category_id ORDER BY price)) t;
```

4.1 窗口函数使用

首尾函数: FIRST_VALUE(expr)

FIRST_VALUE(expr)函数返回第一个expr的值。

```
# 举例：按照价格排序，查询第1个商品的价格信息。
SELECT id, category, NAME, price, stock,FIRST_VALUE(price) OVER w AS first_price
    FROM goods WINDOW w AS (PARTITION BY category_id ORDER BY price);
```

首尾函数: LAST_VALUE(expr)

LAST_VALUE(expr)函数返回最后一个expr的值。

```
# 举例：按照价格排序，查询最后一个商品的价格信息。
 SELECT id, category, NAME, price, stock,LAST_VALUE(price) OVER w AS last_price
    FROM goods WINDOW w AS (PARTITION BY category_id ORDER BY price);
```

4.2 公用表表达式

公用表表达式（或通用表表达式）简称为CTE（Common Table Expressions）。CTE是一个命名的临时结果集，作用范围是当前语句。CTE可以理解成一个可以复用的子查询，当然跟子查询还是有点区别的，CTE可以引用其他CTE，但子查询不能引用其他子查询。所以，可以考虑代替子查询。

公用表表达式分为: 普通公用表表达式和递归公用表表达式

```
# 普通公用表表达式的语法结构是:
WITH CTE名称
    AS （子查询）
    SELECT|DELETE|UPDATE 语句;
```

```
# 举例1：查询员工所在的部门的详细信息。(基于子查询)
SELECT * FROM departments WHERE department_id IN (
   SELECT DISTINCT department_id FROM employees );

#举例2: 普通公用表表达式的方式完成
WITH emp_dept_id AS (SELECT DISTINCT department_id FROM employees)
   SELECT * FROM departments d JOIN emp_dept_id e
   ON d.department_id = e.department_id;
# WITH emp_dept_id AS (SELECT DISTINCT department_id FROM employees) 声明虚拟表
```

感谢观看