数据库第七次上机

2022/5/12

本次上机概览

本次的上机内容为：

- 并发控制
  - 数据库隔离级别实验
  - 死锁
- 本次实验使用MySQL命令行界面进行实验；

本次上机任务

Task1：隔离级别的验证

使用命令行工具连接 MySQL 或使用可视化工具，实验需要设计 2个 MySQL session，对应为 2 个建立连接的命令行窗口或可视化工具中 2 个会话框；

建表：

```
CREATE TABLE account (
  id int NOT NULL PRIMARY KEY,
  name varchar(10) DEFAULT NULL,
  money int DEFAULT 0
)
INSERT into account values(1,  'tom' , 1000);
INSERT into account values(2,  'bob' , 0);
```

隔离级别：

1. read uncommitted
2. read committed
3. repeatable read
4. Serializable

本次上机任务

openGauss的隔离级别：

READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。

REPEATABLE READ：可重复读隔离级别，仅仅能看到事务开始之前提交的数据，不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。

SERIALIZABLE：openGauss目前功能上不支持此隔离级别，等价于REPEATABLE READ。

参考文档：

https://opengauss.org/zh/docs/2.1.0/docs/Developerguide/SET-TRANSACTION.html

为完整实验，选择MySQL作为本次实验平台；

SET TRANSACTION

功能描述

为事务设置特性。事务特性包括事务隔离级别、事务访问模式(读/写或者只读)。可以设置当前事务的特性（LOCAL），也可以设置会话的默认事务特性(SESSION)。

注意事项

设置当前事务特性需要在事务中执行（即执行SET TRANSACTION之前需要执行START TRANSACTION或者BEGIN），否则设置不生效。

语法格式

设置事务的隔离级别、读写模式。

复制代码

```
{ SET [ LOCAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
  { ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
  | { READ WRITE | READ ONLY } } [, ...]
```

参数说明

- **LOCAL**

  声明该命令只在当前事务中有效。

- **SESSION**

  声明这个命令只对当前会话起作用。

  取值范围：字符串，要符合标识符的命名规范。

- **ISOLATION_LEVEL**

  指定事务隔离级别，该参数决定当一个事务中存在其他并发运行事务时能够看到什么数据。

  > 说明：在事务中第一个数据修改语句（SELECT，INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，当前事务的隔离级别就不能再次设置。

  取值范围：

  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
  - REPEATABLE READ：可重复读隔离级别，仅仅能看到事务开始之前提交的数据，不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。
  - SERIALIZABLE：openGauss目前功能上不支持此隔离级别，等价于REPEATABLE READ。

- **READ WRITE | READ ONLY**

  指定事务访问模式（读/写或者只读）。

本次上机任务

请在word或pdf中标明题号，给出对应输出结果，并附带截图

Q1: 按照表格顺序执行，给出语句2、语句4和语句5的输出结果，并分析结果

Q2: 将语句1，语句3的隔离级别修改为 read committed / repeatable read / serializable，重新执行，记录语句2、语句4和语句5的输出结果，并分析结果

注意：以上每次执行时需要确认表格数据还原为原始状态，即 tom 1000，bob 0

脏读

时间	session1	session2

T1	set session transaction isolation level read uncommitted; // 语句1
	set autocommit = 0; // 关闭事务自动提交
	start transaction;
	update account set money = money + 1000 where id = 1;
	select * from account where id = 1; // 语句2	

T2		set session transaction isolation level read uncommitted; // 语句3
		start transaction;
		select * from account where id = 1; // 语句4

T3	rollback; commit;	
T4		select * from account where id = 1; // 语句5

本次上机任务

请在word或pdf中标明题号，给出对应输出结果，并附带截图

Q3: 按照表格顺序执行，给出语句2和语句4的输出结果，并分析结果

Q4: 将语句1，语句3的隔离级别修改为 repeatable read / serializable，重新执行，记录语句2和语句4的输出结果，并分析结果

注意：以上每次执行时需要确认表格数据还原为原始状态，即 tom 1000，bob 0

不可重复读

时间	session1	session2

T1	set session transaction isolation level read committed; // 语句1
	start transaction;
	select * from account where id = 2; // 语句2	

T2		set session transaction isolation level read commited; // 语句3
		start transaction;
		update account set money = money+1000 where id=2;
		commit;

T3	select * from account where id = 2 // 语句4	

本次上机任务

请在word或pdf中标明题号，给出对应输出结果，并附带截图

Q5: 按照表格顺序执行，给出T2与T4时刻语句2的输出结果与状态，并解释其原因；

注意：以上每次执行时需要确认表格数据还原为原始状态，即 tom 1000，bob 0

（tips: innoDB 对一般的select仅作快照读，共享锁（S锁）读需要设置 lock in share mode;）

不可重复读 - 行锁

时间	session1	session2

T1	set session transaction isolation level repeatable read;
	set autocommit = 0;
	start transaction;
	select * from account where id = 2 lock in share mode; // 语句1	

T2		set session transaction isolation level repeatable read; start transaction; update account set money = money+1000 where id=2; // 语句2
T3	commit;	
T4		

本次上机任务

请在word或pdf中标明题号，给出对应输出结果，并附带截图

Q6: 按照表格顺序执行，给出语句2和语句4的输出结果

Q7: 将语句1，语句3的隔离级别修改为 serializable，重新执行，记录语句2和语句4的输出结果

Q8: 若语句2 增加了 lock in share mode; 的设置，T2中语句 5、6 会阻塞；请结合两种隔离级别下解锁前后的现象，分析阻塞原因。

注意：以上每次执行时需要确认表格数据还原为原始状态，即 tom 1000，bob 0

幻读1

时间	session1	session2

T1	set session transaction isolation level repeatable read; // 语句1
	start transaction;
	set autocommit = 0;
	select * from account where id < 10; // 语句2	

T2		set session transaction isolation level repeatable read; // 语句3
		start transaction;
		insert into account values(3, alen, 0); // 语句5
		update account set money = money+1000 where id=2 // 语句6
		select * from account where id < 10;

T3	select * from account where id < 10; // 语句4	

本次上机任务

请在word或pdf中标明题号，给出对应输出结果，并附带截图

Q9: 按照表格顺序执行，给出语句2、语句3的执行结果，并分析原因

注意：以上每次执行时需要确认表格数据还原为原始状态，即 tom 1000，bob 0

tips: innoDB 对insert/update/delete 语句会加排他锁（X锁）

幻读 – 行锁

时间	session1	session2

T1	set session transaction isolation level serializable;
	set autocommit = 0;
	start transaction;
	update account set money=money+1000 where id=1;
	//语句1	

T2		set session transaction isolation level serializable; start transaction; delete from account where id=1; // 语句2
T3	commit;	
T4		delete from account where id=1; // 语句3

本次上机任务

Task2：死锁

使用命令行工具连接 MySQL 或使用可视化工具，实验需要设计 2个 MySQL session，对应为 2 个建立连接的命令行窗口或可视化工具中 2 个会话框；

建表：

create tableA(id, columnA) …
create tableB(id, columnB) …
Insert into tableA values(1, 0);
Insert into tableB values(1, 0);

开启两个命令行界面A和B，按照下面顺序在A、B中分别执行语句：

A: start transaction;（或Begin）
A: update tableA set columnA=1 where id=1;
B: start transaction;
B: update tableB set columnB=2 where id=1;
A: update tableB set columnB=3 where id=1;
B: update tableA set columnA=2 where id=1;

最后一条语句执行时系统会提示死锁，界面A会处于等待状态，需要手工杀掉死锁事务

Q10: 请解释死锁发生的原因，并按照链接方法手工杀死死锁事务
https://blog.csdn.net/lvoelife/article/details/119104757

关于作业提交

TASK1：Q1-Q9
TASK2：Q10
选做 Q11：TASK1中的实验session1与session2均设置为相同的隔离级别，思考不同场景下，若两个会话的隔离级别不同会发生什么；

请在**PDF/WORD等任何方便助教阅读查看的文档**中按照各个作业要求提交相关内容，**记得标清题号**。

若为PDF/WORD单文档文件直接提交即可，其他提交压缩包，命名为“学号_姓名_第*次实验”。

提交网址：软件学院云平台**第七次上机** https://cloud.beihangsoft.cn/#/security/login
（按要求提交）

作业截止时间为**周日24:00之前，提交方式为提交到云平台。**

基础知识

—事务

- 事务：

事务是单个的工作单元。 如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分。 如果事务遇到错误且必须取消或回滚，则所有数据修改均被清除。

- 事务的主要作用是面向多用户的并发访问。

- 事务的四个基本要素（的缩写）：ACID

（原子性） Atomicity ：一个事务中的所有操作，要么成功提交要么完全回滚。（原子操作）
（一致性） Consistency ：业务逻辑一致，数据约束完整。
（隔离性） Isolation ：事务间加锁隔离，控制可见度。
（持久性） Durability ：在事务完成以后，对数据库所作的更改持久的保存在数据库之中，不会被回滚。

- 保证ACID所要用到的技术

A ：TCL
I ：锁
C/D ：事务日志

基础知识

—事务

事务：原子性

- 隐式/自动提交事务
  SQL Server把每条单独的语句作为一个事务

- 运行错误发生时

回滚该发生错误的单语句事务
该批次其他语句继续执行

- 编译错误发生时

该批次语句都不执行

基础知识

—事务

事务：原子性

- 显式/手动提交事务：TCL语法

  BEGIN TRANSACTION [事务名];
  SAVE TRANSACTION <存档点名>;
  COMMIT TRANSACTION [事务名];
  ROLLBACK TRANSACTION [事务名];

- 事务始于BEGIN，终于COMMIT或者ROLLBACK

- 事务可以嵌套，但只能回滚至最外层事务的BEGIN点
  - 因此使用SAVE标记一个事务中的多个存档点
  - 不要使用嵌套事务，时常会引起回滚不可控的现象

- 意外的连接中断发生时
  - 回滚所有已BEGIN的事务

基础知识

—事务

```
-- --------[简单事务控制流]---------
BEGIN TRANSACTION T1;
    PRINT ('Begin T1');
    INSERT INTO Test VALUES(1, 0.1);
SAVE TRANSACTION S1;
    PRINT ('Save S1');
    INSERT INTO Test VALUES(2, 0.2);
SAVE TRANSACTION S2;
    PRINT ('Save S2');
    INSERT INTO Test VALUES(3, 0.3);
ROLLBACK TRANSACTION S1;
COMMIT TRANSACTION T1;
GO
```

```
-- ---[一般的事务框架：加入错误处理]-------
BEGIN TRY
    BEGIN TRANSACTION T;
    INSERT INTO Test VALUES(1, 0.1);
    INSERT INTO Test VALUES(2, 0.2);
    -- 主键冲突!
    INSERT INTO Test VALUES(2, 0.2);
    INSERT INTO Test VALUES(3, 0.3);
    COMMIT TRANSACTION T;
END TRY
BEGIN CATCH
    PRINT ('Error Caught!')
    ROLLBACK TRANSACTION T;
END CATCH
GO
```

基础知识

—事务

事务：独立性

- 锁

共享锁S：用于不修改数据的语句如DQL，读
排他锁X：用于数据修改如DML，写
更新锁U：用于批量UPDATE，读取阶段像S、修改阶段像X
意向锁I：对于锁作标记的假锁，用于自适应式锁升级

- 粒度

数据库、表、键、行、范围（偏逻辑）
文件、索引/堆、分配单元、区、页（偏物理）

```
-- -------[查看当前的锁状况]--------
EXECUTE sp_lock;
SELECT * from sys.dm_tran_locks;
```

当某人查询某张表的一条记录时，就会在该记录上放置共享锁，在而其他人也要查询这张表的此记录时，因为共享锁彼此不互斥，所以也可以再次放置共享锁。
也就是说SQL SERVER允许不同连接同时读取相同的数据。如果此时有人要更新此记录，因为独占锁与共享锁互斥，所以无法放置独占锁，要等到所有读取此记录的人都读取完毕，释放了共享锁，更新数据的人才能对该记录设置独占锁，并进一步更新数据。

锁粒度是被封锁目标的大小,封锁粒度小则并发性高,但开销大,封锁粒度大则并发性低但开销小

排他锁又可以叫独占锁

基础知识

—事务

事务：独立性

- ANSI隔离层级标准：

读取未提交：不使用锁

可见其他事务**未提交的修改**；**脏读**

读取已提交(**默认**的隔离层级)：读时无X锁

可见其他事物已提交的**数据值**修改；**不可重复读**

可见其他事物已提交的**记录数**修改；**幻读**

可重复读：读时加有限范围的S锁并保持

不可见其他事务已提交的数据值修改，避免不可重复读

可序列化：读时全事务范围的S锁并保持

不可见其他事务的任何操作，避免任何读问题

```
-- --------[手动修改事务隔离层级]--------
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
GO
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
GO
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
GO
```

基础知识

—事务

事务：独立性

- 阻塞

有S锁不可写
有X锁不可读写

```
-- --------[锁的忙等待]---------
-- 用户1先到，进行一个需要锁表的操作
BEGIN TRANSACTION T1;
    SELECT * FROM Test WITH (HOLDLOCK, XLOCK);
-- 用户1一直不提交T1

    -- 用户2后来，尝试修改被锁的表
    BEGIN TRANSACTION T2;
        UPDATE Test SET val = 2
            WHERE id=1;
    -- 导致用户2的操作忙等待
    COMMIT TRANSACTION T2;

-- 用户1终于决定提交
COMMIT TRANSACTION T1;
-- 之后用户2的UPDATE才被允许执行
```

注意：此类实验需要开多个查询窗口（比如可以让用户1和用户2从两个xxx.sql文件分步执行）

基础知识

—事务

事务：独立性

- 死锁：两个或多个会话相互请求对方持有的锁资源，导致循环等待的情况

- 死锁：四个必要条件

（1） **互斥条件：**一个资源每次只能被一个进程使用。

（2） **请求与保持条件：**一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3） **不剥夺条件：**进程已获得的资源，在未使用完之前，不能强行剥夺。

（4） **循环等待条件：**若干进程之间形成一种头尾相接的循环等待资源关系。

```
-- -------[手工模拟死锁]-------
-- 用户1：OK
BEGIN TRANSACTION T1;
SELECT * FROM Test WITH (HOLDLOCK, XLOCK) WHERE id = 1;

    -- 用户2：OK
    BEGIN TRANSACTION T2;
    SELECT * FROM Test WITH (HOLDLOCK, XLOCK) WHERE id = 2;

-- 用户1：忙等待
SELECT * FROM Test WITH (HOLDLOCK, XLOCK) WHERE id = 2;

    -- 用户2：死锁构成
    SELECT * FROM Test WITH (HOLDLOCK, XLOCK) WHERE id = 1;
```

```
消息 1205，级别 13，状态 51，第 13 行
事务(进程 ID 53)与另一个进程被死锁在 锁 资源上，并且已被选作死锁牺牲品。请重新运行该事务。
```

（同样需要开多个查询窗口）

表提示：WITH()

https://docs.microsoft.com/zh-cn/sql/t-sql/queries/hints-transact-sql-table?view=sql-server-2017

基础知识

—事务

事务：一致性/持续性

- 预写日志 (Write-Ahead Logging)和惰性写(Lazy Writer)作业流程
  在 缓冲区日志 中写入 Begin Tran记录
  在 缓冲区日志 中写入 要修改的信息(事务主体)
  将 缓冲区日志 的修改写出到 缓冲区数据页
  在 缓冲区日志 中写入 Commit Tran记录
  将 缓冲区日志 写出到 磁盘日志文件
  适时，将 缓冲区数据页 写出到 磁盘主数据文件

- 预写日志WAL的灾难恢复
  由早到晚**顺序**Redo已提交却未写入主数据文件的数据
  由晚到早**逆序**Undo未提交却已写入日志文件的数据

log文件中通常包括redo和undo信息。这样做的目的可以通过一个例子来说明。假设一个程序在执行某些操作的过程中机器掉电了。在重新启动时，程序可能需要知道当时执行的操作是成功了还是部分成功或者是失败了。如果使用了WAL，程序就可以检查log文件，并对突然掉电时计划执行的操作内容跟实际上执行的操作内容进行比较。在这个比较的基础上，程序就可以决定是撤销已做的操作还是继续完成已做的操作，或者是保持原样。

基础知识

—事务

事务：一致性/持续性

- 模拟Redo(此略)
（由于惰性写机制，手动模拟难预料）

- 模拟Undo
在事务提交之前结束服务器进程

```
-- ----[日志Undo]-----
BEGIN TRAN MakeACrash
    INSERT INTO Test VALUES(4, 0.4);
    INSERT INTO Test VALUES(5, 0.5);
    -- 未提交却强制使得脏页写入磁盘
    CHECKPOINT
    GO
    -- 查看一下可见此时确实有新行写入
    SELECT * FROM Test;

-- 然后杀死进程mssqlserver
-- 注意是服务端进程而不是客户端SSMS
-- 然后重新连接，查看Undo恢复后的写入情况
SELECT * FROM Test;
-- 查看数据库事务日志
SELECT * FROM sys.fn_dblog(NULL, NULL);
```

(此实验需要使用任务管理器等方式杀死进程)

基础知识

— FOR MySQL

本次实验用例的一些特性操作可能无法完成，
实验里可以不写
（如锁的忙等待、死锁、日志）

- DCL语法

START TRASACTION;（或BEGIN;）
COMMIT;
ROLLBACK;
SAVEPOINT <存档点名>;
ROLLBACK TO <存档点名>;
SET TRANSACTION ISOLATION LEVEL <隔离级别>;

```
-- 不处理异常(此处除0返回null不报异常)
SELECT 0/0;

-- 改为查询一个不存在的字段
SELECT nothing from t;

-- 加入异常处理
DELIMITER $$
CREATE PROCEDURE SelectException()
BEGIN
    DECLARE EXIT HANDLER FOR 1054 select 'ERROR Occur!';
    SELECT nothing from t;

END;$$
DELIMITER ;
CALL SelectException();
```

```
-- 一般的事务框架：加入错误处理
DELIMITER $$
CREATE PROCEDURE insertError()
BEGIN
    DECLARE EXIT HANDLER FOR 1062 select '主键冲突!';
    START TRANSACTION;
    INSERT INTO t VALUES(2,0.2);
    -- 主键冲突
    INSERT INTO t VALUES(2,0.2);
    INSERT INTO t VALUES(3,0.3);
END;$$
DELIMITER ;
CALL insertError();
```

```
-- 简单事务控制流
START TRANSACTION;
SELECT 'Begin T1';
INSERT INTO t VALUES(1,0.1);
SAVEPOINT S1;
SELECT 'SAVE S1';
INSERT INTO t VALUES(2,0.2);
SAVEPOINT  S2;
SELECT 'SAVE S2';
INSERT INTO t VALUES(3,0.3);
ROLLBACK TO  S1 ;
COMMIT;
```

更多问题
https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-transactions.html

相关参考

一般SQL语法：
http://www.w3school.com.cn/sql/index.asp

官方文档：
SQL Server: https://docs.microsoft.com/zh-cn/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-2017
MySQL: https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-transactions.html

多用搜索引擎：
https://cn.bing.com/
https://www.google.com/
https://www.baidu.com/

以及数据库课程PPT