동시성 제어 구현

배경 및 문제 발생 가능한 애플리케이션 기능

Phase 1에서 ER diagram을 통해 logical database를 설계할 때 real world와는 달리 고객(HOLDER)과 계좌(ACCOUNT)의 HAS (1:N 관계) relationship은 각각의 entity 모두에 대해 total participation이 되도록 설계하였습니다. 따라서 database 상에서 모든 고객 레코드와 모든 계좌 레코드는 HAS relationship에 참여하는 형태가 되어야합니다. 이러한 설계를 유지하기 위해 소유주(고객)가 없는 계좌는 당연히 생성할 수 없으며, 새로운 고객을 생성할 때는 반드시 하나의 계좌도 동시에 생성하는 시스템을 고려하였습니다.

문제 원인

고객(HOLDER) 테이블은 primary key로 H_ID를 가지며 이는 surrogate key 입니다. 계좌(ACCOUNT) 테이블의 primary key는 ACCOUNT_NUMBER이고 foreign key는 H_ID로 고객의 H_ID를 값으로 가집니다.

시스템 관리자가 생성할 고객의 이름, 성별, 주소, 국적, 전화번호, 신규 계좌 비밀번호를 애플리케이션의 폼에 입력해주면 아래의 연속된 두 쿼리를 통해 전달된 정보를 이용하여 신규 고객과 계좌를 생성합니다. 이때 신규 고객을 생성할 때 HODLER.H_ID를 `SELECT MAX(H_ID) + 1 FROM HOLDER` 서브쿼리로 set 하고 해당 고객의 신규 계좌를 생성할 때는 ACCOUNT.H_ID를 `SELECT MAX(H_ID) FROM HOLDER` 서브쿼리로 set 합니다. 따라서 연속된 두 쿼리 사이의 아주 짧은 시간에 다른 프로세스에서 새로운 고객을 추가하는 쿼리를 실행하게 되면 ACCOUNT.H_ID가 잘못된 HOLDER.H_ID로 설정되는 문제가 발생하게 됩니다.

```
INSERT INTO HOLDER (H_ID, NAME, SEX, ADDRESS, NATIONALITY, PHONE_NUMBER)
VALUES ((SELECT MAX(H_ID) + 1 FROM HOLDER), ${input_name}, ${input_sex},
${input_address}, ${input_nationality}, ${input_phone_number});

INSERT INTO ACCOUNT (ACCOUNT_NUMBER, BALANCE, PASSWORD, H_ID) VALUES
(${system_created_account_number}, 0, ${encrypted_input_password}, (SELECT
MAX(H_ID) FROM HOLDER));
```

해결 방법

위 문제를 해결하기 위해 저희는 두 쿼리를 하나의 transaction으로 처리하였으며 isolation level은 SERIALIZABLE을 사용하였습니다. 코드상에서는 createCustomerAndAccount.jsp의 L#45-46에서 autoCommit을 끈 후 `conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);` 를 추가하였습니다. 해당 시점에서 transaction을 시작하게 되고 두 쿼리를 성공적으로 수행하였다면 commit 할 수 있도록 하였습니다. 두 쿼리 중 하나가 실패하게 되면 `SQLException`을 catch하여 `stmt.cancel();`과 `pstmt.cancel();` 을 통해 취소하는 형태로 구현하였습니다.

시행 착오 및 깨달은 점

transaction 처리 구현 후 테스트하는 중에 html tag의 attribute를 잘못 작성하여 고객 성별을 null로 전달받는 문제가 있었습니다. 이를 파악하고 수정 후, 다시 테스트하니 이번에는 쿼리를 실행하지 못하고 계속 wait 하게 되는 문제가 발생하였습니다.

원인을 파악하기 위해 우선 해당 transaction이 실제로 다중 사용자 접속에 잘 대응하는지 확인하는 테스트를 하였습니다. 두 개의 터미널 A, B 에서 sqlplus로 oracle database에 접속하여 transaction을 실행해보았습니다. A에서 첫번째 쿼리까지 실행한 후 B에서도 동일하게 첫 번째 쿼리까지 실행하니 동일한 테이블에 INSERT를 수행하려고 하기 때문에 쿼리 결과를 마치지 못하고 wait 하게 되는 상황이 생겼습니다. A에서 두 번쨔 쿼리까지 실행 후 commit을 하니 B의 첫 번째 쿼리가 수행되는 것을 확인하였습니다. 이를 통해 JDBC의 앞선 테스트에서 에러가 발생하면서 해당 프로세스에서 실행한 transaction이 종료되지 않고 그대로 남아있는 상태에서, 수정 후 다시 테스트를 하니 무한정 기다리게 되는 문제가 발생한 것이라고 추측하였습니다. database에 system user로 접속하여 `select ses_addr, name from v$transaction;`를 통해 진행중인 transaction을 여러 번 확인하니, 동일한 transaction이 계속 존재하는 것을 확인하였습니다.

따라서 모든 transaction을 종료하고, catch {...} 부분에 `stmt.cancel();`과 `pstmt.cancel();` 을 추가한 뒤 의도적으로 에러를 낸 후, 다시 한 번 테스트해보니 정상 동작하는 것을 확인하였습니다. 이를 통해 JDBC에서 transaction 처리 시 commit 뿐만아니라 에러 발생시 모든 statement를 cancel 시키는 것도 매우 중요하다는 것을 깨달았습니다.

추가 사항

```
SELECT MAX(H_ID) FROM HOLDER;

INSERT INTO HOLDER (H_ID, NAME, SEX, ADDRESS, NATIONALITY, PHONE_NUMBER)
VALUES (${max_h_id_var + 1}, ${input_name}, ${input_sex},
${input_address}, ${input_nationality}, ${input_phone_number});

INSERT INTO ACCOUNT (ACCOUNT_NUMBER, BALANCE, PASSWORD, H_ID) VALUES
(max_h_id_var + 1, 0, ${encrypted_input_password}, (SELECT MAX(H_ID) FROM
HOLDER));
```

추가로 JDBC에서 위의 세 쿼리로 transaction 처리없이 동일하게 고객과 계좌 정보를 생성할 수 있습니다. 첫 번째 쿼리를 통해 H_ID의 최댓값을 `max_h_id_var`에 저장해두고 이를 이용하여 아래 두 쿼리를 수행하게 됩니다. 하지만 transaction으로 처리하는 것이 각 두 쿼리에 서브 쿼리가 있음에도 불구하고 실제 database I/O는 두 번이므로 성능은 더 좋을 것이라고 예상됩니다.