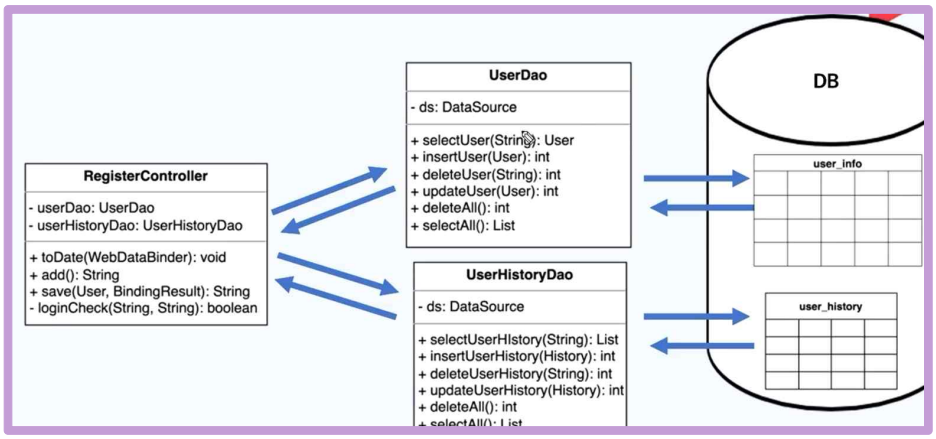
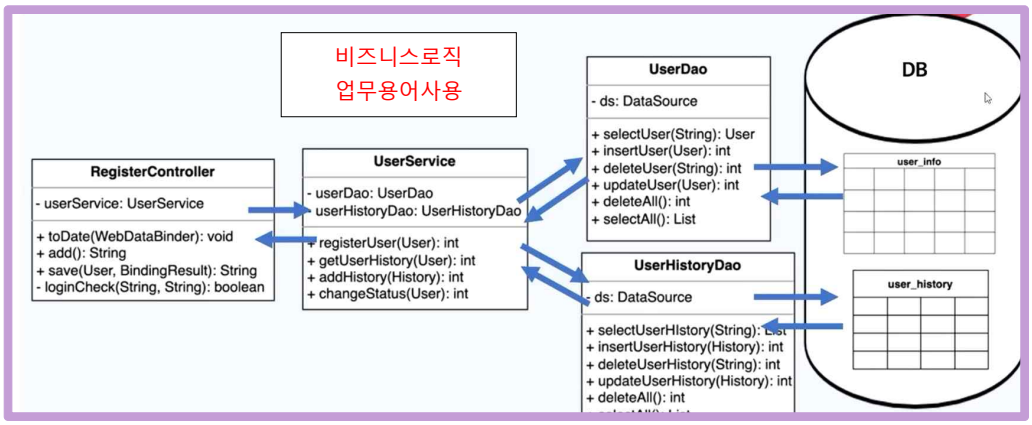
트랜잭션 사용하기



service 만들기

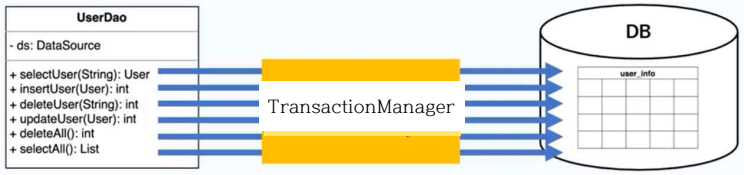


트랜잭션

하나의 논리적인 작업단위
분리될 수 없는 하나의 묶음의 단위

. TransactionManager란?

같은 Tx내에서 같은 Connection을 사용할 수 있게 관리



service

```
public void insertWithTx() throws Exception {
    PlatformTransactionManager tm = new DataSourceTransactionManager(ds);
    TransactionStatus status = tm.getTransaction(new DefaultTransactionDefinition());
    // Tx 시작
    try {
        a1Dao.insert(1,100);
        a1Dao.insert(1,200);
        tm.commit(status);   // Tx 끝 - 성공(커밋)
    } catch(Exception ex) {
        tm.rollback(status); // Tx 끝 - 실패(롤백)
    }
}
```

DAO에서 Connection을 얻거나 반환할 때 DataSourceUtils를 사용해야

```
conn = ds.getConnection();
// ...
try { if(conn!=null)  conn.close();  }
catch (SQLException e) { e.printStackTrace();}
```

```
conn= DataSourceUtils.getConnection(ds);
// ...
DataSourceUtils.releaseConnection(conn, ds)
```

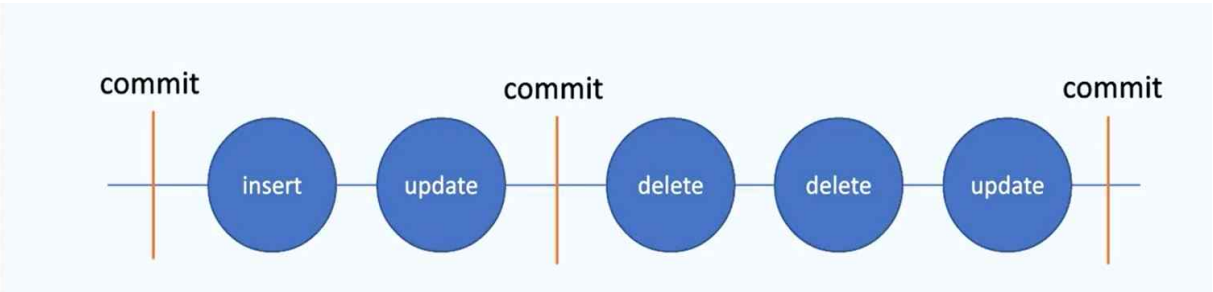
Transaction의 속성 – ACID

원자성(Atomicity) – 나눌 수 없는 하나의 작업으로 다뤄져야 한다
일관성(Consistency) – Tx 수행전과 후가 일관된 상태를 유지해야 한다
고립성(Isolation) – 각 Tx는 독립적으로 수행되어야 한다.
영속성(Durability) – 성공한 Tx의 결과는 유지되어야 한다.

◎ Tx의 isolation level

READ UNCOMMITED – 커밋되지 않은 데이터도 읽기 가능
READ COMMITED – 커밋된 데이터만 읽기 가능
REPEATABLE READ – Tx이 시작된 이후 변경은 무시됨 (기본 임)
SERIALIZABLE – 한번에 하나의 Tx만 독립적으로 수행

■ Commit과 Rollback
◎ 커밋(commit) – 작업 내용을 DB에 영구적으로 저장
◎ 롤백(rollback) – 최근 변경사항을 취소 (마지막 커밋으로 복귀)



◎ 자동커밋 : 명령 실행 후 , 자동으로 커밋이 수행 (rollback 불가)
◎ 수동커밋 : 명령 실행 후 , 명시적으로 commit또는 rollback을 입력
Transaction처리를 위해서 수동커밋으로 설정되어야 함

```java
public class TestDB {

	public static void main(String[] args)	{

			String driver = "oracle.jdbc.driver.OracleDriver" ;
			String url="jdbc:oracle:thin:@localhost:1521:xe";
			String user="system";
			String password="1234";

		Connection conn=null;

			try {
				Class.forName(driver);
				conn = DriverManager.getConnection(url, user, password);
			} catch (ClassNotFoundException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (SQLException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}


			try {
				// transaction  : 같은 connection객체일경우만 가능
				// 자동커밋, 수동커밋

				String sql1= " insert into testdb values( 'a1', 'OOO')";
				PreparedStatement pst  = conn.prepareStatement(sql1);
				int result  =pst.executeUpdate();

				String sql2= " insert into testdb values( 'a1', 'OOO')";
				PreparedStatement pst2  = conn.prepareStatement(sql2);
				int result2  =pst2.executeUpdate();

				// 첫번째의 데이터는 정상으로 insert 됨
				// 두 경우가 성공할 경우만 처리되고 아니라면 모두 취소하고 싶다면
				// 두 개의 쿼리가 하나의 작업단위 트랜잭션으로 묶을 수 있다.

				// 트랜잭션 처리를 위한 필요사항
				// 같은 커넥션을 얻어와야 한다
				// 수동커밋으로 되어 있어야 함
				// 둘 다 정상일때 commit,예외발생했다면  rollback 처리한다


			} catch (SQLException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}



	}
}
```

```
public class TestDB2 {

	public static void main(String[] args)	{

		String driver = "oracle.jdbc.driver.OracleDriver" ;
		String url="jdbc:oracle:thin:@localhost:1521:xe";
		String user="system";
		String password="1234";


		Connection conn=null;

			try {
				Class.forName(driver);
				conn = DriverManager.getConnection(url, user, password);
			} catch (ClassNotFoundException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (SQLException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}


			try {
				//
				conn.setAutoCommit(false);
				String sql1= " insert into testdb values( 'a1', 'OOO')";
				PreparedStatement pst  = conn.prepareStatement(sql1);
				int result  =pst.executeUpdate();

				String sql2= " insert into testdb values( 'a1', 'OOO')";
				PreparedStatement pst2  = conn.prepareStatement(sql2);
				int result2  =pst2.executeUpdate();
				conn.commit();

			} catch (SQLException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
				try {
					conn.rollback();
				} catch (SQLException e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}
			}



	}

}
```

```java
@Component
public class MemberDao{
    @Autowired
    DataSource ds;

    final int FAIL = 0;

    public int insertUser(Member member) throws SQLException  {
        int rowCnt = FAIL;
        Connection conn = null;
        PreparedStatement pstmt = null;

        String sql = "insert into member_info values (?, ?, ?, ?,?,?, sysdate) ";

        try {
            conn= DataSourceUtils.getConnection(ds);          //변경포인트, 같은커넥션을 얻어오기 위한 부분
            conn.setAutoCommit(false);
         System.out.println( conn);
            pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, member.getId());
            pstmt.setString(2, member.getPwd());
            pstmt.setString(3, member.getName());
            pstmt.setString(4, member.getEmail());
            pstmt.setString(5, member.getBirth());
            pstmt.setString(6, member.getSns());
            rowCnt  = pstmt.executeUpdate(); //
        } catch (SQLException e) {
            e.printStackTrace();
            throw e;     //예외 되던지기
        } finally {
         close(pstmt);
         DataSourceUtils.releaseConnection(conn, ds);     // 변경포인트
        }
        return rowCnt;
    }



    private void close(AutoCloseable... acs) {
        for(AutoCloseable ac :acs)
            try { if(ac!=null) ac.close(); } catch(Exception e) { e.printStackTrace(); }
    }


}
```

```java
@Component
public class MemberService  {

	@Autowired
	MemberDao  dao;

	@Autowired
	DataSource ds;


	// 스프링을 이용하여  트랜잭션 관리
	 public void insertA1WithTx(Member member) throws Exception {
		PlatformTransactionManager tm = new DataSourceTransactionManager(ds);
		TransactionStatus status = tm.getTransaction(  new DefaultTransactionDefinition()  );

		try {
			dao.insertUser(member);
			dao.insertUser(member);
			tm.commit(status);
		} catch (Exception e) {
		    e.printStackTrace();
		    tm.rollback(status);
		    throw e;
		}
	   }


}
```

```java
@Component
public class MemberService2  {

	@Autowired
	MemberDao  dao;

	@Autowired
	PlatformTransactionManager tm;


	// 스프링을 이용하여  트랜잭션 관리
	 public void insertA1WithTx(Member member) throws Exception {
	   //    PlatformTransactionManager tm = new DataSourceTransactionManager(ds);
		TransactionStatus status = tm.getTransaction(  new DefaultTransactionDefinition()  );

		try {
		     dao.insertUser(member);
		     dao.insertUser(member);
		     tm.commit(status);
		} catch (Exception e) {
		    e.printStackTrace();
		    tm.rollback(status);
		    throw e;
		}
	   }


}
```

```
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"></property>
            <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"></property>
            <property name="username" value="System"></property>
            <property name="password" value="1234"></property>
    </bean>


    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource"/>
    </bean>
```

pomxml dependency 추가

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${org.springframework-version}</version>
    </dependency>
            <dependency>
        <groupId>com.oracle.database.jdbc</groupId>
        <artifactId>ojdbc8</artifactId>
        <version>21.7.0.0</version>
    </dependency>
```

```java
@Controller
public class TranTestController2 {

	@Autowired
	MemberService2  service;

	//응답코드 수정가능하다 , 서버에서 응답하면 기본적으로 성공으로 간주한다.
	@ResponseStatus(HttpStatus.SERVICE_UNAVAILABLE)
	@ExceptionHandler(Exception.class)
	public String catcher(  Model model, Exception ex) {
	//	model.addAttribute("ex", ex);
		model.addAttribute("ex", "데이터베이스오류입니다  서버관리자에게 문의하세요 !");
		return "err";
	}



	@RequestMapping("/tranTest")
	public void insert1(  ) throws Exception   {
			Member member= new Member();
			member.setId("t2");
		member.setPwd("1234");
		member.setName("hong");
		member.setEmail("email.com");
		member.setBirth("2023-01-01");
		member.setSns("net");
		member.setReg_date(new Date());
			service.insertA1WithTx(member);

	}


}
```