게임서버프로그래밍

2019년 2학기

한국산업기술대학교
게임공학부

정내훈

7장 데이터베이스 기초

7.1 | 플레이어의 정보 저장

- 각 플레이어의 로컬 컴퓨터에 플레이어 정보를 저장하면?

  1. 플레이어가 자리를 옮겼을 때 자기가 플레이하던 정보를 이어서 할 수가 없다.
  2. 플레이어가 해킹을 할 줄 안다면, 자기가 컴퓨터에 저장된 플레이 정보를 조작할 수 있다.

- 플레이어 정보를 서버에 저장하는 방법 : 파일과 데이터베이스

  상용으로 서비스되는 게임들은 대부분 플레이어 정보를 데이터베이스에 저장한다.

  예를 들어 Microsoft SQL Server를 데이터베이스로 사용하는 경우 여러 플레이어의 정보는 덩치 큰 파일 몇개에 모두 저장된다.

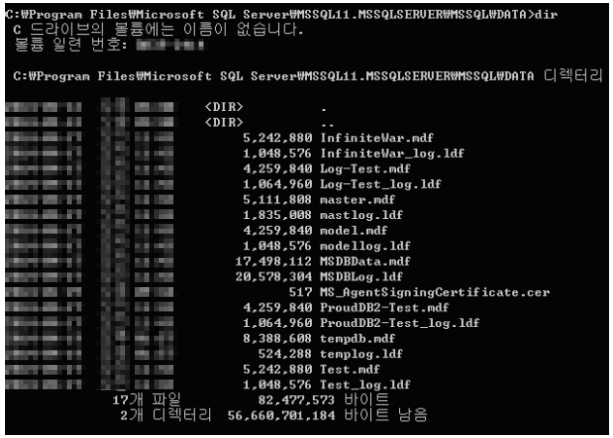


그림 7-2 SQL Server라는 데이터베이스는 mdf file에 데이터를 저장

7.1 | 플레이어의 정보 저장

표 7-1 파일과 데이터베이스 비교

비교 조건	단순 파일	데이터베이스	비고
소프트웨어 비용	없다.	없거나 높다.	오픈 소스 제품은 제한적으로 무료다.
저장 및 로딩 속도	빠르다.	느리다.	데이터베이스도 결국 파일 시스템을 사용한다.
데이터 관리, 분석 속도	느리다.	빠르다.	데이터베이스는 빠른 검색을 위한 인덱스 기능 이 있다.
데이터 백업 및 복원 기능	없다.	있다.	-
원자성(데이터 2개 이상을 전부 변경하고자 할 때 전부 혹은 전무(全無) 처리가 되게 한다.)	불가능하다.	가능하다.	데이터베이스의 트랜잭션 기능이다.
일관성(잘못된 상태의 데이터를 원천 봉쇄한다.)	없다.	있다.	데이터베이스의 제약(constraints) 기능이다.
고립성(경쟁 상태부터 자유롭게 해 주는 기능이다.)	없다.	있다.	데이터베이스의 락 기능이다.
지속성(장애 직전의 상태로 복구 가능 한지에 관한 것이다.)	없다.	있다.	데이터베이스의 로그 버퍼 기능이다.

데이터베이스에서는 다음과 같은 구체적인 상황에서도 빠르게, 다각적으로 데이터를 검색할 수 있어 편리하다.

- 레벨 20~23 사이의 엘프족 남성 플레이어만 찾으시오.
- '+8 집행검'을 가진 플레이어만 찾으시오.
- 플레이어 'Hong Gil Dong'이 가진 화살류 아이템의 개수를 찾으시오.

7.2 | 데이터베이스 사용

- 데이터베이스 엔진

  많이 쓰는 것은 Microsoft SQL Server나 Oracle MySQL이다.

  SQL Server Express는 링크 혹은 구글 검색에서 SQL Server Express로 검색한 후 나타나는 Microsoft 웹 사이트에서 설치한다. 무료.

  SQL Server Management Studio는 링크 혹은 구글 검색에서 SQL Server Management Studio로 검색하여 설치할 수 있다.

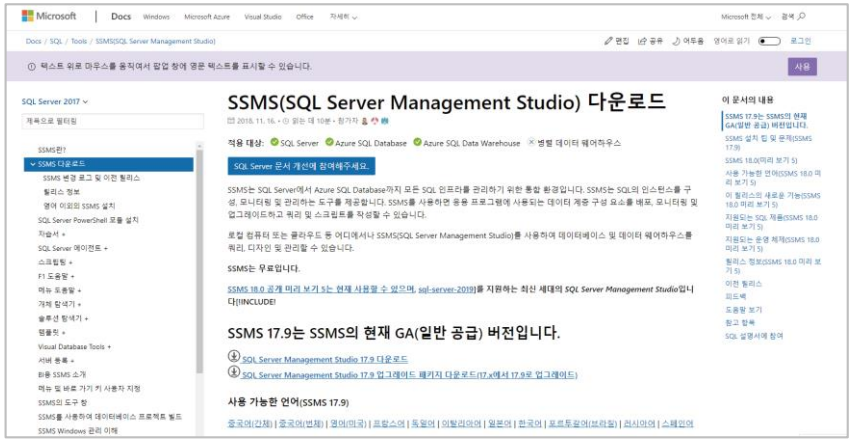


그림 7-3 Microsoft SQL Server 내려받기 페이지 그림 7-4 SQL Server Management Studio 내려받기 페이지

내려 받아서 설치 하시오! 실습 예정

7.3 | 데이터베이스의 데이터 구성

- 데이터베이스 구성

  데이터베이스에서 다루는 데이터는 표(table) 형태의 집합.

  데이터베이스 인스턴스(instance) : 테이블의 집합으로 데이터베이스가 다루는 가장 큰 단위의 데이터 집합

  데이터베이스에서는 행 단위로 데이터를 넣거나 뺄 수 있으며 이 행을 레코드(record)라고 한다.

  레코드 안에는 표의 열이 있는데, 이를 필드(f ield)라고 한다.

  필드는 이름 말고도 타입이라는 것을 추가로 가지고 있으며 타입은 다음 중 하나이다.
  · 정수 | 소수 | 문자열 | 날짜, 시간 | 그 외(바이너리, xml, guid 등)

데이터베이스: GameDB

UserAccount

이메일	패스워드	생년월일
john@mymail.com	@#$@^#$@%$#^	66/6/6
prada@moma.com	&%@#$#&##@$%#	77/7/7

데이터베이스: LogDB

BuyItem

날짜	사용자 ID	아이템
99/99/99 55-55-55	john	Sword
00/00/00 66-66-66	doe	Banana

그림 7-5
플레이어 정보를 저장하는 게임 데이터베이스와
게임 플레이 내역을 저장하는 로그 데이터베이스

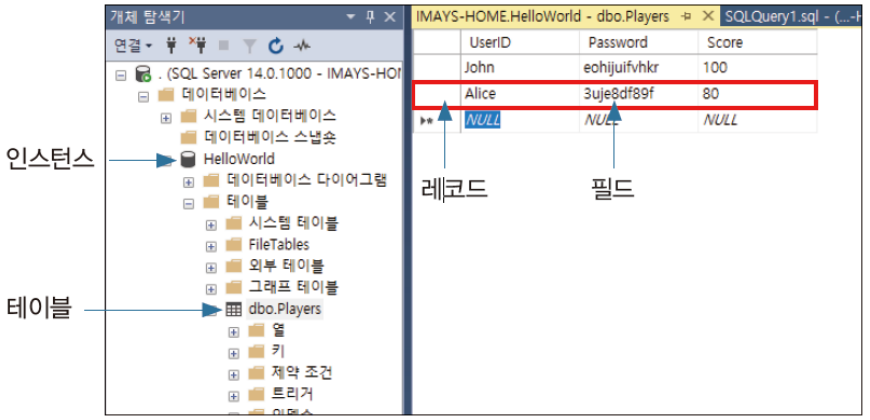


그림 7-6
인스턴스, 테이블, 레코드, 필드

7.4 | 데이터베이스 시작

- 데이터베이스 인스턴스와 테이블 만들기

  SQL Server Management Studio를 열고데이터베이스 항목에서 마우스 오른쪽 버튼을 눌러 새 인스턴스를 만든다.

  새 인스턴스 아래에서 마우스 오른쪽 버튼을 눌러 새 테이블을 만든다.

  원하는 필드를 모두 추가한 후에는 저장 버튼을 누르고 원하는 테이블 이름을 입력한다.

  테이블에서 마우스 오른쪽 버튼을 눌러 테이블 디자인을 선택한다.(만든 테이블의 필드를 수정할 경우)

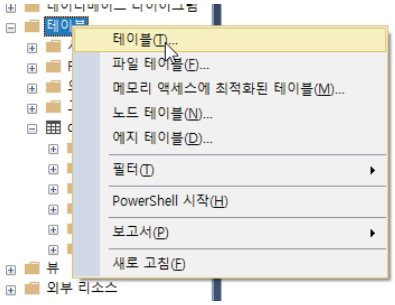


그림 7-7
새 데이터베이스 인스턴스 만들기

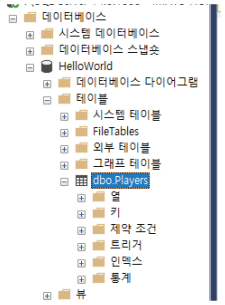


그림 7-8
새 테이블 생성

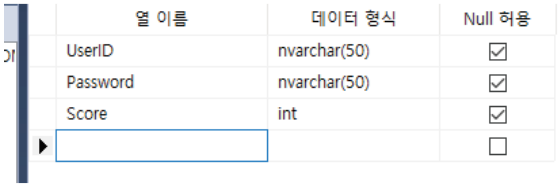


그림 7-9
테이블 디자인 화면

7.4 | 데이터베이스 시작

표 7-2 선택할 수 있는 필드 종류

종류	설명	특징	예시
int	정수	-	345
float	부동소수점	-	123.45
char(n)	길이 n의 문자열	입력한 문자열이 짧으면 뒤에 빈칸이 강제로 채워진다.	"a "
varchar(n)	최대 길이 n의 문자열	-	"a"
nvarchar, nchar	n이 붙으면 유니코드	UTF-16	"こんにちは你好안녕"
text	길이 무제한 문자열	읽기/쓰기 속도가 느리다.	-
binary	길이 무제한 바이너리	상동	11 22 3F 4B BA 00 CB
unique identifier	GUID 또는 UUID	MySQL에서는 사용 불가	{5 8 2 6 9 1 1 5 -1 F D C -4 6 1 7 - BC0F-7F782197270F} 이 중에서

1. int와 f loat는 각각 정수, 소수를 저장한다.
2. 데이터베이스에 저장되는 필드의 글자 수에는 제한이 있습니다. 이 제한은 char, nchar, varchar, nvarchar 뒤에 붙는 숫자로, 글자 수가 넘어가면 글자가 잘리거나 오류가 발생한다. text는 글자 수에 제한이 없지만 액세스 속도가 더 느리다.
3. 데이터베이스의 필드는 null(값 자체가 없음)을 저장할 수도 있다.

7.5 | SQL 질의 구문

- CRUD(Create, Read, Update, Delete)
  레코드 추가하기, 읽기, 변경(업데이트), 삭제하기

```
insert into table1 (a,b,c) values (1,2,3)
```
새 레코드 삽입하기
테이블 table1에 필드 a = 1, b = 2, c = 3이 들어 있는 새 레코드를 하나 추가함

```
select a,b,c from table1 where a=1
```
테이블 table1의 모든 레코드를 얻으려면 select로 시작하는 구문을 쓴다

```
select * from table1 where a=1
```
table1에 있는 레코드 중 필드 a가 1인 레코드를 찾는다.
찾은 레코드의 필드 a, b, c를 얻는다.

```
update table1 set b=2 where a=1
```
table1에서 필드 a가 1인 레코드를 찾는다.
그리고 그것의 모든 필드를 얻는다.

```
delete from table1 where a=1
```
table1에 있는 레코드 중 필드 a가 1인 레코드를 찾아 그것의 필드 b를 2로 변경한다.

이 문장을 실행하면 table1에 있는 레코드 중 필드 a가 1인 것을 모두 삭제한다.

7.6 | 인덱스와 키

- 데이터베이스에서 인덱스 기능 사용하기.

  인덱스는 필드 단위로 설정할 수 있다.

  인덱스를 설정해 놓으면 특정 조건에서 어마어마하게 빠른 속도로 원하는 레코드를 찾을 수 있다.

  인덱스는 select 구문, 즉 검색만을 위한 것이 아니다. 기존 레코드를 변경하거나 레코드를 삭제할 때도 인덱스는 빨리 찾는 데 큰 도움이 된다.

```
delete from table1 where a=100
```

table1에 있는 모든 레코드를 다 순회하면서 a = 100인 것을 찾지 않고, 인덱스를 이용해서 a = 100인 것을 매우 적은 횟수의 순회만으로도 다 찾아내서 지운다.

- 중복된 값을 방지하는 용도로도 사용된다.

```
insert into table1 (a,b,c) values (100,200,300)
insert into table1 (a,b,c) values (100,210,310)
```

table1의 필드 a에 유니크 인덱스가 설정되면, 다음과 같이 a = 100인 레코드를 여럿 추가할 때 두 번째 추가 질의 구문에서 실패가 발생할 것이다.

- 인덱스를 추가하는 방법(SQL Management Studio에서)

그림 7-11 인덱스를 추가하는 화면

7.6 | 인덱스와 키

- 프라이머리 키(primary key)

  테이블 필드에서 마우스오른쪽 버튼을 눌러 프라이머리 키를 설정

그림 7-1 프라이머리 키를 설정하는 중

- 프라이머리 키의 성질

  한 테이블에 하나만 추가할 수 있다.

  중복을 허락하지 않는다. 즉, 한 테이블에 값이 같은 코드가2 개 이상 들어갈 수 없다.

  필드 값은 null이 허락되지 않는다.

7.7 | 플레이어 정보를 데이터베이스에 저장하는 방법 1

- 구조체형 데이터를 테이블에 저장하는 방법
  1. 플레이어 데이터 전체를 문서 형태로 만들어서 테이블에 넣는다.
  2. 플레이어 데이터를 구성하는 트리 노드 각각을 테이블에 넣는다.

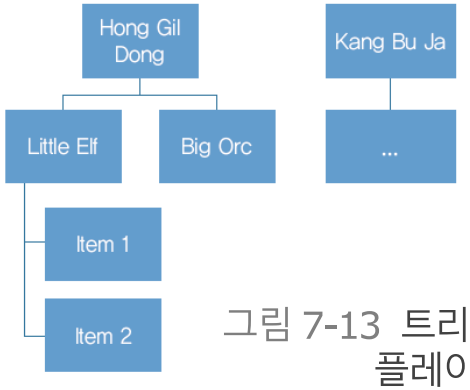


그림 7-13 트리 구조로 표현된 플레이어 데이터

- JSON(JavaScript Object Notation) 문서 형태

```
{
        "ID": "Hong Gil Dong",
        "email": "gildong@foofoomail.com",
        "password": "xxiuhwdqwddwdwafd",
        "Characters": [
                {
                        "ID": "Little Elf",
                        "Gender": "Female",
                        "Level": 35,
                        "Items": [
                                {
                                        "Type": 123,
                                        "Amount": 1
                                }
                        ]
                },
                {
                        "ID": "Big Orc",
                        "Gender": "Male",
                        "Level": 23,
                        "Items": []
                }
        ]
}
```

7.7 | 플레이어 정보를 데이터베이스에 저장하는 방법 1

- 한 줄로 붙이면.


```
{"ID": "Hong Gil Dong", "email": "gildong@foofoomail.com", "password":
"xxiuhwdqwddwdwafd", "Characters": [{"ID": "Little Elf", "Gender": "Female", "Level":
35, "Items": [{"Type": 123, "Amount": 1}]}, {"ID": "Big Orc", "Gender": "Male",
"Level": 23, "Items": []}]}
```


표 7-3 테이블의 필드 1개에 플레이어 모든 정보를 JSON으로 담은 모습
문자열 길이가 제한되지 않는 text 타입의 필드를 사용.

ID	문서
Hong Gil Dong	{"ID": "Hong Gil Dong", "email": "gildong@foofoomail.com", "password": "xxiuhwdqwddwdwafd", "Characters": [{"ID": "Little Elf", "Gender": "Female", "Level": 35, "Items": [{"Type": 123, "Amount": 1}]}, {"ID": "Big Orc", "Gender": "Male", "Level": 23, "Items": []}]}
Kang Bu Ja	{"ID": … }

7.7 | 플레이어 정보를 데이터베이스에 저장하는 방법 1

- JSON 문서를 XML 문서로 변환하면

```
<?xml version="1.0" encoding="UTF-8"?>
	<ID>Hong Gil Dong</ID>
	<email>gildong@foofoomail.com</email>
	<password>xxiuhwdqwddwdwafd</password>
	<Characters>
		<ID>Little Elf</ID>
		<Gender>Female</Gender>
		<Level>35</Level>
		<Items>
			<Type>123</Type>
			<Amount>1</Amount>
			</Items>
	</Characters>
	<Characters>
		<ID>Big Orc</ID>
		<Gender>Male</Gender>
		<Level>23</Level>
	</Characters>
```

(JSON이나 XML중)둘 중 더 편한 것을 쓰면 된다.

여러분이 사용하는 데이터베이스에서는 JSON이나 XML 중 하나 이상을 지원할 것이다.

어느 것도 지원하지 않는 데이터베이스를 사용하고 있을 때는 글자수에 제한이 없는 text 필드를 사용.

7.8 | 플레이어 정보를 데이터베이스에 저장하는 방법 2

- 플레이어의 데이터 정보를 여러 테이블에 나누어 저장하기

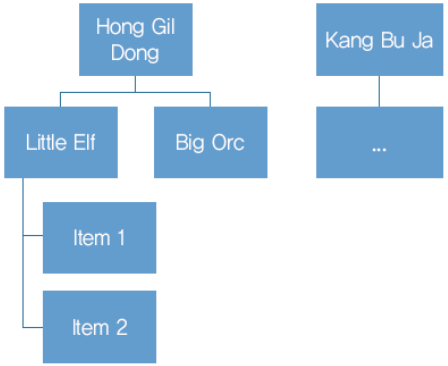


그림 7-14 트리 구조로 표현된 플레이어 데이터

ID	Password	Etc
Hong Gil Dong	xxx	...
Kang Bu Ja	xxx	...

표 7-4 플레이어 정보가 담긴 UserAccount 테이블

플레이어 캐릭터를 테이블에 저장할 때 플레이어 캐릭터의 고유 이름으로 ID뿐만 아니라 "이 플레이어 캐릭터가 누구의 소유인가?"도 저장

ID	OwnerUserAccountID	Etc
Little Elf	Hong Gil Dong	...
Big Orc	Hong Gil Dong	...

표 7-5
Character 테이블로 각 캐릭터별 소유주가 있음

7.8 | 플레이어 정보를 데이터베이스에 저장하는 방법 2

아이템 테이블을 정의하고 이름을 Item이라고 한다.. Item에는 각각의 아이템 개체가 레코드로들어가게 한다.
각 아이템은 소유자로 '플레이어 캐릭터'인 Character의 ID를 가리키게 한다.
OwnerCharacterID라는 필드를 추가하고 이 필드는 Character 테이블의 ID를 가리키는 외래 키가 되게 한다.

ID	OwnerUserAccountID	Etc
xxxx	Little Elf	…
xxxx	Little Elf	…

표 7-6
아이템 개체가 들어 있는 Item 테이블

지금까지 설명한 것을 한데 모아 보면,
트리의 각 노드는 테이블의 레코드가 된다.
각 노드의 소유자, 즉 부모 노드는 외래 키가 된다.

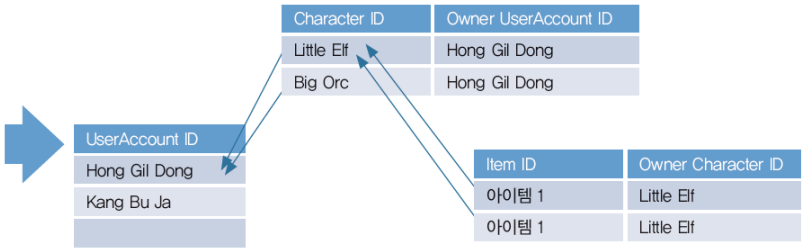


그림 7-15
플레이어 데이터가
여러 테이블에 걸쳐 저장된 모습

7.8 | 플레이어 정보를 데이터베이스에 저장하는 방법 2

- UML을 이용해서 테이블 관계를 표현해 보기.

  두 테이블 A·B가 있다.

  테이블 A는 프라이머리 키 X를 가지며 테이블 B는 테이블 A의 프라이머리 키 X를 가리키는 외래 키 OwnerX를 가진다.

  테이블 B의 레코드는 테이블 A의 어떤 레코드를 가리킬 것이다.

  테이블 B의 두 레코드는 테이블 A의 한 레코드를 가리킬 수도 있겠다.

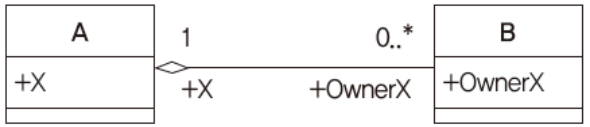
테이블 A와 테이블 B를 표현

그림 7-16 테이블 A와 테이블 B를 UML로 표현

두 테이블 관계를 표시

마름모 표시 : 소유자 쪽이 마름모 표시를 갖고 있다는 의미

그림 7-17 테이블 A와 테이블 B는 관계를 마름모로 표시



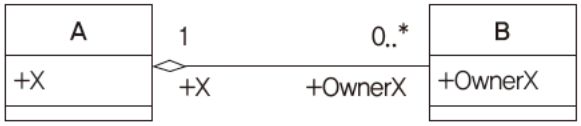
테이블에 어느필드가 오른쪽 테이블의 어느 필드에 대응하는지 표현

그림 7-18 테이블 A의 무슨 필드가 테이블 B의 무슨 필드에 대응하는지를 표시

두 테이블의 레코드 간 개수 관계를 표현

카디널리티(cardinality)나 멀티플리시티(multiplicity)라고 한다.

7.8 | 플레이어 정보를 데이터베이스에 저장하는 방법 2



두 테이블의 레코드 간 개수 관계를 표현
카디널리티(cardinality)나 멀티플리시티(multiplicity)라고 한다.

그림 7-19 테이블 A의 레코드 1개는 테이블 B의 레코드 0~여럿과 연관됨을 표시

- 플레이어 정보를 저장하는 데이터베이스 테이블을 정의해 보면.

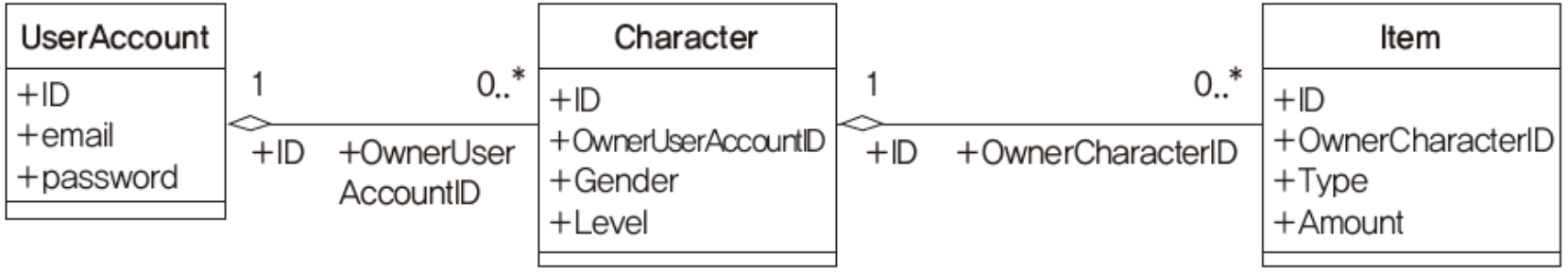


그림 7-20 플레이어 정보를 저장하는 테이블들의 최종 표현

7.9 | 질의 구문 실행

- 플레이어 데이터를 읽고 쓰는 방법을 만들기.

select ID, password from UserAccount where ID='Hong Gil Dong'

플레이어 ID와 비밀번호를 읽으려면 where 구문에서 플레이어 ID를 넣게 하고 password 필드를 가져온다.

select ID from Character where OwnerUserAccountID='Hong Gil Dong'

where 구문에 '소유주 플레이어 캐릭터'의 ID를 넣고 ID만 골라서 가져온다. (select 구문을 사용.)

select * from Character where ID='Little Elf'

플레이어가 가진 캐릭터 하나에 대한 모든 정보를 얻고 싶다면 where 구문에 캐릭터 ID를 넣는다. 모든 필드를 가져와야 하므로 구문에 *를 사용.

select * from Item where OwnerCharacterID='Little Elf'

플레이어 캐릭터 하나가 가진 아이템을 모두 읽으려면 where 구문에 캐릭터 ID를 넣는다.

7.9 | 질의 구문 실행

- 플레이어 데이터를 읽고 쓰는 방법을 만들기.

update Character set Exp=2943 where ID='Little Elf'

캐릭터가 몹 사냥 후 경험치를 획득하면 Character에서 해당하는 레코드를 찾고, 그 레코드의 필드를 업데이트한다.(update 구문 사용)

insert into Item (ID,OwnerCharacterID,Type,Amount) values (37485,'Little Elf',34,3)

예를 들어 플레이어 캐릭터가 타입 34인 아이템을 3개 획득하면 insert 구문을 사용하여 Item 테이블에 새 레코드를 추가.

update Item set Amount=2 where ID=37485

캐릭터가 아이템 3개 중 1개를 소모하여 2개가 남는다. 그러면 update 구문을 사용.

delete from Item where ID=37485

캐릭터가 아이템을 버리면 아이템 레코드를 지워야 함. delete 구문을 사용.

7.9 | 질의 구문 실행

- 게임 서버가 데이터베이스에 질의 구문을 자주 던지는 것은 비효율적이다.

  질의 구문 던지기 후 결과 받기 과정이 수행되는 동안디 바이스 타임(device time)이 발생.

  게임 서버와 데이터베이스 간 네트워크 레이턴시를 다 모으면 꽤 긴 시간이 됨.

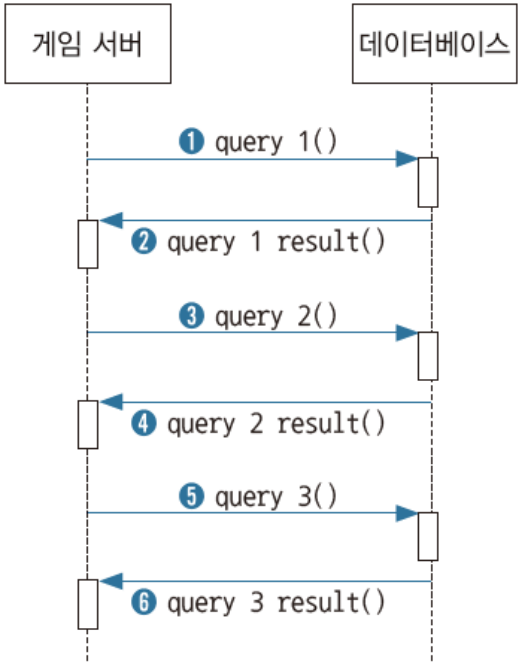


그림 7-21 데이터베이스에 질의 구문을 던지고 받는 과정이 지속

```
for (i = 0 to 100)
{
        select xxx from yyy where zzz=i
}
```

이 경우 게임 서버는 데이터베이스에 select 구문을 100번 던져야 한다.

그러면 게임 서버와 데이터베이스 사이의 대화는 100회 오가게 된다.

7.9 | 질의 구문 실행

- 저장 프로시저(stored procedure)

  데이터베이스 안에서 직접 실행되는 스크립트 프로그램

```
create procedure LoadCharacterAndItem
        @ID nvarchar(50) -- 저장 프로시저의 입력 매개변수
as
begin -- 여기서부터 저장 프로시저 루틴의 시작
        select * from Character where ID=@ID
        select * from Item where OwnerCharacterID=@ID
end -- 루틴의 끝
```

저장 프로시저가 데이터베이스에 추가됨.

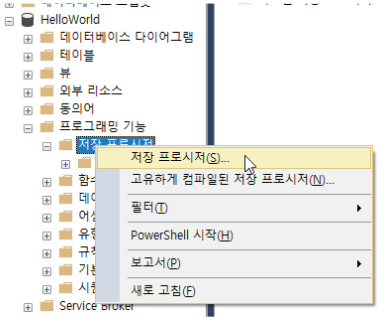


그림 7-22
SQL Server Management Studio에서 저장 프로시저 추가

EXEC LoadCharacterAndItem @ID='xxx'

EXEC 다음에 저장 프로시저 이름과 입력할 매개변수를 나열한다. 그러면 데이터베이스는 저장 프로시저 안에 있는 많은 구문을 알아서 다 실행함.

7.9 | 질의 구문 실행

- 7.9.1 트랜잭션_계좌 이체 과정을 프로그램으로 만들 때

```
update UserAccount set Money=Money+100 where ID='Kang Bu Ja'
update UserAccount set Money=Money-100 where ID='Hong Gil Dong'
```

Hong Gil Dong이 Kang Bu Ja에게 100원을 계좌 이체하려면 다음 처리를 해야 함.

```
begin transaction
update UserAccount set Money=Money+100 where ID='Kang Bu Ja'
update UserAccount set Money=Money-100 where ID='Hong Gil Dong'
commit
```

begin transaction 구문을 먼저 실행 (트랜잭션의시작), 두 구문을 실행한 후 커밋(commit)을 실행.
이렇게 하면 두 update 구문의 실행 결과는 데이터베이스에 영구적으로 남는다.

7.9 | 질의 구문 실행

- SQL Server Management Studio에서 새 질의 구문 창을 또 띄우기.

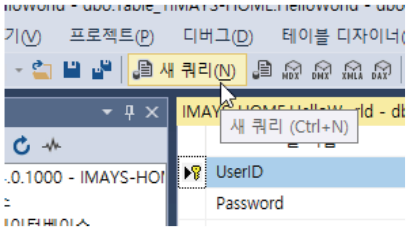


그림 7-23
새 질의 구문 창 열기

```
begin transaction
update UserAccount set Money=Money+100 where ID='Kang Bu Ja'
update UserAccount set Money=Money-100 where ID='Hong Gil Dong'
```

첫 번째 질의 구문 창에서 실행

```
select Money from UserAccount where ID='Kang Bu Ja'
```

두 번째 질의 구문 창에서 실행

Money 값이 무엇이 될지 최종 판단을 할 수가 없기 때문에 두 번째 질의 구문 창의 질의 구문은 실행 결과가 나오지 않고 무한 대기 상태가 된다.

7.9 | 질의 구문 실행

commit transaction 혹은 rollback

첫 번째 질의 구문 창에서 실행. 이제서야 두 번째 질의 구문 창에서 무한 대기 상태가 끝나고 결과가 나온다.

- 알 수 있는 것.

1. 트랜잭션을 걸고 액세스하는 레코드는 멀티스레드 프로그래밍에서 뮤텍스(1장 참고)를 잠금한 것과 비슷하다.
2. 트랜잭션을 걸고 액세스하는 레코드는 다른 질의 구문 세션에서 액세스하면 앞서 트랜잭션이 완료되거나 취소될 때까지 블로킹이 걸림.

- 두 질의 구문 실행 프로세스가 거의 동시에 실행되면서 락이 걸리면.

1. 질의 구문 1: 레코드 A
2. 질의 구문 2: 레코드 B
3. 질의 구문 1: 레코드 B → 대기 발생
4. 질의 구문 2: 레코드 A → 대기 발생

데드락이 발생.

멀티스레드 프로그래밍과 차이점이 있다면, 두 질의 구문 프로세스 중 하나는 몇 분간 대기하다가 오류를 출력한다.

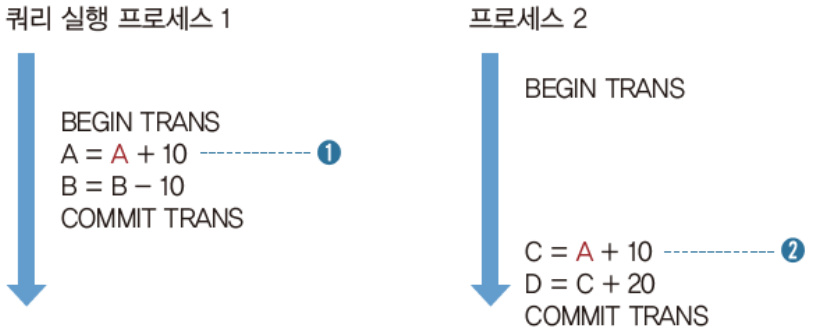


그림 7-24 질의 구문 실행 프로세스가 데드락을 일으킴

7.9 | 질의 구문 실행

- 두 질의 구문은 반드시 레코드 A → B 순서로 액세스한다.

데이터베이스 트랜잭션을 할 때는 꼭 필요한 최소한의 구간에서만 하는 것이 좋다. 트랜잭션의 영향을 받는 레코드는 최소 개수로 하고, 트랜잭션을 시작부터 끝날 때까지 데이터를 액세스하는 횟수 역시 최소로 한다.

교착 상태로 질의 구문 실행이 매우 오랫동안 블로킹되거나 실패를 출력할 때도 반드시 별도로 처리해 주어야 한다.

트랜잭션이 관여되는 레코드는 다른 질의 구문 프로세스에서 액세스할 경우 블로킹을 일으켜 병렬 처리 효율성이 떨어진다.

게임 서버는 플레이어 정보를 메모리에 보관하고, 플레이어의 데이터를 게임 서버 안에 있는 메모리에서 변경하며, 변경된 내용을 데이터베이스에 저장한다.(데이터베이스를 세이브 데이터처럼 사용하는 셈.)

게임 서버 안 메모리가 연산의 주요 대상이므로 두 플레이어 간 아이템 거래 같은 중요한 처리도 게임 서버 안 메모리에서 모두 판정이 끝난 후 뒤늦게 데이터베이스에 기록된다.

```
// 플레이어 1, 2가 각자 가진 아이템 1, 2를 교환한다.
void RequestExchangeItems(player1, player2, item1, item2)
{
	// 각 플레이어는 아이템을 맞게 갖고 있는가?
	if (!player1.hasItem(item1))
	{
		// 실패를 클라이언트들에 알린다.
		ResponseExchangeItemsFail(...);
		return;
	}
	if (!player2.hasItem(item2))
	{
		// 실패를 클라이언트들에 알린다.
```

7.9 | 질의 구문 실행


```
            ResponseExchangeItemsFail(...);
            return;
        }
        if (!player2.hasItem(item2))
        {
            // 실패를 클라이언트들에 알린다.
            ResponseExchangeItemsFail(...);
            return;
        }
        ...; // 기타 필요한 검사를 더 수행한다.

        // 아이템을 주고받을 수 있게 되었다. 서버 메모리에 있는 데이터를 변경한다.
        player1.removeItem(item1);
        player2.removeItem(item2);
        player1.addItem(item2);
        player2.addItem(item1);
        // 데이터베이스에 변경을 가한다.
        // 데이터베이스 입장에서는 제거 & 추가가 아니라 그냥 소유자만 변경한다.
        // 이미 게임 서버 메모리에서 검증을 마쳤으므로 트랜잭션을 할 필요가 없다.
        db.execute("update Item set owner={player2} where itemID={item1}");
        db.execute("update Item set owner={player1} where itemID={item2}");
}
```

7.9 | 질의 구문 실행

- 레코드를 2개 이상 기록하되 전부 아니면 전무하게 기록하는 동시에 트랜잭션을 사용하기 어려운 상황이라면?

  어딘가에 로그를 남겨 놓고 두 플레이어의 레코드를 업데이트, 그리고 두 레코드의 업데이트가 완료되면 해당 로그를 지워 버림.

  트랜잭션에서 발생하는 잠금 수준을 완화.

  트랜잭션의 잠금 수준을 "다른 곳에서도 읽기 하는 것을 허락한다."라고 변경.

- 상업용 게임 서비스를 정식 오픈하고 나서 종종 겪는 상황들.

  게임 서버가 플레이어 정보를 메모리에 항상 보관하고 있다.

  플레이 처리의 주체는 게임 서버 내부의 메모리 안에서 이루어진다.

  데이터베이스는 세이브 데이터 역할 정도만 한다.

7.10 | 게임 서버에서 질의 구문 실행

- 게임 서버에서 데이터베이스에 액세스하기.

```
DbConnection db = new DbConnection();
db.Open("server=db01.mygame.com;userid=serverbot;password=good_day_
one;database=GameDB");
```

데이터베이스 연결 객체 만들기.

```
DbCommand cmd = new DbCommand(db);
cmd.Execute("insert into table1 (a,b,c) values (123,456,789)");
```

명력 객체 생성-명령 객체에 질의 구문을 문자열로 만들어 넣어서 실행

```
DbCommand cmd = new DbCommand(db);
cmd.Parameters[0] = 123;
cmd.Parameters[1] = 456;
cmd.Parameters[2] = 789;
cmd.Execute("insert into table1 (a,b,c) values (?,?,?)");
```

명력 객체 생성-질의 구문에 채워질 값을 별도로 입력하여 실행. 이 방법을 더 권장한다.

7.10 | 게임 서버에서 질의 구문 실행


```
DbCommand cmd = new DbCommand(db);
cmd.Parameters[0] = 123;
cmd.Parameters[1] = 456;
cmd.Parameters[2] = 789;
cmd.Execute("InsertRecord")
```

매개변수들을 지정하고, 저장 프로시저 이름과 함께 "실행하라!"라고 명령을 내림.

```
DbCommand cmd = new DbCommand(db);
cmd.Parameters[0] = 123;
DbRecordset rs = cmd.Execute("select (a,b,c) from table1 where a>=?");
```

저장 프로시저나 질의 구문을 실행했을 때의 레코드 값을 얻어 오고 싶으면 레코드를 지칭하는 객체, 즉 레코드셋 객체를 결과로 얻어 올 수 있다.(a가 123 이상을 가진 레코드를 찾는다면.)

```
foreach(DbRecord r in rs)
{
    int a = r.GetField("a");
    int b = r.GetField("b");
    string c = r.GetField("c");
    ...; // 여기서 얻은 필드들을 사용한다.
}
```

이 select 구문으로 레코드를 0개 이상 얻어 온다.
그러면 얻어 온 레코드들을 순회하면서 내용을 꺼내 오면 된다.
예시에서 필드 c는 문자열 타입이라고 가정.

7.10 | 게임 서버에서 질의 구문 실행

- 필드에 값을 읽거나 쓸 때 혹은 명령 객체에 매개변수를넣거나 꺼내 올 때의 추가 작업 하기

1. 값을 읽을 때는 읽은 값을 가지고 있는 객체, 즉 값 객체 (다음 예시 코드에서는 DbVariable)를 얻는다.
2. 값 객체가 null인지 검사한다. 그리고 값 객체의 실제 값을 원하는 값으로 변환한다.

```
DbVariable v = r.GetField("a");
if (v.isNull() = = false)
{
        int a = v.Convert(int);
}
```

필드에 값을 넣을 때는

1. 여러분이 원하는 값을 넣은 값 객체를 생성한다.
2. 값 객체를 명령 객체나 레코드셋 객체에 넣는다.

```
DbVariable v = new DbVariable(DbType.integer, 123);
record.SetField("a", v);
```

표 7-7 C++와 C#에서 주로 사용하는 모듈, 언어, 데이터베이스, 운영체제의 종류

모듈	지원 언어	지원 데이터베이스	지원 운영체제
ODBC	C, C++	SQL Server, MySQL 등 다양	윈도, 리눅스
MySQL C API	C, C++	MySQL	윈도, 리눅스
ADO	C++, 비주얼 베이직	SQL Server, MySQL 등 다양	윈도
ADO.Net	C#	SQL Server, MySQL 등 다양	윈도, 리눅스
OLE DB	C++	SQL Server	윈도

7.11 | 보안을 위한 주의 사항

데이터베이스의 모든 것을 다룰 수 있는 관리자 계정은 오직 관리자만 직접 다룰 수 있게 한다.
게임 서버가 사용하는 계정은 게임 서버가 다루는 테이블 이외에는 건드리지 못하게 한다.

- 게임 서버나 다른 종류의 서버가 데이터베이스에 질의 구문을 던질 때, 질의 구문을 해커가 조작할 수 있는 상황을 미리 고려해야 한다.

```
sprintf(queryString, "select * from t1 where a='%s'", userName);
db.execute(queryString);
```

```
'; delete from table1; select * from table1 where a='
```

해커가 userName에 고의적으로 코드를 보낼 수도 있다.

```
select * from t1 where a=''; delete from table1; select * from table1 where a=''
```

이 구문을 실행하면 table1의 모든 레코드가 사라진다.(질의 구문 인젝션)

```
DbCommand cmd = new DbCommand(db);
cmd.Parameters[0] = userName;
cmd.Execute("select * from table1 where a=?");
```

사용자 입력 값이 들어갈 곳에 물음표를 넣고 명령 객체에 별도의 매개변수 값으로 사용자의 입력 값을 넣는다.

7.12 | 더 읽을거리

- 데이터베이스를 제대로 공부하려면
  - 데이터베이스 기초 도서를 찾아보세요.
  - 제1~3 정규화 같은 테이블 설계 기초 사용법을 찾아보세요.
  - JOIN 같은 다양한 종류의 데이터베이스 질의 구문 문법을 익혀 두세요.
- ORM(객체 관계 매핑,Object-Relational Mapping)
  - 데이터베이스 우선 방법: 데이터베이스 테이블이나 필드를 정의하면, 이 테이블을 편하게 액세스할 수 있는 데이터베이스 액세스 클래스들이 생성된다. 그 클래스를 그냥 가져다 쓰면 된다.
  - 코드 우선 방법: 데이터베이스를 액세스할 클래스를 정의하고, 클래스 안에는 사용할 멤버 변수들을 정의한다. 그러면 여러분이 정의한 클래스 구조에 따라 데이터베이스가 생성되거나 변경된다.