목차

과목 Ⅰ. 데이터 모델링의 이해

제1장. 데이터 모델링의 이해

1. ②

해설 : 모델링은 단지 시스템 구현만을 위해 수행하는 타스크가 아니며, 시스템 구현을 포함한 업무분석 및 업무형상화를 하는 목적도 있음.

2. ③

해설 : 데이터 모델링을 하는 주요한 이유는 업무정보를 구성하는 기초가 되는 정보들에 대해 일정한 표기법에 의해 표현함으로써 정보시스템 구축의 대상이 되는 업무 내용을 정확하게 분석하는 것이 첫 번째 목적이다. 두 번째는 분석된 모델을 가지고 실제 데이터베이스를 생성하여 개발 및 데이터관리에 사용하기 위한 것이 두 번째 목적이다. 다시 말하면, 데이터모델링이라는 것은 단지 데이터베이스만을 구축 하기 위한 용도로 쓰이는 것이 아니라 데이터모델링 자체로서 업무를 설명하고 분석하는 부분에서도 매우 중요한 의미를 가지고 있다고 할 수 있다.

3. ③

해설 : 데이터모델링을 할 때 유의할 사항은 중복성, 비유연성, 비일관성 등이다.

유의사항1 : 중복(Duplication)

데이터 모델은 같은 데이터를 사용하는 사람, 시간, 그리고 장소를 파악하는데 도움을 줌으로써 데이터베이스가 여러 장소에 같은 정보를 저장하는 잘못을 하지 않도록 한다.

유의사항2 : 비유연성(Inflexibility)

데이터 모델을 어떻게 설계했느냐에 따라 사소한 업무변화에도 데이터 모델이 수시로 변경됨으로써 유지보수의 어려움을 가중시킬 수 있다. 데이터의 정의를 데이터의 사용 프로세스와 분리함으로써 데이터 모델링은 데이터 혹은 프로세스의 작은 변화가 에플리케이션과 데이터베이스에 중대한 변화를 일으킬 수 있는 가능성을 줄인다.

유의사항3 : 비일관성(Inconsistency)

데이터의 중복이 없더라도 비일관성은 발생할 수 있는데, 예를 들면 신용 상태에 대한 갱신 없이 고객의 납부 이력 정보를 갱신하는 경우이다. 개발자가 서로 연관된 다른 데이터와 모순된다는 고려 없이 일련의 데이터를 수정할 수 있기 때문에 이와 같은 문제가 발생할 수 있다. 데이터 모델링을 할 때 데이터와 데이터 간의 상호 연관 관계에 대해 명확하게 정의한다면 이러한 위험을 사전에 예방하는데 도움을 줄 수 있다.

사용자가 처리하는 프로세스 혹은 이와 관련된 프로그램과 테이블의 연계성을 높이는 것은 데이터 모델이 업무 변경에 대해 취약하게 만드는 단점에 해당한다.

4. ②

해설 : 데이터 모델링 시의 유의점에 대한 사항 중 비유연성(Inflexibility)에 대한 설명이다.
데이터 모델을 어떻게 설계했느냐에 따라 사소한 업무변화에도 데이터 모델이 수시로 변경됨으로써 유지보수의 어려움을 가중시킬 수 있다. 데이터의 정의를 데이터의 사용 프로세스와 분리함으로써 데이터 모델링은 데이터 혹은 프로세스의 작은 변화가 애플리케이션과 데이터베이스에 중대한 변화를 일으킬 수 있는 가능성을 줄인다.

5. ①

해설 : 전사적 데이터 모델링이나 EA 수립 시에 많이 하며, 추상화 수준이 높고 업무 중심적이면서 포괄적인 수준의 모델링을 진행하는 것을 개념적 데이터 모델링 이라고 하며, 실제로 데이터베이스에 이식할 수 있도록 성능, 저장 등의 물리적인 성격을 고려한 데이터 모델링 방식은 물리적 데이터 모델링이라고 한다.

6. ②

해설 : 데이터베이스 스키마 구조는 3단계로 구분되고 각각은 상호 독립적인 의미를 가지고 고유한 기능을 가진다. 그 중 통합관점의 스키마구조를 표현한 것을 개념스키마(Conceptual Schema)라고 하며, 데이터 모델링은 통합관점의 뷰를 가지고 있는 개념 스키마를 만들어가는 과정으로 이해할 수 있다.

7. ④

해설 : 부모 엔터티에 데이터가 입력될 때 자식 엔터티에 해당 값이 존재하는지의 여부와 상관없이 입력될 수 있는 구조로 표현되어 있기 때문에, 고객 엔터티에 새로운 고객번호 데이터를 입력하는 것은 주문 엔터티에 해당 고객번호가 존재하고 있는지의 여부와 상관없이 가능하다.

8. ④

해설 : 엔터티를 어디에 배치하는가에 대한 문제는 필수사항은 아니지만 데이터 모델링 툴 사용 여부와 상관없이 데이터 모델의 가독성 측면에서 중요한 문제이다. 일반적으로 사람의 눈은 왼쪽에서 오른쪽, 위쪽에서 아래쪽으로 이동하는 경향이 있기 때문에, 데이터 모델링에서도 가장 중요한 엔터티를 왼쪽 상단에 배치하고, 이것을 중심으로 다른 엔터티를 나열하면서 전개하면 사람의 눈이 따라가기에 편리한 데이터 모델을 작성할 수 있다. 해당 업무에서 가장 중요한 엔터티는 왼쪽 상단에서 조금 아래쪽 중앙에 배치하여 전체 엔터티와 어울릴 수 있도록 하면 향후 관련 엔터티와 관계선을 연결할 때 선이 꼬이지 않고 효과적으로 배치할 수 있게 된다.

9. ②

해설 : 병원은 S병원1개이므로 엔터티로 성립되지 않으며, 이름, 주소는 엔터티의 속성으로 인식될 수 있다. 엔터티는 2개 이상의 속성과 2개 이상의 인스턴스를 가져 소위 면적으로 표현될 수 있어야 비로소 기본적인 엔터티의 자격을 갖췄다 할 수 있으므로 '여러 명'의 복수 인스턴스와 이름, 주소 등의 복수 속성을 가진 '환자'가 엔터티로 가장 적절하다고 할 수 있다.

10. ③

해설 : 엔터티의 특징은 다음과 같다.

☞ 첫 번째, 반드시 해당 업무에서 필요하고 관리하고자 하는 정보이어야 한다.
(예. 환자, 토익의 응시횟수…)

☞ 두 번째, 유일한 식별자에 의해 식별이 가능해야 한다.

☞ 세 번째, 영속적으로 존재하는 (두 개 이상의) 인스턴스의 집합이어야 한다.
("한 개"가 아니라 "두 개 이상")

☞ 네 번째, 엔터티는 업무 프로세스에 의해 이용되어야 한다.

☞ 다섯 번째, 엔터티는 반드시 속성이 있어야 한다.

☞ 여섯 번째, 엔터티는 다른 엔터티와 최소 한 개 이상의 관계가 있어야 한다.

11. ①

해설 : 엔터티의 중요한 특징의 하나는 다른 엔터티와 관계를 가져야 한다는 것이다. 그러나 공통코드, 통계성 엔터티의 경우는 관계를 생략할 수 있다.

12. ①

해설 : 기본엔터티(키엔터티)란 그 업무에 원래 존재하는 정보로서 다른 엔터티와의 관계에 의해 생성되지 않고 독립적으로 생성이 가능하고 자신은 타 엔터티의 부모의 역할을 하게 된다. 다른 엔터티로부터 주식별자를 상속받지 않고 자신의 고유한 주식별자를 가지게 된다. 예를 들어 사원, 부서, 고객, 상품, 자재 등이 기본엔터티가 될 수 있다.

13. ①

해설 : 엔터티를 명명하는 일반적인 기준은 다음과 같다. 용어를 사용하는 모든 표기법이 다 그렇듯이 첫 번째는 가능하면 현업업무에서 사용하는 용어를 사용한다. 두 번째는 가능하면 약어를 사용하지 않는다. 세 번째는 단수명사를 사용한다. 네 번째는 모든 엔터티를 통틀어서 유일하게 이름이 부여되어야 한다. 다섯 번째는 엔터티 생성의미대로 이름을 부여한다.

14. 속성(ATTRIBUTE)

해설 : 속성이란 사전적인 의미로는 사물(事物)의 성질, 특징 또는 본질적인 성질이다. 그것이 없다면 실체를 생각할 수 없는 것으로 정의할 수 있다. 본질적 속성이란 어떤 사물 또는 개념에 없어서는 안 될 징표(徵表)의 전부이다. 이 징표는 사물이나 개념이 어떤 것인지를 나타내고 그것을 다른 것과 구별하는 성질이라고 할 수 있다. 이런 사전적인 정의 외에 데이터모델링 관점에서 속성을 정의하자면, "업무에서 필요로 하는 인스턴스에서 관리하고자 하는 의미상 더 이상 분리되지 않는 최소의 데이터 단위"로 정의할 수 있다. 업무상 관리가 가능한 최소의 의미 단위로 생각할 수 있고, 이것은 엔터티에서 한 분야를 담당하고 있다.

15. ③

해설 : 하나의 인스턴스에서 각각의 속성은 한 개의 속성값을 가져야 한다.

16. ③

해설 : 이자는 계산된 값으로 파생속성이 맞지만, 이자율은 원래 가지고 있어야 하는 속성이므로 기본속성에 해당한다.

17. ①

해설 : 데이터를 조회할 때 빠른 성능을 할 수 있도록 하기 위해 원래 속성의 값을 계산하여 저장할 수 있도록 만든 속성을 파생속성(Derived Attribute)이라 한다

18. ④

해설 : 각 엔터티(테이블)의 속성에 대해서 어떤 유형의 값이 들어가는지를 정의하는 개념은 도메인(Domain)에 해당함

19. ③

해설 : 속성의 명칭은 애매모호하지 않게, 복합 명사를 사용하여 구체적으로 명명함으로써 전체 데이터모델에서 유일성을 확보하는 것이 반정규화, 통합 등의 작업을 할 때 혼란을 방지할 수 있는 방법이 됨

20. ③, ④

해설 : 데이터모델링에서는 존재적 관계와 행위에 의한 관계를 구분하는 표기법이 없으며, UML에서는 연관관계와 의존관계에 대해 다른 표기법을 가지고 표현하게 되어 있다.

21. ②

해설 : 관계 표기법은 관계명, 관계차수, 선택성(선택사양)의 3가지 개념으로 표현한다.

22. ②

해설 : 관계의 기수성을 나타내는 개념은 관계차수에 해당함

23. ③

해설 : ③ 업무기술서, 장표에 관계연결을 가능하게 하는 동사(Verb)가 있는가? 가 되어야 한다.
동사는 관계를 서술하는 업무기술서의 가장 중요한 사항이다.

24. ④

해설 : 4개의 항목 모두 관계를 정의할 때 체크해야 할 항목이다.

25. ④

해설 :
- 주식별자에 의해 엔터티내에 모든 인스턴스들이 유일하게 구분되어야 한다.
- 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 한다.
- 지정된 주식별자의 값은 자주 변하지 않는 것이어야 한다.
- 주식별자가 지정이 되면 반드시 값이 들어와야 한다.

26. ④

해설 : 사번은 업무적으로 의미 있는 식별자로 시스템적으로 부여된 인조식별자가 아니라 일반적으로 사원 인스턴스의 탄생과 함께 업무적으로 부여되는 사원 인스턴스의 본질적인 속성에 해당한다 할 수 있기 때문에 본질식별자로 볼 수 있다.

27. ②

해설 : 명칭, 내역등과 같이 이름으로 기술되는 것들은 주식별자로 지정하기에 적절하지 않다. 특히 사람의 이름은 동명이인이 있을 수 있기 때문에 주식별자로서 더더욱 부적절하다.

28. ②

해설 : 주식별자를 도출하기 위한 기준은 다음과 같다.

- 해당 업무에서 자주 이용되는 속성을 주식별자로 지정한다.
- 명칭, 내역 등과 같이 이름으로 기술되는 것들은 가능하면 주식별자로 지정하지 않는다.
- 복합으로 주식별자로 구성할 경우 너무 많은 속성이 포함되지 않도록 한다.

자주 수정되는 속성이 주식별자가 되면 자식 엔터티에 대한 연쇄 수정이 필요하여 시스템 상에 부하의 원인이 될 수 있기 때문에 주식별자로서 적합하지 않다.

29. ④

해설 : 부모엔터티의 주식별자를 자식엔터티에서 받아 손자엔터티까지 계속 흘려 보내기 위해서는 식별자관계를 고려해야 한다. ③의 경우는 비식별자관계를 선택하는 기준으로 고려하기에 가장 마지막으로 고려할만한 비중을 갖는다고 할 수 있다. 즉, 비식별자관계의 선택이 단순히 SQL 문장의 복잡도를 낮추는 목적에서 고려되는 것은 바람직하지 않음을 의미한다.

30. ②

해설 : 엔터티별로 데이터의 생명주기(LIFE CYCLE)를 다르게 관리할 경우, 예를 들어 부모엔터티의 인스턴스가 자식의 엔터티와 관계를 가지고 있었지만 자식만 남겨두고 먼저 소멸될 수 있는 경우 비식별자관계로 연결하는 것이 적절하다. 부모엔터티의 인스턴스가 자식 엔터티와 같이 소멸되는 경우는 비식별자관계보다 식별자관계로 정의하는 것이 더 적합하다.

제2장. 데이터 모델과 성능

31. ①

해설 : 분석/설계 단계에서 데이터베이스 처리 성능을 향상 시킬 수 있는 방법을 주도 면밀하게 고려해야 한다. 만약 어떤 트랜잭션이 해당 비즈니스 처리에 핵심적이고 사용자 업무처리에 있어 중요함을 가지고 있고 성능이 저하되면 안되는 특징을 가지고 있다면, 프로젝트 초기에 운영환경에 대비한 테스트 환경을 구현하고 그곳에 트랜잭션을 발생시켜 실제 성능을 테스트해 보아야 한다. 이때 데이터 모델의 구조도 변경하면서 어떠한 구조가 해당 사이트에 성능상 가장 적절한 구조인지를 검토하여 성능이 좋은 모습으로 디자인 하는 전략이 요구된다. 보기에서 문제 발생 시점의 SQL을 중심으로 집중 튜닝하는 것은 성능 데이터모델링과 무관한 내용이다.

32. 반정규화(역정규화)

해설 : 데이터 모델링 단계에서 성능을 충분히 고려하기 위한 성능 데이터 모델링 수행 절차에 대한 설명으로, 그 과정은 다음과 같다.

첫번째, 데이터모델링을 할 때 정규화를 정확하게 수행한다.

두번째, 데이터베이스 용량산정을 수행한다.

세번째, 데이터베이스에 발생되는 트랜잭션의 유형을 파악한다.

네번째, 용량과 트랜잭션의 유형에 따라 반정규화를 수행한다.

다섯번째, 이력모델의 조정, PK/FK조정, 슈퍼타입/서브타입 조정 등을 수행한다.

여섯번째, 성능관점에서 데이터모델을 검증한다.

33. ④

해설 : 성능을 고려한 데이터모델링은 정규화를 수행한 이후에 용량산정과 트랜잭션 유형을 파악하여 반정규화를 수행한다. 또한 PK/FK등을 조정하여 인덱스의 특징을 반영한 데이터모델로 만들고 이후에 데이터모델을 검증하는 방법으로 전개한다.

34. ①

해설 : 정규화가 항상 조회 성능을 저하시킨다는 것은 잘못된 생각이며 기본적으로 중복된 데이터를 제거함으로써 조회성능을 향상시킬 수 있음을 알아야 한다.

35. ③

해설 : 함수종속성의 규칙에 따라 {관서번호} → {관리점번호, 관서명, 상태, 관서등록일자}인 관서번호가 PK인 엔터티가 2차 정규화로 분리되어야 한다.

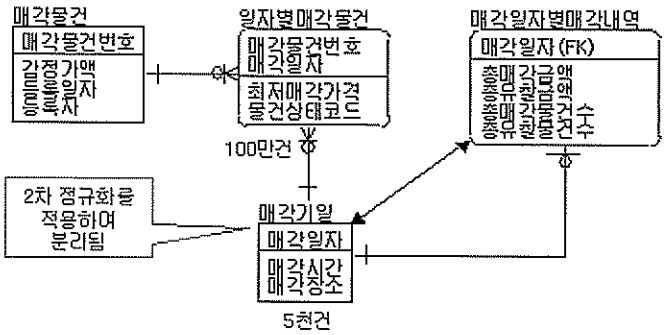
36. ③

해설 : 매각기일은 일자별로 매각이 시행되는 장소와 시간을 의미하는 것으로, 일자별매각물건 엔터티의 매각시간, 매각장소 속성은 두 개의 주식별자 속성 중 매각일자에만 종속되기 때문에 2차 정규화 대상이 된다.

그러므로 매각일자를 주식별자로 하고 매각시간과 매각장소 속성을 포함하는 매각기일 엔터티를 독립시킨다. 이때 매각기일 엔터티는 일자별매각물건의 주식별자 중 일부로부터 독립했기 때문에 매각기일과 일자별매각물건은 1:M 관계로 연결된다.

이와 같은 2차 정규화를 통해 특정 장소에서 이루어진 매각내역을 조회하고자 할 때 100만건의 일자별 매각내역 데이터를 모두 읽어 원하는 장소에 해당하는 인스턴스들을 찾아 매각일자별로 그룹핑한 후 매각일자별매각내역과 조인할 필요가 없이 매우 적은 수의 매각기일 엔터티에서 특정 장소에 해당하는 매각일자들을 찾아 매각일자별매각내역과 1:1로 바로 조인하면 되기 때문에 I/O를 현저하게 감소시킬 수 있어 성능 향상 효과를 얻을 수 있다.

성능이 저하된 반정규화 사례 – 정규화를 통한 성능향상



특정 매각장소에 대해 매각일자를 찾아 매각내역을 조회하려면
500건의 매각기일과 매각일자별매각내역과 조인이 된다.

37. ④

해설 : 칼럼에 의한 반복적인 속성값을 갖는 형태는 속성의 원자성을 위배한 제1차 정규화의 대상이 된다. 이와 같은 반복적인 속성 나열 형태에서는 각 속성에 대해 'or' 연산자로 연결된 조건들이 사용되는데, 이 때 어느 하나의 속성이라도 인덱스가 정의되어 있지 않게 되면 'or'로 연결된 모든 조건절들이 인덱스를 사용하지 않고 한 번의 전체 데이터 스캔으로 처리되게 되어 성능 저하가 나타날 수 있게 되며, 또한 모든 반복 속성에 인덱스를 생성하게 되면 검색 속도는 좋아지겠지만 반대급부적으로 너무 많은 인덱스로 인해 입력, 수정, 삭제의 성능이 저하되므로, 1차 정규화를 통해서 자연스럽게 문제가 해결될 수 있도록 해야 한다.

38. ①

해설 : 컬럼 단위에서 중복된 경우도 1차 정규화의 대상이 된다. 이에 대한 분리는 1:M의 관계로 두 개의 엔터티로 분리된다.

39. ①

해설 : PK에 대해 반복이 되는 그룹(Repeating)이 존재하지 않으므로 1차 정규형이라고 할 수 있으며, 부분함수종속의 규칙을 가지고 있으므로 2차 정규형이라고 할 수 없음. 2차 정규화의 대상이 되는 엔터터임.

40. ①

해설 : • 다량 데이터 탐색의 경우 인덱스가 아닌 파티션 및 데이터 클러스터링 등의 다양한 물리 저장 기법을 활용하여 성능 개선을 유도할 수 있다. 다만, 하나의 결과셋을 추출하기 위해 다량의 데이터를 탐색하는 처리가 반복적으로 빈번하게 발생한다면 이때는 반정규화를 고려하는 것이 좋다.
• 이전 또는 이후 위치의 레코드에 대한 탐색은 window function으로 접근 가능하다.
• 집계 테이블 이외에도 다양한 유형(다수 테이블의 키 연결 테이블 등)에 대하여 반정규화 테이블 적용이 필요할 수 있다.

41. ④

해설 : 반정규화의 기법은 테이블, 속성, 관계에 대해서 반정규화를 적용할 수 있으며, 하나의 테이블의 전체 칼럼 중 자주 이용하는 집중화된 칼럼들이 있을 때 디스크 I/O를 줄이기 위해 해당 칼럼들을 별도로 모아 놓는 반 정규화 기법은 테이블추가 반정규화 기법 중에서 부분테이블 추가에 해당한다.

42. ③

해설 : ③ FK에 대한 속성 추가는 반정규화 기법이라기 보다는 데이터모델링에서 관계를 연결할 때 나타나는 자연스러운 현상이다.

43. ③

해설 : 제품 엔터티에 단가를 주문번호별로 합하는 것은 해당 제품이 여러 주문에 포함될 수 있기 때문에 특정 주문번호만의 단가 합계금액을 갖고 있을 수 없고, 주문목록 엔터티에 주문번호별 단가 합계 금액을 추가하게 되면 하나의 주문에 포함된 제품번호마다 동일한 합계 금액을 반복적으로 저장해야 해서 일관성 문제가 발생할 수 있다. 또한 제품 엔터티에 최근값 여부 칼럼을 추가하는 것은 단가 합계 금액을 빠르게 얻기 위한 반정규화와 무관한 조치이다. 그러므로 주문 엔터티에 전체를 통합한 계산된 칼럼을 추가하는 것이 한번에 데이터를 조회하는 방법이 되므로 가장 효과적인 반정규화 기법이다.

44. ①

해설 : 최근에 변경된 값만을 조회할 경우 과도한 조인으로 인해 성능이 저하되어 나타나게 된다.

45. ③

해설 : 한 테이블에 많은 칼럼들이 존재할 경우 데이터가 물리적으로 저장되는 디스크 상에 넓게 분포할 가능성이 커지게 되어 디스크 I/O가 대량으로 발생할 수 있고, 이로 인해 성능이 저하될 수 있음. 따라서 트랜잭션이 접근하는 칼럼유형을 분석해서 자주 접근하는 칼럼들과 상대적으로 접근 빈도가 낮은 칼럼들을 구분하여 1:1로 테이블을 분리하면 디스크 I/O가 줄어들어 성능을 향상 시킬 수 있다. 테이블 내에서 칼럼의 위치를 조정하는 것은 데이터 주로 채워지는 칼럼을 앞 쪽에 위치시키고, 데이터가 채워지지 않고 주로 NULL 상태로 존재하는 칼럼들을 뒤쪽에 모아둠으로써 로우의 길이를 어느 정도 감소시킬 수 있으나, NULL 상태이던 칼럼에 나중에 데이터가 채워지게 될 경우 더 많은 로우 체인이 발생할 수도 있기 때문에 바람직한 해결책이라고 보기에 부족하며, 무엇보다도 데이터가 채워지지 않고 NULL 상태로 존재하게 되는 칼럼들이 많이 나타나는 경우는, 너무 많은 엔터티들에 무리하게 동질성을 부여하여 통합을 수행했거나, 예측하기 어려운 미래 시점을 겨냥하여 과도하게 의욕적으로 속성을 확장한 경우 등에서 주로 나타나기 때문에, 자주 사용되는 칼럼들이나 현시점에서 주로 사용되는 칼럼들을 한데 모으고, 사용빈도가 낮은 칼럼들이나 미래 시점에 사용될 것으로 예상되는 나머지 칼럼들을 한데 모아 별도의 1:1 관계 엔터티로 분리하는 등의 데이터모델 설계 수정을 고려해 보는 것이 좋다.

46. 파티셔닝(Partitionning)

47. ②

해설 : 개별 테이블을 모두 조회하는 트랜잭션이 대부분이라는 가정이 있으므로 UNION/UNION ALL할 경우 개별조회에 따른 시간소요와 이것을 조합하는 성능저하가 발생된다. 따라서 하나의 테이블로 통합하도록 하고 대신 PK체계나 일반속성에 각 사건을 구분할 수 있도록 구분자를 부여한다.

48. ④

해설 : 트랜잭션은 항상 전체를 통합하여 분석 처리하는데 슈퍼-서브타입이 하나의 테이블로 통합되어 있으면 하나의 테이블에 집적된 데이터만 읽어 처리할 수 있기 때문에 다른 형식에 비해 더 성능이 우수하다. (조인 감소)

49. ④

해설 : 인덱스는 값의 범위에 따라 일정하게 정렬이 되어 있으므로 상수값으로 EQUAL 조건으로 조회되는 칼럼이 가장 앞으로 나오고 범위조회 하는 유형의 칼럼이 그 다음에 오도록 하는 것이 인덱스 액세스 범위를 좁힐 수 있는 가장 좋은 방법이 됨

50. ②

해설 : '='로 들어온 조건에 해당하는 칼럼이 인덱스의 가장 앞쪽에 위치할 때 인덱스의 이용 효율성이 가장 높다고 할 수 있다.

51. ②, ④

해설 : 엔터티 간에 논리적 관계가 있을 경우 즉, 엔터티 간에 관계(Relationship)을 정의하여 관련 엔터티 상호간에 업무적인 연관성이 있음을 표현한 경우에는, 이 데이터들이 업무적으로 밀접하게 연결되어 상호간에 조인이 자주 발생한다는 것을 의미하는 것이기 때문에, 데이터베이스 상에서 DBMS가 제공하는 FK Constraints를 생성했는지 여부와 상관없이 조인 성능을 향상시키기 위한 인덱스를 생성해주는 것이 좋다. 그러므로 수강신청테이블의 학사기준번호에 인덱스가 필요하다.

데이터베이스에 생성하는 FK Constraints는 데이터 모델 상에 표현된 논리적 관계에 따라 관련 인스턴스 간에 일관성을 보장하기 위해 설계된 제약조건을 구현할 수 있도록 DBMS가 제공해주는 하나의 '지원 기능'으로 이해될 수 있다.

52. ④

해설 : Global Single Instance(GSI)는 통합된 한 개의 인스턴스 즉, 통합 데이터베이스 구조를 의미하므로, 분산데이터베이스와는 대치되는 개념이다.

공통코드, 기준정보 등과 같은 마스터 데이터를 한 곳에 두고 운영하는 경우 원격지에서의 접근이 빈번할수록 실시간 업무처리에 대해 좋은 성능을 얻기가 어려울 수 있기 때문에 분산 환경에 복제분산을 하는 방법으로 분산데이터베이스를 구성할 수 있다. 또한 백업 사이트 구성에 대해서도 분산 환경으로 구성하여 적용할 수 있다.

과목 Ⅱ. SQL 기본 및 활용

제1장. SQL 기본

1. ④

해설 : 데이터 제어어(DCL:Data Control Language)는 데이터베이스에 접근하고 객체들을 사용할 수 있도록 권한을 부여하거나 회수하는 명령어로 GRANT, REVOKE가 있다.

2. ②

해설 : 데이터의 구조를 정의하는 명령어는 DDL(데이터 정의어)에 해당하며 DDL 문으로는 CREATE, ALTER, DROP, RENAME 이 있다.

3. TCL

해설 : Transaction를 제어하는 명령어는 TCL(Transaction Control Language) 이다.

4. ①

해설 : As-Is : 비절차적 데이터 조작어(DML)는 사용자가 무슨(What) 데이터를 원하는 지만을 명세함.
To-Be : 비절차적 데이터 조작어(DML)는 사용자가 무슨(What) 데이터를 원하는 지만을 명세하지만, 절차적 데이터 조작어는 어떻게 (How) 데이터를 접근해야 하는지 명세 한다. 절차적 데이터 조작어로는 PL/SQL(오라클), T-SQL(SQL Server 등이 있다.

5. ①, ②

해설 : DDL(Data Definition Language) : CREATE, DROP, ALTER, RENAME
DML(Data Modification Language) : SELECT, INSERT, UPDATE, DELETE
DCL(Data Control Language): GRANT, GRANT, REVOKE
TCL(Transaction Control Language) : COMMIT, ROLLBACK

6. ④

해설 : ①은 PK를 지정하는 ALTER TABLE 문장에 문법 오류가 존재하고, 올라른 문법이 사용된 문장은 다음과 같다.
(오류 발생) ALTER TABLE PRODUCT ADD PRIMARY KEY PRODUCT_PK ON (PROD_ID);
(오류 수정) ALTER TABLE PRODUCT ADD CONSTRAINT PRODUCT_PK PRIMARY KEY (PROD_ID);
②는 NOT NULL 컬럼에 대해서 NOT NULL 제약조건을 지정하지 않았다.
③은 테이블을 생성할 때 PK를 지정하는 문장에 문법 오류가 존재한다.

7. ④

해설 : ①, ② SQLServer에서는 여러개의 컬럼을 동시에 수정하는 구문은 지원하지 않으므로 오류가 발생한다. 또한 SQLServer에서는 괄호를 사용하지 않는다.
③ 분류명을 수정할 때 NOT NULL 구문을 지정하지 않으면, 기존의 NOT NULL 제약조건이 NULL로 변경되므로 NOT NULL 요건을 만족하지 않는다.

8. ③

해설 : NULL은 공백문자(Empty String) 혹은 숫자 0과 동일하지 않다.

9. ②

해설 : DELETE FROM T; 이후 데이터 현황
- T 테이블 : 두건 모두 삭제됨
- S 테이블 (Cascade 옵션) : 두건 모두 삭제됨
- R 테이블 (Set Null 옵션) : Child 해당 필드(FK: B칼럼) 값이 Null로 변경됨

10. ①

해설 : PK = UNIQUE & NOT NULL 특징을 가짐.
UNIQUE는 테이블 내에서 중복되는 값이 없지만, NULL 입력이 가능하다.

11. ①

해설 : 테이블명과 칼럼명은 반드시 문자로 시작해야 한다.
사용되는 글자는 A-Z, a-z, 0-9, _, $, # 만 허용함.

12. ①, ③

해설 : ② SQL 문장은 정상적으로 수행되지만, DEPT_CODE 컬럼에 NOT NULL 제약조건이 생성되지 않는다. NOT NULL 제약조건이 생성되지 않으면 명시적으로 DEPT_CODE 컬럼에 NULL을 입력하게 되면 NULL이 입력되는 문제가 발생한다.
④ 테이블 생성문과 인덱스 생성문은 정상적으로 수행되지만, 테이블 생성문장에서 이미 PRIMARY KEY를 지정하였으므로 ALTER TABLE 문장에서 오류가 발생한다.

13. ③

해설 : 학번 칼럼이 PK 이기 때문에 NULL 값이 없다. count(*)와 COUNT(학번)의 결과는 항상 같다.

14. ②, ③

해설 : ① 테이블 생성시 설정할 수 있다.
② 외래키 값은 널 값을 가질 수 없다. → 있다
③ 한 테이블에 하나만 존재해야 한다. → 여러 개 존재할 수 있다.
④ 외래키 값은 참조 무결성 제약을 받을 수 있다.

15. ③

해설 : 고유키(Unique Key)로 지정된 모든 컬럼은 Null 값을 가질 수도 있으므로 ③번 보기에 오류가 있다.

16. ALTER, DROP COLUMN

해설 : Table 스키마 변경 시 사용하는 SQL문은 DDL(Data Definition Language)로 컬럼 삭제 시 활용되는 문장은 다음과 같다.

ALTER TABLE 테이블명

DROP COLUMN 컬럼명

17. ②

해설 : 참조무결성 규정

DELETE FROM 부서 WHERE 부서번호 = '20'; →

CASCADE 참조 무결성 규정이므로 직원 테이블의 '2000','3000'도 같이 삭제됨

SELECT COUNT(직원번호) FROM 직원

직원 테이블'1000'에 대한 1건이 출력됨

Delete(/Modify) Action : Cascade, Set Null, Set Default, Restrict (부서-사원)

1) Cascade : Master 삭제 시 Child 같이 삭제
2) Set Null : Master 삭제 시 Child 해당 필드 Null
3) Set Default : Master 삭제 시 Child 해당 필드 Default 값으로 설정
4) Restrict : Child 테이블에 PK 값이 없는 경우만 Master 삭제 허용
5) No Action : 참조무결성을 위반하는 삭제/수정 액션을 취하지 않음

18. RENAME STADIUM TO STADIUM_JSC;

해설 : RENAME OLD_OBJECT_NAME TO NEW_OBJECT_NAME (ANSI 표준 기준, 오라클과 동일함)

RENAME STADIUM TO STADIUM_JSC;

19. ④

해설 : - Delete(/Modify) Action : Cascade, Set Null, Set Default, Restrict (부서-사원)

1) Cascade : Master 삭제 시 Child 같이 삭제
2) Set Null : Master 삭제 시 Child 해당 필드 Null
3) Set Default : Master 삭제 시 Child 해당 필드 Default 값으로 설정
4) Restrict : Child 테이블에 PK 값이 없는 경우만 Master 삭제 허용
5) No Action : 참조무결성을 위반하는 삭제/수정 액션을 취하지 않음

- Insert Action : Automatic, Set Null, Set Default, Dependent (부서-사원)

1) Automatic : Master 테이블에 PK가 없는 경우 Master PK를 생성 후 Child 입력
2) Set Null : Master 테이블에 PK가 없는 경우 Child 외부키를 Null 값으로 처리
3) Set Default : Master 테이블에 PK가 없는 경우 Child 외부키를 지정된 기본값으로 입력
4) Dependent : Master 테이블에 PK가 존재할 때만 Child 입력 허용
5) No Action : 참조무결성을 위반하는 입력 액션을 취하지 않음

20. ④

해설 : 1 : 삽입 컬럼을 명시하지 않았을 경우 모든 컬럼을 삽입해야 한다.
2 : DEGREE 컬럼의 길이는 VARCHAR2(1)이다. 'AB'는 컬럼 길이를 초과한다.
3 : Not Null 컬럼인 AMT 컬럼을 명시하지 않았다.

21. ②

해설 : ②번 SQL은 REG_DATE 컬럼에 NOT NULL 제약조건이 있지만 INSERT INTO 구문에는 REG_DATE 컬럼이 대입되지 않아 NULL로 입력되므로 오류가 발생한다.

22. ①, ③

해설 : ②는 고객 테이블에 존재하지 않는 고객ID의 주문을 입력하려고하여 무결성 제약 오류가 발생한다. (①번 SQL을 수행 후에는 정상 입력된다.)
④는 고객 테이블의 고객ID 'C002'를 삭제하려고 할 때 SQL에 의해 추가된 CONSTRAINT에 따라 주문 테이블의 고객ID를 NULL로 업데이트하려고 DBMS에서 시도하지만, 주문 테이블 고객ID 컬럼의 NOT NULL 제약조건에 의해 실패한다.

23. ①

해설 : TRUNCATE TABLE과 DROP TABLE은 로그를 남기지 않으므로 개발 기준과 상충된다.
지문2는 문법에 맞지 않다.

24. DISTINCT

해설 : 데이터의 중복을 제거하는 명령어는 "DISTINCT" 이다. GROUP BY문을 사용하여 다음과 같이 중복 데이터를 제거 할 수 있다.

```
SELECT 거주지, 근무지
FROM   고객지역
GROUP BY 거주지, 근무지 ;
```

25. ①

해설 : ① 특정 테이블의 모든 데이터를 삭제하고, 디스크 사용량을 초기화 하기위해서는 TRUNCATE TABLE 명령을 사용하여야 한다.

② DELETE TABLE은 테이블의 데이터를 모두 삭제하지만, 디스크 사용량을 초기화 하지는 않는다.

③ DROP TABLE은 테이블의 데이터를 모두 삭제하고 디스크 사용량도 없앨(초기화) 수 있지만, 테이블의 스키마 정의도 함께 삭제된다.

④ DELETE TABLE FROM은 존재하지 않는 명령어이다.

26. ①, ④

해설 :

DROP	TRUNCATE	DELETE
DDL	DDL (일부 DML 성격 가짐)	DML
Rollback 불가능	Rollback 불가능	Commit 이전 Rollback 가능
Auto Commit	Auto Commit	사용자 Commit
테이블이 사용했던 Storage를 모두 Release	테이블이 사용했던 Storage중 최초 테이블 생성시 할당된 Storage만 남기고 Release	데이터를 모두 Delete해도 사용했던 Storage는 Release되지 않음
테이블의 정의 자체를 완전히 삭제함	테이블을 최초 생성된 초기상태로 만듬	데이터만 삭제

27. ②, ④

해설 : 데이터베이스 트랜잭션의 4가지 특성 : 일관성과 지속성의 설명이 바뀌었다.

특성	설명
원자성 (atomicity)	트랜잭션에서 정의된 연산들은 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 한다. (All or Nothing)
일관성 (consistency)	트랜잭션이 실행 되기 전의 데이터베이스 내용이 잘못 되어 있지 않다면 트랜잭션이 실행된 이후에도 데이터베이스의 내용에 잘못이 있으면 안된다.
고립성 (isolation)	트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다.
지속성 (durability)	트랜잭션이 성공적으로 수행되면 그 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장된다.

28. ①, ④

해설 : ① Dirty Read : 다른 트랜잭션에 의해 수정되었지만 아직 커밋되지 않은 데이터를 읽는 것을 말한다.
④ isolation(트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다)은 데이터베이스 트랜잭션의 4가지 특성으로 문제점이 아니고 목표라고 할 수 있다.

29. ③

해설 : ① ORACLE에서는 DDL 문장 수행 후 자동으로 COMMIT을 수행한다.
② SQL Server에서는 DDL 문장 수행 후 자동으로 COMMIT을 수행하지 않는다.
③ ORACLE에서 DDL 문장의 수행은 내부적으로 트랜젝션을 종료 시키므로 B 테이블은 생성된다.
④ SQL Server에서는 CREATE TABLE 문장도 TRANSACTION의 범주에 포함된다. 그러므로 ROLLBACK 문장에 의해서 최종적으로 B 테이블은 생성되지 않는다.

30. ㉠ : 트랜젝션 또는 Transaction
㉡ : 커밋 또는 Commit
㉢ : 롤백 또는 Rollback

31. ③

해설 : ROLLBACK 구문은 COMMIT되지 않은 상위의 모든 Transaction을 모두 rollback한다.

32. LCD-TV

해설 : ROLLBACK TRANSACTION SP2 문장에 의해 UPDATE 상품 SET 상품명 = '평면-TV' WHERE 상품ID = '001' 이 ROLLBACK 되었고, 첫 번째 UPDATE 문장만 유효한 상태에서 COMMIT 되었으므로 첫 번째 UPDATE한 내역만 반영 된다. 그러므로 LCD-TV가 된다.

33. WHERE 또는 WHERE 절

해설 : WHERE 절은 SQL을 이용하여 데이터베이스로부터 데이터를 검색할 때 조회되어야 하는 데이터를 필터링하는데 사용된다.

34. ②

해설 : 논리연사자의 우선순위는 NOT 〉 AND 〉 OR 순이다.
(EMPNO 〉 100 AND SAL 〉= 3000) OR EMPNO = 200

35. ④

해설 : NULL 값이 포함된 4칙 연산의 결과는 NULL이다.
30+20=50
NULL+40=NULL
50+NULL=NULL

36. ①

해설 : NULL 값을 조건절에서 사용하는 경우 IS NULL, IS NOT NULL이란 키워드를 사용해야 한다.

37. ④

해설 : ① 서비스번호 컬럼의 모든 레코드가 '001'과 같은 숫자형식으로 입력되어 있어야 오류가 발생하지 않는다.
② ㉡과같이 데이터를 입력하면, 서비스명 컬럼의 데이터에 대해서 ORACLE에서는 NULL로 입력된다.
③ ㉡과같이 데이터가 입력되어있을 때, ORACLE에서 데이터를 조회하려면 서비스명 IS NULL 조건으로 조회하여야 한다.
④ ㉡과같이 데이터가 입력되어있을 때, SQL Server에서 데이터를 조회하려면 서비스명 = '' 로 조회하여야 한다.

38. ④

해설 : ①의 조건은 2014년 03월부터 12월까지 매출금액과 2015년 03월부터 2013년 12월까지의 매출금액의 합이다.
②의 조건은 ①의 조건과 동일하다.
③의 조건은 2014년 01월부터 12월까지의 매출금액과 2015년 01월부터 12월까지의 매출금액의 합이다. 즉, 전체 데이터의 합이다.
④의 조건은 2014년 11월부터 2015년 03월까지의 매출금액의 합이며, 연산자의 우선순위(AND 〉 OR)에 의해 괄호가 없어도 된다.

39. ④

해설 : ①, ②, ③번 SQL은 모두 가입이 2014년 12월 01일 00시에 발생했고 서비스 종료일시가 2015년 01월 01일 00시 00분 00초와 2015년 01월 01일 23시 59분 59초 사이에 만료되는 데이터를 찾는 조건이지만, ④번 SQL은 가입 조건은 동일하지만, 서비스 종료일시가 2015년 01월 01일 00시 00분 00초에 종료되는 SQL을 찾는 조건이다.

40. ②

해설 : 다) 1:M 조인이라 하더라 M쪽에서 출력된 행이 하나씩 단일행 함수의 입력값으로 사용되므로 사용할 수 있다.

라) 다중행 함수도 단일행 함수와 동일하게 단일 값만을 반환한다.

41. ③

해설 : 라인수를 구하기 위해서 함수를 이용해서 작성한 SQL이다.

LENGTH : 문자열의 길이를 반환하는 함수

CHR : 주어진 ASCII 코드에 대한 문자를 반환하는 함수 (CHR(10) -> 줄바꿈)

REPLACE : 문자열을 치환하는 함수 (REPLACE(C1, CHR(10)) -> 줄바꿈 제거)

함수 결과 값

ROWNUM	C1	LENGTH(C1)	REPLACE(C1, CHR(10))		LENGTH(REPLACE(C1, CHR(10)))
1	A A	3	변경 전 A A	변경 후 AA	2
2	B B B	5	변경 전 B B B	변경 후 BBB	3

42. ③

해설 : 오라클에서 날짜의 연산은 숫자의 연산과 같다. 특정 날짜에 1을 더하면 하루를 더한 결과와 같으므로 1/24/60 = 1분을 의미한다. 1/24/(60/10) = 10분과 같으므로 2015년 1월 10일 10시에 10분을 더한 결과와 같다.

43. LOC WHEN 'NEW YORK' THEN 'EAST'

해설 : SEARCHED_CASE_EXPRESSION을 SIMPLE_CASE_EXPRESSION으로 변환하는 문제임.

```
SELECT LOC,
CASE LOC WHEN 'NEW YORK' THEN 'EAST'
ELSE 'ETC'
END as AREA
FROM DEPT;
```

44. ④

해설 : 지문 4는 CASE 문장에서 데이터가 없는 경우를 0으로 표시해야(ELSE 0),
다른 3개의 지문과 같은 결과가 나온다.

45. ②

해설 : ISNULL함수는 결과값이 NULL일 경우 지정된 값을 반환한다. 칼럼의 NULL 값을 확인할 때는 IS NULL을 사용해야 한다.

46. NULLIF

해설 : NULLIF 함수는 EXPR1이 EXPR2와 같으면 NULL을, 같지 않으면 EXPR1을 리턴한다.
특정 값을 NULL로 대체하는 경우에 유용하게 사용할 수 있다.
NULLIF (EXPR1, EXPR2)

47. ④

해설 : NULL이 포함된 연산의 결과는 NULL이다.
분모가 0이 들어가는 경우 연산 자체가 에러를 발생하며 원하는 결과를 얻을 수 없다.
1. 0/300 = 0
2. 5000/0 : 에러 발생
3. 1000/NULL = NULL

48. ③

해설 : 따라서, 결과의 합은 6이다.
COALESCE 함수는 첫번째 NULL이 아닌 값을 반환한다.
COALESCE(C1, C2, C3)는 각 Row에서 첫번째로 NULL이 아닌 값인 1, 2, 3을 반환한다.

49. ㉠ : NVL, ㉡ : NULLIF, ㉢ : COALESCE

해설 : ISNULL, NVL 함수는 표현식1의 결과값이 NULL이면 표현식2의 값을 출력하며, NULLIF는 표현식1일 표현식2와 같으면 NULL을 같지 않으면 표현식1을 리턴한다. COALESCE는 임의의 개수 표현식에서 NULL이 아닌 최초의 표현식을 나타낸다.

50. ③

해설 : SELECT AVG(COL3) FROM TAB_A; → (20+0)/2건=10

→ 세번째 행 COL3의 NULL은 AVG 연산 대상에서 제외됨

COL1	COL2	COL3
30	NULL	20
NULL	40	0
0	10	NULL

SELECT AVG(COL3) FROM TAB_A WHERE COL1 〉 0; → (20)/1건=20

→ WHERE절에 의해 COL1이 NULL인 두번째 행은 NULL 연산 제외 조건으로 제외됨

→ WHERE절에 의해 COL1이 0인 세번째 행은 연산 대상에서 제외됨

COL1	COL2	COL3
30	NULL	20

SELECT AVG(COL3) FROM TAB_A WHERE COL1 IS NOT NULL; → (20)/1건=20

→ COL1이 NULL인 두번째 행은 NOT NULL 조건으로 인해 제외됨

→ 세번째 행 COL3D의 NULL은 AVG 연산 대상에서 제외됨

COL1	COL2	COL3
30	NULL	20
0	10	NULL

51. ③

해설 : SQL1) SELECT COUNT(GRADE) FROM EMP;

645건 : 사원 500명 + 대리 100명 + 과장 30명 + 차장 10명 + 부장 5명

이때 NULL 25건은 제외된다.

SQL2) SELECT GRADE FROM EMP WHERE GRADE IN ('차장','부장','널');

15건 : 차장 10명 + 부장 5명

'널'이 텍스트로 입력된 데이터는 없다고 봐야 함

정의되지 않은 미지의 값인 NULL과 '널' 텍스트 데이터는 다름. 또한 IN ('차장', '부장', NULL) 로 변경하여도 실제 NULL데이터는 출력되지 않음. NULL 비교는 오직 'IS NULL, IS NOT NULL'만 가능

SQL3) SELECT GRADE, COUNT(*) FROM EMP GROUP BY GRADE;

6건 : 5개 직급 + NULL 기준별 데이터 수가 6건 출력됨

52. ②

해설 : 광고게시 테이블에서 광고매체ID별로 광고시작일자가 가장 빠른 데이터를 추출하는 SQL을 작성해야 한다.

①의 경우 연관 서브쿼리를 활용하는 방법이지만, 이를 활용하기위해서는 WHERE 절에서 사용되어야 한다.(Inline View에서는 사용할 수 없다)

③은 광고ID별로 광고매체ID와 광고시작일자의 최소값을 출력하므로 틀린 결과이다.

④은 광고게시의 전체데이터에서 광고매체ID의 최소값과 광고시작일자의 최소값을 가져오므로 틀린 결과이다.

53. ④

해설 : ③ GROUP BY로 그룹핑된 컬럼에 대해서 HAVING 조건절을 사용할 경우 집계된 컬럼의 FILTER 조건으로 사용할 수가 있다. 이런 경우 HAVING절에 집계함수가 없이도 사용할 수 있다.

④ 중첩된 그룹함수의 경우 최종 결과값은 1건이 될 수밖에 없기에 GROUP BY절에 기술된 메뉴ID와 사용유형코드는 SELECT절에 기술될 수 없다.

54. ②

해설 : SQL 실행 순서에 의해 HAVING절은 SELECT절보다 선행처리 되기에, SELECT절 COUNT 함수 사용 여부는 관계없다. 위의 SQL은 나 컬럼으로 GROUP BY를 수행하였을 때 건수가 2건 이상인 데이터를 추출하여 SUM(다) 의 값이 큰 순으로 정렬하는 SQL이므로 아래와 같은 결과에서

가	나	다	CNT
009	A003	600	4
005	A002	500	3
002	A001	300	2
010	A004	200	1

CNT가 2이상인 것만 출력된다.

55. ②

해설 : Group By Having 한 결과에 대해 정렬 연산을 하는 것이다.

ID 건수가 2개이며, ORDER BY절 CASE문에 의해 999는 0으로 치환되고 그 외는 ID 값으로 정렬된다.

56. ③

해설 : ② SQL 실행 순서에 의하면 SELECT절 이후에 ORDER BY 절이 수행되기 때문에 SELECT 절에 기술되지 않는 '년' 칼럼으로 정렬하는 것은 논리적으로 맞지 않다. 하지만 오라클은 행기반 DATABASE 이기에 데이터를 액세스할 때 행 전체 칼럼을 메모리에 로드한다. 이와 같은 특성으로 인해 SELECT절에 기술되지 않은 칼럼으로도 정렬을 할 수 있다.

단, 아래와 같은 SQL일 경우에는 정렬을 할 수 없다.

```
SELECT 지역, 매출금액
  FROM (
        SELECT 지역, 매출금액
          FROM   지역별매출
       )
ORDER BY 년 ASC;
```

이는 IN-LINE VIEW가 먼저 수행됨에 따라 더 이상 SELECT절 외 칼럼을 사용할 수 없기 때문이다.

③ GROUP BY를 사용할 경우 GROUP BY 표현식이 아닌 값은 기술될 수 없다.

④ GROUP BY 표현식이기에 가능하다.

57. ③

해설 : ORDER BY 절에 컬럼명 대신 Alias 명이나 컬럼 순서를 나타내는 정수를 혼용하여 사용할 수 있다.

58. ②

해설 : CASE절을 이용해서 원래의 정렬 순서를 변경하였다. 그래서 ID가 'A'인 것이 가장 먼저 표시되도록 하였다.

59. ④

해설 : SELECT 문장의 실행 순서는 FROM - WHERE - GROUP BY - HAVING - SELECT - ORDER BY 이다.

60. ④

해설 : SQL Server의 TOP N 질의문에서 N에 해당하는 값이 동일한 경우 함께 출력되도록 하는 WITH TIES 옵션을 ORDER BY 절과 함께 사용하여야 한다.

61. ③

해설 : 여러 테이블로부터 원하는 데이터를 조회하기 위해서는 전체 테이블 개수에서
최소 N-1 개 만큼의 JOIN 조건이 필요하다.

62. ④

해설 : 영화명과 배우명은 출연 테이블이 아니라 영화와 배우 테이블에서 가지고 와야 하는 속성이므로 출연테이블의 영화번호와 영화테이블의 영화번호 및 출연테이블의 배우번호와 배우테이블의 배우번호를 조인하는 SQL문을 작성해야 함.

63. ④

해설 : DBMS 옵티마이저는 From 절에 나열된 테이블이 아무리 많아도 항상 2개의 테이블씩 짝을 지어 Join을 수행한다.

64. ③

해설 : LIKE 연산자를 이용한 조인의 이해가 필요하다.
SQL의 실행결과는 다음과 같다.

EMPNO	ENAME	RULE
1000	SMITH	S%
1100	SCOTT	S%
1000	SMITH	%T%
1100	SCOTT	%T%

제2장. SQL 활용

65. ②

해설 : 순수 관계 연산자에는 SELECT, PROJECT, JOIN, DIVIDE가 있다.

66. ③, ④

해설 : ① NOT EXIST 절의 연관서브쿼리에 X.컨텐츠ID = B.컨텐츠ID가 존재하지 않아 단 하나의 컨텐츠라도 비선호로 등록한 고객에 대해서는 모든 컨텐츠가 추천에서 배재 된다.

② 추천컨텐츠를 기준으로 비선호컨텐츠와의 LEFT OUTER JOIN이 수행되고 비선호컨텐츠의 컨텐츠 ID에 대해서 IS NULL 조건(③ 번과 같이)이 있다면 정확히 비선호 컨텐츠만 필터링할 수 있다. (고객이 비선호로 등록하지 않은 컨텐츠는 추천컨텐츠에만 등록 되어있으므로)

67. ①, ②

해설 : ③ 데이터 모델을 보면 제품과 생산라인 엔터티에는 생산제품과 대응되지 않는 레코드가 있을 수 있다.

④ 특정 생산라인에서 생산되는 제품의 제품명을 알기위해서는 제품과 생산제품까지 2개의 엔터티만을 Inner Join 하면 된다.

68. ②

해설 : 구매이력이 있어야 하므로 INNER JOIN이 필요하며, 구매 횟수이므로 COUNT함수를 사용한다.

69. ③

해설 : ① 두 번째 ON 절이 B.사용시간대 BETWEEN C.시작시간대 AND C.시작시간대 가 되어야 한다.

② INNER JOIN 구문 오류가 발생한다.

④ BETWEEN JOIN 이란 구문은 없다. 구문 오류가 발생한다.

70. ①

해설 : TEAM, STADIUM 두 테이블을 조인하여 사용한다.

① USING 조건절을 이용한 EQUI JOIN에서도 NATURAL JOIN과 마찬가지로 JOIN 칼럼에 대해서는 ALIAS나 테이블 이름과 같은 접두사를 붙일 수 없다. 지문 1는 SYNTAX 에러 발생함.

USING T.STADIUM_ID = S.STADIUM_ID

→ USING (STADIUM_ID)

SELECT T.REGION_NAME, T.TEAM_NAME, T.STADIUM_ID, S.STADIUM_NAME

→ SELECT T.REGION_NAME, T.TEAM_NAME, STADIUM_ID, S.STADIUM_NAME

71. CROSS JOIN

해설 : CROSS JOIN은 E.F.CODD 박사가 언급한 일반 집합 연산자의 PRODUCT의 개념으로 테이블 간 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말한다. 조건절이 없거나 CROSS JOIN 키워드를 사용할 수 있다.

72. ①

해설 : WHERE 절에 A.고객번호 IN (11000, 12000) 조건을 넣었다면 정답은 ② 번이 되었을 것이나, ON 절에 A.고객번호 IN (11000, 12000) 조건을 넣었기 때문에 모든 고객에 대해서 출력을 하되 JOIN 대상 데이터를 고객번호 11000과 12000으로 제한되어 ① 번과 같은 결과가 출력된다.

73. ④

해설 : 보기의 3개의 SQL은 모두 Full Outer Join과 동일한 결과를 반환한다.

74. ①

해설 : 주키와 외래키는 영향을 미치지 않는다.

LEFT OUTER JOIN

A	B	C	D	E
1	b	w	1	10
3	d	w	1	10
5	y	y		

FULL OUTER JOIN

A	B	C	D	E
1	b	w	1	10
3	d	w	1	10
5	y	y		
		z	4	11
		v	2	22

RIGHT OUTER JOIN

A	B	C	D	E
1	b	w	1	10
3	d	w	1	10
		z	4	11
		v	2	22

75. LEFT JOIN 또는 LEFT OUTER JOIN

해설 : LEFT OUTER JOIN은 좌측 테이블이 기준이 되어 결과를 생성한다. 즉, TABLE A와 B가 있을 때 (TABLE 'A'가 기준이 됨), A와 B를 비교해서 B의 JOIN 칼럼에서 같은 값이 있을 때 B테이블에서 해당 데이터를 가져오고, B의 JOIN 칼럼에서 같은 값이 없는 경우에는 B 테이블에서 가져오는 칼럼들은 NULL 값으로 채운다. 그리고, LEFT JOIN으로 OUTER 키워드를 생략해서 사용할 수 있다.

76. ②

해설 : 아우터 조인에서 ON절은 조인할 대상을 결정한다. 그러나 기준 테이블은 항상 모두 표시된다. 결과 건에 대한 필터링은 WHERE절에서 수행된다.

77. ①

해설 : 보기는 게시판별 게시글의 개수를 조회하는 SQL이다. 이때 게시글이 존재하지 않는 게시판도 조회되어야 한다. ORACLE에서는 OUTER JOIN 구문을 (+) 기호를 사용하여 처리할 수도 있으며, 이를 ANSI 문장으로 변경하기 위해서는 Inner쪽 테이블(게시글)에 조건절을 ON절에 함께 위치시켜야 정상적인 OUTER JOIN을 수행할 수 있다.

②번의 경우는 Outer 대상이 되는 테이블(게시판)의 조건절이 ON절에 위치하였으므로 원하는 결과가 출력되지 않는다.

78. 5

해설 : 조건에 맞는 Student 데이터는 다음과 같다.

st_num	st_name	d_num
1001	Yoo	10
1003	Lee	20
1004	Park	10
1005	Choi	20
1006	Jeong	10

79. ④

해설 : EXCEPT는 차집합에 대한 연산이므로 NOT IN 또는 NOT EXISTS로 대체하여 처리가 가능하다. ②는 NOT IN을 사용하였으나, PK컬럼 A, B에 대하여 각각 NOT IN 연산을 수행하여 다른 결과가 생성된다.

80. ②

해설 : 수행한 SQL은 이용된 적이 있었던 서비스를 추출하는 SQL이다.

① 이용된 적이 있었던 서비스를 추출하는 것은 동일하나 서비스와 서비스이용은 1:n 관계이므로 서비스이용건수 만큼 추출되므로 전체 결과가 다르다. GROUP BY를 수행하면 동일한 결과를 출력할 수 있다.

② 전체 서비스에서 이용된 적이 있었던 서비스를 MINUS하였으므로 이용된 적이 없었던 서비스가 서브쿼리에서 추출된다. 그러므로 NOT EXISTS 구문을 적용하면 이용된 적이 있었던 서비스가 출력된다.(정답)

③ 서비스를 기준으로 OUTER JOIN을 수행하였으므로, 이용된 적이 없었던 서비스만 출력된다. B.서비스ID IS NOT NULL로 변경해야 동일한 결과가 출력된다.

④ 서비스와 서비스이용 테이블의 순서를 변경하고 IN 절을 NOT IN으로 변경하면 동일한 결과를 출력할 수 있다.

81. ②

해설 : SET OPERATOR : 합집합은 UNION, 교집합은 INTERSECT, 차집합은 MINUS/EXCEPT 이다.

82. ②

해설 : UNION ALL을 사용하는 경우 칼럼 ALIAS는 첫번째 SQL 모듈 기준으로 표시되며, 정렬 기준은 마지막 SQL 모듈에 표시하면 됨.

83. ①

해설 : 집합 연산자는 SQL에서 위에 정의된 연산자가 먼저 수행된다. 그러므로 UNION이 나중에 수행되므로 결과적으로 중복 데이터가 모두 제거되어 ①과 같은 결과가 도출된다. 만일 UNION과 UNION ALL의 순서를 바꾼다면 ②과 같은 결과가 도출된다.

84. ①

해설 : ㉮ SELECT A, B, C FROM R1
UNION ALL
SELECT A, B, C FROM R2;
(중복 레코드 유지, 정렬 안함)

R(A,B,C)

A	B	C
A3	B2	C3
A1	B1	C1
A2	B1	C2
A1	B1	C1
A3	B2	C3

㉯ SELECT A, B, C FROM R1
UNION
SELECT A, B, C FROM R2;
(중복 레코드 제거함, 정렬 발생)

R(A,B,C)

A	B	C
A1	B1	C1
A2	B1	C2
A3	B2	C3

85. ③

해설 : 집합 C는 집합 A와 집합 B의 교집합이며, 데이터베이스에서 교집합 기능을 하는 집합 연산은 Intersection 이다.

86. ③

해설 : ① 1:1, 양쪽 필수 관계를 시스템적으로 보장하므로 두 엔터티간의 EXCEPT 결과는 항상 공집합이다.
② 1:1, 양쪽 필수 관계를 시스템적으로 보장하므로 UNION을 수행한 결과는 회원기본정보의 전체건수와 동일하지만, UNION ALL을 수행하였으므로 결과건수는 회원기본정보의 전체건수에 2배가 된다.
④ 1:1, 양쪽 필수 관계를 시스템적으로 보장하므로 연산 수행결과는 같다.

87. C

해설 : SQL의 실행결과는 다음과 같다.

C3
A
C
B
D

88. ④

해설 : Oracle 계층형 질의에서 루트 노드의 LEVEL 값은 1이다.

89. ①

해설 : CONNECT BY 절에 작성된 조건절은 WHERE 절에 작성된 조건절과 다르다. START WITH 절에서 필터링된 시작 데이터는 결과목록에 포함되어지며, 이후 CONNECT BY 절에 의해 필터링 된다. 그러므로 매니저 사원번호가 NULL인 데이터는 결과목록에 포함되며, 이후 리커시브 조인에 의해 입사일자가 필터링 된다.
④번은 AND PRIOR 입사일자 BETWEEN '2013-01-01' AND '2013-12-31' 에 대한 결과이다.

90. ④

해설 : ④ 오라클 계층형 질의문에서 PRIOR 키워드는 SELECT, WHERE 절에서도 사용할 수 있다.

91. ①

해설 : 위의 결과는 중간 레벨인 도쿄지점(120)을 시작으로 상위의 전체 노드(역방향 전개)와 하위의 전체 노드(순방향 전개)를 검색하여 매출액을 추출하는 SQL이다. 부서 테이블의 전체 데이터를 보면 LEVLE은 1 ~ 3 까지 이지만 추출된 데이터의 LEVEL은 1과 2만 추출된 것으로 보면 중간 LEVEL에서 추출된 것을 짐작할 수 있다.

② 최상위 노드인 아시아지부(100)를 시작으로 하위의 모든 부서를 추출(순방향 전개)하므로 아래와 같은 결과가 추출된다.

부서코드	부서명	상위부서코드	매출액	LVL
100	아시아지부	NULL	NULL	2
110	한국지사	100	NULL	2
111	서울지점	110	1000	3
112	부산지점	110	2000	3
120	일본지사	100	NULL	2
121	도쿄지점	120	1500	3
122	오사카지점	120	1000	3
130	중국지사	100	NULL	2
131	베이징지점	130	1500	3
132	상하이지점	130	2000	3

③ 최하위 노드인 도쿄 지점(121)에서 상위의 모든 노드(역방향 전개)를 추출하게 되므로 아래와 같은 결과가 추출된다.

부서코드	부서명	상위부서코드	매출액	LVL
100	아시아지부	NULL	NULL	3
120	일본지사	100	NULL	2
121	도쿄지점	120	1500	1

④ WHERE 절의 서브쿼리를 보면 일본 지사(120)를 시작으로 역방향 전개하여 최상위 노드를 추출하여 다시 순방향 전개를 수행하고 있다. 이렇게 되면 ② 와 동일한 결과를 추출하게 된다.

92. ①

해설 : SELF JOIN은 하나의 테이블에서 두 개의 칼럼이 연관 관계를 가지고 있는 경우에 사용한다.

93. ③

해설 : ①는 일자별매출액에 일자별 매출 테이블과 동일하게 출력된다.

②,④는 작은 날짜쪽에 제일 큰 누적금액이 출력된다.

③은 일자별매출 테이블을 Self Join하여, A Alias 쪽에 먼저 읽혔다고 가정하면 다음처럼 데이터가 생성될 것이다.

1. A가 {2015.11.01. 1000} 일 때 B는 {2015.11.01. 1000}
2. A가 {2015.11.02., 1000} 일 때 B는 {{2015.11.01. 1000}, {2015.11.02. 1000}}
3. A가 {2015.11.03., 1000} 일 때 B는 {{2015.11.01. 1000}, {2015.11.02. 1000}, {2015.11.03. 1000}}

위의 Self Join은 Equi Join이 아닌 Range Join이므로 A의 레코드는 B의 레코드 수 만큼 증가하게 된다.(A * B) 그러므로 위의 3번의 경우 A는 B의 레코드 개수와 동일하게 되므로 SUM(매출금액)을 하면 3,000이 된다. 이런 식으로 A Alias의 모든 레코드 개수를 Scan하면 누적 값을 출력하게 된다.

94. ③

해설 : WHERE 절의 단일행 서브쿼리인 (SELECT D FROM DEPT WHERE E = 'i') 에 의해서 DEPT 테이블의 D 컬럼 값이 x인 행이 선택되고, D = (SELECT D FROM DEPT WHERE E = 'i') 조건에 의해 EMP 테이블의 (A=1, B=a), (A=2, B=a) 인 2건이 출력된다. 출력된 결과가 모두 UNIQUE하기 때문에 DISTINCT 연산자는 결과 건수에 영향을 주지 않는다.

95. ②

해설 : 다) 서브쿼리의 결과가 복수 행 결과를 반환하는 경우에는 IN, ALL, ANY 등의 복수 행 비교 연산자와 사용하여야 한다.

마) 다중 컬럼 서브쿼리는 서브쿼리의 결과로 여러 개의 컬럼이 반환되어 메인 쿼리의 조건과 비교되는데, SQL Server에서는 현재 지원하지 않는 기능이다.

96. ③

해설 : '현재 부양하는 가족들이 없는 사원들의 이름을 구하라'를 구현하는 방법은 가족 테이블에 부양사번이 없는 사원 이름을 사원 테이블에서 추출 하면 되고, SQL 문장으로 NOT EXISTS, NOT IN , LEFT OUTER JOIN을 사용하여 구현 할 수 있다.

1. NOT EXISTS

```
SELECT 이름
FROM   사원
WHERE NOT EXISTS (SELECT * FROM 가족 WHERE 사번 = 부양사번)
```

2. NOT IN

```
SELECT 이름
FROM   사원
WHERE 사번 NOT IN (SELECT 부양사번 FROM 가족)
```

3. LEFT OUTER JOIN

```
SELECT 이름
FROM   사원 LEFT OUTER JOIN 가족 ON (사번 = 부양사번)
WHERE 부양사번 IS NULL
```

97. ③

해설 : 위의 SQL은 약관항목 중 단 하나라도 동의를 하지 않은 회원을 구하는 SQL이다. HAVING 절에서 동의여부가 N인 데이터가 한 건이라도 존재하는 데이터를 추출한다.

①은 회원 테이블과 동의항목 테이블의 회원번호 컬럼으로 연관 서브쿼리를 수행하여 동의여부 컬럼의 값이 N인 데이터가 한 건이라도 존재하면 회원 데이터를 출력하게 된다.

②는 동의항목 테이블에서 동의여부가 N인 한 건이라도 존재하는 회원을 추출하여 회원테이블과 IN 연산을 수행한다.

③의 회원 테이블과 동의항목 테이블간에 회원번호 컬럼으로 연관 서브쿼리로 처리되어야 정상적으로 처리할 수 있다.

④는 HAVING절로 처리되던 조건을 WHERE절에 위치시켜 더 간편하게 Join으로 처리하였다. 또한 회원과 동의항목은 1:N 관계이므로 JOIN된 결과는 N건으로 발생됨에 따라 GROUP BY를 추가하여 중복을 제거 하였다.

98. ③

해설 : 이벤트 시작일자가 '2014.10.01.'과 같거나 큰 이벤트를 기준으로 단 한차례라도 이메일 발송이 누락된 회원을 추출하는 SQL문장이다.

㉡을 제거하고 ㉠의 EXISTS 연산자를 IN연산자로 변경하게 되면 회원별로 메일을 발송한 건수를 계산할 수 없으므로 원하는 결과를 추출할 수 없다.

GROUP BY 및 집계함수를 사용하지 않고 HAVING 절을 사용하였다고 하여 SQL문장이 오류가 발생하지는 않는다.

99. ②

해설 : ① 단일 행 서브쿼리의 비교연산자로는 =, <, <=, >, >=, <>가 되어야 한다. IN, ALL 등의 비교연산자는 다중 행 서브쿼리의 비교연산자 이다.

② 단일 행 서브쿼리의 비교연산자는 다중 행 서브쿼리의 비교연산자로 사용할 수 없지만, 반대의 경우는 가능하다.

③ 비 연관 서브쿼리가 주로 메인쿼리에 값을 제공하기 위한 목적으로 사용된다.

④ 메인 쿼리의 결과가 서브쿼리로 제공될 수도 있고, 서브쿼리의 결과가 메인쿼리로 제공될 수도 있으므로 실행 순서는 상황에 따라 달라진다.

100. ③

해설 : 2014년에 입사한 사원들의 사원, 부서 정보와 부양가족수를 추출하는 SQL이다.

SELECT 절에 사용된 서브쿼리는 단일행 연관 서브쿼리로 JOIN 으로도 변경이 가능하며, FROM 절에 사용된 서브쿼리는 Inline View 또는 Dynamic View 이고, WHERE 절에 사용된 서브쿼리는 다중행 연관 서브쿼리 이다.

③번 보기의 경우 이미 FROM절에 Inline View로 사원 테이블의 입사년도 조건을 명시하였으므로 WHERE 절의 EXISTS 조건은 부서와 사원 테이블간의 JOIN 조건에 의해 결과에 어떠한 영향도 미치지 못하므로 삭제되어도 무방하다.

101. ②

해설 : ① Inline View D 에서 평가결과 엔터티의 특정상품 및 평가항목에 대한 최종 평가회차가 아닌 전체 데이터 중 평가회차가 가장 큰 값을 가지고 JOIN을 수행하므로 원하는 결과가 아니다.

② 연관 서브쿼리를 활용하여 특정 상품, 평가항목별로 최종 평가회차와 Join을 수행하여 원하는 결과를 출력한다.

③ 특정 평가회차에 대한 결과가 아닌, 평가결과 엔터티의 평가회차, 평가등급, 평가일자 속성에 대해서 개별 MAX 값을 구하므로 원하는 결과가 아니다.

④ 특정 평가회차에 대한 결과가 아닌, 상품ID, 평가항목ID별로 개별 MAX값을 구하므로 원하는 결과가 아니다.

102. ③

해설 : ① 연관 서브쿼리를 활용한 UPDATE 에서 WHERE절은 UPDATE 대상이 되는 데이터의 범위를 결정하게 되는데, WHERE 절이 누락되어 부서의 모든 데이터가 UPDATE 대상이 되므로 부서코드 A007, A008을 제외한 모든 데이터가 NULL 값으로 변경된다.

② WHERE 절 조건이 부서임시가 아닌 부서 테이블이므로 A007, A008을 제외한 모든 데이터가 NULL 값으로 변경된다.

④ ①과 같은 사유로 부서코드 A007, A008을 제외한 모든 데이터가 NULL 값으로 변경된다. 또한 변경일자를 하드 코딩하는 것은 답이 될 수 없다.

103. ②

해설 : ② 뷰의 장점중 독립성은 테이블 구조가 변경되어도 뷰를 사용하는 응용 프로그램은 변경하지 않아도 된다.

104. ②

해설 : 조회 SQL 실행시 V_TBL은 뷰 스크립트로 치환되어 수행된다. 뷰 생성 스크립트에서 부여된 조건과 조회 SQL에서 부여된 조건 모두를 만족해야 한다.

105. ③

해설 : ROLLUP은 계층 구조를 가진 SUB TOTAL을 생성하는 함수로 나열된 컬럼의 순서가 변경되면 수행결과도 변경된다. 위의 SQL문장은 서비스ID에 대해서 가입일자별 가입건수 및 소계와 전체 가입건수를 구하되 Outer Join을 수행하였으므로 가입내역이 없는 서비스ID(004)에 대해서도 SUB TOTAL을 출력하고 있다.

①은 서비스ID 에 대해서 가입일자별 가입건수 및 소계와 전체 가입건수를 구한 것은 맞으나 LEFT OUTER JOIN이 아닌 INNER JOIN에 대한 결과로 서비스ID 004가 출력되지 않았다.

106. ②

해설 : 위의 결과 데이터는 지역에 대해서 월별 이용량 및 소계와 전체 이용량을 출력하였으므로, ROLLUP 함수를 활용할 수 있다. ROLLUP 집계 그룹 함수는 나열된 컬럼에 대해 계층 구조로 집계를 출력하는 함수로서 ROLLUP(A, B)를 수행하면 (A, B)별 집계, A별 집계와 전체 집계를 출력할 수 있다. ①번 보기의 경우 CASE 절의 GROUPING 함수의 사용이 잘못(0이 아닌 1이 되어야 함) 되었으며, ③번 보기처럼 CUBE를 사용하게 되면, 결합 가능한 모든 값에 대하여 다차원 집계를 생성하게 된다. ④번 보기처럼 GROUPING SETS를 사용하게 되면 계층구조 없이 지역에 대한 합계와 월별 합계를 각각 생성하게 된다.

107. ROLLUP

해설 : 위 SQL의 결과는 (구매고객, 구매월)별, 구매고객별 그리고 전체에 대한 구매건수와 구매금액을 출력한 결과이다. 집계에 계층 구조가 있으므로 나열된 컬럼에 대해 계층 구조로 집계를 출력하는ROLLUP을 사용하여 집계 SQL을 작성할 수 있다.

108. ④

해설 : ① CUBE, GROUPING SETS, ROLLUP 세가지 그룹 함수 모두 일반 그룹 함수로 동일한 결과를 추출할 수 있다.

② 함수의 인자로 주어진 컬럼의 순서에 따라 다른 결과를 추출하게 되는 그룹 함수는 ROLLUP 이며, 나열된 컬럼에 대해 계층 구조로 집계를 출력한다.

③ CUBE, ROLLUP, GROUPING SETS 함수들에 의해 집계된 레코드에서 집계 대상 컬럼 이외의 GROUP 대상 컬럼의 값은 NULL을 반환한다.

109. ②, ③

해설 : SQL의 결과를 보면 설비ID와 에너지코드의 모든 조합에 대하여 사용량합계를 추출하고 있다. CUBE 함수는 인수로 나열된 항목의 가능한 모든 조합에 대하여 GROUPING을 수행한다. 또한 GROUPING SETS은 사용자가 원하는 다양한 조합을 인수로 사용할 수 있다. 위 문제에서 ②번은 CUBE를 사용하였으므로 CUBE절에 나열된 컬럼의 모든 조합 즉, ((설비ID), (에너지코드), (설비ID, 에너지코드))에 대해 SUB TOTAL을 만들게 된다. ③번은 GROUPING SETS를 활용하여 ②번의 모든 조합을 직접 기술 하였다. ①, ④의 보기별 결과는 아래와 같다.

①

설비ID	에너지코드	사용량
1	바람	300
1	바람	300
1	바람	300
1	바람	300
1	바람	300
1	용수	200
1	용수	200
1	용수	200
1	용수	200
1	용수	200
1	전기	100
1	전기	100
1	전기	100
1	전기	100
1	전기	100
1	NULL	600
2	용수	300
2	용수	300
2	용수	300
2	용수	300
2	용수	300
2	전기	200
2	전기	200
2	전기	200
2	전기	200
2	전기	200
2	NULL	500
3	전기	300
3	전기	300
3	전기	300
3	전기	300
3	전기	300
3	NULL	300
NULL	바람	300
NULL	용수	500
NULL	전기	600
NULL		1400

④

설비ID	에너지코드	사용량
1	바람	300
1	용수	200
1	전기	100
1	NULL	600
2	용수	300
2	전기	200
2	NULL	500
3	전기	300
3	NULL	300
NULL	바람	300
NULL	용수	500
NULL	전기	600

110. ④

해설 : 집계 그룹 함수에는 ROLLUP, CUBE, GROUPING SETS 함수가 있다.
문제의 결과 데이터는 (자재번호별) SUB TOTAL과 (자재번호, 발주처별) SUB TOTAL을 출력하고 있다. GROUPING SETS 함수를 사용하여 입력된 인수들에 대한 개별 집계를 구할 수 있으며, CUBE 함수의 경우는 나열된 모든 인수의 결합 가능한 집계가 출력 된다.그러므로 위의 문제에서는 GROUP BY GROUPING SETS(자재번호, (발주처ID, 발주일자))가 되어야 한다. ①, ②, ③의 보기별 결과는 아래와 같다.

①

자재번호	발주처ID	발주일자	발주수량
1	1	20140102	100
1	1	20140103	200
1	발주처전체	발주일자전체	300
2	1	20140102	200
2	2	20140102	100
2	발주처전체	발주일자전체	300
3	1	20140103	100
3	2	20140103	200
3	발주처전체	발주일자전체	300
자재전체	1	20140102	300
자재전체	1	20140103	300
자재전체	2	20140102	100
자재전체	2	20140103	200
자재전체	발주처전체	발주일자전체	900

②

자재번호	발주처ID	발주일자	발주수량
1	1	20140102	100
1	1	20140103	200
1	1	발주일자전체	300
1	발주처전체	20140102	100
1	발주처전체	20140103	200
1	발주처전체	발주일자전체	300
2	1	20140102	200
2	1	발주일자전체	200
2	2	20140102	100
2	2	발주일자전체	100
2	발주처전체	20140102	300
2	발주처전체	발주일자전체	300
3	1	20140103	100
3	1	발주일자전체	100
3	2	20140103	200
3	2	발주일자전체	200
3	발주처전체	20140103	300
3	발주처전체	발주일자전체	300
자재전체	1	20140102	300
자재전체	1	20140103	300
자재전체	1	발주일자전체	600
자재전체	2	20140102	100
자재전체	2	20140103	200
자재전체	2	발주일자전체	300
자재전체	발주처전체	20140102	400
자재전체	발주처전체	20140103	500
자재전체	발주처전체	발주일자전체	900

③

자재번호	발주처ID	발주일자	발주수량
1	발주처전체	발주일자전체	300
2	발주처전체	발주일자전체	300
3	발주처전체	발주일자전체	300
자재전체	1	발주일자전체	600
자재전체	2	발주일자전체	300
자재전체	발주처전체	20140102	400
자재전체	발주처전체	20140103	500

111. ②

해설 : GROUPPING SETS 함수는 표시된 인수들에 대한 개별 집계를 구하는 기능을 하며, 위의 SQL은 (상품ID, 월)별 집계 데이터를 출력한다. 각 보기별 SQL은 아래와 같다.

① GROUPING SETS에 괄호를 사용하지 않아 월별과 상품ID별로 각각 집계되었다.

```
SELECT  상품ID, 월, SUM(매출액) AS 매출액
FROM    월별매출
WHERE   월 BETWEEN '2014.10' AND '2014.12'
GROUP BY GROUPING SETS(월, 상품ID);
```

③ GROUPING SETS에 월별, 상품ID별과 전체가 각각 집계되었다.

```
SELECT  상품ID, 월, SUM(매출액) AS 매출액
FROM    월별매출
WHERE   월 BETWEEN '2014.10' AND '2014.12'
GROUP BY GROUPING SETS(월, 상품ID, ());
```

④ GROUPING SETS에 (월, 상품ID)별, 월별로 집계되었다.

```
SELECT  상품ID, 월, SUM(매출액) AS 매출액
FROM    월별매출
WHERE   월 BETWEEN '2014.10' AND '2014.12'
GROUP BY GROUPING SETS((월, 상품ID), 월);
```

112. ③

해설 : 윈도우 함수는 결과에 대한 함수처리이기 때문에 결과 건수는 줄지 않는다.

113. ①

해설 : 위의 SQL은 고객별 매출액과 매출 순위를 구하되 동일 순위일 경우 중간 순위를 비워둔 데이터를 추출한다. 순위를 구하는 함수로는 RANK, DENSE_RANK, ROW_NUMBER 함수가 있다. RANK WINDOW FUNCTION은 동일 값에 대해서는 동일 순위를 부여하고 중간 순위는 비워 두지만, DENSE_RANK 함수는 동일 순위를 부여하되 중간 순위를 비우지 않는다. ROW_NUMBER 함수는 동일 값에 대해서도 유일한 순위를 부여한다.

114. ④

해설 : 게임상품별로 고객 목록을 추출하기위해서는 OVER절에 "PARTITION BY 게임상품ID"를 적용하여 게임상품별 활동점수로 순위가 추출될 수 있도록 하여야 한다.
RANK WINDOW 함수는 OVER절의 ORDER BY에 대한 결과에 따라 동일한 값을 동일한 등수로 처리함과 동시에 중간 순위를 비우는 반면, DENSE_RANK WINDOW 함수는 중간 순위를 비우지 않는다.

115. ③

해설 : ROW_NUMBER 함수는 ORDER BY절에 의해 정렬된 데이터에 동일 값이 존재하더라도 유일한 순위를 부여하는 함수로서 데이터 그룹 내에 유일한 순위를 추출할 때 사용할 수 있는 함수이다.
문제의 SQL은 추천경로별(PARTITION BY 추천경로)로 추천점수가 가장 높은(ORDER BY 추천점수 DESC) 데이터를 한건씩만 출력하지만,
①은 전체 데이터를 그대로 출력하였으며
②는 전체에서 추천점수가 가장 높은 데이터 한건만을 출력하였고
④은 추천경로별로 추천점수가 가장 낮은 데이터를 각 한건씩 출력하였다.

116. ③

해설 : GROUP BY 절의 집합을 원본으로 하는 데이터를 WINDOW FUNCTION과 함께 사용한다면 GROUP BY 절과 함께 WINDOW FUNCTION을 사용한다고 하더라도 오류가 발생하지 않으며, 유사개수 컬럼은 상품분류코드로 GROUPING된 집합을 원본집합으로 하여 상품분류코드별 평균상품가격을 서로 비교하여 현재 읽혀진 상품분류코드의 평균가격 대비 -10000 ~ +10000사이에 존재하는 상품분류코드의 개수를 구한 것이다.

117. ①

해설 : 안쪽 IN-LINE VIEW에 의해 아래와 같이 사원ID와 부서별 최고연봉이 결과로 생성되며

사원ID	최고연봉
001	3000
002	3000
003	4500
004	4500
005	4500
006	4500
007	4500

이를 다시 사원 테이블과 사원ID = 사원ID AND 최고연봉 = 연봉으로 JOIN을 하게 되면 부서별 최고연봉의 사원이 출력된다. 아래의 SQL로도 동일한 결과를 얻을 수 있다.

```
SELECT 사원ID, 사원명, 부서ID, 연봉
FROM  (SELECT 사원ID, 사원명, 부서ID, 연봉
             ,MAX(연봉) OVER(PARTITION BY 부서ID) AS 최고연봉
       FROM  사원)
WHERE 연봉 = 최고연봉
```

118. ①

해설 : LAG 함수는 현재 읽혀진 데이터의 이전 값을, LEAD 함수는 이후 값을 알아내는 함수이다.
위의 SQL에서 각 레코드별 FLAG1, FLAG2의 값은 다음과 같으며,
메인 쿼리의 WHERE절이 FLAG1 = 0 OR FLAG2 = 0 이므로 1,4,5,6번째의 행이 출력된다.

ID	START_VAL	END_VAL	FLAG1	FLAG2
A	10	14	0	1
A	14	15	1	1
A	15	15	1	1
A	15	18	1	0
A	20	25	0	1
A	25		1	0

119. ㉠ : GRANT
㉡ : REVOKE

해설 : GRANT 명령은 DBMS 사용자에게 권한을 부여할 때 사용하며, REVOKE 명령은 부여된 권한을 회수할 때 사용한다.

120. ④

해설 : 권한을 부여하는 명령어는 GRANT이며, WHERE 조건의 데이터를 찾기 위한 SELECT 권한과 데이터 변경을 위한 UPDATE 권한이 필요하다.

121. ROLE

해설 : ROLE은 많은 DBMS사용자에게 개별적으로 많은 권한을 부여하는 번거로움과 어려움을 해소하기 위해 다양한 권한을 하나의 그룹으로 묶어놓은 논리적인 권한의 그룹이다.

122. ①, ③

해설 : 1. Lee: GRANT SELECT, INSERT, DELETE ON R TO Kim WITH GRANT OPTION;
→ Kim에게 테이블 R에 SELECT, INSERT, DELETE 권한을 주면서, Kim 이 다른 유저에게 테이블 R에 동일한 권한을 줄 수 있다.
2. Kim: GRANT SELECT, INSERT, DELETE ON R TO Park;
→ Kim이 테이블 R에 Lee에게 받은 권한을 Park에게 준다.
3. Lee: REVOKE DELETE ON R FROM Kim;
→ Kim에서 테이블 R의 DELETE 권한을 취소한다.
4. Lee: REVOKE INSERT ON R FROM Kim CASCADE;
→ Kim과 Park에서 INSERT 권한을 취소한다. WITH GRANT OPTION으로 Kim으로부터 받은 Park의 권한은 CASCADE 명령어로 받은 권한을 취소 할 수 있다.

123. ③

해설 : PL/SQL로 작성된 Procedure, User Defined Function은 작성자의 기준으로 트랜젝션을 분할할 수 있으며, 또한 프로시저 내에서 다른 프로시저를 호출할 경우에 호출 프로시저의 트랜젝션과는 별도로 PRAGMA AUTONOMOUS_TRANSACTION을 선언하여 자율 트랜젝션 처리를 할 수 있다.

124. ③

해설 : PL/SQL 에서는 동적 SQL 또는 DDL 문장을 실행할 때 EXECUTE IMMEDIATE를 사용하여야 한다. ② 번은 ROLLBACK이 가능하도록 삭제하는 것이 아니므로 옳은 답이 아니다.

125. ④

해설 : Stored Module(ex: PL/SQL, LP/SQL, T-SQL)로 구현 가능한 기능은 ①,②,③ 세 가지이며, ④ 데이터의 무결성과 일관성을 위해서 사용자 정의 함수를 사용하는 것은 트리거의 용도이다.

126. ③

해설 : Trigger는 Procedure와 달리 Commit 및 Rollback 과 같은 TCL을 사용할 수 없다.

127. ④

해설 : TRIGGER는 테이블과 뷰, 데이터베이스 작업을 대상으로 정의할 수 있으며, 전체 트랜잭션 작업에 대해 발생되는 TRIGGER와 각 행에 대해서 발생되는 TRIGGER가 있다.

제3장. SQL 최적화 기본 원리

128. CBO, 비용기반 옵티마이저, Cost Based Optimizer

129. ④

해설 : 실행계획은 예상 정보이다. 실제 처리 건수는 트레이스 정보를 통해서 알 수 있다.

130. ㉠ : 3, ㉡ : 4, ㉢ : 2

해설 : 실행계획을 읽는 순서는 위에서 아래로, 안에서 밖으로 읽는다.
그러므로 3 → 4 → 2 → 6 → 5 → 1 순으로 수행된다.

131. ③

해설 : 실행계획 즉, 실행방법이 달라진다고 해서 결과가 달라지지는 않는다.

132. ②, ④

해설 : SQL 처리 흐름도는 SQL 실행계획을 시각화해서 표현한 것이다. SQL 처리 흐름도만 보고 실행 시간을 알 수는 없다.

133. ①, ④

해설 : 규칙기반 옵티마이저에서 제일 낮은 우선순위는 전체 테이블 스캔이고, 제일 높은 우선순위는 ROWID를 활용하여 테이블을 액세스하는 방법이다.
SQL 처리 흐름도는 SQL문의 처리 절차를 시각적으로 표현한 것으로, 인덱스 스캔 및 전체 테이블 스캔 등의 액세스 기법을 표현할 수 있으며, 성능적인 측면도 표현할 수 있다.
인덱스 범위 스캔은 결과 건수만큼 반환하지만, 결과가 없으면 한 건도 반환하지 않을 수 있다.

134. ②

해설 : 기본 인덱스(Primary Key)는 UNIQUE & NOT NULL의 제약조건을 가진다.
보조 인덱스는 UNIQUE 인덱스가 아니라면 중복 데이터의 입력 가능하며, 자주 변경되는 속성을 인덱스로 선정할 경우 UPDATE, DELETE 성능에 좋지 않은 영향을 미치므로 인덱스 후보로 적절하지 않다.

135. ②, ④

해설 : 테이블의 전체 데이터를 읽는 경우는 인덱스를 사용하지 않는 FTS를 사용한다.
인덱스는 조회만을 위한 오브젝트이며, 삽입, 삭제, 갱신의 경우 오히려 부하를 가중한다.
Balance Tree는 관계형 데이터베이스에서 가장 많이 사용되는 인덱스이다.
인덱스가 존재하는 상황에서 데이터를 입력하면, 매번 인덱스 정렬이 일어나므로 데이터 마이그레이션 같이 대량의 데이터를 삽입할 때는 모든 인덱스를 제거하고, 데이터 삽입이 끝난 후에 인덱스를 다시 생성하는 것이 좋다.

136. ②

137. ③, ④

해설 : ① 인덱스를 생성할 때 정렬 순서를 내림차순으로 하면 내림차순으로 정렬된다.
② 비용기반 옵티마이저는 SQL을 수행하는데 있어 소요되는 비용을 계산하여 실행계획을 생성하므로 인덱스가 존재하더라도 전체 테이블 스캔이 유리하다고 판단할 수도 있다.
③ 규칙기반 옵티마이저의 규칙에 따라 적절한 인덱스가 존재하면 전체 테이블 스캔보다는 항상 인덱스를 사용하려고 한다.
④ 인덱스 범위 스캔은 결과 건수만큼 반환하지만, 결과가 없으면 한 건도 반환하지 않을 수 있다.

138. ①, ③

해설 : ② REGIST_DATE 조건이 범위 조건이고 DEPTNO 컬럼이 후행 컬럼이므로 효율적인 조건 검색을 할 수 없다.
④ b*tree index는 일반적으로 테이블 내의 데이터 중 10%이하의 데이터를 검색할 때 유리하다.

139. ④

해설 : 인덱스를 스캔하여 테이블로 데이터를 찾아가는 방식이 랜덤 엑세스인데, 이러한 랜덤 엑세스의 부하가 크기 때문에 매우 많은 양의 데이터를 읽을 경우에는 인덱스 스캔보다 테이블 전체 스캔이 유리할 수도 있다.

140. ②

해설 : 대량의 데이터를 조회하는 경우 인덱스를 이용한 조회보다는 테이블 전체 스캔 방식으로 조회 하는것이 더 빠를 수도 있으며, 인덱스를 구성하는 컬럼들의 순서는 데이터 조회 시 성능적인 관점에서 매우 중요한 역할을 한다. 또한 인덱스를 구성하는 컬럼 이외의 데이터가 UPDATE될 때는 인덱스로 인한 부하가 발생하지 않는다.

141. ②

해설 : NL Join은 데이터를 집계하는 업무 보다는 OLTP의 목록 처리 업무에 많이 사용된다. DW 등의 데이터 집계 업무에서 많이 사용되는 Join 기법은 Hash Join 또는 Sort Merge Join 이다.

142. ③

해설 : 소트 머지 조인(Sort Merge Join)을 수행하기에 두 테이블이 너무 커서 소트(Sort) 부하가 심할 때는 Hash Join이 유용하다.

143. ④

해설 : EXISTS 절은 실행계획상에 주로 SEMI JOIN으로 나타난다. NESTED LOOP, HASH, SORT MERGE의 SEMI JOIN이 모두 나타날 수 있지만, 위의 인덱스 정보와 SQL을 볼 때 HASH SEMI JOIN 보다는 NESTED LOOP SEMI JOIN 이 나타날 가능성이 가장 크다.

144. ④

해설 : EQUI JOIN에서만 동작하는 Join 방식은 Hash Join 이다. Sort Merge Join은 Non-EQUI JOIN 조건에서도 사용할 수 있다.

145. ④

해설 : 유니크 인덱스를 활용하여 수행시간이 적게 걸리는 소량 테이블을 조인할 때는 NL 조인이 적합하다.

146. ①

해설 : ② Sort Merge Join은 비 동등 Join(Not Equi Join)에서도 사용할 수 있다.

③ Hash Join은 행의 수가 작은 테이블을 선행 테이블로 선택하는 것이 유리하다.

④ Hash Join은 Sort Merge Join보다 일반적으로 더 우수한 성능을 보이지만, Join 대상 테이블이 Join Key 컬럼으로 정렬되어 있을 때는 Sort Merge Join이 더 우수한 성능을 낼 수도 있다.

과목 Ⅲ. SQL 고급활용 및 튜닝

제1장. 아키텍처 기반 튜닝 원리

1. ①

해설 : 다중 사용자 환경에서 서버와 모든 클라이언트 간 연결상태를 지속하면 서버 자원을 낭비하게 된다. 그렇다고 SQL을 수행할 때마다 연결 요청을 반복하면 서버 프로세스(또는 쓰레드)의 생성과 해제도 반복하므로 성능에 좋지 않다. 따라서 OLTP성 애플리케이션에선 Connection Pooling 기법의 활용이 필수적이다.

2. ③

해설 : 익스텐트 내 블록들은 서로 인접하지만, 익스텐트끼리 서로 인접하지는 않는다.

3. ①

해설 : Log Force at commit은, 로그 버퍼를 주기적으로 로그 파일에 기록하되 늦어도 커밋 시점에는 반드시 기록해야 함을 뜻한다.

Fast Commit은, 사용자의 갱신내용이 메모리상의 버퍼 블록에만 기록된 채 아직 디스크에 기록되지 않았지만 Redo 로그를 믿고 빠르게 커밋을 완료하는 것을 말한다.

Delayed Block Cleanout은 오라클만의 독특한 메커니즘으로서, 변경된 블록을 커밋 시점에 바로 Cleanout(로우 Lock 정보 해제, 커밋 정보 기록)하지 않고 그대로 두었다가 나중에 해당 블록을 처음 읽는 세션에 의해 정리되도록 하는 것을 말한다.

4. ④

해설 : Table Full Scan한 데이터 블록은 LRU end에 위치하기 때문에 버퍼 캐시에 오래 머물지 않는다.

5. ①, ④

해설 : Response Time = Service Time + Wait Time = CPU Time + Queue Time

6. ①

해설 : SQL 커서의 공유와 재사용성에 관한 문제다. 캐싱된 SQL 커서는 반복 재사용할 수 있을 뿐만 아니라 여러 세션 간에 공유될 수 있다.

7. ①

해설 : 캐시에서 SQL과 실행계획을 식별하는 식별자는 SQL 문장 그 자체다. 따라서 옵티마이저는 문자 하나만 달라도 서로 다른 SQL로 인식해 각각 하드파싱을 일으키고 다른 캐시 공간을 사용한다.

1~3번 SQL은 SQL Text가 서로 다르다. SQL Text가 달라 하드파싱은 각각 일어나지만, 의미상 전혀 차이가 없으므로 실행계획은 같다.

8. ③

해설 : 사용자의 입력 조건이 다양해서 조건절을 동적으로 구성하더라도 조건절 비교 값만큼은 바인드 변수를 사용하려고 노력해야 한다.

9. ①, ②

해설 : Dynamic SQL 방식으로 코딩했지만, 바인드 변수를 사용했으므로 불필요한 하드파싱을 많이 일으킨다고 말하기는 어렵다.
바인드 변수를 사용했으므로 컬럼 히스토그램은 활용하지 못하지만, 레코드 건수, 컬럼 값의 종류(NDV), Null 값 개수 등을 활용해 실행계획을 수립한다.

10. ④

해설 : 바인드 변수를 사용하기만 하면 루프 내에서 반복 수행되는 SQL이더라도 캐싱된 SQL을 공유할 수 있다. 조건절을 바꾸지 않고 반복 수행하는 경우도 있으므로 3번 설명은 옳지 않다.
Static SQL은 PreCompile 과정을 거치므로 런타임 시 안정적인 프로그램 Build가 가능하다. 그리고 Dynamic SQL을 사용하면 애플리케이션 커서 캐싱이 작동하지 않는 경우가 있다. 따라서 Static SQL을 지원하는 개발환경에선 가급적 이 방식을 사용하는 것이 좋다.

11. ①, ③

해설 : SELECT 문장을 수행할 땐Parse, Execute, Fetch 순으로 Call이 발생한다.
Group By 결과집합을 만드는 과정에서의 I/O는 첫 번째 Fetch Call 단계에서 일어난다.

12. ②

해설 : 제시된 Call Statistics만으로는 Order By 또는 Group By 연산의 포함여부를 판단할 수 없다.

13. ②

해설 : 모든 데이터 처리가 서버 내에서 이루어지는 프로그램에선 부분범위처리에 의한 성능 개선 효과가 나타나지 않는다.

14. ②, ③

해설 : SQL을 포함하지 않는 형태의 사용자 정의 함수라도 문맥전환 (context switch)에 의한 부하가 발생하므로 성능저하가 발생한다.
작은 코드 테이블로부터 코드명을 가져오는 경우 사용자 정의 함수보다는 스칼라 서브쿼리를 사용하여 캐싱효과를 누리는 것이 성능상 유리하다.

15. ①

해설 : 함수를 실행할 때마다 컴파일하지는 않는다.

16. ④

해설 : 가입일자, 고객명을 선두로 갖는 인덱스를 사용한다면 '가' SQL은 인덱스만 읽고 처리를 완료하므로 블록 I/O가 더 적게 발생한다.

가입일자만으로 구성된 단일 컬럼 인덱스를 사용한다면 두 SQL 모두 테이블 액세스가 불가피하므로 블록 I/O는 똑같다.

'가' SQL은 소트 공간에 고객명만 저장하면 되지만, '나' SQL은 모든 컬럼을 저장해야 하므로 더 많은 소트 공간을 사용한다.

'나' SQL을 수행하면 조건절을 만족하는 모든 컬럼을 클라이언트에게 전송해야 하므로 네트워크 트래픽이 더 많이 발생한다.

17. ③

해설 : 변경이 거의 없는 테이블까지 매일 통계정보를 수집할 필요는 없다.

18. ②

해설 : Direct Path I/O는 일반적으로 병렬 쿼리로 Full Scan을 수행할 때 발생한다.

19. ④

해설 : Multiblock I/O 방식으로 읽더라도 Extent 범위를 넘어서까지 읽지는 않는다. 따라서 작은 Extent로 구성된 테이블을 Full Table Scan하면 I/O Call이 더 많이 발생한다.

반면, 인덱스를 통한 테이블 액세스 시에는 Single Block I/O 방식을 사용하므로 Extent 크기가 I/O Call 횟수에 영향을 미치지 않는다.

20. ③

해설 : Sequential I/O 방식은 테이블이나 인덱스를 스캔할 때 사용한다. Random I/O 방식은 인덱스를 스캔하면서 테이블을 액세스할 때 사용한다.

요즘은 NAS 서버나 SAN가 보편적으로 사용되기 때문에 네트워크 속도가 I/O 성능에 큰 영향을 미친다.

RAC 같은 클러스터링 데이터베이스 환경에선 메모리도 I/O 성능에 영향을 미친다.

제2장. Lock과 트랜잭션 동시성제어

21. ③

해설 : for update 구문을 반드시 사용해야 할 경우가 많은데, 성능을 이유로 이를 사용 못하게 하면 데이터 정합성을 해칠 수 있다. 성능보다 중요한 것은 데이터 정합성이다.
nowait이나 wait 옵션을 잘 활용하면 select for update 문장을 통해 오히려 동시성을 높일 수도 있다.

22. ④

해설 : SQL Server에서 Share Lock과 Exclusive Lock 호환되지 않으므로 1번 SELECT문에는 블록킹이 발생할 수 있다.
4번 INSERT문은 문제의 UPDATE문과 DEPTNO가 서로 다르고 PK 중복도 없으므로 블록킹 없이 진행된다.

23. ①, ②

해설 : READPAST는 Lock이 걸린 행은 읽지 않고 건너뛰도록 하는 힌트다.
TABLOCK은 테이블 레벨 Lock을 설정하고자 할 때 사용하는 힌트다.

24. ③

해설 : 테이블 Lock(=TM Lock)이 Exclusive 모드이므로 Append 모드로 입력한 ③번 SQL 실행 후 Lock 발생현황을 모니터링한 결과이다.

25. ②

해설 : Update 문은 TM Lock은 호환성이 있는 Row-X (SX)모드로, TX Lock은 호환성이 없는 Exclusive 모드로 Lock을 획득한다.

26. ④

해설 : 트랜잭션의 주요 특징은 원자성(Atomicity), 일관성(Consistency), 격리성(Isolation), 영속성(Durability)이며, 영문 첫 글자를 따서 'ACID'라고 부른다.
가.는 원자성, 나.는 일관성, 다.는 격리성, 라.는 영속성에 대한 설명이다.

27. ④

해설 : 100번 세션의 트랜잭션 격리성 수준을 Serializable Read로 설정하였으므로 쿼리(1)은 5건이 출력되고 쿼리(2)는 COMMIT이 수행되어 트랜잭션 격리성 수준이 Read Comitted이므로 200번 세션의 결과가 반영되어 1건이 출력된다.

28. ②

해설 : 대부분 DBMS가 Read Committed를 기본 트랜잭션 격리성 수준으로 채택하고 있으므로 Dirty Read가 발생할까 걱정하지 않아도 되지만, Non-Repeatable Read, Phantom Read 현상에 대해선 세심한 주의가 필요하다.

29. ③

해설 : 트랜잭션 격리성 수준(Transaction Isolation Level)을 상향 조정할수록 일관성은 높아지지만 동시성은 낮아진다.

30. 2000, 3000

해설 : Oracle은 Update 문장이 시작되는 시작을 기준으로 갱신 대상 레코드를 식별하므로 TX2 트랜잭션의 update는 실패한다. 따라서 TX1 트랜잭션의 결과가 7788 사원의 최종 결과가 된다.
SQL Server에서 TX2 트랜잭션은 TX1 트랜잭션이 완료될 때까지 기다린다. TX1이 끝났을 때 7788 사원의 sal 값은 2000이므로 TX2 트랜잭션이 정상적으로 진행해 값을 3000으로 바꾼다.

31. ③

해설 : MVCC 모델은 문장 수준의 읽기 일관성을 완벽히 보장하지만, 트랜잭션 수준의 읽기 일관성을 보장하지는 않는다.

제3장. 옵티아미저 원리

32. ②

해설 : 데이터베이스 Call은 옵티마이저가 수립한 실행계획에 따라 SQL을 수행하는 과정에, 또는 옵티마이저에게 실행계획을 수립해 달라고 요청하는 과정에 발생한다.
1번은 전통적인 I/O 비용 모델에서 사용하는 비용 개념이다.
3번과 4번은 최신의 CPU 비용 모델에서 사용하는 비용 개념이다. CPU 비용 모델에서도 I/O는 가장 중요한 비용 요소다.

33. ④

해설 : 직급의 종류 개수(NDV, Number Of Distinct Value)는 CBO가 사용하는 가장 대표적인 통계정보다.

34. ④

해설 : 전체범위 최적화는 빠른 Response Time보다 Throughput 중심으로 최적화를 시행한다.

35. ①, ④

해설 : Selectivity = 1 / NDV
Cardinality = 총 로우 수 × Selectivity = 총 로우 수 / NDV

36. ②

해설 : 통계정보 수집 시 시스템에 많은 부하를 주므로 대용량 테이블에는 흔히 표본 검사 방식을 사용한다. 표본 검사 시, 가능한 한 적은 양의 데이터를 읽고도 전수 검사할 때의 통계치에 근접하도록 해야 한다.

37. ②

해설 : 3번은 View Merging이 발생하기 때문에 함수호출 횟수가 1번과 같다.
4번은 스칼라 서브쿼리의 캐싱효과를 이용해 함수호출 횟수를 줄이려 했지만, 상품 테이블에서 상품코드는 Unique하기 때문에 캐싱 효과가 없고 오히려 캐시를 탐색하는 비용만 추가된다.
2번은 ROWNUM을 이용해 View Merging을 방지했으므로 성능 개선에 도움이 된다.

38. ②

해설 : 뷰(View) 안에 rownum을 사용하면 뷰 머징(View Merging)을 방지하는 효과가 나타난다.

39. ②

해설 : Group By를 포함한 뷰는 자주 Merging이 발생한다.

40. ③

해설 : deptno 조건이 인라인 뷰 안으로 파고 들어간다. 따라서 다음과 같은 SQL을 기준으로 문제를 풀면 쉽다.

```
select  deptno, empno, ename, job, sal, sal * 1.1 sal2, hiredate
from    emp
where   job = 'CLERK'
and     deptno = 30
union   all
select  deptno, empno, ename, job, sal, sal * 1.2 sal2, hiredate
from    emp
where   job = 'SALESMAN'
and     deptno = 30
```

41. ①

해설 : ①번 방식으로 처리할 경우, Cartesian Product가 발생해 결과가 틀릴 수 있다.

제4장. 인덱스와 조인

42. ④

해설 : ERD에서 주문과 고객은 M:1 관계이고, 주문 테이블 고객번호는 Null 값을 허용하지 않는다. 뿐만 아니라 SQL문에서 조인 조건 외에 어디서도 고객 테이블을 참조하지 않고 있다. 따라서 고객과의 조인은 불필요하다.

43. ②

해설 : 인덱스 정렬 순서 상 deptno = 20 and sal = 2000 조건을 만족하는 첫 번째 레코드에서부터 수평적 탐색을 시작한다. comm 조건이 less than or equal 인 점에 주목하자.
comm 조건이 more than or equal 이면, deptno = 20 and sal = 2000 and comm = 100 조건을 만족하는 첫 번째 레코드에서부터 스캔을 시작한다.

44. ④

해설 : Index Range Scan이 가능하려면 인덱스 선두 컬럼이 조건절에 사용되어야 한다.

45. ④

해설 : Index Skip Scan을 활용하려면 인덱스 선행칼럼이 누락됐거나 부등호, between, like 같은 범위검색 조건이어야 한다.
1번의 경우, 인덱스 선두 칼럼이 사용됐지만 성별 칼럼이 조건에서 누락됐으므로 Index Skip Scan 활용이 가능하다.

46. ②, ④

해설 : sal 컬럼을 선두로 갖는 인덱스가 없으므로 range scan은 사용할 수 없다.
fast full scan은 인덱스에 포함된 칼럼으로만 조회할 때 사용할 수 있다.

47. ①

해설 : 성별처럼 Distinct Value 개수가 적은 컬럼에 사용할 때 B*Tree인덱스보다 훨씬 적은 공간을 차지하기 때문에 유리하지만, 테이블 Random 액세스 발생 측면에선 그다지 잇점이 없다.

48. ②

해설 : 계좌번호 = , 지점번호 between 조건만으로도 충분히 스캔범위를 줄일 수 있다.

49. ③

해설 : 배송상태 = 'ING' 조건절이 없더라도 고객ID, 연락처, 고객등급을 읽기 위해 테이블을 액세스해야 한다.

50. ①, ③

해설 : 1번과 같은 인덱스 구성에서 SQL을 오른쪽과 같이 변환하면 인덱스에서 가입일자 like 조건에 해당하는 범위를 2번 스캔하게 된다. 고객등급을 테이블에서 필터링하므로 테이블 Random 액세스량도 2배 증가한다.

2번과 같은 인덱스 구성에서는 SQL을 변환하기 전후 블록 I/O 발생량이 거의 동일하다.

3번과 같은 인덱스 구성에서는 고객등급을 인덱스에서 필터링하므로 테이블 Random 액세스량은 늘지 않지만, 인덱스에서 같은 범위를 2번 스캔하므로 블록 I/O가 오히려 늘어난다.

4번과 같은 인덱스 구성에서는 SQL을 변환한 후 블록 I/O 발생량이 크게 줄어든다.

51. ②, ④

해설 : TAB1_X01 인덱스의 클러스터링 팩터는 매우 나쁜 상태다. 테이블 액세스 횟수만큼 블록 I/O가 발생한 것을 통해 이를 알 수 있다.

511개 인덱스 블록을 스캔했는데, 이것은 인덱스 스캔 과정에서 얻는 266,476개 레코드 수에 비하면 그리 큰 수치가 아닌다. 비효율이 크지 않다는 뜻이다. TAB1_X01 인덱스 컬럼 순서를 조정하면 블록 I/O가 약간 감소할 수 있을지 모르지만, 전체 성능에 미치는 영향은 크지 않을 것이다.

52. ③

해설 : 인덱스를 설계하는 가장 중요한 선택 기준은, 조건절에 항상 또는 자주 사용되는지 여부와 '=' 조건으로 자주 조회되는지 여부다. 인덱스 선두 칼럼이 일단 조건절에 사용돼야 해당 인덱스가 사용될 수 있으므로 전자가 특히 중요하다.

53. ③

해설 : 상품코드는 인덱스 구성컬럼이 아니므로 주문 테이블 액세스(ID=1) 단계의 필터조건이다.

54. ②

해설 : 고객_IDX가 연령+고객등급 구성이라면 순서를 바꿨을 때 액세스량 개선 가능
한 달간 주문 건수는 평균 50만 건이므로 조인 순서 변경은 성능에 도움이 되지 않는다.

55. ③

해설 : 직업코드 조건으로 인덱스를 Range Scan하려면 직업코드를 선두로 갖는 인덱스여야 한다. 고객_X01 인덱스가 여기에 해당한다.

인덱스 구성컬럼 중 하나라도 NOT NULL 컬럼이면, IS NULL 조회에 인덱스를 사용할 수 있다. 따라서 직업코드 IS NULL 조회에 고객_X01 인덱스를 사용할 수 있다.

56. ①, ③

해설 : 테이블 Alias가 있은 상황에선 반드시 Alias를 사용해야 한다.

57. ②, ③

해설 : 해시 조인할 때는 작은 집합으로 해시 테이블을 생성하는 것이 유리하다. OLTP 환경에서 수행빈도가 아주 높은 쿼리를 해시 조인으로 처리하면 CPU Usage가 높아질 수 있다.

58. ④

해설 : Result Cache가 아니라 PGA에 캐싱한다.

59. ③

해설 : 대량 집합을 기준으로 NL 조인하면 많은 랜덤 I/O가 발생한다.

SQL1에서 판매시작일자 조건을 만족하는 상품 건수가 적다면, 일별매출보다 상품 테이블을 먼저 드라이빙하는 게 유리할 수 있다.

SQL2에서 v_판매시작일자 변수에 최근일자를 입력하면 판매시작일자 조건을 만족하는 상품 건수가 적어지므로 그만큼 비효율이 커진다. 기준일자 Between 조건에 해당하는 많은 일별매출 데이터를 읽고 Group By 처리까지 마쳤는데, 상품 테이블과 조인하는 과정에 많은 데이터가 필터링되기 때문이다.

60. ②

해설 : Hash 조인은 "=" 조인만 가능함

61. ③

해설 : 선분이력 각각을 between으로 조회하면 된다.

62. ③

해설 : 인덱스 파티션 키가 인덱스 선두 컬럼이어야 한다는 제약은 Global 파티션 인덱스에 해당한다. Local 파티션이든 Global 파티션이든, 인덱스 파티션 키가 조건절에 없으면 인덱스 사용 시 비효율이 발생한다. 문제를 풀 때, Local 파티션 인덱스의 경우, 테이블 파티션 키가 인덱스 파티션 키가 된다는 점에 주의하기 바란다.

63. ①, ②

해설 : List 파티셔닝과 Range 파티셔닝 모두 가능하다. Range 파티셔닝의 경우, 예를 들어, 매년 1~3, 4~6, 7, 8, 9, 10~12월 6개로 파티셔닝하면 된다.

Hash 파티셔닝은 정해진 파티션 개수로 파티션 키 값에 따라 DBMS가 기계적으로 분할 저장하기 때문에 월별 매출 특성을 고려한 파티셔닝을 하기가 곤란하다.

64. ③

해설 : 칼럼끼리 연산할 때 null을 포함하면 결과는 null이다.

레코드끼리 연산할 때 null을 포함하면 결과가 null이 아니며, 이유는 null을 연산에서 제외하기 때문이다.

65. ④

해설 : 윈도우 함수가 사용된 SQL 실행계획에는 WINDOW SORT 오퍼레이션이 나타난다.

66. ②

해설 : union all은 정렬 오퍼레이션을 발생시키지 않는다.
④번 option 구문은 오라클 ordered use_merge에 해당하는 MS-SQL Server 힌트다. Sort Merge 조인이므로 정렬 오프레이션이 발생한다.

67. ④

해설 : union을 union all로 대체하려면 아래 두 조건을 만족해야 한다.
1번은 위아래 두 집합이 서로 배타적이어야 한다.
2번은 위아래 각 집합에 중복 값이 없어야 한다.
3번은 위아래 두 집합이 서로 배타적이지만, 각 집합에 중복 값이 있을 수 있다.
4번은 PK 컬럼을 포함하기 때문에 모든 레코드가 완전 배타적이다.

68. ①

해설 : ①번 방식으로 처리할 경우, 결과가 틀릴 수 있다.

69. ③

해설 : 1번은 Random 액세스 방식이므로 전체 고객을 대상으로 조회할 때 비효율적이다. 2번은 고객변경이력 테이블을 2번 액세스하는 비효율이 있다. 4번은 Top-N 쿼리 알고리즘이 작동하지 않아 3번에 비해 소트 부하가 더 크다.

제5장. 고급 SQL 튜닝

70. ③

해설 : INSERT ALL 구분에 APPEND 힌트를 사용하면 모든 테이블에 Exclusive 모드의 TM 락이 설정된다.

SID	TYPE	MODE_HELD	MODE_REQD	USN/TABLE	SLOT	SQN	BLOCKING
100	TM	Exclusive	None	주식월별시세			<<<<<
100	TM	Exclusive	None	선물월별시세			
100	TX	Exclusive	None	9	4	1421	
200	TM	None	Exclusive	주식월별시세			

71. ①

해설 : Append 모드로 INSERT하면 Exclusive 모드 테이블 Lock이 걸린다.

72. ②

해설 : nologging 모드는 INSERT문일 때만 기능이 작동한다.

73. ③

해설 : (A)의 서브쿼리를 세미조인 방식으로 변경해도 조인에 참여하는 테이블이 더 많으므로 (B)보다 I/O가 더 많이 발생한다.

74. ③

해설 : Oracle에서 nologging 기능은 insert 문장에서만 효과가 있다.

75. ②

해설 : 테이블이나 인덱스를 파티셔닝하면 저장 효율은 오히려 나빠질 수 있다.

76. ③

해설 : 파티션 칼럼에 대한 검색조건을 바인드 변수로 제공하더라도 Partition Pruning은 작동한다.

77. ①, ②

해설 : ③은 파티션 키 칼럼을 가공했으므로 모든 파티션을 Full Scan하게 된다.
④은 주문 테이블이 월단위로 파티션돼 있는데 일 단위로 조회 조건을 제공하므로 각 월에 속한 일자 개수만큼 파티션 Full Scan을 반복하게 된다.

78. ①

해설 : Local Prefixed 파티션 인덱스는 인덱스 선두 컬럼이 파티션 키인 경우를 말한다.
거래일시 기준으로 파티셔닝된 테이블에 인덱스를 Local 파티셔닝한다면 인덱스 파티션 키도 거래일시가 된다.

79. ①, ④

해설 : Global 파티션 인덱스인 거래_IDX1의 파티션 키는 거래일자이고 파티션 키가 인덱스 선두 컬럼이므로 Prefixed 파티션이다.
거래_IDX2는 Local 인덱스이므로 파티션 키는 거래일자가 된다. 파티션 키가 인덱스 선두 컬럼이 아니므로 Nonprefixed 파티션이다.

80. ①

해설 : 같은 시간대에 수많은 프로그램이 집중적으로 수행되면 총 수행시간이 더 늘어난다. 자원(CPU, Memory, Disk 등)과 Lock(Latch와 같은 내부 Lock까지 포함)에 대한 경합이 발생하면서 프로세스가 실제 일한 시간보다 대기하는 시간이 더 많아지기 때문이다.

81. ④

해설 : 테이블을 nologging 모드로 바꾸면 Redo Log가 생성되지 않도록 할 수 있지만, 이 기능은 append 또는 parallel 힌트를 사용해 Direct Load Insert 할 때만 작동한다.

82. ③

해설 : 주문 테이블이 고객번호 기준으로 Hash 서브 파티셔닝 돼 있으므로 Partial Partition Wise Join을 활용하는 것이 유리하다.

SQL 전문가 실기문제 정답 및 해설

실기문제 1 – 정답

① 윈도우 함수를 이용한 방식

```
select 지점, 판매월
     , 매출
     , sum(매출) over (partition by 지점 order by 판매월) 누적매출
from   월별지점매출;
```

또는

```
select 지점, 판매월
     , 매출
     , sum(매출) over (partition by 지점 order by 판매월 range between unbounded preceding and
       current row) 누적매출
from   월별지점매출;
```

② 윈도우 함수를 이용하지 않는 방식

```
select t1.지점, t1.판매월
     , min(t1.매출) 매출
     , sum(t2.매출) 누적매출
from   월별지점매출 t1, 월별지점매출 t2
where    t2.지점 = t1.지점
and      t2.판매월 <= t1.판매월
group by t1.지점, t1.판매월
order by t1.지점, t1.판매월;
```

또는

```
select t1.지점, t1.판매월
     , t1.매출
     , sum(t2.매출) 누적매출
from   월별지점매출 t1, 월별지점매출 t2
where  t2.지점 = t1.지점
and    t2.판매월 <= t1.판매월
group by t1.지점, t1.판매월, t1.매출
order by t1.지점, t1.판매월;
```

실기문제 1 - 해설

누적매출(running total)을 구할 때, 윈도우 함수를 지원하는 DBMS 버전이라면 윈도우 함수가 가장 효과적이고 성능도 빠르다. 윈도우 함수를 사용할 때는 partition by절을 정확히 작성하는 것이 무엇보다 중요하다.

윈도우 함수를 지원하는 DBMS 버전이라면 부등호 조인을 이용해 누적매출을 구할 수 있다. 스칼라 서브쿼리를 이용하는 방법도 있지만, 전체범위처리에 결코 효과적이지 못하다.

실기문제 2 - 정답

```
create index 고객_idx on 고객(거주지역코드, 고객명);
create index 주문_idx on 주문(고객번호, 주문일시) local;

select /*+ leading(c) use_nl(o) index(c 고객_idx) index(o 주문_idx) */
       o.주문일시, o.주문번호, c.고객번호, c.고객명, o.주문금액
from   고객 c, 주문 o
where  o.주문일시 between to_date('20150301', 'yyyymmdd')
and to_date('20150314235959', 'yyyymmddhh24miss')
and    o.고객번호 = c.고객번호
and    (c.거주지역코드, c.고객명) in (('02', '김철수'), ('05', '홍길동'))
order by o.주문일시, c.고객명
```

실기문제 2 - 해설

고객 20여 건을 읽어 주문과 조인한 후 최종적으로 5건을 출력하는 SQL문이다. 소량 데이터를 조인할 때는 가급적 인덱스를 이용한 NL 조인을 활용해야 한다.

고객 테이블에 인덱스를 사용하려면, 거주지역코드와 고객명을 가공하지 않아야 한다. 문자열로 결합한 부분을 아래와 같이 변경하면 된다.

```
and    (c.거주지역코드, c.고객명) in (('02', '김철수'), ('05', '홍길동'))
```

OR 또는 UNION ALL 방식을 사용해도 성능은 같다. OR 조건을 사용할 때는 가급적 USE_CONCAT 힌트를 같이 사용하는 것이 좋다.
인덱스는 「거주지역코드 + 고객명」 또는 「고객명 + 거주지역코드」 순으로 구성하면 된다.

인덱스를 이용해 주문 테이블과 NL 조인하려면 고객번호 + 주문일시 순으로 인덱스를 구성해야 한다. 주문일시 + 고객번호 순으로 인덱스를 구성해도 인덱스 사용은 가능하지만, Between 조건이 선두컬럼이므로 매우 비효율적이다.

실기문제 3 - 정답

```
[ SQL1 모범 답안 ]
select 주문번호, 업체번호, 주문일자, 주문금액
     , count(*) over (partition by 업체번호) 총주문횟수
     , avg(주문금액) over (partition by 업체번호) 평균주문금액
     , max(주문금액) over (partition by 업체번호) 최대주문금액
from   주문
where  주문일자 like '201509%'
order by 평균주문금액 desc
```

```
[ SQL2 모범 답안 - 1]
select 주문번호, 업체번호, 주문일자, 주문금액
from (
  select row_number() over (partition by 업체번호 order by 주문번호 desc) rnum
       , 주문번호, 업체번호, 주문일자, 주문금액
  from   주문
  where  주문일자 like '201509%'
)
where rnum = 1
```

```
[ SQL2 모범 답안 - 2]
select 주문번호, 업체번호, 주문일자, 주문금액
from (
  select max(주문번호) over (partition by 업체번호) 마지막주문번호
       , 주문번호, 업체번호, 주문일자, 주문금액
  from   주문
  where  주문일자 like '201509%'
)
where 주문번호 = 마지막주문번호
```

실기문제 3 - 해설

실무적으로 활용도가 매우 높은 윈도우 함수를 적절히 구사할 줄 아는지 확인하는 단순한 문제다. 참고로, SQL2는 모범답안 1이 더 효과적이다. 모범답안 2에 비해 소트 공간을 덜 사용하기 때문이다.

실기문제 4 – 정답

[SQL 모범 답안1]

```
select 고객번호, 주문일시, 주문금액, 우편번호, 배송지
from   주문
where  고객번호 = nvl(:cust_no, 고객번호)
and    주문일시 >= to_date(:ord_dt1, 'yyyymmdd')
and    주문일시 <  to_date(:ord_dt2, 'yyyymmdd') + 1
order by 주문일시 desc
```

[SQL 모범 답안2]

```
select 고객번호, 주문일시, 주문금액, 우편번호, 배송지
from   주문
where  고객번호 = decode(:cust_no, null, 고객번호, :cust_no)
and    주문일시 >= to_date(:ord_dt1, 'yyyymmdd')
and    주문일시 <  to_date(:ord_dt2, 'yyyymmdd') + 1
order by 주문일시 desc
```

[SQL 모범 답안3]

```
select 고객번호, 주문일시, 주문금액, 우편번호, 배송지
from   주문
where  :cust_no is not null
and    고객번호 = :cust_no
and    주문일시 >= to_date(:ord_dt1, 'yyyymmdd')
and    주문일시 <  to_date(:ord_dt2, 'yyyymmdd') + 1
union all
select 고객번호, 주문일시, 주문금액, 우편번호, 배송지
from   주문
where  :cust_no is null
and    주문일시 >= to_date(:ord_dt1, 'yyyymmdd')
and    주문일시 <  to_date(:ord_dt2, 'yyyymmdd') + 1
order by 2 desc
```

[인덱스 구성안]

X01 : 고객번호 + 주문일시

X02 : 주문일시

실기문제 4 - 해설

옵션조건에 대한 최적 SQL 작성능력을 확인하는 문제다.

고객번호처럼 조회조건 포함 여부를 사용자가 선택할 수 있는 옵션조건을 처리할 때 모범답안 3처럼 UNION ALL을 사용하면 가장 확실한 성능을 보장할 수 있다. 오라클의 경우, NVL 또는 DECODE를 사용하면 옵티마이저가 UNION ALL 방식으로 자동 변환해 준다.

오라클도 모든 NVL/DECODE를 UNION ALL로 변환해 주지는 않는다. 또한, 조건절 컬럼이 Null 허용컬럼일 때 NVL/DECODE를 사용하면 결과집합에 오류가 발생한다. 대개 옵션조건이 여러 개이고 그 중 Null 허용 컬럼을 포함할 수 있으므로 실무적으로 NVL/DECODE와 UNION ALL을 적절히 혼용해야 SQL을 최적화할 수 있다.

다행히 본 문제에서는 옵션조건이 단 하나뿐이고 Not Null 컬럼이므로 NVL/DECODE 사용이 가장 효과적이다.

프로그램 사용자가 고객번호를 입력할 때 가장 최적으로 수행하려면 인덱스를 고객번호 + 주문일시 순으로 구성해야 한다. 고객번호를 입력하지 않을 때는 주문일시만으로 조회하므로 인덱스를 주문일시 단일컬럼으로 구성하면 최적이다.

인덱스를 주문일시 + 고객번호 순으로 구성하면 2가지 케이스를 모두 처리할 수 있다. 하지만, 특정 고객을 조회하고자 할 때마다 인덱스에서 평균적으로 60,000건을 스캔해야 하므로 매우 비효율적이다. 주문일자에 최대 1주일까지 입력할 수 있다고 했으므로 이때는 140,000건을 스캔해야 한다.

실무적으로 많이 사용하는 옵션조건 처리방안이 2가지 더 있다.

첫째, 아래와 같이 OR 조건을 사용하는 방법이다. 문제는, OR 조건을 사용했으므로 고객번호를 인덱스 조건으로 사용할 수 없다는 점이다. 인덱스를 사용하려면 주문일시 인덱스를 사용해야 하므로 사용자가 고객번호를 입력하더라도 주문일시 조건에 해당하는 데이터를 모두 액세스하게 된다. 따라서 고객번호처럼 변별력이 좋아 인덱스 활용성이 높은 컬럼에 OR 조건을 사용해선 안 된다.

```
select 고객번호, 주문일시, 주문금액, 우편번호, 배송지
from   주문
where  (고객번호 = :cust_no or :cust_no is null)
and    주문일시 >= to_date(:ord_dt1, 'yyyymmdd')
and    주문일시 <  to_date(:ord_dt2, 'yyyymmdd') + 1
order by 주문일시 desc
```

둘째, 아래와 같이 LIKE 조건을 사용하는 방법이다. 문제는, 고객번호가 Number 형인 상황에서 LIKE 조건을 사용하면 묵시적 형변환이 발생한다는 점이다. 따라서 고객번호를 선두로 갖는 인덱스로는 Index Range Scan이 불가능해진다. Index Range Scan이 가능하게 하려면 주문일시 + 고객번호 순으로 인덱스를 만들어야 하므로, 인덱스 스캔 과정에 많은 비효율이 발생한다.

```
select   고객번호, 주문일시, 주문금액, 우편번호, 배송지
from     주문
where    고객번호 like :cust_no || '%'
and      주문일시 >= to_date(:ord_dt1, 'yyyymmdd')
and      주문일시 <  to_date(:ord_dt2, 'yyyymmdd') + 1
order by 주문일시 desc
```

실기문제 5 - 정답

[페이징 처리용 SQL]

```
select a.고객번호, a.고객명, a.등록일시, a.연락처, a.주소
     ,(select max(접속일시)
       from   고객접속이력
       where  고객번호 = a.고객번호
       and    접속일지 >= trunc(add_months(sysdate, -1))) 최근접속일시
from (
    select rownum as no, a.*
    from (
        select 고객번호, 고객명, 등록일시, 연락처, 주소
        from   고객
        where  고객상태코드 = 'AC'
        order by 등록일시, 고객번호
    ) a
    where rownum <= :page * 20
) a
where no >= (:page-1) * 20 + 1
```

[파일 출력용SQL]

```
select a.고객번호, a.고객명, a.등록일시, a.연락처, a.주소, b.최근접속일시
from   고객 a
     ,(select 고객번호, max(접속일시) 최근접속일시
       from   고객접속이력
       where  접속일지 >= trunc(add_months(sysdate, -1))
       group by 고객번호) b
where  a.고객상태코드 = 'AC'
and    b.고객번호 = a.고객번호
order by a.등록일시, a.고객번호
```

[인덱스 설계]

고객 인덱스 : 고객상태코드 + 등록일시 + 고객번호

고객접속이력 인덱스 : 고객번호 + 접속일시

실기문제 5 - 해설

1. 화면 페이징 처리용 SQL을 정확히 작성해야 한다.
   온라인 화면 페이징 처리용 SQL은 최초 응답속도 최적화(first_rows) 목표에 맞게 SQL을 작성해야 한다. 인덱스를 이용해 부분범위처리가 가능하도록 구현해야 한다.
   모범답안으로 제시한 SQL의 유일한 단점은 앞 페이지에서 읽은 데이터를 다시 읽어야 하므로 뒤 페이지로 이동할수록 블록 I/O가 늘어난다는 점이다. 하지만 뒤쪽 페이지로 이동하는 경우가 흔치 않다는 요건을 명시하였으므로 표준적인 페이지 처리 방안으로 가장 적합하다. 실제 대부분 시스템에서 이 방식을 사용하고 있다.
   참고로, 앞 페이지에서 읽은 데이터를 다시 읽지 않게 구현하려면 UNION ALL 방식으로 매우 복잡하게 구현해야만 한다. 실제 구현해 본 독자라면, order by를 명시할 수 없다는 사실을 알 것이다. sort order by 연산이 나타나므로 의도했던 바와 다르게 전체범위 처리가 불가피해지기 때문이다. 결국, order by 절을 생략한 채 index 힌트를 이용해 정렬된 결과집합을 얻게 된다. 향후에 혹시 인덱스 구성이 변경되기라도 하면 정확한 결과집합을 보장할 수 없게 되므로 시스템 운영 과정에 주의가 필요하다.

2. 화면 페이징 처리용 SQL에서 최근접속일시는 맨 바깥쪽 SELECT-LIST에서 스칼라 서브쿼리로 구현해야 한다. 화면에 출력하는 20건에 대해서만 스칼라 서브쿼리를 수행하도록 하기 위함이다.

3. 파일 출력용 SQL을 정확히 작성해야 한다. 전체 데이터 출력용 SQL은 전체 응답속도 최적화(all_rows) 목표에 맞게 SQL을 작성해야 한다.

4. 고객 인덱스를 정확히 설계해야 한다.
   파일 출력용 SQL은 인덱스를 사용하면 오히려 비효율적이다. 따라서 인덱스는 화면 페이징 처리용 SQL에 최적화되도록 설계해야 한다.
   고객상태코드로 인덱스를 Range Scan하려면 선두 컬럼은 고객상태코드여야 한다. 부분범위처리가 가능하게 하려면, order by절 컬럼인 등록일시, 고객번호를 뒤쪽에 추가하면 된다.

5. 고객접속이력 인덱스를 정확히 설계해야 한다.
   '=' 조건인 고객번호를 선두에 두고, 부등호 조건인 접속일지를 뒤쪽에 추가하면 된다.

실기문제 6 - 정답

```
alter session enable parallel dml;

insert /*+ parallel(t 4) */ into 주문배송 t
select /*+ leading(d) use_hash(o) use_hash(c)
           full(o) full(d) index_ffs(c)
           parallel(o 4) parallel(d 4) parallel_index(c 4) */
       o.주문번호, o.주문일자, o.주문상품수, o.주문상태코드, o.주문고객번호, c.고객명
     , d.배송번호, d.배송일자, d.배송상태코드, d.배송업체번호, d.배송기사연락처
from   주문 o, 배송 d, 고객 c
where  o.주문일자 between '20160601' and '20160831'
and    o.주문번호 = d.주문번호
and    d.배송일자 >= '20160601'
and    c.고객번호 = o.주문고객번호
```

실기문제 6 - 해설

1. 다른 트랜잭션에 의한 동시 DML이 없는 야간 배치용 SQL이므로 병렬 DML 활용이 가능하다. 병렬로 Insert 하려면 우선 아래와 같이 parallel DML을 활성화해야 한다.

   ```
   alter session enable parallel dml;
   ```

   Oracle 11gR2부터는 enable_parallel_dml 힌트도 제공된다.
   parallel DML을 활성화한 상태에서 insert 바로 뒤에 parallel 힌트를 추가하면 된다.
   3,000만건 정도라면 Direct Path Load 기능만 활용해도 충분히 빠르게 Insert 할 수 있다. Direct Path Load 기능을 사용하려면 insert 바로 뒤에 append 힌트를 추가하면 된다. 참고로, parallel Insert는 따로 append 힌트를 사용하지 않아도 기본적으로 Direct Path Load 방식으로 작동한다.

2. 인덱스를 이용한 NL 조인은 소량 데이터를 조인하는 데 적합하다. 수십만 건 이상 데이터를 조인할 때는 캐시히트율이 좋지 않는 한 결코 빠른 성능을 기대할 수 없다. 3,000만 건에 이르는 데이터를 조인하면서 캐시히트율이 좋기를 기대할 수는 없다. 따라서 주문, 배송을 Full Scan과 해시 조인으로 유도해야 한다. 특히 주문 테이블은 주문일자 기준으로 월단위 Range 파티션된 상태다. 다른 조건절 없이 주문일자(파티션키) 조건만으로 3개월치를 조회하는 데 인덱스를 이용할 하등의 이유가 없다. 예를 들어, 어떤 초등학교에서 1층부터 6층까지 각 층을 한 학년씩 사용한다고 하자. 설문조사를 위해 3학년 학생 전체를 만나고자 할 때, 교무실에 비치된 학적부가 필요할까? 3층 전체를 스캔하는 것이 가장 빠르다.

3. 배송 테이블은 배송일자 기준으로 Range 파티션된 상태인데, 배송일자가 조건절에 없다. 따라서 Full Scan으로 처리한다면 전체 파티션을 읽어야 한다. 3,000만건 조인하기 위해 수십억 건을 읽어야 할 수도 있다. 실행계획 10번 라인 'PARTITION RANGE ALL'을 통해 이 사실을 확인할 수 있다.
이 문제를 어떻게 해결할 수 있을까? 배송은 주문이 완료된 후에 시작된다는 데서 힌트를 얻을 수 있다. 배송일자는 주문일자보다 크고, 주문일자는 '20160601'보다 크다. 따라서 배송일자도 '20160601'보다 크다. 아래 조건절을 추가해 주면 전체 파티션을 읽지 않아도 된다.

   and 배송일자 >= '20160601'

   설령 이 조건절을 추가하지 않아 전체 파티션을 읽더라도 인덱스를 이용한 NL 조인보다 Full Scan과 해시 조인이 빠르다.

4. 스칼라 서브쿼리는 NL 조인과 같은 방식으로 작동한다. 스칼라 서브쿼리는 입력 값과 결과 값을 PGA에 캐싱한다는 점이 다르다. 따라서 입력 값 종류가 적을 때 실제 조인 횟수를 줄여줌으로써 빠른 성능을 기대할 수 있다.
여기서는 고객 수가 500만 명이라고 명시했다. 스칼라 서브쿼리 캐싱효과가 도움이 되지 않는 상황이다. 따라서 고객에 대한 스칼라 서브쿼리를 일반 조인문으로 변경한 후 해시 조인으로 유도해야 한다.

5. 해시 조인이라고 해서 항상 Full Scan으로 처리해야 하는 것은 아니지만, 고객 테이블에는 조인 외에 다른 조건절 없으므로 Full Scan을 피할 수 없다.
그리고 고객명과 고객번호만 읽으면 되는 상황이므로 테이블 전체를 스캔할 필요없이 고객_N1 인덱스를 Fast Full Scan 방식으로 처리하면 된다. 인덱스 전체를 스캔하므로 인덱스 컬럼순서 변경은 불필요하다.

6. 온라인 트랜잭션이 없는 야간 배치용 SQL이고, 3,000만건에 이르는 대용량 데이터를 조인해야 하므로 병렬처리를 활용하지 않을 이유가 없다. 주문, 배송 테이블에 parallel 힌트를 사용하면 된다. 고객 테이블은 인덱스만 읽도록 유도했으므로 병렬 처리를 위해 parallel_index 힌트를 사용해야 한다. (최근 버전에서는 테이블이나 인덱스명을 지정하지 않고 아래와 같이 Degree만 지정한 parallel 힌트를 사용할 수 있다.)