Chapter 1.

DBMS (DataBase Management System): 데이터를 저장하고 처리하고 관리하는 시스템

File System을 이용할 때의 단점:

Data redundancy and inconsistency: App마다 FS에 다르게 저장하므로 공유할 수 없고, 따라서 데이터가 App에 종속된다.

Integrity problem: Integrity constraint을 App에서 지정해주어야 하고 체계적이지 못함.

Atomicity of updates: 업데이트 중 atomicity가 보장되지 않아 치명적 문제 발생 가능.

Concurrency: 역시 같은 문제

Levels of Abstraction

Physical level: 레코드가 실제로 물리 장치에 저장되어 있는 방식을 기술

Logical level: 데이터가 DB에 저장된 방식과, 데이터간의 관계를 기술

View level: 데이터 타입을 은폐할 수 있음.

Data Independence: 상위 레벨의 schema 정의에 영향을 끼치지 않고도 하위 레벨의 schema를 바꿀 수 있음.

Schema: the logical structure of the database

Instance: the actual content of the database at a particular point in time

Data models

Entity-Relationship Model

현실 세계를 entity와 그들의 relationship으로 서술.

Conceptual model에서 이렇게 사용하고 logical level은 Relational Model로 사용하는 경우가 많음.

Relational Model

모든 정보를 표로 표현.

Database Languages

Data Definition Language: DB schema를 정의하는 데 사용.

Data Manipulation Language: DB 안의 데이터를 조작하는데 사용

Query: 정보를 가져오는 데 사용하는 statement. DML로 취급되기도 함.

Database Administrator

DBA라고도 하며, 사람임.

DB 시스템의 모든 활동을 관리 감독한다.

Schema definition, data organization modification, granting authority to DB, integrity constraints specifying, responding to changes in requirements, monitoring performance, …

Chapter 2

모든 attribute values는 atomic하며, null은 모든 domain의 원소이다.

Relation schema:

Relation variable on the schema R: r(R)

Let K is a subset of R. Then,

K is a superkey of R

iff values for K are sufficient to identify a unique tuple of each possible relation r(R).

K is a candidate key

iff K is minimal

iff K is a superkey and no subset of K is also a superkey.

K is a primary key

iff a candidate key K is chosen to be one.

K is a foreign key

iff K references another relation.

Query Languages

Procedural: 무슨 데이터가 필요한지, 어떻게 구하는지 명시해줌. Relational Algebra.

Nonprocedural: declarative, 어떤 데이터가 필요한지만 명시함. SQL.

Pure Languages: DB를 다루는 기능을 이론적으로 완전히 명시함. QL의 기초가 됨. Relational Algebra.

Natural join을 cartesian product, selection, projection의 조합으로 나타내기:

Chapter 6

Select Operation

p is called the selection predicate which consists of terms connected by ^, v, ㄱ, where each term is one of [<attr> op <attr>] or [<const>], and each op is one of =, !=, >, >=, <, <=.

Project Operation

전체 튜플 수가 Project 후 달라질 수 있다! 왜냐하면 어쨌든 집합으로 정의되기에 duplicates가 사라지니까.

Union Operation

Union compatibility

r, s must have same arity. 즉 r과 s의 attributes 수는 같아야 한다!

The attribute domains must be compatible. 즉 각 attribute의 타입도 같아야 한다!

Set Difference Operation

차집합도 Union compatible해야 한다. 단, compatible 하지 않아도 이론적으로 동작은 함.

Cartesian Product Operation

r(R)과 s(S)가 disjoint하다고 가정하자. 그렇지 않으면 renaming 필요함.

Rename Operation

Expression E에 N이라는 이름을 붙여 앞으로 refer할 수 있음.

Expression E의 각 Attributes의 이름을 A\_1, …, A\_N으로 이름하고 E의 이름을 N으로 함.

로 모든 relational algebra expression을 표현할 수 있다.

e.g.

Natural Join Operation

과정: r, s 각각에서 튜플 t\_r과 t\_s를 추출하여, 만약 의 모든 attributes에 대해 값이 같다면, R의 attributes에 대해서는 t\_r와, S의 attributes에 대해서는 t\_s와 같은 밸류를 지닌 튜플 t를 결과에 추가한다.

: natural join에서는 공유 attr의 값이 같아야 결과에 나옴. 그런데 그 결과에서 공유하는 attr만 뽑아오겠다는 것은, 결국 공유하는 attr로만 제한한 상태의 두 relation의 교집합일 수 밖에 없음.

: 와! 결합법칙!

If , then : 공유 attr가 없다면 그냥 cartesian product와 같다. 애초에 정의할 때 공유하는 attr이 같은 애들은 그거만 두고 cartesian product하는거니까.

If then : 모든 attr가 공유 attr이란 소리니까, 결국 교집합임.

Chapter 3

SQL: IBM에서 Sequel이라는 이름으로 언어를 만듦. 이걸 나중에 Structured Query Language로 이름을 바꿈.

Create Table

create table r (A1 D1, A2 D2, …, An Dn,

Integrity-Constraint1,

…,

Integrity-Constraintk)

Data types:

char(n)

varchar(n)

int

smallint

numeric(p, d)

real, double precision

float(n)

e.g.

create table instructor (

ID char(5),

name varchar(20) not null,

dept varchar(20),

salary numeric(8, 2),

primary key (ID),

foreign key (dept) references department);

이 때 primary key로 선언된 attribute는 자동적으로 not null, unique IC가 적용된다.

Insert Table

insert into instructor values (‘12345’, ‘Smith’, ‘Biology’, ‘66000’);

insert into instructor (ID, name, dept, salary)

values (‘12345’, ‘Smith’, ‘Biology’, ‘66000’);

Alter Table

alter table r add A D

A: 더할 attribute 이름

D: A의 domain

이 때 기존의 튜플들의 A 값에는 null이 더해진다.

alter table r drop A

A: 뺄 attribute 이름

이름에 ‘-‘ 불허. 쓸거면 ‘\_’ 써라.

Select Clause

SQL은 implementational language이기도 하고, 현실적인 이유로 중복을 허용함. 따라서 중복을 지울거면 distinct를 attr 옆에 써라. 명시적으로 중복을 허용하려면 all을 쓰고.

String Operations

%: matches any substring

\_: matches any character

패턴을 찾을 때는 = 대신에 like를 쓴다!

\로 escape 가능

Set Operations

union, intersect, except

얘네는 전부 중복을 불허함. 중복 허용할거면 union all, intersect all, except all.

만약 한 튜플이 r에서 m번, s에서 n번 나타난다면:

r union all s에서 m+n번

r intersect all s에서 min(m, n)번

r except all s에서 max(0, m-n)번

Null Values

null에 어떤 연산을 하든 unknown을 리턴함.

이 때 true, false, unknown으로 구성되는 로직을 Three-Valued Logic이라 한다.

단 is null, is unknown은 T/F로 반환됨

Aggregate function: 일단 select 수행 후 그다음에 수행.

group by: group by 사용하면 select 문에 aggregate function 제외한 attributes는 group by로 묶여있어야 함

count(\*)를 제외한 모든 aggregate operations는 null value를 무시한다!

Nested Subqueries

: A select-from-where expression nested within another query.

Set membership, set comparison, set cardinality할 때 주로 쓴다.

이 때 발생하는 비효율은 DBMS가 알아서 최적화해줌.

Set Comparison

>some (), >all () 등등

Correlation Variables

현재 튜플의 포인터를 nested subquery에서 참조할 수 있다!

Not exists

Delete

delete from instructor

where dept\_name in (~);

문제:

delete from instructor

where salary < (select avg(salary) from instructor);

이렇게 하면 가면 갈수록 avg가 높아져 우리가 예상하는 답이 나오지 않게 된다!

이를 해결하기 위해 Query Optimizer는 미리 계산한 후에 비교한다.

Insert

Insert into student

Select ID, name, dept\_name, 0

From instructor;

Select from where 구문은 실제로 insert 하기 전에 모두 evaluate하여 다음과 같은 상황의 무한루프를 막는다.

Insert into table1 select \* from table1;

Update

update instructor

set salary = salary \* 1.03

where salary > 10000;

Chapter 4

Join Operation

Select \*

From r1 join r2 on condition;

Is equivalent to

Select \*

From r1, r2

Where condition;

From E1, E2, .., En도 사용할 수 있다. 이 때 Ek는 table일 필요 없이 table을 내놓는 subquery여도 가눙!

Join Types

Inner join = join (default)

Left outer join

Right outer join

Full outer join

Join Conditions

Natural

On <predicate> : 그냥 조건 거는거

Using (A1, …, An) : attr들만 가지고 equality condition 계산

Natural Join

만약에 natural join하려는 테이블끼리 혹시 몰랐는데 이름이 같은 attribute가 있으면 걔도 기준으로 삼아서 합치므로 항상항상조심해야함

Outer Join

Inner Join은 공유 attr의 value가 같은 튜플끼리 곱해준다. 이 말은 같은 value가 상대편에 없는 튜플은 무시당한다는, 즉 information loss가 일어난다는 말이다. Outer join은 만약 그렇다고 한다면 이런 친구들도 짝을 지어주되 다만 상대편의 attributes에는 null을 넣어준다.

Integrity Constraint

데이터베이스에 대한 변화가 consistency를 파괴하지 않도록 보장해줌.

IC 설정은 각 인스턴스에 대해 하는 것이 아니라, schema에 대해 해주어야 한다. 따라서 table을 define하는 단계에서 이루어져야 한다.

not null

unique(A1, …, An): A1, …, An의 조합이 unique해야 한다! 조합이 유니크하다 조합이! 각각이 아니라! 각각이 unique하다고 하고 싶으면 unique(A1), …, unique(An)해주어야 함.

primary key: not null & unique

check (P): predicate P가 항상 참이 되도록 인스턴스가 생성되어야 한다. 예를 들어 check(semester in (‘Fall’, ‘Winter’, ‘Spring’, ‘Summer’)) 정도로 써줄 수 있겠음.

Referential Integrity: Foreign Key Constraint. Foreign key는 반드시 레퍼런스되는 테이블의 primary key여야 한다.

Cascading Actions in Referential Integrity

Foreign key (dept\_name) references department

On delete cascade on update cascade

: 참조되는 값이 delete되거나 update됨으로써 constraint가 위반되면 그걸 참조하던 record도 지우거나 업데이트한다. Set null, set default같은 변종도 있음.

Authorization Specification in SQL

Grant <privilege list>

On <relation name, view name>

To <user list>

<user list>는 user id, public, role 셋 중 하나이다.

Grantor는 이미 해당 아이템에 대한 권한을 가지고 있어야 한다.

<privilege list>는 insert, update, delete, select, all privileges 등이 있다.

Revoke <privilege list>

On <relation name, view name>

from <user list>

<privilege list>는 all로 줄 수 있다.

Revoke된 사람이 revoke되는 privilege를 받은 사람의 previlige도 revoke. 단 다른 사람에게 똑같은 previlige를 또 받으면 그건 삶.

Create role grader;

Grant select on student to grader;

Grant select, update on takes to grader;

Grant grader to U1, U2, U3;

Create role professor;

Grant grader to professor;

Chapter 4

현실에서 App Requirements가 내려오면, Conceptual design단계에서 conceptual schema, 즉 ER schema를 만들어내고, 이를 Logical design 단계에서 DBMS의 implementation model, 즉 relational model로 변환한 뒤 Physical design 단계에서 physical schema, 즉 index 등을 사용하는 레벨로 구현한다.

Entity-Relationship Model

NOT an implementational model: ER 모델을 기반으로 하는 DBMS는 없음.

Entity: an object that exists and is distinguishable from other objects. Entity set의 element라고 보면 될듯

Entity set: a set of entities of the same type that share the same properties

Types of Attributes

Simple attribute: attribute의 value가 atomic -> relational model의 가정과 같음. Values cannot be divided into subparts.

Composite attribute: 하나의 attr. Value가 여러 개로 구성 -> ER model에서는 허용됨. 왜 허용하냐? Requirement만 잘 표현하면 되기 때문에 나중에 atomic하게 바꾸더라도 일단은 허용함.

Multivalued attribute: attribute에 대한 인스턴스는 여러 개로 나뉠 수 있음. 아니어도 되고. Composite attribute는 attribute레벨에서 여러개인 것으로 다른 개념.

Derived attribute: 다른 attribute나 entity로 유추가능한 attribute. Duration, count, sum, …

Degree of a relationship set

Most are binary. Unary로 보이는 것도 사실은 unary가 아니라 binary의 특수한 경우이다.

Relationship Cardinality

한 entity가 다른 entity와 relationship set을 통해 대응될 수 있는 수.

Mapping 수의 Max값에 대한 constraint이다.

즉 한 Entity가 다른 entity 최대 1개에 매핑되던가, 아니면 제한이 없던가.

Participation of an Entity set in a Relationship set

한 entity가 다른 entity와 relationship set을 통해 대응될 수 있는 수

Mapping 수의 Min값에 대한 constraint이다.

Total participation: 그 entity set의 모든 entity는 최소한 한 번은 relationship에 참여해야 함.

Partial participation: 그런 제한 없음.

Keys

Key for an Entity: relational model과 같음

Key for a relationship: combination of PKs of the participating entity sets forms a super key

Redundant attributes

ER 모델 레벨에서 한 entity set이 FK를 가지고 있음. 이미 relationship을 통해 이어져 있으면 이는 redundant attributes. Consistency 등의 문제가 발생하니 없애는게 좋음.

Weak Entity Sets

An entity set that does not have sufficient attributes to form a PK. 이 때 attributes를 discriminator이라 한다.

Identifying relationship

Weak Entity Sets의 entity들을 구분하게 해주는 relationship.

PK(WES) = PK(identifying strong entity) U Discriminator(WES)

Reducing ER to tables

Basic rules:

Entity set -> unique table

Relationship set -> unique table

Entity Sets:

Attributes a1, …, an은 그대로 schema E(a1, …, an)으로 가져와서 r(E)로 만듦

Relationship Sets:

Entity E1, …, Ek와 관계하고 있었다면, PK(E1) U … U PK(Ek) U attr(R)을 attribute로 가짐

Special Case: one-to-many or one-to-one

E1 ~ E2가 one-to-many이면, E2를 나타내는 테이블에 PK(E1) U attr(R)을 추가하면 됨. 그러면 PK(E1)을 FK로 참조하게 할 수 있음.

Generalization/Specialization to Relation Schema

Option 1. Entity sets을 전부 가지고 있고, FK로 참조하기

Partial and/or overlapping generalization에 좋음.

Option 2. 하위 Entity set만 가지고 있기

Total and/or disjoint generalization에 좋음.

Design Issues

Entity vs Attribute: 중요하면 entity, 간단하면 attribute

Entity vs Relationship: 이것저것 할 수 있어야하면 entity, 간단하면 attribute

모든 n-ary relationships는 binary relationships로 바꿀 수 있다. Entity랑 relationships 몇개 더 하면 됨. 하지만 항상 좋은 방법인 건 아니다.

Chapter 8

그렇다면 ER schema를 잘 짰다는 가정 하에 어떻게 해야 좋은 DB를 만들 수 있을 것인가?

Attribute끼리의 relationship이 모두 구현되어야 한다.

Redundancy가 없어야 한다.

IC가 부여되어야 한다.

Anomalies

Insertion Anomaly: 들어오는 레코드가 없으면, 그 레코드가 가지고 있는 다른 attribute의 정보를 알 수 없음

Deletion Anomaly: 한 attribute의 값을 나타내는 마지막 레코드가 삭제되면 그 값도 사라짐

Update Anomaly: 한 레코드에서만 값을 변경하면 다른 레코드와 inconsistent.

이는 모두 redundancy에서 나온다! 따라서 분리해야 한다.

First Normal Form (1NF)

Domain is atomic iff its elements are considered to be indivisible units

A relational schema is in 1NF iff domains of all attriutes are atomic.

A DB is in 1NF iff all relationship schema are in 1NF.

Relational Theory

그렇다면 어떤 relation R이 good form인지 아닌지 어떻게 알 수 있을까?

Good form이 아닌 relation은 set of relation으로 decompose하되, 다음 조건을 따른다.

Each relation is in good form

The decomposition is lossless

Functional Dependencies

Legal relations에 대한 constraint.

어떤 set of relations가 다른 set of relations를 unique하게 determine한다!

즉 attribute set A가 attribute set B를 functionally determine

⬄ A가 같으면 B도 같다

⬄ B는 A에 dependent

Functional dependency는 key의 generalization이다.

Let R be a relation schema, and a and b are a subset of R.

Then functioinal dependency a -> b holds on R iff

For any legal relations r(R), whenever any two tuples t1 and t2 of r agree on the attributes a, they also agree on the attributes b.

That is to say, t1[a] = t2[a] => t1[b] = t2[b]

Application of FD

K is a superkey for relation schema R iff K -> R

K is a candidate key for R iff K -> R and for no proper subset a of K, a -> R

For a set of functional dependencies F, F holds on R

iff all legal relations on R satisfy the set of functional dependencies F.

If a relation r is legal under F, we say r satisfies F.

그러니까 schema R과 그 인스턴스 r들에 대해, 모든 인스턴스 r이 F를 만족하면 F holds on R인 것이고, 어떤 인스턴스 r이 F를 만족하면 r satisfies F인 것이다.

즉, functional dependency가 모든 legal instance에 대해 성립하지 않는다고 하더라도, 어떤 특정한 instance는 이를 만족할 수 있다.