|  |  |
| --- | --- |
| 데이터관리와 분석 | Project #1 TEAM 08 |

|  |
| --- |
| **The Conceptual DB design for the website A**  사이트 A가 사용하는 DB에 대한 ER diagram과 관계 스키마 도식화하기 |



|  |  |  |  |
| --- | --- | --- | --- |
| 제출일 | 2018. 10. 13. | 담당교수 | 박종헌 |
| 작성자 | 자유전공학부 2017-15556  원자핵공학과 2013-11311  자유전공학부 2013-13475 | | 김경준  유승룡  조시현 |

1. 문제 정의

사이트 A는 식당, 미용실 등의 업소에 대한 정보와 사진, 그리고 사용자의 리뷰를 제공한다. 우리는 사이트 A의 요청을 받아 그들을 위한 데이터베이스를 설계하게 되었다. 본 보고서는 사이트 A가 사용하는 DB에 대한 ER diagram과 관계 스키마를 도식화하여 제시하고, 그에 대한 설명을 기술하는 것을 목표로 한다.

사이트 A에 대한 데이터베이스를 설계하는 문제는 일종의 최적화 문제의 관점으로 접근할 수 있다. 즉, 주어진 requirement와 스키마 자체의 constraint를 만족하면서 redundancy를 최소화하는 스키마를 찾아낸다는 것이다. 우리는 iterative refinement process를 거치면서 주어진 문제에 대한 적절한 수준의 해답을 찾을 수 있었다. ER diagram과 관계 스키마를 설계하면서 필요했던 추가적인 가정들까지 모두 보고서에 명시하고자 하였다.

2. 완성된 ER diagram

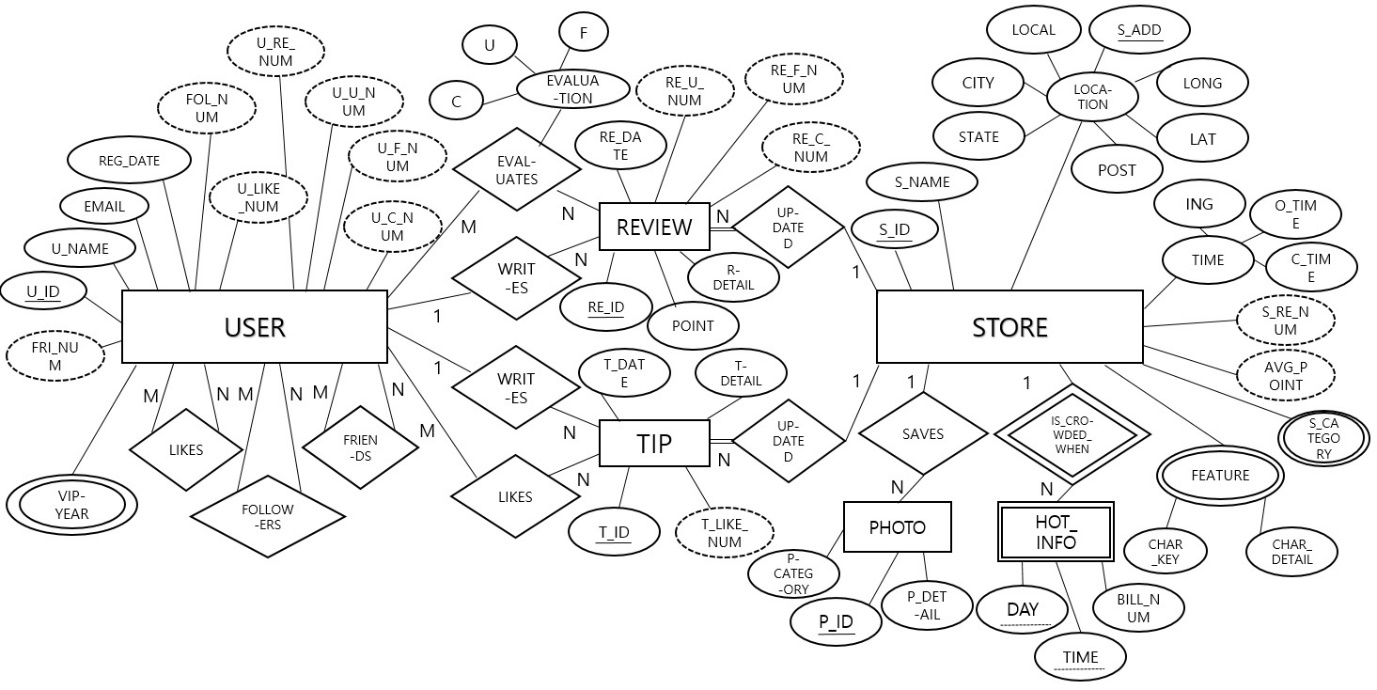


Figure 1 The ER diagram for the website A

3. 완성된 관계 스키마

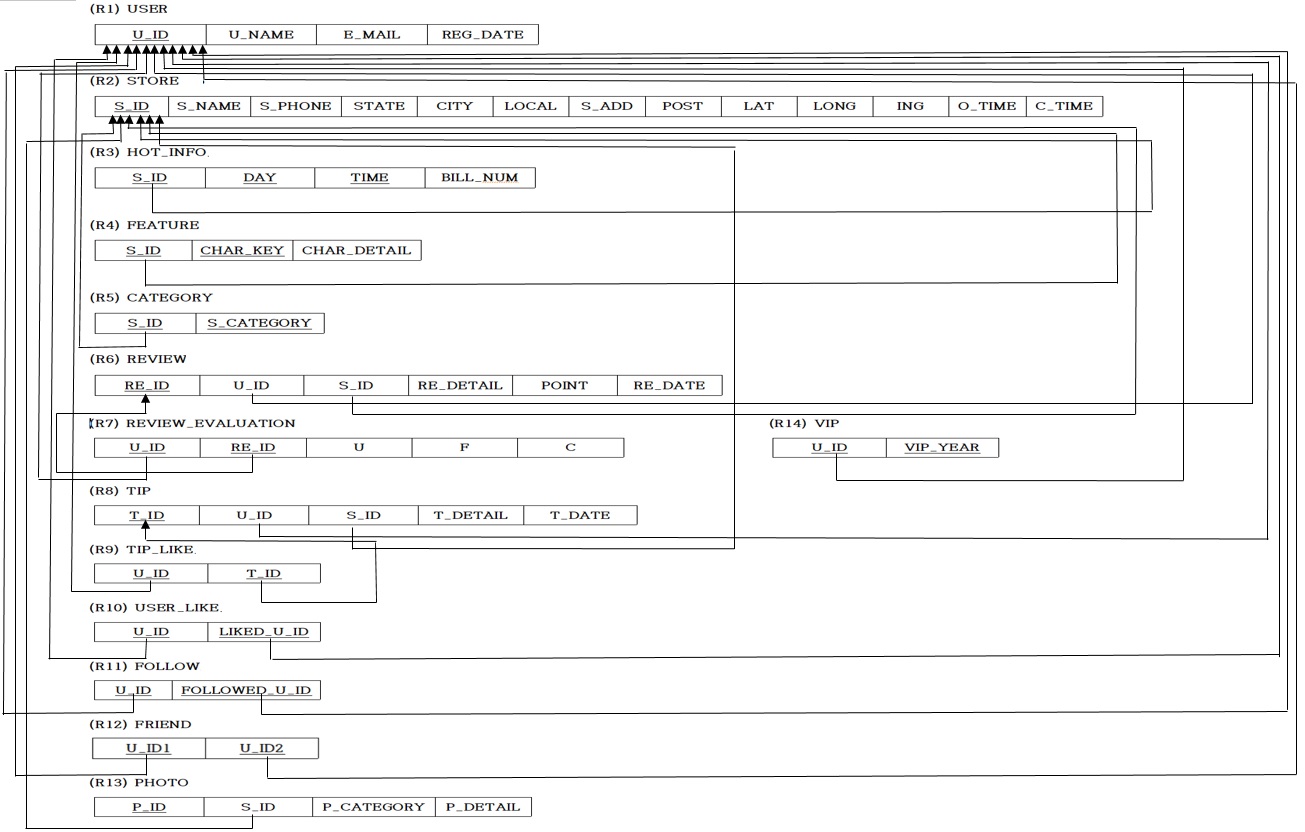


Figure 2 The Relational DB schema for the website A

3. 1. Domain constraints and constraints on null values

(Relation 1) USER: 사용자의 정보에 대한 relation

* U\_ID: 사용자마다 고유한 값으로 지정되는 22자 이하의 string type (PK)
* U\_NAME: 사용자의 이름을 나타내는 22자 이하의 string type, null 허용 / 이름이라 일단 string이고, 외국인도 웬만한 이름이면 full name으로 가입을 허용하기 위해 22자 이하로 제한했다.
* E\_MAIL: 사용자의 이메일 주소를 나타내는 30자 이하의 string type, null 허용
* REG\_DATE: 사용자가 계정을 생성한 날짜를 나타내며, 연-월-일이 ‘yyyy-mm-dd’의 꼴로 나타내어지는 string type

(Relation 2) STORE: 업소의 정보에 대한 relation

* S\_ID: 업소마다 고유한 값으로 지정되는 22자 이하의 string type (PK)
* S\_NAME: 업소의 이름을 나타내는 22자 이하의 string type / 이름이라 일단 string이고, 꽤 긴 이름의 업소까지 full name으로 가입을 허용하기 위해 22자 이하로 제한했다.
* S\_PHONE: 업소의 연락처를 나타내며, 숫자와 ‘-‘의 조합으로 나타내어지는 13자 이하의 string type, null 허용
* STATE: 업소가 위치한 주를 나타내는 string type, null 허용 / 이름이라 string으로 정했다.
* CITY: 업소가 위치한 도시를 나타내는 string type, null 허용 / 이름이라 string으로 정했다.
* LOCAL: 업소가 위치한 지역구를 나타내는 string type, null 허용 / 이름이라 string으로 정했다.
* S\_ADD: 업소가 위치한 상세 주소를 나타내는 string type, null 허용 / 주소라 string으로 정했다.
* POST: 업소의 우편 주소를 나타내며, 숫자와 ‘-‘의 조합으로 이뤄지는 string type, null 허용 / 우편 번호라 string이고, 해당 꼴로 정했다.
* LAT: 업소의 위도를 나타내며, -90에서 90 사이의 값을 가지는 float type, null 허용 / 위도라 값의 범위를 제한했고, 양수는 N을 음수는 S를 의미한다.
* LONG: 업소의 경도를 나타내며, -180에서 180 사이의 값을 가지는 float type, null 허용 / 경도라 값의 범위를 제한했고, 양수는 E를 음수는 W를 의미한다.
* ING: 업소가 영업 중이면 1, 휴업 중이면 0을 값으로 가지는 integer type
* O\_TIME: 업소의 개점 시간을 나타내며, ‘hh:mm’의 형태로 나타내어지는 string type, null 허용 / 시간이라 해당 꼴로 정했다. Multivalued가 아니라는 숨은 가정이 있다.
* C\_TIME: 업소의 폐점 시간을 나타내며, ‘hh:mm’의 형태로 나타내어지는 string type, null 허용 / 시간이라 해당 꼴로 정했다. Multivalued가 아니라는 숨은 가정이 있다.

(Relation 3) HOT\_INFO: 업소가 특정 요일, 시간에 얼마나 붐빌 것인지에 대한 relation

* {S\_ID, DAY, TIME}이 (PK)이다.
* S\_ID: 업소마다 고유한 값으로 지정되는 22자 이하의 string type (FK)
* DAY: 업소가 영업을 하는 요일을 나타내며, {MON, TUE, WED, THU, FRI, SAT, SUN} 중 하나의 값을 가지는 string type / 요일이라 string이고 가질 수 있는 값의 집합을 정했다.
* TIME: 업소가 영업을 하는 시간 구간에 대한 정보를 나타내며, ‘hh:00’(hh:00~hh:59의 구간을 의미)의 꼴로 나타내어지는 string type
* BILL\_NUM: 특정 업소에서 특정 요일, 시간 구간에 발생한 계산서의 수를 나타내는 integer type, null 허용

(Relation 4) FEATURE: 업소의 특징 정보에 대한 relation

* {S\_ID, CHAR\_KEY}가 (PK)이다.
* S\_ID: 업소마다 고유한 값으로 지정되는 22자 이하의 string type (FK)
* CHAR\_KEY: 업소가 가지는 특징적인 요소를 나타내는 10자 이하의 string type, null 허용 / 특징을 간단하게 표현하도록 10자라는 제한을 두었다.
* CHAR\_DETAIL: 업소의 특정 특징 요소에 대한 상세 정보를 나타내는 string type, null 허용

(Relation 5) CATEGORY: 업소의 카테고리 정보에 대한 relation

* {S\_ID, S\_CATEGORY}가 (PK)이다.
* S\_ID: 업소마다 고유한 값으로 지정되는 22자 이하의 string type (FK)
* S\_CATRGORY: 업소에 대한 분류(카페, 양식, 주점 등)를 나타내는 string type, null 허용

(Relation 6) REVIEW: 사용자가 업소에 남긴 후기 정보에 대한 relation

* RE\_ID: 리뷰마다 고유한 값으로 지정되는 22자 이하의 string type (PK)
* U\_ID: 리뷰를 남긴 사용자의 id, 22자 이하의 string type (FK)
* S\_ID: 리뷰가 남겨진 업소의 id, 22자 이하의 string type (FK)
* RE\_DETAIL: 사용자가 작성한 후기를 나타내는 string type
* POINT: 사용자가 업소에 할당한 별점의 점수를 나타내며, 1부터 5 사이의 값을 가지는 integer type
* RE\_DATE: 사용자가 리뷰를 작성한 일시를 나타내며, 연-월-일 시간-분-초가 ‘yyyy-mm-dd hh-mm-ss’의 꼴로 나타내어지는 string type

(Relation 7) REVIEW EVALUATION: 후기에 대해 다른 사용자들이 남긴 평가에 대한 relation

* {RE\_ID, U\_ID}가 (PK)이다.
* RE\_ID: 평가를 받은 리뷰의 id, 22자 이하의 string type (FK)
* U\_ID: 리뷰에 평가를 남긴 사용자의 id, 22자 이하의 string type (FK)
* U: 리뷰에 대해 useful 평가를 내린 경우이며, 평가에 U가 있으면 1, 없으면 0의 값을 가지는 integer type / 개수 합산이 쉽고 U, F, C의 조합을 직관적으로 표현할 수 있으므로 1과 0으로 표현했다.
* F: 리뷰에 대해 funny 평가를 내린 경우이며, 평가에 F가 있으면 1, 없으면 0의 값을 가지는 integer type / 개수 합산이 쉽고 U, F, C의 조합을 직관적으로 표현할 수 있으므로 1과 0으로 표현했다.
* C: 리뷰에 대해 cool 평가를 내린 경우이며, 평가에 C가 있으면 1, 없으면 0의 값을 가지는 integer type / 개수 합산이 쉽고 U, F, C의 조합을 직관적으로 표현할 수 있으므로 1과 0으로 표현했다.

(Relation 8) TIP: 사용자가 업소에 대해 남긴, 알아두면 좋은 팁에 대한 relation

* T\_ID: 팁마다 고유하게 지정되는 0보다 큰 integer type (PK)
* U\_ID: 팁을 남긴 사용자의 id, 22자 이하의 string type (FK)
* S\_ID: 팁이 남겨진 업소의 id, 22자 이하의 string type (FK)
* T\_DETAIL: 사용자가 작성한 팁 내용을 나타내는 string type
* T\_DATE: 팁이 작성된 일시를 나타내며, 연-월-일 시간-분-초가 ‘yyyy-mm-dd hh-mm-dd’의 형태로 나타내어지는 string type

(Relation 9) TIP\_LIKE: 다른 사용자가 팁에 남긴 '좋아요'에 대한 relation

* {T\_ID, U\_ID}가 (PK)이다.
* T\_ID: '좋아요'를 받은 팁의 id, 0보다 큰 integer type (FK)
* U\_ID: 팁에 '좋아요'를 남긴 사용자의 id, 22자 이하의 string type (FK)

(Relation 10) USER\_LIKE: 한 사용자가 다른 사용자에게 남긴 '좋아요'에 대한 relation

* {U\_ID, LIKED\_U\_ID}가 (PK)이다.
* U\_ID: LIKED\_U\_ID 사용자에게 '좋아요'를 남긴 사용자의 id, 22자 이하의 string type (FK)
* LIKED\_U\_ID: U\_ID 사용자로부터 '좋아요'를 받은 사용자의 id, 22자 이하의 string type (FK)

(Relation 11) FOLLOW: 한 사용자가 다른 사용자를 팔로우하는 정보에 대한 relation

* {U\_ID, FOLLOWED\_U\_ID}가 (PK)이다.
* U\_ID: FOLLOWED\_U\_ID 사용자를 팔로우하는 사용자의 id, 22자 이하의 string type (FK)
* FOLLOWED\_U\_ID: U\_ID 사용자로부터 팔로우를 받은 사용자의 id, 22자 이하의 string type (FK)

(Relation 12) FRIEND: 사용자들 사이의 '친구' 정보에 대한 relation

* {U\_ID1, U\_ID2}가 (PK)이다.
* U\_ID1: U\_ID2와 '친구' 관계에 있는 사용자의 id, 22자 이하의 string type (FK)
* U\_ID2: U\_ID1과 '친구' 관계에 있는 사용자의 id, 22자 이하의 string type (FK)

(Relation 13) PHOTO: 업소가 등록한 사진들에 대한 relation

* P\_ID: 사진마다 고유하게 지정된 링크를 나타내는 string type (PK)
* S\_ID: 사진을 등록한 업소의 id, 22자 이하의 string type (FK)
* P\_CATEGORY: 사진에 대한 분류(음식, 인테리어 등)를 나타내는 string type, null 허용
* P\_DETAIL: 사진에 대한 상세 정보를 나타내는 string type, null 허용

(Relation 14) VIP: 사용자가 각 해에 받은 VIP 칭호 정보에 대한 relation

* {U\_ID, VIP\_YEAR}가 (PK)이다.
* U\_ID: VIP\_YEAR에 VIP로 선정된 사용자의 id, 22자 이하의 string type (FK)
* VIP\_YEAR: U\_ID 사용자가 VIP로 선정된 해를 나타내며, 4자리의 integer type

\*\* RE\_DETAIL, T\_DETAIL, P\_DETAIL과 같이 길이가 명시되지 않은 문자열 타입의 data들은 전부

최대 255자로 제한한다.

3. 2. About Derived attributes in the ER diagram

이상과 같이 설계한 relational DB schema에는 requirement에서 요구하는 몇 가지 정보가 직접 저장되어 있지는 않다. 왜냐하면 기존에 저장된 다른 정보들로부터 유추될 수 있는 정보이고, 꼭 대단히 빠른 시간 안에 불러와야 할 필요까지는 없다고 판단했기 때문이다. 이러한 조건에 해당하는 attribute는 ER diagram에 derived attribute로 나타나 있다(총 13개).

다음은 각 derived attribute가 기존의 정보로부터 JOIN 연산()과 aggregate function(F)을 통해 어떻게 유도될 수 있는지를 나타낸 것이다. 이들 중 만약 website A 관리자가 반드시 좀 더 빠른 시간 내에 나타내야 할 정보가 있다고 새로 판단을 내린다면, 기존 relational DB schema의 관련 relation에 attribute 항목을 추가하여 해당 정보를 직접 저장하는 방식으로 고칠 수 있을 것이다.

(Relation 1) USER 관련

* U\_RE\_NUM: 사용자가 남긴 리뷰의 수, 0이상의 integer type

→ U\_ID **F** COUNT RE\_ID (REVIEW)

* U\_U\_NUM: 사용자가 남긴 리뷰가 받은 useful 평가의 수, 0 이상의 integer type

→ UR(RE\_ID,U\_SUM)(RE\_ID **F** SUM U (REVIEW\_EVALUATION)

U\_ID **F** SUM U\_SUM (URREVIEW)

* U\_F\_NUM: 사용자가 남긴 리뷰가 받은 funny 평가의 수, 0 이상의 integer type

→ FR(RE\_ID,F\_SUM)(RE\_ID **F** SUM F (REVIEW\_EVALUATION))

U\_ID **F** SUM F\_SUM (FRREVIEW)

* U\_C\_NUM: 사용자가 남긴 리뷰가 받은 cool 평가의 수, 0 이상의 integer type

→ CR(RE\_ID,C\_SUM)(RE\_ID **F** SUM C (REVIEW\_EVALUATION))

U\_ID **F** SUM C\_SUM (CRREVIEW)

* U\_LIKE\_NUM: 사용자가 받은 '좋아요'의 수, 0 이상의 integer type

→ UL(U\_ID,U\_LIKE\_NUM)(LIKED\_U\_ID **F** COUNT U\_ID (USER\_LIKE))

* FOL\_NUM: 사용자를 팔로우하는 사용자들의 수, 0 이상의 integer type

→ FO(U\_ID,FOL\_NUM)(FOLLOWED\_U\_ID **F** COUNT U\_ID (FOLLOW))

* FRI\_NUM: 사용자와 '친구'인 사용자의 수, 0 이상의 integer type

→ FR(U\_ID,FRI\_SUM)(U\_ID1 **F** COUNT U\_ID2 (FRIEND))

(Relation 2) STORE 관련

* S\_RE\_NUM: 업소에 남겨진 리뷰의 수, 0 이상의 integer type

→ S\_ID **F** COUNT RE\_ID (REVIEW)

* AVG\_POINT: 업소가 받은 별점의 평균값, 0 이상의 float type

→ S\_ID **F** AVERAGE POINT (REVIEW)

(Relation 6) REVIEW 관련

* RE\_U\_NUM: 리뷰가 받은 useful 평가의 수, 0 이상의 integer type

→ RE\_ID **F** SUM U (REVIEW\_EVALUATION)

* RE\_F\_NUM: 리뷰가 받은 funny 평가의 수, 0 이상의 integer type

→ RE\_ID **F** SUM F (REVIEW\_EVALUATION)

* RE\_C\_NUM: 리뷰가 받은 cool 평가의 수, 0 이상의 integer type

→ RE\_ID **F** SUM C (REVIEW\_EVALUATION)

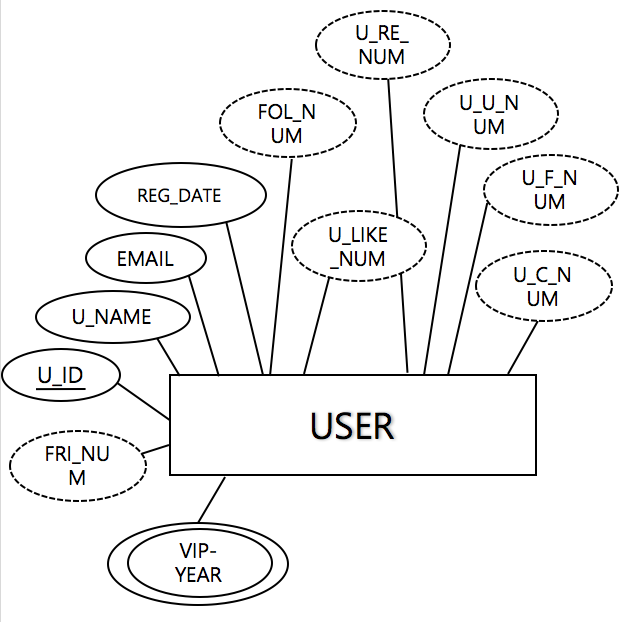
(Relation 8) TIP 관련

* T\_LIKE\_NUM: 팁이 받은 '좋아요'의 수, 0 이상의 integer type

→ T\_ID **F** COUNT U\_ID (TIP\_LIKE)

4. 설계된 DB의 Requirements 만족 여부 검토

**(R1) USER: 사이트에 가입된 사용자들의 정보에 대한 relation**

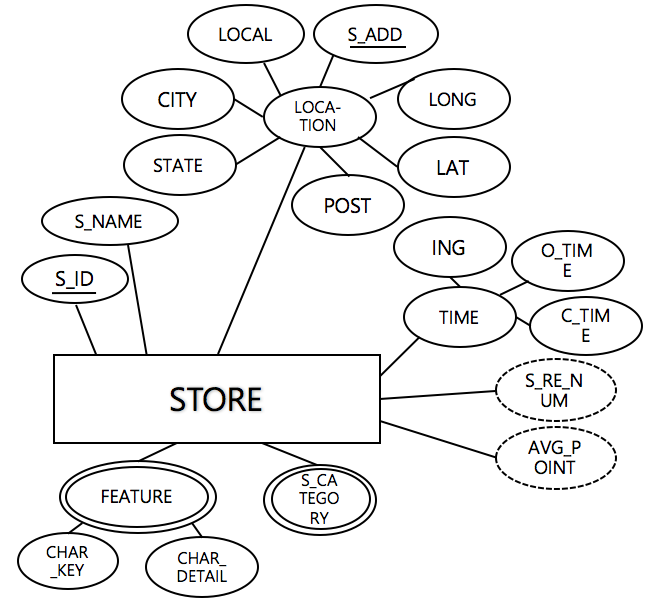
****

USER

**https://lh5.googleusercontent.com/0aygNPB6AOg_HSl1xdEBVVhnpTJHOkxiAdJ7xtx-hh5Y9J-B5QMufWmAgXNoT2J8IpIKRxfbOtMT8Gs6fwRtR1tADEJZ-7r-I8IE6P8JzLuUUUS3yjpXlMjMyqj0-_2ETR42qkfu**

* U\_ID: 사용자의 id, null 값 허용하지 않고 고유하므로 USER의 primary key로 설정하였다. 이는 entity integrity constraint를 만족한다.
* U\_NAME: 사용자의 이름 / E\_MAIL: 사용자의 이메일 주소 / REG\_DATE: 사용자의 계정 생성 날짜
* U\_LIKE\_NUM, FOL\_NUM, FRI\_NUM은 각각 사용자가 다른 사용자와 교류하는 세 가지 종류(좋아요, 팔로우, 친구)의 수를 의미한다. U\_U\_NUM, U\_F\_NUM, U\_C\_NUM은 각각 사용자가 남긴 리뷰에 다른 사용자가 남긴 평가(useful, funny, cool)의 수를 의미한다. 이들은 relation schema에 attribute로 지정하지 않았는데 3.2절에서 설명한대로 derived attribute 형태로 구할 수 있다.

**(R2) STORE: 업소들의 정보에 대한 relation**

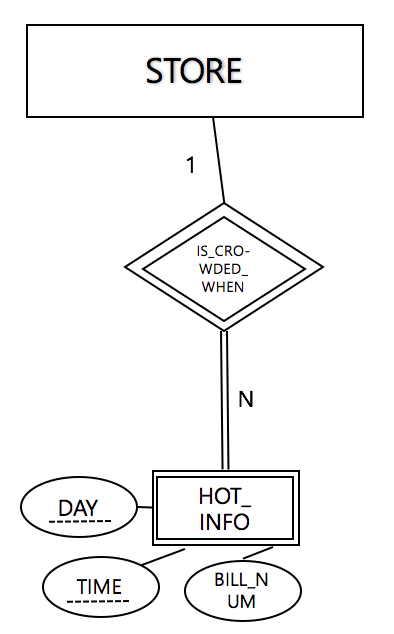
****

STORE

**https://lh3.googleusercontent.com/NPQcA__HQBhDyTAZQTKO_i2mqz2In-cxxR9eojK_5d1jeDumPQASWfuAzykfWXlogOjuaHrgFW6-xjhCsyFLQvqYMRaEZ4MX46RofUHRlMw_I1ZlaXJTW9-N6kKq7pSq8tt9TuYC**

* S\_ID: 업소의 id, null 값을 허용하지 않고 고유하므로 primary key로 설정했다. 이는 entity integrity constraint를 만족한다.
* S\_NAME: 업소명 / S\_PHONE: 업소의 전화번호 / STATE: 업소가 위치한 주 / CITY: 업소가 위치한 도시 / LOCAL: 업소가 위치한 지역구 / S\_ADD: 업소가 위치한 상세 주소 / POST: 업소의 우편번호 / LAT: 업소가 위치한 위도 / LONG: 업소가 위치한 경도 / ING: 업소의 현재 영업 여부 / O\_TIME: 업소의 개점 시간 / C\_TIME: 업소의 폐점 시간
* S\_RE\_NUM, AVG\_POINT는 각각 업소가 받은 리뷰의 수와 평균 별점을 의미한다. 이들은 ER diagram에는 derived attribute로 표현되어 있고, relation schema에는 나타나지 않는다. 3.2절에서 설명한 바와 같이 이 두 종류의 값은 직접 저장하지 않아도 연산을 통해 유도될 수 있다. 단, AVG\_POINT의 자료형이 실수형이므로 값을 불러와 실수형으로 변환을 해준 다음에 연산을 해야 한다.

**(R3) HOT\_INFO: 각 업소가 특정 요일, 시간에 얼마나 붐빌 것인지에 대한 relation**

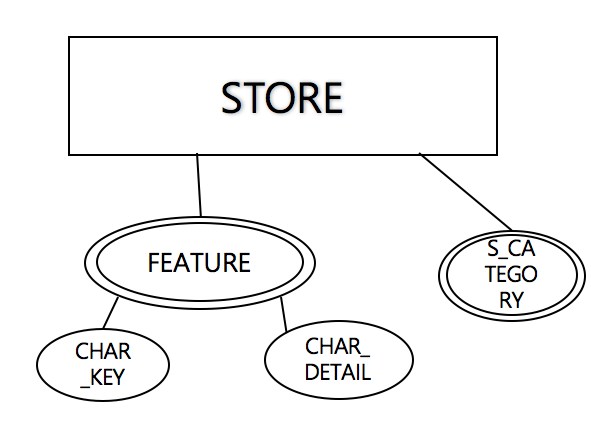
****

HOT\_INFO

**https://lh3.googleusercontent.com/WC9gHNq7AycApHuwCuN3ywFexPnKlOHa5czDQazv2813Ky0DVDSHs-hJjv5dF6wLZQIeRAtv6o77_vNtOs6cZbIB-nFhoVU3Ydl4YgcUN3i8cVYHAAymkgn8roiJuXDnYz8bzuja**

* S\_ID: 업소의 id로, STORE의 primary key인 S\_ID를 참조하는 foreign key이다. 이는 referential integrity constraint를 만족해야 한다.
* DAY: 업소 영업의 요일 / TIME: 업소 영업의 시간 구간 / BILL\_NUM: 해당 시간 구간 한 시간에 발생한 대략적인 계산서의 수
* {S\_ID, DAY, TIME}이 HOT\_INFO의 primary key가 된다. S\_ID와 고유 시간인 DAY, TIME이 결정되면 tuple간의 유일성이 보장 된다. DAY, TIME, BILL\_NUM이 전부 null인 경우는 저장하지 않고, 그게 아니라면 DAY, TIME이 동시에 null이면 안 된다는 constraint를 설정해주어야 한다. 또한 S\_ID null을 허용하지 않으므로 {S\_ID, DAY, TIME}이 null이 될 수 없으므로 entity integrity constraint를 만족한다.
* HOT\_INFO는 HOT\_INFO만의 key가 없으므로 weak entity이다. STORE의 S\_ID를 참조하여 identify되므로 IS\_CROWDED\_WHEN이 HOT\_INFO의 identifying relationship이고, partial key는 {DAY, TIME}이다. 하나의 STORE entity 당 여러 개의 HOT\_INFO가 related될 수 있으므로 1:N relationship으로 보았다.

**(R4) FEATURE: 업소의 특징 정보에 대한 relation, CATEGORY: 업소의 분류에 대한 relation**

****

FEATURE

**https://lh4.googleusercontent.com/7Nl-G2gvCzu6wex7scfYzyMJvUWYWz00P1I7i6EJ9T_beCrVjooc-AQvRA7h9DFU0Ah9K4YfnYmfHv5RpdiGi0F6HY_ri0uyHmqymUx1CGIOSsHqw1rx0xsuvldFBrhsoBXd3DOK**

* S\_ID: 특징을 저장한 업소의 id / CHAR\_KEY: 업소의 특징적인 요소 / CHAR\_DETAIL: 특징의 상세 정보

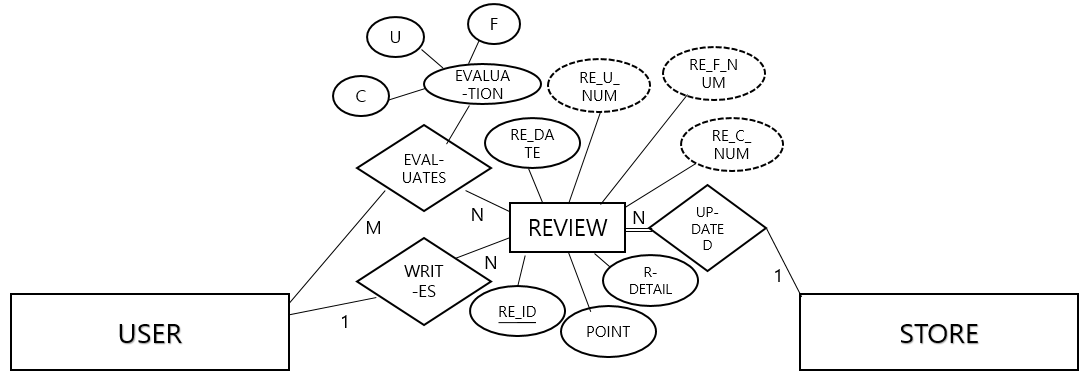
CATEGORY

**https://lh5.googleusercontent.com/sKlbb0lVRM8wFtqLmeQXVCVggiyA0SR8u3satV4f0yMwSbRrOTyOEIECZ2pOptxvw7khx2hNhnXHJwGH1SVjGscVnSNrEC6LE9svuARdg509lwSz3NbZ4Si6GW-RKgx_Ld77B5yK**

* S\_ID: 분류를 저장한 업소의 id / S\_CATEGORY: 해당 업소에 대한 분류
* FEATURE는 업소의 특징을 나타내는 relation이고, CATEGORY는 업소의 분류에 대한 relation이다. 각각을 multivalued attribute로 취급하여 따로 relation을 만들어 준 것이다. 둘 다 업소의 정보이므로 STORE의 S\_ID를 참조하는 foreign key를 가진다. 이로써 각각은 referential integrity constraint를 만족해야 한다. CHAR\_KEY와 S\_CATEGORY가 null을 허용하지만 FEATURE와 CATEGORY의 primary key를 각각 {S\_ID, CHAR\_KEY}, {S\_ID, S\_CATEGORY}로 지정했기 때문에 primary key 자체는 null을 허용하지 않고 각 relation 안에서의 unique함도 보장한다. 따라서 각각에 대해서 entity integrity constraint도 만족한다.

**(R5) REVIEW: 사용자가 업소에 남긴 리뷰 기록에 대한 relation**

**(R6) REVIEW EVALUATION: 사용자가 다른 사용자의 리뷰에 대해 남긴 평가에 대한 relation**

****

REVIEW

**https://lh6.googleusercontent.com/UvhO1g93noXbx7oGfkuXjaipMWSqEhwK6aGcJfAo6WLb7hbzP8Iz5l-Mf_N7nhrG1lARN3zQikwhKXZlFIyeN9JrwUAYlmWFwOXektieB6g3Wxy2nBkFWe9M9je-ZPQCRbK9e01F**

* RE\_ID: 리뷰의 id / U\_ID: 리뷰를 작성한 사용자의 id / S\_ID: 리뷰가 남겨진 업소의 id / RE\_DETAIL: 리뷰의 내용 / POINT: 사용자가 리뷰와 함께 남긴 별점 / RE\_DATE: 리뷰가 작성된 일시

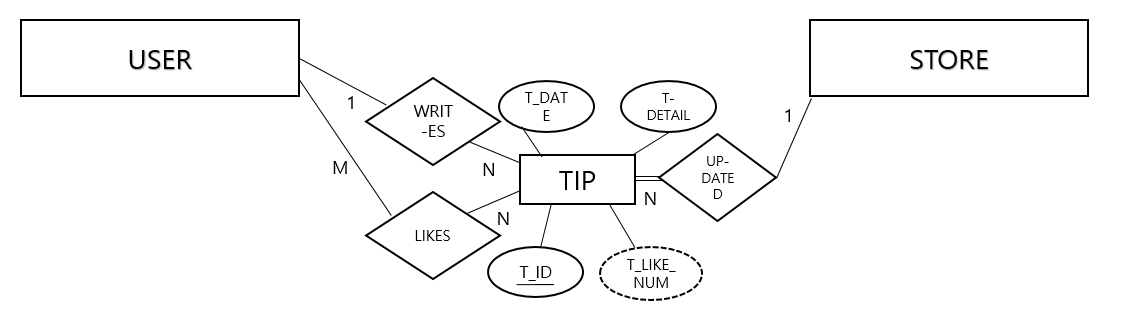
REVIEW\_EVALUATION

https://lh3.googleusercontent.com/SfqGfNGPaRFYTd4wOkXFpnyShkFb4w4GBVreCl5TvrYp7gnUjg0bCws9gs2hfZufOa5KM3YNWtW8zWKfYJAGcXXOVZIXmvX2wrp5-bv10iv7y_YAzxQ2V5NdccrGzhi3QGQi_L0W

* U\_ID: 리뷰에 평가를 남긴 사용자의 id / RE\_ID: 평가가 남겨진 업소의 id / U: useful 평가 / F: funny 평가 / C: cool 평가
* REVIEW는 USER와 WRITES, STORE와 UPDATED의 relationship을 가진다. 한 명의 사용자가 여러 개의 리뷰를 작성할 수 있고 리뷰는 하나의 작성자에 해당하므로 WRITES는 1:N relationship이다. 마찬가지로 UPDATED도 1:N relationship임을 알 수 있고 WRITES, UPDATED에 대해 REVIEW가 N-side에 해당하므로 각각 USER의 U\_ID(PK)와 STORE의 S\_ID(PK)를 foreign key로써 가져올 수 있다.
* REVIEW는 USER와 REVIEW\_EVALUATE라는 relationship 또한 가진다. 한 명의 사용자는 여러 개의 리뷰에 평가를 남길 수 있고 리뷰 또한 여러 명의 사용자로부터 평가를 받을 수 있으므로 이 relationship은 M:N relationship이다. 따라서 REVIEW\_EVALUATION이라는 relation을 새로 만들어 주었고, 이 relation은 각각 USER, REVIEW의 primary key인 U\_ID, RE\_ID를 foreign key로 가져올 수 있다.
* constraint에 대해 살펴보면, RE\_ID는 리뷰마다 고유한 값이며 null을 허용하지 않으므로 entity integrity constraint를 만족하고, RE\_ID를 참조하는 REVIEW\_EVALUATION의 referential integrity constraint도 만족해야 한다. 또한 REVIEW, REVIEW\_EVALUATION은 각각 U\_ID, S\_ID / U\_ID를 참조하는데, 이 또한 referential integrity constraint를 만족해야 한다. REVIEW\_EVALUATION은 {U\_ID, RE\_ID}를 primary key로 가지는데, 사용자는 한 리뷰에 대해  한 번의 평가만 남길 수 있으므로 unique함을 보장하고 U\_ID, RE\_ID 각각이 null을 허용하지 않으므로 entity integrity constraint 또한 만족한다. 다른 constraint로는 한 리뷰 평가에서 U, F, C의 합은 1 이상 3이어야 하고, 각각은 0 아니면 1의 값을 가진다.

**(R7) TIP: 사용자가 남긴, 업소를 방문할 때 알아두면 좋은 팁에 대한 relation**

**(R8) TIP\_LIKE: 사용자들이 다른 사용자들의 팁에 남긴 '좋아요'에 대한 relation**

****

TIP

**https://lh6.googleusercontent.com/_WUU9rNmYpKxdXrMTiKYArnJVJUgG6fWCIfAuy6okOqsEbbE7F4EXwHMpS3dhxssVwOm8IqZGuekTsCLXNmIj9CC_QrmMZvIFDnFsHHnqSrWch42R712DOhrQpPHZ2AzjnQ9Qvng**

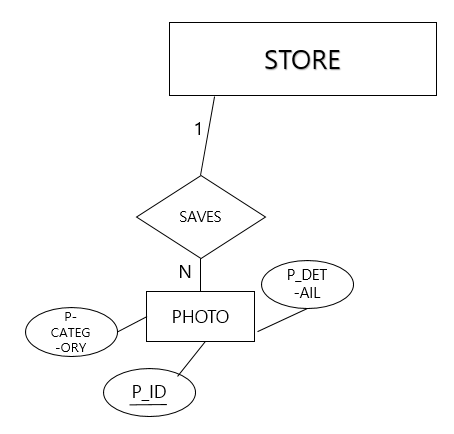
* T\_ID: 팁의 id / U\_ID: 작성자의 id / S\_ID: 팁이 남겨진 업소의 id / T\_DETAIL : 팁의 내용 / T\_DATE: 팁이 작성된 일시

TIP\_LIKE

**https://lh5.googleusercontent.com/cZzOqSGwZy6UfAUekHTn93dCET3jppAmUFkBu1O61wPlXiXLwa9Hkm7wFNF9fIJM50ckVBcXnsV7Dii4OkeNXAGLYVVY3dHUAj2LHBOr-rfFkp3oiR7gJkK88XnPFi-j2_ukQwj4**

* U\_ID: 팁에 좋아요를 표시한 사용자의 id / T\_ID: 좋아요를 받은 팁의 id
* TIP은 작성한 사용자와 팁이 남겨지는 업소의 정보 또한 저장해야 하므로 USER의 U\_ID(PK)와 STORE의 S\_ID(PK)를 참조하는 U\_ID, S\_ID를 foreign key로 가진다. TIP\_LIKE relation에는 primary key로 {U\_ID, T\_ID}를 지정하였다. 각각 USER의 U\_ID(PK), TIP의 T\_ID(PK)를 참조하는 foreign key의 조합이다.
* 한편, 각 relation을 살펴보면 relationship의 정의가 더 명확해진다. USER는 여러 개의 TIP을 작성할 수 있고, TIP은 작성되는 대상에 해당되므로 WRITES는 1:N relationship이다. STORE 역시 여러 개의 TIP이 하나의 업소를 대상으로 작성될 수 있다는 점에서 UPDATED는 1:N relationship임을 유추할 수 있다. USER와의 WRITES relationship, STORE와의 UPDATED relationship  각각에 대해 TIP이 N-side relation에 해당하므로 그들의 PK를 foreign key로 가져올 수 있었다.
* 한편, 사용자들이 다른 사용자 남긴 팁에 대해서 ‘좋아요’를 누름으로 인해 발생하는 LIKES relationship의 경우, 하나의 사용자가 다수의 팁에 좋아요를 누를 수 있고, 한 사용자의 팁이 다수에 의해 좋아요를 받을 수도 있으므로 M:N relationship으로 연결될 수 밖에 없다. 그렇기에 M:N relationship인 LIKES를 TIP\_LIKES라는 relation을 만들어 schema에 저장한 것이고, 이 relation에 USER와 TIP의 primary key인 U\_ID, T\_ID를 포함할 수 있다.
* 마지막으로, constraints에 관해 논해보고자 한다. 가장 기본적인 domain constraints에 관해서는 3.1장에서 논한 바가 있고, null 처리에 관한 constraints 역시 동시에 진행되었다. 한편, FK인  U\_ID, S\_ID, T\_ID는 referential integrity constraint를 만족해야 한다. 또한, PK인 T\_ID와 (U\_ID, T\_ID) set은 기본적으로 entity integrity constraint를 준수해야 함은 자명하다.

**(R9) PHOTO: 업소가 등록한 사진들에 대한 relation**

****

PHOTO

https://lh5.googleusercontent.com/P9vA3gZ0ymfNA53USA1w8Uh2aKpdniRKWYES7tvo43T7SsIE-_f7LUBOuijj26uUKOoGD70DgxH5RcIVq5YEYC0TlfghmdPibBr5-qInkaftBBgOkdRBLUebsE2YEZ_5nA6fkcah

* P\_ID: 사진이 저장된 링크 / S\_ID: 사진을 업로드한 업소의 id / P\_ID: 해당 사진의 분류 /  P\_DETAIL: 사진의 상세 정보
* PHOTO는 그 자체가 가지는 ID인 P\_ID 하나로도 충분히 unique한 조건을 만족한다. 따라서 P\_ID가 해당 relation의 PK이다. 제시된 문제에 따르면, 사진은 업소가 등록하였기에 S\_ID를 STORE 의 relation으로부터 참조했으며(FK), 업소가 여러 장의 사진을 등록할 수 있으므로 PHOTO와 STORE 의 관계는 1:N으로 정의됨을 알 수 있다. 한편, S\_ID(업소의 고유 번호)와 P\_CATEGORY(사진의 분류), P\_DETAIL(사진의 상세정보)이 각각 PK가 될 수 없는 이유는 다음과 같다. 업소 하나는 여러 장의 사진을 등록할 수 있으므로 unique하지 못하며, 사진의 분류는 서로 같은 업소 간에도, 서로 다른 업소 간에도 겹칠 수 있는 위험성이 있다. 또한, 상세정보의 경우 null값이 가능하므로 entity integrity constraints에 위배되기 때문에 PK의 조건을 만족시킬 수 없다.
* 마찬가지로, Domain constraints는 3.1에 제시된 바처럼 만족하고, P\_ID는 FK이므로 referential integrity constraints를 만족해야 한다.

**(R10-1) USER\_LIKE: 한 사용자가 다른 사용자에게 남긴 ‘좋아요’에 대한 relation**

USER\_LIKE

**https://lh4.googleusercontent.com/hxXyXx6bN38uMZjxi0lYPqtDSfu4Ybwxx8gYIq4VET80LQizlgPq8AOXSmk32Z6lUy4SXp4kaH0xQ365J6FJMVQhV34ytPoY7l6RbciVfvourCcYo7_28NreWx092Ti9btfvw32b**

* U\_ID: 좋아요를 누른 사용자의 id / LIKED\_U\_ID:  좋아요를 받은 사용자의 id

**(R10-2) FOLLOW: 한 사용자가 다른 사용자를 팔로우하는 정보에 대한 relation**

FOLLOW

**https://lh3.googleusercontent.com/ba_oS2ABushlIynOPiGzm_O5dcqN9XGC9jAEtlm5VL8XVm9feMi34RBYpWaSf6coPxVjmhbrGwi8qsTU18AIoXsMRxhvJAVpb8dXKaFlOc3cza_e1-xHT_eROBQTHw4i2vyLnikG**

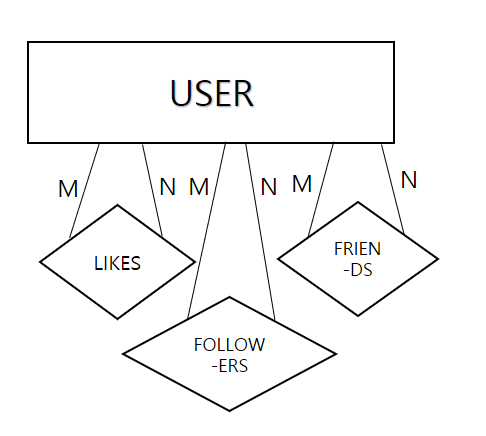
* U\_ID: 팔로우를 하는 사용자의 id / FOLLOWED\_U\_ID: 팔로우를 당하는 사용자의 id

**(R10-3) FRIEND: 사용자들 사이의 ‘친구’ 정보에 대한 relation**

FRIEND

**https://lh6.googleusercontent.com/hpggjSMBHG45bb16zzuTUIrY7Gdz2NWf1Sirs2XZh5hSSiF2mUUqhZ37y71rGw2lunpEWlowchi1RfxeV4lmYNFlpHBgDuD6jfaq-2K_tebQ5Oxdq31R0S0tS8Kj-mnvJAQqdgHO**

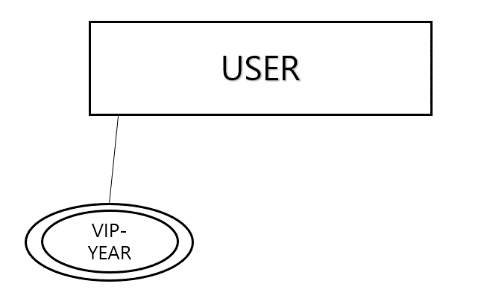
* U\_ID1 및 U\_ID2: 서로 친구 관계에 있는 사용자의 id



**(R10-1~R10-3에 해당)**

* 문제에 제시된 바에 따르면, 사용자들의 좋아요, 팔로우, 친구 관계는 서로 다른 사용자 간의 상호작용으로부터 창출됨을 확인할 수 있다.  즉, 이러한 조건으로부터 USER와 세 가지 relation 간의 관계는 M:N임을 유추할 수 있다.
* 각각의 정보를 살펴보면, 우선 USER\_LIKE relation은 좋아요를 누른 사용자(U\_ID)와 좋아요를 받은 사용자(LIKED\_U\_ID)의 조합으로 Primary Key가 설정되어야 한다. 사용자 한 명이 여러 명에게 좋아요를 누를 수 있으며, 마찬가지로 사용자 한 명이 다른 여러 명의 사용자들로부터 좋아요를 받을 수 있기 때문이다. 이를 FOLLOW와 FRIEND의 relationship까지 같은 원리로 적용시킨다면, FOLLOW는 팔로잉을 하는 사용자(U\_ID)와 팔로우를 받은 사용자(Followed\_U\_ID)의 조합이 PK가 됨을, FRIEND에서는 각자 여러 명의 사용자들과 친구가 될 수 있으므로 서로 친구인 사용자들의 조합(U\_ID1 & U\_ID2)가 PK가 됨을 알 수 있다. 이때, 6개의 attribute 모두는 각각 USER relation 상의 U\_ID를 참조하는 FK이다.
* PK들은 자명하게 entity integrity constraints를 따라야하며, domain에 대해서는 3.1에서 언급한 바와 같다. 한편, 위의 6가지 attributes는 각각의 요소가 FK로 이루어진 만큼 referential integrity constraints를 반드시 준수해야 한다. 추가적으로, FRIEND에 저장되는 U\_ID의 쌍은 (a,b)와 (b,a)가 같은 의미를 지니므로, 중복되는 경우(후자)는 배제하여 불필요한 공간 낭비를 줄일 수 있다.

**(R11) VIP: 사용자가 매년 연말에 받은 VIP 칭호 정보에 대한 relation**

****

VIP

**https://lh3.googleusercontent.com/sbGUsCuyn8WocME8BVCEre6CWfZ7PsjeQbMY0hnrJzt_qA5FjkTPVKshM-NZu9hltYtLxrMcTZFe9mbk5z3wcxnCn37YqpKXHj9mpMNTW-PHzbIAHbqoldmt3f47qo-lIuDCk9h7**

* U\_ID: vip에 선정된 사용자의 id / VIP\_YEAR: 사용자가 vip에 선정된 해
* VIP relation에서 Primary Key는 U\_ID와 VIP\_YEAR의 조합이 될 것이다. 이를 multivalued attribute로 취급하여 따로 relation을 만들어 준 것이다. VIP 칭호를 받은 연도는 사용자에 대한 정보이므로 USER의 U\_ID를 참조하는 foreign key를 가진다. 따라서 referential integrity constraint를 만족해야 한다. 한편, Domain Constraints의 경우 3.1에서 이미 언급한 바를 따른다. 또한 이미 언급한 바와 같이 U\_ID와 VIP\_YEAR의 조합은 VIP relation 상의 PK이므로, entity integrity constraint를 준수해야 한다.

**(R12) 자료형이 명시되지 않은 데이터에 대한 처리**

* data의 자료형이나 포맷, 길이가 명시되지 않은 경우가 있었다. 예를 들어 개/폐점 시간은 포맷은 ‘hh:00’ 꼴의 문자열로 지정하였고, 후기와 같이 문자열이라고 지정은 되어 있지만 그 길이가 명시되지 않은 경우는 전부 조건에서 주어진 대로 최대 255자의 제한을 갖도록 설계하였다. 자세한 내용은 3.1절에서 domain constraint를 설명하면서 언급하였다.

**5. 결론**

이상과 같이 우리는 사이트 A를 위해 ER diagram과 relational DB schema를 설계하였다. 우리가 따른 데이터베이스 설계 원칙을 한 마디로 표현하자면 'Minimizing redundancy while satisfying requirements and constraints'이다. 이러한 원칙을 만족하는 데이터베이스를 구현하기 위해 추가적인 가정들(적합한 자료형의 제시)과 저장 공간을 줄이기 위해 derived attribute 등의 개념을 도입하였다.