تمرین سری ۲ DB (۸۱۰۱۹۴۳۴۶)

DDL	DML
Data Definition Language	Data Manipulation Language
شامل دستوراتی مثل Create، Drop، Alter	شامل دستوراتی مثل Update، Insert، Delete، Select
زیرمجموعه‌ای (بخشی) از SQL که کارهایی مثل ساختن [create]، از بین بردن [delete] و ایجاد تغییرات [manipulation] جدول‌ها را به عهده دارد	دستورات DML دستوراتی هستند (بخشی از زبان SQL هستند) که برای تغییر و اضافه کردن داده‌ها داخل جدول استفاده می‌شوند
دستورات DDL برای ساختن و دربست کردن ساختار کلی پایگاه داده و مدیریت relationها (relation ≡ table) استفاده می‌شوند	دستورات DML برای مدیریت داده‌ها داخل یک Table استفاده می‌شوند
دستورات DDL روی کل یک جدول کار می‌کنند	دستورات DML روی یک یا چند tuple (یعنی سطر یا row) جدول کار می‌کنند

- relationها در SQL، Table گفته می‌شوند
- where که برای مشخص کردن سطرهایی که تغییرات روی آن‌ها اعمال خواهد شد و یا به طور کلی سطرهایی هدف هستند، روی دستورات DDL بی‌معنی هستند در حالی که روی اغلب دستورات DML قابل اعمال هستند.

(۲) مجموعه‌ای از فیلد (Field) ها که می‌توانند یک tuple در relation (جدول) را به صورت یکتا مشخص کنند کلید کاندیدا نام دارد اگر ۲ شرط زیر را داشته باشد:

۱- هیچ ۲ سطر یا tuple ای نباید وجود داشته باشند که به ازای تمام فیلدهای کلید کاندیدا مقدار یکسان داشته باشند

۲- هیچ زیر مجموعه‌ای از فیلدهای کلید کاندیدا نباید بتواند به طور یکتا tuple ها را شناسایی کند

به عنوان مثال اگر داشته باشیم Etudent (sid, sname, nationalid, gpa) در این صورت {sid} می‌تواند یک کلید کاندیدا باشد حال اگر مجموعه‌ای از فیلدها وجود داشته باشد که tuple ها را به صورت یکتا مشخص کند اما شرط ۲ در آن رعایت نشده باشد به آن ابرکلید (superkey) گفته می‌شود به عنوان مثال {sid, sname} یک ابرکلید است زیرا {sid} که زیر مجموعه‌ای از آن است به طور یکتا و بدون نیاز به sname می‌تواند tuple را به صورت یکتا مشخص کند.

حال اگر بیش از ۱ کلید کاندیدا داشته باشیم مثلاً sid و nationalid (شماره ملی) در مثال بالا هر یک می‌توانند کلید کاندیدا باشند، یکی از این کلید ها باید به عنوان کلید اصلی مشخص شوند (primary key) این کلید علاوه بر مشخص کردن سطرها در جدول به صورت یکتا، فیلدی است

که در سایر جدول ها برای اشاره به این جدول (برای refer کردن به این جدول) استفاده می‌شود، یعنی مثلاً اگر sid کلید اصلی

از در جدول درس‌ها، ... در sid برای اشاره به سطر خاصی در جدول Student استفاده می‌شود.

به عبارت دیگر FK در (Foreign Key) در referring relation باید با PK در (Primary Key) در referred relation یکی باشد (match شود)

این به این معنی است که باید نوع و تعداد ستون‌های جای داشته باشند نه اینکه الزاما نام یکسان داشته باشند.

(۲۰) view مفهومی مشابه relation دارد (یعنی یک table است) با این تفاوت که اعضای آن (tuple های آن) در حافظه ذخیره شده اند بلکه تعریف آن (definition آن) در حافظه ذخیره شده است، اعضای آن در صورت نیاز براساس تعریف محاسبه می‌شوند.

برای توصیف view در بعضی سایت‌ها از آن با نام virtual table یاد کرده بود که با توجه به تعریف بالا منطقی نظر می‌رسد.

در درست کردن یک view می‌توان فقط از ستون های یک جدول استفاده کرد یا از ستون‌های چندین جدول متصل، شکل کلی دستور ایجاد (create کردن) یک view به صورت زیر است.

```
Create View  view_name As
    Select  col1 , col2 , ....
    From    table_name
    Where   condition ;
```

این یک view می‌توان در query ها یا در ساخت view های دیگر هم استفاده کرد.

۲ دلیل اصلی برای استفاده از view داریم:

۱- حفظ logical data independance 2- امنیت (security)

- توضیح مورد 2: گفته بودیم که یکی از قدرت‌های استفاده از DBMS ها امنیت آن‌ها است و اینکه هر فرد با توجه به سطح دسترسی خود می‌تواند اطلاعات را مشاهده کند، view ها این امکان را فراهم می‌کنند یعنی به ما توانایی این را می‌دهند که فقط بخشی از اطلاعات یا جدول ها را نمایش دهیم و با تعریف view های مختلف سطوح جزئیات مختلف و ستون های مختلف و ... را نمایش دهیم و با این کار امنیت پایگاه داده بالا رود و هرکسی تمام اطلاعات را نبیند.

- توضیح مورد ۱: در اصل ۱ و ۲ درباره‌ی استقلال نیز یکی و منطقی داده ها صحبت کردیم، view ها به ما این امکان را می‌دهند که اگر شمای relation تغییر کرد ما با تعریف یک view با شکل شمای قبلی این تغییر را از دید کاربر مخفی نگه داریم و استقلال داده را ایجاد کنیم.

اطلاعات را مشاهده کند ، view ها این امکان را فراهم می‌کنند یعنی به ما توانایی این را می‌دهند که فقط بخشی از اطلاعات یا جدول‌ها را نمایش دهیم و با تعریف view های مختلف سطوح جزئیات مختلف و ستون‌های مختلف و ... را نمایش دهیم و با این کار امنیت پایگاه داده بالا رود و هر کسی تمام اطلاعات را نبیند.

- توضیح مورد ۱: در فصل ۱ و ۲ درباره‌ی استقلال فیزیکی و منطقی داده‌ها صحبت کردیم ، view ها به ما این امکان را می‌دهند که اگر شمای relation تغییر کرد ما با تعریف یک view به شکل شمای قبلی این تغییر را از دید کاربر مخفی نگه داریم و استقلال داده را ایجاد کنیم.

(۴) موجودیت ضعیف یا weak entity ، موجودیتی است که primary key ندارد یعنی attribute ای ندارد که به تنهایی بتواند آن‌ها را شناسایی کند. برای شناسایی tuple های یک موجودیت ضعیف باید از ترکیب primary key موجودیت owner و partial key موجودیت ضعیف استفاده کرد. برای یک موجودیت ضعیف ۲ نکته باید رعایت شوند:

۱- موجودیت owner و موجودیت ضعیف باید در یک رابطه‌ی شناسا با هم باشند

۲- موجودیت ضعیف باید در رابطه (۱) total paraipitation داشته باشد.

به لحاظ مفهومی موجودیتی است که در ارتباط با موجودیت دیگری است و به تنهایی برای ما ارزشی ندارد و فقط در ارتباط با موجودیت اصلی معنی دارد و می‌خواهیم در صورت پاک شدن موجودیت owner این اطلاعات موجودیت ضعیف هم پاک شود.

نازنین صبری - تمرین ۲ DB - (۸۱۰۱۹۴۳۴۶)

به عنوان مثال اطلاعات اقوام یک کارمند در حالت عادی و به تنهایی برای شرکت ارزشی ندارد و فقط در ارتباط با آن کارمند ارزش خواهد داشت در صورت اخراج شدن کارمند می‌خواهیم اطلاعات اقوام او نیز پاک شود. پس اقوام به صورت موجودیت ضعیف کارمند تعریف می‌شود.

مثال دیگر موجودیت ضعیف می‌تواند اطلاعات اتاق‌های یک ساختمان است اگر ساختمان خراب شود دیگر اطلاعات اتاق‌ها برای ما ارزشی ندارد و اطلاعات اتاق‌ها هم باید پاک شوند پس می‌توانیم اتاق را به عنوان موجودیت ضعیف ساختمان تعریف کنیم.

⑤ محدودیت دامنه یا Domain constraint یعنی پایگاه داده به ما امکان insert کردن داده‌هایی که دارای نوع (type) متفاوت باشند را نمی‌دهد.

Integrity Constraint شامل ۳ مورد می‌شود: 1) key constraint 2) Foreign key Cons. 3) General Cons.

۱ ← مشخص کردن primary key یا مواردی که حتماً باید unique باشند

۲ ← مشخص کردن اینکه کدام فیلد FK است و به کدام جدول و PK آن اشاره می‌کند و در زمان delete چه اتفاقی باید رخ دهد

* PK بودن شرط not null را در دل خود دارد.

```
Student ( ssn  char (11),
          name  char (20),
          Primary key (ssn) )

Register ( ssn  char (11)  not NULL,
           courseNo  char (11)  not NULL,
           primary key (ssn, courseNo),
           Foreign key (ssn) References Student on delete cascade,
           Foreign key (courseNo) References Course on delete no action )

Course ( courseNo  char(11),
         name  char(20),
```

استدلال: اگر دانشجو از سیستم حذف شود دیگر اطلاعات درس‌های او برای ما ارزشی ندارد در حالی که اگر دانشجو در سیستم باشد و درس بخواهد از سیستم حذف شود نباید اجازه دهیم پاک شود چون دانشجو آن را برداشته است و رکوردهای او دچار مشکل می‌شود

استدلال: اگر درسی بخواهد از سیستم پاک شود اطلاعات پیش‌نیازی آن هم باید پاک شود یعنی مثلاً اگر دارا

```
Student ( SSn    char (11) ,
          name   char (20) ,
          Primary key (SSn) )

Register ( SSn       char (11)  not NULL ,
           CourseNo  char (11)  not NULL ,
           Primary key (SSn , courseNo) ,
           Foreign key (SSn) References Student on delete cascade ,
           Foreign key (courseNo) References Course on delete no action )
```

استدلال : اگر دانشجو از سیستم حذف شود دیگر اطلاعات درس های او برای ما ارزشی ندارد در حالیکه اگر دانشجو در سیستم باشد و درس بخواهد از سیستم حذف شود نباید اجازه دهیم پاک شود چون دانشجو آن را برداشته است و رکورد های او دچار مشکل می شود

```
Course ( courseNo   char(11) ,
         name       char(20) ,
         year       Date یا integer ,
         term       char (11) ,
         primary key ( courseNo ) )

PreRequisite ( courseNo     char(11) not NULL ,
               PreCourseNo  char (11) not NULL ,
               Primary Key (courseNo , PreCourseNo) ,
               Foreign key (courseNo) References Course on delete cascade ,
               Foreign key (PrecourseNo) References Course on delete no action )
```

استدلال : اگر درسی بخواهد از سیستم پاک شود اطلاعات پیش نیازی آن هم باید پاک شود یعنی مثلاً "اگر" ۱ پیش نیاز X است اگر X را پاک کنیم دیگر برای ما مهم نیست چه درسی پیش نیاز آن بوده است در حالی که اگر بخواهد ۱ پاک شود نباید اجازه دهیم چون پیش نیاز X است و X بدون ۲ بی معنی خواهد بود پس اجازه پاک شدن آن را نمی دهیم .

④

```
Create View Student-Department AS
    Select  S.S_name , Sup.SV_department
    From    Student S , Supervisor Sup
    Where   S.supervisor_id = sup.SV_id ;
```

SV-department	S-name
supervisor department	Student name

view تعریف شده با اسم Student-Department جدولی به شکل که برای هر دانشجو دپارتمان Supervisor او را مشخص می کند

درست می کند

فرض میکنیم جدول ، صورت بورد باشد:

S-name	Sv-department
John Smit	CS
John Doe	CS
Jerry Swift	Pharm Sci.
John Smit	CE

* بررسی هزینه‌ی delete : اگر گفته شود که سطرهای مربوط به John Smit را پاک کن در آنجایی که S-name یک key نیست پس یکتا نیست و اتصال نگاری بودن هست (همان طور که در جدول بالا دیده می‌شود) پس اگر بخواهیم عمل delete را انجام دهیم باید روی کل جدول Student حرکت کنیم و هر سطری که اسم John Smit را دارد پاک کنیم پس $\Theta(n)$ زمان خواهد برد (n تعداد سطرهای جدول)

حال اگر از ما خواسته شود که تمام سطرهایی که دپارتمان آن‌ها CS است را حذف کنیم نمی‌توانیم کل سطرهای Supervisor را بررسی کنیم هر جا اسم دپارتمان CS بود پاک کنیم چون ممکن است آن استاد ، استاد راهنمای کسی نباشد پس اصلا در جدول نیست. پس اگر بخواهیم حذف را انجام دهیم باید در Student حرکت کنیم برای هر دانشجو به سطر متناظر supervisor_id آن در Supervisor مراجعه کنیم ، اگر نام دپارتمان این استاد CS بود آنگاه سطر مورد نظر را پیدا کرده ایم و می هم چنان نمی‌توانیم اطلاعات آن را پاک کنیم چون پیوند دانشجو با این رشته‌ای دارد و بدون استاد بعضی خواهد بود پس کلا اجازه delete بر اساس نام دپارتمان نباید داده شود.

* بررسی هزینه‌ی Update : اگر از ما خواسته شود که نام دپارتمان استاد John smit را به math تغییر دهیم باید در کل جدول Student حرکت کنیم و برای هر کسی که نام او John smit است (چون کلید اصلی نیست انگار "یکتا نیست") به سطر متناظر استاد او در Supervisor مراجعه کنیم و نام دپارتمان آن را تغییر دهیم. این کار به این شکل باید یک دور روی کل Student حرکت کنیم مشابه حالت delete بر اساس نام $\Theta(n)$ هزینه خواهد داشت با این تفاوت که در Update وقتی سطرها را پیدا کردیم یک مراجعه با استفاده از supervisor_id موجود در آن سطر به جدول دیگری (Supervisor) داریم که این خود هزینه بر است.

* پس با توجه به بررسی های انجام شده در بالا نتیجه می‌گیریم که هزینه‌ی Update کردن روی این view بیشتر از هزینه‌ی delete است.

(۷) آزمون

۷) از آنجا که به نظر می‌رسد رابطه ISA ، Covering Constr. را دارد یعنی هر بازیگر یا star است یا other Actor می‌توانیم به جای ۳ جدول آن را به صورت ۲ جدول پیاده سازی کنیم ولی رابطه playIn نیز باید ۲ بار پیاده سازی می‌شد که برای جلوگیری از این اتفاق تصمیم گرفتیم ISA را به صورت ۳ relation پیاده سازی کنیم و PlayIn هم یکبار پیاده سازی شود

∴ level اگر عدد است type آن می‌تواند integer باشد اگر یک رشته است type آن می‌تواند char(10) باشد

```
Create Table Actors ( Actor_id   char(11),
                      Salary   real,
                      Actor_name   char(20),
                      Primary key (Actor_id)) ;

Create Table Stars ( level   integer یا char(10),
                     Actor_id   char(11),
                     Primary key (Actor_id),
                     Foreign key (Actor_id)
                     references Actors on delete
                     cascade ) ;
```

∴ اگر tuple مربوط به Actor-id پاک شود دیگر اطلاعات اضافه آن در Stars نیز برای ما ارزشی ندارد پس delete را cascade می‌کنیم

نازنین سبزه - تمرین ۲ DB - (۸۱۰۱۹۴۳۴۹)

```
Create Table Other_Actors( phone_num   char(11),
                           main_job    char(20),
                           Actor_id    char(11),
                           Primary Key (Actor_id),
                           Foreign Key (Actor_id) references Actors on delete cascade);

Create Table Movie ( movie_id   char(10),
                     name       char(20),
                     Primary Key (movie_id));

Create Table Filming_Cast_WorkIn ( Cast_id  char(10),
                     name  char (20),
                     position  char (20),
                     movie_id  char(10) not NULL,
                     Primary key ( movie_id, cast_id),
                     Foreign Key (movie_id) references Movie on delete cascade);

Create Table PlayIn ( Actor_id   char(11)  not NULL,
                      movie_id   char(10)  not NULL,
                      primary Key (Actor_id, movie_id),
                      Foreign Key (Actor_id) references Actors on delete no action,
                      Foreign Key (movie_id) references Movie on delete no action);
```

طبق اسلایدهای استاد برای نشان دادن موجودیت ضعیف کل موجودیت ضعیف و رابطه‌ی شناسای آن به یک table تبدیل می‌شوند

∴ برای موجودیت‌های ضعیف همواره هنگام delet، cascade می‌شود چون موجودیت ضعیف بدون owner آن برای ما ارزشی ندارد.

علت اینکه برای هر دو مورد no action را انتخاب کرده ام این است که فیلم‌هایی که بازیگر بازی کرده بخشی از رزومه‌ی بازیگر هستند و نباید بتوانند پاک

Foreign key (Actor_id) references Actors on delete no action,
Foreign key (movie_id) references Movie on delete no action);

علت اینکه برای هردو مورد no action را انتخاب کرده ایم این است که فیلم‌هایی که بازیگر بازی کرده بخشی از رزومه‌ی بازیگر هستند و نباید بتوانند پاک شوند از طرفی بازیگر نیز بخشی از کمپانی است و فیلم باید کمپانی بازیگرهای خود را در Credit آن داشته باشد پس نباید بتوان اطلاعات بازیگری را که در فیلمی بازی می‌کند پاک کرد.

-- بخش عملی --

به دلیل تعداد زیاد attribute ها آن‌ها را در شکل نکشیده‌ام.