نازنین صبره - تمرین ۱ DB

① الف) مزایای DBMS بر اساس اسلایدهای استاد و کتاب مرجع:

1- Data Independence استقلال داده ⇐ برنامه های مختلف و یا کاربران مختلف نیاز ندارند همه تمام جزئیات مختلف را ببینند، سیستم های مدیریت پایگاه داده این امکان را میدهند که view های مختلف و جزئیات متفاوتی را به کاربران/برنامه های مختلف نشان دهیم

2- Efficiant data Access دسترسی به داده بهینه ⇐ DBMA ها از تکنیک های مختلف برای ذخیره و استفاده ی بهینه ی داده استفاده میکنند

3- Data integrity & security استفاده از DBMS به ما این اجازه را میدهد که شرایط مختلفی را روی داده ها اعمال کنیم هم چنین مانع دسترسی افراد به داده هایی میشود که اجازه ی دیدن آن ها را ندارند

4- Data Administration طوری داده ها را مدیریت میکند که دسترسی به آن ها کمترین زمان را ببرد و مدیریت داده ها به نحوی که به طور بهینه از حافظه استفاده شود

5- Concurrent Access طوری دسترسی هم زمان را مدیریت میکند که کاربران فکر میکنند فقط خودشان در حال دسترسی به DB اند هم چنین از غلط شدن داده ها در صورت دسترسی همزمان و تغییر هم زمان جلوگیری میکند

6- Crash recovery در صورتی که سیستم crash کند DBMS از از دست رفتن داده جلوگیری میکند و همه ی داده ها از بین نمی روند

7- Reduce App-developement Time DBMS کارهای مربوط به ذخیره ی مناسب و دسترسی در زمان مناسب و ... را هندل میکند و برنامه نویس دیگر لازم نیست نگران این موارد باشد و پروسه ی نوشتن برنامه سریع تر پیش میرود

خلاصه ی خودم به کمک یک مثال (مقایسه ی استفاده و عدم استفاده از DBMS و بیان مزیت ها):

فرض میکنیم میخواهیم اطلاعات یک بانک را نگه داریم، اگر از DBMS استفاده نکنیم میتوانیم اطلاعات را در فایل ها ذخیره کنیم ولی چون حجم داده ها خیلی بالا است میتوانیم آن ها را براساس شعبه از هم جدا کنیم در این صورت اگر مشتری به شعبه ای غیر از شعبه ای که حساب را باز کرده مراجعه کند و بخواهد کاری انجام دهد باید یک یک فایل ها را بررسی کنیم تا اطلاعات او را پیدا کنیم و این کار به دلیل حجم زیاد زمان زیادی میبرد (مورد ۲ و ۴ بالا) علاوه بر آن علاوه بر کدی که باید برای خدمات مختلف بانک بزنیم باید کدی بزنیم که ذخیره سازی و دسترسی به اطلاعات و به روز رسانی و ... را انجام دهد در حالی که DBMS میتوانست این کار را انجام دهد (مورد ۷ بالا) هم چنین اگر ۲ نفر به طور همزمان روی اطلاعات یک نفر کار کنند ممکن است اطلاعات غلط شود یا یکی از آن ها درست ثبت نشود (مورد ۵ بالا) و اگر در حین کار سیستم crash کند قبل از اینکه فایل را به درستی بسته و ذخیره کرده باشیم اطلاعات از دست میرود و شاید نتوانیم آن اطلاعات را دوباره برگردانیم (مورد ۶ بالا) هم چنین ممکن است بخواهیم مثلاً فقط مدیران شعبه ها به حساب های دولتی دسترسی داشته باشند و این محدودیت نیاز دارد کدهای اضافه بنویسیم در حالی که DBMS میتواند آن را هندل کند (مورد ۱ و ۳ بالا) هم چنین در سیستم های فایلی شاید بخواهیم برای جلوگیری از اشتباه شدن داده ها به علت دسترسی هم زمان آزاد جلوی دسترسی هم زمان را بگیریم و این باعث میشود که افراد مجبور باشند منتظر بمانند ولی DBMS این محدودیت را ندارد (مورد ۵)

ب) به وجود آوردن استقلال داده یکی از مزیت‌های DBMS است. استقلال فیزیکی داده‌ها به این معنی است جزئیاتی مثل چگونگی ذخیره شدن داده‌ها در دیسک، ساختار فایل‌های داده‌ها و نحوه‌ی index شدن داده‌ها از دید کاربر دور نگه می‌دارد. استقلال منطقی داده‌ها به این معنی است که حتی اگر نحوه‌ی دسته‌بندی داده‌ها عوض شود و ساختار داده‌ها تغییر کند و مثلاً نحوه‌ی شکستن جدول‌ها و... کاربر هم چنان همان پاسخ تمیز را می‌گیرد از نظر او چیزی عوض نشده است. یعنی اگر مثلاً فرمت جدول را عوض کنیم، این عوض شدن فرمت تاثیری روی نحوه‌ی داده یا نحوه‌ی کار کردن کاربر با DBMS نخواهد داشت. استقلال فیزیکی داده‌ها این امکان را فراهم می‌کند که بتوانیم داده‌های فیزیکی (بیت‌های ذخیره شده در حافظه) را جابجا کنیم، نحوه‌ی ذخیره سازی آن را تغییر دهیم یا ... بدون اینکه ارتباط منطقی داده‌ها تغییری کند. یعنی مثلاً اگر بخواهیم حافظه‌ای (دیسکی) که داده‌هایی آن ذخیره شده‌اند را upgrade کنیم این اتفاق تاثیری روی schema و شمای منطقی داده نمی‌گذارد.

* استقلال فیزیکی: شمای logical (منطقی) می‌تواند عوض شود بدون اینکه روی شمای خارجی و نحوه‌ی دیده شدن توسط کاربر تاثیر بگذارد
* استقلال منطقی: شمای فیزیکی داده‌ها می‌تواند عوض شود بدون اینکه روی شمای conceptual تاثیری بگذارد

« جزئیات »

اهمیت استقلال داده: 1) تغییر در یک بخش فقط نیازمند تغییرهای همان بخش است و روی بقیه بخش‌ها تاثیری ندارد و این پیچیدگی‌هایی که می‌شود و به روزرسانی یک بخش شکستن برای سایر بخش‌ها ایجاد می‌کند 2) جزئیات پیاده سازی از دید کاربر پنهان می‌ماند 3) هزینه‌ی به روزرسانی هر بخش کم‌تر می‌شود چون نیازمندی‌های بخش‌ها نیست

4) انعطاف‌پذیری برنامه بیشتر می‌شود (بازهم به همان علت 1)

مثلاً FAT32 محدودیت 4GB دارد.

... انعطاف پذیری برنامه بیشتر می‌شود (بازهم به همان علت ۱)

مثلاً FAT32 محدودیت 4GB دارد.

(۲) الف) بله مناسب است، از طرفی برخی سیستم‌های فایل محدودیت اندازه دارند و 2 TB داده را نمی‌توانند نگه دارند در حالی که این اندازه برای DBMS کاملاً عادی است. از طرفی با اینکه اگر از فایل هم باشد ۵۰۰ بار خواندن کل آن از ابتدا تا انتها کاری ندارد و با یک for loop قابل انجام است ولی چون می‌خواهیم تحلیلی روی داده‌ها انجام دهیم اگر بتوانیم از خاصیت پایگاه داده رابطه‌ای استفاده کنیم و آنها به صورت entity هایی که باهم رابطه دارند ذخیره کنیم و از خاصیت DBMS که پاسخ سریع به query ها است بهره بگیریم سرعت تحلیل داده‌ها و اطلاعاتی که از آن بدست می‌آوریم خیلی بیشتر می‌تواند باشد.

ب) بله مفید است، مثال حساب‌های بانکی یکی از مثال‌های پایه‌ای بود که استاد در کلاس زدند به این علت که مثلاً می‌خواهیم پول از حسابی کم شود ولی به حساب دیگر اضافه نشود و یا در حین محاسبه‌ی موجودی پولی جابجا شود که در صورت استفاده از فایل این اتفاق می‌تواند بیفتد ولی چون DBMS ویژگی‌های concurren Access و امنیت را دارد استفاده از DBMS مفیدتر است. از طرفی نیز ساختار کلی داده‌های بانک شکل entity و relationship بسیار واضح و مناسبی دارد که قرار دادن داده‌ها در این ساختار فهم داده‌ها و منطق نگهداری را بهبود می‌بخشد. مثلاً مشتری‌ها و حساب‌ها entity باشند که هر حساب متعلق به حداقل یک مشتری است (حساب مشترک را هم داریم) و هر مشتری تعدادی حساب دارد و ....

ج) بله مفید است، چون یکی از ویژگی‌های DBMS فراهم کردن view های مختلف برای افراد با سطح دسترسی‌های گوناگون است و استفاده نکردن از DBMS به این معنی خواهد بود که برنامه‌نویس باید حجم زیادی از کد را به برنامه خود اضافه کند تا بتواند این امکان را در صورت استفاده از فایل فراهم کند.

د) با توجه به خواسته‌ی اول (داده‌ی متغیر) چندان مفید نیست ولی با توجه به خواسته‌ی دوم (ضروری بودن امکان از دست ندادن داده) بله مفید است. با توجه به جستجوهایی در اینترنت گفته شده است که پایگاه داده‌های رابطه‌ای برای داده‌های متغیر (همواره در حال تغییر) ساخته نشده است مخصوصاً در حجم‌های داده‌ی بالا چون اضافه کردن یا تغییر یک ستون در یک جدول با توجه به رابطه‌هایی که دارد می‌تواند موجب تغییر تعداد زیادی جدول شود که چون حجم داده زیاد است مدت زمان زیادی می‌برد. از طرفی از مزیت‌های اصلی DBMS ها crash recory است و در حالی که اگر مثلاً فایل‌ها در حین کار به طور درست بسته نشوند ممکن است اطلاعاتی از دست برود یا ابداً درست باز نشوند ولی DBMS ها در صورت رخ دادن مشکل سخت‌افزاری می‌توانند سیستم را از یک حالت درست مجدداً به راه اندازند.

نازنین صبوری - تمرین ۱ DB

(۳) خوب است که تلفن را به عنوان یک موجودیت (entity) تعریف کنیم اگر: ۱- هر فرد می‌تواند بیش از یک شماره تلفن داشته باشد (چون اگر attribute باشد فقط می‌تواند یکی باشد و نمی‌تواند تعداد بیشتر از ۱ داشته باشد چون فقط اگر بگذاریم که ۲ تا بودن شکلی ندارد اگر کسی بخواهد ۳ شماره تلفن داشته باشد چه می‌شود؟ اگر ۴ تا داشت چی؟ پس اگر می‌خواهیم هر فرد بتواند بیشتر از یک شماره تلفن داشته باشد باید entity تعریف کنیم)

۲- اگر بخش‌های مختلف شماره تلفن برای ما معنی دار باشد، یعنی مثلاً اگر شماره تلفن ۳۸۰۶۲۴ ۵۵۵ بود ۵۵۵ اول نشان دهنده منطقه باشد یا مثلاً زوج یا فرد بودن شماره‌ها برای ما مهم باشد و بخواهیم تحلیلی روی آن‌ها انجام دهیم یا ... به طور کلی زمانی که چیزی به عنوان خصیصه تعریف می‌شود نباید به بخش‌های کوچیک شکسته شود

ولی اگر شماره تلفن‌ها یکی بود و فقط با کل string شماره و نه بخش‌هایی از آن کار داشتیم خوب است که به عنوان خصیصه (attribute) تعریف شود

۴) الف) نمودار داده شده در کتاب: (با توجه به شکل ← بین درس و درس دادن مشارکت کامل است و ضرورت ← نیست)

← هر درس حداکثر توسط یک استاد درس داده می‌شود — اگر ضرورت ← بود → هر درس دقیقاً توسط یک استاد درس داده می‌شود

• سناریوی که مشکل دارد: در دنیای واقعی یک درس می‌تواند توسط اساتید مختلف ارائه شود که این طراحی این اجازه را به ما نمی‌دهد.

آ — این نمودار به نحوی تعداد درس‌هایی که استاد ارائه می‌کند را نشان نمی‌دهد یعنی مثلاً به entity استاد attribute تعداد دروس ارائه شده ندارد ولی اگر جدول درست شده با توجه به این طرح را نگاه کنیم می‌توانیم تعداد سطرهایی که نام استاد در آن‌ها تکرار شده است بشماریم و این تعداد درس‌هایی است که استاد مسئول تدریس آن‌ها است. query که این تعداد را به ما می‌دهد می‌تواند به شکل زیر باشد:

```
select count(*)
from   teaching      ← اسم جدول
where  name = "اسم استاد" ;
```

پس این طراحی این اجازه را می‌دهد که یک استاد چندین درس را تدریس کند

ب — میزان حقوق کلی که استاد می‌گیرد از روی اطلاعات entity استاد مشخص نمی‌شود یعنی با توجه به نمودار فرض شده است که استاد با توجه به هر درسی که ارائه می‌کند حقوق جداگانه‌ای می‌گیرد و اگر کل حقوق را بخواهیم باید تمام درس‌هایی که استاد درس می‌دهد را پیدا کرده و حقوق‌ها را با هم جمع کنیم

اگر بخواهیم حقوق به ازای درس تعریف نشده باشد و یک حقوق کل برای استاد داشته باشیم فارغ از اینکه چه تعداد درس یا چه درس‌هایی ارائه می‌کند باید حقوق را به عنوان یک attribute در entity تعریف شود.

ج — این مسئله با توجه به نمودار مشخص می‌شود یعنی چون حقوق به عنوان ویژگی درس دادن تعریف شده است پس می‌توانیم پیدا کنیم که استاد به ازای تدریس هر درس چه حقوقی می‌گیرد

ب) نمودار در یک جمله: هر دانشجو دقیقاً یک اتاق را رزرو کرده است و هر دانشجو دقیقاً در یک ساختمان ساکن است و هر ساختمان حداقل یک دانشجو ساکن دارد

• سناریوی که مشکل دارد: با توجه به شکل هر ساختمان حداقل یک دانشجو ساکن دارد ولی ممکن است در دانشگاه یکی از خوابگاه‌ها در حال تعمیر باشد یا به دلیل وجود اتاق خالی در ساختمان‌های دیگر در یکی از ساختمان‌ها را بسته باشیم و ساکن نداشته باشد که با توجه به محدودیت‌ها این ممکن نیست.

آ- این مسئله مشخص است یعنی چون هر دانشجو دقیقاً در ساختمان ساکن است پس مشخص خواهد بود که در کدام ساختمان است.

ب- اگر به جدول حاصل از رابطه (ساکن در) نگاه کنیم تعداد سطرهایی که ساختمان b است را بشماریم تعداد ساکنین ساختمان b مشخص می‌شود

یعنی

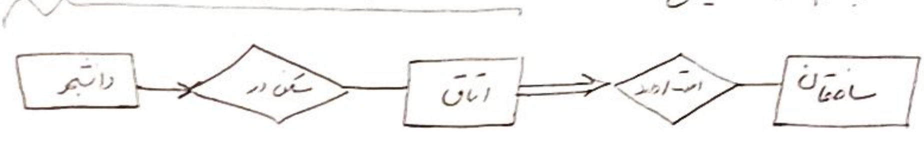
```
select  count(*)
from    lives_in
where   building = b ;
```

ج- این سوال ممکن است قابل پاسخ نباشد چون اگر دانشجویی در اتاق ۳ ساکن باشد چون برای دانشجو هم اتاق و هم ساختمان مشخص است کافی است ۲ جدول را باهم ترکیب کنیم و ببینیم اتاق ۳ در کدام ساختمان است ولی اگر اتاق ۳ خالی باشد و دانشجویی در آن ساکن نباشد چون جدولی با سطرهای (اتاق | ساختمان) نداریم و دانشجو لینک ارتباط این ۲ است پس نخواهیم توانست بگوییم اتاق ۳ در کدام ساختمان است.

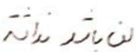
د- این سوال با توجه به بخش ج اما قابل پاسخ دادن نیست یعنی اگر همه‌ی اتاق‌های یک ساختمان توسط دانشجویان پر شده باشند می‌توان ۲ جدول را ترکیب کرد و با تعداد سطرهایی که اسم ساختمان b است را بشماریم ولی اگر اتاق خالی داشته باشیم با توجه به توضیحات ج در جدول نخواهد بود پس نمی‌توان گفت ساختمان دقیقاً چند اتاق دارد.

نموداری که به تمام سوالات پاسخ دهد:

هر دانشجو دقیقاً در یک اتاق است و چون هر اتاق هم دقیقاً در یک ساختمان است مشخص می‌شود هر دانشجو در کدام ساختمان است.



هر اتاق دقیقاً در یک ساختمان است

اگر هر ساختمان حداقل یک اتاق داشته باشد قبلاً ساختمان که فقط سالن باشد نداشته باشیم  تعریف می‌کنیم.

یک مشکل که ⟸ تعریف کردن در این مثال وجود دارد این است که ممکن است دانشجو خوابگاه داشته باشد و اتاق رزرو نکند تا در اتاق ساکن نباشد ولی ⟸ این اجازه را نمی‌دهد در حالی که تعریف کردن به صورت ⟵ باعث می‌شود که دانشجو حداکثر در یک اتاق ساکن باشد ولی مجبور نباشد که حتماً اتاق بگیرد و روش‌های اقامت دیگر هم ممکن باشد

ج) نمودار در یک جمله: هر شناسه (بخش، کارفرما و پروژه) یکتا است.

- سناریوهایی که مشکل دارد: با توجه به اینکه طبق شکل پروژه فقط یک attribute دارد پس این attribute باید همان primary key هم باشد (چون نمی‌تواند primary key نداشته باشد و attribute دیگری هم نداریم) و این مسئله ما را دچار مشکل می‌کند چون آنگاه یک بخش خاص با یک کارفرمای خاص نمی‌تواند ۲ پروژه متمایز را در یک تاریخ یکسان شروع کند چون کلید اصلی یکی است.

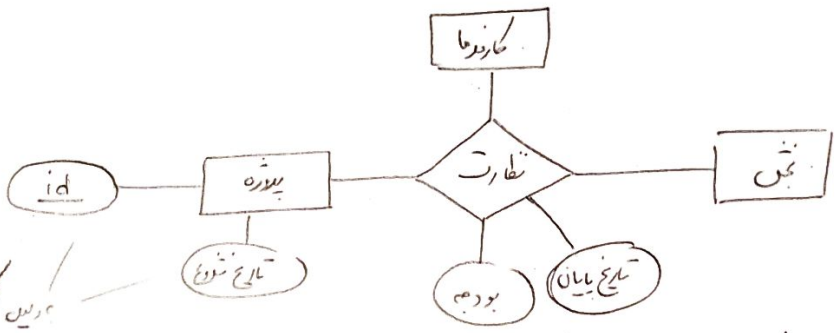
تمرین صدری - تمرین ۱ DB

۱- این سوال مشخص میشه چون بودجه و عنوان و دیگر بخش تعریف شده پس می‌تواند مربوط به یک یا چند نظارت باشد پس بودجه‌ی نظارت مشخص نیست

۲- زمان پایان نظارت نیز با توجه به عنوان مشخص نیست چون دیگر نظارت تعریف شده است.

۳- این سوال قابل پاسخ است یعنی کلی وقت یک query بنویسیم که تعداد سطرهایی که نام کارفرما یک اسم را دارد

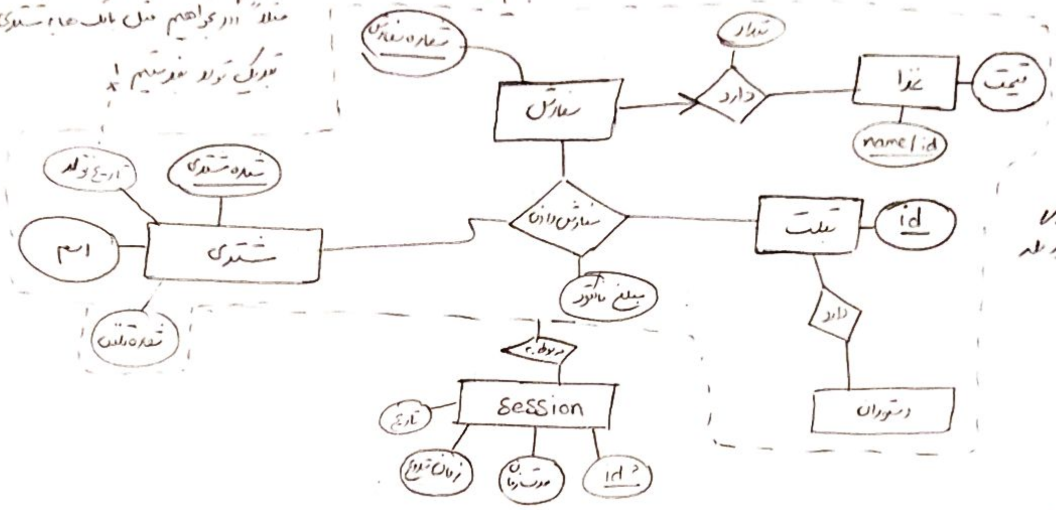
نمودار تصحیح شده



دیدگاه بودجه‌ی بخش می‌توانیم جمع بودجه‌ی تمام نظارت‌هایی که بخش X در آن‌ها هست را حساب کنیم

و در اینجا مثلاً می‌توانیم جمع بودجه‌ی تمام سفارش‌هایی که غذای X در آن‌ها هست را حساب کنیم.

مثلاً اگر بخواهیم مثل بالا به مشتری‌ها email تبریک تولد بفرستیم.



(۵) با توجه به نمودار کشیده شده هر سفارش مواردی شامل یک غذا می‌شود و مبلغ را نیز در خود دارد. علاوه بر آن، رستوران تعدادی تبلت دارد.

session زمان شروع دارد تا اگر مشتری چند بار در یک روز مراجعه کرد مشکلی نداشته باشد.

* هر ۳ تایی (سفارش، مشتری، session) باید یکتا باشد ولی اگر رابطه ۳ تایی بگذاریم به ازای مشتری و session یکسان و سفارش‌های گوناگون سطرهای مختلفی خواهیم داشت ولی اگر سفارش‌ها گوناگون باشیم یعنی شماره سفارش‌های گوناگون.

هر مشتری یک تبلت و سفارش دارد که همه‌ی این‌ها در ارتباط با این session است، چون در صورت سوال گفته شده که به مشتری تبلت داده می‌شود و قطعاً این مشتری است که سفارش خود را می‌دهد؛ session و تبلت‌ها هم متعلق به رستوران اند.

(۶) مشتری می‌تواند کتاب امانت بگیرد که تاریخ امانت گرفتن به عنوان ویژگی امانت گرفتن تعریف شده و نام کتاب و شماره مشتری از ویژگی‌های کتاب و مشتری و هر کتاب دقیقاً در یک قفسه قرار دارد که ظرفیتی تا چند ... برای آن قفسه مشخص شده است.