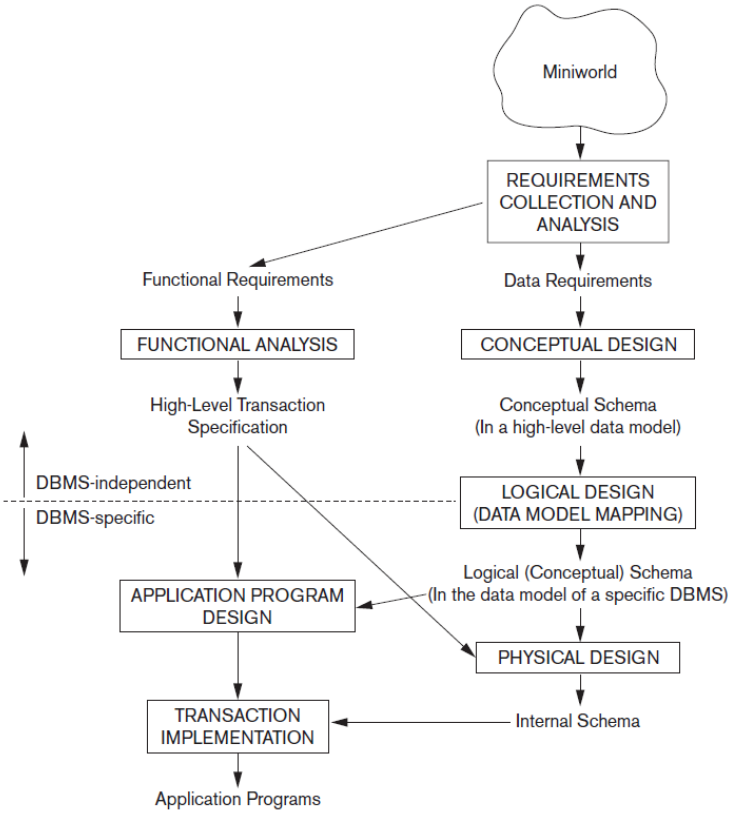
پروژه‌ی درس پایگاه‌داده

خشایار میرمحمدصادق ۹۲۱۰۰۹۸۶
آریا ادیبی ۹۲۱۱۰۴۷۶
سید سروش هاشمی ۹۳۱۰۰۹۷۷

چکیده

شکل ۱ فازهای اصلی طراحی یک پایگاه داده را نشان می‌دهد [۱، ص. ۶۱]. با کمک گیری از این نمودار و خواست‌های پروژه، فازهای خود را طراحی کردیم.



شکل ۱: نمودار ساده شده‌ی فازهای اصلی طراحی پایگاه‌داده

در فاز اول مدل مفهومی [1] را ارایه کردیم که برای این کار از نمودار ER استفاده شده است. همچنین کلیت خواسته‌های پروژه را شناسایی کرده و ذکر کردیم.

در فاز دوم "طراحی مفهومی" ارائه شده در فاز قبل را به مدل رابطه‌ای تبدیل کردیم و از روی نمودار ER جدول‌ها را به دست آوردیم. البته لازم به ذکر است که این جدول‌ها را در فاز سوم بررسی و آزمایش کرده‌ایم و امکان تغییر آن‌ها در فاز سوم وجود داشته است. همچنین در این فاز "دید"های مختلف را تشخیص داده و بیان کردیم.

در فاز سوم جزییات بررسی و آزمایش جدول‌های خود به همراه جزییات پیاده‌سازی آورده شده است. این فاز دارای قسمت‌های زیر است:

- **دامنه‌ی صفات**

در این قسمت دامنه‌ی تمامی صفات جدول‌ها را مشخص می‌کنیم. دقت کنید که اگر در قسمت "نرمال سازی" [2] تصمیم به تغییر دامنه‌ای گرفته باشیم، هم در قسمت نرمال سازی و هم در انتهای این قسمت ذکر می‌کنیم.

- **نرمال سازی** [2]

پس از بررسی دامنه‌ها در این قسمت جدول‌های خود را آزمایش کرده و نرمال می‌کنیم. برای نرمال سازی از تعریف تعمیم یافته‌ی آن استفاده می‌کنیم [۱، ص. ۴۸۳]. برای روشن شدن این تعریف‌ها آن‌ها را در زیر می‌آوریم. به دلیل اینکه کلماتی خاص در این تعریف‌ها استفاده شده است و معادل دقیق فارسی آن‌ها را نمی‌دانستیم از فارسی سازی آن‌ها صرف نظر کرده‌ایم.

Definition. *A relation schema R is in **first normal form (1NF)** if it has no multivalued or nested relations; in another words all of it's attributes are atomic.*

Definition. *A relation schema R is in **second normal form (2NF)** if every nonprime attribute A in R is not partially dependent on any key of R.*

Definition. *A relation schema R is in **third normal form (3NF)** if, whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.*

Definition. *A relation schema R is in **BCNF** if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R.*

دقت کنید که اگر رابطه‌ای در BCNF باشد در 3NF نیز است.

- **محدودیت‌های جامعیتی**

پس از نرمال کردن محدودیت‌های جامعیت جدول‌ها را در این قسمت ذکر می‌کنیم.

- **ادعاها و بررسی‌ها**

در این قسمت "ادعا"ها [3] و "بررسی"های [4] گذاشته شده در پیاده سازی را ذکر می‌کنیم.

- **رهاناها و تابع‌ها**

در آخر هم "رهانا"ها [5] و "تابع"های [6] پیاده‌سازی شده را در این قسمت می‌آوریم.

پس از فاز سوم در قسمتی به نام "نیازهای وب گاه از پایگاه داده" به ۴ مورد زیر می‌پردازیم.

- رویه‌ی بارگذاری داده‌ها و آزمایش‌ها
- پرس‌وجوهای [7] لازم
- کپی پشتیبان [8] از پایگاه‌داده
- دیدهای پیاده‌سازی شده

---

[1] conceptual modeling
[2] normalization
[3] assertion
[4] check
[5] trigger
[6] function
[7] query
[8] backup

برای هر ۲ پروژه این مطالب را می‌نویسیم. به علت تشابه زیاد در خیلی از موضوع‌ها توضیح‌های پروژه‌ی ۲ را به صورت بسیار خلاصه‌تر می‌آوریم. در آخر نیز بخشی را به دلیل انتخاب PostgreSQL به عنوان DBMS و بخشی را به نیازهای نرم‌افزاری اختصاص داده‌ایم.

فهرست مطلب‌ها

فهرست نگاره‌ها

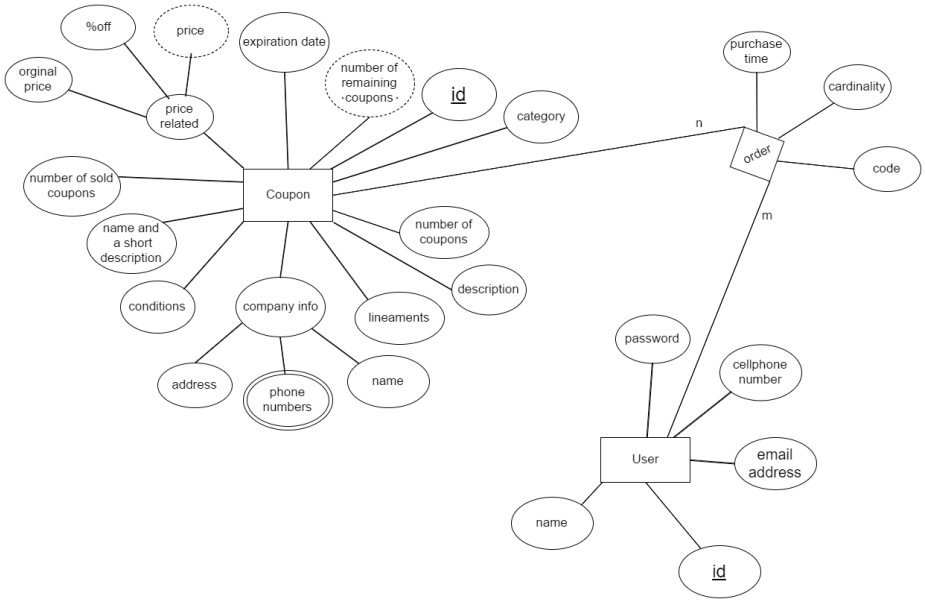
فهرست جدول‌ها

۱ پروژه اول

فازهای پروژه به ترتیب در زیر آمده‌اند.

۱.۱ فاز اول

این پروژه مربوط به طراحی مکانی مناسب برای ارائه‌ی کوپن‌های مختلف از جاهای مختلف است. این پروژه در نهایت باید چیزی شبیه به http://takhfifan.com شود. نمودار ER این پروژه در شکل ۲ آورده شده است.



شکل ۲: نمودار ER پروژه‌ی اول

۱.۱.۱ داده‌های مورد نیاز

در این قسمت داده‌های مورد نیاز [۹] این پروژه که در پایگاه داده باید ذخیره شوند شناسایی شده و همراه با معنی آن‌ها آورده شده است. به دلیل انگلیسی بودن کد نهایی و نمودار ER ارائه شده اسم‌ها را معادل سازی فارسی نمی‌کنیم.

- **Coupon**

  لازم است کپن‌ها را در وب گاه ذخیره کنیم. برای هر کپن این اطلاعات را ذخیره می‌کنیم:

  - **id**

    به هر کپن شناسه‌ای یکتا می‌دهیم که با آن به راحتی بتوان آن را شناسایی کرد.

  - **category**

    کپن‌ها را بر اساس موضوعی که مربوط آن‌ها است دسته بندی می‌کنیم. این دسته‌ها عبارتند از:

    * تخفیف‌های جدید
    * رستوران و کافی‌شاپ

[9] data requirements

* هنر و تئاتر
* تفریحی و ورزشی
* آموزشی
* سلامتی و پزشکی
* زیبایی و آرایشی
* مسافرتی
* کالا

- **description**

قسمتی برای توضیحات می‌گذاریم که گذارنده‌ی کپن بتواند در مورد کپن خود توضیح دهد و آن را تبلیغ کند.

- **lineaments**

قسمتی برای ویژگی‌های شرکت یا مکان ارائه دهنده سرویس است.

- **company info**

در این قسمت مشخصات مکانی که کپن مربوط به آن جا است پذاشته می‌شود. این قسمت شامل

* name
* phone numbers
* address

است.

- **conditions**

در این قسمت شرایط استفاده‌ی کپن توسط گذارنده‌ی کپن ذکر می‌شود.

- **name and a short description**

این جا نام کپن (و شاید توضیح بسیار مختصری) گذاشته می‌شود که برای نمایش نام کپن از آن استفاده بشود. اطلاعات جزیی‌تر به کاربران وب گاه زمانی نشان داده می‌شود که وارد کپن مورد نظر شده باشند.

- **number of coupons**

در این جا تعداد کل کپن‌ها (چه فروخته شده و چه فروخته نشده) گذاشته می‌شود.

- **number of sold coupons**

تعداد کپن‌های فروخته شده را مشخص می‌کند. این اطلاعات را برای نشان دادن محبوبیت کپن و آمار گیری استفاده می‌کنیم. اگر این تعداد برابر با تعداد کپن موجود بود کپن از وب گاه برداشته می‌شود.

- **number of remaining coupons**

تعداد کپن‌های مانده را مشخص می‌کند. این خصوصیت مشتق است و ذخیره نمی‌شود.

- **price related**

در این قسمت اطلاعات مربوط به قیمت کپن نگه داری می‌شود که شامل بخش‌های زیر است:

* original price
* %off
* price

دقت کنید که price یک داده‌ی مشتق است و ذخیره نمی‌شود.

- **expiration date**

زمانی است که کپن از وب گاه برداشته می‌شود. زمانی که یک کپن در وب گاه است نمی‌تواند بیشتر از ۳ ماه باشد.

- **User**

کاربران وب گاه در این قسمت ذخیره می‌شوند که خصوصیات زیر از آن‌ها نگه داری می‌شود:

  - **id**

  یک شناسه که آن‌ها را به صورت یکتا مشخص کند. چنین شناسه‌ای را تنها برای بهبود سرعت خصوصیتی جدا گرفتیم و به عنوان مثال می‌شد email این افراد را به عنوان شناسه انتخاب کرد.

  - **name**

  اسم کاربر به صورت کامل نگه‌داری می‌شود.

  - **email address**

  در اینجا پست الکترونیکی این افراد نگه داری می‌شود. دقت کنید که برای ورود به وب گاه باید پست الکترونیکی و رمز عبور را وارد کرد.

  - **cell phone number**

  در اینجا شماره‌ی همراه کاربران نگه‌داری می‌شود.

  - **password**

  رمز عبور کاربر برای ورود به وب گاه.

- **order**

در این طراحی نیاز است که خریدهای کاربران را ذخیره کنیم. این کار با برقرار کردن رابطه‌ای بین کاربران و کپن صورت می‌گیرد که خصوصیات زیر هم ذخیره می‌کند.

  - **purchase time**

  زمان خرید را مشخص می‌کند.

  - **cardinality**

  تعداد کپن خریداری شده را مشخص می‌کند. دقت کنید که تنها ۱ کپن است که حال ممکن است تعدادی از آن خریداری شده باشد. این تعداد نباید از تعداد کپن باقی مانده بیشتر باشد.

  - **code**

  کدی ۱۲ رقمی به کاربر داده می‌شود که با استفاده از آن بتواند از کپن خود استفاده کند.

۲.۱.۱ کارکردهای مورد نیاز

در این قسمت کارکردهای مورد نیاز [10] این پروژه در پایگاه داده شناسایی شده و توضیح داده شده است.

- کاربران باید بتوانند بر حسب نام کوپن یا گذارنده‌ی آن در وب گاه جست‌وجو کنند.

۳.۱.۱ توضیحات مربوط به وب گاه

چند نکته در مورد نحوه‌ی عملکرد وب گاه است که لازم دانستیم این جا اشاره کنیم:

- هر فردی برای گذاشتن کوپن خود در وب گاه ۲ راه دارد. راه اول این است که با مسئولین مربوطه تماس بگیرد و یا قراردادی با وب گاه ما ببندد. راه دوم این است که در قسمت "کسب و کار خود را تبلیغ کنید" وارد شود و فرم انتهای صفحه را پر کند که در آن مشخصات لازم خود و کوپن مورد نظر نوشته می‌شود. اطلاعات لازم برای کوپن را به راحتی می‌توان از نمودار ER فهمید. در هر ۲ راه بعد از بررسی کوپن اگر کوپن پذیرفته شد در وب گاه قرار می‌گیرد و در غیر این صورت نمی‌گیرد.
- در این وب گاه (اگر DBA staff را کنار بگذارید) تنها ۱ نوع کاربر وجود دارد. کاربران هم می‌توانند خرید و هم کسب و کار خود را تبلیغ کنند. توضیحات بیشتر در قسمت "دید"های فاز دوم وجود دارد.

[10] functional requirements

- در این وب گاه هزینه هر سبد خرید مستقیم باید به وب گاه ریخته شود و در انتها پس از بررسی درست بودن همه ضوابت، کدی به کاربر داده می‌شود که می‌تواند با آن کپنی را که خریده است را استفاده کند و این وب گاه است که هزینه‌ی آن عمل را به شرکت یا مکان ارائه دهنده می‌دهد.

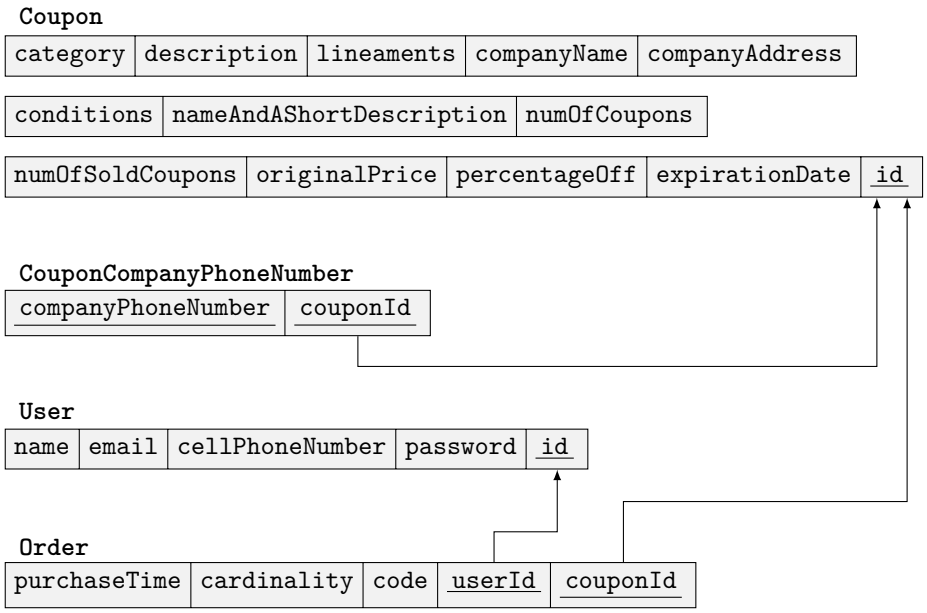
  وب گاه درصد ثابتی را از هر خرید بر می‌دارد و بقیه مبلغ را به مکان ارائه دهنده می‌دهد.

۲.۱ فاز دوم

در این فاز ابتدا جدول‌های مدل رابطه‌ای مربوط را طراحی می‌کنیم سپس به شناسای "دید"های[11] لازم در پایگاه داده می‌پردازیم.

۱.۲.۱ طراحی جدول‌های مدل رابطه‌ای

در این بخش جدول‌های مدل رابطه‌ای را با توجه به نمودار ER شکل ۲ طراحی می‌کنیم. دقت کنید که در فاز سوم ممکن است این طراحی به دلیل نرمال سازی تغییر کند. طراحی انجام شده در شکل ۳ قابل مشاهده است.



شکل ۳: جدول‌های مدل رابطه‌ای طراحی شده برای پروژه‌ی اول از روی نمودار ER شکل ۲

۲.۲.۱ شناسایی دیدها

همان طور که در بخش ۳.۱.۱ به طور مختصر اشاره شد اگر DBA staff را در نظر نگیریم نیاز شناسایی شده برای وب گاه وجود تنها یک نوع کاربر است که در این صورت یک دید بیشتر نخواهیم داشت. این دید برای وب گاه ما است و به تمامی جدول‌ها همان‌گونه که هستند نیاز دارد. یعنی تمام جدول‌های پایه را نیاز دارد و دیدی متفاوت نسبت به این جدول‌ها ندارد.

۳.۱ فاز سوم

در این فاز به آزمایش جدول‌های طرح شده و جزییات پیاده‌سازی خواهیم پرداخت. توضیحات بخش‌های مختلف این فاز در قسمت "چکیده" آمده است.

[11] views

۱.۳.۱ دامنه‌ی صفت‌ها

تمامی دامنه‌های انتخاب شده را می‌توان از عنصرهای "اتمی" [۱۲] فرض کرد (در واقع اتمی هستند ولی در SQL گاهی ممکن است با دستوراتی مثل LIKE به صورت موقت اتمی در نظر گرفته نشوند) جزییات آن‌ها را می‌توانید از [۲] نگاه کنید. چند دامنه وجود که آن‌ها را با * مشخص کرده‌ایم و توضیحات لازم در مورد آن‌ها را بعد از جدول ۱ آورده‌ایم.

Coupon

category:: couponCategories*	description:: text
lineaments:: text	companyName:: nameDomain*
companyAddress:: text	conditions:: text
nameAndAShortDescription:: character varying(150)	
numOfCoupons:: integer	numOfSoldCoupons:: integer
originalPrice:: money	percentageOff:: integer
expirationDate:: date	id:: serial

CouponCompanyPhoneNumber

companyPhoneNumber:: phoneNumDomain*	couponId:: integer

User

name:: nameDomain*	email:: emailDomain*
cellPhoneNumber:: cellPhoneNumDomain*	
password:: PasswordDomain*	id:: serial

Order

purchaseTime:: date	cardinality:: integer
code:: character varying(12)	
userId:: integer	couponId:: integer

جدول ۱: دامنه‌ی صفت‌های جدول‌های شکل ۳

توضیح دامنه‌های باعلامت‌دار:

couponCategories

همان‌طور که در فاز ۱ گفته شد هر کپن باید در دسته‌ای قرار داشته باشد از این رو یک type ساخته شده است که از نوع enum است دستور ساخت آن به همراه مقدارهایش از این قرار است.

```
CREATE TYPE "couponCategories" AS ENUM
   ('RESTAURANT_COFFEESHOP',
    'ART_THEATER',
    'ENTERTAINMENT_SPORT',
    'EDUCATIONAL',
    'HEALTH_MED',
    'COSMETIC',
    'TRAVEL',
    'GOODS');
```

[12] atomic

nameDomain

این دامنه را ساختیم برای این که اگر بعداً خواستیم قانون خاصی روی نام‌ها قرار دهیم این کار به سادگی ممکن باشد. در حال حاضر تعریف ساده‌ای دارد که از این قرار است.

```
CREATE DOMAIN "nameDomain" AS
   character varying(50);
```

phoneNumDomain

این دامنه را ساختیم که قانون‌های تلفن را روی آن بگذاریم و اگر بعداً تصمیم گرفتیم آن را عوض کنیم مثلاً کد منطقه و غیره را هم قبول کنیم به سادگی بتوان این کار را کرد. در حال حاضر فرض کردیم تنها شماره‌های تهران مورد قبول است (دقت کنید بسیار ساده می‌توان این محدودیت را حتی به جهانی تغییر داد ولی پیچیدگی بی‌فایده را درست می‌کرد که به این دلیل این محدودیت را گذاشتیم). در اینجا شماره‌ها باید ۸ رقم باشد. تعریف آن در زید آمده است.

```
CREATE DOMAIN "phoneNumDomain" AS
   character varying(8)
   CONSTRAINT "allDigitConstraint"
      CHECK (VALUE ~ '[\d+]')
   CONSTRAINT "phoneLenConstraint"
      CHECK (length(VALUE) = 8);
```

cellPhoneNumDomain

این دامنه بسیار شبیه به phoneNumDomain است و توضیحات مشابه است. تعریف در زیر آمده است.

```
CREATE DOMAIN "cellPhoneNumDomain" AS
   character varying(11)
   CONSTRAINT "allDigitConstraint"
      CHECK (VALUE ~ '[\d+]')
   CONSTRAINT "cellPhoneNumLenConstraint"
      CHECK (length(VALUE) = 11);
```

emailDomain

این دامنه درست شد تا محدودیت‌های موجود در پست‌های الکتریکی را در آن وارد کنیم که اگر مکان دیگری لازم شد این محدودیت‌ها را لازم نباشد دوباره وارد کنیم. در حال حاضر خیلی از محدودیت‌ها را کنار گذاشتیم و تنها ۲ تا از آن‌ها را بررسی می‌کنیم. تعریف از این قرار است.

```
CREATE DOMAIN "emailDomain" AS
   character varying(320)  NOT NULL
   CONSTRAINT "emailConstraint"
      CHECK (VALUE ~ '_%@_%._%');
```

passwordDomain

در این دامنه تمام شرایط دلخواه برای یک گذرواژه را می‌گذاریم. در ابتدا این دامنه را به صورت زیر تعریف کردیم.

```
CREATE DOMAIN passwordDomain AS
   character varying(25)   NOT NULL
   CONSTRAINT "passwordConstraints"
      CHECK (VALUE LIKE '%[0-9]%' AND
             VALUE LIKE '%[A-Z]%' AND
             VALUE LIKE '%[!@#$\%a^&*()-\_+=.,;:''"`~]%'
                ESCAPE '\' AND
             length(VALUE) >= 8);
```

بعد تصمیم گرفتیم که از سیاست django برای hash کردن گذرواژه پیروی کنیم و همچنین محدودیت‌های بالا را در نظر نگرفتیم. تعریف جدید از این قرار است.

```
CREATE DOMAIN "passwordDomain" AS
   character varying(128)  NOT NULL;
```

۲.۳.۱ نرمال سازی

در ایتدا از روی معنای صفت‌ها وابستگی‌های تابعی هر جدول را باید به دست بیاوریم. دقت کنید ما در طراحی خود code را یکتا برای هر order قرارا داده‌ایم همچنین email هم که یکتا است. نتیجه‌ی این شناسایی وابستگی‌های تابعی در شکل ۴ آورده شده است.
همان طور که در ۱.۳.۱ نیز توضیح داده شد دامنه‌های تمامی صفت‌ها اتمی بوده و با تعریف 1NF سازگاری دارد. [۱۳] همچنین به سادگی با توضیحات داده شده می‌توان دید که جدول‌ها در 2NF، 3NF و BCNF صدق می‌کنند. از این رو جدول‌ها نرمال هستند و تغییری در آن‌ها نمی‌دهیم.

۳.۳.۱ محدودیت‌های جامعیتی

محدودیت‌های جامعیتی شامل ۲ نوع "جامعیت موجودیت" [۱۴] و "جامعیت ارجاعی" [۱۵] است. کلیدهای تمامی جدول‌ها و همچنین جامعیت‌های ارجاعی بین آن‌ها را در شکل ۳ مشخص کرده‌ایم. در پیاده‌سازی با SQL این کلیدها و جامعیت‌های ارجاعی را مشخص کرده‌ایم. از این رو خود DBMS این جامعیت‌ها را چک می‌کند و از برقرار بودن آن اطمینان حاصل می‌کند.

۴.۳.۱ ادعاها و بررسی‌ها

در این قسمت ادعاها [۳] و بررسی‌ها [۴] استفاده شده در پیاده‌سازی را ذکر می‌کنیم. دقت کنید که از محدودیت‌های معمول (که در این ۲ دسته نیستند و به علت فراوانی در SQL جداگانه قرار داده‌شده‌اند) NOT NULL و DEFAULT در این مستندات ذکر نشده است. اما بقیه یعنی PRIMARY KEY، FOREIGN KEY و UNIQUE در بخش‌های قبلی اشاره شده‌اند. برای دیدن NOT NULL و DEFAULT می‌توانید به پایگاه‌داده نوشته شده مراجعه کنید.
در این بخش به قانون‌های گذاشته شده برای به روز رسانی "کلیدهای خارجی" [۱۶] هم اشاره می‌کنیم.

ادعاها

در پیاده‌سازی جایی از ادعا استفاده نشده است چرا که در صورت برقرار نبودن شرط، کل جدول را قفل می‌کرد که این موضوع در جایی مورد نیاز ما نبود.

---

[۱۳] لازم به ذکر است که اتمی بودن یک صفت بستگی به دید طراح از داده دارد. برای مثال معمولاً در نرمال سازی email را اتمی نمی‌گیرند و به بخش نام و نامنه تقسیم می‌کنند که دامنه‌های یکسان با اسم‌های متفاوت یا اسم‌های یکسان با دامنه‌های متفاوت چند بار ذخیره نشود. ما در این جا این پیچیدگی را غیر ضروری دانسته و اتمی در نظر می‌گیریم.

[14] entity integrity
[15] referential integrity
[16] foreign key

شکل ۴: وابستگی‌های تابعی جدول‌های مدل رابطه‌ای پروژه‌ی اول. این وابستگی‌ها از روی معناشناسی صفت‌ها به دست آمده‌اند.

بررسی‌ها و قانون به روز رسانی کلیدهای خارجی

در این‌جا لازم به ذکر است که چندی از این بررسی‌ها در تعریف دامنه‌ها است (و حتی ممکن است در typeها هم باشد ولی ما استفاده نکردیم) که آن‌ها را در بخش ۱.۳.۱ ذکر کردیم. در اینجا به بقیه آن‌ها اشاره می‌کنیم. تعریف این بررسی‌ها خود گویای موضوع هستند و نیازی به توضیح ندارند از این رو توضیحات بسیار مختصری در مورد آن‌ها می‌دهیم.

در جدول Coupon

```
CONSTRAINT "expiredDateConstraints"
   CHECK ("expirationDate" > CURRENT_DATE AND
          "expirationDate" < (CURRENT_DATE+
                              '3 months'::interval))
```

اطمینان حاصل می‌کند که زمان باقی ماندن کپن در وب گاه درست و کمتر از ۳ ماه باشد.

```
CONSTRAINT "numConstraints"
   CHECK ("numOfCoupons" >= 0 AND
          "numOfSoldCoupons" >= 0),
```

این شمارها نمی‌توانند عددهایی منفی باشند.

```
CONSTRAINT "percentageConstraint"
   CHECK ("percentageOff" >= 0 AND
          "percentageOff" <= 100)
```

درصد تخفیف به دلیل درصد بودن باید عددی بین ۰ تا ۱۰۰ باشد.

در جدول CouponCompanyPhoneNumber

```
CONSTRAINT "couponIdFK" FOREIGN KEY ("couponId")
   REFERENCES "Coupon" (id)
   ON UPDATE CASCADE
   ON DELETE CASCADE
```

دلیل آبشاری گذاشتن این به روز رسانی‌ها روشن است چرا که بدون وجود کپن دلیلی برای نگه‌داری این اطلاعات نیست.

در جدول Order

```
CONSTRAINT "couponIdFK" FOREIGN KEY ("couponId")
   REFERENCES "Coupon" (id)
   ON UPDATE CASCADE
   ON DELETE RESTRICT
```

در اینجا به روز رسانی آبشاری را انتخاب کردیم جز در حذف کردن که اجازه‌ی حذف شدن کپن یا کاربری را نمی‌دهیم مگر اینکه تمامی سفارشات مربوط به آن‌ها انجام شده باشد.

```
CONSTRAINT "userIdFK" FOREIGN KEY ("userId")
   REFERENCES "Users" (id)
   ON UPDATE CASCADE
   ON DELETE RESTRICT
```

دلیل مشابه قبلی دارد.

```
CONSTRAINT "cardinalityConstraints"
   CHECK (cardinality > 0 AND
          cardinality <=
               "numOfRemainingCoupons"("couponId")),
```

در این‌جا کار بررسی روشن است. تابع numOfRemainingCoupons() تابعی است که خود ساختیم و در بخش ۵.۳.۱ آن را توضیح خواهیم داد.

```
CONSTRAINT "codeConstraints"
   CHECK (length(code) = 12)
```

در اینجا مطمین می‌شویم که code ما از ۱۲ رقم تشکیل شده باشد.

۵.۳.۱ رهاناها و تابع‌ها

در این قسمت رهاناها و تابع‌های پیاده سازی را توضیح می‌دهیم.

تابع‌ها

در بخش ۴.۳.۱ به تابع زیر برخورد کردیم.

```
CREATE OR REPLACE
   FUNCTION "numOfRemainingCoupons" ("couponId" integer)
RETURNS integer AS
   $BODY$
   SELECT ALL "numOfCoupons"- "numOfSoldCoupons"
   FROM "Coupon"
   WHERE "Coupon".id = "couponId"
   $BODY$
LANGUAGE sql
```

همان طور که به سادگی از تعریف و حتی اسم آن پیداست شمار کپن‌های باقی مانده از کپنی خاص را بر می‌گرداند.

رهاناها

رهاناهای به روز رسان کپن‌های جدید

```
CREATE TRIGGER "insertCouponTrigger"
AFTER INSERT
ON "Coupon"
FOR EACH ROW
EXECUTE PROCEDURE "updateNewCoupons"();
```

این رهانا بعد از هر درجی در جدول Coupon جدول NewCoupons را به روز رسانی می‌کند. این جدول مسئول نگه‌داری کپن‌های تازه وارد شده است. اگر کپن جدیدی وارد شود تنها کپن‌های همان روز در این جدول باقی می‌مانند و بقیه حذف می‌شوند اما اگر کپن جدیدی درج نشود جدول در هر روزی بدون تغییر باقی می‌ماند. با این جدول در قسمت ۴.۴.۱ آشنا خواهید شد.

این رهانا تابع مخصوصی به نام updateNewCoupons() (که مختص به رهاناها است [۳]) را صدا می‌زند. تعریف این تابع به این صورت است.

```
CREATE OR REPLACE FUNCTION "updateNewCoupons"()
RETURNS trigger AS
   $BODY$
   BEGIN
   INSERT INTO newCoupons
   VALUES(NEW.id, CURRENT_DATE);

   DELETE
   FROM newCoupons
   WHERE insertDate < CURRENT_DATE;

   RETURN NEW;
   END
   $BODY$
LANGUAGE plpgsql
```

کار این تابع همان‌طور که از اسمش پیداست به روز رسانی NewCoupons است که نحوه‌ی به روز رسانی توضیح داده شد.

رهاناهای به روز رسان شمار کپن‌های خریداری شده

```
CREATE TRIGGER "insertOrderTrigger"
   AFTER INSERT
   ON "Order"
   FOR EACH ROW
   EXECUTE PROCEDURE "updateNumOfSoldCoupons"();
```

این رهانا وظیفه دارد که بعد از هر خرید میزان کپن‌های فروخته شده را به روز رسانی کند. برای این کار از تابع زیر استفاده می‌کند.

```
CREATE OR REPLACE FUNCTION "updateNumOfSoldCoupons"()
RETURNS trigger AS
   $BODY$
   BEGIN
   UPDATE "Coupon"
   SET "numOfSoldCoupons"=
      "numOfSoldCoupons" + NEW.cardinality
   WHERE id = NEW."couponId";

   RETURN NEW;
   END
   $BODY$
LANGUAGE plpgsql
```

۴.۱ نیازهای وب گاه از پایگاه داده

در این بخش به توضیح چهار بخش زیر می‌پردازیم.

- رویه‌ی بارگذاری داده‌ها و آزمایش‌ها
- پرس‌وجوهای [۷] لازم
- کپی پشتیبان [۸] از پایگاه‌داده
- دیدهای پیاده‌سازی شده

۱.۴.۱ رویه‌ی بارگذاری داده‌ها و آزمایش‌ها

رویه‌ی بارگذاری:
برای بارگذاری داده در پایگاه داده‌ها، برنامه‌ی adminای تهیه شده بود که به دلیل پیچیدگی زیاد راه‌اندازی آن، کار کردن با آن توضیح داده نمی‌شود. به همین دلیل یک برنامه به زبان python نوشته شده است که به طور مصنوعی داده‌هایی را تولید می‌کند که تولید آن‌ها وظیفه‌ی DBA Staff است. به دلیل داشتن پیشنیازهای نرم‌افزاری، برای آشنایی با نحوه‌ی اجرای این کد به بخش ۱.۴ مراجعه کنید.

رویه‌ی آزمایش:
از آن جا که تمام ساختارها و پرس‌وجوهای پیاده‌سازی شده به منظور استفاده در وب گاه نوشته شده‌اند، آزمایش کردن و درست کار کردن آن‌ها معادل آزمایش کردن و درست کار کردن تمامی قابلیت‌های وب گاه است. برای آزمایش کردن وب گاه در هنگام

development، عملیات آزمایش دو بار انجام شده است. اولین بار در هنگام نوشتن صفحه‌ی مربوط به یک قابلیت، درست اجرا شدن آن مورد آزمایش قرار گرفته است. بار دوم، بعد از تمام شدن نوشتن همه‌ی صفحه‌ها، با گذر کردن در وب گاه و استفاده از هر قابلیتی که در آن موجود است، همه چیز مورد آزمایش قرار گرفته است.

۲.۴.۱ پرس‌وجوهای لازم

در این بخش برای هر کدام از پرس‌وجوهای لازم، توصیف تابع مربوط به آن در کد نوشته شده برای وب گاه، توضیح فارسی نیازها و طرز کار آن و در نهایت کد SQL آن آمده است. در ابتدا چند اصطلاح را برای راحتی توضیح تعریف می‌کنیم که به این صورت هستند:

- خلاصه‌ی یک تخفیف:
منظور از خلاصه‌ی یک تخفیف انتخاب ستون‌های id، nameAndAShortDescription، numOfCoupons، numOfSoldCoupons، originalPrice و percentageOff از جدول Coupon می‌باشد.

- [تمام] یک تخفیف:
منظور از [تمام] یک تخفیف اطلاعات تمام ستون‌های جدول Coupn به همراه شماره تلفن‌هایی است که از پیوند این جدول با جدول CouponCompanyPhoneNumber بر اساس کلیدش به دست می‌آیند.

توضیح آخر این که در بعضی از پرس‌وجوها از عبارت «%s» استفاده شده است. در چنین پرسش‌هایی نیاز به یک متغیر خارجی بوده است که زمان اجرا به رابط پایگاه داده پاس داده می‌شود.

۱. **get_categories():**
وظیفه‌ی این تابع برگرداندن نام دسته‌های مختلف کوپن‌هاست که توسط پرس‌وجویی از جدول pg_enum به دست می‌آید. لازم به ذکر است که برای نام دسته‌هایی که در این بخش برگردانده می‌شوند برای تغییر وب گاه کافی نیست، چون این عبارت‌ها به زبان انگلیسی هستند و در پیاده‌سازی وب گاه هم نیاز است که این عبارت‌ها از انگلیسی به فارسی و هم از فارسی به انگلیسی ترجمه شوند.

پرس‌وجوی انجام شده:

```
SELECT enumlabel
FROM pg_enum;
```

۲. **get_all_discounts(s, quantity):**
وظیفه‌های این تابع برگرداندن خلاصه‌ی تمامی تخفیف‌ها است. برای این که این تعداد ممکن است زیاد باشد در این قسمت، داده‌ها به صفحه‌های مختلف تقسیم می‌شوند. ورودی‌های تابع این موضوع را مطرح می‌کنند که نتیجه‌ی به دست آمده از چندمین تخفیف شروع شود و چند تخفیف را شامل شود. این تابع در جواب خلاصه‌ی تخفیف‌ها خواسته شده را به همراه تعداد صفحه‌های مورد نیاز برای نمایش تمام اطلاعات بر می‌گرداند.

پرس‌وجوی انجام شده برای به دست آوردن تعداد صفحه‌ها:

```
SELECT CEIL(COUNT(id)::float / %s)
FROM "Coupon";
```

پرس‌وجوی انجام شده برای به دست آوردن خلاصه‌ی تخفیف‌ها:

```
SELECT id, "nameAndAShortDescription", "numOfCoupons",
"numOfSoldCoupons", "originalPrice", "percentageOff"
FROM "Coupon"
LIMIT %s OFFSET %s;
```

۳. **get_new_discounts(s, quantity):**
وظیفه‌ی این تابع برگرداندن خلاصه‌ی تخفیف‌های «تازه» است. ورودی‌ها و خروجی‌های این تابع مانند تابع قبل هستند.

تفاوت این تابع این است که چون اطلاعاتی که نیاز دارد در جدول NewCoupons هستند، باید ابتدا این جدول را با جدول Coupon پیوند کند.

پرس‌وجوی انجام شده برای به دست آوردن تعداد صفحه‌ها:

```
SELECT CEIL(COUNT("couponId")::float / %s)
FROM "NewCoupons";
```

پرس‌وجوی انجام شده برای به دست آوردن خلاصه‌ی تخفیف‌ها:

```
SELECT id, "nameAndAShortDescription", "numOfCoupons",
"numOfSoldCoupons", "originalPrice", "percentageOff"
FROM "Coupon" NATURAL JOIN
      (SELECT "couponId" AS id
       FROM "NewCoupons") AS news
LIMIT %s OFFSET %s;
```

۴. **get_discounts(category, s, quantity):**

وظیفه‌ی این تابع برگرداندن خلاصه‌ی تخفیف‌های یک دسته‌ی خاص است که نام آن در ورودی‌ها به تابع پاس داده می‌شود. بقیه‌ی ورودی‌ها و خروجی‌ها و طرز کار این تابع مانند پرس‌وجوهای پیشین است.

پرس‌وجوی انجام شده برای به دست آوردن تعداد صفحه‌ها:

```
SELECT CEIL(COUNT(id)::float / %s)
FROM "Coupon"
WHERE category = %s;
```

پرس‌وجوی انجام شده برای به دست آوردن خلاصه‌ی تخفیف‌ها:

```
SELECT id, "nameAndAShortDescription", "numOfCoupons",
"numOfSoldCoupons", "originalPrice", "percentageOff"
FROM "Coupon"
WHERE category = %s
LIMIT %s OFFSET %s;
```

۵. **search_discounts(keyword, s, quantity):**

وظیفه‌ی این تابع جست‌وجو در میان تمامی تخفیف‌ها و برگرداندن خلاصه‌ی آن‌هایی است که در یکی از ستون‌های nameAndAShortDescription، companyName، description یا category عبارت مورد نظر را داشته باشند. باقی ورودی‌ها و خروجی‌های این تابع مانند توابع پیشین هستند.

پرس‌وجوی انجام شده برای به دست آوردن تعداد صفحه‌ها:

```
SELECT CEIL(COUNT(id)::float / %s)
FROM "Coupon"
WHERE "nameAndAShortDescription" ~~ ('%' || %s || '%')
   OR "companyName" ~~ ('%' || %s || '%')
   OR description ~~ ('%' || %s || '%')
   OR "category"::text ~~ ('%' || %s || '%');
```

پرس‌وجوی انجام شده برای به دست آوردن خلاصه‌ی تخفیف‌ها:

```
SELECT id, "nameAndAShortDescription", "numOfCoupons",
"numOfSoldCoupons", "originalPrice", "percentageOff"
FROM "Coupon"
WHERE "nameAndAShortDescription" ~~ ('%' || %s || '%')
   OR "companyName" ~~ ('%' || %s || '%')
   OR description ~~ ('%' || %s || '%')
   OR "category"::text ~~ ('%' || %s || '%')
LIMIT %s OFFSET %s;
```

۶. **get_one_discount(ID):**

وظیفه‌ی این تابع برگرداندن تمام یک تخفیف خاص است که شناسه‌ی آن پاس داده شده است. این تابع در جواب تخفیف مورد نظر را برمی‌گرداند.

پرس‌وجوهای انجام شده برای به دست آوردن تخفیف مورد نظر:

```
SELECT *
FROM "Coupon"
WHERE id = %s;

SELECT "companyPhoneNumber"
FROM "CouponCompanyPhoneNumber"
WHERE "couponId" = %s;
```

۷. **confirm_user(email, password):**

این تابع وظیفه دارد با دریافت یک آدرس پست الکترونیکی و یک رمز عبور (به صورت خام)،اعلام کند که آیا کاربری با این مشخصات اجازه‌ی ورود به وب گاه را دارد یا خیر. در صورتی که این اجازه وجود داشته باشد، شناسه‌ی کاربر مورد نظر در جواب برگردانده می‌شود و در صورتی که چنین اجازه‌ای وجود نداشته باشد، NULL برگردانده می‌شود.

پرس‌وجوی انجام شده برای تأیید کاربر:

```
SELECT id, password
FROM "User"
WHERE email = %s;
```

پس از این پرس‌وجو رمز عبور برگردانده شده با ورودی مقایسه می‌شود تا اجازه‌ی ورود داده یا رد شود.

۸. **check_existance(email):**

این تابع وظیفه دارد تا بررسی کند آیا کاربری با پست الکترونیکی داده شده وجود دارد یا نه تا در صورت وجود نداشتن، کاربر جدید بتواند با این آدرس پست الکترونیکی ثبت نام کند. خروجی این تابع یک boolean است.

پرس‌وجوی انجام شده برای چک کردن وجود پست الکترونیکی:

```
SELECT 1
FROM "User"
WHERE email = %s;
```

۹. **add_user(**data):**

این تابع وظیفه دارد تا کاربر جدیدی را با اطلاعات وارد شده در سیستم ثبت کند. از آن جا که شماره‌ی تلفن همراه اختیاری است و با توجه به ساختار رابط استفاده شده در کد، اطلاعات کاربر به صورت یک dictionary به تابع پاس داده می‌شود. بعد از ثبت کاربر، شناسه‌ای که به او تعلق گرفته برگردانده می‌شود.

پرس‌وجوهای انجام شده برای افزودن کاربر:

```
INSERT INTO "User"(name, email, password)
VALUES(%s, %s, %s)
RETURNING id;

INSERT INTO "User"(name, email, "cellPhoneNumber", password)
VALUES(%s, %s, %s, %s)
RETURNING id;
```

۱۰. **get_user(ID):**

این تابع وظیفه دارد با دریافت کردن شناسه‌ی یک کاربر، اطلاعات او را برگرداند.

پرس‌وجوی انجام شده برای گرفتن اطلاعات کاربر:

```
SELECT name, email, "cellPhoneNumber"
FROM "User"
WHERE id = %s;
```

۱۱. **update_user(ID, **data):**

وظیفه‌ی این تابع به روز رسانی اطلاعات کاربری است که در ورودی تعیین می‌شود. عملیات انجام شده در این تابع کاملاً مشابه add_user است. اما به دلایلی که هم‌اکنون بیان خواهد شد پیاده‌سازی آن‌ها تفاوت دارد. در فرم "به روز رسانی اطلاعات شخصی" بر خلاف فرم "ثبت نام"، اطلاعات مقادیر قدیمی خود را دارند و می‌توان در هنگام اعتبارسنجی از مقادیر قدیم آن‌ها استفاده کرد. دومین نکته این است که در این جا صرفاً دو ستون شماره‌ی همراه و نام قابلیت عوض شدن دارند و نوشتن پرس‌وجوی آن ساده‌تر است.

پرس‌وجوی انجام شده برای به روز رسانی اطلاعات کاربر:

```
UPDATE "User"
SET "name" = %s, "cellPhoneNumber" = %s
WHERE id = %s;'
```

۱۲. **change_password(ID, password):**

وظیفه‌ی این تابع تغییر رمز عبور کاربر داده شده است. دلیل جدا بودن این تابع از تابع قبلی این است که کاربران در اکثر موارد یا می‌خواهند اطلاعات شخصی خود را تغییر دهند و یا می‌خواند رمز عبور خود را عوض کنند و کمتر پیش می‌آید که این دو کار را هم زمان انجام دهند. به همین دلیل برای راحتی کاربر فرم‌های انجام این دو کار جدا از هم هستند.

پرس‌وجوی انجام شده برای تغییر رمز عبور:

```
UPDATE "User"
SET "password" = %s
WHERE id = %s;
```

۱۳. **submit_order(userID, couponID, count):**

وظیفه‌ی این تابع ثبت سفارشات کاربر است. داخل این تابع کد مورد نیاز کاربر و شرکت ارائه‌دهنده‌ی تخفیف تولید می‌شود تا وارد اطلاعات گردد.

پرس‌وجوی انجام شده برای ثبت سفارش:

```
INSERT INTO "Order"("userId", "couponId",
                    "cardinality", "code")
VALUES(%s, %s, %s, %s);
```

تنها یک تفاوت در این پرس‌وجو با تمام پرس‌وجوهای دیگر وجود دارد: هرچند کد تولید شده یک عبارت تصادفی ۱۲ کاراکتری شامل اعداد و حروف بزرگ و کوچک انگلیسی است، اما برای جلوگیری از تکراری بودن کد و موفق نشدن در ثبت سفارش، پرس‌وجوی بالا تا وقتی که موفق نباشد تکرار می‌شود.

۱۴. **get_orders(ID, s, quantity):**

وظیفه‌ی این تابع نمایش سفارشات یک کاربر است. از آن جا که ممکن است این سفارشات زیاد باشند، مانند پرس‌وجوهای اولیه، این کار در صفحات مختلف انجام می‌پذیرد و خروجی تابع علاوه بر سفارشات درخواست شده، تعداد کل صفحات جواب را نیز در بر خواهد داشت.

پرس‌وجوی انجام شده برای به دست آوردن تعداد صفحه‌ها:

```
SELECT CEIL(COUNT(code)::float / %s)
FROM "Order"
WHERE "userId" = %s;
```

پرس‌وجوی انجام شده برای به دست آوردن سفارشات:

```
SELECT "purchaseTime", "cardinality", "code",
       "nameAndAShortDescription"
FROM "Order" NATURAL JOIN
         (SELECT id AS "couponId",
                       "nameAndAShortDescription"
          FROM "Coupon") AS coupons
WHERE "userId" = %s
LIMIT %s OFFSET %s;
```

۳.۴.۱ کپی پشتیبان از پایگاه داده

کپی‌های پشتیبان در پوشه‌ی Backups قرار داده شده‌اند. در نام این کپی‌ها نوع پشتیبان گیری نیز ذکر شده است. برای آشنایی با نحوه‌ی کار با آن‌ها به [۴] مراجعه کنید.
به طور بسیار خلاصه برای "بازیابی" اطلاعات [۱۷] کارهای زیر باید به ترتیب انجام شود. به نام کاربری توجه کنید.

۱. یک کاربر به نام CouponUser DBA staff درست کنید.

۲. یک پایگاه داده به نام CouponUserDB بسازید.

۳. داخل این پایگاه داده اطلاعات را بازیابی کنید.

۴.۴.۱ دیدهای پیاده‌سازی شده

در این پروژه جدول NewCoupons یک دید در نظر گرفته شده است که به صورت شکل ۵ است.
این جدول مسئول نگه‌داری کپن‌های تازه وارد شده است. اگر کپن جدیدی وارد شود تنها کپن‌های همان روز در این جدول باقی می‌مانند و بقیه حذف می‌شوند اما اگر کپن جدیدی درج نشود جدول در هر روزی بدون تغییر باقی می‌ماند.

ملاحضه. در *PostgreSQL* در حال حاضر نمی‌توان دیدی که صفتی دارد که در جدول‌های پایه نیست را به روز رسانی کرد اگر با صفت کار داشته باشیم. از این رو ما در پیاده‌سازی این دید را به عنوان جدول پایه گرفتیم.

لازم به ذکر است که couponId یک کلید خارجی است که هم درج و هم به روز رسانی آن به صورت cascade است.

[17] restore

NewCoupons

couponId	insertDate

شکل ۵: دیدهای پروژه‌ی اول که در اینجا تنها جدول NewCoupon است.

۲ پروژه دوم

فازهای پروژه به ترتیب در زیر آمده‌اند.

۱.۲ فاز اول

این پروژه مربوط به طراحی مکانی مناسب برای امکان سفارش online از رستوران‌های مناطق مختلف است. این پروژه در نهایت باید چیزی شبیه به https://www.zoodfood.com شود. نمودار ER این پروژه در شکل ۶ آورده شده است.

۱.۱.۲ داده‌های مورد نیاز

در این قسمت داده‌های مورد نیاز این پروژه که در پایگاه داده باید ذخیره شوند شناسایی شده و همراه با معنی آن‌ها آورده شده است. به دلیل انگلیسی بودن کد نهایی و نمودار ER ارائه شده اسم‌ها را معادل سازی فارسی نمی‌کنیم.

- **ZoodFoodUser**

  لازم است بتوانیم کاربرها را در پایگاه داده ذخیره کنیم. برای این موضوع این موجودیت را داریم که دارای ۳ نوع است:

  - **Manager**

    مدیر رستوران است. مدیر رستوران کسی است که رستورانش را در وب گاه نیز باید وارد کند.

  - **Agent**

    کسی است که مسئول نگاه کردن به صفحه‌ی ایجاد شده برای رستورانی است که آن شخص در آن کار می‌کند. در این صفحه سفارش‌ها گذاشته شده‌اند.

  - **Customer**

    کاربران معمولی هستند که برای سفارش غذا به وب گاه مراجعه کرده‌اند.

  لازم به ذکر است که یک کاربر ممکن است در چندتای این‌ها شرکت داشته باشد. صفت‌های این موجودیت‌ها مشابه User پروژه‌ی اول در بخش ۱.۱.۱ است و نیازی نمی‌بینیم توضیحات را تکرار کنیم.

- **restaurant**

  در این موجودیت رستوران‌ها را ذخیره می‌کنیم. نام صفت‌ها خود گویای این که چه صفت‌هایی لازم است ذخیره شود هستند و به دلیل توضیح‌های مشابه در پروژه‌ی اول در این جا آن‌ها را توضیح نمی‌دهیم.

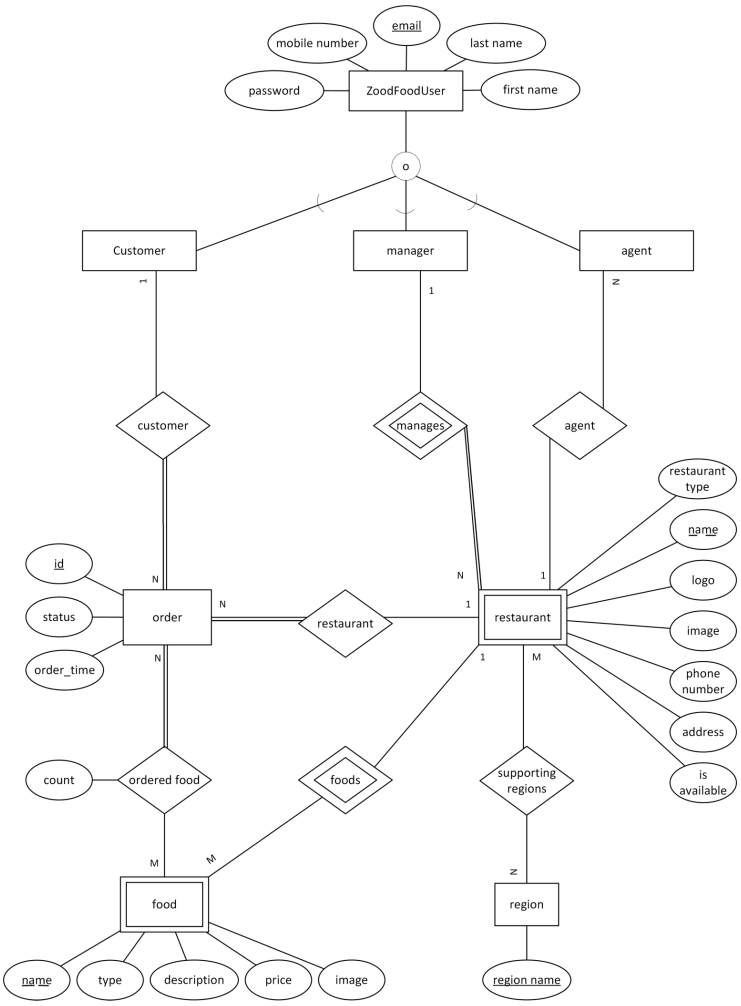
  دقت کنید که رستورانی بدون مدیر نمی‌تواند وجودداشته باشد.

  در مورد رابطه‌های agent و manages باید گفت که همان طور که از نسبت‌های گذاشته شده در نمودار ER معلوم است، نیاز را اینگونه تشخیص دادیم که هر رستوران می‌تواند چند نماینده [۱۸] داشته باشد ولی تنها یک مدیر دارد. یک کاربر نماینده می‌تواند نماینده‌ی تنها یک رستوران باشد ولی هر کاربر مدیر می‌تواند مدیر چند رستوران باشد.

- **region**

  هر رستورانی در منطقه‌ای واقع است. در وب گاه ما رستوران‌ها را بر اساس منطقه دسته بندی شده‌اند برای این که هر کاربر به راحتی بتواند رستوران‌های منطقه‌ی مورد نظر خود را ببیند و به سادگی سفارشش را از منطقه‌ی مورد نظر بدهد.

[18] agent



شکل ۶: نمودار ER پروژه‌ی دوم

- **food**

همان طور که از اسم آن معلوم است ما نیاز دیدیم که غذاهای هر رستوران را نیز ذخیره کنیم که کاربران وب گاه ما بتوانند از آن غذا انتخاب کنند و ما این سفارش‌ها را به رستوران مورد نظر ارائه کنیم. واضح است که هر غذا به رستورانی نیاز دارد که به آن وابسته باشد و بدون رستوران نمی‌شود.

صفت‌های غذا روشن است و همان طور که قبلاً گفتیم به دلیل خلاصه‌سازی توضیح آن‌ها را نمی‌آوریم.

- **order**

نیاز دیدیم که هر سفارش را ذخیره کنیم که اگر مشکلی پیش آمد و کاربر غذای خود را دریافت نکرد بتواند موضوع را پی‌گیری کند. هر سفارش صفتی با نام status دارد که وضعیت سفارش را نشان می‌دهد. وضعیت سفارش یکی از حالت‌های زیر است:

  - pending

- approved
- delivered
- rejected

از اسم این حالت‌ها روشن است که هر کدام نشان‌گر کدام وضعیت ممکن هستند. اگر سفارش در یکی از حالت‌های rejected یا delivered باشد پس از ۱ روز از پایگاه داده حذف می‌شود. در غیر این صورت در پایگاه داده باقی می‌ماند.

همچنین تعداد غذاهای سفارش داده شده و غذاهای سفارش داده را نیز نگه می‌داریم. لازم به ذکر است که هر سفارش به تنها یک رستوران و یک کاربر وابسته است همچنین به کمینه یک غذا نیز وابسطه است.

۲.۱.۲ کارکردهای مورد نیاز

در این قسمت کارکردهای مورد نیاز این پروژه در پایگاه داده شناسایی شده و توضیح داده شده است.

- باید بتوان رستوران‌های مربوط به یک منطقه را به دست آورد. برای توضیحات این نیاز در وب گاه به بخش ۳.۱.۲ مراجعه کنید.
- یک مدیر می‌تواند تمام رستوران‌های خود را ببیند. همچنین یک نماینده می‌تواند سفراش‌های رستورانش را ببیند.

۳.۱.۲ توضیحات مربوط به وب گاه

چند نکته در مورد نحوه‌ی عملکرد وب گاه است که لازم دانستیم این جا اشاره کنیم:

- کاریران باید بتوانند با انتخاب منطقه‌ای رستوران‌های مربوط به آن منطقه را ببینند و در اگر خواستند به آن‌ها سفارش خود را بدهند.
- نحوه‌ی عملکرد برای دادن سفارش و تحویل آن اینگونه است که وب گاه سفارش را به همراه شماره‌ی تماس مشتری در اختیار رستورانن می‌گذارد. رستوران‌ها پس از دریافت این سفارش به کاربر زنگ زده و اگر آدرس را نداشته باشند می‌گیرند و اگر داشته باشند تنها تأییدیه گرفته و غذا را ارسال می‌کنند. دست برنامه نویس برای گذاشتن محدودیت به دلیل ایمن کردن سیستم را باز می‌گذاریم.
- نحوه‌ی درآمد وب گاه هم اینگونه است که درصدی ثابت از سود سفارشاتی که به رستوران‌ها می‌دهد را می‌گیرد.

۲.۲ فاز دوم

در این فاز ابتدا جدول‌های مدل رابطه‌ای مربوط را طراحی می‌کنیم سپس به شناسای "دید"های لازم در پایگاه داده می‌پردازیم.

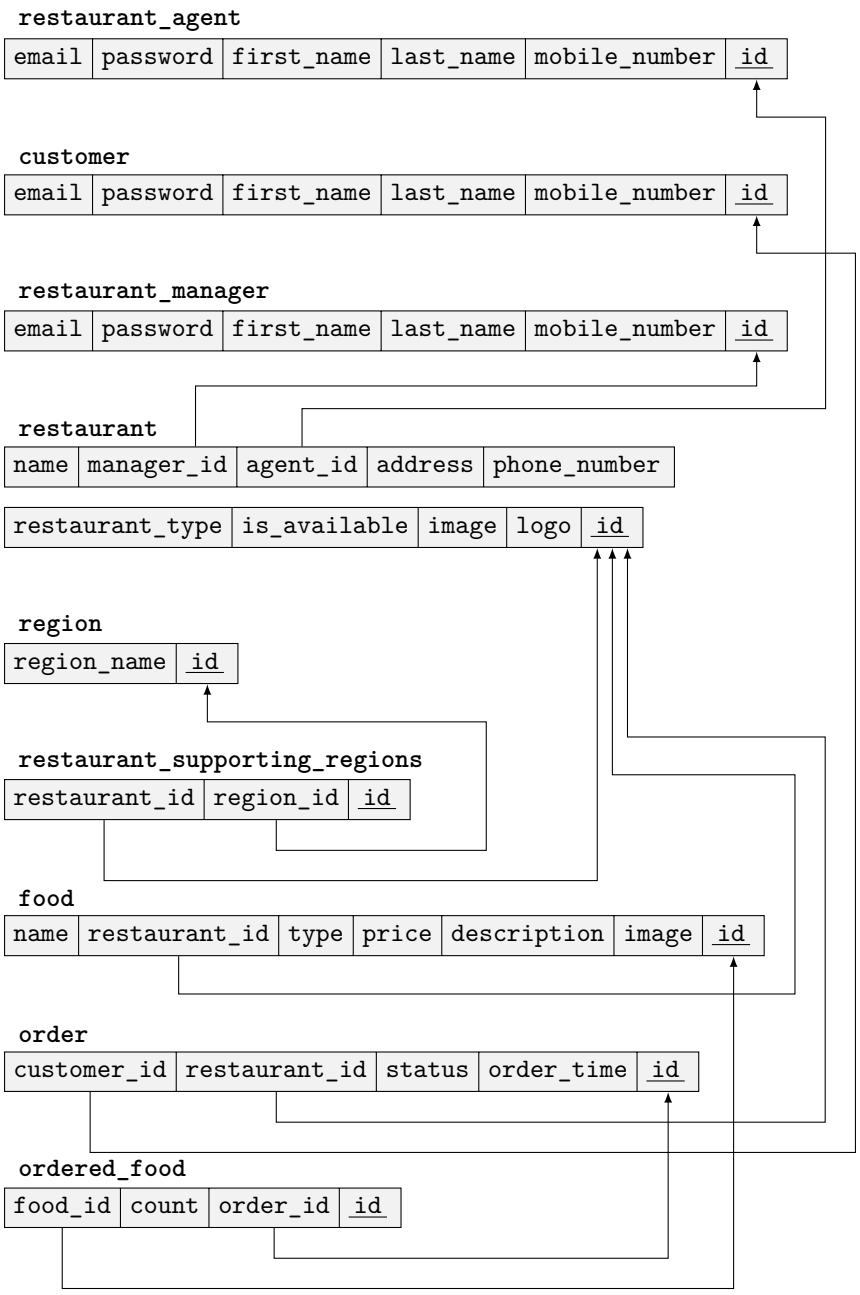
۱.۲.۲ طراحی جدول‌های مدل رابطه‌ای

در این بخش جدول‌های مدل رابطه‌ای را با توجه به نمودار ER شکل ۶ طراحی می‌کنیم. دقت کنید که در فاز سوم ممکن است این طراحی به دلیل نرمال سازی تغییر کند. طراحی انجام شده در شکل ۷ قابل مشاهده است.

۲.۲.۲ شناسایی دیدها

سه دید در این پروژه وجود دارد که از این قرار اند:

- **دید مربوط به کاربران عادی**
  این دید مربوط به کاربران عادی است برای این که سفارش‌های خود را بدهند. در این دید امکان دسترسی و دیدن مدیرها یا نماینده‌ها و یا سفارش‌ها وجود ندارد ولی بقیه اطلاعات را در اختیار دارد. همچنین اطلاعات رستوران‌ها و غذای آن‌ها را نمی‌توانند تغییر دهند.
- **دید مربوط به مدیر رستوران**
  یک دید مربوط به مدیرها وجود دارد که آنها می‌توانند رستوران‌های خود را مشاهده کرده و اطلاعات و غذاهای آن را تغییر دهند. در این دید نمی‌توان سفارش‌ها را تغییر داد.



شکل ۷: جدول‌های مدل رابطه‌ای طراحی شده برای پروژه‌ی دوم از روی نمودار ER شکل ۶

- **دید مربوط به نماینده‌ی رستوران**

دید نماینده‌ی رستوران دیدی است که به سفارش‌های آن رستوران‌ها دسترسی داشته و می‌تواند وضعیت آن‌ها را تغییر دهد. در

این دید به اطلاعات دیگری دسترسی نداریم.

دقت کنید به دلیل محدودیت‌های موجود در PostgreSQL در مورد دیدها(برای مثال امکان به روز رسانی ندارد و تنها با رهاناها باید انجام شود که خود مشکلاتی دیگر به وجود می‌آورد و اینکه صفتی خارج از صفت‌های جدول‌های پایه به عنوان صفت تغییر پذیر حتی با رهانا نمی‌تواند داشته باشد) همان طور که در فاز سوم نیز اشاره خ.اهد شد تصمیم گرفتیم که دیدی را پیاده سازی نکنیم.

۳.۲ فاز سوم

در این فاز به آزمایش جدول‌های طرح شده و جزییات پیاده‌سازی خواهیم پرداخت. توضیحات بخش‌های مختلف این فاز در قسمت "چکیده" آمده است.

۱.۳.۲ دامنه‌ی صفت‌ها

تمامی دامنه‌های انتخاب شده را می‌توان از عنصرهای "اتمی" فرض کرد جزییات آن‌ها را می‌توانید از [۲] نگاه کنید. چند دامنه و صفت وجود که آن‌ها را با * مشخص کرده‌ایم و توضحات لازم در مورد آن‌ها را بعد از جدول ۲ آورده‌ایم.

agent

email:: emailDomain*	password:: passwordDomain*
first_name:: nameDomain*	last_name:: nameDomain*
mobile_number:: phoneNumDomain*	id:: serial

customer

email:: emailDomain*	password:: passwordDomain*
first_name:: nameDomain*	last_name:: nameDomain*
mobile_number:: phoneNumDomain*	id:: serial

manager

email:: emailDomain*	password:: passwordDomain*
first_name:: nameDomain*	last_name:: nameDomain*
mobile_number:: phoneNumDomain*	id:: serial

restaurant

name:: nameDomain*	manager_id:: integer
agent_id:: integer	address:: text
phone_number:: phoneNumDomain*	restaurant_type:: nameDomain*
is_available:: boolean	image*:: text
logo*:: text	id:: serial

region

region_name:: text	id:: serial

restaurant_supporting_regions

restaurant_id:: integer	region_id:: integer
id:: serial	

food

name:: nameDomain*	restaurant_id:: integer
type:: nameDomain*	price:: money
description:: text	image*:: text
id:: serial	

order

customer_id:: integer	restaurant_id:: integer
status:: OrderStatus*	order_time:: date
id:: serial	

ordered_food

food_id:: integer	count:: integer
order_id:: integer	id:: serial

جدول ۲: دامنه‌ی صفت‌های جدول‌های شکل ۷

لازم به ذکر است که دامنه‌ی image و text logo گذاشته شده است. این به این دلیل است که ما آدرس مکان ذخیره‌ی توضیح دامنه‌های باعلامت‌دار: دامنه‌های

- cellPhoneNumDomain
- emailDomain
- nameDomain
- passwordDomain
- phoneNumDomain

در بخش ۱.۳.۱ توضیح داده شده است.

OrderStatus

در فاز ۱ در بخش ۱.۱.۲ اشاره شد که هر سفارش دارای وضعیتی است که این وضعیت یکی از ۴ مقدار گفته شده را داراست. از رو یک type ساخته شده است که از نوع enum است دستور ساخت آن به همراه مقدارهایش از این قرار است.

```
CREATE TYPE "OrderStatus" AS ENUM
('pending',
 'approved',
 'delivered',
 'rejected');
```

۲.۳.۲ نرمال سازی

هدف از نرمال سازی کاهش (و در حالت بسیار خوب از بین بردن) افزونگی‌ها [19] و همچنین به کمینه رساندن (در حالت بسیار خوب از بین بردن) ناهنجاری‌های [20] احتمالی است. [۱، ص. ۴۷۵]
در طراحی فعلی هیچ ناهنجاری به روز رسانی‌ای مشاهده نمی‌شود.

[19] redundancy
[20] anomaly

در طراحی یک وب گاه [۲۱] زمان واکنش [۲۲] هم در کنار استفاده درست از منابع سخت افزاری مطرح است. اجرا کامل نرمال سازی‌ها باعث تولید جدول‌های زیادی می‌شود و در نتیجه نیاز به اجرا تعداد زیادی عمل پیوند [۲۳] برای پاسخ گویی به نیازهای وب گاه می‌شود. عمل پیوند عملی زمان بر است و اجرا چندبار آن برای پاسخ گویی به نیازهای وب گاه زمان واکنش وب گاه را بالا میبرد. بنابراین نرمال سازی انجام شده در این پروژه تا حدی بوده که هم از منابع سخت افزاری به درستی استفاده شود و هم زمان واکنش وب گاه خوب باشد. این حد همین طراحی فعلی است. البته وابستگی‌ها بررسی شدند ولی به دلیل مشابه بودن کارها با بخش ۲.۳.۱ توضیحات آن آورده نشده است.

۳.۳.۲ محدودیت‌های جامعیتی

محدودیت‌های جامعیتی شامل ۲ نوع "جامعیت موجودیت" [۲۴] و "جامعیت ارجاعی" [۲۵] است. کلیدهای تمامی جدول‌ها و همچنین جامعیت‌های ارجاعی بین آن‌ها را در شکل ۷ مشخص کرده‌ایم. در پیاده‌سازی با SQL این کلیدها و جامعیت‌های ارجاعی را مشخص کرده‌ایم. از این رو خود DBMS این جامعیت‌ها را چک می‌کند و از برقرار بودن آن اطمینان حاصل می‌کند.

۴.۳.۲ ادعاها و بررسی‌ها

همانند توضیح‌های بخش ۴.۳.۱ عمل می‌کنیم یا این تفاوت که اینجا محدودیت‌های UNIQUE هم می‌آوریم. به دلیل مشابه بودت توضیحات در این بخش توضیحا را نمی‌آوریم و تنها به ذکر آن‌ها بسنده می‌کنیم.

ادعاها

در پیاده‌سازی جایی از ادعا استفاده نشده است چرا که در صورت برقرار نبودن شرط، کل جدول را قفل می‌کرد که این موضوع در جایی مورد نیاز ما نبود.

بررسی‌ها و قانون به روز رسانی کلیدهای خارجی

در این‌جا لازم به ذکر است که چندی از این بررسی‌ها در تعریف دامنه‌ها است (و حتی ممکن است در typeها هم باشد ولی ما استفاده نکردیم) که آن‌ها را در بخش ۱.۳.۱ ذکر کردیم. در اینجا به بقیه آن‌ها اشاره می‌کنیم.

در جدول restaurant_agent

```
CONSTRAINT "customerUniqueEmail"
   UNIQUE (email)
```

در جدول customer

```
CONSTRAINT "customerUniqueEmail"
   UNIQUE (email)
```

در جدول restaurant_manager

```
CONSTRAINT "customerUniqueEmail"
   UNIQUE (email)
```

در جدول restaurant

---

[21] website
[22] response time
[23] join
[24] entity integrity
[25] referential integrity

```
CONSTRAINT "agentIdFK" FOREIGN KEY (agent_id)
   REFERENCES restaurant_agent (id)
   ON UPDATE CASCADE ON DELETE SET NULL

CONSTRAINT "managerIdFK" FOREIGN KEY (manager_id)
   REFERENCES restaurant_manager (id)
   ON UPDATE CASCADE ON DELETE RESTRICT

CONSTRAINT "restaurantUniqueManagerRName"
   UNIQUE (manager_id, name)
```

در جدول restaurant_supporting_regions

```
CONSTRAINT "regionIdFK" FOREIGN KEY (region_id)
   REFERENCES region (id)
   ON UPDATE CASCADE ON DELETE CASCADE

CONSTRAINT "restaurantIdFK" FOREIGN KEY (restaurant_id)
   REFERENCES restaurant (id)
   ON UPDATE CASCADE ON DELETE CASCADE

CONSTRAINT "rSupportingRegionsUniqueRestaurantRegion"
   UNIQUE (restaurant_id, region_id)
```

در جدول food

```
CONSTRAINT "restaurantIdFK" FOREIGN KEY (restaurant_id)
   REFERENCES restaurant (id)
   ON UPDATE CASCADE ON DELETE CASCADE

CONSTRAINT "foodUniqueRestaurantFName"
   UNIQUE (name, restaurant_id)
```

در جدول order

```
CONSTRAINT "customerIdFK" FOREIGN KEY (customer_id)
   REFERENCES customer (id)
   ON UPDATE CASCADE ON DELETE RESTRICT

CONSTRAINT "restaurantIdFK" FOREIGN KEY (restaurant_id)
   REFERENCES restaurant (id)
   ON UPDATE CASCADE ON DELETE RESTRICT
```

در جدول ordered_food

```
CONSTRAINT "foodIdFK" FOREIGN KEY (food_id)
   REFERENCES food (id)
   ON UPDATE CASCADE ON DELETE RESTRICT
```

```
CONSTRAINT "orderIdFK" FOREIGN KEY (order_id)
   REFERENCES "order" (id)
   ON UPDATE CASCADE ON DELETE RESTRICT,

CONSTRAINT "orderedFoodUniqueOrderFood"
   UNIQUE (food_id, order_id)

CONSTRAINT "countConstraint"
   CHECK (count > 0)
```

۵.۳.۲ رهاناها و تابع‌ها

در این پروژه نیازی به رهانا یا تابعی مشاهده نکردیم.

۴.۲ نیازهای وب گاه از پایگاه داده

در این بخش به توضیح چهار بخش زیر می‌پردازیم.

- رویه‌ی بارگذاری داده‌ها و آزمایش‌ها
- پرس‌وجوهای [۷] لازم
- کپی پشتیبان [۸] از پایگاه‌داده
- دیدهای پیاده‌سازی شده

۱.۴.۲ رویه‌ی بارگذاری داده‌ها و آزمایش‌ها

رویه‌ی بارگذاری:
یک برنامه به زبان python نوشته شده است که به طور مصنوعی داده‌هایی را تولید می‌کند که تولید آن‌ها وظیفه‌ی DBA Staff است. به دلیل داشتن پیشنیازهای نرم‌افزاری، برای آشنایی با نحوه‌ی اجرای این کد به بخش ۱.۴ مراجعه کنید.

رویه‌ی آزمایش:
از آن جا که تمام ساختارها و پرس‌وجوهای پیاده‌سازی شده به منظور استفاده در وب گاه نوشته شده‌اند، آزمایش کردن و درست کار کردن آن‌ها معادل آزمایش کردن و درست کار کردن تمامی قابلیت‌های وب گاه است. برای آزمایش کردن وب گاه در هنگام development، عملیات آزمایش دو بار انجام شده است. اولین بار در هنگام نوشتن صفحه‌ی مربوط به یک قابلیت، درست اجرا شدن آن مورد آزمایش قرار گرفته است. بار دوم، بعد از تمام شدن نوشتن همه‌ی صفحه‌ها، با گذر کردن در وب گاه و استفاده از هر قابلیتی که در آن موجود است، همه چیز مورد آزمایش قرار گرفته است.

۲.۴.۲ پرس‌وجوهای لازم

تمام پرس‌وجوهای مورد استفاده به طور روشن در فایل db_stuff در پوشه‌ی base_app در پوشه‌ی پروژه قرار دارد.

۳.۴.۲ کپی پشتیبان از پایگاه داده

کپی‌های پشتیبان در پوشه‌ی Backups قرار داده شده‌اند. در نام این کپی‌ها نوع پشتیبان گیری نیز ذکر شده است. برای آشنایی با نحوه‌ی کار با آن‌ها به [۴] مراجعه کنید.
به طور بسیار خلاصه برای "بازیابی" اطلاعات کارهای زیر باید به ترتیب انجام شود. به نام کاربری توجه کنید.

۱. یک کاربر به نام my_food_role درست کنید.

۲. یک پایگاه داده به نام my_food_db بسازید.

۳. داخل این پایگاه داده اطلاعات را بازیابی کنید.

۴.۴.۲ دیدهای پیاده‌سازی شده

همان طور که در بخش ۲.۲.۲ توضیح داده شد به دلیل مشکلاتی که دید در DBMS ما دارد در این پروژه دیدی نباز دیده نشد و پیاده سازی نشده است. در اینجا لازم دانستیم که اشاره کنیم که برنامه نویس وب گاه نیاز دید که جدولی به نام session در پایگاه داده داشته باشد که بتواند از آن استفاده کند. شکل ۸ این جدول را نشان می‌دهد. دامنه‌ی صفت‌های این جدول نیز در جدول ۳ قابل مشاهده

session

data	id

شکل ۸: جدول session ساخته شده به درخوات برنامه نویس وب گاه

است.

session

data:: text	id:: serial

جدول ۳: دامنه‌ی صفت‌های جدول session

۳ دلیل انتخاب PostgreSQL

PostgreSQL یک ORDBMS یا سیستم پایگاه داده‌ی object-relational می‌باشد. از خواص این سیستم open source بودن و قدرت بالای این نرم‌افزار در پشتیبانی از اکثر زبان‌های برنامه نویسی مانند java است.
امروزه PostgreSQL به عنوان پیشگام بسیاری از مفاهیم object-relational در بسیاری از پایگاه‌های تجاری عرضه می‌گردد. در سیستم مدیریت پایگاه داده قدیمی، از مجموعه نام‌های وابسته، که همگی شامل صفت‌هایی همگون بودند پشتیبانی می‌شد و در سیستم‌های تجاری فعلی، انواعی شامل Money، Character String، Integer، Floating Point Number و Date قابل پشتیبانی می‌باشند.
PostgreSQL توانایی عرضه و پشتیبانی مفاهیم Inheritance، Data Type و Function و همچنین ابزارهایی با قابلیت‌های بیشتر مانند: Constraints، Triggers، Rules و Transaction Integrity را داراست.
PostgreSQL دارای ویژگی‌های پیچیده از قبیل کنترل هم‌زمانی چند نسخه، ریکاوری در زمان، تکرار ناهم‌زمان، تراکنش‌های تودرتو، پشتیبان‌گیری آنلاین و بسیاری دیگر می‌باشد. محودیت‌های این نرم‌افزار در جدول آمده است.

Limit	Value
Maximum Database	unlimited
Maximum Table Size	32 terabyte
Maximum Row Size	1.6 terabyte
Maximum Rows per Table	unlimited
Maximum Columns per Table	1600-2500 depending on column types
Maximum Indexes per Table	unlimited

جدول ۴: محدودیت‌های PostgreSQL به عنوان یک DBMS

برای مقایسه‌ی فنی بین این نرم‌افزار و MySQL به عنوان ابزار محبوب و مشابه دیگر می‌توان به ویژگی‌های زیر اشاره کرد:

- PostgreSQL یک پایگاه داده یکپارچه است و فقط یک موتور ذخیره‌سازی [۲۶] دارد در حالی که MySQL دو لایه دارد، یک لایه SQL و یک لایه موتور ذخیره‌سازی که این قابلیت را به MySQL می‌دهد که موتور ذخیره‌سازی‌های خاص منظوره داشته باشد.

[26] storage engine

- PostgreSQL بر روی سرورهایی که پردازنده‌های چند هسته‌ای دارند بهتر عمل می‌کند و استفاده بهینه‌تری از هسته‌های پردازنده می‌کند. البته MySQL هم در نسخه‌های جدید بهبودهای خوبی داشته است.
- PostgreSQL یک API کامل برای ارتباط ناهنگام [۲۷] برای برنامه‌های client فراهم کرده است که با عث بالا رفتن کارایی می‌شود اما MySQL فقط بر روی سیستم عامل‌های خانواده‌ی Unix و از طریق کتابخانه‌های native این قابلیت را در اختیار برنامه‌های client قرار می‌دهد.
- PostgreSQL کاملاً سازگار با ACID است اما تمام موتورهای ذخیره‌سازی MySQL سازگار با ACID نیستند برای مثال InnoDB کاملاً با ACID سازگار است.

۴ نیازهای نرم‌افزاری

در این بخش نیازمندی‌های نرم‌افزاری هر پروژه آورده شده است.

۱.۴ پروژه‌ِ اول

برای اجرای کدهای داده شده انجام مراحل زیر لازم است:

۱. نصب برنامه‌ی pgAdmin III

۲. پایگاه داده را با روش گفته شده در بخش ۳.۴.۱ بازیابی کنید. برای این که نیاز به تغییر کدهای وب گاه نباشد، رمز عبور را برای کاربر CouponUser DBA staffای که می‌سازید برابر با عبارت "mms373" قرار دهید. در فایل ارائه شده داده‌هایی نیز از پیش وجود دارند که بتوان وب گاه را با آن‌ها آزمایش کرد.

۳. نسخه‌ی ۳ام python را نصب کنید. دقت کنید که نسخه‌ی دوم این زبان برنامه‌نویسی به هیچ وجه توانایی اجرای کدهای داده شده را ندارد.

۴. در صورتی که python نصب شده‌ی شما قدیمی است و به صورت پیش‌فرض pip را به همراه خود ندارد، pip را نصب کنید.

۵. کتاب‌خانه‌های زیر به وسیله‌ی pip نصب کنید. دقت کنید که نسخه‌ای از هر کتاب‌خانه که نصب می‌کنید نه جدیدتر و نه قدیمی‌تر از چیزی که در اینجا نوشته شده است باشد. زیرا اکثر این کتاب‌خانه‌ها در نسخه‌های قبلی و بعدی خود تفاوت‌های زیادی با نسخه‌ی گفته شده دارند.

- Django (1.9.6)
- colorama (0.3.7)
- psycopg2 (2.6.1)

۶. در صورتی که نیاز به بارگذاری داده‌های بیشتر و تست وب‌گاه باشد، با اجرای کد dba_staff_bot.py این عملیات را می‌توان به تعداد دفعات دلخواه اجرا کرد تا تعداد داده‌ها به اندازه‌ی لازم زیاد شود.

۷. هرچند امکانات استفاده شده در نوشتن این وب گاه امکانات مشترکی است، اما ترجیحاً آزمایش آن باید با مرورگر chrome انجام شود و cookieها و javascript در مرورگر فعال باشند.

۸. در صورتی که فونت وب گاه مشکل داشت، فونت "B Nazanin" را که در فایل‌های ارائه شده، در پوشه‌ی Takhfifan/discounts/static/fonts قرار دارد را نصب کنید؛ هرچند که در صورت استفاده از مرورگر استاندارد، خود وب گاه این فونت را برای مرورگر شما ارسال و روی آن نصب خواهد کرد.

[27] asynchronous

۹. بعد از انجام مراحل ذکر شده در بالا، با استفاده از shell سیستم عامل خود (terminal یا cmd) به پوشه‌ی Takhfifan وارد شوید و دستور زیر را اجرا کنید:

Unix-based OS: python3 manage.py runserver Windows: python manage.py runserver

بعد از انجام این کار، سرور شروع به کار کرده است و روی port ۸۰۰۰ سیستم شما در حال گوش کردن می‌باشد. می‌توانید برای شروع به آدرس localhost:8000 بروید. آدرس‌های وب گاه طوری طراحی شده‌اند که اگر جایی آدرسی به نظرتان منطقی برسد، احتمالاً این آدرس یا پیاده‌سازی شده و یا شما را به صفحه‌ی متناظرش ارجاع خواهد داد.

لازم به ذکر است که برای ساختن کپن‌های بیشتر باید از برنامه‌ی pgAdmin استفاده کنید. دلیل این امر آن است که برای خود DBA staff برنامه‌ی adminای در نظر گرفته شده است که با node.js کار می‌کند، اما راه‌اندازی این برنامه نیاز به آدرس فایل‌های سرور PostgreSQL دارد و از آن جا که چنین فایلی به راحتی قابل دسترس نیست، این برنامه بخشی از پروژه در نظر گرفته نشده است.

۲.۴ پروژه‌ی دوم

۱. ابتدا python را نصب کنید و pip را نیز همراه آن نصب کنید تا در مرحله‌های نصب بعدی راحت تر باشید برای نصب pip کافی است به [۵] مراجعه کنید.

۲. کتابخانه‌های زیر را با استفاده از pip نصب کنید:

- adium-theme-ubuntu==0.3.4
- astroid==1.4.5
- beautifulsoup4==4.4.1
- chardet==2.3.0
- colorama==0.3.7
- decorator==4.0.6
- Django==1.9.6
- html5lib==0.999
- ipython==2.4.1
- lazy-object-proxy==1.2.2
- lxml==3.5.0
- numpy==1.11.0
- pexpect==4.0.1
- Pillow==3.2.0
- psycopg2==2.6.2
- ptyprocess==0.5
- pylint==1.5.5
- pyttsx==1.1
- pytube==6.1.8
- simplegeneric==0.8.1
- six==1.10.0
- SQLAlchemy==1.0.13
- unity-lens-photos==1.0
- wrapt==1.10.8
- XlsxWriter==0.9.0

۳. سپس PostgreSQL را نصب کرده و به آن roleای به نام my_food_role با کلمه عبور 12345678 و databaseای با نام my_food_db که به my_food_role تعلق دارد بسازید.

۴. حال با استفاده از psql plugin و فایل backup جدول‌های مورد نیاز را در my_food_db بسازید.

۵. حال می توانید با استفاده از برنامه populate_db.py در پوشه وب گاه، داده تصادفی تولید کرده و به جدول‌ها درج کنید تا در هنگام آزمایش وب گاه راحت تر باشید.

۶. حال در shell سیستم عامل خود (cmd یا terminal) به پوشه‌ای که پروژه در آن قرار دارد بروید و دستور زیر را وارد کنید: python manage.py runserver.

۷. حال می‌توانید از طریق آدرس زیر به وب گاه دسرسی پیدا کنید. localhost:8000

مرجع‌ها

[1] Fundamentals of Database Systems 7th edition
by ELMASRI ● NAVATHE

[2] PostgreSQL Data Type - TutorialsPoint

[3] PostgreSQL Trigger Procedures

[4] How to Dump and Restore Postgres Databases

[5] pip Installation